

www.vuCTx.com

CS604 Complete MERGED PPT Slides 1 TO 45

Regards : Zarva Chaudhary

Admin:

* Zarva Chaudhary *

Merged File date
31-07-2021

Chandhary Moazzam

* Laiba Maki *

Operating Systems

Lecture 1

Agenda for Today

- Introduction and purpose of the course
- Organization of a computer system
- Purpose of a computer system—setting the stage for OS concepts and principles
- Outline of topics to be discussed in the course
- What is an operating system?
- Recap of the lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Computer System

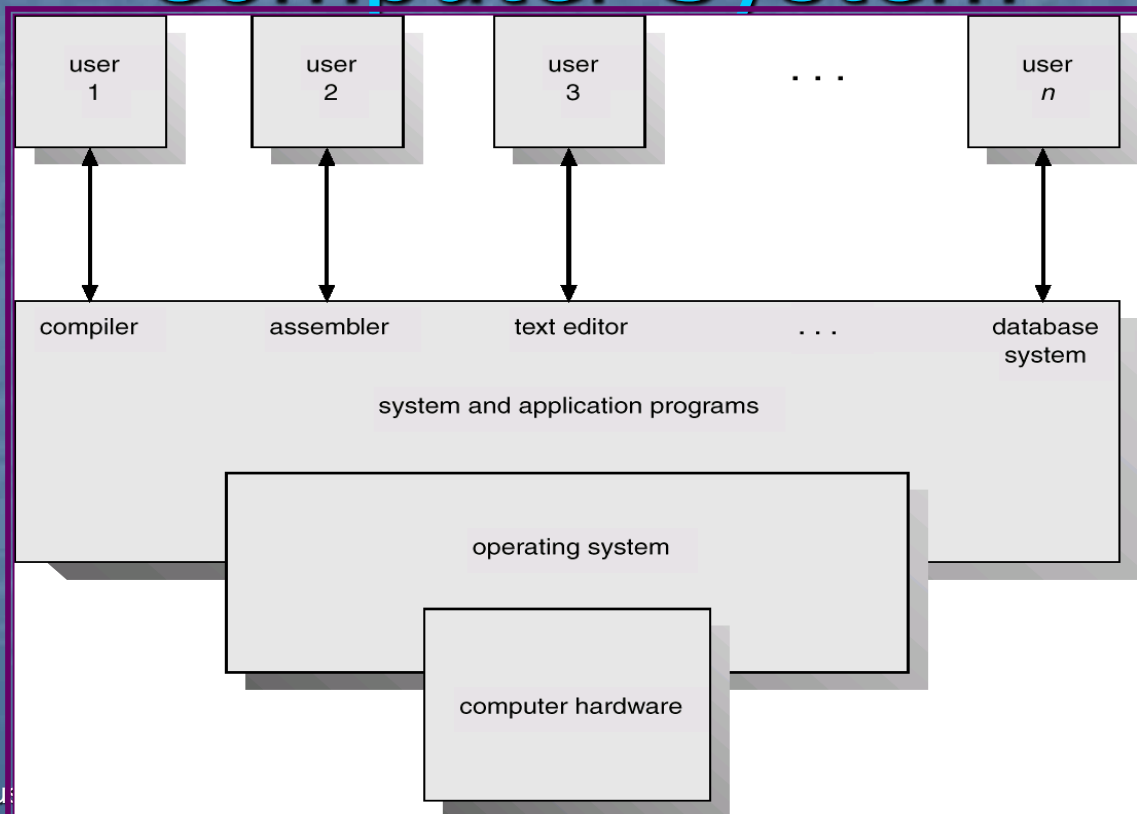
1. Hardware
2. Operating system
3. Applications programs
4. Users

7 August 2021

© Copyright Virtual University of
Pakistan



Layered View of a Computer System



7 August

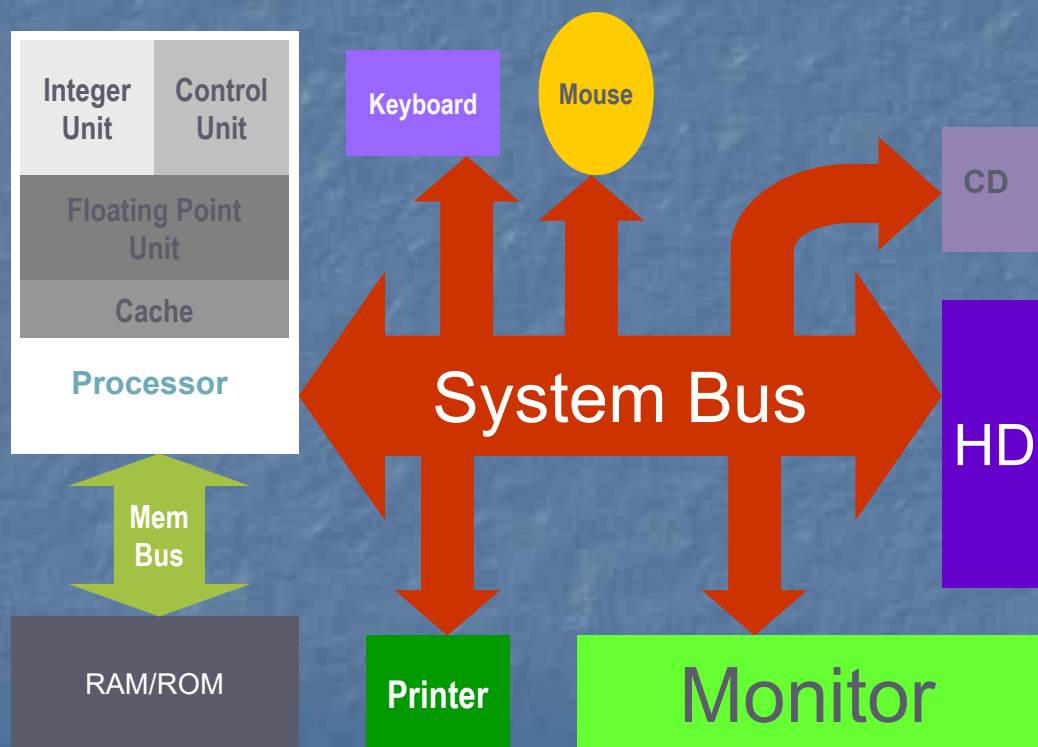


Purpose of a Computer System

- Computer systems consist of software and hardware that are combined to provide a tool to solve specific problems in an efficient manner
- Execute programs



Computer System Hardware



7 August 2021

© Copyright Virtual University of
Pakistan



Course Outline

- Operating system concepts
- Operating system structures
- Introduction to UNIX/Linux user interface
- Processes and threads—scheduling, concurrency, synchronization, etc.
- Deadlocks
- Memory management
- Virtual memory
- File system
- Secondary storage management

7 August 2021

© Copyright Virtual University of
Pakistan



Operating System Concepts

- Single-user systems
- Batch systems
- Multiprogrammed systems
- Time-sharing systems
- Real-time systems
- Interrupts, traps, and software interrupts (UNIX signals)
- Hardware protection

7 August 2021

© Copyright Virtual University of
Pakistan



Operating System Structures

- Operating system services
- System calls
- Semantics of system call execution
- Operating system structures (monolithic, microkernel-based, layered, virtual machines, DOS-Windows, UNIX)
- System design and implementation

7 August 2021

© Copyright Virtual University of
Pakistan



Introduction to UNIX/Linux

- Directory structure
- Browsing directory structure
- Useful commands

7 August 2021

© Copyright Virtual University of
Pakistan



Processes

- Process concept (process, states, attributes, etc.)
- Process scheduling (scheduler)
- Context switching (dispatcher)
- Operations on processes (creation, termination, signaling, suspend, foreground, background, etc.)
- Process management in UNIX (fork, wait, exec, exit, etc.)

7 August 2021

© Copyright Virtual University of
Pakistan



Processes (continued)

- Sample code for UNIX/Linux process management
- Cooperating processes
- Interprocess communication (IPC)
- IPC in UNIX/Linux (pipe, FIFO, socket, message queue, etc.)

7 August 2021

© Copyright Virtual University of
Pakistan



Processes (continued)

- Communication between UNIX/Linux processes (pipe, mkfifo, read, write, close, etc.)
- Sample code
- UNIX/Linux processes (process images, control structures, etc. explained with sample code)
- Managing UNIX/Linux processes (ps, top, fg, bg, <Ctrl-Z>, <Ctrl-C>, etc.)

7 August 2021

© Copyright Virtual University of
Pakistan



Threads

- Thread concept (thread, states, attributes, etc.)
- User- and kernel-level threads
- POSIX threads (the pthread library)
- Sample code

7 August 2021

© Copyright Virtual University of
Pakistan



CPU Scheduling

- Basic concepts
- Scheduling criteria
- Scheduling triggers
- Scheduling algorithms
- UNIX System V scheduling algorithm
- Optimal scheduling
- Algorithm evaluation

7 August 2021

© Copyright Virtual University of
Pakistan



Process Synchronization

- Basic concept
- The Critical Section Problem
- Solutions for the Critical Section Problem
- Software-based solutions—the Bakery Algorithm
- Hardware-based solutions
- Semaphores
- Binary and counting semaphores

7 August 2021

© Copyright Virtual University of
Pakistan



Process Synchronization ...

- Classic problems of synchronization
- Deadlocks and starvation
- Critical regions
- Monitors
- Synchronization tools used in Solaris, Linux, and Windows
- Deadlocks and starvation
- Pthread library functions

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlocks

- Basic concept
- Deadlock characterization
- Deadlock handling (prevention, avoidance, detection and recovery)
- Banker's algorithm

7 August 2021

© Copyright Virtual University of
Pakistan



Memory Management

- Basic concepts
- Various techniques for memory management
- Logical to physical address translation
- Swapping
- Contiguous memory allocation: MFT, MVT
- External fragmentation
- Paging
- Hardware support for paging
- Internal fragmentation

7 August 2021

© Copyright Virtual University of
Pakistan



Memory Management ...

- Performance of paging
- Protection and sharing
- Page table issues: Multi-level paging, Hashed page tables, Inverted page tables
- Segmentation
- Protection and sharing
- Segmentation with paging
- Intel P4 example

7 August 2021

© Copyright Virtual University of
Pakistan



Virtual Memory

- Basic concept
- Demand paging
- Page fault
- Performance of demand paging
- Page replacement
- Allocation of frames
- Thrashing
- Operating-system examples
- Other considerations (I/O locking, page size, ...)

7 August 2021

© Copyright Virtual University of
Pakistan



File System Interface

- Basic concepts (file attributes, operations, types, structure, etc.)
- Access methods (sequential, random, etc.)
- Directory structure
- UNIX/Linux directory structure (links in UNIX)
- File system mounting, sharing, and protection
- UNIX/Linux examples for sharing and protection, and relevant commands (chmod, ln, ln -s, etc.)

7 August 2021

© Copyright Virtual University of
Pakistan



File System Implementation

- Basic concepts (overview of disk structure, file structure, boot control block, super block, inode, per process file descriptor table, system-wide open-file table, etc.)
- Directory implementation
- Free space management methods
- Space Allocation Methods
- Time and space performance of allocation methods
- Brief introduction to Network File System (NFS)

7 August 2021

© Copyright Virtual University of
Pakistan



Mass Storage Structure and Scheduling

- Disk structure and scheduling
- Disk management (formatting, boot block, bad blocks, etc.)
- Course Recap

7 August 2021

© Copyright Virtual University of
Pakistan



What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware—provides the user a simpler (virtual) machine to work with
- A program that allocates and deallocates computer system resources in an efficient, fair, and secure manner—a resource manager

7 August 2021

© Copyright Virtual University of
Pakistan



Operating System Goals

- Execute user programs and make solving user problems easier.
- Make the computer system convenient to use.
- Use the computer hardware in an efficient manner.

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 1



Operating Systems

Lecture 2



Agenda for Today

- Review of previous lecture
- Single-user systems
- Batch systems
- Multiprogrammed systems
- Time-sharing systems
- Real-time systems
- Interrupts, traps, and signals
- CPU, I/O, and memory protection
- Recap of the lecture



Single User Systems

- *Personal computers* – computer system dedicated to a single user.
- Interactive
- User convenience and responsiveness.



7 August 2021

© Copyright Virtual University of Pakistan

VUI
Virtual University

Single User Systems

- Can adopt technology developed for larger operating systems—multi-process, multi-user
- Individuals usually have sole use of computer and do not need advanced protection features.
- May run several different types of operating system (Windows, MacOS, UNIX, Linux)



7 August 2021

© Copyright Virtual University of Pakistan

VUI
Virtual University

Batch Systems

- First rudimentary system.
- User $\neq$ operator
- Reduce setup time by batching similar jobs
- Automatic job sequencing – automatically transfers control from one job to the next
- Resident monitor :
 - initial control in monitor
 - control transfers to job
 - when job completes control transfers back to monitor



7 August 2021

© Copyright Virtual University of Pakistan



Memory Layout

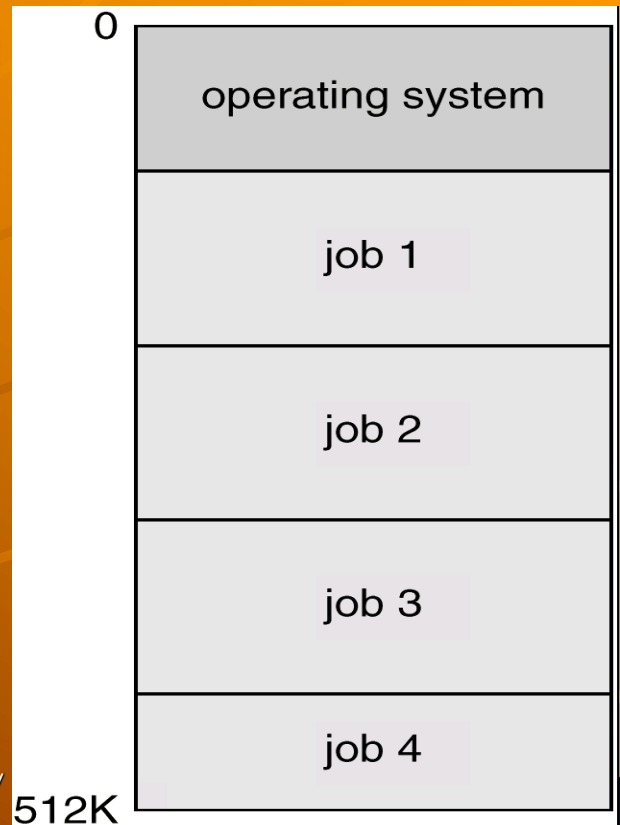


7 August 2021



Multiprogrammed Systems

Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them.



7 August 2021

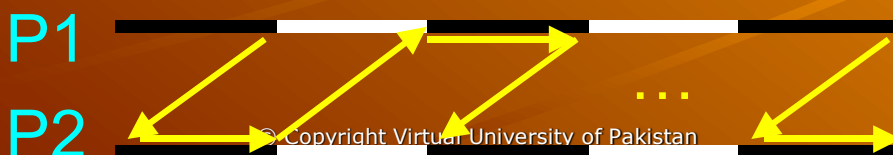
© Copyright Virtual University

Multiprogrammed Systems

Example: Two processes P1 and P2 with CPU and I/O bursts of one time unit each

CPU Burst

I/O Burst

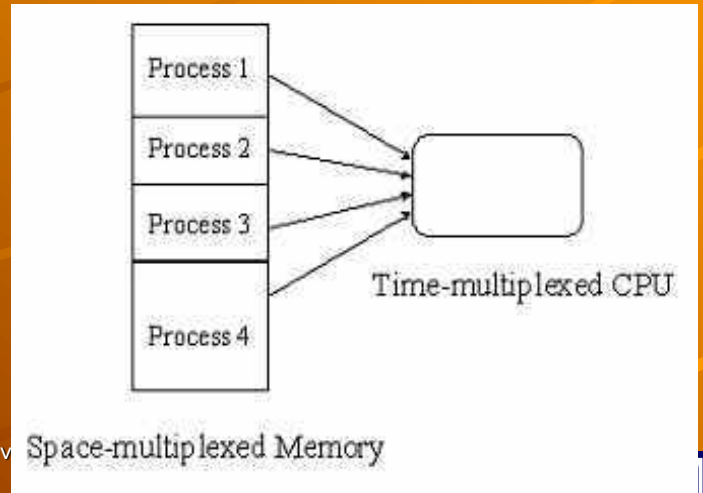


7 August 2021

© Copyright Virtual University of Pakistan

OS Features Needed for Multiprogramming

- SPOOLing (Simultaneous Peripheral Operation On-Line)
- Memory management
- CPU scheduling



7 August 2021

© Copyright Virtual University of

Space-multiplexed Memory

Virtual University

Time-sharing Systems



- An interactive system with multiprogramming
- A job is swapped in and out of memory to the disk if needed.
- On-line file system must be available for users to access data and code.

7 August 2021

© Copyright Virtual University of

Virtual University

Real-time Systems

- Well-defined fixed-time constraints.
- Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems.
- Real-Time systems may be either *hard* or *soft* real-time.

7 August 2021

© Copyright Virtual University of Pakistan



Real-time Systems ...

- Hard real-time systems:
 - Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM)
 - No virtual memory—time cannot be “wasted” on translation of logical to physical addresses
 - OS code structured for efficiency
 - Plane landing systems, process control in nuclear power plants, respirators, etc.

7 August 2021

© Copyright Virtual University of Pakistan



Real-time Systems ...

- Soft real-time systems
 - Output should be produced within the given time constraints but if it is not, the result is not life threatening
 - Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.

7 August 2021

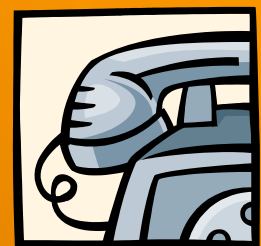
© Copyright Virtual University of Pakistan



VUI
Virtual University

Interrupts, Traps, and Signals

- The occurrence of an event is usually signaled by an interrupt from either the hardware or the software.
- Hardware may trigger an interrupt at any time by sending a signal to the CPU usually by way of the system bus.



Answer
the Phone

...

- Software may trigger an interrupt by executing a

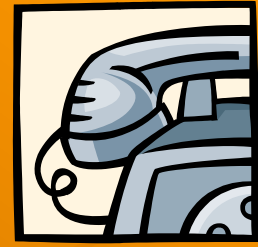
7 August 2021

© Copyright Virtual University of Pakistan

Resume VUI
Virtual University

Interrupts, Traps, and Signals

- A process can generate a trap, for example, by dividing a number by zero.
- A user or a process may generate a signal (an interrupt to a process)



Answer
the Phone

7 August 2021

© Copyright Virtual University of Pakistan

Resume 

Interrupt Handling

- Interrupt transfers control to the interrupt service routine, generally, through the *interrupt vector*, which contains addresses of all the interrupt service routines.
- Interrupt architecture must save the address of the instruction after the interrupted instruction and the CPU state so that execution of the interrupted process may continue after the interrupt has been serviced.

7 August 2021

© Copyright Virtual University of Pakistan



Interrupt Handling ...

- Incoming interrupts are *disabled* while another interrupt is being processed to prevent *lost interrupts*.
- An operating system is *interrupt driven*.



7 August 2021

© Copyright Virtual University of Pakistan



Hardware Protection

- Dual-Mode Operation
- I/O Protection
- Memory Protection
- CPU Protection



7 August 2021

© Copyright Virtual University of Pakistan



Dual-Mode Operation

- Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.
- Provide hardware support to differentiate between at least two modes of operations.
 - *User mode* – execution done on behalf of a user.
 - *Monitor mode* (also *kernel mode* or *system mode*) – execution done on behalf of operating system.

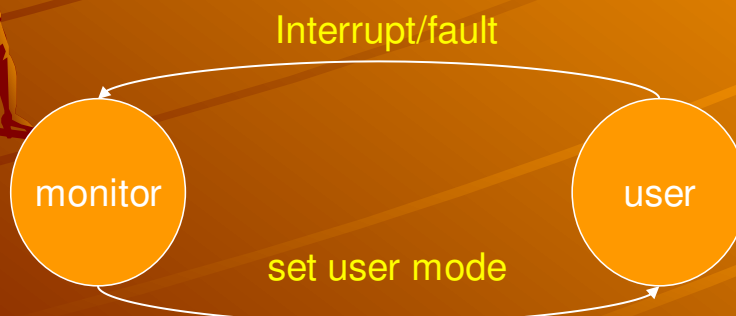
7 August 2021

© Copyright Virtual University of Pakistan



Dual-Mode Operation ...

- *Mode bit* added to computer hardware to indicate the current mode: monitor (0) or user (1).
- When an interrupt or fault occurs hardware switches to monitor mode.



7 August 2021

Privileged instructions can be issued only in monitor mode.

© Copyright Virtual University of Pakistan



Recap of Lecture

- Single-user systems
- Batch systems
- Multiprogrammed systems
- Time-sharing systems
- Real-time systems
- Interrupts, traps, and signals
- Privileged instructions
- I/O protection
- Recap of the lecture

7 August 2021

© Copyright Virtual University of Pakistan



Operating Systems

Lecture 2

7 August 2021

© Copyright Virtual University of Pakistan



Operating Systems

Lecture 3

Agenda for Today

- Review of previous lecture
- Hardware (I/O, memory, and CPU) protection
- Operating system components and services
- System calls
- Operating system structures
- Operating system design and implementation

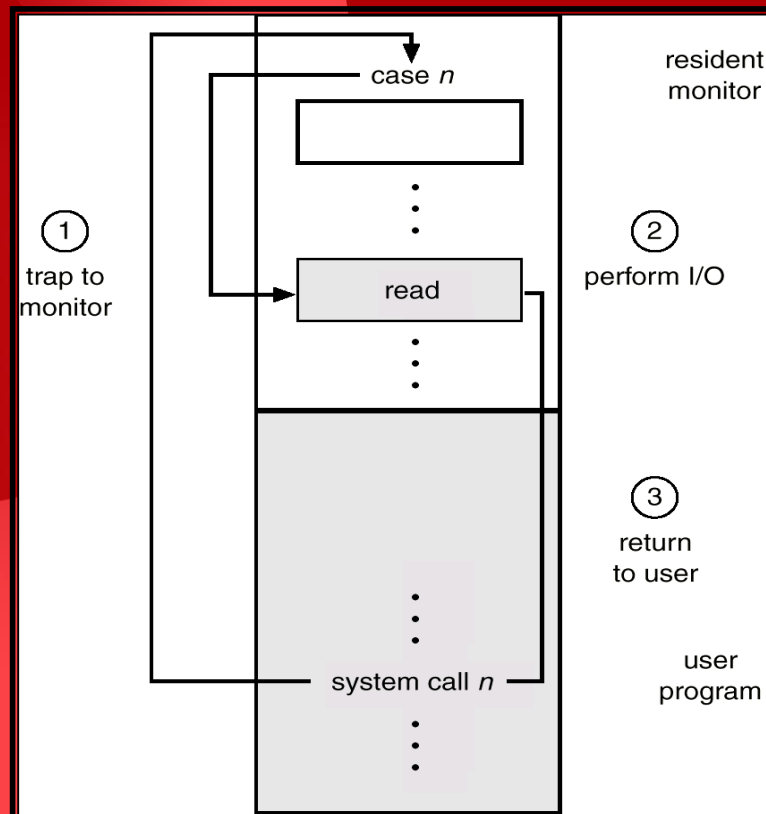
Hardware Protection

- Dual-Mode Operation
- I/O Protection
- Memory Protection
- CPU Protection

I/O Protection

- All I/O instructions are privileged instructions.
- Must ensure that a user program could never gain control of the computer in monitor mode (i.e., a user program that, as part of its execution, stores a new address in the interrupt vector).

Use of A System Call to Perform I/O



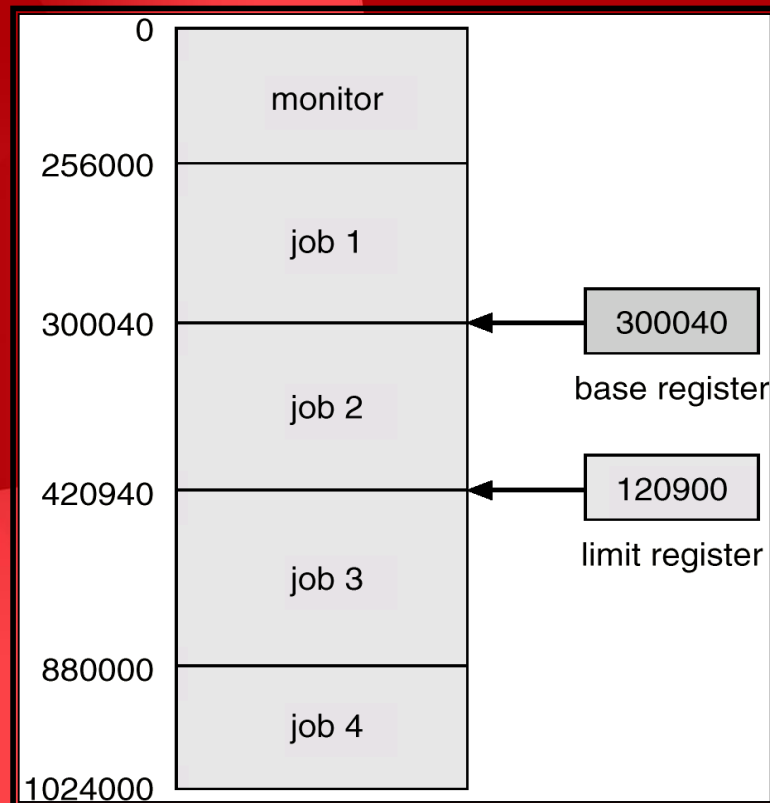
7 August 2021

Memory Protection

- Must provide memory protection outside the address space of a process.
- In order to have memory protection, add two registers that determine the range of legal addresses a program may access:
 - **Base register** – holds the smallest legal physical memory address.
 - **Limit register** – contains the size of the range
- Memory outside the defined range is protected.

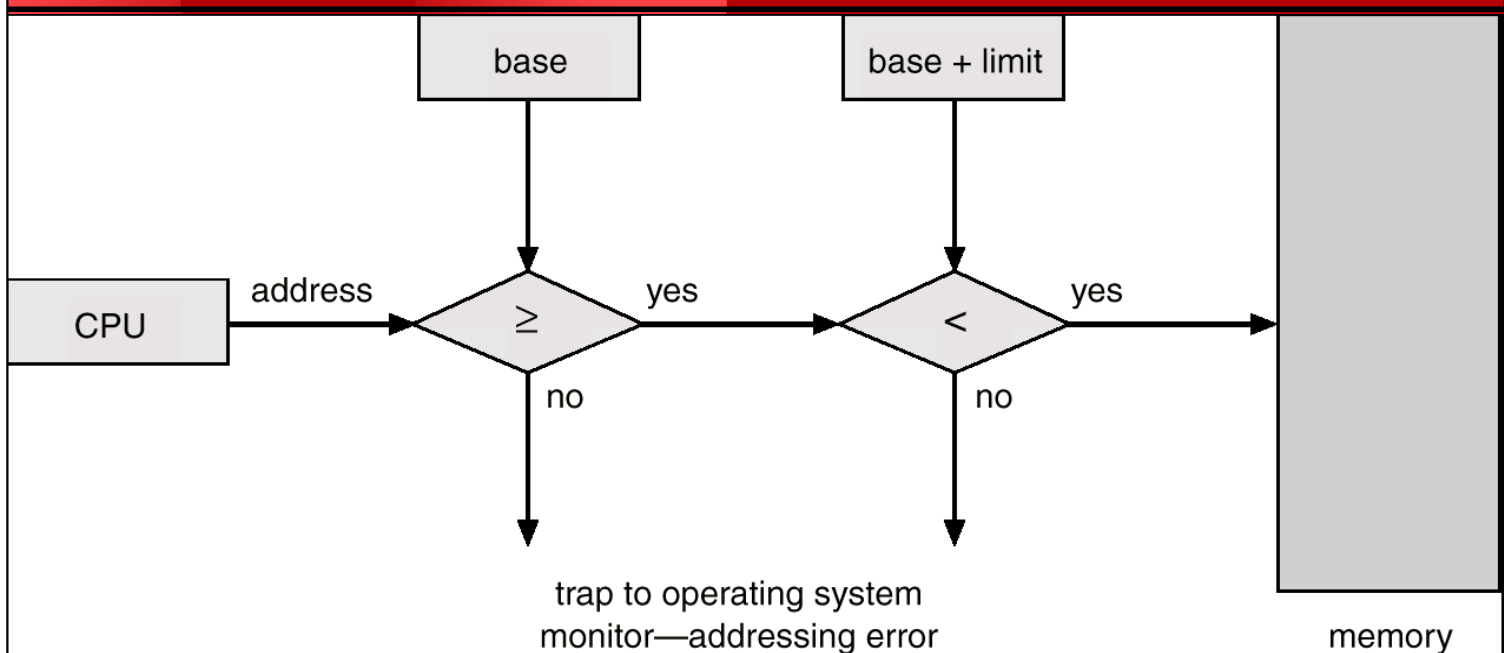
7 August 2021

Use of Base and Limit Register



7 August 2021

Hardware Support



7 August 2021

CPU Protection

- *Timer* – interrupts computer after specified period to ensure operating system maintains control.
 - Timer is decremented every clock tick.
 - When timer reaches the value 0, an interrupt occurs.
- Timer commonly used to implement time sharing.
- Time also used to compute the current time.
- Load-timer is a privileged instruction.

7 August 2021

© Copyright Virtual University of
Pakistan



OS Components

- Process management
- Main memory management
- Secondary storage management
- I/O system management
- File management
- Protection system
- Networking
- Command-line interpreter (shells)

7 August 2021

© Copyright Virtual University of
Pakistan



Operating System Services

Services for user and users of programs:

- **Program execution**
- **I/O Operations**
- **File System Manipulation**
- **Communications between processes/users**
- **Error detection and handling**

Operating System Services ...

Services for efficient system operation:

- **Resource management**
- **Accounting**
- **Protection**

OS Kernel

Users

Applications

Operating System API, AUI

Operating System Kernel

Computer Hardware

Real work is done in the kernel

Entry Points into Kernel



System Calls

- User processes must not be given open access to the kernel code
- The system call interface layer contains entry point in the kernel code
- Any user or application request that involves access to any system resource must be handled by the kernel code

Types Of System Calls

- Process Control
- File Management
- Device Management
- Information maintenance
- Communications

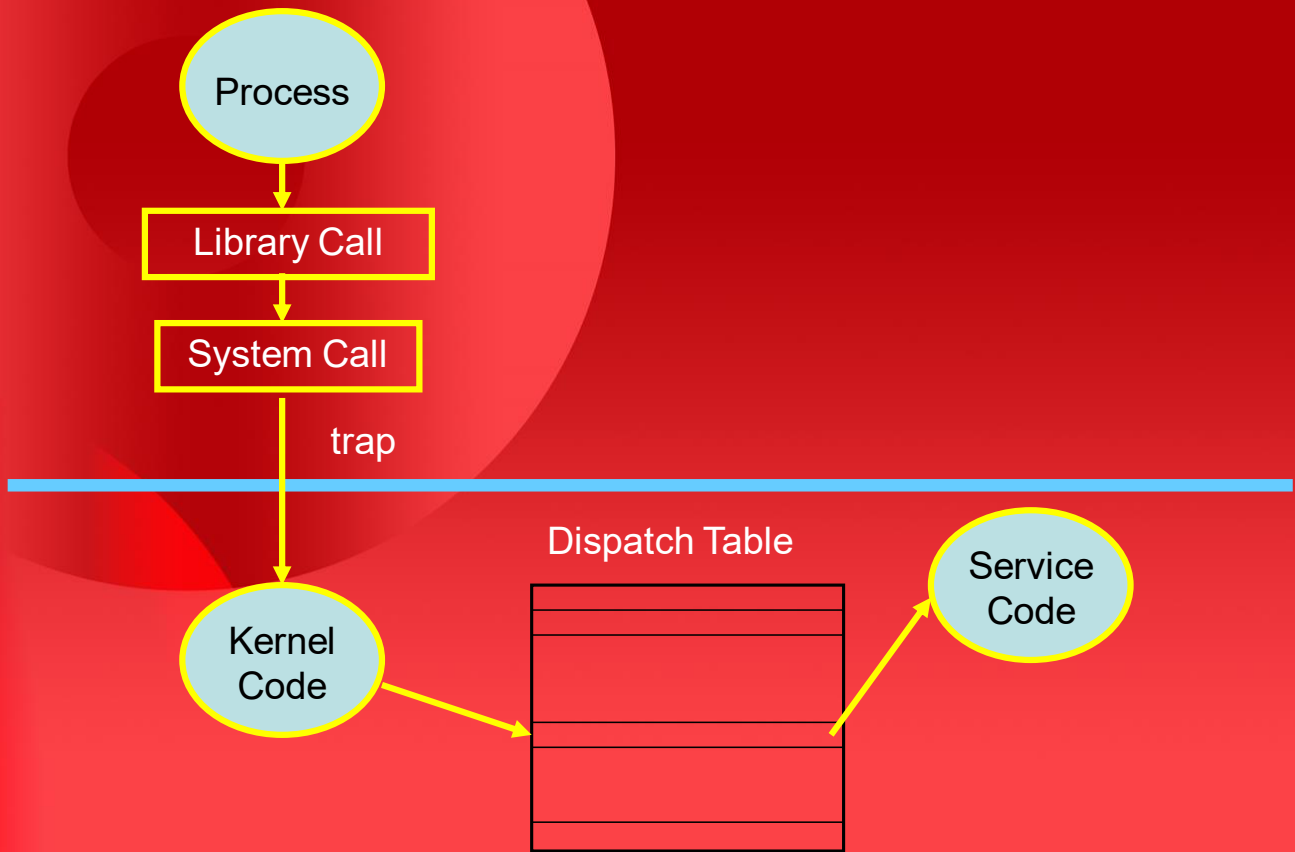
System Call Execution

- The user program makes a call to a library function.
- Library routine puts appropriate parameters at a well-known place (registers, stack, or a table in memory).
- The `trap` instruction is executed to change mode from user to kernel.
- Control goes to operating system.
- Operating system determines which system call is to be carried out.

Semantics of System Call Execution ...

- Kernel indexes the **dispatch table**, which contains pointers to service routines for system calls.
- Service routine is executed and return parameter or error code placed at well-known places (usually a CPU register).
- Control given back to user program.
- Library function executes the instruction following `trap`.

System Call ...



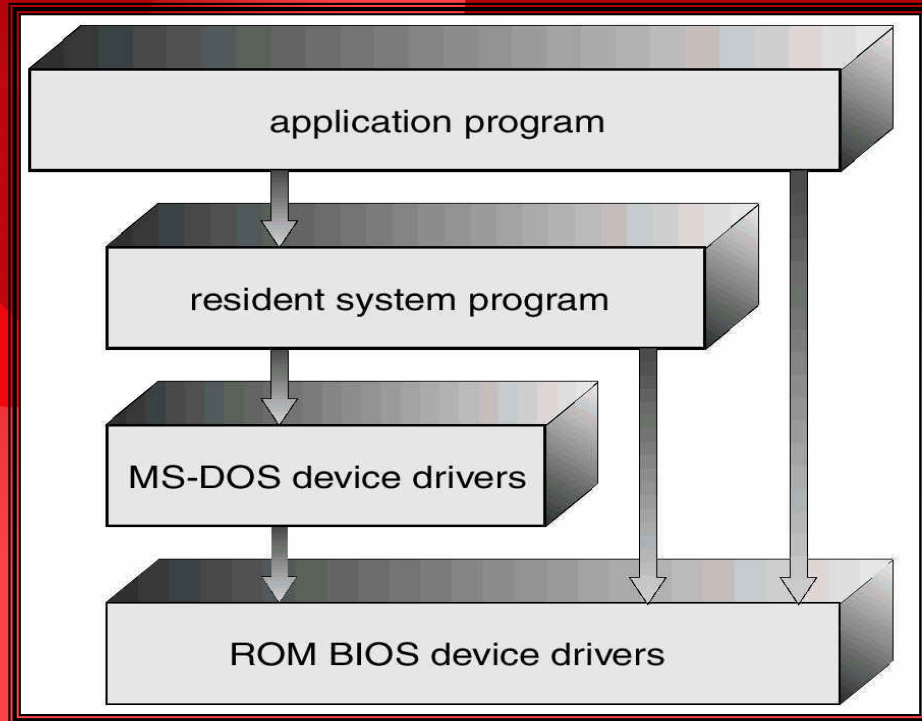
Operating System Structures

- Simple Structure
- Layered Approach
- Microkernels
- Virtual Machines



Simple Structures

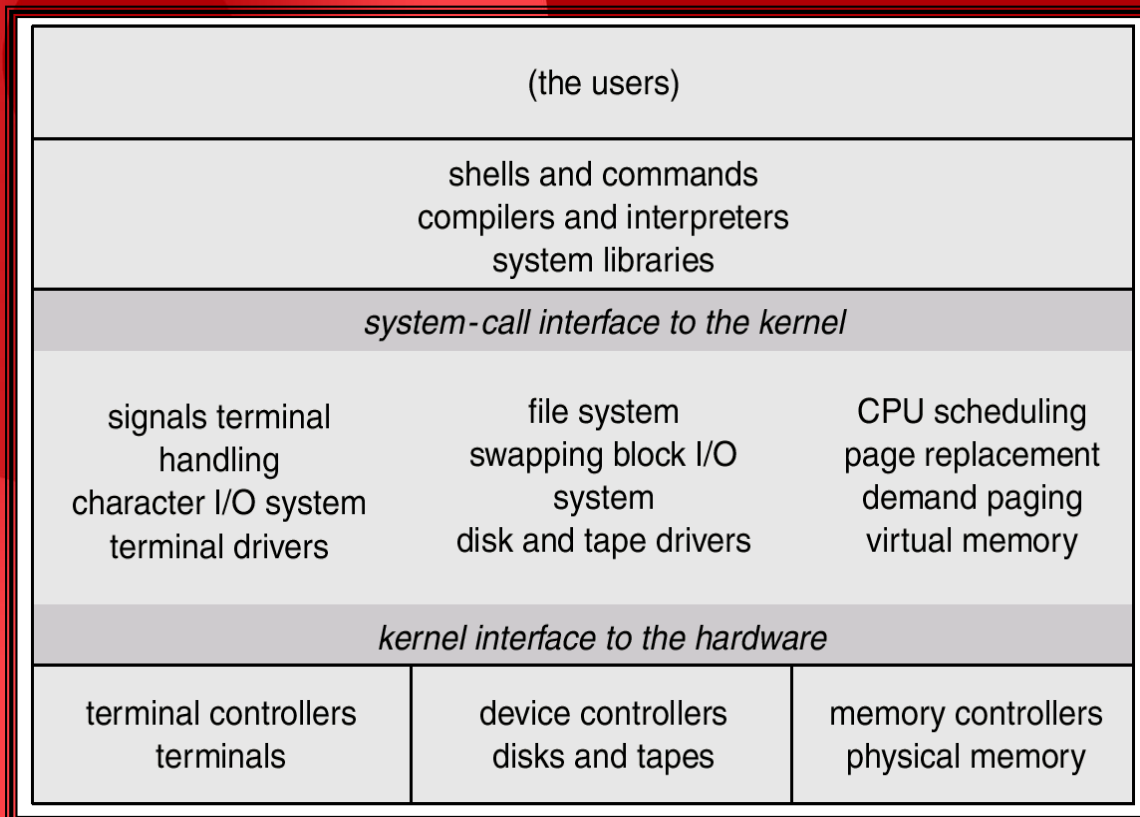
MS-DOS : written to provide the most functionality in the least space, so it was not divided into modules carefully



Simple Structures ...

- **UNIX** consists of two separable parts, the kernel and the system programs.
- Every thing below the system call interface and above the physical hardware is the kernel.
- An enormous amount of functionality combined in one level, UNIX is difficult to enhance as changes in one section could adversely affect other areas.

UNIX System Structure



Recap of lecture

- Review of previous lecture
- Operating system components
- Operating system services
- System calls
- Operating system structures
- Recap of the lecture

Operating Systems

Lecture 3

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 4



Agenda for Today

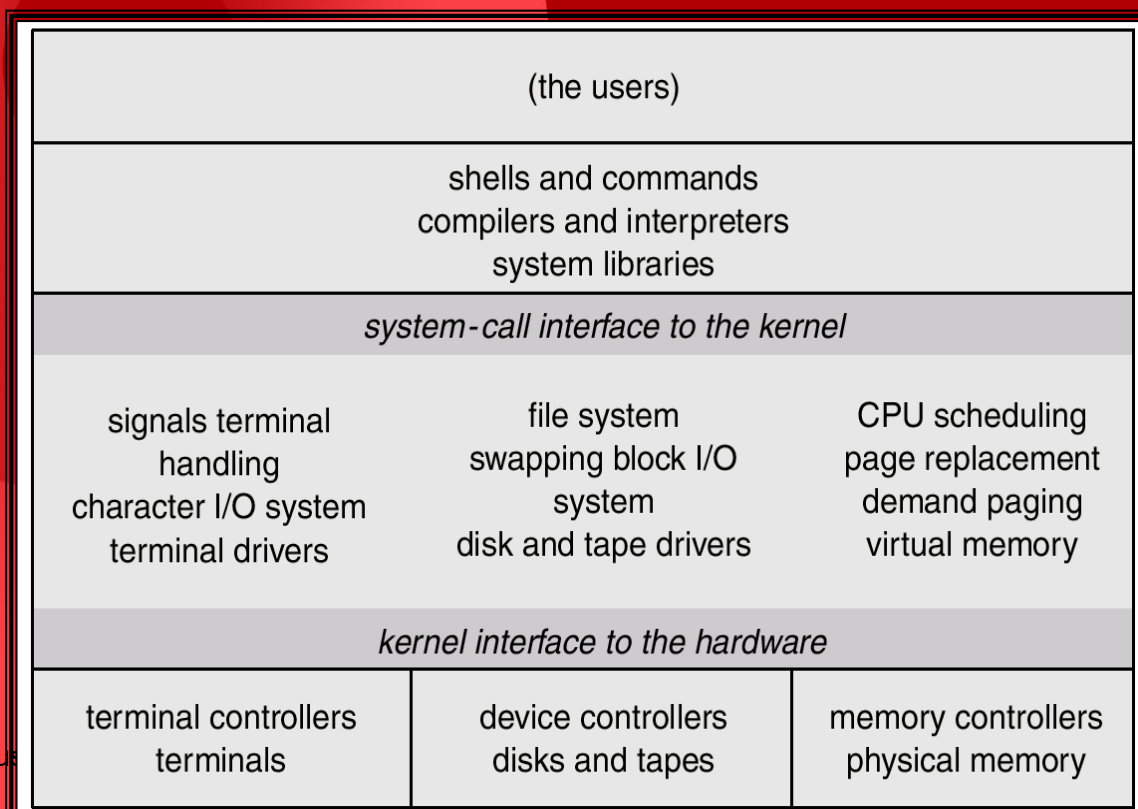
- Review of previous lecture
- Operating system structures
- Operating system design and implementation
- UNIX/Linux directory structure
- Browsing UNIX/Linux directory structure
- Useful UNIX/Linux commands
- Recap of the lecture

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX System Structure



7 August



Layered Approach

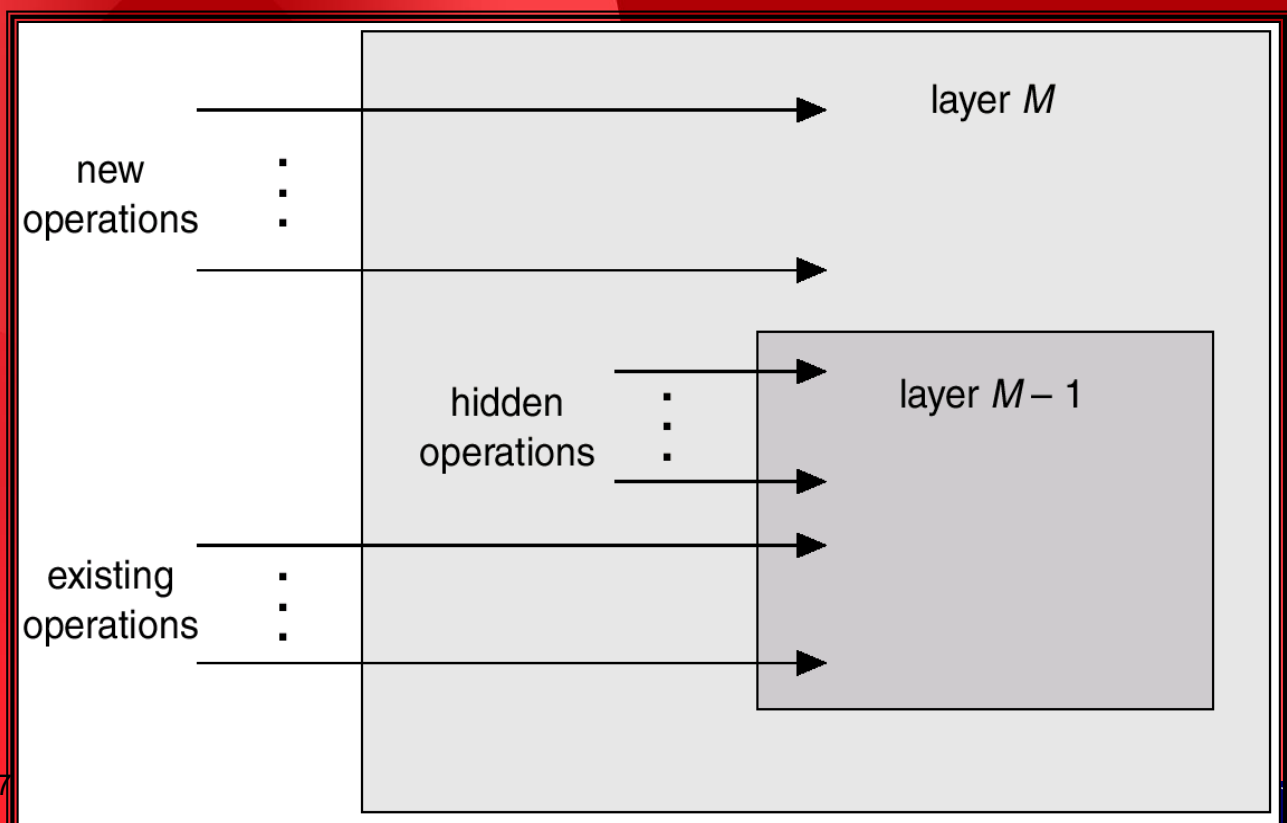
- The OS is broken up into a number of layers
- Bottom layer is hardware and the topmost layer (layer N) is the user interface
- A typical layer consists of data structures and a set of routines to service the layer above it
- THE operating system by Dijkstra
- IBM's OS/2

7 August 2021

© Copyright Virtual University of
Pakistan



Layered Approach ...



Layered Approach ...

- Modularity
- Each layer uses functions and services of only lower layers
- Simplifies debugging and system verification.
- The major difficulty with layered approach is careful definition of layers, because a layer can only use the layers below it
- Less efficient than other approaches

7 August 2021

© Copyright Virtual University of
Pakistan



Microkernel

- Structures the operating system by removing all non-essential components from the kernel and implementing them as system and user level programs
- Smaller kernel
- Main function is to provide a communication facility between client programs and the various services that are also running in the user space.

7 August 2021

© Copyright Virtual University of
Pakistan



Microkernel ...

- Easier to extend the OS—new services are added to user space and consequently do not require modification of the kernel and/or its recompilation
- Easier to maintain operating system code (enhancement, debugging, etc.)
- OS is easier to port from one hardware to another
- More security and reliability
- Mach, MacOS X Server, QNX, OS/2, and

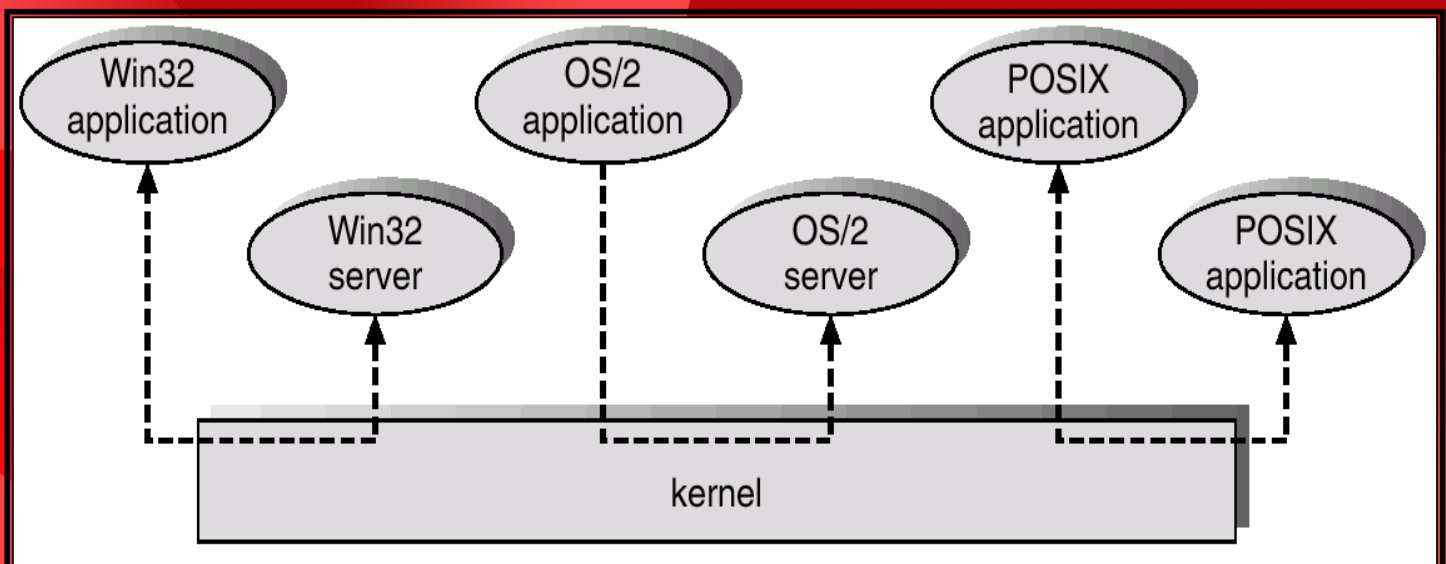
7 August 2021

Windows NT

© Copyright Virtual University of
Pakistan



Windows NT Client-Server Structure



7 August 2021

© Copyright Virtual University of
Pakistan



Virtual Machines

- CPU scheduling and virtual memory techniques used to emulate hardware of the underlying machine, on which user can install an operating system that the virtual machine supports
- On a time-sharing system with virtual machine support, users may be working on different operating systems
- Pioneered by IBM VM operating system that ran CMS, a single-user interactive operating system

7 August 2021

© Copyright Virtual University of
Pakistan



Virtual Machines ...

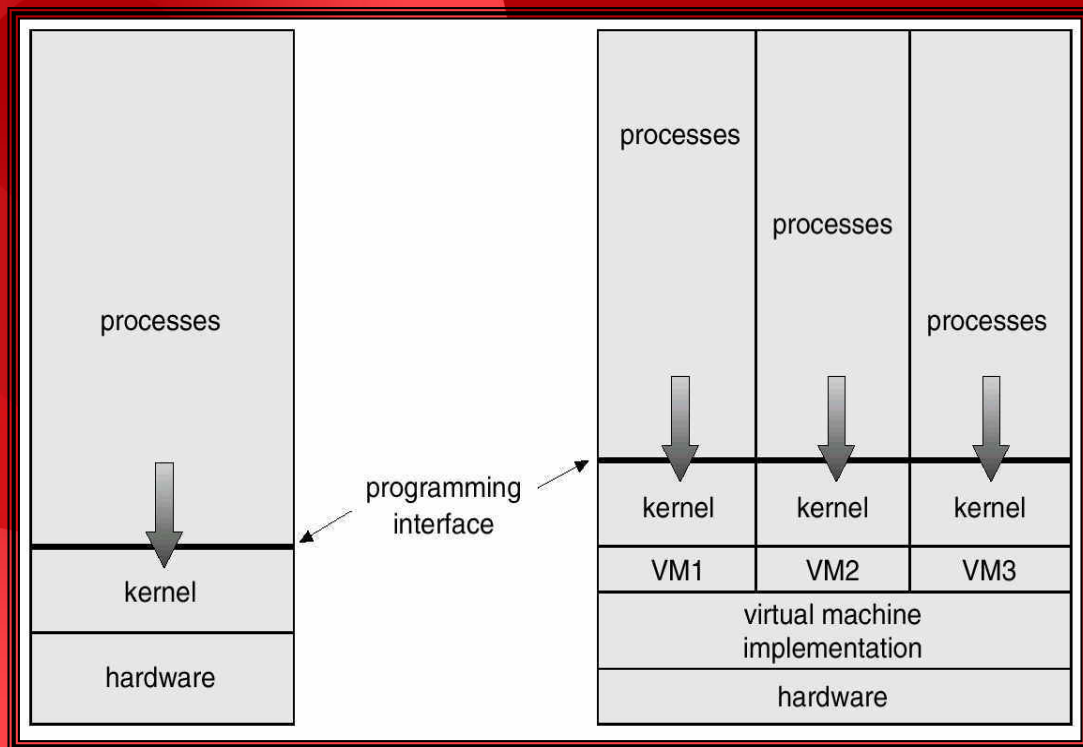
- Difficult to implement.
- System development done without disrupting normal system operation.

7 August 2021

© Copyright Virtual University of
Pakistan



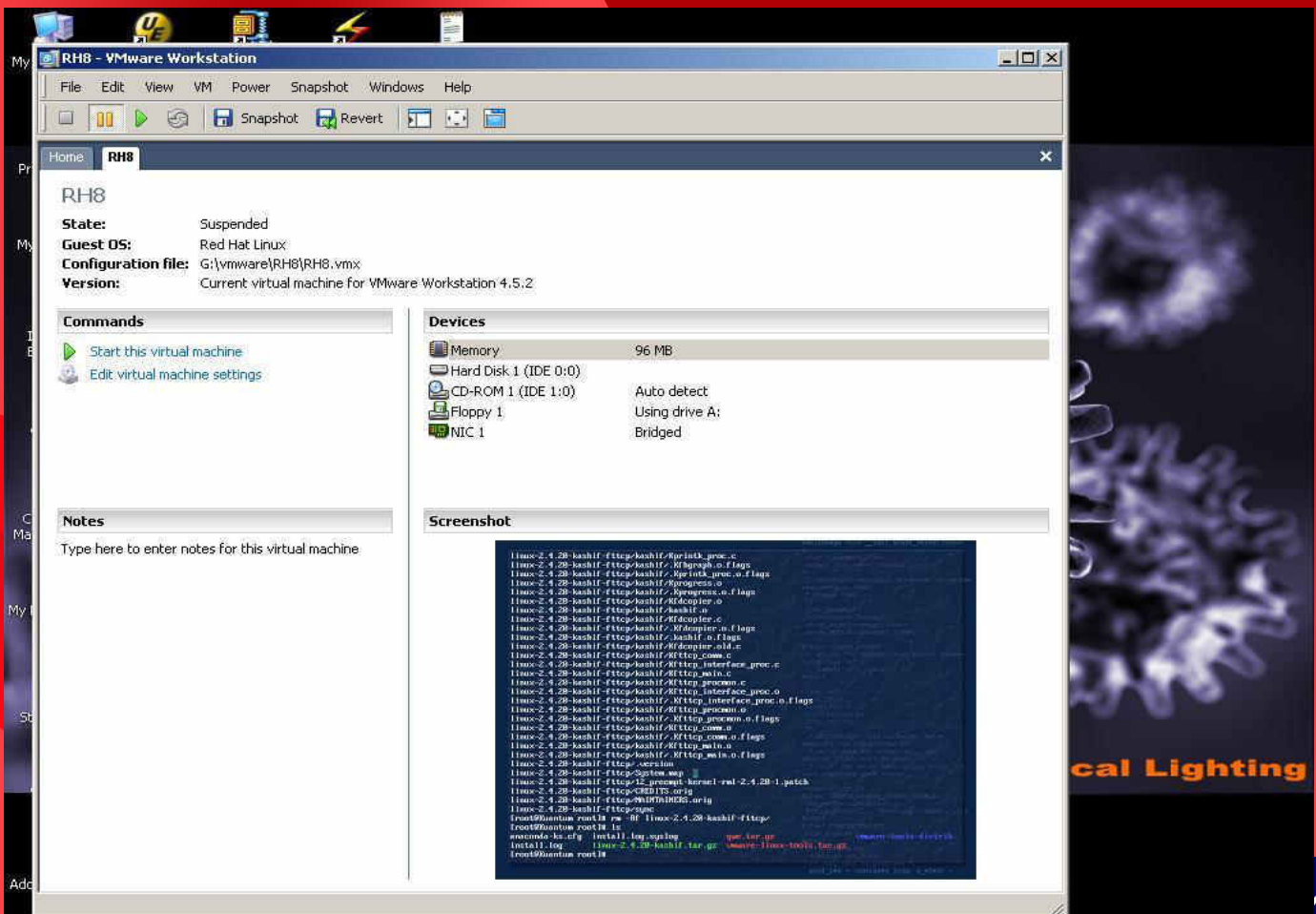
Virtual Machines ...



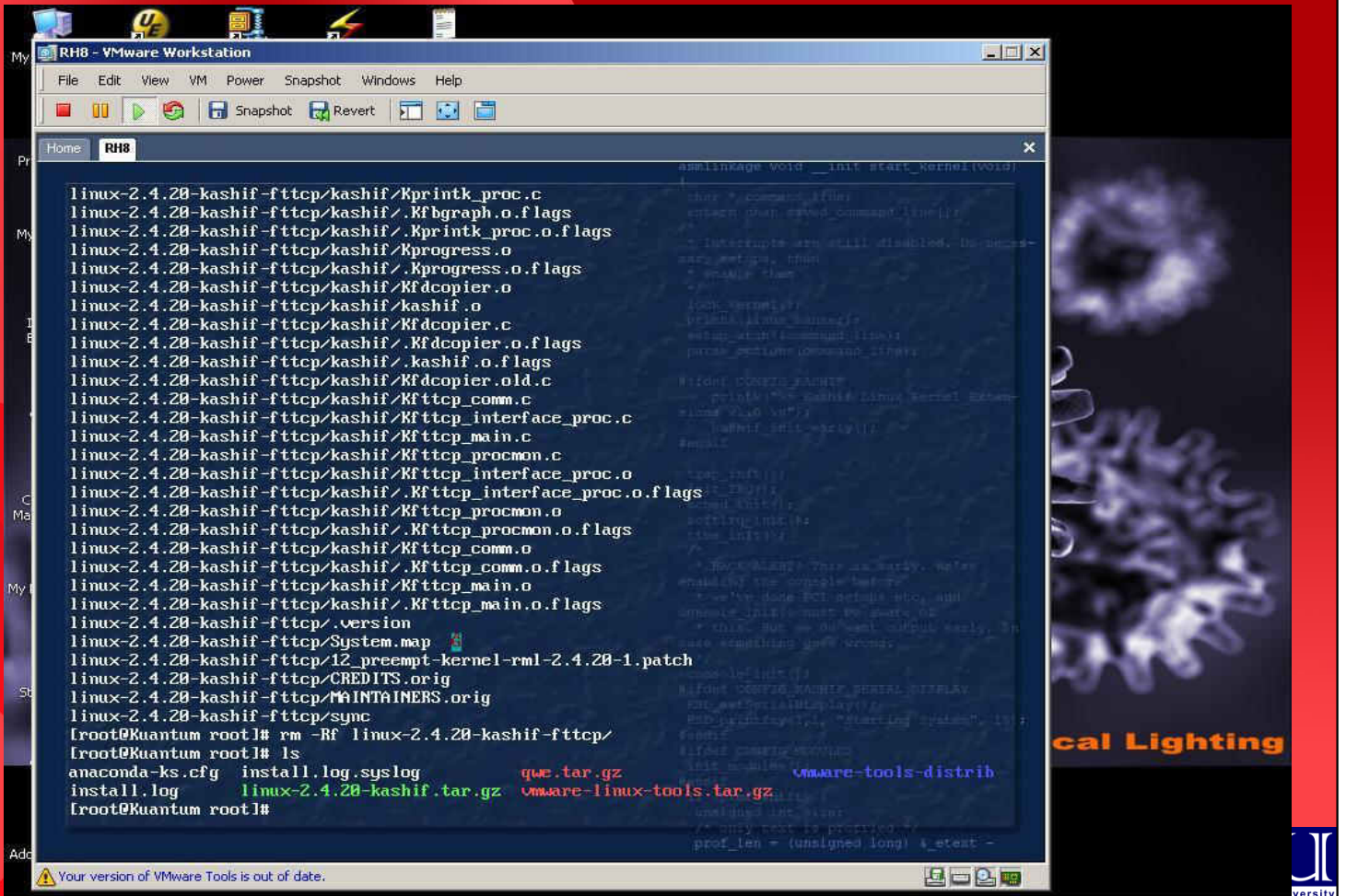
7 August 2021 Non Virtual Machine Copyright Virtual University of Pakistan Virtual Machine



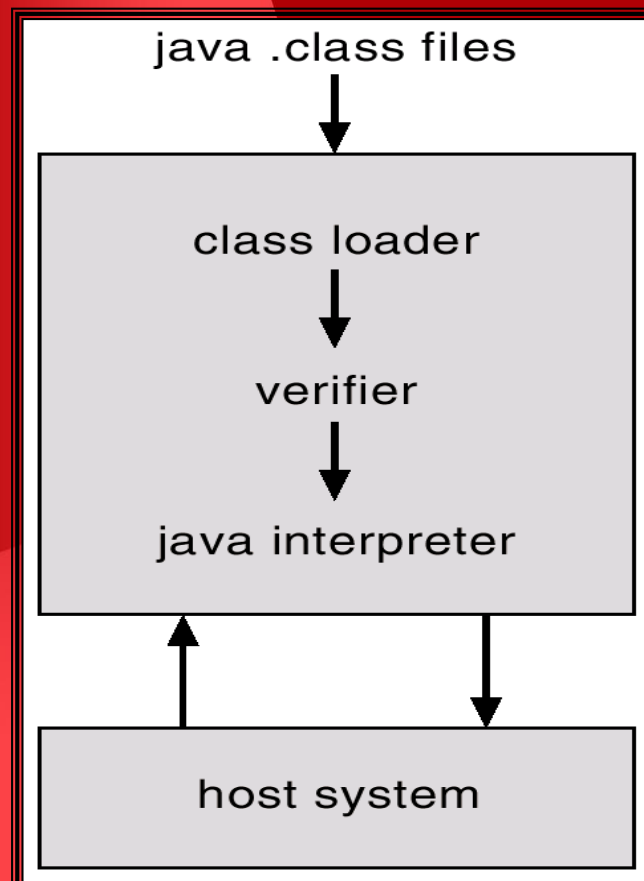
VMWare on Windows



VMWare on Windows



Java Virtual Machine



System Design and Implementation

- Design Goals
- **User:** operating system should be convenient to use, easy to learn, reliable, safe, and fast.
- **System designer and administrator:** operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.

7 August 2021

© Copyright Virtual University of
Pakistan



System Design and Implementation ...

- **Mechanism:** determine how to do something
- **Policy:** determine what will be done
- Separation of mechanism and policy is important for flexibility.

7 August 2021

© Copyright Virtual University of
Pakistan



System Design and Implementation ...

Implementation in:

- Assembly language
- Higher level languages:
 - Easier to code
 - Compact code
 - Easier to port

7 August 2021

© Copyright Virtual University of
Pakistan



Introduction to UNIX and Linux

- Written by Dennis Ritchie and Ken Thomsom in at Bell Labs in 1969
- Initially written in assembly language and a high-level language called B. Later converted from B to C language.
- Linux written by Linus Torvalds (an undergraduate student at the Univ. of Helsinki, Finland) in 1991.
- Most popular operating systems
- Internet runs on UNIX and Linux

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX/Linux File System

- UNIX has a **hierarchical file system structure** consisting of a **root directory** with other directories and files hanging under it
- In a **command-line user interface**, typed commands are used to navigate the system
- Directories and files are specified by **filenames**
 - cs604/assignments/assign1.c
 - /home/students/haroon/courses/cs604

7 August 2021

© Copyright Virtual University of
Pakistan



Browsing Directory Structure

/	The root directory is the directory that contains all other directories. When a directory structure is displayed as a tree, the root directory is at the top.
/bin	This directory holds binary executable files that are essential for correct operation of the system
/boot	This directory includes essential system boot files including the kernel image .

7 August 2021

© Copyright Virtual University of
Pakistan



Browsing Directory Structure ...

/dev	This directory contains the devices available to on the machine
/etc	Linux uses this directory to store system configuration files
/home	This is where every user on a Linux system has a personal directory
/lib	Shared libraries and kernel modules are stored in this directory

7 August 2021

© Copyright Virtual University of
Pakistan



Browsing Directory Structure ...

/root	The home directory for the superuser
/sbin	Utilities used for system administration (halt, ifconfig, fdisk, etc.) are stored in this directory
/tmp	Used for storing temporary files. Similar to C:\Windows\Temp.

7 August 2021

© Copyright Virtual University of
Pakistan



Browsing Directory Structure ...

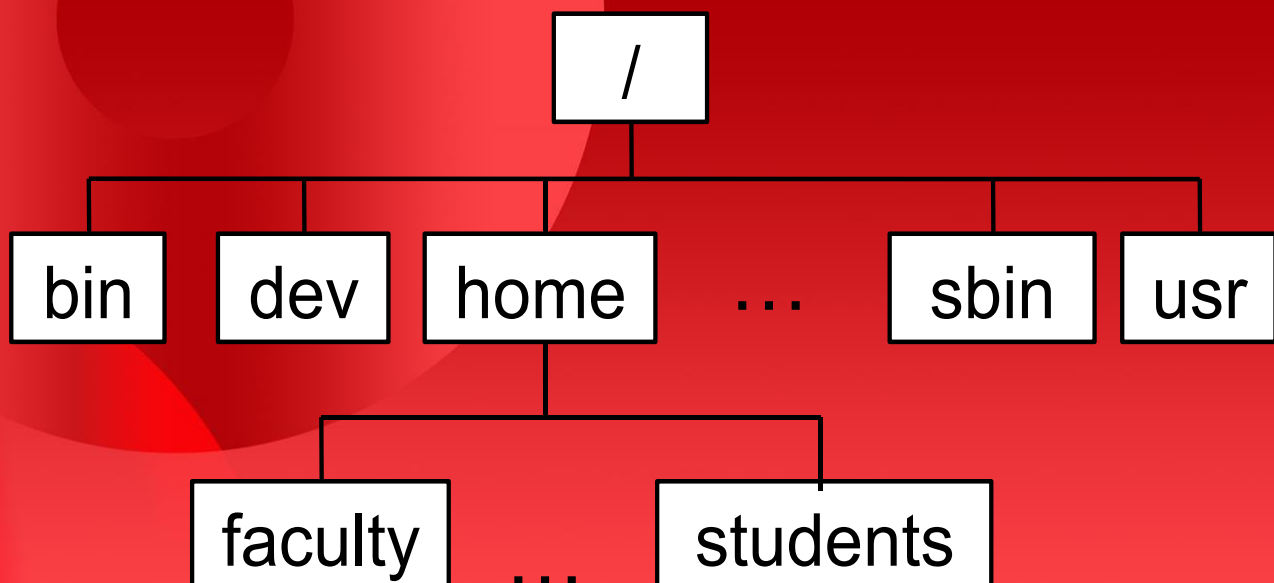
/usr	Typically a shareable, read-only directory. Contains user applications and supporting files for those applications.
/var	This directory contains variable data files such as logs (/var/log), mail (/var/mail), and spools (/var/spool) among other things.

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX/Linux Directory Hierarchy

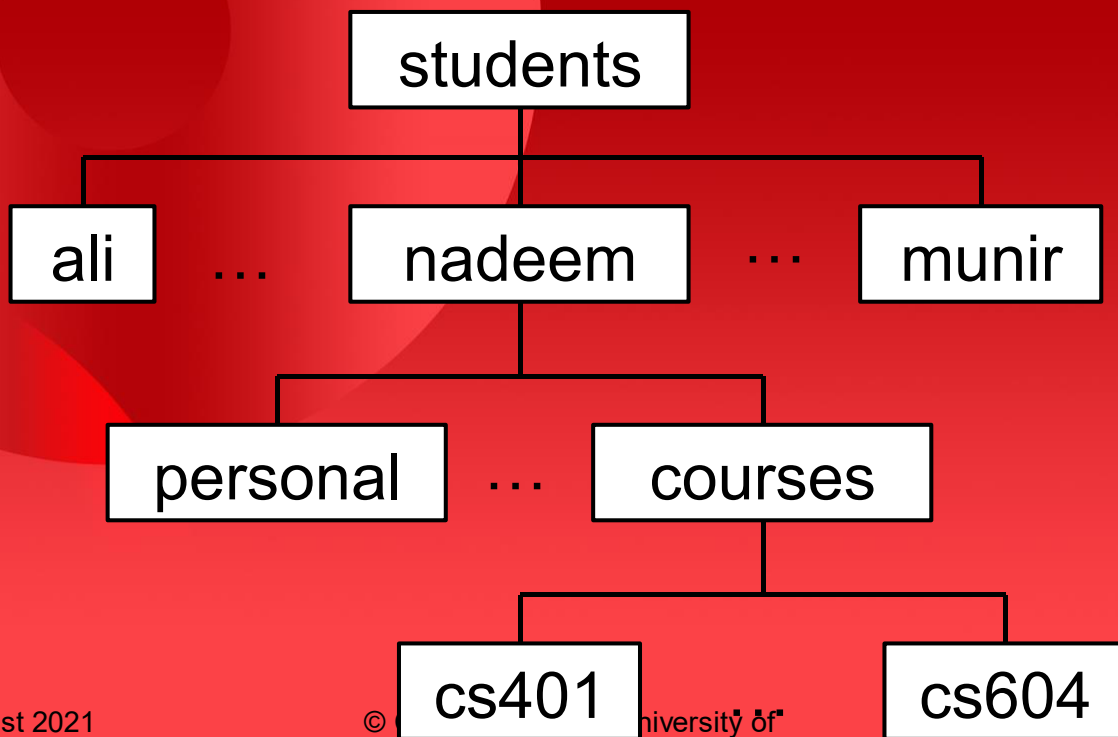


7 August 2021

© Copyright Virtual University of
Pakistan



UNIX/Linux Directory Hierarchy



7 August 2021

© Virtual University of
Pakistan



Recap

- Review of previous lecture
- Operating system structures
- Operating system design and implementation
- UNIX/Linux directory structure
- Recap of the lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 4

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 5



Agenda for Today

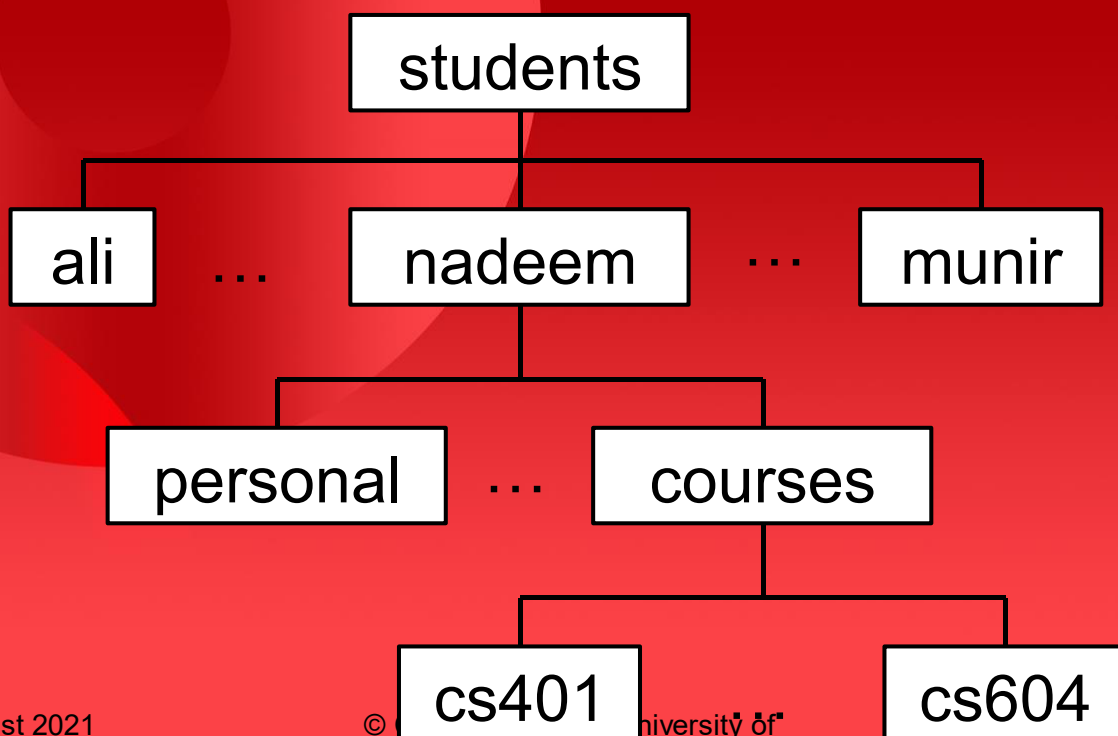
- Review of previous lecture
- Browsing UNIX/Linux directory structure
- Useful UNIX/Linux commands
- Process concept
- Process scheduling concepts
- Process creation and termination
- Recap of the lecture

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX/Linux Directory Hierarchy



7 August 2021

© Virtual University of
Pakistan



UNIX/Linux Directory Hierarchy

- Root directory (/)
- Home/login directory (~, \$HOME, \$home)
- Current working directory (.)
- Parent of the current working directory (..)

7 August 2021

© Copyright Virtual University of
Pakistan



Browsing the File Hierarchy

- ls Display contents of a directory
- cd Change directory
- pwd Print working directory
- mkdir Create directory
- rmdir Remove directory
- cp Copy file
- mv Move file
- rm Remove file

7 August 2021

© Copyright Virtual University of
Pakistan



Browsing the File Hierarchy

- `ls` Display contents of a directory
- `cd` Change directory
- `pwd` Print working directory
- `mkdir` Create directory
- `rmdir` Remove directory
- `cp` Copy file
- `mv` Move file
- `rm` Remove file

7 August 2021

© Copyright Virtual University of
Pakistan



Browsing the File Hierarchy

- `mkdir temp`
Create the 'temp' directory in your current directory
`mkdir ~/courses/cs604/programs`
Create the 'programs' directory in your ~/courses/cs604 directory
- `rmdir ~/courses/cs604/programs`
Remove the 'programs' directory under your ~/courses/cs604 directory

7 August 2021

© Copyright Virtual University of
Pakistan



Browsing the File Hierarchy

- `cp file1 file2`

Copy 'file1' in your current directory to 'file2' in your current directory

```
cp ~/file1 ~/memos/file2
```

Copy '~/'file1' to '~/'memos/file2'

- `mv file1 file2`

Move 'file1' in your current directory to 'file2' in your current directory

```
mv ~/file1 ~/memos/file2
```

Move '~/'file1' to '~/'memos/file2'

7 August 2021

© Copyright Virtual University of
Pakistan



Browsing the File Hierarchy

- `rm file1`

Remove 'file1' from your current directory

```
rm ~/courses/cs604/programs/test.c
```

Remove 'test1' in the 'programs' directory in your ~/courses/cs604 directory

- `rm *.o`

Remove all .o (i.e., object) files from your current directory

7 August 2021

© Copyright Virtual University of
Pakistan



Compiling and Running C Programs

```
$ gcc program.c
```

```
$ ../a.out
```

```
[ program output ]
```

```
$ gcc program.c -o assignment
```

```
$ assignment
```

```
[ program output ]
```

```
$ gcc program.c -o assignment -lm
```

```
$ assignment
```

```
[ program output ]
```

```
$ August 2021
```

© Copyright Virtual University of
Pakistan



Useful Internet Resources

UNIX Tutorial for Beginners

<http://www.ee.surrey.ac.uk/Teaching/Unix/>

<http://www.isu.edu/departments/comcom/unix/workshop/unixindex.html>

Useful Internet Resources

emacs tutorial

<http://www.linuxjunkies.org/programming/IDE/emacs/>

vi tutorial

<http://www.networkcomputing.com/unixworld/tutorial/009/009.html>

https://engineering.purdue.edu/ECN/Resources/KnowledgeBase/search_results?query=vi

pico tutorial

<http://www.itd.umich.edu/itdoc/r/r1168/>

7 August 2021

© Copyright Virtual University of
Pakistan



What is a process?

- Process – a program in execution; process execution must progress in sequential fashion.
- A process consists of:
 - Code (text) section
 - Data section
 - Stack
 - Heap
 - Environment
 - CPU state (program counter, etc.)
 - Process control block (PCB)

7 August 2021

© Copyright Virtual University of
Pakistan



CPU and I/O Bound Processes

Processes can be:

- *I/O-bound process* – spends more time doing I/O than computations, many short CPU bursts.



- *CPU-bound process* – spends more time doing computations; few very long CPU bursts.



7 August 2021

© Copyright Virtual University of
Pakistan



Process States

As a process executes, it changes *state*

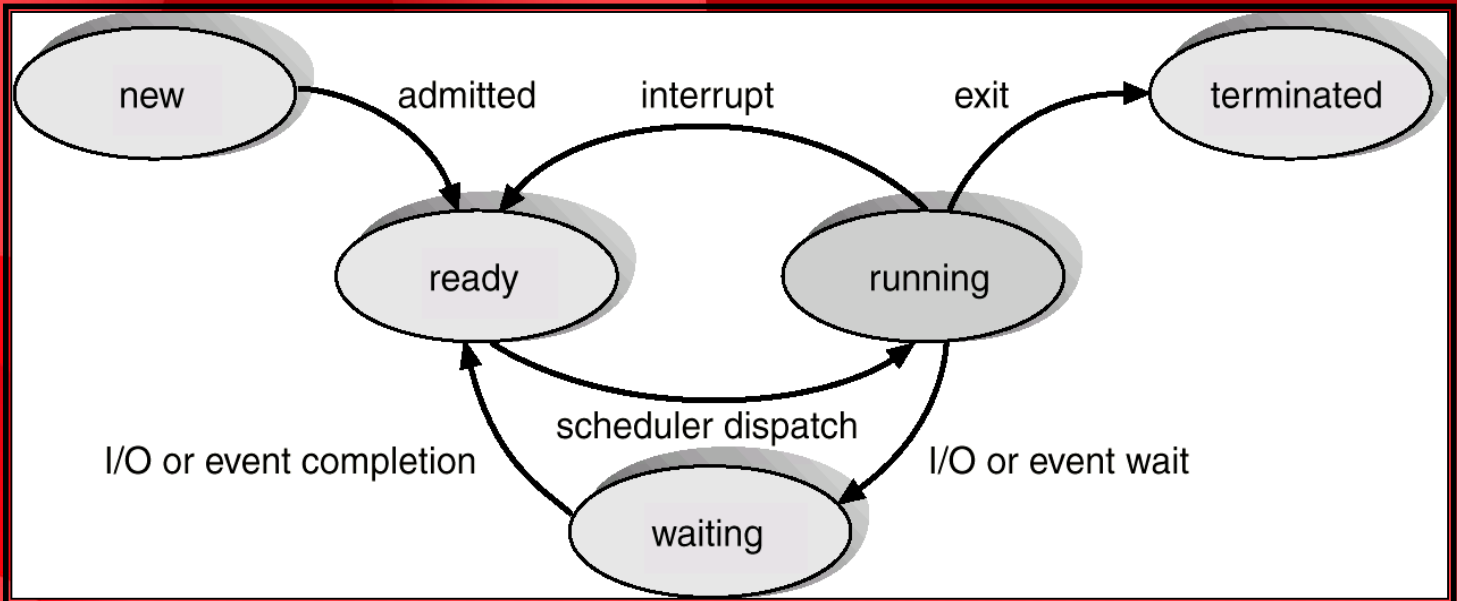
- **new:** The process is being created.
- **ready:** The process is waiting to be assigned to a processor.
- **running:** Instructions are being executed.
- **waiting:** The process is waiting for some event to occur.
- **terminated:** The process has finished execution.

7 August 2021

© Copyright Virtual University of
Pakistan



Process States



7 August 2021

© Copyright Virtual University of
Pakistan



Process Control Block (PCB)

Process information and attributes

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information
- Per process file table
- Process ID (PID)
- Parent PID, etc.

7 August 2021

© Copyright Virtual University of
Pakistan



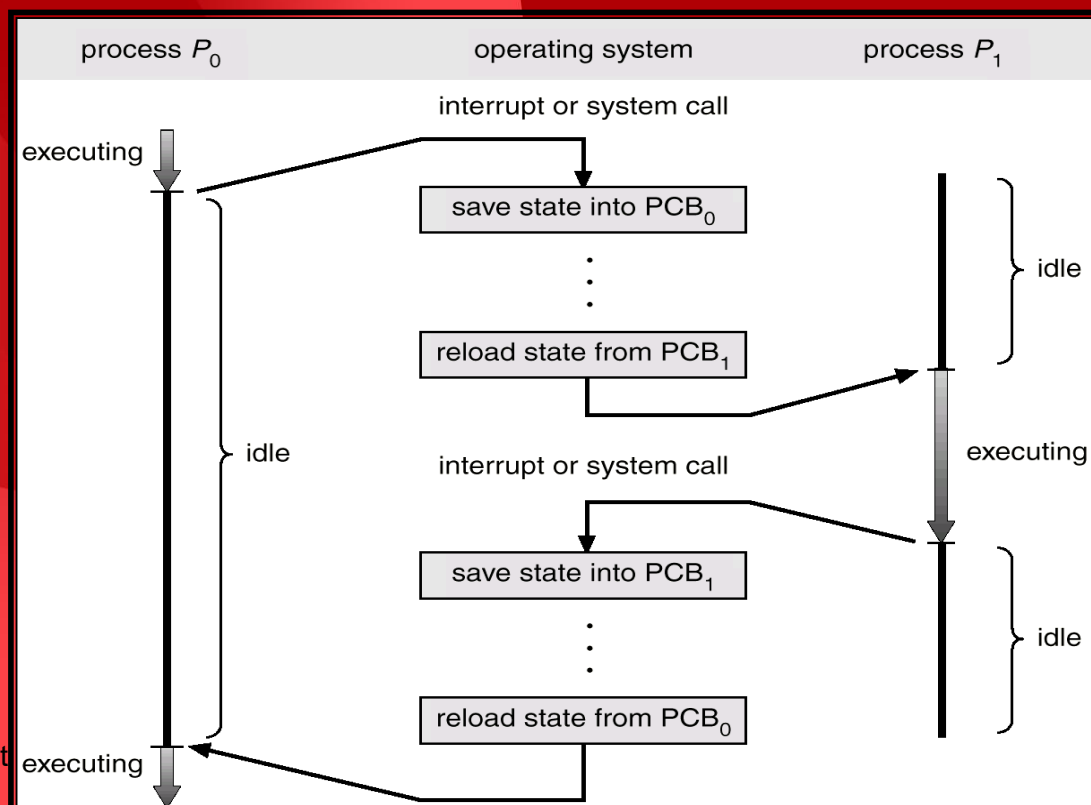
Process Control Block (PCB)

pointer	process state
process number	
program counter	
registers	
memory limits	
list of open files	
⋮	

7 August 2021



CPU Switch From Process to Process



7 August



Process Scheduling Queues

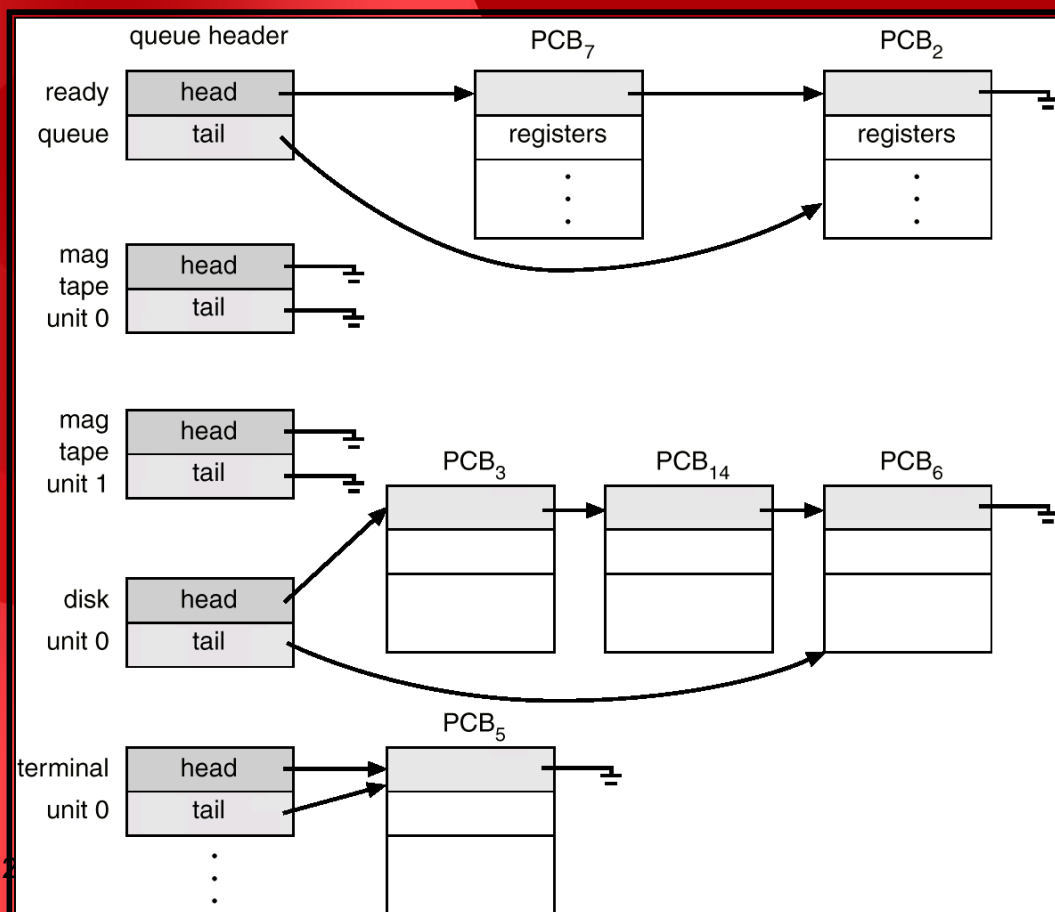
- Job queue – set of all processes in the system.
- Ready queue – set of all processes residing in main memory, ready and waiting to execute.
- Device queues – set of processes waiting for I/O devices.
- Process migration between the various queues.

7 August 2021

© Copyright Virtual University of Pakistan



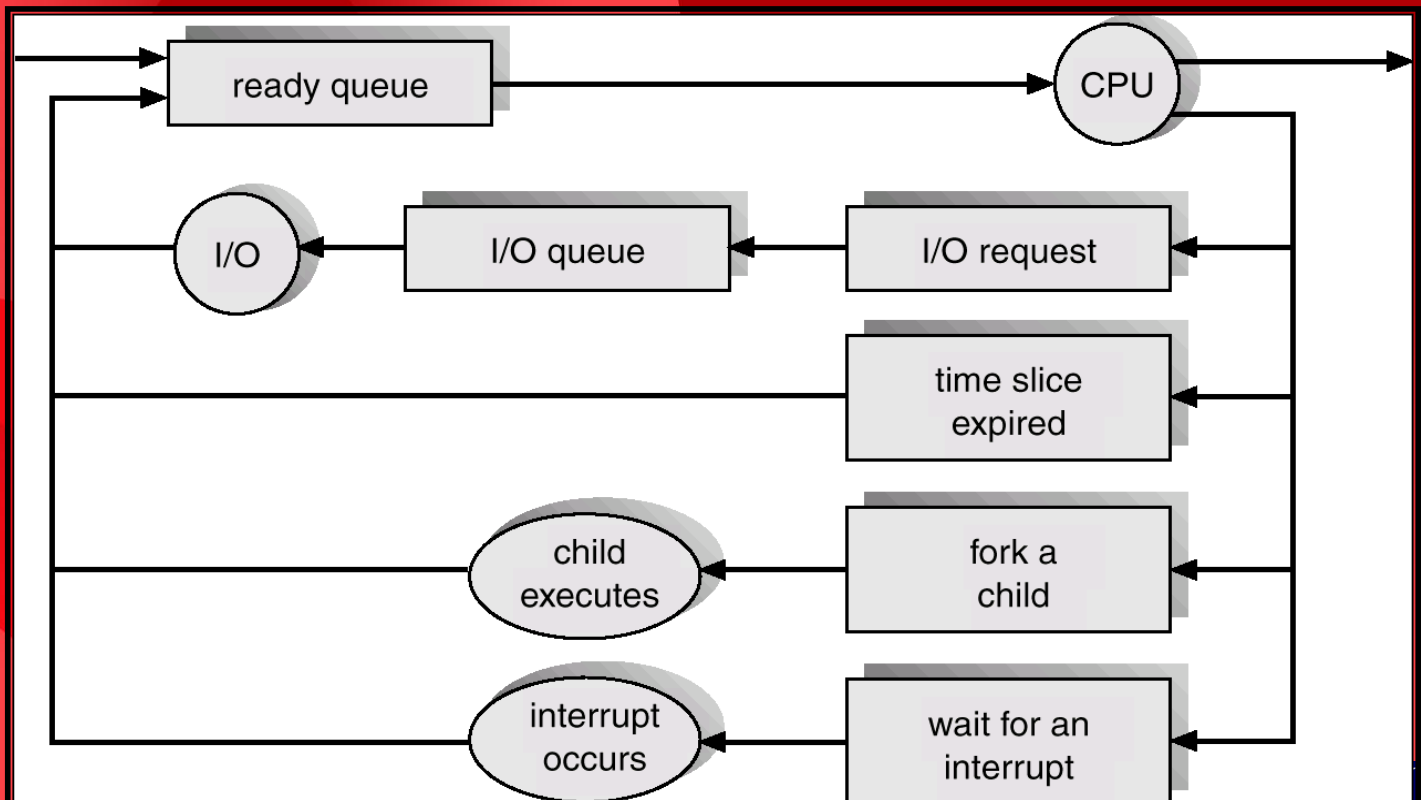
Queues in the OS



7 August 2021



Queues in a Computer System



Schedulers

- Long term scheduler
- Short term scheduler
- Medium term scheduler

Recap of Lecture

- Review of previous lecture
- Browsing UNIX/Linux directory structure
- Useful UNIX/Linux commands
- Process concept
- Process scheduling concepts

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 6



Agenda for Today

- Review of previous lecture
- Process scheduling concepts
- Process creation and termination
- Process management in UNIX/Linux—
system calls: `fork`, `exec`, `wait`, `exit`
- Sample codes
- Recap of the lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 5

- Browsing the UNIX/Linux directory structure
- Useful UNIX/Linux commands
- Process concept—attributes, states, types (CPU- and I/O-bound), PCB, OS queues, CPU scheduling, and context switch

7 August 2021

© Copyright Virtual University of
Pakistan



Schedulers

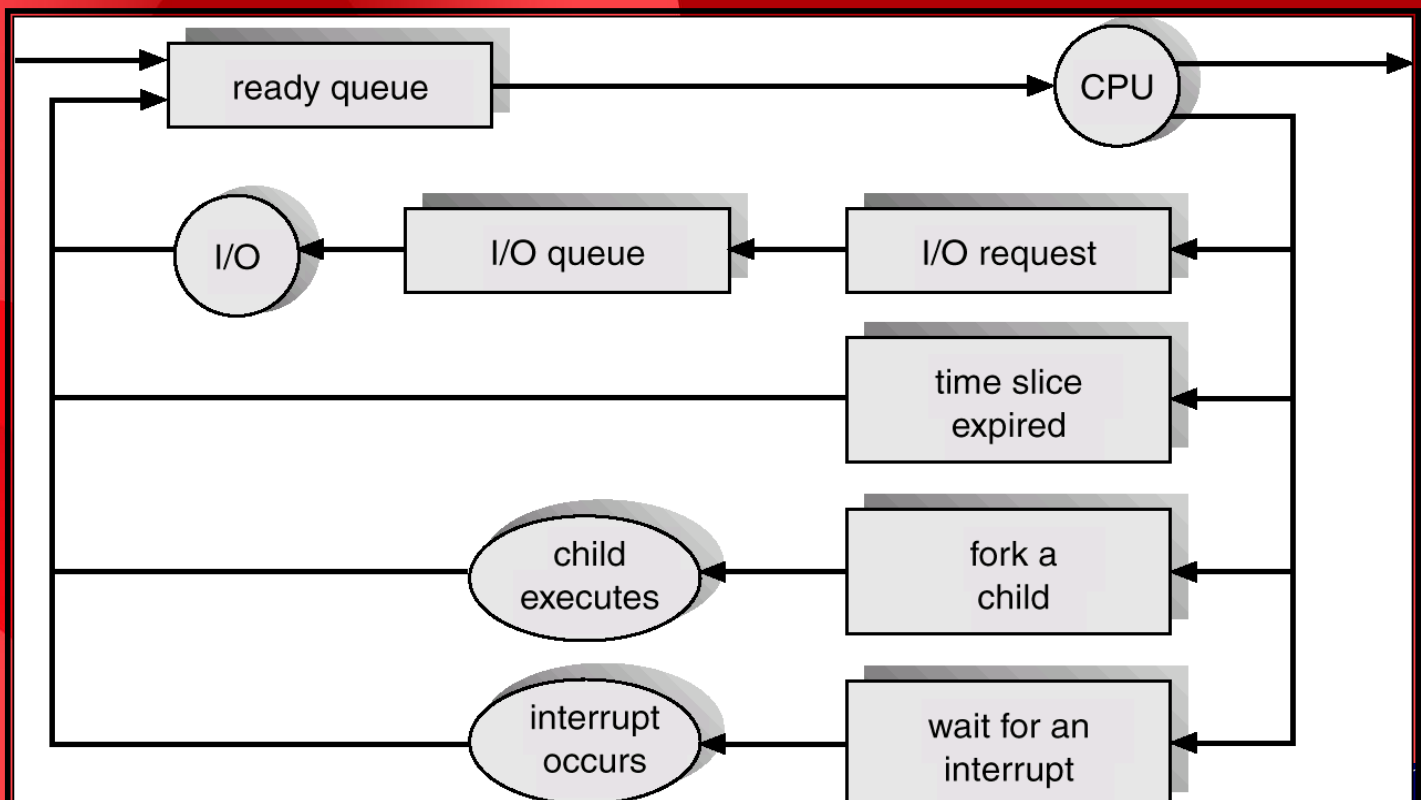
- Long term scheduler
- Short term scheduler
- Medium term scheduler

7 August 2021

© Copyright Virtual University of
Pakistan



Queues in a Computer System



Long Term Scheduler

- Long-term scheduler (or job scheduler) – selects processes from the job pool to be brought into the ready queue.
 - Long-term scheduler is invoked very infrequently (seconds, minutes) $\Rightarrow$ (may be slow).
 - The long-term scheduler controls the *degree of multiprogramming*.
 - More processes, smaller percentage of time each process is executed

7 August 2021

© Copyright Virtual University of
Pakistan



Short Term Scheduler

- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates it the CPU through the **dispatcher**.
 - Short-term scheduler is invoked very frequently (milliseconds) $\Rightarrow$ (must be fast).
 - Invoked when following events occur
 - CPU slice of the current process finishes
 - Current process needs to wait for an event
 - Clock interrupt
 - I/O interrupt
 - System call
 - Signal

7 August 2021

© Copyright Virtual University of
Pakistan



Medium Term Scheduler

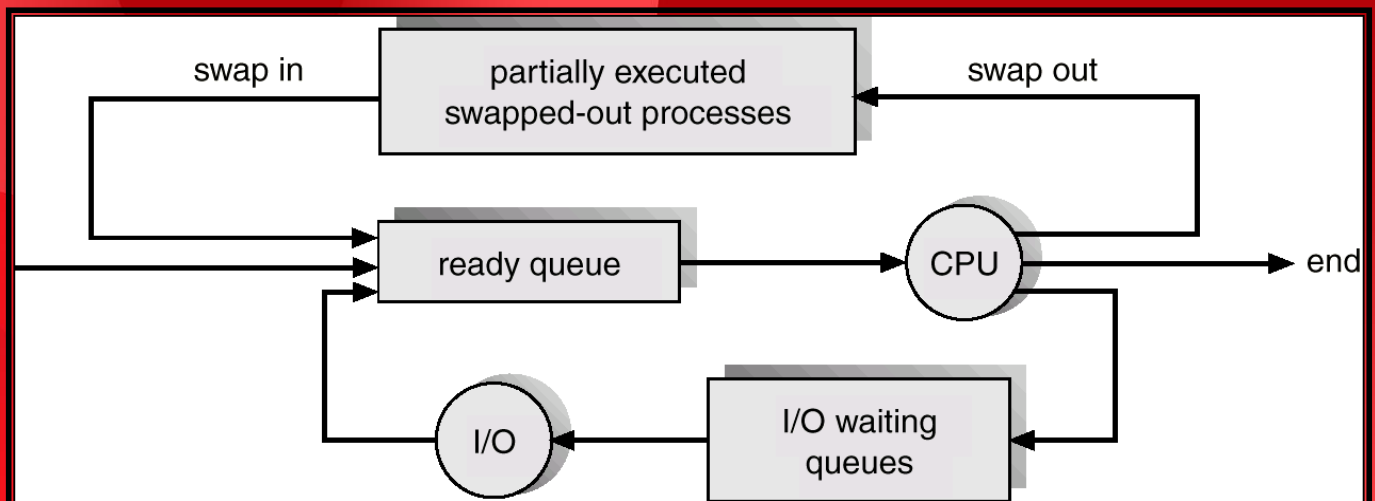
- Also known as **swapper**
- Selects an in-memory process and swaps it out to the disk temporarily
- Swapping decision is based on several factors
 - Arrival of a higher priority process but no memory available
 - Poor mix of jobs
 - Memory request of a process cannot be met

7 August 2021

© Copyright Virtual University of
Pakistan



Addition of Medium Term Scheduling



7 August 2021

© Copyright Virtual University of
Pakistan



Context Switch

- When CPU switches to another process, the system must save the state (context) of the 'current' (old) process and load the saved state for the new process.
- Context-switch time is overhead; the system does no useful work while switching.
- Time dependent on hardware support; typically in microseconds

7 August 2021

© Copyright Virtual University of
Pakistan



Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes.
- Resource sharing
 - Parent and children share all resources.
 - Children share a subset of parent's resources.
 - Parent and child share no resources.
- Execution
 - Parent and children execute concurrently.
 - Parent waits until children terminate.

7 August 2021

© Copyright Virtual University of
Pakistan



Process Creation ...

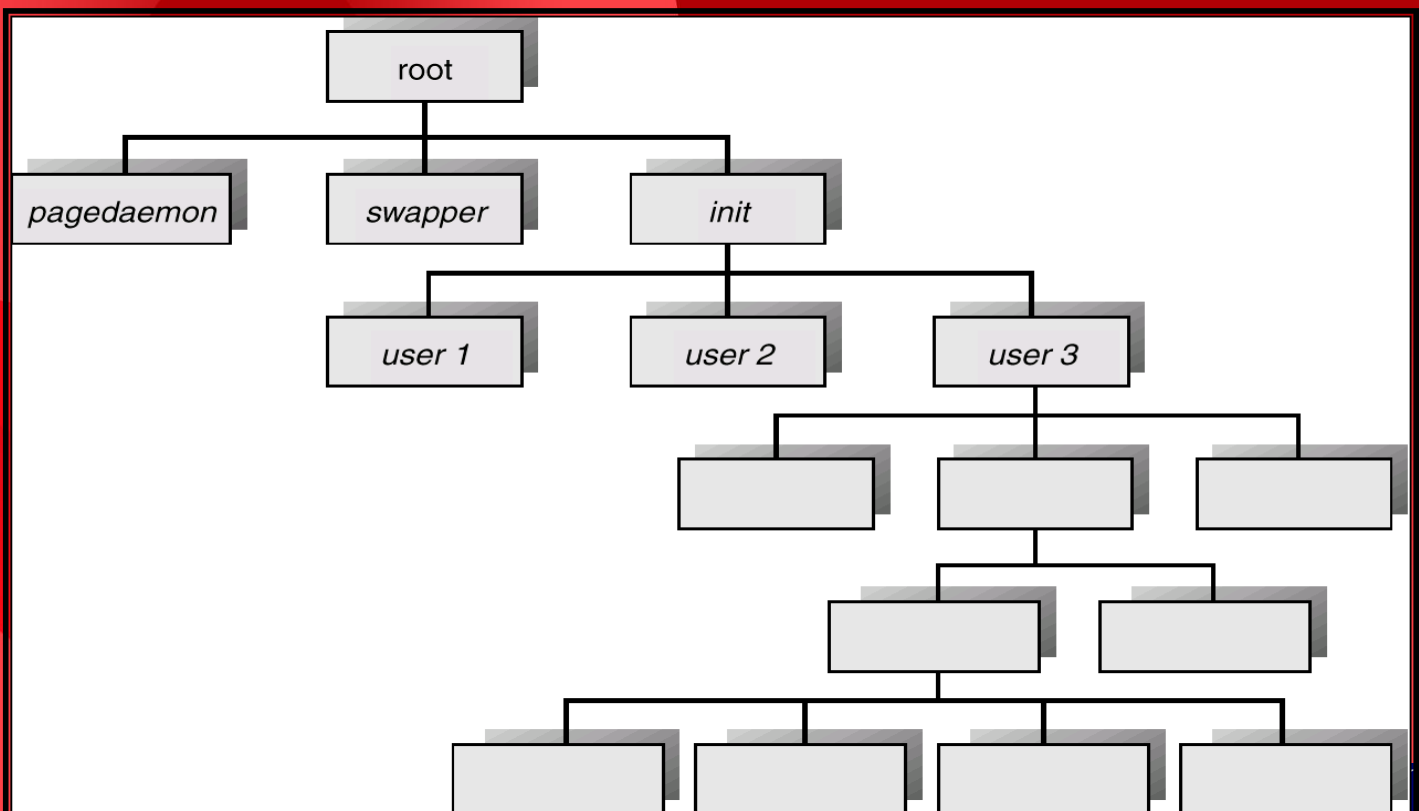
- Address space
 - Child duplicate of parent.
 - Child has a program loaded onto it.
- UNIX examples
 - `fork` system call creates a new process
 - `exec` system call used after a `fork` to replace the process' memory image with a new executable.

7 August 2021

© Copyright Virtual University of
Pakistan



Processes Tree on a UNIX System



Process Termination

- Process executes the last statement and requests the operating system to terminate it (`exit`).
 - Output data from child to parent (via `wait`).
 - Process resources are deallocated by the operating system, to be recycled later.

7 August 2021

© Copyright Virtual University of
Pakistan



Process Termination ...

- Parent may terminate execution of children processes (`abort`).
 - Child has exceeded allocated resources (main memory, execution time, etc.).
 - Parent needs to create another child but has reached its maximum children limit
 - Task performed by the child is no longer required.
 - Parent exits.
 - Operating system does not allow child to continue if its parent terminates.
 - Cascaded termination

7 August 2021

© Copyright Virtual University of
Pakistan



Process Management in UNIX/Linux

Important process-related UNIX/Linux system calls

- `fork`
- `wait`
- `exec`
- `exit`

7 August 2021

© Copyright Virtual University of
Pakistan



fork()

- When the `fork` system call is executed, a new process is created which consists of a copy of the address space of the parent.
- This mechanism allows the parent process to communicate easily with the child process.

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

7 August 2021

© Copyright Virtual University of
Pakistan



fork() ...

- The return code for `fork` is zero for the child process and the process identifier of child is returned to the parent process.
- On success, both processes continue execution at the instruction after the `fork` call.
- On failure, -1 is returned to the parent process and `errno` is set appropriately to indicate the reason of failure; no child is created

7 Aug 19, 2021

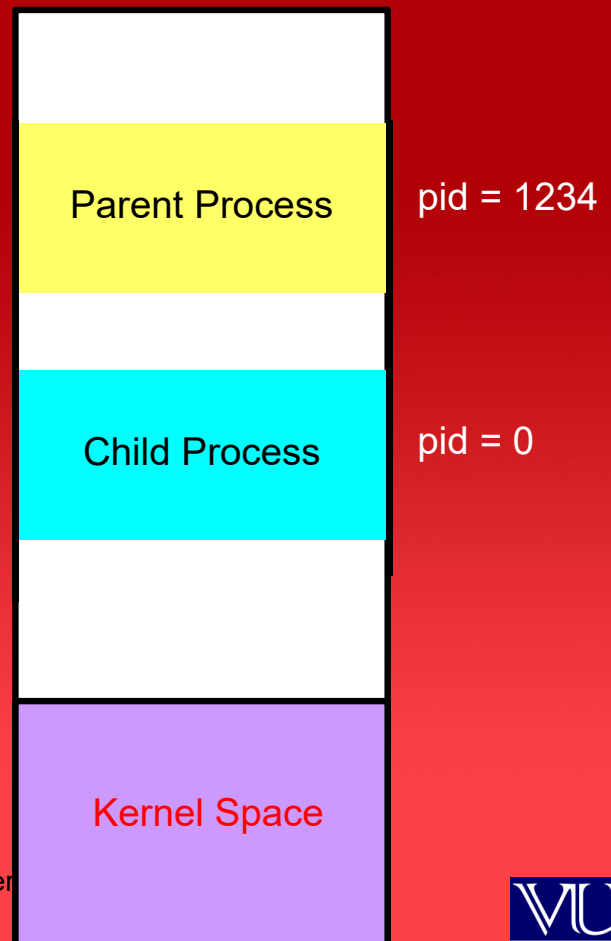
© Copyright Virtual University of
Pakistan



fork()—Sample Code

```
main()
{
    int pid;

    ...
    → pid = fork();
    → if (pid == 0) {
        /* Code for child */
        ...
    }
    else {
        /* Code for parent */
        ...
    }
    ...
}
```



fork()—Inherits from the Parent

The child process inherits the following attributes from the parent:

- Environment
- Open file descriptor table
- Signal handling settings
- Nice value
- Current working directory
- Root directory
- File mode creation mask (`umask`)
- Etc.

7 August 2021

© Copyright Virtual University of
Pakistan



fork()—Child Differs from the Parent

The child process differs from the parent process:

- Different process ID (PID)
- Different parent process ID (PPID)
- Child has its own copy of parent's file descriptors
- Etc.

7 August 2021

© Copyright Virtual University of
Pakistan



fork()—Reasons for Failure

- Maximum number of processes allowed to execute under one user has exceeded
- Maximum number of processes allowed on the system has exceeded
- Not enough swap space

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Review of previous lecture
- Process scheduling concepts
- Process creation and termination
- Process management in UNIX/Linux—
system calls: `fork`, `wait`, `exit`
- Sample codes
- Recap of the lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 7



Agenda for Today

- Review of previous lecture
- The `wait` and `exec` system calls and sample code
- Cooperating processes
- Producer-consumer problem
- Interprocess communication (IPC) and process synchronization
- Recap of the lecture

Review of Lecture 6

- Schedulers (long-, and short-, and medium-term)
- Dispatcher
- Process creation and termination
- `fork` and `exit` system calls

7 August 2021

© Copyright Virtual University of
Pakistan



`wait()`

- The `wait` system call suspends the calling process until one of its immediate children terminates, or until a child that is being traced stops because it has hit an event of interest.
- `wait` returns prematurely if a signal is received. If all children processes stopped or terminated prior to the call on `wait`, return is immediate.

7 August 2021

© Copyright Virtual University of
Pakistan



Synopsis of wait()

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *stat_loc);
```

<sys/types.h>:

/usr/include/sys/types.h

7 August 2021

© Copyright Virtual University of
Pakistan



wait() ...

- If the call is successful, the process ID of the terminating child is returned.
- If parent terminates all its children have assigned as their new parent, the **init** process. Thus the children still have a parent to collect their status and execution statistics.

7 August 2021

© Copyright Virtual University of
Pakistan



wait() ...

- **Zombie process**—a process that has terminated but whose exit status has not yet been received by its parent process or by **init**.

Sample Code—fork

```
#include <stdio.h>
void main()
{
    int pid, status;

    pid = fork();
    if(pid == -1) {
        printf("fork failed\n");
        exit(1);
    }
}
```

Sample Code—fork

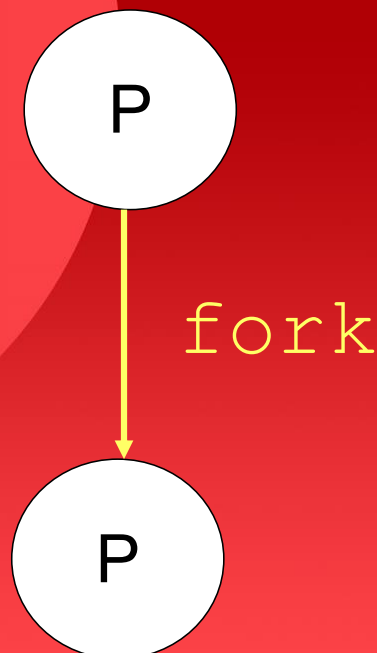
```
if(pid == 0) { /* Child */  
    printf("Child here!\n");  
    exit(0);  
}  
else { /* Parent */  
    wait(&status);  
    printf("Well done kid!\n");  
    exit(0);  
}
```

} 7 August 2021

© Copyright Virtual University of
Pakistan



Semantics of fork



7 August 2021

© Copyright Virtual University of
Pakistan



exec()

- Typically the `exec` system call is used after a `fork` system call by one of the two processes to replace the process' memory space with a new executable program.
- The new process image is constructed from an ordinary, executable file.

exec()

- There can be no return from a successful `exec` because the calling process image is overlaid by the new process image

Synopsis of exec()

```
#include <unistd.h>

int execlp (const char *file, const
            char *arg0, ..., const char *argn,
            (char *)0);
```

7 August 2021

© Copyright Virtual University of
Pakistan



Sample Code—fork and exec

```
#include <stdio.h>

void main()
{
    int pid, status;

    pid = fork();
    if(pid == -1) {
        printf("fork failed\n");
        exit(1);
    }
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Sample Code—`fork` and `exec`

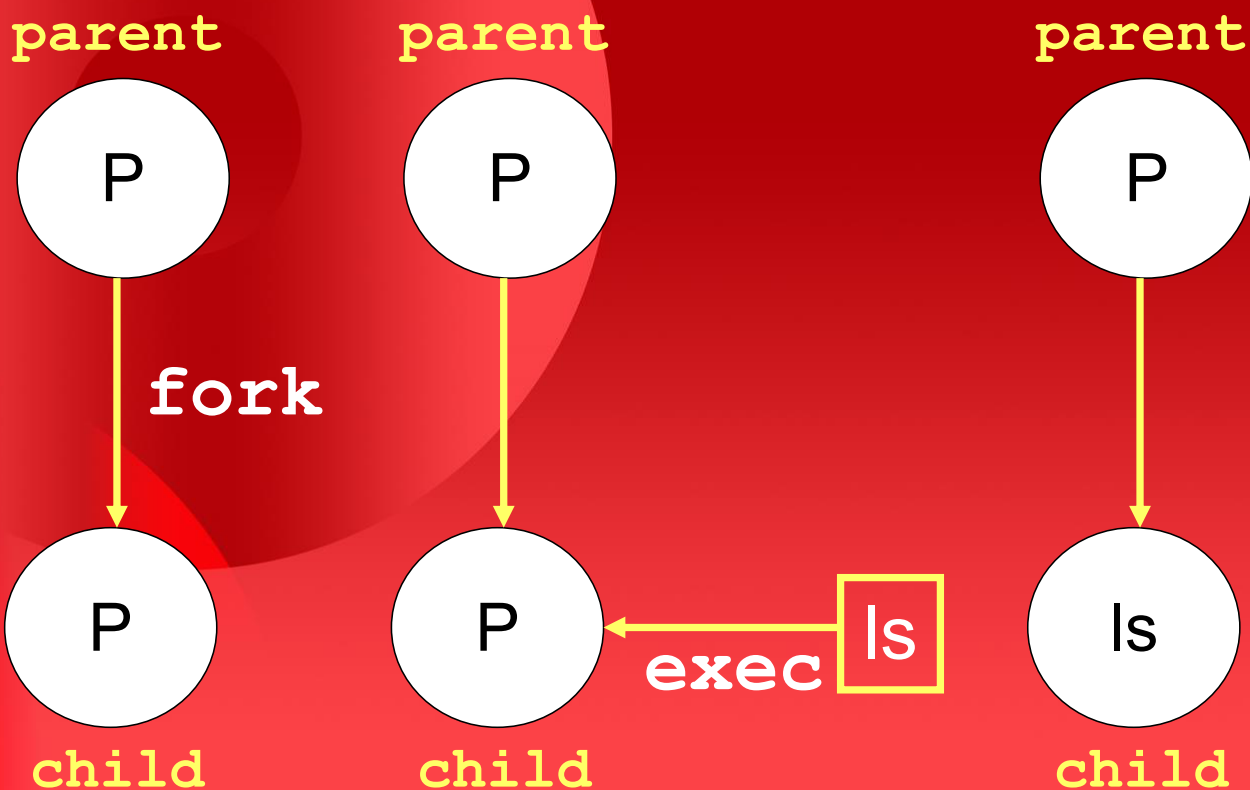
```
if(pid == 0) { /* Child */
    if (execlp("/bin/ls", "ls", NULL) < 0) {
        printf("exec failed\n");
        exit(1);
    }
}
else { /* Parent */
    wait(&status);
    printf("Well done kid!\n");
    exit(0);
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Semantics of `fork`



7 August 2021

1

© Copyright Virtual University of
Pakistan

2

3



Cooperating Processes

- *Independent* process cannot affect or be affected by the execution of another process.
- *Cooperating* process can affect or be affected by the execution of another process



7 August 2021

© Copyright Virtual University of
Pakistan

Cooperating Processes

- Advantages of process cooperation
 - Information sharing
 - Computation speed-up
 - Modularity
 - Convenience



7 August 2021

© Copyright Virtual University of
Pakistan

Producer-Consumer Problem

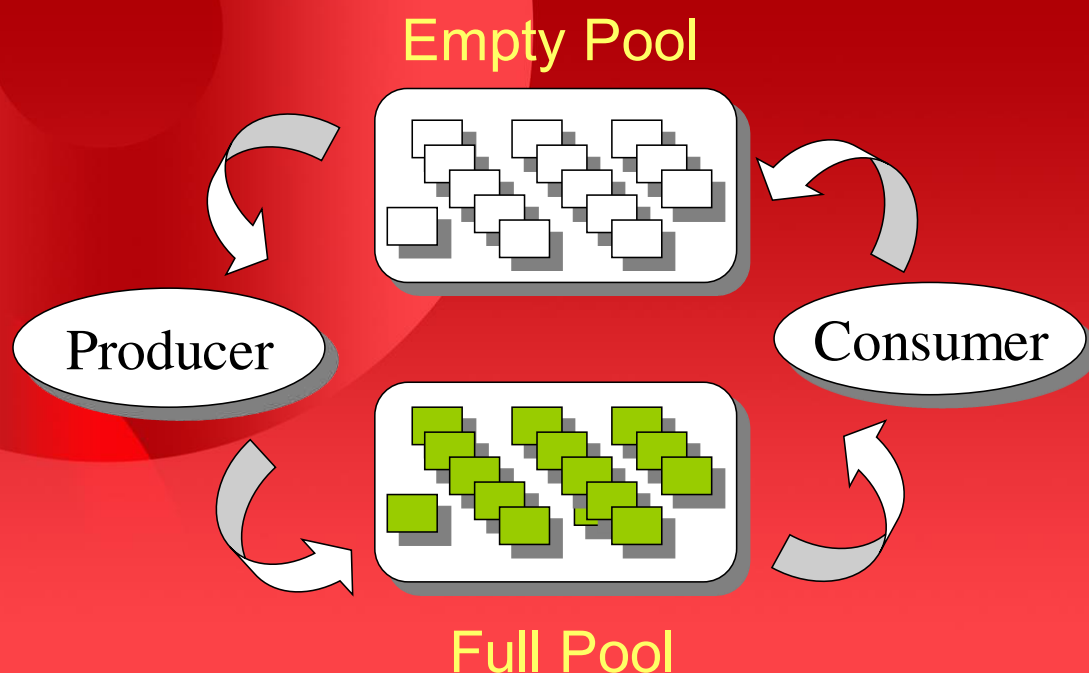
- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process.
 - *unbounded-buffer* places no practical limit on the size of the buffer
 - *bounded-buffer* assumes that there is a fixed buffer size

7 August 2021

© Copyright Virtual University of
Pakistan



Bounded-Buffer Problem



7 August 2021

© Copyright Virtual University of
Pakistan



Bounded-Buffer Solution

- Shared data

```
#define BUFFER_SIZE 10
typedef struct {
    . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- Solution is correct, but can only use **BUFFER_SIZE-1** elements

7 August 2021

© Copyright Virtual University of
Pakistan



Producer Process

```
item nextProduced;

while (1) {
    while (((in + 1) % BUFFER_SIZE) == out)
        ; /* do nothing */
    buffer[in] = nextProduced;
    in = (in + 1) % BUFFER_SIZE;
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Consumer Process

```
item nextConsumed;  
  
while (1) {  
    while (in == out)  
        ; /* do nothing */  
    nextConsumed = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Interprocess Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions.
- Message system – processes communicate with each other without resorting to shared variables.

7 August 2021

© Copyright Virtual University of
Pakistan



Interprocess Communication (IPC)

- IPC facility provides two operations:
 - **Send** (*message*) – message size fixed or variable
 - **Receive** (*message*)

7 August 2021

© Copyright Virtual University of
Pakistan



Interprocess Communication (IPC)

- If P and Q wish to communicate, they need to:
 - establish a *communication link* between them
 - exchange messages via send/receive

7 August 2021

© Copyright Virtual University of
Pakistan



Interprocess Communication (IPC)

- Implementation of communication link
 - **physical** (e.g., shared memory, hardware bus)
 - **logical** (e.g., logical properties)

7 August 2021

© Copyright Virtual University of
Pakistan



Implementation Questions

- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?

7 August 2021

© Copyright Virtual University of
Pakistan



Implementation Questions

- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?

7 August 2021

© Copyright Virtual University of
Pakistan



Direct Communication

- Processes must name each other explicitly:
 - **send** (P , *message*) – send a message to process P
 - **Receive** (Q , *message*) – receive a message from process Q

7 August 2021

© Copyright Virtual University of
Pakistan



Direct Communication

- Properties of communication link
 - Links are established automatically.
 - A link is associated with exactly one pair of communicating processes.
 - Between each pair there exists exactly one link.
 - The link may be unidirectional, but is usually bi-directional.

7 August 2021

© Copyright Virtual University of
Pakistan

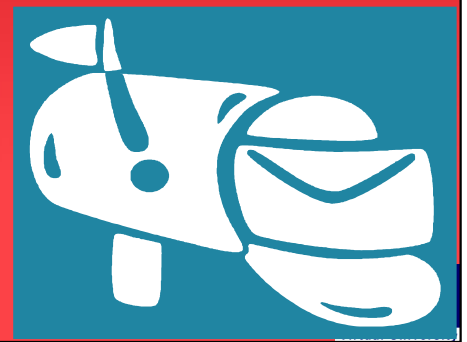


Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports).
 - Each mailbox has a unique id.
 - Processes can communicate only if they share a mailbox.

7 August 2021

© Copyright Virtual University of
Pakistan



Indirect Communication ...

- Properties of communication link
 - Link established only if processes share a common mailbox
 - A link may be associated with many processes.
 - Each pair of processes may share several communication links.
 - Link may be unidirectional or bi-directional.

7 August 2021

© Copyright Virtual University of
Pakistan



Indirect Communication ...

- Operations
 - create a new mailbox
 - send and receive messages through mailbox
 - destroy a mailbox
- Primitives are defined as:
send (*A, message*)
receive (*A, message*)

7 August 2021

© Copyright Virtual University of
Pakistan



Indirect Communication ...

Mailbox sharing

- P_1 , P_2 , and P_3 share mailbox A.
- P_1 sends; P_2 and P_3 receive.
- Who gets the message?

7 August 2021

© Copyright Virtual University of
Pakistan



Indirect Communication ...

Solutions

- Allow a link to be associated with at most two processes.
- Allow only one process at a time to execute a receive operation.
- Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

7 August 2021

© Copyright Virtual University of
Pakistan



Synchronization

- Message passing may be either blocking or non-blocking.
- **Blocking** is considered **synchronous**
- **Non-blocking** is considered **asynchronous**
- **send** and **receive** primitives may be either blocking or non-blocking.

7 August 2021

© Copyright Virtual University of
Pakistan



Buffering

- Queue of messages attached to the link; implemented in one of three ways.

Zero capacity – No messages

Sender must wait for receiver

Bounded capacity – n messages

Sender must wait if link full.

Unbounded capacity – infinite length

Sender never waits.

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Review of previous lecture
- The `wait` and `exec` system call and sample code
- Cooperating processes
- Producer-consumer problem
- Sample codes
- Interprocess communication (IPC) and process synchronization
- Recap of the lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 8



Agenda for Today

- Review of previous lecture
- Interprocess communication (IPC) and process synchronization
- UNIX/Linux IPC tools (pipe, named pipe—FIFO, socket, message queue, shared memory)
- Use of pipe
- Recap of the lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 7

- The `wait` and `exec` system calls and sample code
- Cooperating processes
- Producer-consumer problem

7 August 2021

© Copyright Virtual University of
Pakistan



Interprocess Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions.
- Message system – processes communicate with each other without resorting to shared variables.

7 August 2021

© Copyright Virtual University of
Pakistan



Interprocess Communication (IPC)

- IPC facility provides two operations:
 - **Send** (*message*) – message size fixed or variable
 - **Receive** (*message*)

7 August 2021

© Copyright Virtual University of
Pakistan



Interprocess Communication (IPC)

- If P and Q wish to communicate, they need to:
 - establish a *communication link* between them
 - exchange messages via send/receive

7 August 2021

© Copyright Virtual University of
Pakistan



Interprocess Communication (IPC)

- Implementation of communication link
 - **physical** (e.g., shared memory, hardware bus)
 - **logical** (e.g., logical properties)

7 August 2021

© Copyright Virtual University of
Pakistan



Implementation Questions

- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?

7 August 2021

© Copyright Virtual University of
Pakistan



Implementation Questions

- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?

7 August 2021

© Copyright Virtual University of
Pakistan



Direct Communication

- Processes must name each other explicitly:
 - **send** (P , *message*) – send a message to process P
 - **Receive** (Q , *message*) – receive a message from process Q

7 August 2021

© Copyright Virtual University of
Pakistan



Direct Communication

- Properties of communication link
 - Links are established automatically.
 - A link is associated with exactly one pair of communicating processes.
 - Between each pair there exists exactly one link.
 - The link may be unidirectional, but is usually bi-directional.

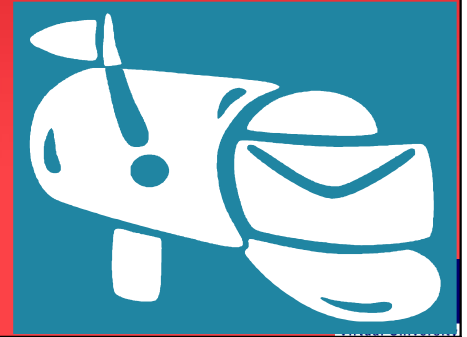
7 August 2021

© Copyright Virtual University of
Pakistan



Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports).
- Each mailbox has a unique id.
- Processes can communicate only if they share a mailbox.



7 August 2021

© Copyright Virtual University of
Pakistan

Indirect Communication ...

- Properties of communication link
 - Link established only if processes share a common mailbox
 - A link may be associated with many processes.
 - Each pair of processes may share several communication links.
 - Link may be unidirectional or bi-directional.

7 August 2021

© Copyright Virtual University of
Pakistan

Indirect Communication ...

- Operations
 - create a new mailbox
 - send and receive messages through mailbox
 - destroy a mailbox
- Primitives are defined as:
send (A , $message$)
receive (A , $message$)

7 August 2021

© Copyright Virtual University of
Pakistan



Indirect Communication ...

Mailbox sharing

- P_1 , P_2 , and P_3 share mailbox A .
- P_1 , sends; P_2 and P_3 receive.
- Who gets the message?

7 August 2021

© Copyright Virtual University of
Pakistan



Indirect Communication ...

Solutions

- Allow a link to be associated with at most two processes.
- Allow only one process at a time to execute a receive operation.
- Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

7 August 2021

© Copyright Virtual University of
Pakistan



Synchronization

- Message passing may be either blocking or non-blocking.
- **Blocking** is considered **synchronous**
- **Non-blocking** is considered **asynchronous**
- **send** and **receive** primitives may be either blocking or non-blocking.

7 August 2021

© Copyright Virtual University of
Pakistan



Buffering

- Queue of messages attached to the link; implemented in one of three ways.

Zero capacity – No messages

Sender must wait for receiver

Bounded capacity – n messages

Sender must wait if link full.

Unbounded capacity – infinite length

Sender never waits.

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX/Linux IPC Tools

- Pipe
- Named pipe (FIFO)
- BSD Socket
- TLI
- Message queue
- Shared memory
- Etc.

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX/Linux Pipe

- Important system calls
`pipe`, `read`, `write`, `close`
- `pipe`: Create a pipe for IPC
- `read`: Read from a pipe
- `write`: Write data to a pipe
- `close`: Close/destroy a pipe

7 August 2021

© Copyright Virtual University of
Pakistan



File Descriptors in UNIX/Linux

- An integer returned by `open()` system call
- Used as an index in the per process file descriptor table (PPFDT)
- Used in `read`, `write`, and `close` calls

7 August 2021

© Copyright Virtual University of
Pakistan



File Descriptors in UNIX/Linux

- Size of PPFDT is equal to the number of files that a process can open simultaneously (`OPEN_MAX` in Linux—see `<linux/limits.h>`)
- Used as an index in the per process file descriptor table (PPFDT)
- Used in `read`, `write`, and `close` calls

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX/Linux Pipe

- Important characteristics of a pipe
 - Stream of bytes
 - Used as half-duplex channel
 - Bounded buffer
 - Maximum data written is `PIPE_BUF` (defined in `<sys/param.h>` under UNIX and in `<linux/param.h>` in Linux)—5120 and 4096, respectively

7 August 2021

© Copyright Virtual University of
Pakistan



Synopsis of pipe System Call

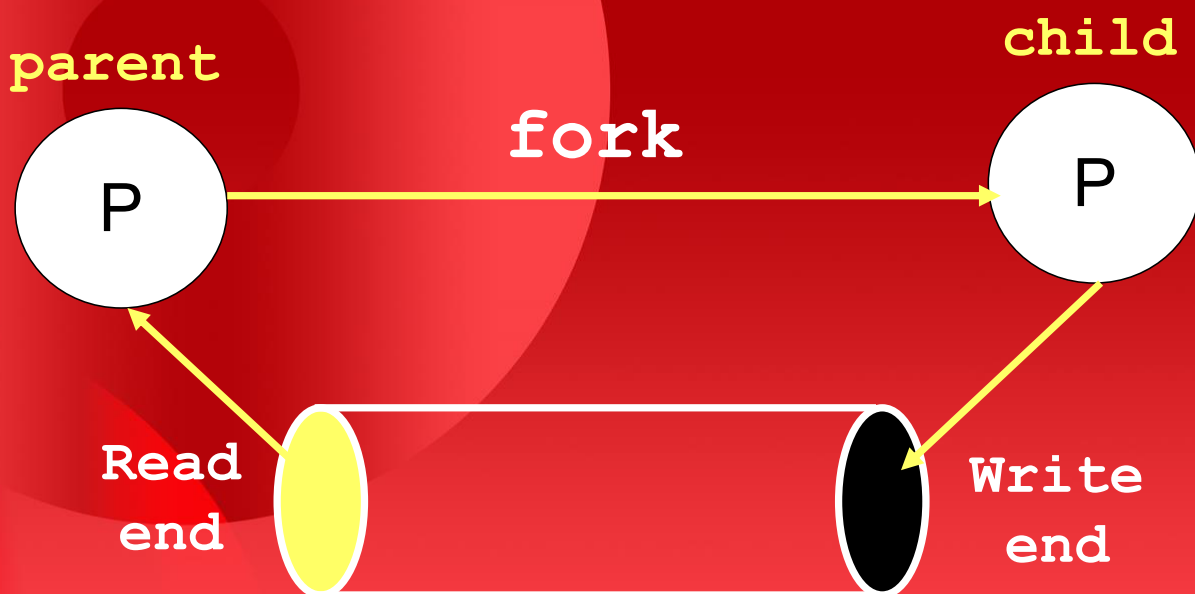
```
#include <unistd.h>
int pipe (int filedes[2]);
```

7 August 2021

© Copyright Virtual University of
Pakistan



Example



7 August 2021

© Copyright Virtual University of
Pakistan



Sample Code

```
/* Parent creates pipe, forks a child, child writes into
   pipe, and parent reads from pipe */
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
main()
{
    int pipefd[2], pid, n, rc, nr, status;
    char *testString = "Hello, world!\n", buf[1024];
```

7 August 2021

© Copyright Virtual University of
Pakistan



Sample Code

```
rc = pipe (pipefd);
if (rc < 0) {
    perror("pipe");
    exit(1);
}
pid = fork ();
if (pid < 0) {
    perror("fork");
    exit(1);
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Sample Code

```
if (pid == 0) { /* Child's Code */  
    close(pipefd[0]);  
    write(pipefd[1], testString, strlen(testString));  
    close(pipefd[1]);  
    exit(0);  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Sample Code

```
/* Parent's Code */  
close(pipefd[1]);  
n = strlen(testString);  
nr = read(pipefd[0], buf, n);  
rc = write(1, buf, nr);  
wait(&status);  
printf("Good work child!\n");  
return(0);
```

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Review of previous lecture
- Interprocess communication (IPC) and process synchronization
- UNIX/Linux IPC tools (pipe, FIFO, socket, message queue, shared memory, etc.)
- Use of UNIX pipe
- Recap of the lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 9



Agenda for Today

- Review of previous lecture
- UNIX/Linux IPC tools and associated system calls
- UNIX/Linux standard files and kernel's mechanism for file access
- Use of pipe in a program and at the command line
- Recap of the lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 8

- Interprocess communication (IPC)
 - Establish link and use `send/recv`
 - Issues about links: establishing links, link capacity, links per pair of processes, processes per link, message size, uni- or bi-directional
- Direct communication
- Indirect communication

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 8

- Process synchronization
- Buffering capacity of a link
- IPC in UNIX and Linux
 - Pipe, FIFO, Socket, etc.
 - Per process file descriptor table
 - `pipe`, `read`, `write`, `close` system calls

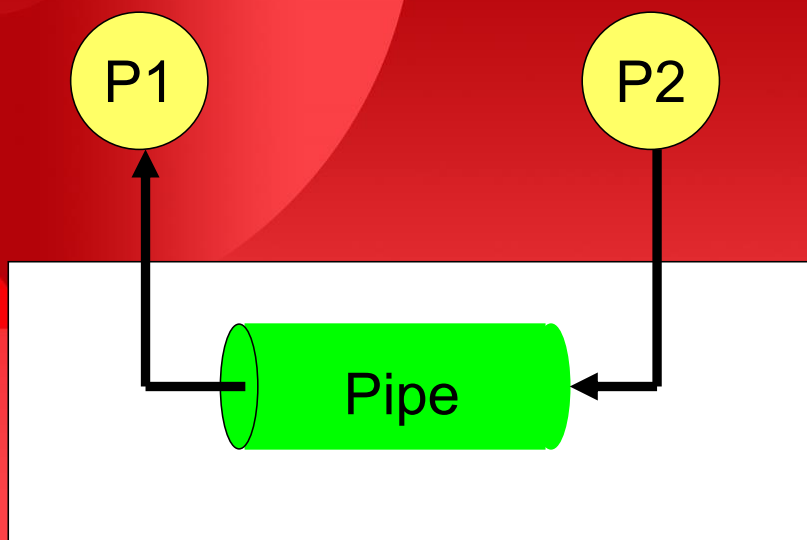
7 August 2021

© Copyright Virtual University of
Pakistan



UNIX/Linux IPC Tools

- **Pipe:** For communication between related processes on a system



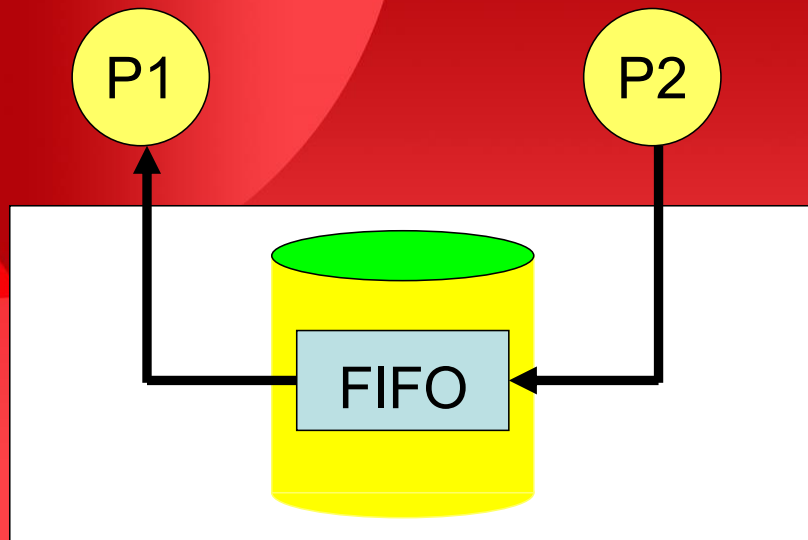
7 August 2021

© Copyright Virtual University of
Pakistan



UNIX/Linux IPC Tools

- **Named pipe (FIFO):** For communication between unrelated processes on a system



7 August 2021

© Copyright Virtual University of
Pakistan



UNIX/Linux IPC Tools

- **Socket:** For communication between unrelated processes on the same or different systems



7 August 2021

© Copyright Virtual University of
Pakistan

Computer 2



UNIX/Linux Pipe

- Important system calls

open, read, write, close, pipe

- open: Open or create a file
- read: Read from a pipe
- write: Write data to a pipe
- close: Close/destroy a pipe
- pipe: Create a pipe for IPC

7 August 2021

© Copyright Virtual University of
Pakistan



open System Call

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
int open (const char *path, int oflag,
          /* mode_t mode */ ...);
```

- 'oflag' specifies purpose of opening the file and 'mode' specifies permission on the file if it is to be created.

7 August 2021

© Copyright Virtual University of
Pakistan



open **System Call**

- Returns a file descriptor on success and -1 on failure
- Can specify that read and write will be **blocking** or **non-blocking**
- 'oflag' value is constructed by ORing various flags: O_RDONLY, O_WRONLY, O_RDWR, O_NDELAY (or O_NONBLOCK), O_APPEND, O_CREAT, etc.

7 August 2021

© Copyright Virtual University of
Pakistan



open **System Call**

- Call fails
 - Non-existent file
 - Operation specified is not allowed due to file permissions
 - Search not allowed on a component of pathname
 - User's disk quota on the file system has been exhausted

7 August 2021

© Copyright Virtual University of
Pakistan



open **System Call**

- Call fails
 - No write permission on the directory in which the file is being created
 - Signal was caught during `open`
 - Process has reached the limit of maximum open files
 - System limit reached on maximum number of simultaneous open files

7 August 2021

© Copyright Virtual University of
Pakistan



read **System Call**

```
#include <sys/types.h>
```

```
#include <sys/uio.h>
```

```
#include <unistd.h>
```

```
ssize_t read(int fildes, void *buf, size_t nbyte);
```

- Returns number of bytes read or -1
- Call fails and `errno` set accordingly
 - Invalid 'fildes', 'buf', or 'nbyte'
 - Signal caught during read

7 August 2021

© Copyright Virtual University of
Pakistan



write **System Call**

```
#include <sys/types.h>
```

```
#include <unistd.h>
```

```
ssize_t write (int fildes, const void *buf,  
              size_t nbyte);
```

- Returns the number of bytes written or -1

7 August 2021

© Copyright Virtual University of
Pakistan



write **System Call**

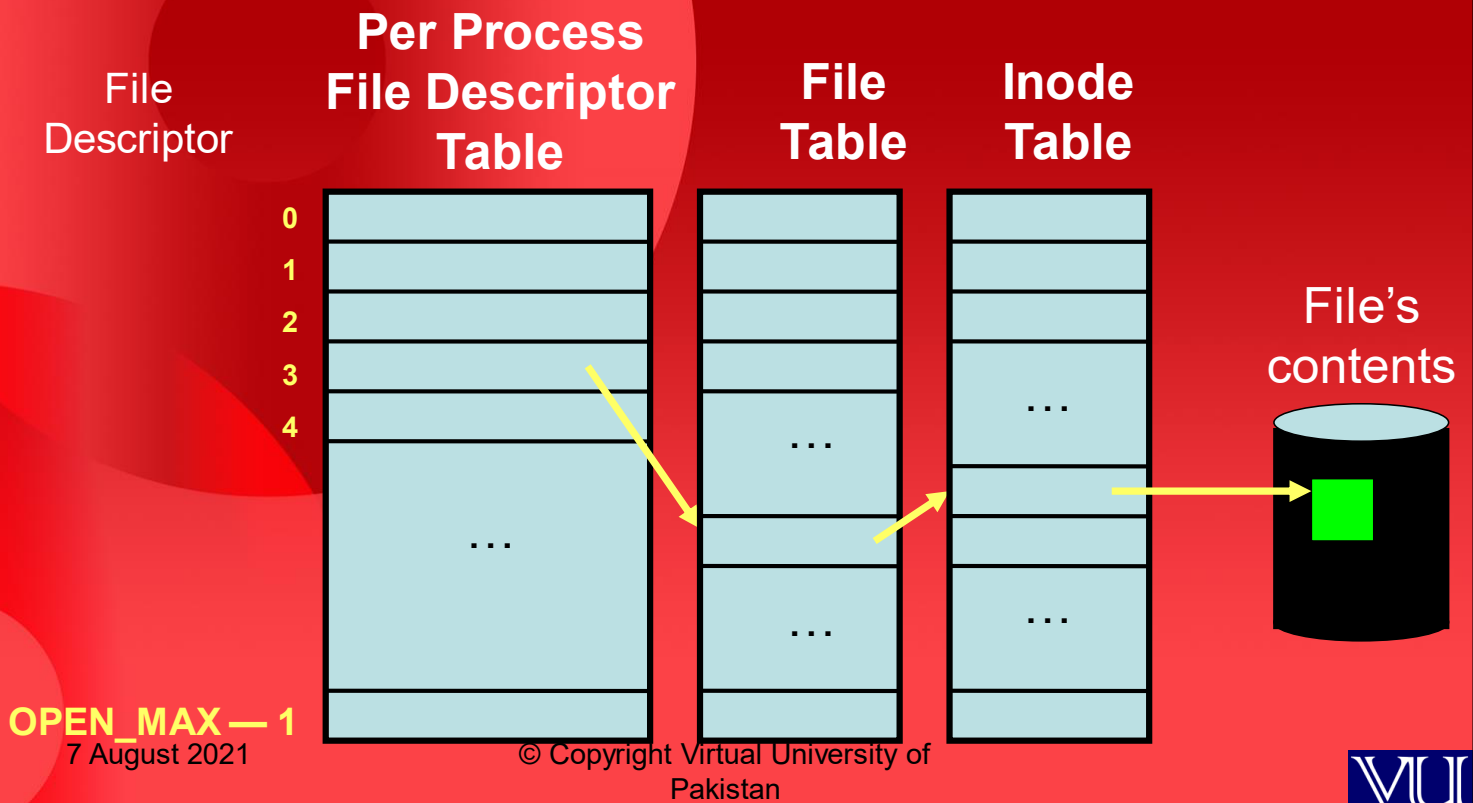
- Call fails
 - Invalid argument
 - File size limit for process or for system would exceed
 - Disk is full

7 August 2021

© Copyright Virtual University of
Pakistan



File Descriptor to File Contents



Standard Descriptors in UNIX/Linux

- Three files are automatically opened for every process for the process to read its input from and send its output and error messages to. These files are called standard files: **standard input**, **standard output**, and **standard error**.

Standard Descriptors in UNIX/Linux

- By default, standard files are attached to the terminal on which a process runs
- Descriptors for standard files are known as **standard file descriptors**.
 - Standard input: 0 (keyboard)
 - Standard output: 1 (display screen)
 - Standard error: 2 (display screen)

7 August 2021

© Copyright Virtual University of
Pakistan



pipe() Call

- `int pipe (int pippedes[2]);`
- Call fails
 - At least two slots not empty in the PPFDT—too many files or pipe open in the process
 - Buffer space not available in the kernel
 - File table is full

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX/Linux Pipe

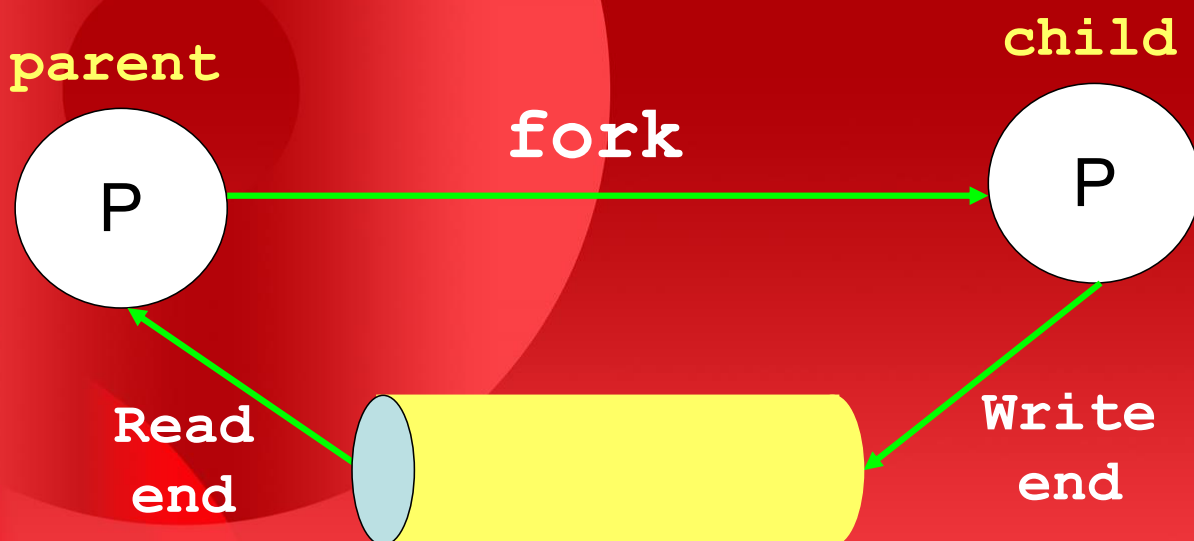
- Important characteristics of a pipe
 - Used for communication between related processes
 - Used as half-duplex channel
 - Bounded buffer
 - Maximum data written is PIPE_BUF (defined in `<sys/param.h>` in UNIX and in `<linux/param.h>` in Linux)—5120 and 4096, respectively

7 August 2021

© Copyright Virtual University of
Pakistan



Example



7 August 2021

© Copyright Virtual University of
Pakistan



Sample Code

```
/* Parent creates pipe, forks a child, child writes into
   pipe, and parent reads from pipe */
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
main()
{
    int pipefd[2], pid, n, rc, nr, status;
    char *testString = "Hello, world!\n", buf[1024];
```

7 August 2021

© Copyright Virtual University of
Pakistan



Sample Code

```
rc = pipe (pipefd);
if (rc < 0) {
    perror("pipe");
    exit(1);
}
pid = fork ();
if (pid < 0) {
    perror("fork");
    exit(1);
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Sample Code

```
if (pid == 0) { /* Child's Code */  
    close(pipefd[0]);  
    write(pipefd[1], testString, strlen(testString));  
    close(pipefd[1]);  
    exit(0);  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Sample Code

```
/* Parent's Code */  
close(pipefd[1]);  
n = strlen(testString);  
nr = read(pipefd[0], buf, n);  
rc = write(1, buf, nr);  
wait(&status);  
printf("Good work child!\n");  
return(0);
```

7 August 2021

© Copyright Virtual University of
Pakistan



Command Line Use of Pipes

- Connect standard output of a command to standard input of another
- Use the pipe operator, |
- Syntax:

cmd1 | cmd2 | ... | cmdN

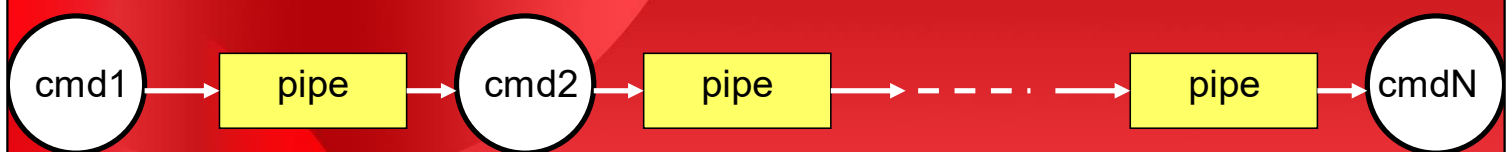
7 August 2021

© Copyright Virtual University of
Pakistan



Command Line Use of Pipes

cmd1 | cmd2 | ... | cmdN



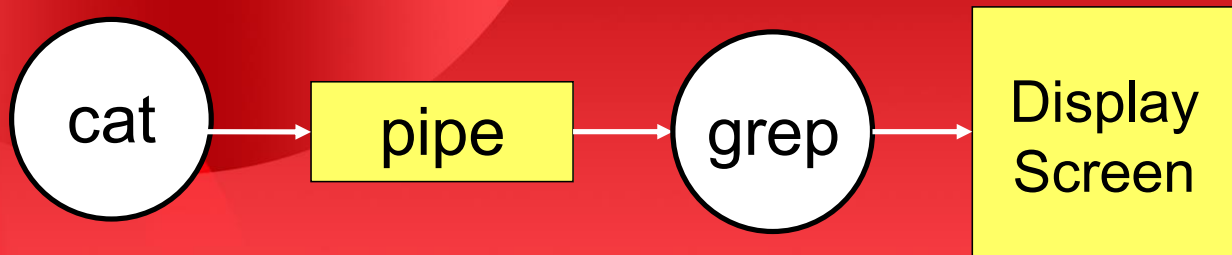
7 August 2021

© Copyright Virtual University of
Pakistan



Command Line Use of Pipes

```
cat /etc/passwd | grep zaheer
```



7 August 2021

© Copyright Virtual University of
Pakistan



Without Using a Pipe

```
cat /etc/passwd | grep zaheer
```

```
$ cat /etc/passwd > temp1
```

```
$ grep "zaheer" temp1
```

```
$ rm temp1
```

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Review of previous lecture
- UNIX/Linux IPC tools and associated system calls
- UNIX/Linux standard files and kernel's mechanism for file access
- Use of pipe in a program and at the command line
- Recap of the lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 10



Agenda for Today

- Review of previous lecture
- Input, output, and error redirection in UNIX/Linux
- FIFOs in UNIX/Linux
- Use of FIFOs in a program and at the command line
- Recap of lecture

Review of Lecture 9

- UNIX/Linux IPC tools and associated system calls
- UNIX/Linux standard files and kernel's mechanism for file access
- Use of pipe in a program and at the command line

Input, Output, Error Redirection

You can use the Linux **redirection** features to detach the default files from `stdin`, `stdout`, and `stderr` and attach other files with them.

Input Redirection

Input Redirection:

`command < input-file`

`command 0< input-file`

Purpose: Detach keyboard from `stdin` and attach 'input-file' to it, i.e., 'command' reads input from 'input-file' and not keyboard

Input Redirection

```
$ cat < Phones
```

```
[ contents of Phones ]
```

```
$ grep "Nauman" < Phones
```

```
[ output of grep ]
```

```
$
```

Output Redirection

Output Redirection:

```
command > output-file
```

```
command 1> output-file
```

Purpose: Detach the display screen from stdout and attach 'output-file' to it, i.e., 'command' sends output to 'output-file' and not the display screen

Output Redirection

```
$ cat > Phones  
[ your input ]  
<Ctrl-D>  
$ grep "Ali" Phones > Ali.phones  
[ output of grep ]  
$ find ~ -name foo -print > foo.log  
[ error messages ]  
$
```

Error Redirection

Error Redirection :

command 2> error-file

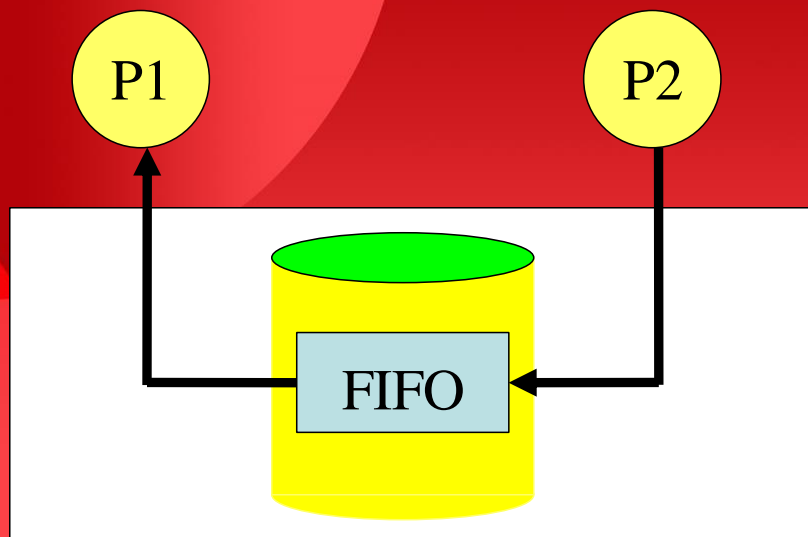
Purpose: Detach the display screen from stderr and attach 'error-file' to it, i.e., error messages are sent to 'error-file' and not the display screen

Error Redirection

```
$ find ~ -name foo -print 2> errors
[ output of the find command ]
$ ls -l foo 2> error.log
[ output of the find command ]
$ cat error.log
ls: foo: No such file or directory
$ find / -name ls -print 2> /dev/null
/bin/ls
$
```

UNIX/Linux FIFOs

- IPC for communication between related or unrelated processes on a computer



UNIX/Linux FIFOs

- A file type in UNIX
- Created with `mknod()` or `mkfifo()` system call or by `mkfifo` command

UNIX/Linux FIFOs

- Unlike a pipe, a FIFO must be opened before using it for communication
- A write to a FIFO that no process has opened for reading results in a `SIGPIPE` signal

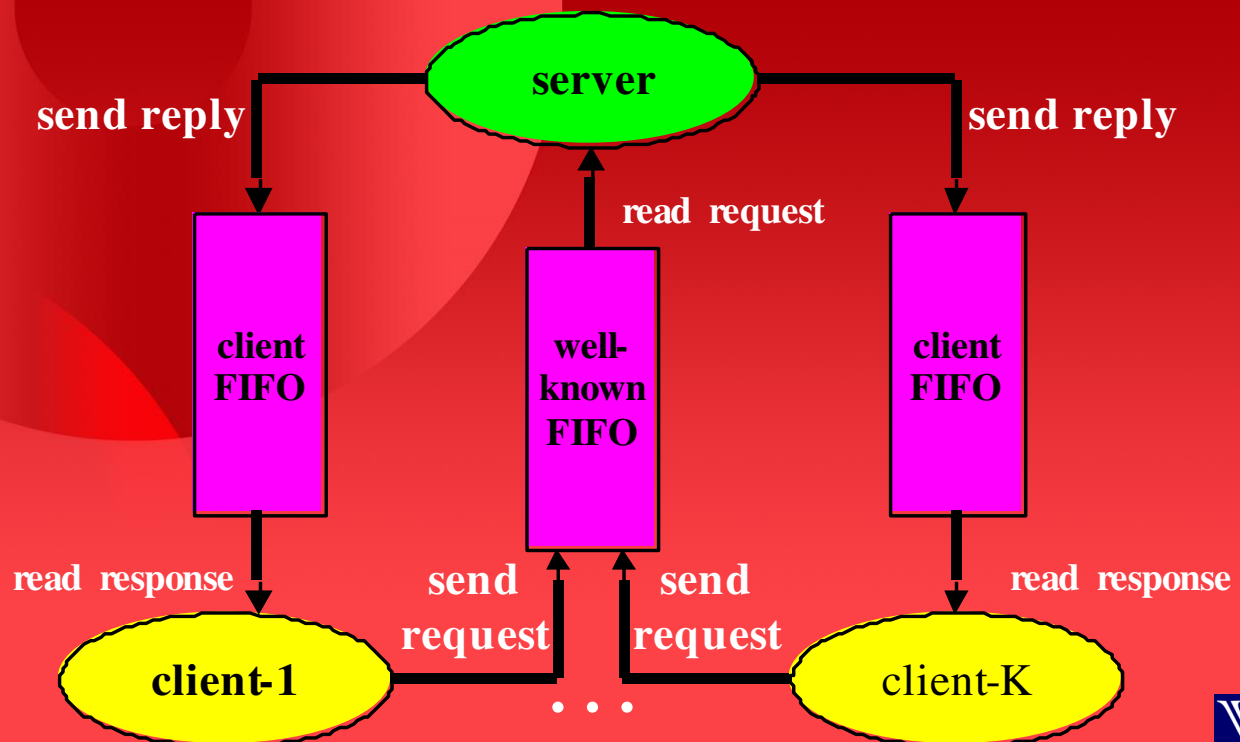
UNIX/Linux FIFOs

- When the last process to write to a FIFO closes it, an EOF is sent to the reader
- Multiple processes can write to a FIFO → atomic writes to prevent interleaving of multiple writes

UNIX/Linux FIFOs

- Two common uses of FIFOs
 - In client-server applications, FIFOs are used to pass data between a server process and client processes
 - Used by shell commands to pass data from one shell pipeline to another, without creating temporary files

Client-Server Communication with FIFOs

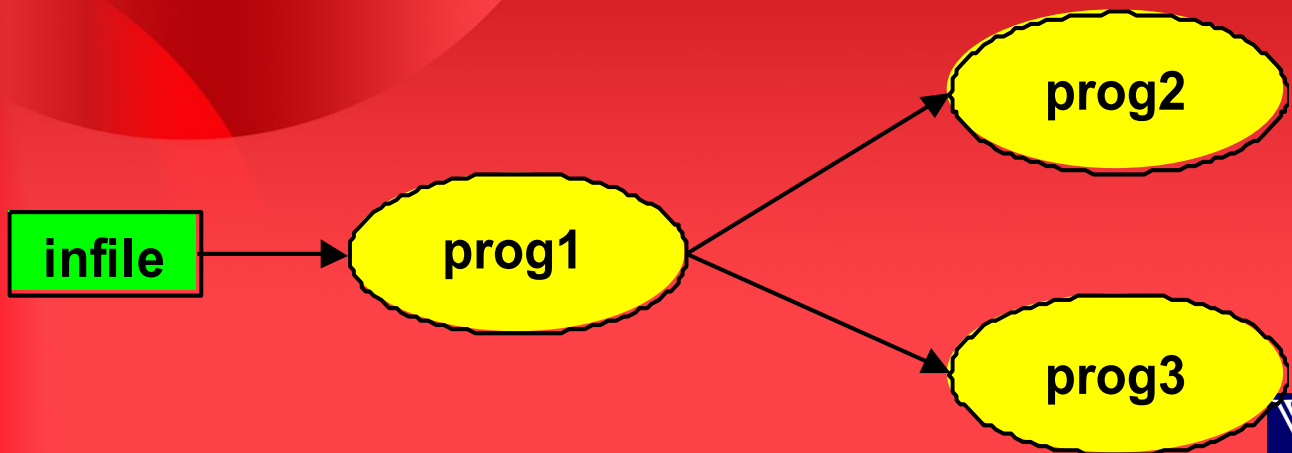


Creating FIFOs

- `mknod` system call
 - Designed to create special (device) files
- `mkfifo` Command
- `mkfifo` C library call
 - Invokes `mknod` system call

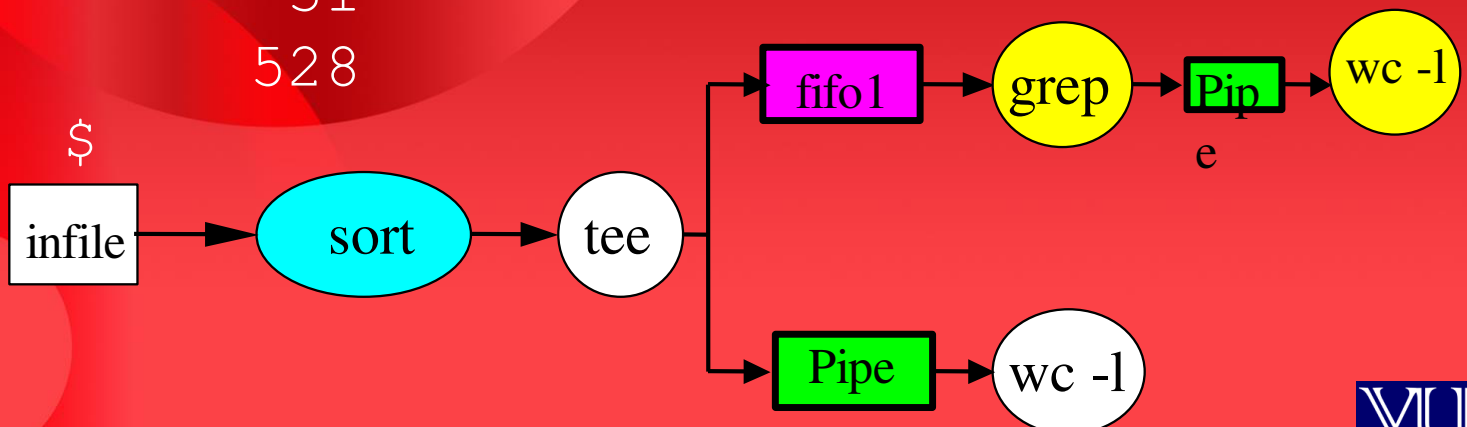
Command Line Use of FIFOs

```
$ mkfifo fifo1
$ prog3 < fifo1 &
$ prog1 < infile | tee fifo1 | prog2
[ Output ]
$
```



Command Line Use of FIFOs

```
$ man ls > ls.dat
$ cat < fifo1 | grep ls | wc -l &
[1] 21108
$ sort < ls.dat | tee fifo1 | wc -l
31
528
$
```



Recap of Lecture

- Review of previous lecture
- Input, output, and error redirection in UNIX/Linux
- FIFOs in UNIX/Linux
- Use of FIFOs at the command line
- Recap of lecture

Operating Systems

Lecture 11

Agenda for Today

- Review of previous lecture
- Use of FIFOs in a program
- Example code
- Process management commands in UNIX/Linux
- Recap of lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 10

- Input, output, and error redirection
- Use of FIFOs at the command line
- UNIX/Linux commands: `cat`, `find`, `grep`, `mkfifo`, `tee`, `wc`

7 August 2021

© Copyright Virtual University of
Pakistan



mknod **System Call**

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
int mknod (const char *path, mode_t mode,  
           dev_t dev);
```

- 'mode' should be permission mode OR-ed with S_IFIFO
- 'dev' is set to 0 for a FIFO

7 August 2021

© Copyright Virtual University of
Pakistan



mknod **System Call**

- mknod fails:
 - File with the given name exists
 - Pathname too long
 - A component in the pathname not searchable, non-existent, or non-directory
 - Destination directory is read-only
 - Not enough memory space available
 - Signal caught during mknod

7 August 2021

© Copyright Virtual University of
Pakistan



mkfifo **Library Call**

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
int mkfifo(const char *path, mode_t mode);
```

- 'mode' sets permission on FIFO
- Calls mknod

7 August 2021

© Copyright Virtual University of
Pakistan



mkfifo **Library Call**

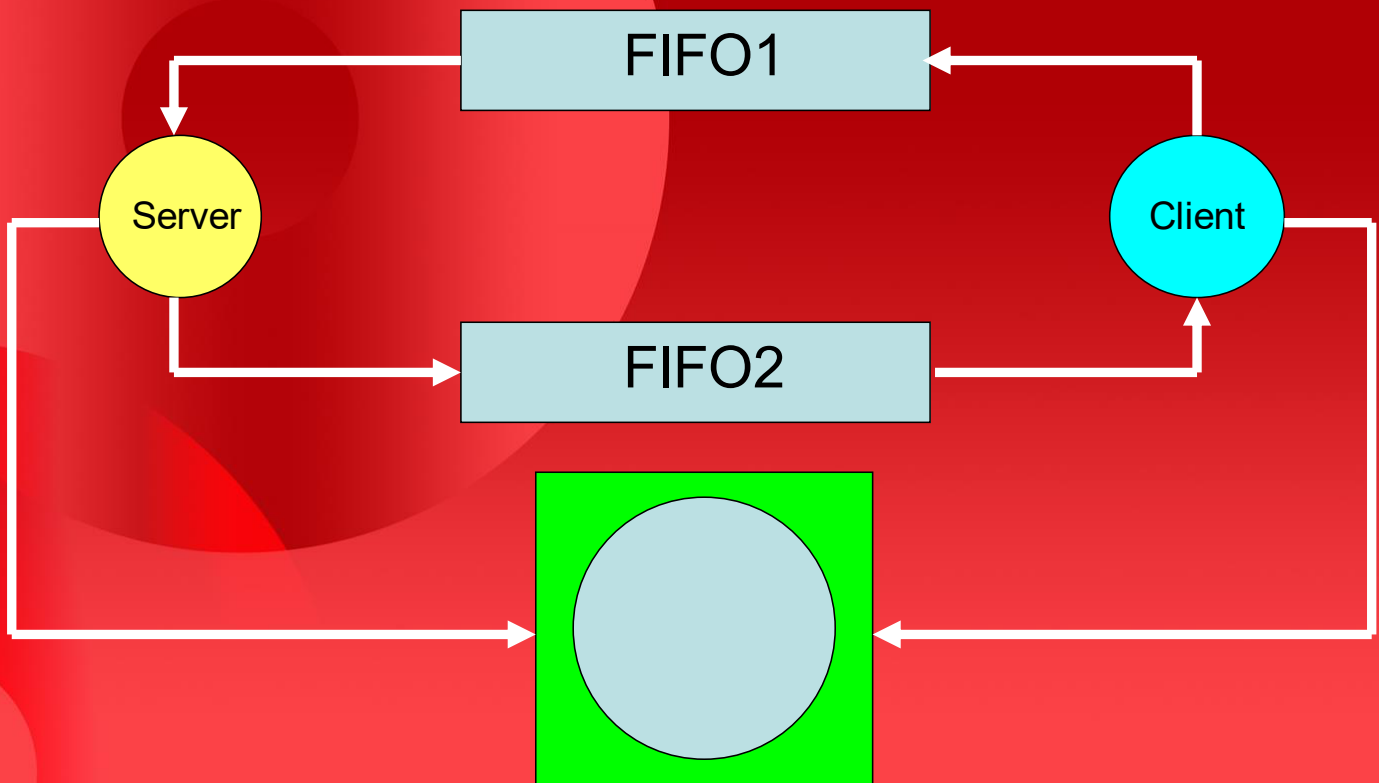
- mkfifo fails:
 - File with the given name exists
 - Pathname too long
 - A component in the pathname not searchable, non-existent, or non-directory
 - Destination directory is read-only
 - Not enough memory space available

7 August 2021

© Copyright Virtual University of
Pakistan



Example



7 August 2021

© Copyright Virtual University of
Pakistan



Client-Server Code

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/errno.h>

extern int      errno;

#define FIFO1    "/tmp/fifo.1"
#define FIFO2    "/tmp/fifo.2"
#define PERMS    0666

#define MESSAGE1    "Hello, world!\n"
#define MESSAGE2    "Hello, class!\n"
```

7 August 2021

© Copyright Virtual University of
Pakistan



Server Code

```
#include "fifo.h"

main()
{
    char buff[BUFSIZ];
    int readfd, writefd;
    int n, size;

    if ((mknod (FIFO1, S_IFIFO | PERMS, 0) < 0)
        && (errno != EEXIST)) {
        perror ("mknod FIFO1");
        exit (1);
    }
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Server Code

```
if (mkfifo(FIFO2, PERMS) < 0) {
    unlink (FIFO1);
    perror("mknod FIFO2");
    exit (1);
}

if ((readfd = open(FIFO1, 0)) < 0) {
    perror ("open FIFO1");
    exit (1);
}

if ((writefd = open(FIFO2, 1)) < 0) {
    perror ("open FIFO2");
    exit (1);
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Server Code

```
size = strlen(MESSAGE1) + 1;
if ((n = read(readfd, buff, size)) < 0) {
    perror ("server read"); exit (1);
}
if (write (1, buff, n) < n) {
    perror("server write1"); exit (1);
}
size = strlen(MESSAGE2) + 1;
if (write (writefd, MESSAGE2, size) != size) {
    perror ("server write2"); exit (1);
}

close (readfd); close (writefd);
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Client Code

```
#include "fifo.h"
main()
{
    char buff[BUFSIZ];
    int readfd, writefd, n, size;

    if ((writefd = open(FIFO1, 1)) < 0) {
        perror ("client open FIFO1"); exit (1);
    }

    if ((readfd = open(FIFO2, 0)) < 0) {
        perror ("client open FIFO2"); exit (1);
    }
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Client Code

```
size = strlen(MESSAGE1) + 1;
if (write(writefd, MESSAGE1, size) != size) {
    perror ("client write1"); exit (1);
}
if ((n = read(readfd, buff, size)) < 0) {
    perror ("client read"); exit (1);
}
else
    if (write(1, buff, n) != n) {
        perror ("client write2"); exit (1);
    }
close(readfd); close(writefd);
```

7 August 2021 © Copyright Virtual University of
Pakistan



Client Code

```
/* Remove FIFOs now that we are done using them */
if (unlink (FIFO1) < 0) {
    perror("client unlink FIFO1");
    exit (1);
}
if (unlink (FIFO2) < 0) {
    perror("client unlink FIFO2");
    exit (1);
}
exit (0);
}
```

7 August 2021 © Copyright Virtual University of
Pakistan



Running the Code

```
$ gcc server.c -o server
$ gcc client.c -o client
$ server &
[1] 432056
$ client
Hello, world!
Hello, class!
$
```

7 August 2021

© Copyright Virtual University of
Pakistan



Managing Unix/Linux Processes

- Commonly used process management commands:
 - `ps` – Display status of processes
 - `top` – Display CPU usage of processes

7 August 2021

© Copyright Virtual University of
Pakistan



Managing Unix/Linux Processes

`bg` – Put a process in the background

`fg` – Bring a process into the foreground

`jobs` – Display status of jobs

7 August 2021

© Copyright Virtual University of
Pakistan



Managing Unix/Linux Processes

`<Ctrl-Z>` – Suspend/stop a process

`<Ctrl-C>` – Send SIGINT signal to the foreground process

`kill` – Terminate a process

7 August 2021

© Copyright Virtual University of
Pakistan



The `ps` command

- `ps` command prints information about active processes.
- Without options, `ps` prints information about processes owned by the user
- Commonly used option: `-u`, `-e`, and `-l`

7 August 2021

© Copyright Virtual University of
Pakistan



The `ps` Command

```
$ ps
```

PID	TTY	TIME	CMD
1321	pts/0	00:00:00	csch
1345	pts/0	00:00:00	bash
1346	pts/0	00:00:00	ps

```
$
```

7 August 2021

© Copyright Virtual University of
Pakistan



The `ps` Command

```
$ ps -u root | more
```

PID	TTY	TIME	CMD
1	?	00:00:04	init
5	?	00:00:01	kswapd
712	?	00:00:00	inetd
799	?	00:00:00	cron
864	?	00:00:00	sshd
934	?	00:00:00	httpd
1029	tty1	00:00:00	getty

...

August 2021

© Copyright Virtual University of
Pakistan



The `ps` Command

```
$ ps -e | more
```

PID	TTY	TIME	CMD
1	?	00:00:04	init
2	?	00:00:00	keventd
3	?	00:00:00	ksoftirqd_CPU0
4	?	00:00:00	ksoftirqd_CPU1
5	?	00:00:01	kswapd
6	?	00:00:00	kreclaimd
7	?	00:00:00	bdflood
8	?	00:00:00	kupdated

...

August 2021

© Copyright Virtual University of
Pakistan



The top Command

- Displays information about the top processes (as many as can fit on the terminal or around 20 by default) on the system and periodically updates this information.
- Raw cpu percentage is used to rank the processes.

7 August 2021

© Copyright Virtual University of
Pakistan



The top Command

```
$ top
```

```
 9:42am up 5:15, 2 users, load average: 0.00, 0.00, 0.00
55 processes: 54 sleeping, 1 running, 0 zombie, 0 stopped
CPU0 states: 0.0% user, 0.1% system, 0.0% nice, 99.4% idle
CPU1 states: 0.0% user, 0.2% system, 0.0% nice, 99.3% idle
Mem: 513376K av, 237732K used, 275644K free, 60K shrd, 17944K buff
Swap: 257032K av, 0K used, 257032K free 106960K cached

  PID USER      PRI  NI  SIZE  RSS SHARE STAT  %CPU %MEM    TIME COMMAND
 1406 sarwar    19   0   896   896   700 R    0.3  0.1    0:00 top
 1382 nobody    10   0   832   832   660 S    0.1  0.1    0:00 in.telnetd
    1 root        9    0   536   536   460 S    0.0  0.1    0:04 init
    2 root        9    0     0     0     0 SW   0.0  0.0    0:00 keventd
...
```

```
$
```

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Use of FIFOs in a program
- Example code
- Process management commands in UNIX/Linux (`ps`, `top`)
- Recap of lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 12



Agenda for Today

- Review of previous lecture
- Process management commands:
bg, fg, ^Z, jobs, ^C, kill
- Thread concept
- Pros and cons of threads
- User and kernel threads
- Pthreads library with example
- Recap of lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 11

- Use of FIFOs in a program
- Sample code
- Process management in
UNIX/Linux
- UNIX/Linux commands/calls:
mknod, mkfifo, ps, top

The fg Command

- `fg [%job_id]`
- `fg` resumes the execution of a suspended job in the foreground and moves a background job into the foreground.
- If `%job_id` is omitted the current job is assumed.

7 August 2021

© Copyright Virtual University of
Pakistan



The bg command

- `bg [%job_id]`
- `bg` resumes the execution of a suspended job in the background.
- If `%job_id` is omitted the current job is assumed.
- `bg` runs the current or specified jobs in the background.

7 August 2021

© Copyright Virtual University of
Pakistan



The jobs command

- `jobs`
- `jobs` displays status of suspended and background processes.

Suspending a Process

- `<Ctrl-Z>` sends a STOP/SUSPEND signal to the current process. The shell displays a message saying that the job has been suspended and displays its prompt.
- You can then manipulate the state of this job, put it in the background with the `bg` command, run some other commands, and then eventually bring the job back into the foreground with the `fg` command.

Sample Session

```
$ find / -name foo -print 2> /dev/null
```

```
^Z
```

```
[1]+  Stopped      find / -name foo -print 2> /dev/null
```

```
$ bg
```

```
[1]+  find / -name foo -print 2> /dev/null &
```

```
$ jobs
```

```
[1]+  Running      find / -name foo -print 2> /dev/null &
```

```
$ fg
```

```
find / -name foo -print 2> /dev/null
```

```
[ command output ]
```

\$7 August 2021

© Copyright Virtual University of
Pakistan



Terminating Current Process

- `<Ctrl-C>` sends the SIGINT signal to the current process, which, by default, terminates the process.

```
$ find / -name foo -print 1> out 2> /dev/null
```

```
^C
```

```
$
```

7 August 2021

© Copyright Virtual University of
Pakistan



Terminating a Process

- `kill [-signal] PID` sends signal number 'signal' to process with ID 'PID'
- `kill 123` sends the SIGTERM signal to the process with ID 123
- `kill -9` sends the 'sure kill' signal, e.g., `kill -9 304294`

7 August 2021

© Copyright Virtual University of
Pakistan



Terminating a Process

- `kill -1` displays the signals supported by the system and their numbers

```
$ kill -1
```

1) SIGHUP	2) SIGINT	3) SIGQUIT	4) SIGILL
5) SIGTRAP	6) SIGABRT	7) SIGEMT	8) SIGFPE
9) SIGKILL	10) SIGBUS	11) SIGSEGV	12) SIGSYS
13) SIGPIPE	14) SIGALRM	15) SIGTERM	16) SIGUSR1

...

```
$
```

7 August 2021

© Copyright Virtual University of
Pakistan



Issues with Processes

- The fork() system call is expensive
- IPC is required to pass information between a parent and its child processes.

7 August 2021

© Copyright Virtual University of
Pakistan



Thread Concept

- A thread is a “lightweight” process which executes within the address space of a process.
- A thread can be scheduled to run on a CPU as an independent unit and terminate.
- Multiple threads can run simultaneously.

7 August 2021

© Copyright Virtual University of
Pakistan



Thread Concept

- Threads have their own
 - Thread ID
 - CPU context (PC, SP, register set, etc.)
 - Stack
 - Priority
 - errno

7 August 2021

© Copyright Virtual University of
Pakistan



Thread Concept

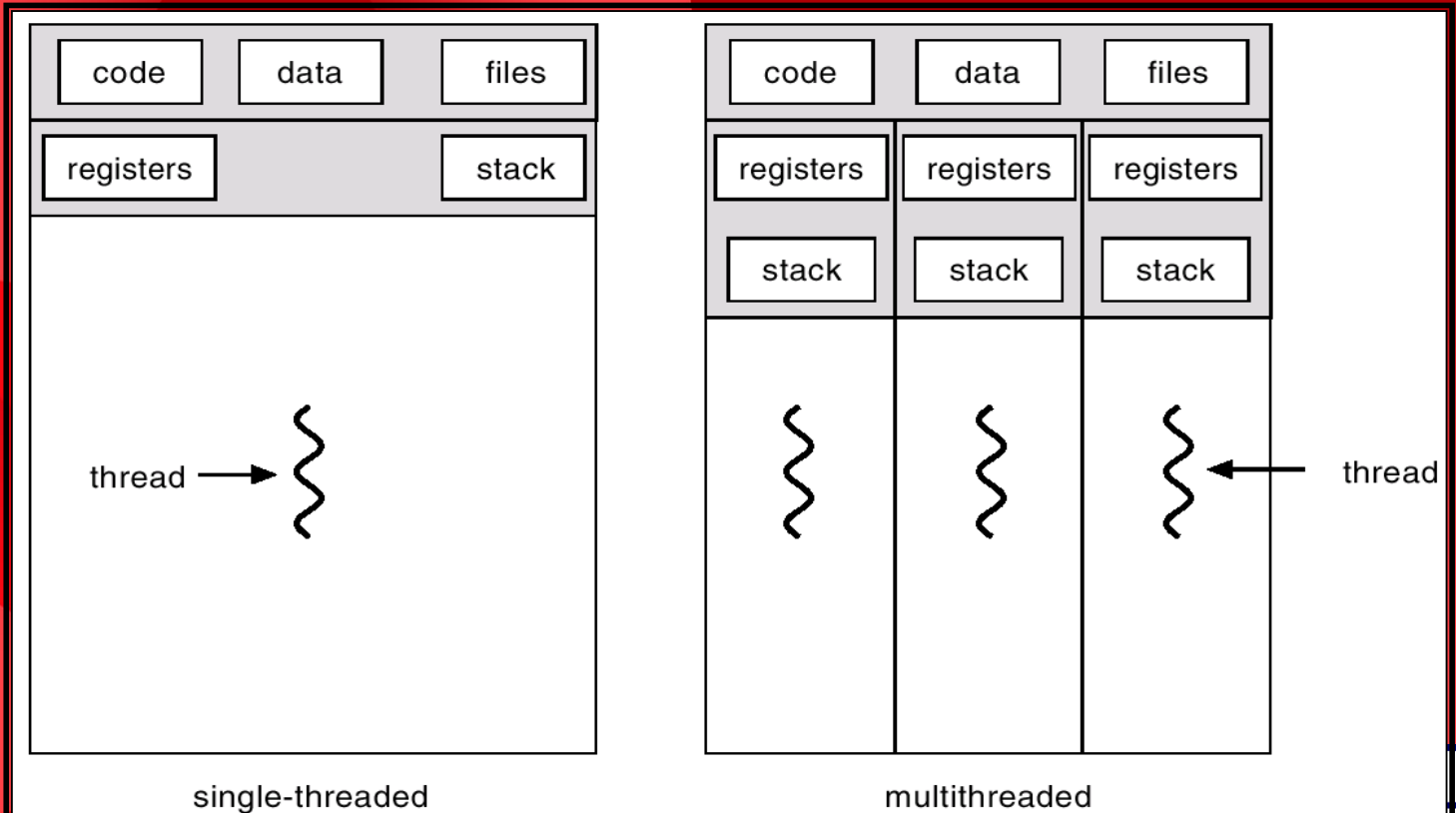
- Threads share
 - Code and data
 - Open files (through the PPFDT)
 - Current working directory
 - User and group IDs
 - Signal setups and handlers
 - PCB

7 August 2021

© Copyright Virtual University of
Pakistan



Single and Multithreaded Processes



Threads are Similar to Processes

- A thread can be in states similar to a process (new, ready, running, blocked, terminated)
- A thread can create another thread

Threads are Different from Processes

- Multiple threads can operate within the same address space
- No “automatic” protection mechanism is in place for threads—they are meant to help each other

7 August 2021

© Copyright Virtual University of
Pakistan



Advantages of Threads

- **Responsiveness**
 - Multi-threaded servers (e.g., browsers) can allow interaction with user while a thread is formulating response to a previous user query (e.g., rendering a web page)

7 August 2021

© Copyright Virtual University of
Pakistan



Advantages of Threads

- Resource sharing
 - Process resources (code, data, etc.)
 - OS resources (PCB, PPFDT, etc.)

7 August 2021

© Copyright Virtual University of
Pakistan



Advantages of Threads

- Economy
 - Take less time to create, schedule, and terminate
 - Solaris 2: thread creation is 30 times faster than process creation and thread switching is five times faster than process switching

7 August 2021

© Copyright Virtual University of
Pakistan



Advantages of Threads

- Performance in multi-processor and multi-threaded architectures (e.g., Intel's P4 HT)
 - Multiple threads can run simultaneously

7 August 2021

© Copyright Virtual University of
Pakistan



Disadvantages of Threads

- Resource sharing—synchronization needed between threads
- Difficult to write and debug multi-threaded programs

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- User and kernel threads
- Pthreads
- Sample code
- UNIX/Linux commands/calls:
`pthread_create,`
`pthread_join,`
`pthread_exit`

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 13



Agenda for Today

- Review of previous lecture
- Single- and multi-threaded process
- Thread models
- User and kernel threads
- Pthreads library with examples
- Recap of lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 12

- Process management commands:
fg, bg, jobs, ^Z, ^C, kill
- Issues with processes
- Thread concept
- Similarities and differences
between processes and threads
- Pros and cons of threads

7 August 2021

© Copyright Virtual University of
Pakistan



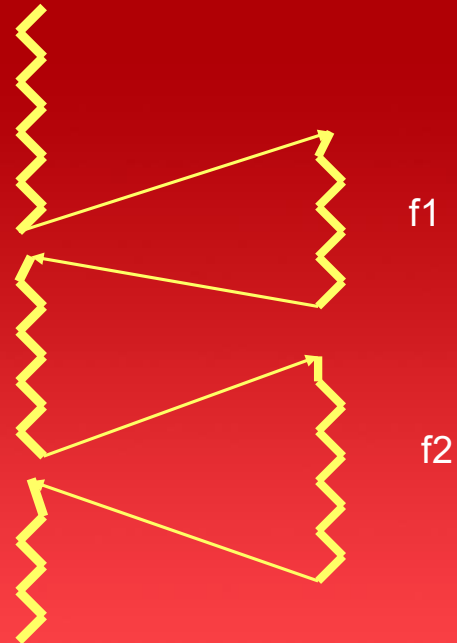
Single-Threaded Process

```
main()
{
    ...
    f1 (...);
    ...
    f2 (...);
    ...
}

f1 (...)
{ ... }

f2 (...)
{ ... }
```

Thread



Process
Terminated

7 August 2021

© Copyright Virtual University of
Pakistan



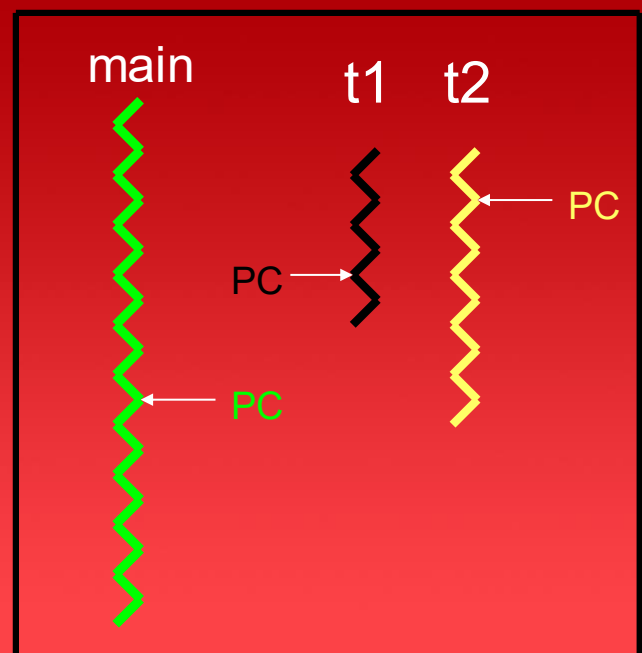
Multi-Threaded Process

```
main()
{
    ...
    thread(t1, f1);
    ...
    thread(t2, f2);
    ...
}

f1 (...)
{ ... }

f2 (...)
{ ... }
```

Process Address Space



7 August 2021

© Copyright Virtual University of
Pakistan



User Threads

- Thread management done by user-level threads libraries
 - Kernel not aware of threads
 - CPU not interrupted during thread switching
 - A system call by a thread blocks the whole process
 - Fair scheduling: P1 has one thread and P2 has 100 threads

7 August 2021

© Copyright Virtual University of
Pakistan



User Threads

- Examples
 - POSIX *Pthreads*
 - Mach *C-threads*
 - Solaris 2 *threads*

7 August 2021

© Copyright Virtual University of
Pakistan



Kernel Threads

- Thread management done by kernel
 - Kernel aware of threads
 - CPU switched during context switching
 - A system call does not block the whole process
 - Fair scheduling: P1 has one thread and P2 has 100

7 August 2021

© Copyright Virtual University of
Pakistan



Kernel Threads

- Examples
 - Windows NT/2000
 - Solaris 2
 - Linux

7 August 2021

© Copyright Virtual University of
Pakistan



Multithreading Models

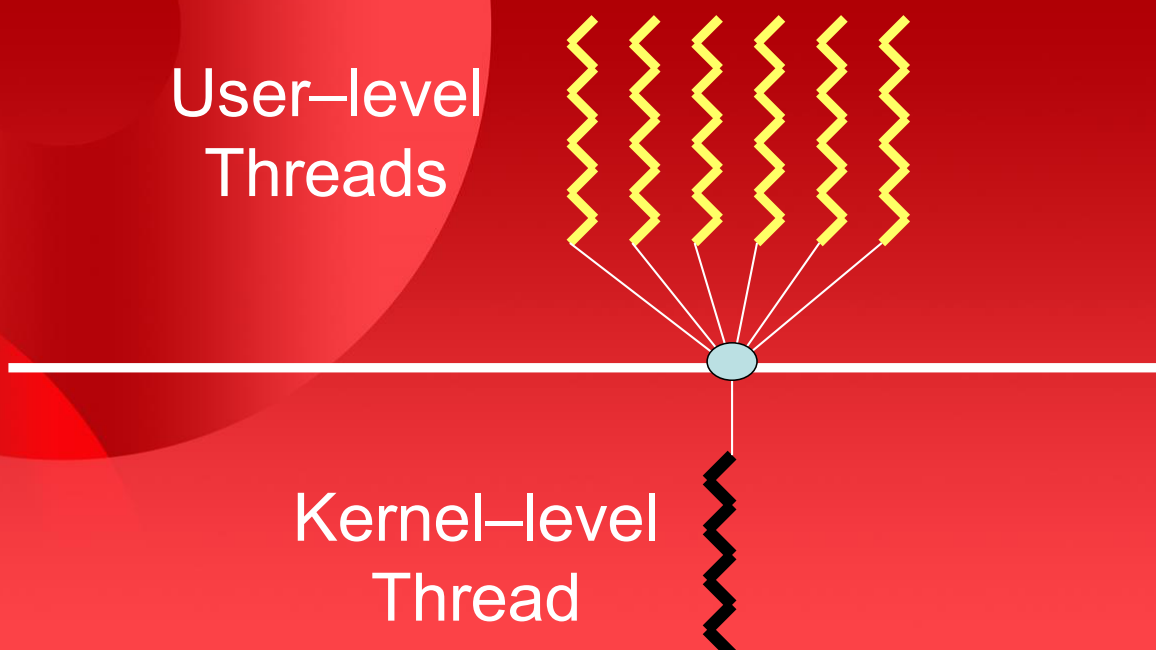
- Support for both user and kernel threads
- **Many-to-One:** Many user threads per kernel thread; process blocks when a thread makes a system call
- Solaris Green threads
- Pthreads

7 August 2021

© Copyright Virtual University of
Pakistan



Many-to-One Model



7 August 2021

© Copyright Virtual University of
Pakistan



Multithreading Models

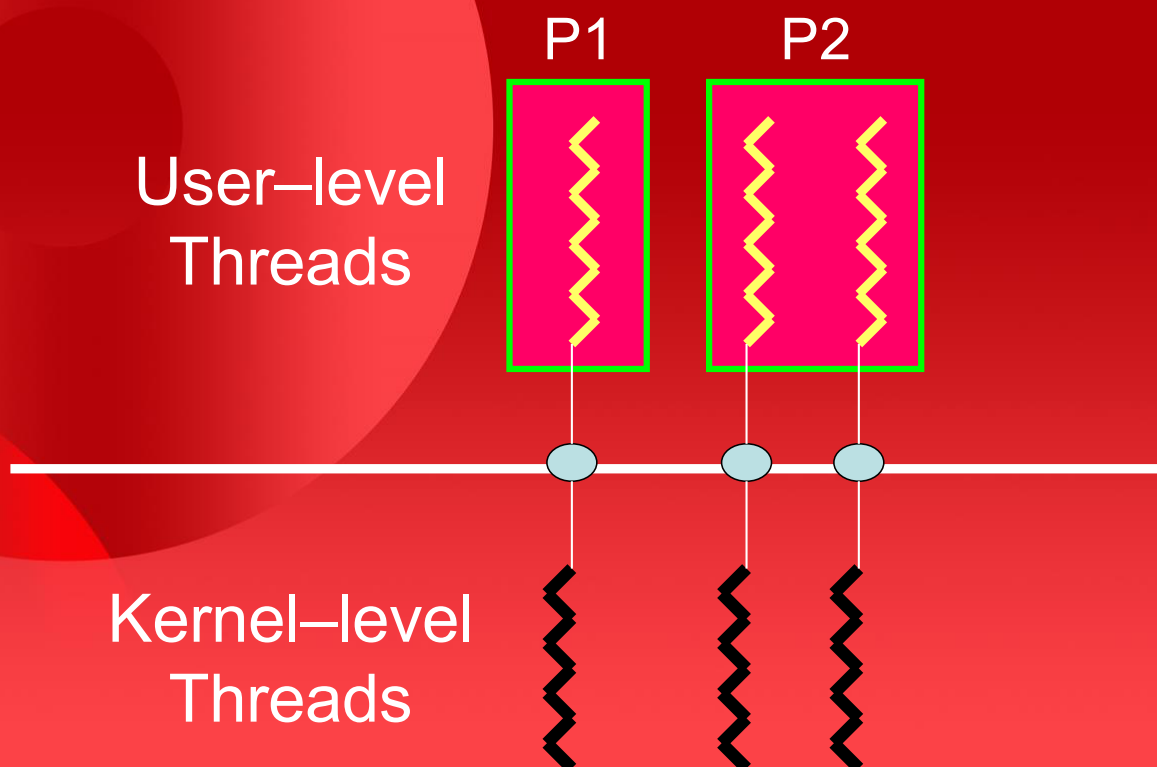
- **One-to-One:** One user thread per kernel thread; process does not block when a thread makes a system call
- Overhead for creating a kernel thread per user thread
- True concurrency achieved
- Windows NT/2000, OS/2

7 August 2021

© Copyright Virtual University of
Pakistan



One-to-One Model



7 August 2021

© Copyright Virtual University of
Pakistan



Multithreading Models

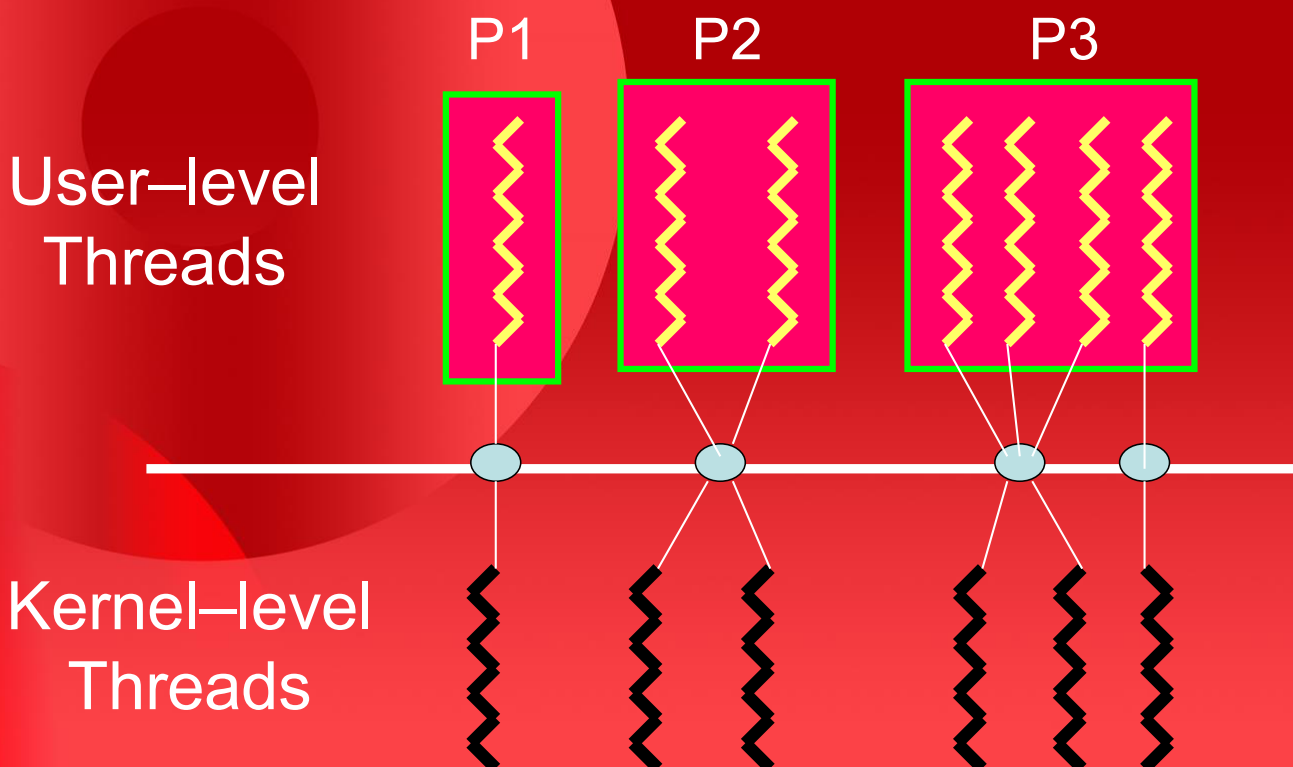
- **Many-to-Many:** Multiple user threads multiplexed over a smaller or equal number of kernel threads
- True concurrency not achieved because kernel can only schedule one thread at a time
- Kernel can schedule another thread when a user thread makes a blocking system call
- Solaris 2, HP-UX

7 August 2021

© Copyright Virtual University of Pakistan



Many-to-Many Model



7 August 2021

© Copyright Virtual University of Pakistan



Solaris 2 Threads Model

- Solaris 2: threads, light-weight processes (LWPs), and processes
 - At least one LWP per process to allow a user thread to talk to a kernel thread
 - User level threads switched and scheduled among LWPs without kernel's knowledge

7 August 2021

© Copyright Virtual University of
Pakistan



Solaris 2 Threads Model

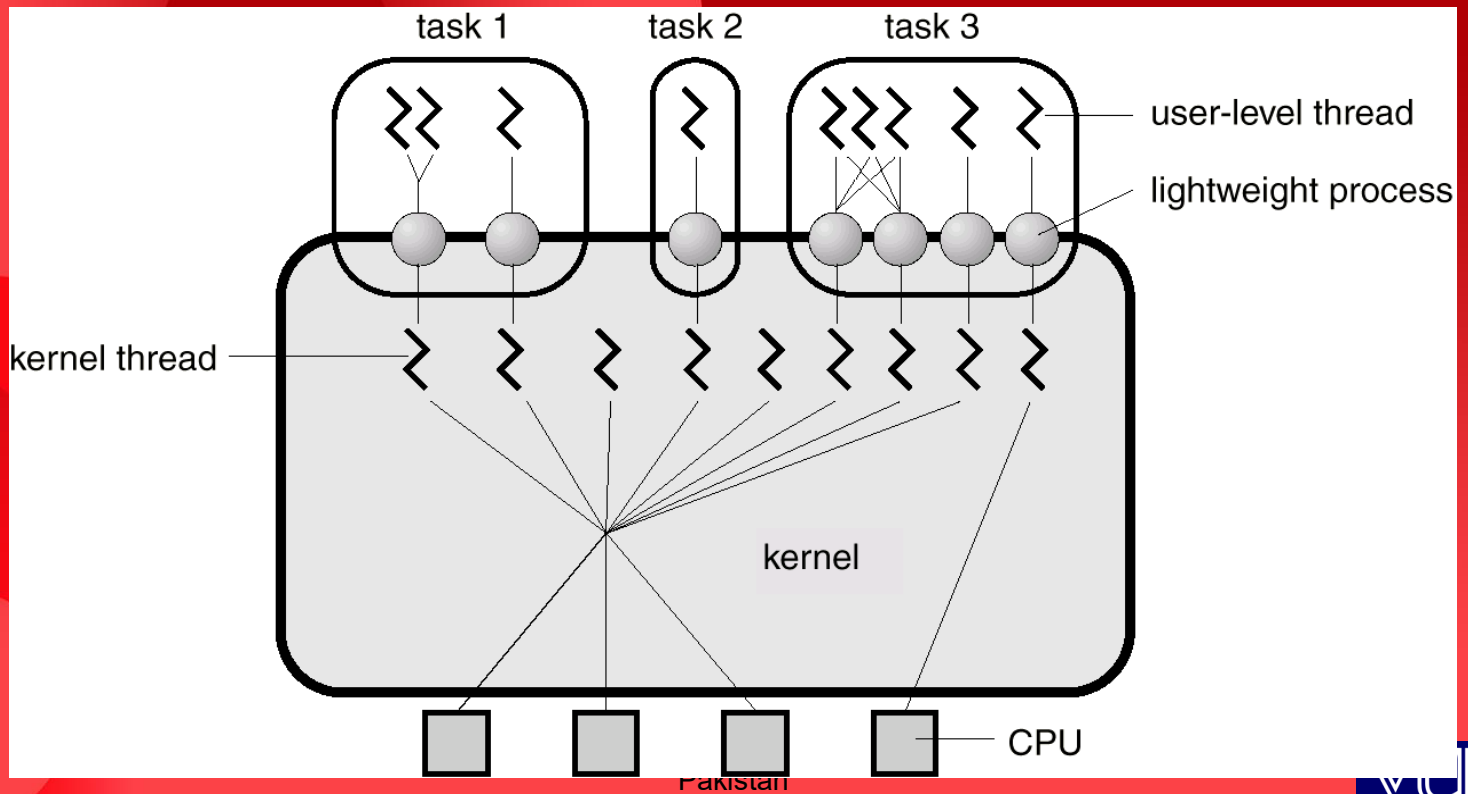
- One kernel thread per LWP; some kernel threads have no LWP (e.g., threads for clock interrupt handler and scheduling)

7 August 2021

© Copyright Virtual University of
Pakistan



Solaris 2 Threads Model



Pthreads

- A POSIX standard (IEEE 1003.1c) API for thread creation, termination, and synchronization.
- API specifies the behavior of the thread library, implementation is up to developers of the library.
- Common in UNIX operating systems.

Creating a Thread

```
■ int pthread_create (pthread_t *threadp,  
                      const pthread_attr_t *attr,  
                      void* (*routine)(void *),  
                      arg *arg);
```

7 August 2021

© Copyright Virtual University of
Pakistan



Creating a Thread

Where:

<code>threadp</code>	The thread we are trying to create—thread ID (TID)
<code>attr</code>	Used to modify the thread attributes (stack size, stack address, detached, joinable, priority, etc.)
<code>routine</code>	The thread function
<code>arg</code>	Any argument we want to pass to the thread function. This does not have to be a simple native type, it can be a 'struct' of whatever we want to pass in.

7 August 2021

© Copyright Virtual University of
Pakistan



Error Handling

- `pthread_create()` fails and returns the corresponding value if any of the following conditions is detected:
 - **EAGAIN** The system-imposed limit on the total number of threads in a process has been exceeded or some system resource has been exceeded (for example, too many LWPs were created).

7 August 2021

© Copyright Virtual University of
Pakistan



Error Handling

- **EINVAL** The value specified by `attr` is invalid.
- **ENOMEM** Not enough memory was available to create the new thread.
- **Error handling:**
 - `#include <errno.h>`
 - Error handling code

7 August 2021

© Copyright Virtual University of
Pakistan



Joining a Thread

- Waiting for a thread
- `int pthread_join(pthread_t aThread, void **statusp);`
- 'statusp' get return value of `pthread_exit`

7 August 2021

© Copyright Virtual University of
Pakistan



Joining a Thread

- Cannot join with a detached thread
- Can only join with thread's in the same process address space
- Multiple threads can join with one thread but only one returns successfully; others return with an error that no thread could be found with the given TID

7 August 2021

© Copyright Virtual University of
Pakistan



Terminating a Thread

- Main thread terminates
- Thread returns
- `void pthread_exit(void *valuep)`
- Returns value pointed to by 'valuep' to a joining thread, provided the exiting thread is not detached

7 August 2021

© Copyright Virtual University of
Pakistan



Example 1

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
/* Prototype for a function to be passed to our thread */
void* MyThreadFunc(void *arg);
int main()
{
    pthread_t aThread;
    /* Create a thread and have it run the MyThreadFunction */
    pthread_create(&aThread, NULL, MyThreadFunc, NULL);
    /* Parent waits for the aThread thread to exit */
    pthread_join(aThread, NULL);
    printf ("Exiting the main function.\n");
    return 0;
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Example 1

```
void* MyThreadFunc(void* arg)
{
    printf ("Hello, world! ... The threaded version.\n");
    return NULL;
}
```

```
$ gcc hello.c -o hello -lpthread -D_REENTRANT
```

```
$ hello
```

```
Hello, world! ... The threaded version.
```

```
Exiting the main function.
```

```
$
```

7 August 2021

© Copyright Virtual University of
Pakistan



Example 2

```
#include <pthread.h>
```

```
#include <stdio.h>
```

```
#define NUM_THREADS 5
```

```
void *PrintHello(void *threadid)
```

```
{
```

```
    printf("\n%d: Hello World!\n", threadid);
```

```
    pthread_exit(NULL);
```

```
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Example 2

```
int main (int argc, char *argv[])
{
    pthread_t threads[NUM_THREADS];
    int rc, t;

    for (t=0; t < NUM_THREADS; t++) {
        printf("Creating thread %d\n", t);
        rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
        if (rc) {
            printf("ERROR; return code is %d\n", rc);
            exit(-1);
        }
    }
    pthread_exit(NULL);
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Single- and multi-threaded processes
- User and kernel threads
- Thread models
- Pthreads library
- Sample code
- Pthread calls: pthread_create, pthread_join, pthread_exit

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 14



Agenda for Today

- Review of previous lecture
- Short-term scheduler
- Dispatcher
- Reasons for invoking scheduler
- Optimization criteria
- FCFS, SJF, and SRTF

Review of Lecture 13

- Single and multi-threaded processes
- Thread models
- User- and kernel-level threads
- Pthreads library
- Sample multi-threaded code

7 August 2021

© Copyright Virtual University of
Pakistan



Example 1

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
/* Prototype for a function to be passed to our thread */
void* MyThreadFunc(void *arg);
int main()
{
    pthread_t aThread;
    /* Create a thread and have it run the MyThreadFunction */
    pthread_create(&aThread, NULL, MyThreadFunc, NULL);
    /* Parent waits for the aThread thread to exit */
    pthread_join(aThread, NULL);
    printf ("Exiting the main function.\n");
    return 0;
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Example 1

```
void* MyThreadFunc(void* arg)
{
    printf ("Hello, world! ... The threaded version.\n");
    return NULL;
}
```

```
$ gcc hello.c -o hello -lpthread -D_REENTRANT
```

```
$ hello
```

Hello, world! ... The threaded version.

Exiting the main function.

```
$
```

7 August 2021

© Copyright Virtual University of
Pakistan



Example 2

```
#include <pthread.h>
```

```
#include <stdio.h>
```

```
#define NUM_THREADS 5
```

```
void *PrintHello(void *threadid)
```

```
{
```

```
    printf("\n%d: Hello World!\n", threadid);
```

```
    pthread_exit(NULL);
```

```
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Example 2

```
int main (int argc, char *argv[])
{
    pthread_t threads[NUM_THREADS];
    int rc, t;

    for (t=0; t < NUM_THREADS; t++) {
        printf("Creating thread %d\n", t);
        rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
        if (rc) {
            printf("ERROR; return code is %d\n", rc);
            exit(-1);
        }
    }
    pthread_exit(NULL);
}
```

7 August 2021

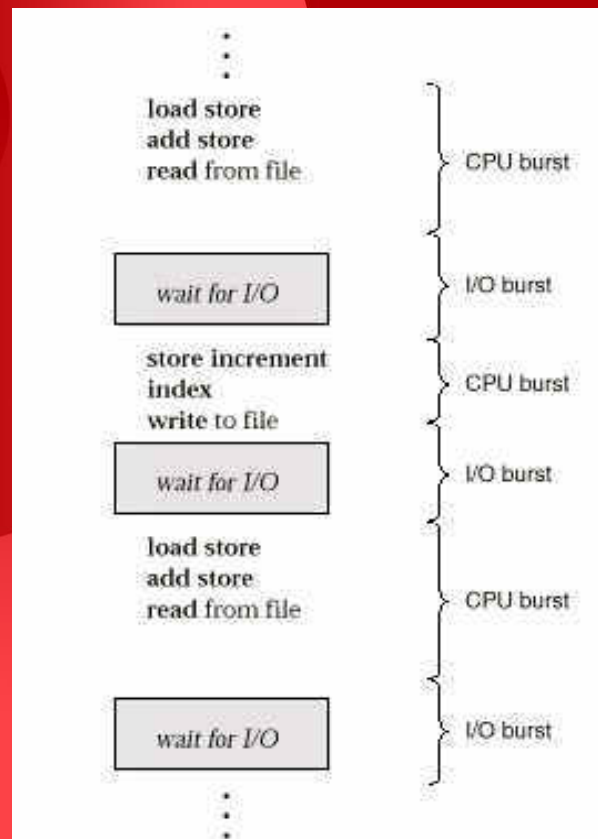
© Copyright Virtual University of
Pakistan



CPU Scheduling

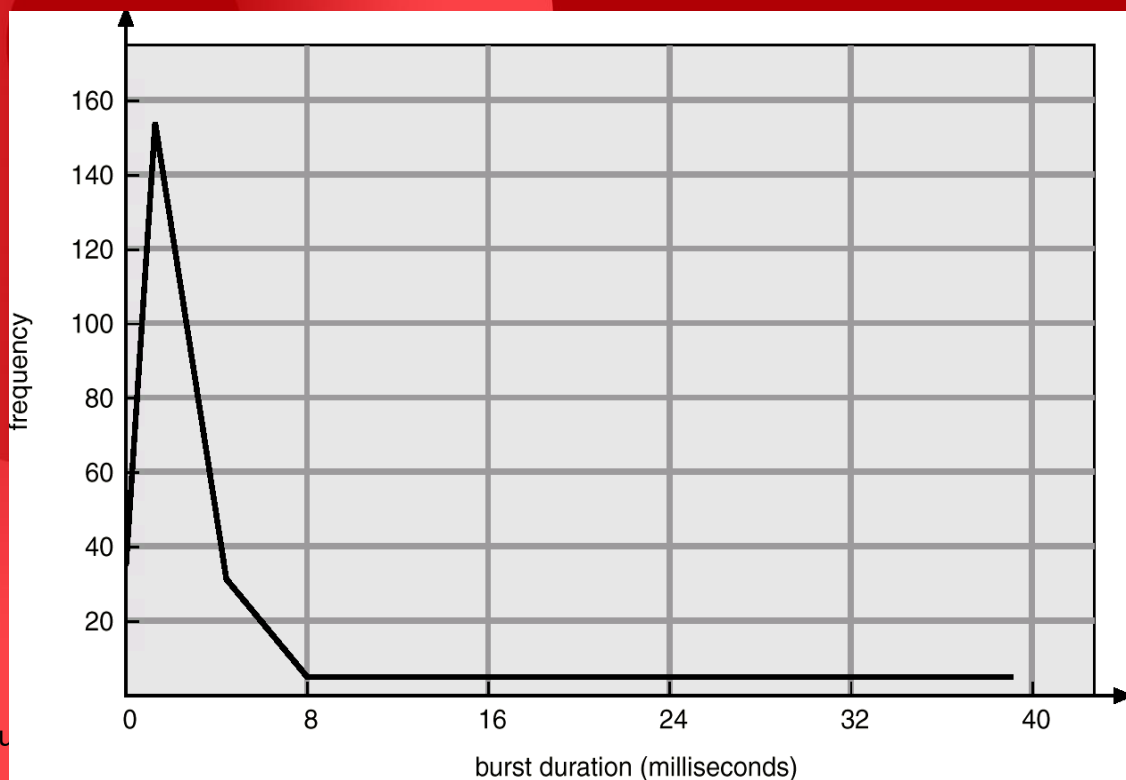
- Scheduling processes in the ready queue
- Short-term scheduler
- Different types of schedulers

Life of a Process



7 August 2021

Histogram of CPU-burst Times



7 August

CPU Scheduler

- Short-term scheduler
- Selects a process from among the processes in the ready queue
- Invokes the dispatcher to have the CPU allocated to the selected process

7 August 2021

© Copyright Virtual University of
Pakistan



Dispatcher

- Dispatcher gives control of the CPU to the process selected by the short-term scheduler; this involves:
 - switching context
 - switching to user mode
 - jumping to the proper location in the user program to start (or restart) it

7 August 2021

© Copyright Virtual University of
Pakistan



Dispatcher

- *Dispatch latency* – time it takes for the dispatcher to stop one process and start another running.
- Typically, a few microseconds

7 August 2021

© Copyright Virtual University of
Pakistan



CPU Scheduler

- CPU scheduling decisions may take place when a process:
 1. Switches from running to waiting state
 2. Switches from running to ready state
 3. Switches from waiting to ready
 4. Terminates

7 August 2021

© Copyright Virtual University of
Pakistan



CPU Scheduler

- Scheduling under 1 and 4 is *nonpreemptive*.
- All other scheduling is *preemptive*.

7 August 2021

© Copyright Virtual University of
Pakistan



Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process

7 August 2021

© Copyright Virtual University of
Pakistan



Scheduling Criteria

- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, **not** output (for time-sharing environment)

7 August 2021

© Copyright Virtual University of
Pakistan



Optimization Criteria

- Maximize CPU utilization
- Maximize throughput
- Minimize turnaround time
- Minimize waiting time
- Minimize response time

7 August 2021

© Copyright Virtual University of
Pakistan



FCFS Scheduling

- The process that enters the ready queue first is scheduled first, regardless of the size of its next CPU burst

- Example: Process Burst Time

P_1	24
P_2	3
P_3	3

- Suppose that processes arrive into the system in the order: P_1, P_2, P_3

7 August 2021

© Copyright Virtual University of
Pakistan



FCFS Scheduling

- Processes are served in the order: P_1, P_2, P_3
- The Gantt Chart for the schedule is:



- Waiting times $P_1 = 0; P_2 = 24; P_3 = 27$
- Average waiting time: $(0+24+27)/3 = 17$

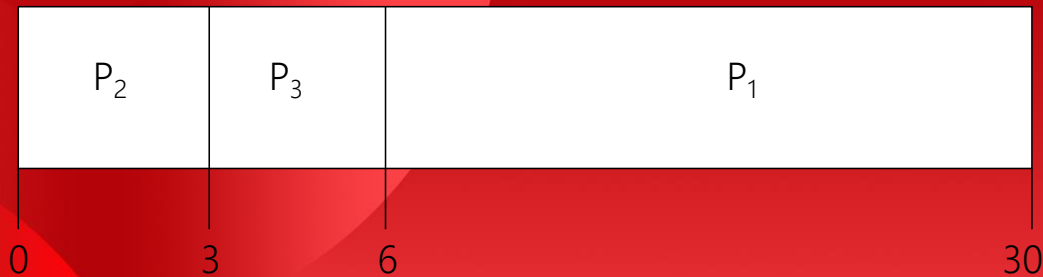
7 August 2021

© Copyright Virtual University of
Pakistan



FCFS Scheduling

- Suppose that processes arrive in the order: P_2 , P_3 , P_1 .
- The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- *Convoy effect* short process behind long process

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Short-term scheduler
- Examples of Pthreads
- Dispatcher
- Reasons for invoking scheduler
- Optimization criteria
- FCFS

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 15



Agenda for Today

- Review of previous lecture
- Shortest-job-first
- Round-Robin scheduling
- Multi-level queues scheduling
- Multi-level feedback queues scheduling
- Recap of lecture



Review of Lecture 14

- Short-term scheduler
- Dispatcher
- Reasons for invoking scheduler
- Preemptive and non-preemptive schedulers
- Optimization criteria
- FCFS

7 August 2021

© Copyright Virtual University of
Pakistan



Shortest-Job-First (SJF) Scheduling

- Process with the shortest CPU burst is scheduled first.
- **Non-preemptive** – once CPU given to a process it cannot be preempted until completes its CPU burst.

Shortest-Job-First (SJF) Scheduling

- **Preemptive** – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt it—**Shortest-Remaining-Time-First (SRTF)**.
- SJF is optimal non-preemptive scheduling algorithm – gives minimum average waiting time for a given set of processes.

7 August 2021

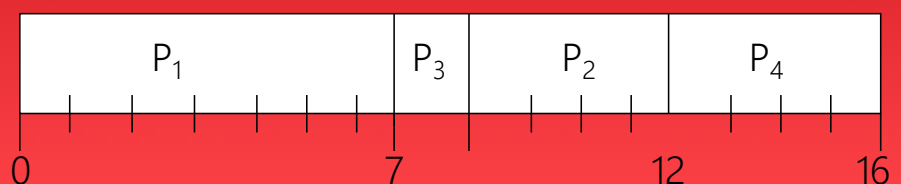
© Copyright Virtual University of Pakistan



Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- Gantt chart



- Average waiting time = $(0+6+3+7)/4 = 4$

7 August 2021

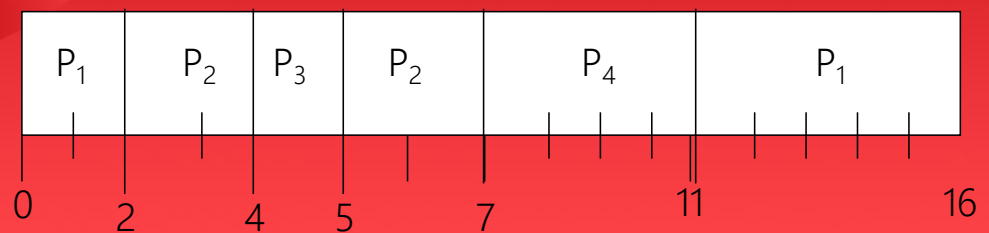
© Copyright Virtual University of Pakistan



Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

Gantt chart



- Average waiting time = $(9 + 1 + 0 + 2)/4 = 3$

7 August 2021

© Copyright Virtual University of
Pakistan



SJF—CPU Burst of a Process

- Can only estimate the length of the next CPU burst.
- Can be done by using the length of previous CPU bursts, using exponential averaging.

7 August 2021

© Copyright Virtual University of
Pakistan



Exponential Averaging

- Estimation based on historical data
- t_n = Length of n^{th} CPU burst
- τ_{n+1} = Estimate for $n+1^{st}$ CPU burst
- $\alpha, 0 \leq \alpha \leq 1$

$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n$$

7 August 20

Pakistan



Exponential Averaging

- Plugging in value for τ_n , we get

$$\begin{aligned}\tau_{n+1} &= \alpha t_n + (1 - \alpha)[\alpha t_{n-1} + (1 - \alpha)\tau_{n-1}] \\ &= \alpha t_n + (1 - \alpha)\alpha t_{n-1} + (1 - \alpha)^2 \tau_{n-1}\end{aligned}$$

- Continuing like this results in

$$\begin{aligned}\tau_{n+1} &= \alpha t_n + (1 - \alpha) \alpha t_{n-1} + \dots \\ &\quad + (1 - \alpha)^j \alpha t_{n-j} + \dots \\ &\quad + (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

7 August 2021

© Copyright Virtual University of
Pakistan



Exponential Averaging

- If $\alpha = 0$
 - $\tau_{n+1} = \tau_n$
 - Actual previous bursts do not count
- If $\alpha = 1$
 - $\tau_{n+1} = t_n$
 - Estimates do not count

7 August 2021

© Copyright Virtual University of
Pakistan



Exponential Averaging

- Typical value used for α is $\frac{1}{2}$.
With this value, our $n+1^{\text{st}}$ estimate is

$$\tau_{n+1} = t_n/2 + t_{n-1}/2^2 + t_{n-2}/2^3 + t_{n-3}/2^4 + \dots$$

- Hence the name exponential averaging

7 August 2021

© Copyright Virtual University of
Pakistan



Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer $\equiv$ highest priority).
 - Preemptive
 - Non-preemptive

7 August 2021

© Copyright Virtual University of
Pakistan



Priority Scheduling

- SJF is a priority scheduling where priority is the predicted next CPU burst time.
- Problem $\equiv$ Starvation – low priority processes may never execute.
- Solution $\equiv$ Aging – as time progresses increase the priority of the process.

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Shortest-Job-First (SJF)
- Shortest-Remaining-Time-First (SRTF)
- Exponential averaging
- Priority scheduling
- Recap of lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 16

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- SJF is optimal
- Round-Robin scheduling
- Multi-level queues scheduling
- Multi-level feedback queues scheduling
- Recap of lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 15

- Shortest-Job-First (SJF)
- Shortest-Remaining-Time-First (SRTF)
- Exponential averaging
- Priority scheduling

Review of Lecture 15

■ Exponential Averaging

$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n$$

for $\alpha = 1/2$

$$\tau_{n+1} = t_n/2 + t_{n-1}/2^2 + t_{n-2}/2^3 + t_{n-3}/2^4 + \dots$$

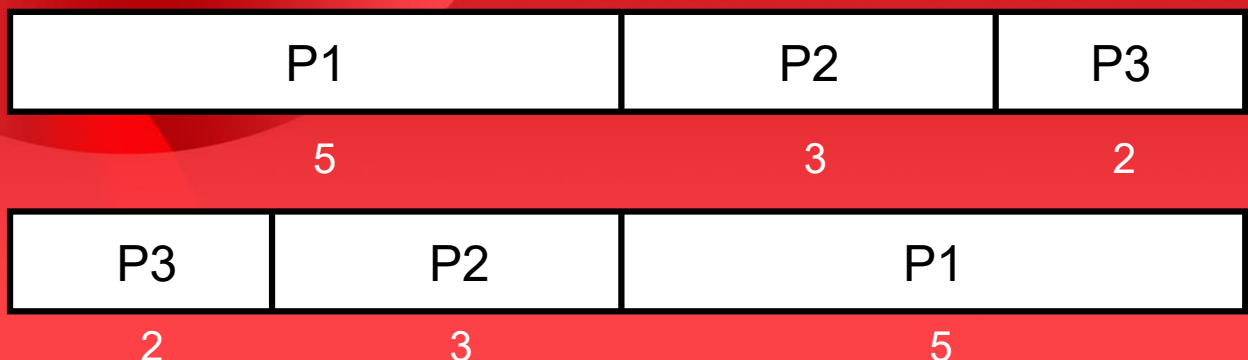
7 August 2021

© Copyright Virtual University of
Pakistan



SJF is Optimal

- **Logical Argument:** Decrease in the wait times for short processes is much more than increase in the wait times for long processes



7 August 2021

© Copyright Virtual University of
Pakistan



Round Robin (RR)

- Each process gets a small unit of CPU time, called *time slice* or *quantum*, which is usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.

7 August 2021

© Copyright Virtual University of
Pakistan



Round Robin (RR)

- If there are n processes in the ready queue, the time quantum is q , and context switch time is t_{cs} , then no process waits more than $(n-1)(q+t_{cs})$ time units
- Used in time-sharing systems where response time is an important performance criteria

7 August 2021

© Copyright Virtual University of
Pakistan



Round Robin (RR)

Performance

- q large $\Rightarrow$ FCFS
- q small $\Rightarrow q$ must be large with respect to context switch, otherwise overhead is too high.

7 August 2021

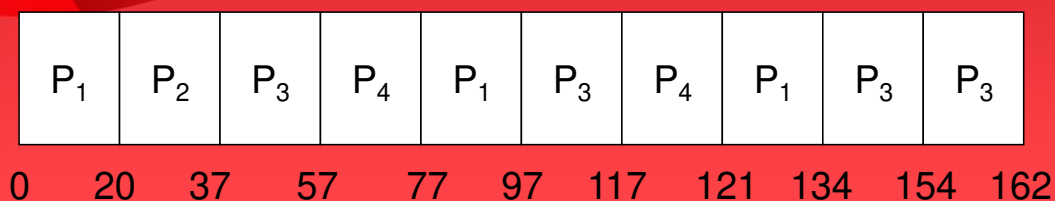
© Copyright Virtual University of
Pakistan



Round Robin Example

<u>Process</u>	<u>Burst Time</u>
P_1	53 — 33 — 13
P_2	17
P_3	68 — 48 — 28 — 8
P_4	24 — 4

- The Gantt chart with quantum 20 is:



7 August 2021

© Copyright Virtual University of
Pakistan



Round Robin Example

<u>Process</u>	<u>Turnaround Time</u>	<u>Waiting Time</u>
P_1	134	$134 - 53 = 81$
P_2	37	$37 - 17 = 20$
P_3	162	$162 - 68 = 94$
P_4	121	$121 - 24 = 97$

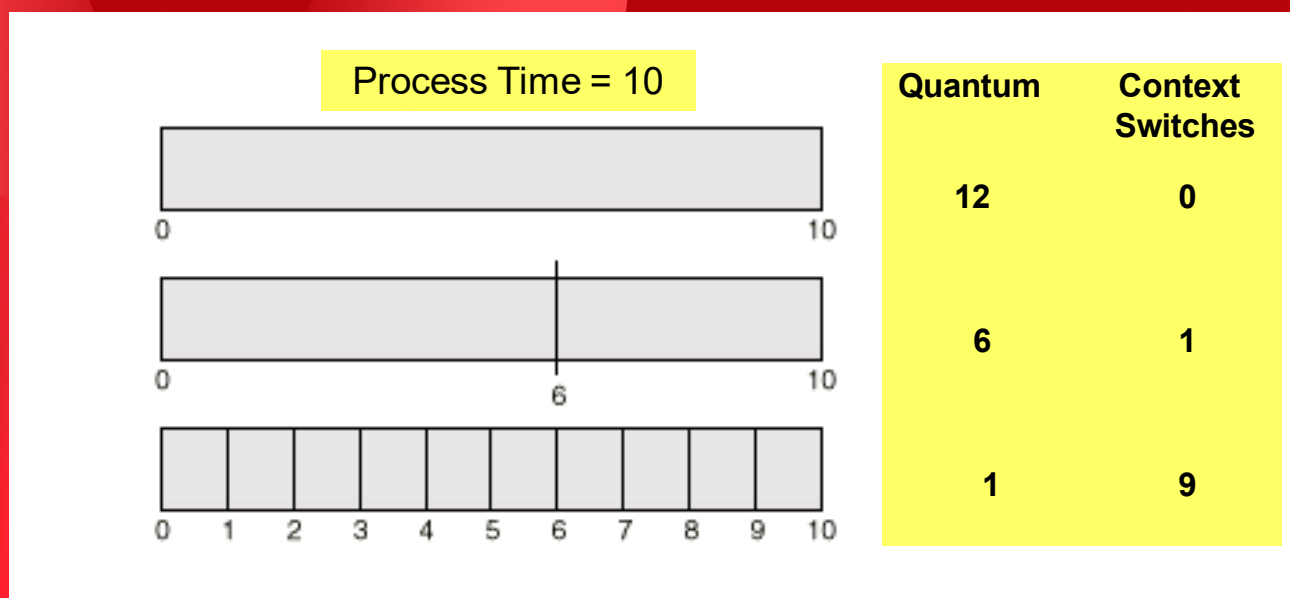
- Average waiting time = 73
- Average waiting time for SJF = 38
- Typically, higher average turnaround than SJF, but better *response*.

7 August 2021

© Copyright Virtual University of
Pakistan



Quantum vs Context Switch

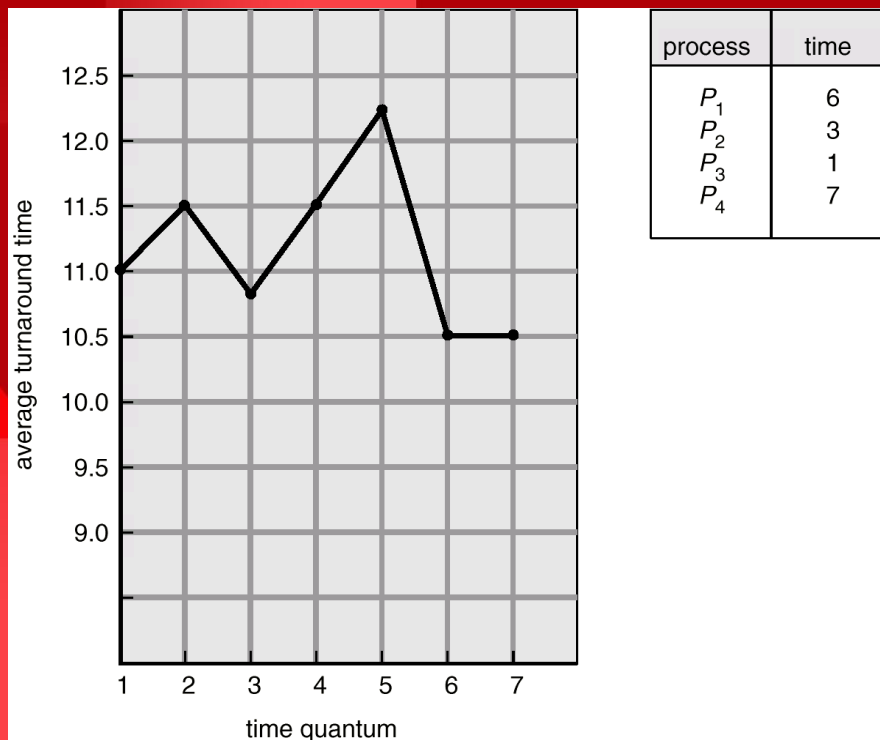


7 August 2021

© Copyright Virtual University of
Pakistan



Turnaround Time vs Quantum



7 August 2021

Multilevel Queues

- Ready queue is partitioned into separate queues:
 - foreground (interactive)
 - background (batch)
- Each queue has its own priority and scheduling algorithm:
 - foreground – RR
 - background – FCFS

7 August 2021

Multilevel Queues

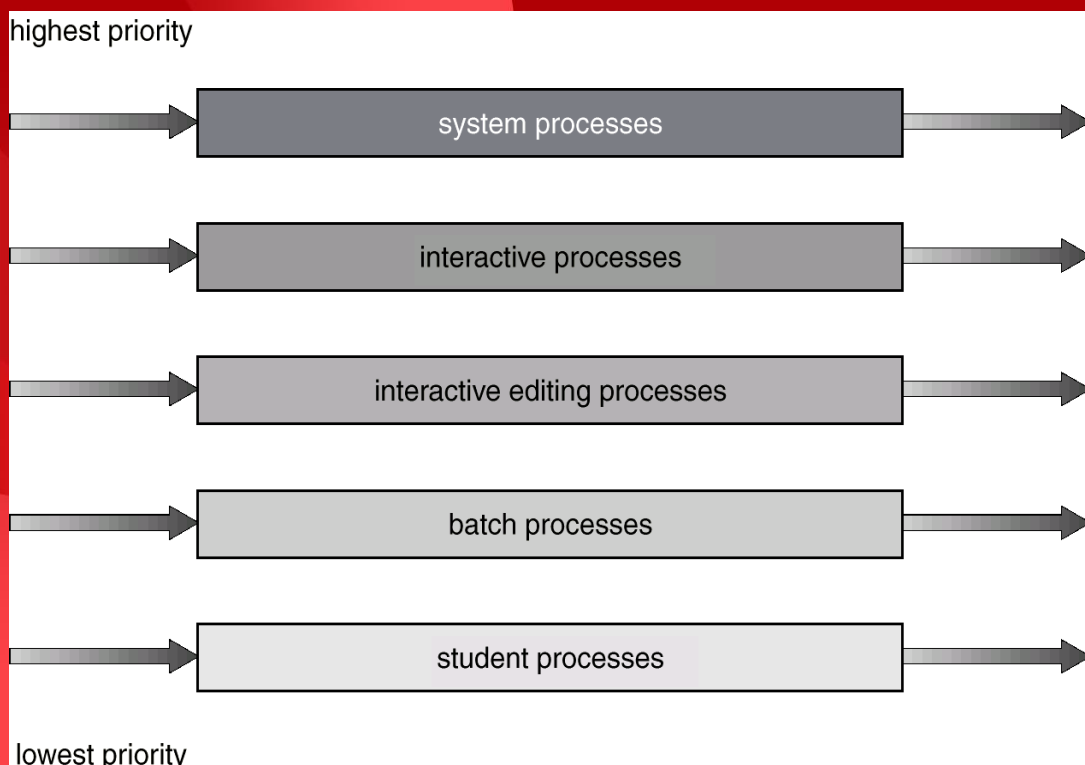
- Scheduling must be done across queues.
 - Fixed priority scheduling; i.e., serve all from foreground then from background.
 - Time slice – each queue gets a certain percentage of CPU time, e.g., 80% to foreground in RR and 20% to background in FCFS

7 August 2021

© Copyright Virtual University of
Pakistan



Multilevel Queues



7 August 2021

Pakistan



Recap of Lecture

- SJF is optimal
- Round-Robin scheduling
- Multi-level queues scheduling

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 17

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Multi-level queues scheduling
- Multi-level feedback queues scheduling
- UNIX System V scheduling
- Algorithm evaluation
- Recap of lecture

7 August 2021

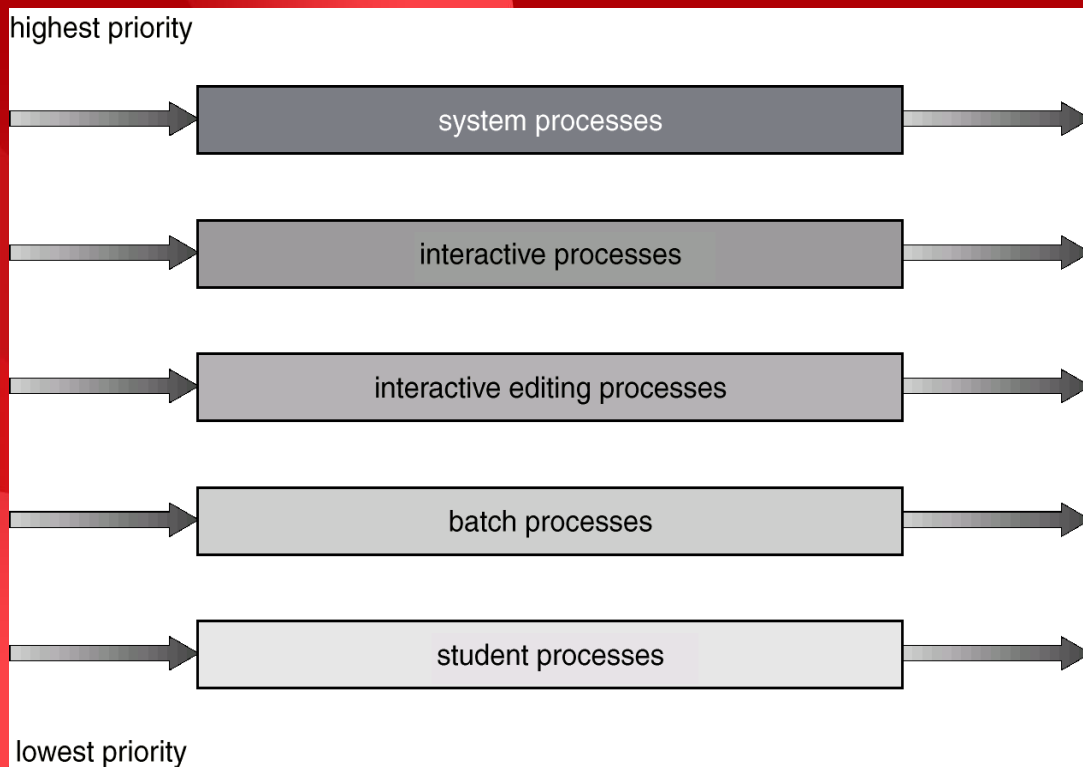
© Copyright Virtual University of
Pakistan



Review of Lecture 16

- Shortest-Job-First is optimal
- Round-Robin scheduling
- Multi-level queues scheduling

Multilevel Queues



7 August

Pakistan

Multilevel Feedback Queues

- A process can move between the various queues; aging can be implemented this way.
- Multilevel-feedback-queue scheduler defined by the following parameters:
 - Number of queues
 - Scheduling algorithms for each queue

7 August 2021

Multilevel Feedback Queues

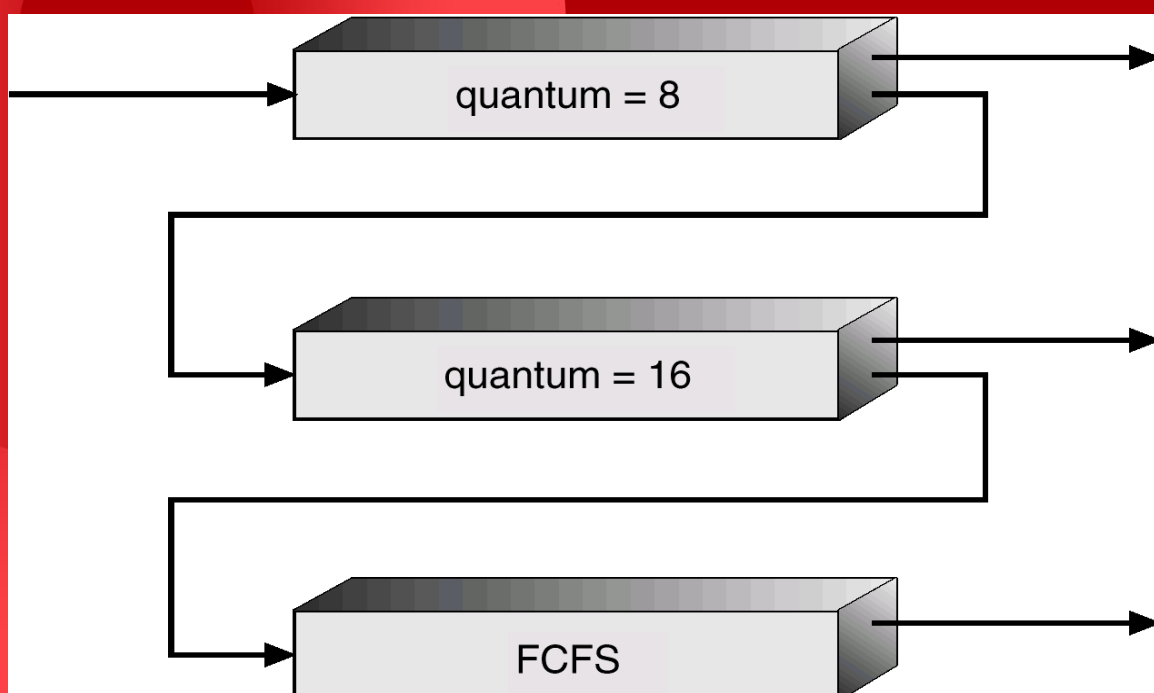
- Method used to determine when to upgrade a process
- Method used to determine when to demote a process
- Method used to determine which queue a process will enter when that process needs service

7 August 2021

© Copyright Virtual University of
Pakistan



Multilevel Feedback Queues



7 August 2021

© Copyright Virtual University of
Pakistan



UNIX System V Scheduling Algorithm

- Multilevel feedback priority queues with round robin within each queue
- Quantum = 1 second
- Priorities are divided into two groups/bands:
 - Kernel Group
 - User Group

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX System V Scheduling Algorithm

- Priorities in the Kernel Group are assigned in a manner to minimize bottlenecks: processes waiting in a lower-level routine get higher priorities than those waiting at relatively higher-level routines.

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX System V Scheduling Algorithm

- In decreasing order of priority, the bands are:
 - Swapper
 - Block I/O device control processes
 - File manipulation
 - Character I/O device control processes
 - User processes

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX System V Scheduling Algorithm

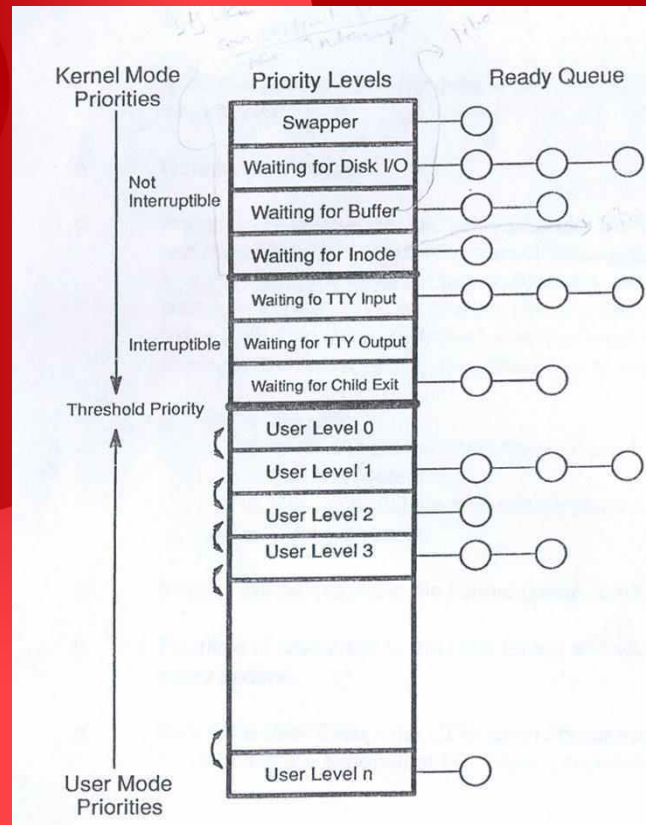
- Priorities of processes in the Kernel Group remain fixed.
- Priorities of processes in the User Group are recalculated every second.
- Inside the User Group, the CPU-bound processes are penalized at the expense of I/O-bound processes

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX System V Scheduling Algorithm



7 August 2021



UNIX System V Scheduling Algorithm

- The CPU usage of each process is updated every clock tick by the clock ISR.
- Every second, a *decay function* is applied to the CPU usage of each process in the ready queue:

$$\text{Decay (CPU Usage)} = (\text{CPU Usage}) / 2$$

- Priority number of each process is recomputed every second

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX System V Scheduling Algorithm

- Every second, the priority number of all those processes that are in the main memory and ready to run is updated by using the following formula:

$$\text{Priority\#} = (\text{Recent CPU Usage})/2 + \text{Thr. Pri.} + \text{nice}$$

- Threshold priority and nice values are always positive to prevent a user from migrating out of its assigned group

7 August 2021

© Copyright Virtual University of Pakistan



UNIX System V Example

P _A			P _B		P _C	
Time	Priority	CPU Count	Priority	CPU Count	Priority	CPU Count
0	60	0 1 ...	60	0	60	0
1	75	60 30	60	0 1 ...	60	0
2	67	30 15	75	60 30	60	0 1
3	63	7 8 ...	67	30 15	75	...
4	76	67 33	63	7 8 ...	67	60 30 15
5	68	16	76	33	63	7

7 August 2021

© Copyright Virtual University of Pakistan



Recap of Lecture

- Multi-level queues scheduling
- Multi-level feedback queues scheduling
- UNIX System V scheduling

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 18

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- UNIX System V scheduling
- Algorithm evaluation
- Process synchronization
- Recap of lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 17

- Multi-level queues scheduling
- Multi-level feedback queues scheduling
- UNIX System V scheduling algorithm

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX System V Scheduling Algorithm

- Every second, the priority number of all those processes that are in the main memory and ready to run is updated by using the following formula:

$$\text{Priority\#} = (\text{Recent CPU Usage})/2 + \text{Thr. Pri.} + \text{nice}$$

- Threshold priority and nice values are always positive to prevent a user from migrating out of its assigned group

7 August 2021

© Copyright Virtual University of Pakistan



UNIX System V Example

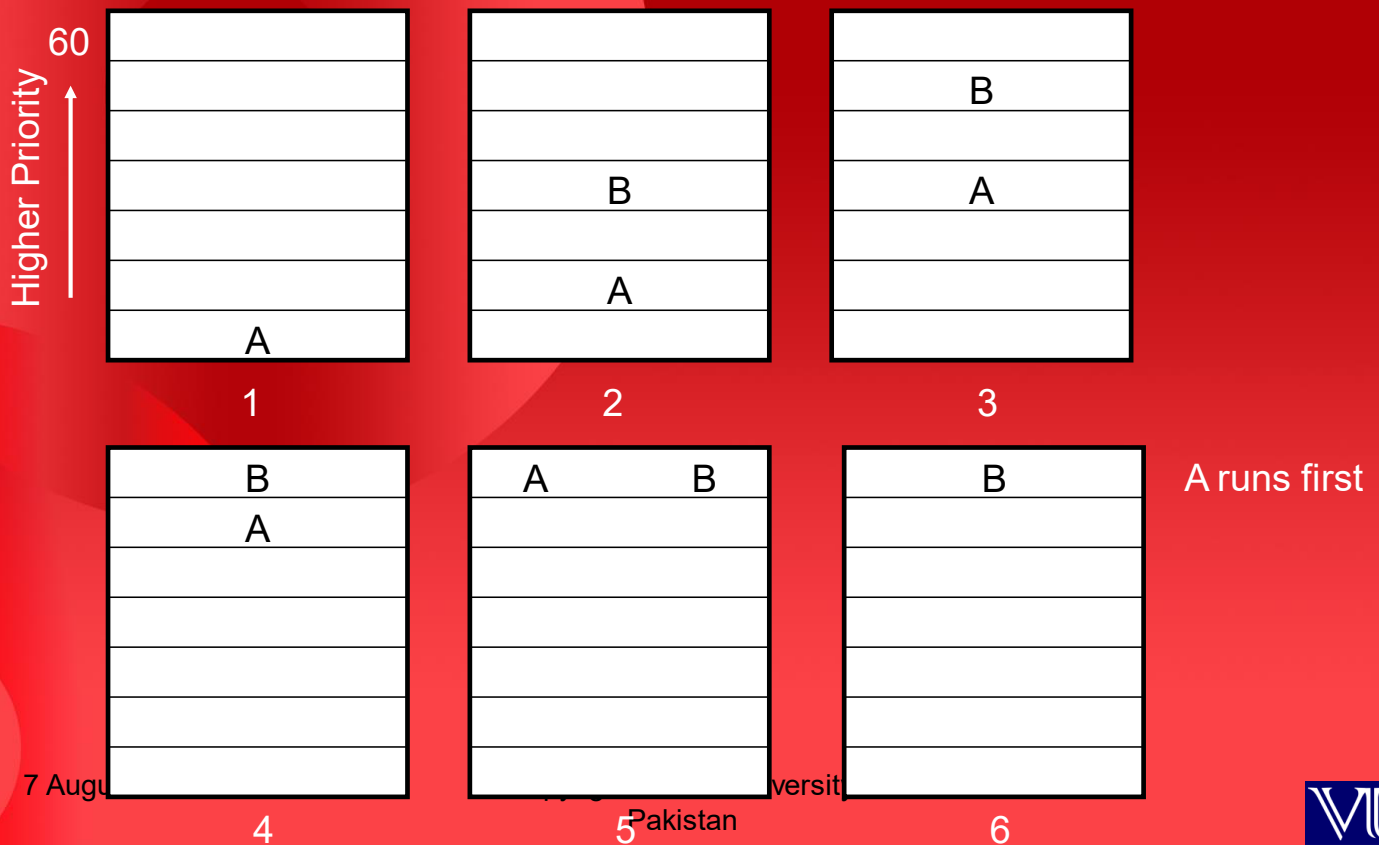
P _A			P _B		P _C	
Time	Priority	CPU Count	Priority	CPU Count	Priority	CPU Count
0	60	0 1 ...	60	0	60	0
1	75	60 30	60	0 1 ...	60	0
2	67	30 15	75	60 30	60	0 1
3	63	7 8 ...	67	30 15	75	...
4	76	67 33	63	7 8 ...	67	60 30
5	68	16	76	33	63	15 7

7 August 2021

© Copyright Virtual University of Pakistan



Round Robin Scheduling and Process Priorities



Algorithm Evaluation

- Analytic Evaluation
 - The algorithm and some system workload are used to produce a formula or number which gives the performance of the algorithm for that workload.
 - Deterministic modeling
 - Queuing models
- Implementation

Deterministic Modeling

- Predetermined workload and performance of each algorithm for that workload. Use of Gantt charts.
- Simple and fast
- Exact numbers for comparison
- Requires exact input
- Performance figures may not be true in general

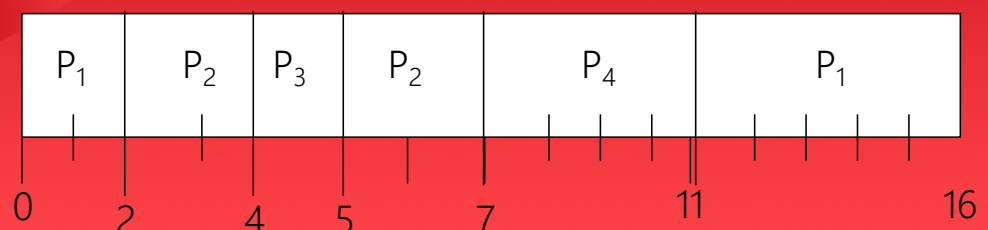
7 August 2021

© Copyright Virtual University of
Pakistan



Deterministic Modeling

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4
- Gantt chart



- Average waiting time = $(9 + 1 + 0 + 2)/4 = 3$

7 August 2021

© Copyright Virtual University of
Pakistan



Queuing Modeling

- Computer system viewed as a network of queues and servers: ready queue, I/O queue, event queues, CPUs, I/O device controllers, etc.
- **Input:** Arrival and service rates
- **Output:** CPU utilization, average queue length, average waiting time, ...

7 August 2021

© Copyright Virtual University of
Pakistan



Queuing Modeling

Little's Formula:

$$n = \lambda * W$$

where

n = average queue length

λ = average arrival rate

W = average waiting time in a queue

7 August 2021

© Copyright Virtual University of
Pakistan



Queuing Modeling

Let the average job arrival rate be 0.5

Algorithm	Average Wait Time $W=t_w$	Average Queue Length(n)
FCFS	4.6	2.3
SJF	3.6	1.8
SRTF	3.2	1.6
RR (q=1)	7.0	3.5
RR (q=4)	6.0	3.0

7 August 2021

© Copyright Virtual University of
Pakistan



Queuing Modeling

- Complicated mathematics
- Distributions (Poisson, uniform, exponential, etc) for the arrival and departure rates can be difficult to work with
- Assumptions may not be accurate
- Approximation of the real system

7 August 2021

© Copyright Virtual University of
Pakistan



Simulation

- Programming model for the computer system
- Workload generated by assuming some distribution and a random number generator, or by collecting data from the actual system.

7 August 2021

© Copyright Virtual University of
Pakistan



Simulation

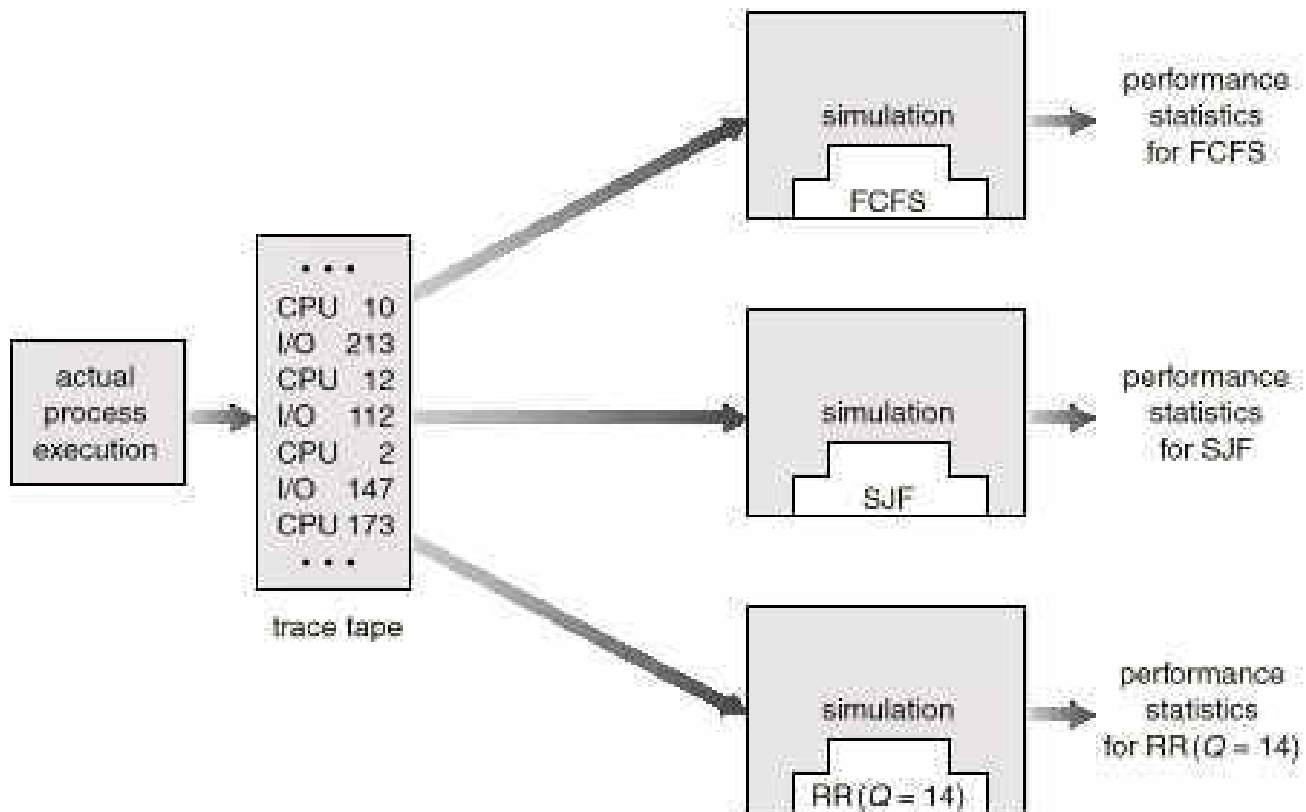
- Characteristics
 - Expensive: hours of programming and execution time
 - May be erroneous because of the assumptions about distributions

7 August 2021

© Copyright Virtual University of
Pakistan



Simulation



Implementation

- Best
- Most expensive
 - Good option due to Open Source kernels such as Linux

Process Synchronization

- Concurrent access to shared data may result in data inconsistency.
- Maintaining data consistency requires mechanisms to ensure that cooperating processes access shared data sequentially.

7 August 2021

© Copyright Virtual University of Pakistan



Bounded-Buffer Problem

Shared data

```
#define BUFFER_SIZE 10
```

```
typedef struct {
```

```
    . . .
```

```
} item;
```

```
item buffer[BUFFER_SIZE];
```

```
int in = 0, out = 0;
```

```
int counter = 0;
```

7 August 2021

© Copyright Virtual University of Pakistan



Bounded-Buffer Problem

Producer process

```
item nextProduced;  
...  
while (1) {  
    while (counter == BUFFER_SIZE) ;  
    buffer[in] = nextProduced;  
    in = (in + 1) % BUFFER_SIZE;  
    counter++;  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Bounded-Buffer Problem

Consumer process

```
item nextConsumed;  
while (1) {  
    while (counter == 0) ;  
    nextConsumed = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
    counter--;  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Bounded-Buffer Problem

- “counter++” in assembly language

MOV R1, counter

INC R1

MOV counter, R1

- “counter--” in assembly language

MOV R2, counter

DEC R2

MOV counter, R2

7 August 2021

© Copyright Virtual University of
Pakistan



Bounded-Buffer Problem

- If both the producer and consumer attempt to update the buffer concurrently, the machine language statements may get interleaved.
- Interleaving depends upon how the producer and consumer processes are scheduled.

7 August 2021

© Copyright Virtual University of
Pakistan



Bounded-Buffer Problem

- Assume counter is initially 5. One interleaving of statements is:

producer:	MOV R1, counter	(R1 = 5)
	INC R1	(R1 = 6)
consumer:	MOV R2, counter	(R2 = 5)
	DEC R2	(R2 = 4)
producer:	MOV counter, R1	(counter = 6)
consumer:	MOV counter, R2	(counter = 4)

- The value of count may be either 4 or 6, where the correct result should be 5.

7 August 2021

© Copyright Virtual University of
Pakistan



Process Synchronization

- Race Condition:** The situation where several processes access and manipulate shared data concurrently, the final value of the data depends on which process finishes last.

7 August 2021

© Copyright Virtual University of
Pakistan



Process Synchronization

- **Critical Section:** A piece of code in a cooperating process in which the process may updates shared data (variable, file, database, etc.).
- **Critical Section Problem:** Serialize executions of critical sections in cooperating processes

7 August 2021

© Copyright Virtual University of
Pakistan



Solution of the Critical Problem

- Software based solutions
- Hardware based solutions
- Operating system based solution

7 August 2021

© Copyright Virtual University of
Pakistan



Structure of Solution

do {

entry section

critical section

exit section

reminder section

} while (1);

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- UNIX System V scheduling
- Algorithm evaluation
- Process synchronization
- Recap of lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 18

Syed Mansoor Sarwar



Operating Systems

Lecture 19

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Process synchronization
- The Critical Section Problem
- Conditions for a Good Solution
- 2-Process Critical Section Problem Solutions
- Recap of lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 18

- UNIX System V scheduling algorithm
- Algorithm Evaluation
- Process Synchronization
- The Critical Section Problem

Bounded-Buffer Problem

Producer process

```
item nextProduced;  
...  
while (1) {  
    nextProduced = getNewItem();  
    while (counter == BUFFER_SIZE) ;  
    buffer[in] = nextProduced;  
    in = (in + 1) % BUFFER_SIZE;  
    counter++;  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Bounded-Buffer Problem

Consumer process

```
item nextConsumed;  
while (1) {  
    while (counter == 0) ;  
    nextConsumed = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
    counter--;  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Bounded-Buffer Problem

- “counter++” in assembly language

MOV R1, counter

INC R1

MOV counter, R1

- “counter--” in assembly language

MOV R2, counter

DEC R2

MOV counter, R2

7 August 2021

© Copyright Virtual University of
Pakistan



Bounded-Buffer Problem

- If both the producer and consumer attempt to update the buffer concurrently, the machine language statements may get interleaved.
- Interleaving depends upon how the producer and consumer processes are scheduled.

7 August 2021

© Copyright Virtual University of
Pakistan



Bounded-Buffer Problem

- Assume counter is initially 5. One interleaving of statements is:

producer:	MOV R1, counter	(R1 = 5)
	INC R1	(R1 = 6)
consumer:	MOV R2, counter	(R2 = 5)
	DEC R2	(R2 = 4)
producer:	MOV counter, R1	(counter = 6)
consumer:	MOV counter, R2	(counter = 4)

- The value of count may be either 4 or 6, where the correct result should be 5.

7 August 2021

© Copyright Virtual University of
Pakistan



Process Synchronization

- Race Condition:** The situation where several processes access and manipulate shared data concurrently, the final value of the data depends on which process finishes last.

7 August 2021

© Copyright Virtual University of
Pakistan



Process Synchronization

- **Critical Section:** A piece of code in a cooperating process in which the process may updates shared data (variable, file, database, etc.).
- **Critical Section Problem:** Serialize executions of critical sections in cooperating processes

7 August 2021

© Copyright Virtual University of
Pakistan



Process Synchronization

- **More Examples**
 - Bank transactions
 - Airline reservation

7 August 2021

© Copyright Virtual University of
Pakistan



Bank Transactions



Deposit

MOV A, Balance

ADD A, Deposited

MOV Balance, A

Withdrawal

MOV B, Balance

SUB B, Withdrawn

MOV Balance, B

7 August 2021

© Copyright Virtual University of
Pakistan



Bank Transactions

■ Bank Transactions

- Current balance = Rs. 50,000
- Check deposited = Rs. 10,000
- ATM withdrawn = Rs. 5,000

7 August 2021

© Copyright Virtual University of
Pakistan



Bank Transactions

Check Deposit:

MOV A, Balance // A = 50,000

ADD A, Deposited // A = 60,000

ATM Withdrawal:

MOV B, Balance // B = 50,000

SUB B, Withdrawn // B = 45,000

Check Deposit:

MOV Balance, A // Balance = 60,000

ATM Withdrawal:

MOV Balance, B // Balance = 45,000

7 August 2021

© Copyright Virtual University of
Pakistan



Solution of the Critical Problem

- Software based solutions
- Hardware based solutions
- Operating system based solution

7 August 2021

© Copyright Virtual University of
Pakistan



Structure of Solution

do {

entry section

critical section

exit section

reminder section

} while (1);

7 August 2021

© Copyright Virtual University of
Pakistan



Solution to Critical-Section Problem

- 2-Process Critical Section Problem
- N-Process Critical Section Problem
- Conditions for a good solution:
 1. **Mutual Exclusion:** If a process is executing in its critical section, then no other processes can be executing in their critical sections.

7 August 2021

© Copyright Virtual University of
Pakistan



Solution to Critical-Section Problem

- 2. Progress:** If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can decide which process will enter its critical section next, and this decision cannot be postponed indefinitely.

7 August 2021

© Copyright Virtual University of
Pakistan



Solution to Critical-Section Problem

- 3. Bounded Waiting:** A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

7 August 2021

© Copyright Virtual University of
Pakistan



Solution to Critical-Section Problem

- Assumptions
 - Assume that each process executes at a nonzero speed
 - No assumption can be made regarding the relative speeds of the N processes.

7 August 2021

© Copyright Virtual University of
Pakistan



Possible Solutions

- Only 2 processes, P_0 and P_1
- Processes may share some common variables to synchronize their actions.
- General structure of process P_i

```
do {
```

```
    entry section
```

```
    critical section
```

```
    exit section
```

```
    remainder section
```

```
} while (1);
```

7 August 2021

© Copyright Virtual University of
Pakistan



Algorithm 1

- Shared variables:
 - int turn;
initially turn = 0
 - $\text{turn} = i \Rightarrow P_i$ can enter its critical section

7 August 2021

© Copyright Virtual University of
Pakistan



Algorithm 1

- Process P_i

```
do {
```

```
    while (turn != i)  
        ;
```

```
    critical section
```

```
    turn = j;
```

```
    remainder section
```

```
} while (1);
```

- Does not satisfy the progress condition

7 August 2021

© Copyright Virtual University of
Pakistan



Algorithm 2

Shared variables

- boolean flag[2]; // Set to false
- flag [i] = true $\Rightarrow$ P_i ready to enter its critical section

7 August 2021

© Copyright Virtual University of
Pakistan



Algorithm 2

■ Process P_i

do {

```
flag[i] = true;  
while (flag[j])  
;
```

critical section

```
flag[i] = false;
```

remainder section

} while (1);

- Does not satisfy the progress condition

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Process synchronization
- The Critical Section Problem
- Conditions for a Good Solution
- 2-Process Critical Section Problem Solutions
- Recap of lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 19

Syed Mansoor Sarwar



Operating Systems

Lecture 20

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- 2-Process Critical Section Problem (continued)
- n-Process Critical Section Problem
- The Bakery Algorithm
- Recap of lecture



Review of Lecture 19

- Process Synchronization
- The Critical Section Problem
- 2-Process Critical Section Problem Solutions

7 August 2021

© Copyright Virtual University of
Pakistan



Algorithm 3

- Combined shared variables of algorithms 1 and 2.
- `boolean flag[2]; // Set to false`
- `int turn=0;`

7 August 2021

© Copyright Virtual University of
Pakistan



Algorithm 3

- Process P_i

do {

```
flag[i] = true;  
turn = j;  
while (flag[j] && turn == j) ;
```

critical section

```
flag[i] = false;
```

remainder section

} while (1);

7 August 2021

© Copyright Virtual University of
Pakistan



Algorithm 3

- Meets all three requirements:
 - **Mutual Exclusion:** 'turn' can have one value at a given time (0 or 1)
 - **Bounded-waiting:** At most one entry by a process and then the second process enters into its CS

7 August 2021

© Copyright Virtual University of
Pakistan



Algorithm 3

- **Progress:** Exiting process sets its 'flag' to false ... comes back quickly and set it to true again ... but sets turn to the number of the other process

n-Process Critical Section Problem

- Consider a system of n processes ($P_0, P_1 \dots P_{n-1}$).
- Each process has a segment of code called a critical section in which the process may change shared data.

n-Process Critical Section Problem

- When one process is executing its critical section, no other process is allowed to execute in its critical section.
- The critical section problem is to design a protocol to serialize executions of critical sections.

7 August 2021

© Copyright Virtual University of
Pakistan



Bakery Algorithm

- By Leslie Lamport
- Before entering its critical section, process receives a ticket number. Holder of the smallest ticket number enters the critical section.
- If processes P_i and P_j receive the same number, if $i < j$, then P_i is served first; else P_j is served first.

7 August 2021

© Copyright Virtual University of
Pakistan



Bakery Algorithm

- The ticket numbering scheme always generates numbers in the increasing order of enumeration; i.e., 1, 2, 3, 4, 5 ...

7 August 2021

© Copyright Virtual University of
Pakistan



Bakery Algorithm

Notations

- (ticket #, process id #)

$(a,b) < (c,d)$ if $a < c$ or
if $a == c$ and $b < d$

- $\max(a_0, \dots, a_{n-1})$ is a number, k , such that $k \geq a_i$ for $i = 0, \dots, n-1$

7 August 2021

© Copyright Virtual University of
Pakistan



Bakery Algorithm

Data Structures

- boolean choosing[n];
- int number[n];

These data structures are initialized to false and 0, respectively

7 August 2021

© Copyright Virtual University of
Pakistan



Bakery Algorithm

Structure of P_i

```
do {  
    choosing[i] = true;  
    number[i] = max(number[0],  
                    number[1], ...,  
                    number [n - 1]) + 1;  
    choosing[i] = false;
```

Bakery Algorithm

```
for (j = 0; j < n; j++) {  
    while (choosing[j]) ;  
    while ( (number[j] != 0) &&  
        ((number[j], j) < (number[i], i)) ) ;  
}
```

Critical Section

7 August 2021

© Copyright Virtual University of
Pakistan



Bakery Algorithm

```
number[i] = 0;
```

remainder section

```
} while (1);
```

7 August 2021

© Copyright Virtual University of
Pakistan



Bakery Algorithm

Process	Number
P0	3
P1	0
P2	7
P3	4
P4	8

7 August 2021

© Copyright Virtual University of
Pakistan



Bakery Algorithm

P0	P2	P3	P4
$(3,0) < (3,0)$	$(3,0) < (7,2)$	$(3,0) < (4,3)$	$(3,0) < (8,4)$
Number[1] = 0	Number[1] = 0	Number[1] = 0	Number[1] = 0
$(7,2) < (3,0)$	$(7,2) < (7,2)$	$(7,2) < (4,3)$	$(7,2) < (8,4)$
$(4,3) < (3,0)$	$(4,3) < (7,2)$	$(4,3) < (4,3)$	$(4,3) < (8,4)$
$(8,4) < (3,0)$	$(8,4) < (7,2)$	$(8,4) < (4,3)$	$(8,4) < (8,4)$

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- n-Process Critical Section Problem
- The Bakery Algorithm

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 20

Syed Mansoor Sarwar



Operating Systems

Lecture 21

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- 2-Process CS problem solution
- N-Process CS problem
- The Bakery Algorithm
- Hardware solutions
- Recap of lecture



Review of Lecture 20

- 2-Process Critical Section Problem Solution
- n-Process Critical Section Problem
- The Bakery Algorithm

7 August 2021

© Copyright Virtual University of
Pakistan



Bakery Algorithm

Structure of P_i

```
do {  
    choosing[i] = true;  
    number[i] = max(number[0],  
                    number[1], ...,  
                    number [n - 1]) + 1;  
    choosing[i] = false;
```

Bakery Algorithm

```
for (j = 0; j < n; j++) {  
    while (choosing[j]) ;  
    while ( (number[j] != 0) &&  
        ((number[j], j) < (number[i], i)) ) ;  
}
```

Critical Section

7 August 2021

© Copyright Virtual University of
Pakistan



Bakery Algorithm

```
number[i] = 0;
```

remainder section

```
} while (1);
```

7 August 2021

© Copyright Virtual University of
Pakistan



Bakery Algorithm

Process	Number
P0	3
P1	0
P2	7
P3	4
P4	8

7 August 2021

© Copyright Virtual University of
Pakistan



Bakery Algorithm

P0	P2	P3	P4
$(3,0) < (3,0)$	$(3,0) < (7,2)$	$(3,0) < (4,3)$	$(3,0) < (8,4)$
Number[1] = 0	Number[1] = 0	Number[1] = 0	Number[1] = 0
$(7,2) < (3,0)$	$(7,2) < (7,2)$	$(7,2) < (4,3)$	$(7,2) < (8,4)$
$(4,3) < (3,0)$	$(4,3) < (7,2)$	$(4,3) < (4,3)$	$(4,3) < (8,4)$
$(8,4) < (3,0)$	$(8,4) < (7,2)$	$(8,4) < (4,3)$	$(8,4) < (8,4)$

7 August 2021

© Copyright Virtual University of
Pakistan



1

3

2

4

Bakery Algorithm

- P1 not interested to get into its critical section $\Rightarrow$ number[1] is 0
- P2, P3, and P4 wait for P0
- P0 gets into its CS, get out, and sets its number to 0
- P3 get into its CS and P2 and P4 wait for it to get out of its CS

7 August 2021

© Copyright Virtual University of
Pakistan



Bakery Algorithm

- P2 gets into its CS and P4 waits for it to get out
- P4 gets into its CS
- Sequence of execution of processes:

$\langle P0, P3, P2, P4 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Bakery Algorithm

- Meets all three requirements:
 - **Mutual Exclusion:**
(number[j], j) < (number[i], i) cannot be true for both P_i and P_j
 - **Bounded-waiting:** At most one entry by each process (n-1 processes) and then a requesting process enters its critical section (First-Come-First-Serve)

7 August 2021

© Copyright Virtual University of
Pakistan



Bakery Algorithm

- **Progress:**
 - Decision takes complete execution of the 'for loop' by one process
 - No process in its 'Remainder Section' (with its number set to 0) participates in the decision making

7 August 2021

© Copyright Virtual University of
Pakistan



Synchronization Hardware

- Normally, access to a memory location excludes other accesses to that same location.
- Extension: designers have proposed machine instructions that perform two operations atomically (indivisibly) on the same memory location (e.g., reading and writing).

7 August 2021

© Copyright Virtual University of
Pakistan



Synchronization Hardware

- The execution of such an instruction is also mutually exclusive (even on Multiprocessors).
- They can be used to provide mutual exclusion but other mechanisms are needed to satisfy the other two requirements of a good solution to the CS problem.

7 August 2021

© Copyright Virtual University of
Pakistan



Test-And-Set (TSL) Instruction

- Test and modify a word atomically.

```
boolean TestAndSet(boolean &target)
{
    boolean rv = target;
    target = true;
    return rv;
}
```

Virtual University

Solution with TSL

- Structure for Pi ('lock' is set to false)

```
do {
```

```
    while ( TestAndSet(lock) ) ;
```

Critical Section

```
    lock = false;
```

Remainder Section

```
}
```


Solution with TSL

- Is the TSL-based solution good?

No

- Mutual Exclusion: Satisfied
- Progress: Satisfied
- Bounded Waiting: Not satisfied

7 August 2021

© Copyright Virtual University of
Pakistan



Swap Instruction

- Swaps two variables atomically

```
void swap (boolean &a, boolean &b)
{
    boolean temp = a;
    a = b;
    b = temp;
}
```

Solution with Swap

- Structure for Pi
- 'key' is local and set to false

do {

key = true;

while (key == true) swap(lock, key);

Critical Section

lock = false;

Remainder Section

}

Solution with swap

- Is the swap-based solution good?

No

- Mutual Exclusion: Satisfied
- Progress: Satisfied
- Bounded Waiting: Not satisfied

Recap of Lecture

- The Bakery algorithm
- Synchronization hardware
- The **test-and-set** and **swap** instructions
- Solutions based on the **TSL** and **swap** instructions

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 21

Syed Mansoor Sarwar



Operating Systems

Lecture 22

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Semaphores based solutions for the critical section problem
- Deadlock and starvation
- Binary and counting semaphores
- Recap of lecture



Review of Lecture 21

- The Bakery algorithm
- Hardware based solutions

7 August 2021

© Copyright Virtual University of
Pakistan



A Good Solution

- 'key' local; 'lock' and 'waiting' global
- All variables set to false

do {

```
    waiting[i] = true;  
    key = true;  
    while (waiting[i] && key)  
        key = TestAndSet(lock);  
    waiting[i] = false;
```

Critical Section

A Good Solution

```
j = (i+1) % n;  
while ( (j != i) && !waiting[j] )  
    j = (j+1) % n;  
if (j == i)  
    lock = false;  
else  
    waiting[j] = false;
```

Remainder Section

}

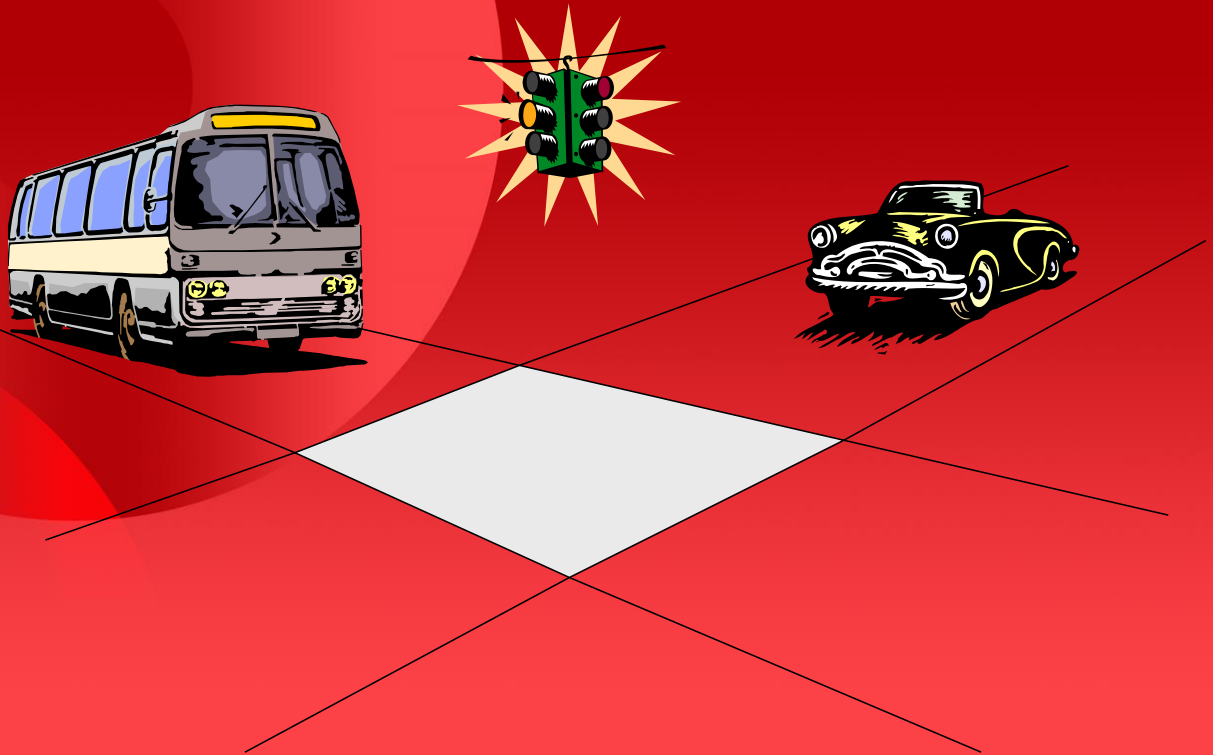
Solution with Test-And-Set

- Is the given solution good?

Yes

- Mutual Exclusion: Satisfied
- Progress: Satisfied
- Bounded Waiting: Satisfied

Semaphores



7 August 2021

© Copyright Virtual University of
Pakistan



Semaphores

- Synchronization tool
- Available in operating systems
- Semaphore S – integer variable that can only be accessed via two indivisible (atomic) operations, called `wait` and `signal`

7 August 2021

© Copyright Virtual University of
Pakistan



Semaphores

```
wait(S) {  
    while S ≤ 0  
        ; //no-op  
    S--;  
}
```

```
signal(S) {  
    S++;  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



n-Processes CS Problem

- Shared data:
semaphore mutex = 1;
- Structure of P_i:

```
do {  
    wait(mutex) ;  
    critical section  
    signal(mutex) ;  
    remainder section  
} while (1) ;
```

7 August 2021

© Copyright Virtual University of
Pakistan



Is it a Good Solution?

- **Mutual Exclusion:** Yes
- **Progress:** Yes
- **Bounded Waiting:** No

7 August 2021

© Copyright Virtual University of
Pakistan



Atomic Execution

- **Uni-Processor Environment**
 - Inhibit interrupts before executing code for `wait()` and `signal()`
- **Bus-based Multi-Processor Environment**
 - Lock the data bus
 - Use a software solution

7 August 2021

© Copyright Virtual University of
Pakistan



Busy Waiting

- Processes wait by executing CPU instructions
- **Problem?** Wasted CPU cycles
- **Solution?** Modify the definition of semaphore

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Synchronization hardware
- Semaphores
- Semaphore based solutions for the critical section problem
- Busy waiting and modified definition of semaphore

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 22

Syed Mansoor Sarwar



Operating Systems

Lecture 23

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Busy waiting
- New definition of semaphore
- Process synchronization
- Deadlock and starvation
- Binary and counting semaphores
- Recap of lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 22

- A good hardware based solution
- Semaphores
- Semaphore-based solution for the critical section problem
- Busy waiting

Busy Waiting

- Processes wait by executing CPU instructions
- **Problem?** Wasted CPU cycles
- **Solution?** Modify the definition of semaphore

7 August 2021

© Copyright Virtual University of
Pakistan



Semaphore Implementation

- Define a semaphore as a record

```
typedef struct {  
    int value;  
    struct process *L;  
} semaphore;
```

7 August 2021

© Copyright Virtual University of
Pakistan



Semaphore Implementation

- Assume two simple operations:
 - `block()` suspends the process that invokes it.
 - `wakeup(P)` resumes the execution of a blocked process P.

7 August 2021

© Copyright Virtual University of
Pakistan



Semaphore Implementation

- The negative value of `S.value` indicates the number of processes waiting for the semaphore
- A pointer in the PCB needed to maintain a queue of processes waiting for a semaphore

7 August 2021

© Copyright Virtual University of
Pakistan



Semaphore wait()

- Semaphore operations now defined as

```
void wait(semaphore S) {  
    S.value--;  
    if (S.value < 0) {  
        add this process to S.L;  
        block();  
    }  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Semaphore signal()

```
void signal(semaphore S) {  
    S.value++;  
    if (S.value <= 0) {  
        remove process P from S.L;  
        wakeup(P);  
    }  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Two Implementations

- Busy-waiting version is better when critical sections are small and queue-waiting version is better for long critical sections (when waiting is for longer periods of time).

7 August 2021

© Copyright Virtual University of
Pakistan



Process Synchronization

Execute statement B in P_j only
after statement A has been
executed in P_i

- Use semaphore s initialized to 0

P_i
:
:

A

signal(s)

P_j
:
:

wait(s)

B

7 August 2021

© Copyright Virtual University of
Pakistan



Process Synchronization

- Give a semaphore based solution for the following problem:

Statement S1 in P1 executes only after statement S2 in P2 has executed, and statement S2 in P2 should execute only after statement S3 in P3 has executed.

7 August 2021

© Copyright Virtual University of
Pakistan



Process Synchronization

P1

⋮

S1

⋮

P2

⋮

S2

⋮

P3

⋮

S3

⋮

7 August 2021

© Copyright Virtual University of
Pakistan



Solution

Semaphore A=0, B=0;

P1	P2	P3
:	:	:
wait(A)	wait(B)	S3
S1	S2	signal(B)
:	signal(A)	:
:	:	:

7 August 2021

© Copyright Virtual University of
Pakistan



Problems with Semaphores

- Semaphores provide a powerful tool for enforcing mutual exclusion and coordinating processes.
- But `wait(S)` and `signal(S)` are scattered among several processes. Hence, difficult to understand their effects.

7 August 2021

© Copyright Virtual University of
Pakistan



Problems with Semaphores

- Usage must be correct in all the processes.
- One bad (or malicious) process can fail the entire collection of processes.

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlocks and Starvation

- A set of processes are said to be in a deadlock state if every process is waiting for an event that can be caused only by another process in the set.
 - Traffic deadlocks
 - One-way bridge-crossing
- Starvation (infinite blocking) due to unavailability of resources

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock

P0

```
wait(S) ;  
wait(Q) ;  
:  
signal(S) ;  
signal(Q) ;
```

P1

```
wait(Q) ;  
wait(S) ;  
:  
signal(Q) ;  
signal(S) ;
```

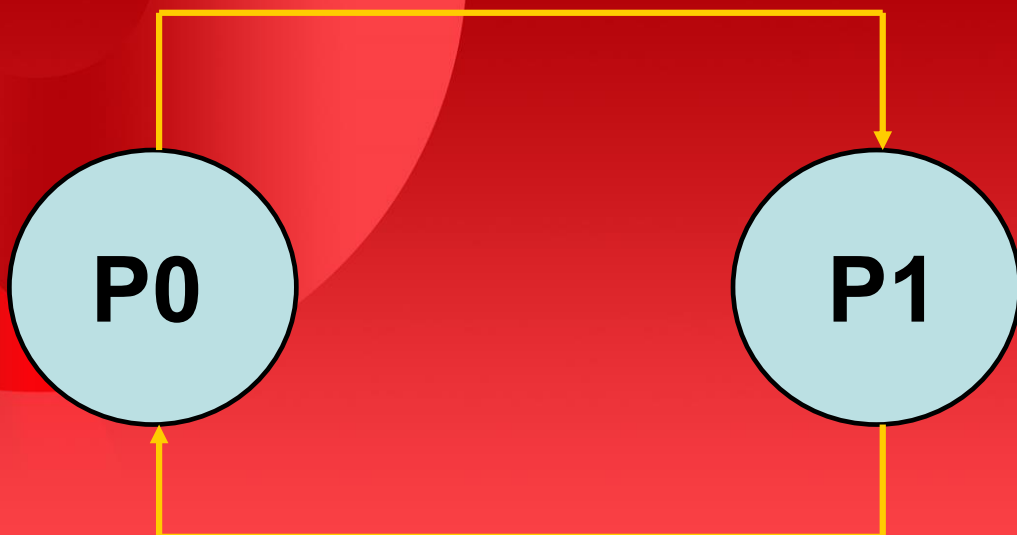
7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock

signal(S) ;



signal(Q) ;

7 August 2021

© Copyright Virtual University of
Pakistan



Starvation (Infinite Blocking)

P0

```
wait(S) ;  
:  
wait(S) ;
```

P1

```
wait(S) ;  
:  
signal(S) ;
```

7 August 2021

© Copyright Virtual University of
Pakistan



Violation of Mutual Exclusion

P0

```
signal(S) ;  
:  
wait(S) ;
```

P1

```
wait(S) ;  
:  
signal(S) ;
```

7 August 2021

© Copyright Virtual University of
Pakistan



Main Cause of Problem and Solution

- **Cause:** Programming errors due to the tandem use of `wait()` and `signal()` operations
- **Solution:** Higher-level language constructs such as critical region (region statement) and monitor.

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Busy waiting
- New definition of semaphore
- Process synchronization
- Deadlock and starvation
- Binary and counting semaphores

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 23

Syed Mansoor Sarwar



Operating Systems

Lecture 24

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Counting semaphores
- Classical synchronization problems
- Bounded buffer problem
- Readers and writers problem
- Dining philosophers problem
- Recap of lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 23

- Busy waiting
- New definition of semaphore
- Process synchronization
- Deadlock and starvation
- Binary and counting semaphores

7 August 2021

© Copyright Virtual University of
Pakistan



Types of Semaphores

- **Counting semaphore** – integer value can range over an unrestricted integer domain.
- **Binary semaphore** – integer value cannot be > 1 ; can be simpler to implement.

7 August 2021

© Copyright Virtual University of
Pakistan



Implementing a Counting Semaphore

- **Data structures**
binary-semaphore S1, S2;
int C;
- **Initialization**
 $S1 = 1$
 $S2 = 0$
 $C = \text{initial value of semaphore } S$

7 August 2021

© Copyright Virtual University of
Pakistan



Implementing a Counting Semaphore

wait(S):

```
wait(S1);  
C--;  
if (C < 0) {  
    signal(S1);  
    wait(S2);  
}  
signal(S1);
```

7 August 2021

© Copyright Virtual University of
Pakistan



Implementing a Counting Semaphore

signal(S):

```
wait(S1);  
C++;  
if (C <= 0)  
    signal(S2);  
else  
    signal(S1);
```

7 August 2021

© Copyright Virtual University of
Pakistan



Classical Problems of Synchronization

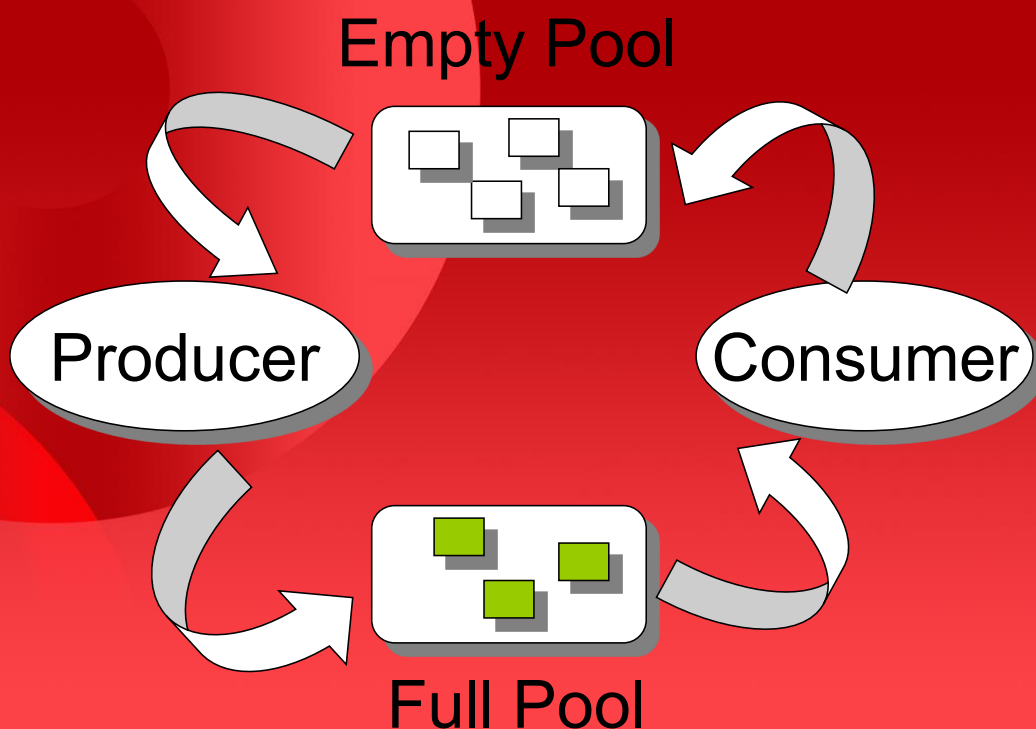
- Bounded-Buffer Problem
- Readers and Writers Problem
- Dining Philosophers Problem

7 August 2021

© Copyright Virtual University of
Pakistan



Bounded Buffer Problem



7 August 2021

© Copyright Virtual University of
Pakistan



Bounded-Buffer Problem

Shared data

semaphore full, empty, mutex;

Initialization

full = 0, empty = n, mutex = 1;

7 August 2021

© Copyright Virtual University of
Pakistan



Producer Process

```
do {  
    ...  
    produce an item in nextp  
    ...  
    wait(empty) ;  
    wait(mutex) ;  
    ...  
    add nextp to buffer  
    ...  
    signal(mutex) ;  
    signal(full) ;  
} while (1);
```

7 August 2021

© Copyright Virtual University of
Pakistan



Consumer Process

```
do {  
    wait(full)  
    wait(mutex) ;  
    ...  
    remove an item from buffer to nextc  
    ...  
    signal(mutex) ;  
    signal(empty) ;  
    ...  
    consume the item in nextc  
    ...  
} while (1);
```

7 August 2021

© Copyright Virtual University of
Pakistan



Readers and Writers Problem



7 August 2021

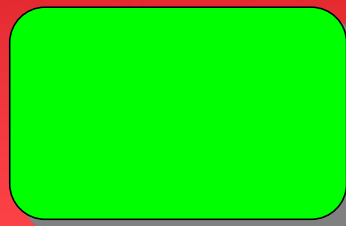
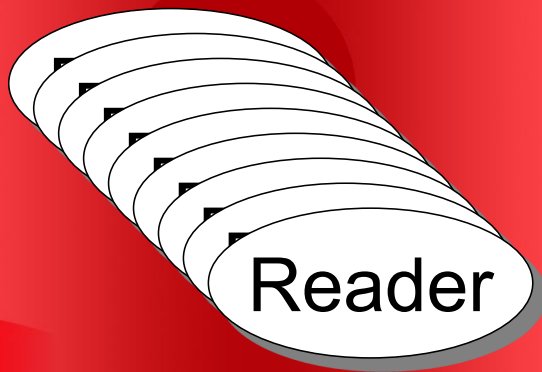
Readers

© Copyright Virtual University of
Pakistan

Writers



Readers and Writers Problem



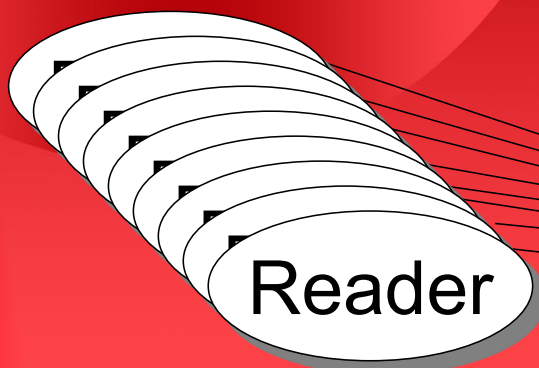
Shared Resource

7 August 2021

© Copyright Virtual University of
Pakistan



Readers and Writers Problem



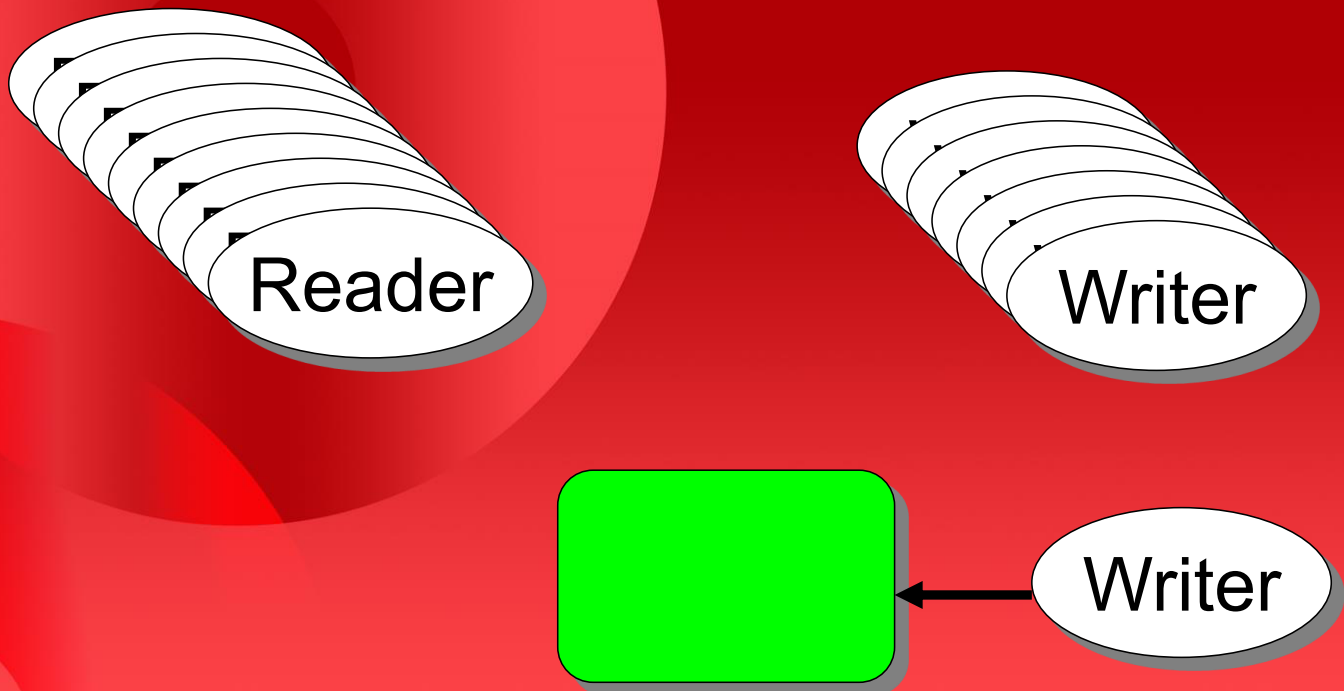
Shared Resource

7 August 2021

© Copyright Virtual University of
Pakistan



Readers and Writers Problem



7 August 2021

Shared Resource

© Copyright Virtual University of
Pakistan



First Readers and Writers Problem

No reader will be kept waiting unless a writer has already obtained permission to use the shared object.

7 August 2021

© Copyright Virtual University of
Pakistan



Second Readers and Writers Problem

If a writer is ready, it waits for the minimum amount of time.

7 August 2021

© Copyright Virtual University of
Pakistan



First Readers and Writers Problem

Shared data
semaphore mutex, wrt;

Initialization

mutex = 1, wrt = 1;
readcount = 0;

7 August 2021

© Copyright Virtual University of
Pakistan



Writer Process

```
wait(wrt) ;  
...  
writing is performed  
...  
signal(wrt) ;
```

7 August 2021

© Copyright Virtual University of
Pakistan



Reader Process

```
wait(mutex) ;  
readcount++ ;  
if (readcount == 1)  
    wait(wrt) ;  
signal(mutex) ;  
...  
reading is performed  
...  
wait(mutex) ;  
readcount-- ;  
if (readcount == 0)  
    signal(wrt) ;  
signal(mutex) ;
```

7 August 2021

© Copyright Virtual University of
Pakistan



Dining Philosophers Problem

- Five philosophers who spend their lives just thinking and eating.
- Only five chopsticks are available to the philosophers

7 August 2021

© Copyright Virtual University of
Pakistan



Dining Philosophers Problem

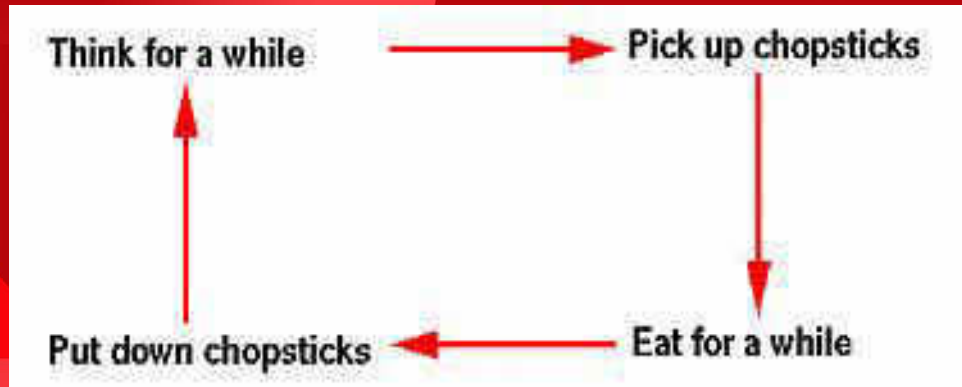
- Each philosopher thinks. When he becomes hungry, he sits down and picks up the two chopsticks that are closest to him and eats.
- After a philosopher finishes eating, he puts down the chopsticks and starts to think.

7 August 2021

© Copyright Virtual University of
Pakistan



Dining-Philosophers Problem



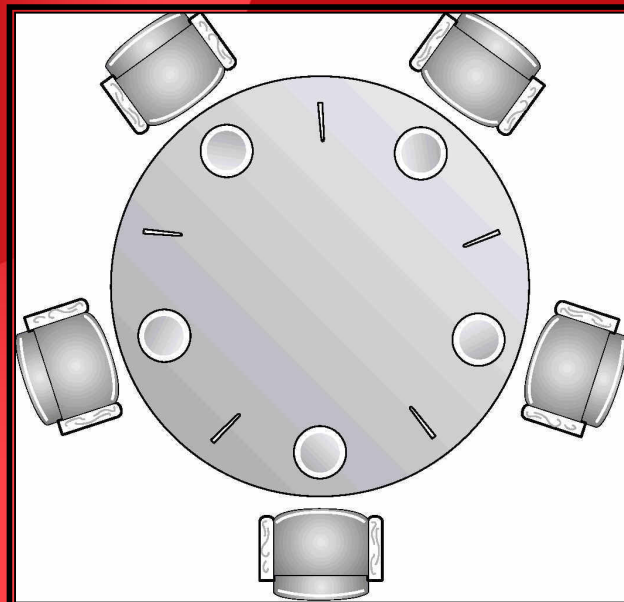
7 August 2021

© Copyright Virtual University of
Pakistan



Dining-Philosophers Problem

Shared data : semaphore chopstick[5];
// Initialized to 1



7 August 2021



Dining-Philosophers Problem

Philosopher i

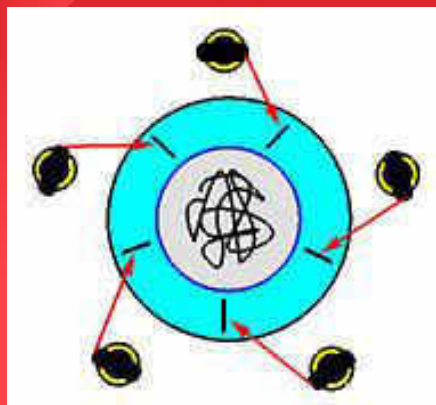
```
do {  
    wait(chopstick[i])  
    wait(chopstick[(i+1)% 5])  
    eat  
    signal(chopstick[i]);  
    signal(chopstick[(i+1)% 5])  
    think  
} while (1);
```

© Copyright Virtual University of
Pakistan



Possibility of Deadlock

If all philosophers become hungry at the same time and pick up their left chopstick, a deadlock occurs.



Recap of Lecture

- Counting semaphores
- Classical synchronization problems
- Bounded buffer problem
- Readers and writers problem
- Dining philosophers problem

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 24

Syed Mansoor Sarwar



Operating Systems

Lecture 25

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Dining philosophers problem
- High-level synchronization constructs
- Critical region
- Monitor
- Recap of lecture



Review of Lecture 24

- Counting semaphores
- Classical synchronization problems
- Bounded buffer problem
- Readers and writers problem
- Dining philosophers problem

7 August 2021

© Copyright Virtual University of
Pakistan



Dining Philosophers Problem

- Five philosophers who spend their lives just thinking and eating.
- Only five chopsticks are available to the philosophers

7 August 2021

© Copyright Virtual University of
Pakistan



Dining Philosophers Problem

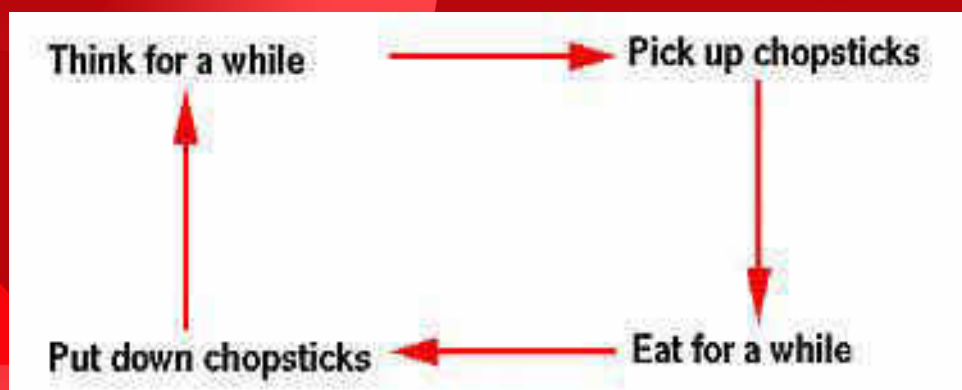
- Each philosopher thinks. When he becomes hungry, he sits down and picks up the two chopsticks that are closest to him and eats.
- After a philosopher finishes eating, he puts down the chopsticks and starts to think.

7 August 2021

© Copyright Virtual University of
Pakistan



Dining-Philosophers Problem



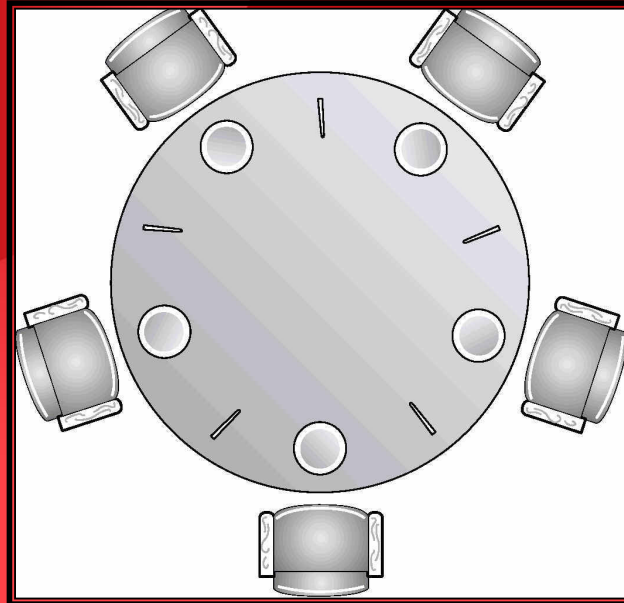
7 August 2021

© Copyright Virtual University of
Pakistan



Dining-Philosophers Problem

Shared data : semaphore chopstick[5];
// Initialized to 1



7 August 2021



Dining-Philosophers Problem

Philosopher i

do {

wait(chopstick[i])

wait(chopstick[(i+1)% 5])

eat

signal(chopstick[i]);

signal(chopstick[(i+1)% 5])

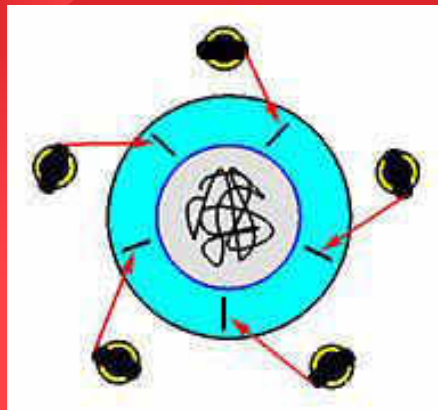
think

} while (1);



Possibility of Deadlock

If all philosophers become hungry at the same time and pick up their left chopstick, a deadlock occurs.



7 August 2021

VUI
Virtual University

Possible Solutions

- Allow at most four philosophers to be sitting simultaneously at the table.
- Allow a philosopher to pick up his/her chopsticks only if both chopsticks are available (to do this she must pick them up in a critical section)

7 August 2021

© Copyright Virtual University of
Pakistan

VUI
Virtual University

Possible Solutions

Use an asymmetric solution; that is, an odd philosopher picks up first her left chopstick, whereas an even philosopher picks up her right chopstick and then her left chopstick.

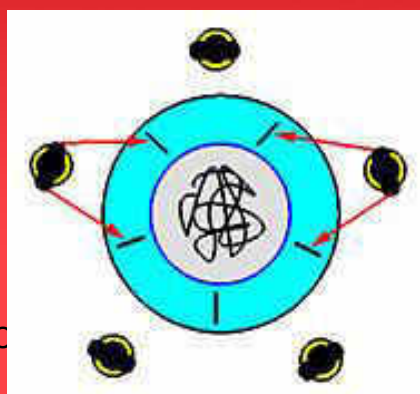
7 August 2021

© Copyright Virtual University of
Pakistan



Possibility of Starvation

- Two philosophers who are fast eaters and fast thinkers, and lock the chopsticks before others every time.



7 August 2021

© C



High-level Synchronization Constructs

Shift the responsibility of enforcing mutual exclusion from the programmer (where it resides when semaphores are used) to the compiler.

7 August 2021

© Copyright Virtual University of
Pakistan



Critical Regions

A **critical region** is a section of code that is always executed under mutual exclusion.

7 August 2021

© Copyright Virtual University of
Pakistan



Critical Regions

- They consist of two parts:
 1. Variables that must be accessed under mutual exclusion.
 2. A new language statement that identifies a critical region in which the variables are accessed.

7 August 2021

© Copyright Virtual University of
Pakistan



The region Statement

- A shared variable v of type T , is declared as:

v : shared T ;

region v when B do S ;

Variable v is accessed only inside S when B is true and when S is being executed, no other process can access variable v .

7 August 2021

© Copyright Virtual University of
Pakistan



The region Statement

- When a process tries to execute the region statement, the Boolean expression B is evaluated. If B is true, statement S is executed. If it is false, the process is delayed until B becomes true and no other process is in the region associated with v .

7 August 2021

© Copyright Virtual University of
Pakistan



Bounded Buffer Example

- Shared data

```
struct buffer {  
    int pool[n];  
    int count, in, out;  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Producer Process

- Producer process inserts `nextp` into the shared buffer

```
region buffer when (count < n) {  
    pool[in] = nextp;  
    in := (in+1) % n;  
    count++;  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Consumer Process

- Consumer process removes an item from the buffer and puts it in `nextc`

```
region buffer when (count > 0) {  
    nextc = pool[out];  
    out = (out+1) % n;  
    count--;  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



The region Statement

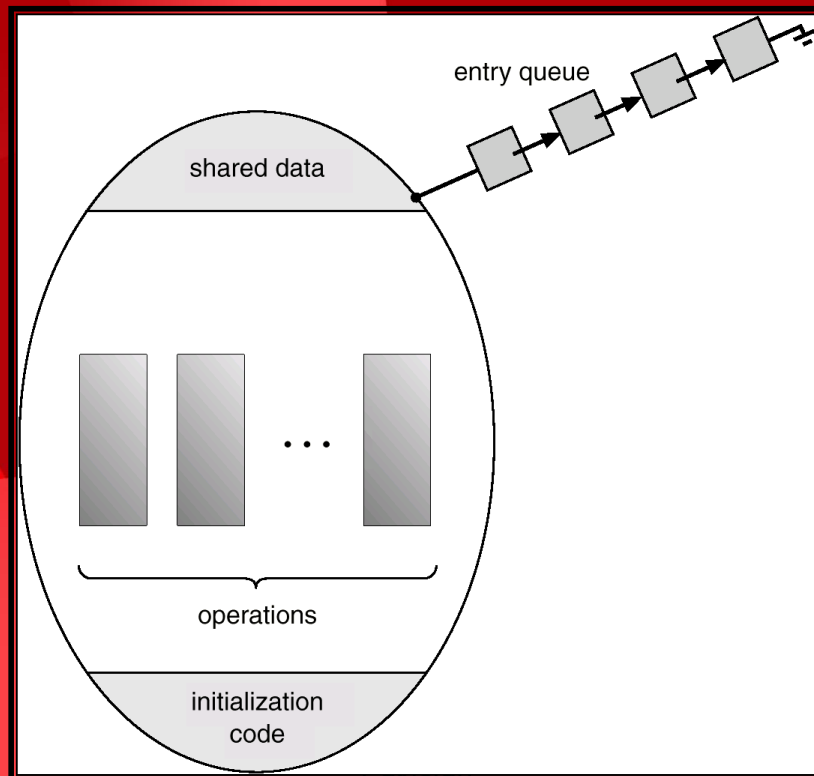
- What happens when two region statements are executed simultaneously?

```
region v when B do S1;  
region v when B do S2;
```

Monitors

- High-level synchronization construct that allows the safe sharing of an abstract data type among concurrent cooperating processes.
- Only one process at a time is active within a monitor.

Schematic View of a Monitor



7 August 2021

Monitors

```
monitor monitor-name
{
    shared variable declarations
    procedure P1 (...) { ... }
    procedure P2 (...) { ... }
    ...
    {
        initialization code
    }
}
```

7 August 2021

Monitors with Condition Variables

- Additional synchronization constructs are needed to model some synchronization problems. They can be modeled with **condition variables**.

`condition x, y;`

7 August 2021

© Copyright Virtual University of
Pakistan



Monitors with Condition Variables

- Only two operations can be performed on condition variables, `wait` and `signal`.
- `x.wait`: Process invoking this operation is suspended until another process invokes `x.signal`.

7 August 2021

© Copyright Virtual University of
Pakistan



Monitors with Condition Variables

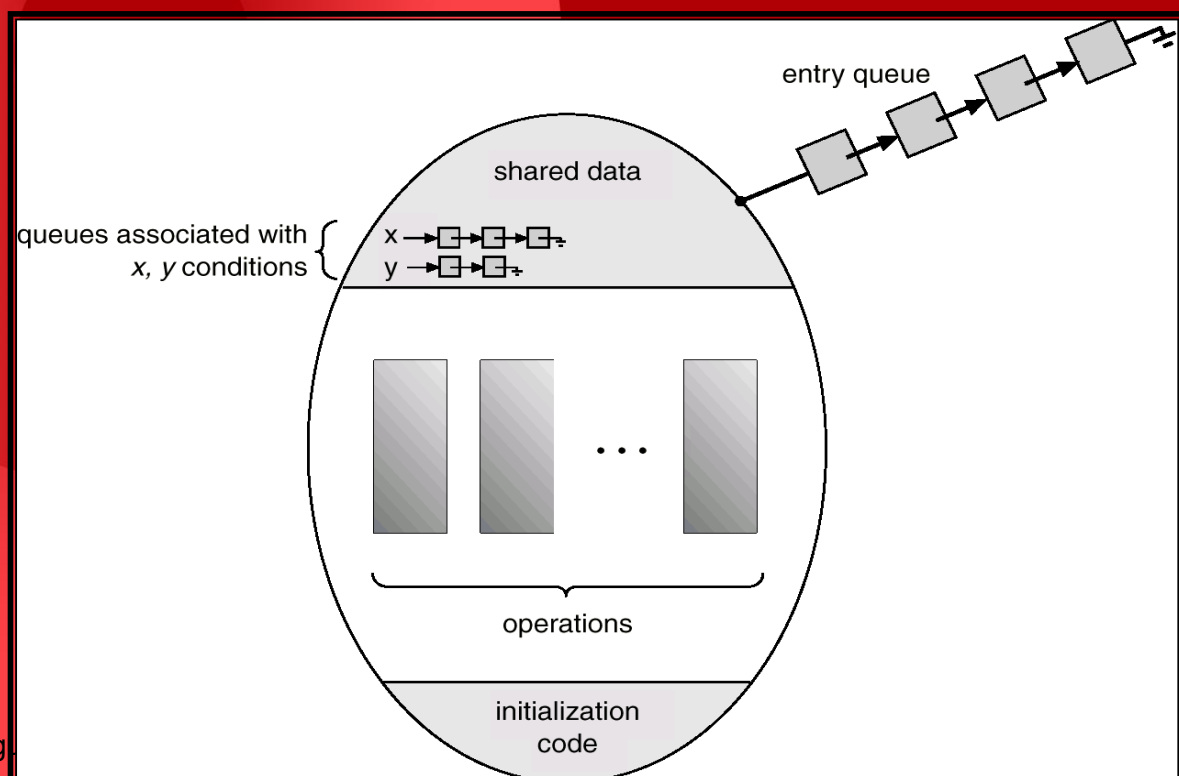
- `x.signal`: Resumes exactly one suspended process. If no process is suspended, this is a null operation, i.e., the state of `x` is unaffected.

7 August 2021

© Copyright Virtual University of Pakistan



Monitor with Condition Variables



7 Aug



Dining Philosophers Example

```
monitor dp
{
    enum {thinking, hungry, eating} state[5];
    condition self[5];
    void pickup(int i)    // Following slides
    void putdown(int i)  // Following slides
    void test(int i)     // Following slides
    void init() {
        for (int i = 0; i < 5; i++)
            state[i] = thinking;
    }
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Dining Philosophers

```
void pickup(int i) {
    state[i] = hungry;
    test(i);
    if (state[i] != eating)
        self[i].wait();
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Dining Philosophers

```
void putdown(int i) {  
    state[i] = thinking;  
    // test left and right  
    // neighbors  
    test((i+4) % 5);  
    test((i+1) % 5);  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Dining Philosophers

```
void test(int i) {  
    if ((state[(i+4)%5] != eating)  
        && (state[i] == hungry) &&  
        (state[(i+1)%5] != eating))  
    {  
        state[i] = eating;  
        self[i].signal();  
    }  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Dining philosophers problem
- High-level language constructs
 - Critical region
 - Monitor

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 25

Syed Mansoor Sarwar



Operating Systems

Lecture 26

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Monitors—Dining Philosophers
- The deadlock problem
- Deadlock characterization
- Deadlock handling
- Deadlock prevention
- Recap of lecture



Review of Lecture 25

- Dining philosophers problem
- High-level language constructs
 - Critical region
 - Monitor

7 August 2021

© Copyright Virtual University of
Pakistan



Dining Philosophers Example

```
monitor dp
{
    enum {thinking, hungry, eating} state[5];
    condition self[5];
    void pickup(int i)    // Following slides
    void putdown(int i)  // Following slides
    void test(int i)     // Following slides
    void init() {
        for (int i = 0; i < 5; i++)
            state[i] = thinking;
    }
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Dining Philosophers

```
void pickup(int i) {  
    state[i] = hungry;  
    test(i);  
    if (state[i] != eating)  
        self[i].wait();  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Dining Philosophers

```
void putdown(int i) {  
    state[i] = thinking;  
    // test left and right  
    // neighbors  
    test((i+4) % 5);  
    test((i+1) % 5);  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



Dining Philosophers

```
void test(int i) {  
    if ((state[(i+4)%5] != eating)  
        && (state[i] == hungry) &&  
        (state[(i+1)%5] != eating))  
    {  
        state[i] = eating;  
        self[i].signal();  
    }  
}
```

7 August 2021

© Copyright Virtual University of
Pakistan



The Deadlock Problem

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.

7 August 2021

© Copyright Virtual University of
Pakistan



The Deadlock Problem

- Example
 - System has 2 tape drives.
 - P1 and P2 each hold one tape drive and each needs another one.

7 August 2021

© Copyright Virtual University of
Pakistan



The Deadlock Problem

- Example: semaphores A and B, initialized to 1

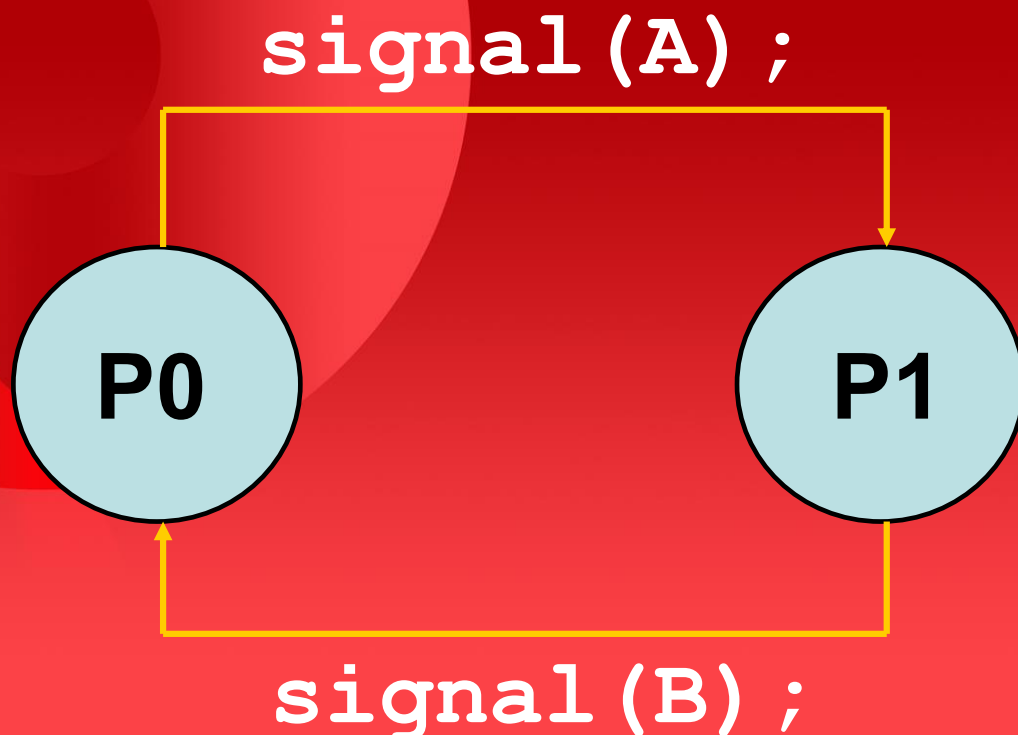
P0	P1
wait (A);	wait(B);
wait (B);	wait(A);

7 August 2021

© Copyright Virtual University of
Pakistan



The Deadlock Problem



7 August 2021

© Copyright Virtual University of
Pakistan



The Deadlock Problem

- The Dining Philosophers problem ... all philosophers become hungry at the same time, pick up the chopsticks on their right and wait for getting the chopsticks on their left.

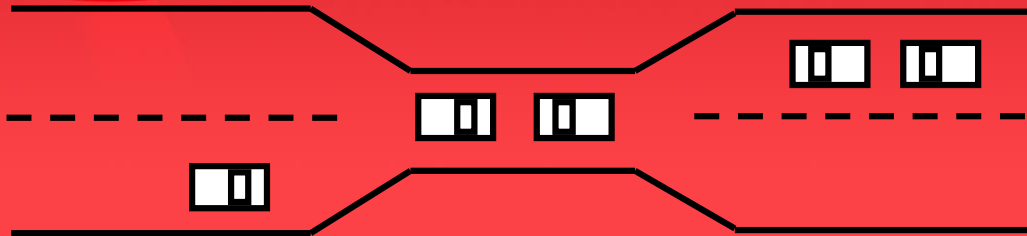
7 August 2021

© Copyright Virtual University of
Pakistan



Bridge Crossing Example

- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.



7 August 2021

© Copyright Virtual University of
Pakistan



Bridge Crossing Example

- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.

7 August 2021

© Copyright Virtual University of
Pakistan



System Model

- Resource types $R_1, R_2, \dots, R_m$ (CPU cycles, memory space, I/O devices)
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
request $\rightarrow$ use $\rightarrow$ release

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource.

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock Characterization

- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock Characterization

- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task.

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock

Characterization

- **Circular wait:** There exists a set $\{P_0, P_1, \dots, P_0\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2 , ..., P_{n-1} is waiting for a resource that is held by P_n , and P_0 is waiting for a resource that is held by P_0 .

$P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow \dots \rightarrow P_n \rightarrow P_0$

7 August 2021

© Copyright Virtual University of
Pakistan



Resource Allocation Graph

A set of vertices V and a set of edges E .

- V is partitioned into two types:
 - $P = \{P_1, P_2, \dots, P_n\}$
 - $R = \{R_1, R_2, \dots, R_m\}$
- $E = \{\text{Request Edges, Assignment Edges}\}$
 - **Request Edge:** $P_i \rightarrow R_j$
 - **Assignment Edge:** $R_j \rightarrow P_i$

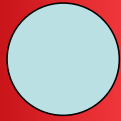
7 August 2021

© Copyright Virtual University of
Pakistan

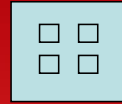


Resource Allocation Graph

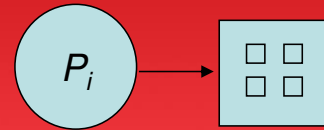
- Process



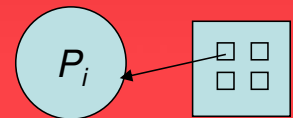
- Resource Type with 4 instances



- P_i requests instance of R_j



- P_i is holding an instance of R_j

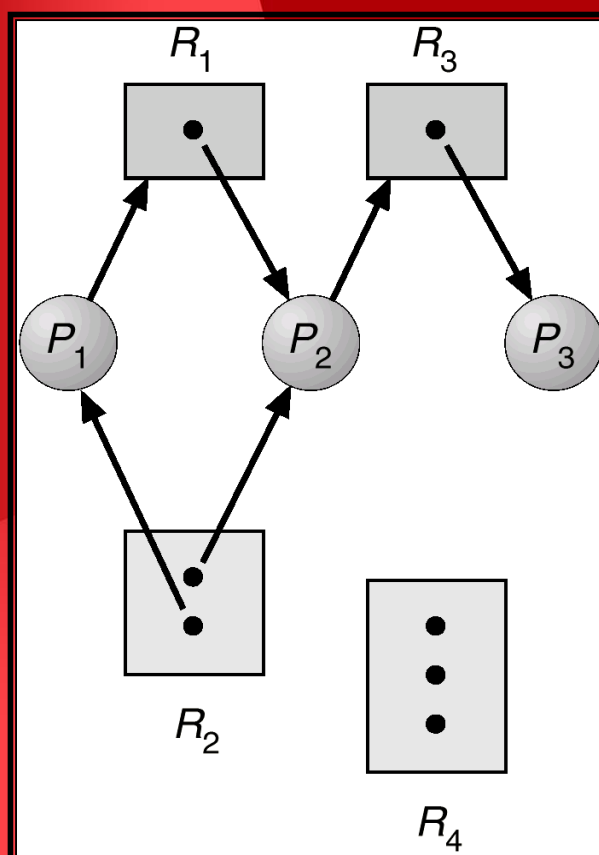


7 August 2021

© Copyright Virtual University of
Pakistan



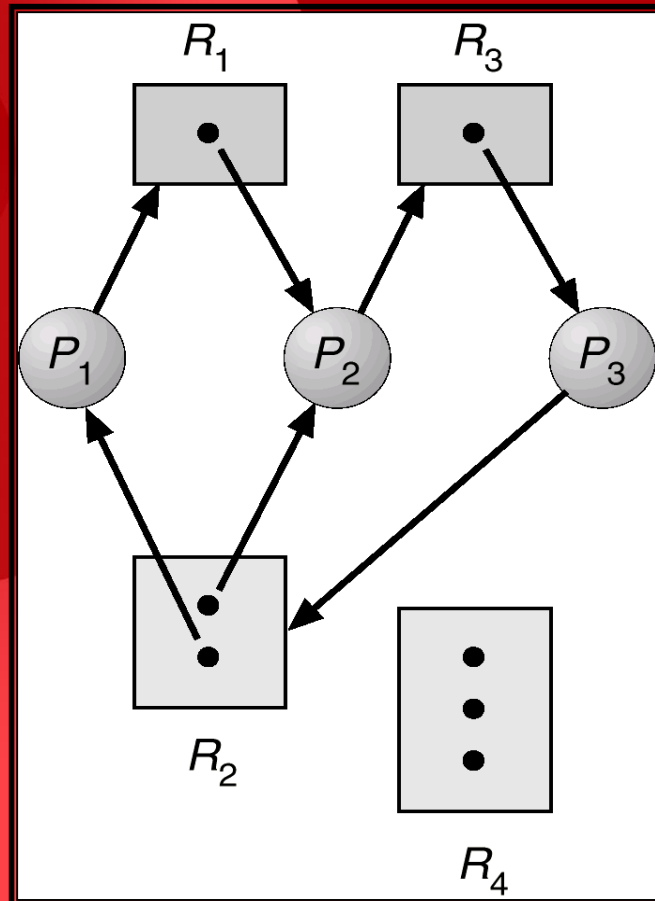
Example Graph



7 August 2021

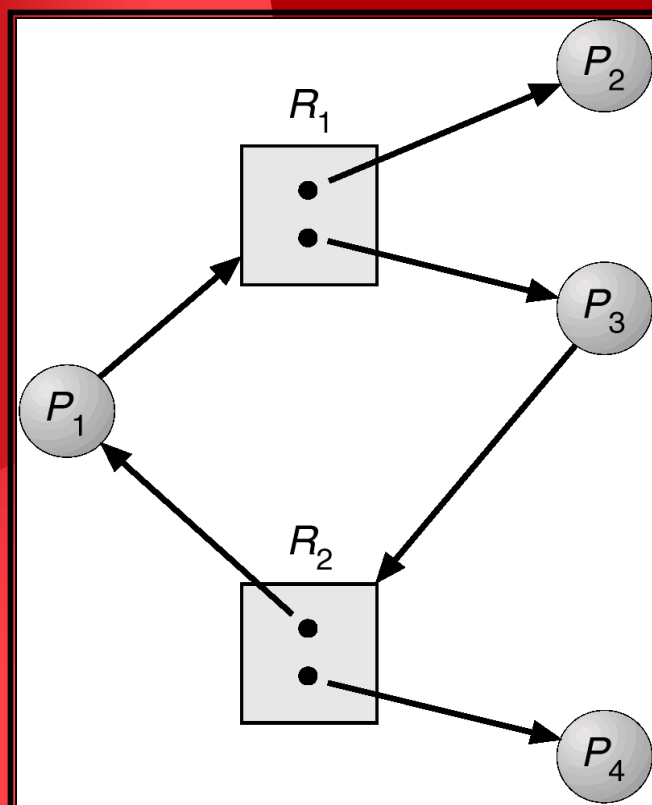


Graph with a Deadlock



7 August 2021

Graph with a Cycle but No Deadlock



7 August 2021

Basic Facts

- If graph contains no cycles $\Rightarrow$ no deadlock.
- If graph contains a cycle $\Rightarrow$
 - if only one instance per resource type, then deadlock.
 - if several instances per resource type, possibility of deadlock.

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Monitor-based implementation of the dining philosophers problem
- The deadlock problem
- The system model
- Deadlock characterization
- The resource allocation graph

August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 26

Syed Mansoor Sarwar



Operating Systems

Lecture 27

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Deadlock handling
- Deadlock prevention
- Deadlock avoidance
- Recap of lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 26

- Dining philosophers problem with monitor
- Deadlocks in computers
- Resource allocation graph
- Deadlock characterization

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock Handling

- Ensure that the system will never enter a deadlock state.
- Allow the system to enter a deadlock state and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system.

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock Handling

- Deadlock prevention
- Deadlock avoidance
- Deadlock detection and recovery

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock Prevention

- Restrain the ways resource allocation requests can be made to insure that at least one of the four necessary conditions is violated.

Deadlock Prevention

■ Mutual Exclusion

Cannot be prevented for all resources. Some resources are inherently non-sharable because their states cannot be saved and restored without ill effects, such as a printer.

Deadlock Prevention

- **Hold and Wait** – we must guarantee that whenever a process requests a resource, it does not hold any other resources.

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock Prevention

- Require a process to request and be allocated all its resources before it begins execution, or allow a process to request resources only when the process has none.
- Low resource utilization; starvation possible.

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock Prevention

- **No Preemption**

If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock Prevention

- Preempted resources are added to the list of resources for which the process is waiting for.
- Process will be restarted only when it can regain its old resources as well as the new ones that it has requested.

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock Prevention

- **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock Prevention

- We assign a unique number to each resource type by using function

$$F: R \rightarrow N$$

and make sure that processes request resources in an increasing order of enumeration.

- For example, tape drive = 1, disk drive = 5, and printer = 12.

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock Prevention

■ Proof

Let's assume that there is a cycle

$$P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow \dots \rightarrow P_k \rightarrow P_0$$
$$R_0 \quad R_1 \quad R_2 \quad \quad R_k \quad R_0$$
$$\Rightarrow F(R_0) < F(R_1) < \dots F(R_k) < F(R_0)$$
$$\Rightarrow F(R_0) < F(R_0), \text{ which is impossible}$$
$$\Rightarrow \text{There can be no circular wait.}$$

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock Avoidance

- Requires that the system has additional a priori information available about the use of resources by processes.
- Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock Avoidance

- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
- Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

7 August 2021

© Copyright Virtual University of
Pakistan



Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.
- System is in a safe state if there exists a safe sequence of all processes.

7 August 2021

© Copyright Virtual University of
Pakistan



Safe State

- Sequence $\langle P_1, P_2, \dots, P_n \rangle$ is safe if for each P_i , the resources that P_i can still request can be satisfied by the currently available resources, plus the resources held by all the P_j , with $j < i$. In other words, a safe sequence specifies the order in which processes can be finished.

7 August 2021

© Copyright Virtual University of
Pakistan



Safe State

- If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.
- When P_j are finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
- When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

7 August 2021

© Copyright Virtual University of
Pakistan



Basic Facts

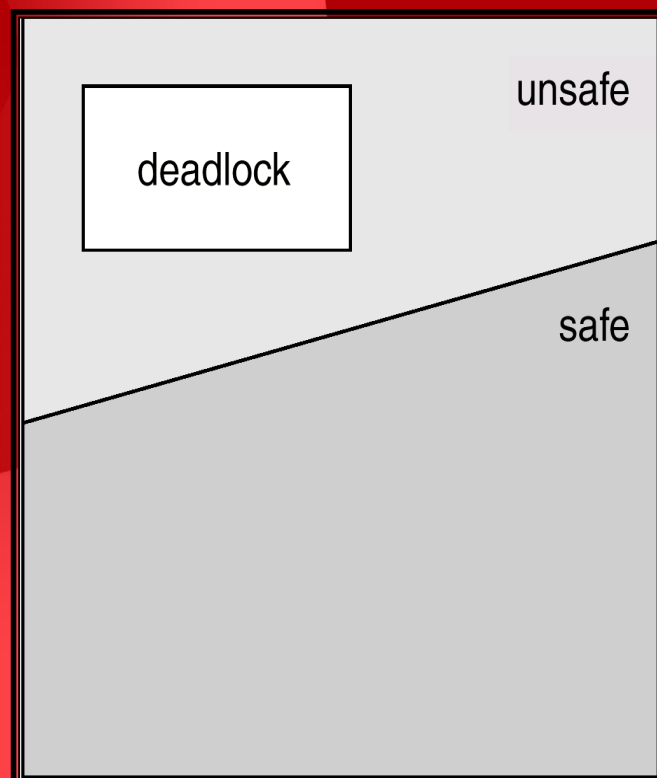
- If a system is in safe state $\Rightarrow$ no deadlocks.
- If a system is in unsafe state $\Rightarrow$ possibility of deadlock due to the behavior of processes.
- Avoidance $\Rightarrow$ ensure that a system never enters an unsafe state.

7 August 2021

© Copyright Virtual University of
Pakistan



Safe, Unsafe, and Deadlock States



7 August 2021



Example

- System with 12 tape drives and three processes
- Current system state:

Process	Max Need	Allocated
P ₀	10	5
P ₁	4	2
P ₂	9	2

7 August 2021

© Copyright Virtual University of
Pakistan



Example

- System is in a safe state with the safe sequence $\langle P_1, P_0, P_2 \rangle$
- P₂ requests and is allocated one more tape drive.

Process	Max Need	Allocated
P ₀	10	5
P ₁	4	2
P ₂	9	2

7 August 2021

© Copyright Virtual University of
Pakistan



Example

- Assuming the the tape drive is allocated to P_2 , the new system state will be:

Process	Max Need	Allocated
P_0	10	5
P_1	4	2
P_2	9	3

- System gets into an unsafe state

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Deadlock handling
- Deadlock prevention
- Deadlock avoidance

Operating Systems

Lecture 27

Syed Mansoor Sarwar



Operating Systems

Lecture 28

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Deadlock avoidance
- Banker's algorithms
- Safety algorithm
- Safe Sequence
- Recap of lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 27

- Deadlock handling
- Deadlock prevention
- Deadlock avoidance

7 August 2021

© Copyright Virtual University of
Pakistan



RAG Algorithm

- Claim edge $P_i \rightarrow R_j$ indicates that process P_i may request an instance of resource R_j ; represented by a dashed line.
- Claim edge converts to request edge when a process requests a resource.

7 August 2021

© Copyright Virtual University of
Pakistan



RAG Algorithm

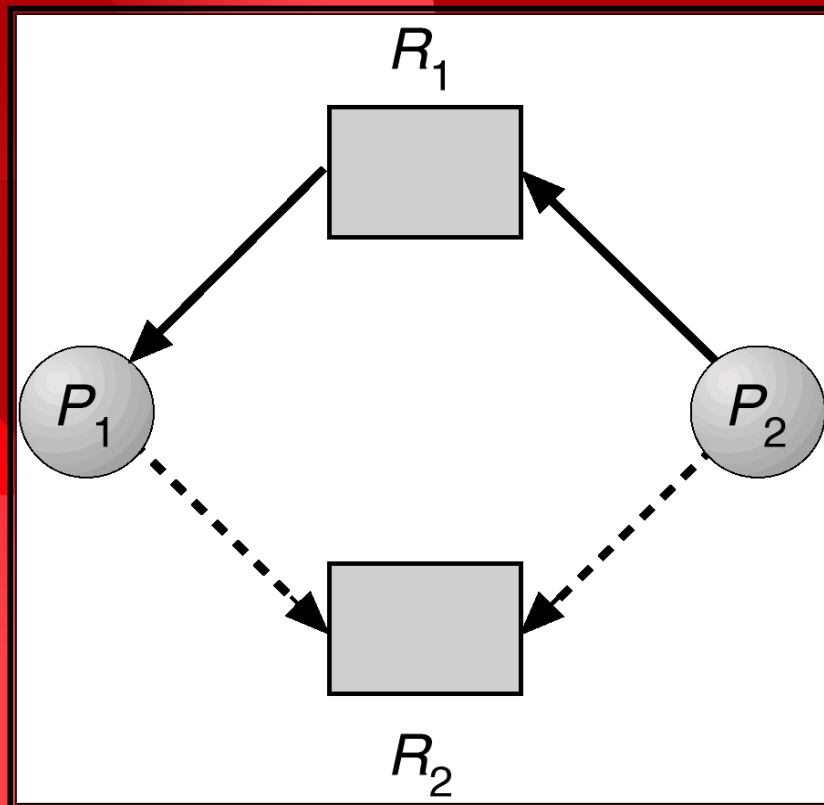
- When a resource is assigned to a process, request edge reconverts to an assignment edge.
- When a resource is released by a process, assignment edge reconverts to a claim edge.
- Resources must be claimed a priori in the system.

7 August 2021

© Copyright Virtual University of
Pakistan

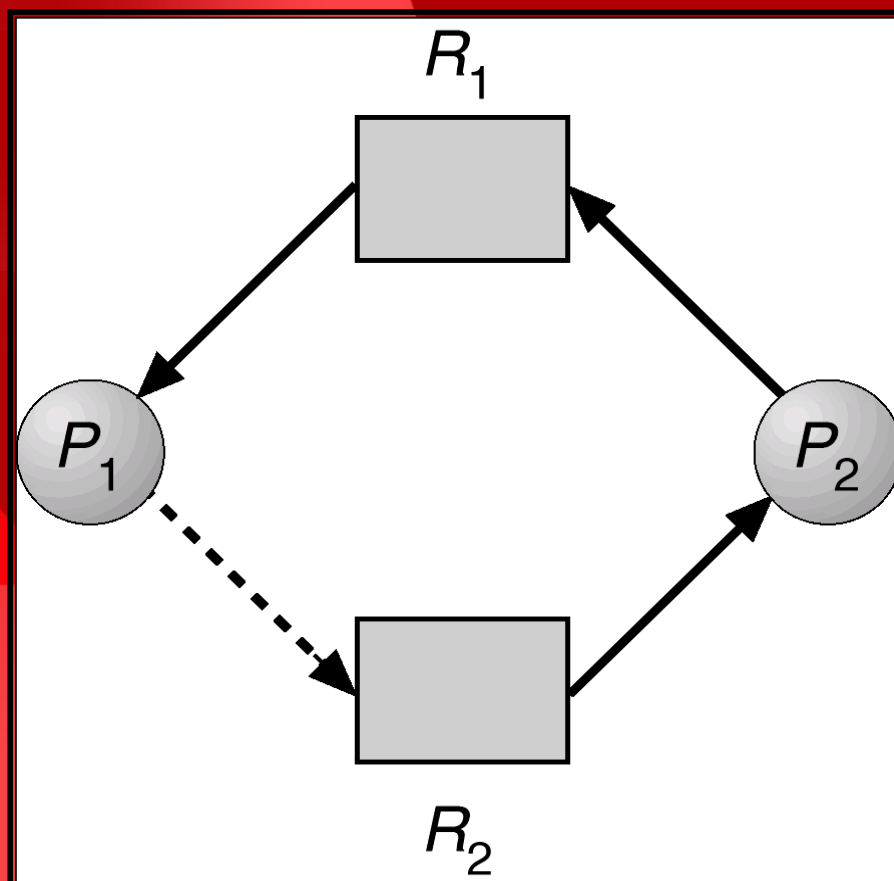


RAG For Deadlock Avoidance



7 August 2021

Unsafe State In RAG



7 August 2021

Banker's Algorithm

- Multiple instances of resources.
- Each process must a priori claim maximum use.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.

7 August 2021

© Copyright Virtual University of
Pakistan



Data Structures for the Banker's Algorithm

- n = number of processes
- m = number of resource types
- **Available:** Vector of length m .
If $\text{available}[j] = k$, there are k instances of resource type R_j available.

7 August 2021

© Copyright Virtual University of
Pakistan



Data Structures for the Banker's Algorithm

- **Max:** $n \times m$ matrix. If $\text{Max}[i,j] = k$, then process P_i may request at most k instances of resource type R_j .
- **Allocation:** $n \times m$ matrix. If $\text{Allocation}[i,j] = k$ then P_i is currently allocated k instances of R_j .

7 August 2021

© Copyright Virtual University of
Pakistan



Data Structures for the Banker's Algorithm

- **Need:** $n \times m$ matrix. If $\text{Need}[i,j] = k$, then P_i may need k more instances of R_j to complete its task.

$\text{Need}[i,j] =$

$\text{Max}[i,j] - \text{Allocation}[i,j]$

7 August 2021

© Copyright Virtual University of
Pakistan



Safety Algorithm

1. Let **Work** and **Finish** be vectors of length m and n , respectively. Initialize:
Work = Available
Finish[i] = false; $i = 1, 2, \dots, n$.

7 August 2021

© Copyright Virtual University of
Pakistan



Safety Algorithm

2. Find and i such that both:
 - (a) Finish[i] = false
 - (b) $Need_i \leq Work$If no such i exists, go to step 4.

7 August 2021

© Copyright Virtual University of
Pakistan



Safety Algorithm

3. $Work = Work + Allocation_i$
 $Finish[i] = true$
go to step 2.
4. If $Finish[i] == true$ for all i ,
then the system is in a safe state.

7 August 2021

© Copyright Virtual University of
Pakistan



Resource-Request Algorithm for P_i

- $Request_i$ = request vector for P_i .
 - If $Request_i[j] = k$ then process P_i wants k instances of resource type R_j .
1. If $Request_i \leq Need_i$ go to step 2.
Otherwise, report error since process has exceeded its maximum claim.

7 August 2021

© Copyright Virtual University of
Pakistan



Resource-Request Algorithm for P_i

2. If $\text{Request}_i \leq \text{Available}$, go to step 3. Otherwise P_i must wait, since resources are not available.
3. Pretend to allocate requested resources to P_i by modifying the state as follows:

7 August 2021

© Copyright Virtual University of
Pakistan



Resource-Request Algorithm for P_i

$\text{Available} = \text{Available} - \text{Request}_i;$
 $\text{Allocation}_i = \text{Allocation}_i + \text{Request}_i;$
 $\text{Need}_i = \text{Need}_i - \text{Request}_i;$

- If safe $\Rightarrow$ the resources are allocated to P_i .
- If unsafe $\Rightarrow P_i$ must wait, and the old resource-allocation state is restored

7 August 2021

© Copyright Virtual University of
Pakistan



Example System

- Five processes: $P_0 \dots P_4$
- Three resource types:
 - A (10 instances)
 - B (5 instances)
 - C (7 instances)

7 August 2021

© Copyright Virtual University of
Pakistan



Example System

System state: Is it safe?

Process	Max			Allocation			Available		
	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	3	3	2
P_1	3	2	2	2	0	0			
P_2	9	0	2	3	0	2			
P_3	2	2	2	2	1	1			
P_4	4	3	3	0	0	2			

7 August 2021

© Copyright Virtual University of
Pakistan



Example System

- Need_i = Max_i – Allocation_i
- Need matrix

Process	A	B	C
P ₀	7	4	3
P ₁	1	2	2
P ₂	6	0	0
P ₃	0	1	1
P ₄	4	3	1

7 August 2021

© Copyright Virtual University of Pakistan



Example System

Process	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P ₀	7	4	3	0	1	0	3	3	2
P ₁	1	2	2	2	0	0			
P ₂	6	0	0	3	0	2			
P ₃	0	1	1	2	1	1			
P ₄	4	3	1	0	0	2			

- Safe Sequence: <>

7 August 2021

© Copyright Virtual University of Pakistan



Example System

Process	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P ₀	7	4	3	0	1	0	3	3	2
P ₁	1	2	2	2	0	0	5	3	2
P ₂	6	0	0	3	0	2			
P ₃	0	1	1	2	1	1			
P ₄	4	3	1	0	0	2			

■ Safe Sequence: $\langle P_1 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example System

Process	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P ₀	7	4	3	0	1	0	3	3	2
P ₁	1	2	2	2	0	0	5	3	2
P ₂	6	0	0	3	0	2	7	4	3
P ₃	0	1	1	2	1	1			
P ₄	4	3	1	0	0	2			

■ Safe Sequence: $\langle P_1, P_3 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example System

Process	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P ₀	7	4	3	0	1	0	3	3	2
P ₁	1	2	2	2	0	0	5	3	2
P ₂	6	0	0	3	0	2	7	4	3
P ₃	0	1	1	2	1	1	7	4	5
P ₄	4	3	1	0	0	2			

■ Safe Sequence: $\langle P_1, P_3, P_4 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example System

Process	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P ₀	7	4	3	0	1	0	3	3	2
P ₁	1	2	2	2	0	0	5	3	2
P ₂	6	0	0	3	0	2	7	4	3
P ₃	0	1	1	2	1	1	7	4	5
P ₄	4	3	1	0	0	2	7	5	5

■ Safe Sequence: $\langle P_1, P_3, P_4, P_0 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example System

- Final safe sequence:

$\langle P_1, P_3, P_4, P_0, P_2 \rangle$

- Not a unique sequence
- Possible safe sequences for the this example:

$\langle P_1, P_3, P_4, P_0, P_2 \rangle$, $\langle P_1, P_3, P_4, P_2, P_0 \rangle$,
 $\langle P_1, P_3, P_2, P_0, P_4 \rangle$, $\langle P_1, P_3, P_2, P_4, P_0 \rangle$,
 $\langle P_1, P_3, P_0, P_2, P_4 \rangle$, $\langle P_1, P_3, P_0, P_4, P_2 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example

- P_1 requests (1,0,2). Can this request be granted immediately?

1. Is $\text{Request}_1 \leq \text{Need}_1$?

$(1,0,2) \leq (1,2,2) \Rightarrow \text{true}$

2. Is $\text{Request}_1 \leq \text{Available}$?

$(1,0,2) \leq (3,3,2) \Rightarrow \text{true}$

7 August 2021

© Copyright Virtual University of
Pakistan



Example

Process	Need			Allocation			Available		
	A	B	C	A	B	C	A	B	C
P ₀	7	4	3	0	1	0	3	3	2
P ₁	1	2	2	2	0	0			
P ₂	6	0	0	3	0	2			
P ₃	0	1	1	2	1	1			
P ₄	4	3	1	0	0	2			

7 August 2021

© Copyright Virtual University of Pakistan



Example

Process	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P ₀	7	4	3	0	1	0	2	3	0
P ₁	0	2	0	3	0	2			
P ₂	6	0	0	3	0	2			
P ₃	0	1	1	2	1	1			
P ₄	4	3	1	0	0	2			

■ Safe Sequence: <>

7 August 2021

© Copyright Virtual University of Pakistan



Example

Process	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P ₀	7	4	3	0	1	0	2	3	0
P ₁	0	2	0	3	0	2	5	3	2
P ₂	6	0	0	3	0	2			
P ₃	0	1	1	2	1	1			
P ₄	4	3	1	0	0	2			

■ Safe Sequence: $\langle P_1 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example

Process	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P ₀	7	4	3	0	1	0	2	3	0
P ₁	0	2	0	3	0	2	5	3	2
P ₂	6	0	0	3	0	2	7	4	3
P ₃	0	1	1	2	1	1			
P ₄	4	3	1	0	0	2			

■ Safe Sequence: $\langle P_1, P_3 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example

Process	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P ₀	7	4	3	0	1	0	2	3	0
P ₁	0	2	0	3	0	2	5	3	2
P ₂	6	0	0	3	0	2	7	4	3
P ₃	0	1	1	2	1	1	7	4	5
P ₄	4	3	1	0	0	2			

■ Safe Sequence: $\langle P_1, P_3, P_4 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example

Process	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P ₀	7	4	3	0	1	0	2	3	0
P ₁	0	2	0	3	0	2	5	3	2
P ₂	6	0	0	3	0	2	7	4	3
P ₃	0	1	1	2	1	1	7	4	5
P ₄	4	3	1	0	0	2	7	4	5

■ Safe Sequence: $\langle P_1, P_3, P_4, P_0 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example

- Executing safety algorithm shows that sequence $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ satisfies safety requirement.
- Yes, P_1 's request may be granted immediately.

More Examples

- Can request for (0,2,0) by P_0 be granted?
 1. Is $\text{Request}_0 \leq \text{Need}_0$?
 $(0,2,0) \leq (7,4,3) \Rightarrow \text{true}$
 2. Is $\text{Request}_1 \leq \text{Available}$?
 $(0,2,0) \leq (3,3,2) \Rightarrow \text{true}$

Example

Process	Need			Allocation			Available		
	A	B	C	A	B	C	A	B	C
P ₀	7	4	3	0	1	0	3	3	2
P ₁	1	2	2	2	0	0			
P ₂	6	0	0	3	0	2			
P ₃	0	1	1	2	1	1			
P ₄	4	3	1	0	0	2			

7 August 2021

© Copyright Virtual University of Pakistan



Example

Process	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P ₀	7	2	3	0	3	0	3	1	2
P ₁	1	2	2	2	0	0			
P ₂	6	0	0	3	0	2			
P ₃	0	1	1	2	1	1			
P ₄	4	3	1	0	0	2			

■ Safe Sequence: <>

7 August 2021

© Copyright Virtual University of Pakistan



Example

Process	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P ₀	7	2	3	0	3	0	3	1	2
P ₁	1	2	2	2	0	0	5	2	3
P ₂	6	0	0	3	0	2			
P ₃	0	1	1	2	1	1			
P ₄	4	3	1	0	0	2			

■ Safe Sequence: < P₃ >

7 August 2021

© Copyright Virtual University of
Pakistan



Example

Process	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P ₀	7	2	3	0	3	0	3	1	2
P ₁	1	2	2	2	0	0	5	2	3
P ₂	6	0	0	3	0	2	7	2	3
P ₃	0	1	1	2	1	1			
P ₄	4	3	1	0	0	2			

■ Safe Sequence: < P₃, P₁ >

7 August 2021

© Copyright Virtual University of
Pakistan



Example

Process	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P ₀	7	2	3	0	3	0	3	1	2
P ₁	1	2	2	2	0	0	5	2	3
P ₂	6	0	0	3	0	2	7	2	3
P ₃	0	1	1	2	1	1	10	2	5
P ₄	4	3	1	0	0	2			

■ Safe Sequence: $\langle P_3, P_1, P_2 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example

Process	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P ₀	7	2	3	0	3	0	3	1	2
P ₁	1	2	2	2	0	0	5	2	3
P ₂	6	0	0	3	0	2	7	2	3
P ₃	0	1	1	2	1	1	10	2	5
P ₄	4	3	1	0	0	2	10	5	5

■ Safe Sequence: $\langle P_3, P_1, P_2, P_0, P_4 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example

- Executing safety algorithm shows that sequence $\langle P_3, P_1, P_2, P_0, P_4 \rangle$ satisfies safety requirement.
- Yes, P_0 's request may be granted immediately.

More Examples

- Can request for (3,3,0) by P_4 be granted?
- Do it yourself!

Recap of Lecture

- Deadlock avoidance
- Banker's algorithm
- Safety algorithm
- Safe state
- Safe sequence

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 28

Syed Mansoor Sarwar



Operating Systems

Lecture 29

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Deadlock detection: resources with single and multiple instances
- Recovery from deadlocks
 - Process termination
 - Resource preemption
- Recap of lecture



Review of Lecture 28

- Deadlock avoidance
- Resource with single instances
- Resources with multiple instances: Banker's algorithm

7 August 2021

© Copyright Virtual University of
Pakistan



Deadlock Detection

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

7 August 2021

© Copyright Virtual University of
Pakistan



Single Instance of Each Resource Type

- Maintain a wait-for graph
 - Nodes are processes.
 - $P_i \rightarrow P_j$ if P_i is waiting for P_j .

7 August 2021

© Copyright Virtual University of
Pakistan



Single Instance of Each Resource Type

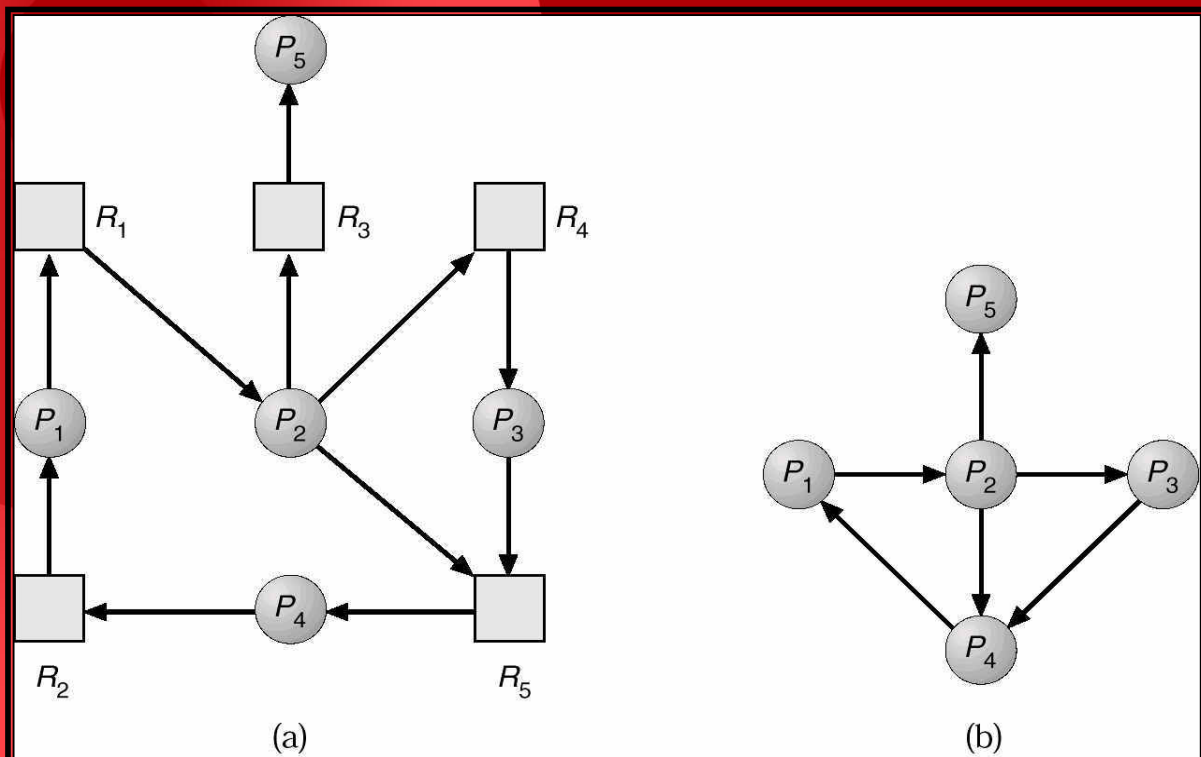
- Periodically invoke an algorithm that searches for a cycle in the wait-for graph.
- The algorithm is expensive—it requires an order of n^2 operations, where n is the number of vertices in the graph.

7 August 2021

© Copyright Virtual University of
Pakistan



RAG and Wait-for Graph



7 August 2021

© Copyright Virtual University of Pakistan

Corresponding wait-for graph



Multiple Instance of Each Resource Type

- **Available:** A vector of length m indicates the number of available resources of each type.
- **Allocation:** An $n \times m$ matrix defines the number of resources of each type currently allocated to each process.

7 August 2021

© Copyright Virtual University of Pakistan



Multiple Instance of Each Resource Type

- **Request:** An $n \times m$ matrix indicates the current request of each process. If $\text{Request}[i,j]=k$, then process P_i is requesting k more instances of resource type. P_j .

7 August 2021

© Copyright Virtual University of
Pakistan



Detection Algorithm

1. Let Work and Finish be vectors of length m and n , respectively
Initialize:

- $\text{Work} = \text{Available};$
- for $i = 1, 2, \dots, n$
if $\text{Allocation}_i \neq 0$
then
 $\text{Finish}[i] = \text{false};$
else
 $\text{Finish}[i] = \text{true};$

7 August 2021

© Copyright Virtual University of
Pakistan



Detection Algorithm

2. Find an index i such that both:

(a) $\text{Finish}[i] == \text{false}$

(b) $\text{Request}_i \leq \text{Work}$

If no such i exists, go to step 4.

7 August 2021

© Copyright Virtual University of
Pakistan



Detection Algorithm

3. $\text{Work} = \text{Work} + \text{Allocation}_i$
 $\text{Finish}[i] = \text{true}$
go to step 2.

4. If $\text{Finish}[i] == \text{false}$, for some i , $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if $\text{Finish}[i] == \text{false}$, then P_i is deadlocked.

7 August 2021

© Copyright Virtual University of
Pakistan



Example of Detection Algorithm

- Consider the following system:

$$P = \{ P_0, P_1, P_2, P_3, P_4 \}$$

$$R = \{ A, B, C \}$$

A: 7 instances

B: 2 instances

C: 6 instances

7 August 2021

© Copyright Virtual University of
Pakistan



Example System State

Process	Allocation			Request			Work		
	A	B	C	A	B	C	A	B	C
P ₀	0	1	0	0	0	0	0	0	0
P ₁	2	0	0	2	0	2			
P ₂	3	0	2	0	0	0			
P ₃	2	1	1	1	0	0			
P ₄	0	0	2	0	0	2			

- Finish Sequence: <>

7 August 2021

© Copyright Virtual University of
Pakistan



Example System State

Process	Allocation	Request	Work		
	A B C	A B C	A	B	C
P ₀	0 1 0	0 0 0	0	0	0
P ₁	2 0 0	2 0 2	0	1	0
P ₂	3 0 2	0 0 0			
P ₃	2 1 1	1 0 0			
P ₄	0 0 2	0 0 2			

■ Finish Sequence: $\langle P_0 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example System State

Process	Allocation	Request	Work		
	A B C	A B C	A	B	C
P ₀	0 1 0	0 0 0	0	0	0
P ₁	2 0 0	2 0 2	0	1	0
P ₂	3 0 2	0 0 0	3	1	2
P ₃	2 1 1	1 0 0			
P ₄	0 0 2	0 0 2			

■ Finish Sequence: $\langle P_0, P_2 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example System State

Process	Allocation	Request	Work		
	A B C	A B C	A	B	C
P ₀	0 1 0	0 0 0	0	0	0
P ₁	2 0 0	2 0 2	0	1	0
P ₂	3 0 2	0 0 0	3	1	2
P ₃	2 1 1	1 0 0	5	2	3
P ₄	0 0 2	0 0 2			

■ Finish Sequence: $\langle P_0, P_2, P_3 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example System State

Process	Allocation	Request	Work		
	A B C	A B C	A	B	C
P ₀	0 1 0	0 0 0	0	0	0
P ₁	2 0 0	2 0 2	0	1	0
P ₂	3 0 2	0 0 0	3	1	2
P ₃	2 1 1	1 0 0	5	2	3
P ₄	0 0 2	0 0 2	5	2	5

■ Finish Sequence: $\langle P_0, P_2, P_3, P_4 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example System State

- Final finish sequence:

$\langle P_0, P_2, P_3, P_4, P_1 \rangle$

Other possible finish sequences:

$\langle P_0, P_2, P_3, P_1, P_4 \rangle$

$\langle P_0, P_2, P_4, P_1, P_3 \rangle$

$\langle P_0, P_2, P_4, P_3, P_1 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example System State

- P_2 requests an additional instance of C.
- Do we have a finish sequence?

Process	Request		
	A	B	C
P_0	0	0	0
P_1	2	0	2
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

7 August 2021

© Copyright Virtual University of
Pakistan



Example System State

Process	Allocation	Request	Work		
	A B C	A B C	A	B	C
P ₀	0 1 0	0 0 0	0	0	0
P ₁	2 0 0	2 0 2			
P ₂	3 0 2	0 0 1			
P ₃	2 1 1	1 0 0			
P ₄	0 0 2	0 0 2			

■ Finish Sequence: $\langle \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example System State

Process	Allocation	Request	Work		
	A B C	A B C	A	B	C
P ₀	0 1 0	0 0 0	0	0	0
P ₁	2 0 0	2 0 2	0	1	0
P ₂	3 0 2	0 0 1			
P ₃	2 1 1	1 0 0			
P ₄	0 0 2	0 0 2			

■ Finish Sequence: $\langle P_0 \rangle$

7 August 2021

© Copyright Virtual University of
Pakistan



Example System State

- P_0 's request can be satisfied with currently available resources, but request for no other process can be satisfied after that.

⇒ Deadlock exists, consisting of processes P_1 , P_2 , P_3 , and P_4 .

7 August 2021

© Copyright Virtual University of
Pakistan



Detection Algorithm

- How often should the detection algorithm be invoked?
 - Every time a request for allocation cannot be granted immediately—expensive but process causing the deadlock is identified, along with processes involved in deadlock

7 August 2021

© Copyright Virtual University of
Pakistan



Detection Algorithm

- Periodically, or based on CPU utilization
- Arbitrarily—there may be many cycles in the resource graph and we would not be able to tell which of the many deadlocked processes “caused” the deadlock.

7 August 2021

© Copyright Virtual University of
Pakistan



Recovery from Deadlock: Process Termination

- Abort all deadlocked processes.
- Abort one process at a time until the deadlock cycle is eliminated.

7 August 2021

© Copyright Virtual University of
Pakistan



Recovery from Deadlock: Process Termination

- In which order should we choose to abort processes?
 - Priority of a process.
 - How long the process has computed, and how much longer to completion.

7 August 2021

© Copyright Virtual University of
Pakistan



Recovery from Deadlock: Process Termination

- Resources the process has used.
- Resources the process needs to complete.
- How many processes will need to be terminated.
- Is the process interactive or batch?

7 August 2021

© Copyright Virtual University of
Pakistan



Recovery from Deadlock: Resource Preemption

- **Selecting a victim** – minimize cost.
- **Rollback** – return to some safe state, restart process from that state.
- **Starvation** – same process may always be picked as victim; include number of rollbacks in cost factor.

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Deadlock detection
- Recovery from deadlock
 - Process termination
 - Resource preemption

Operating Systems

Lecture 29

Syed Mansoor Sarwar

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 30

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- What is memory management
- Source code to execution
- Address binding
- Logical and physical address spaces
- Dynamic loading, dynamic linking, and overlays
- Recap of lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 29

- Deadlock detection
- Deadlock recovery
 - Process termination
 - Resource preemption

Memory Hierarchy

- Very small, extremely fast, extremely expensive, and volatile CPU registers
- Small, very fast, expensive, and volatile cache
- Hundreds of megabytes of medium-speed, medium-price, volatile main memory
- Hundreds of gigabytes of slow, cheap, and non-volatile secondary storage

7 August 2021

© Copyright Virtual University of
Pakistan



Purpose of Memory Management

- To ensure fair, secure, orderly, and efficient use of memory

7 August 2021

© Copyright Virtual University of
Pakistan



Memory Management

- Keeping track of used and free memory space
- When, where, and how much memory to allocate and deallocate
- Swapping processes in and out of main memory

7 August 2021

© Copyright Virtual University of
Pakistan



Source to Execution

Compile/Assemble



Link



Load

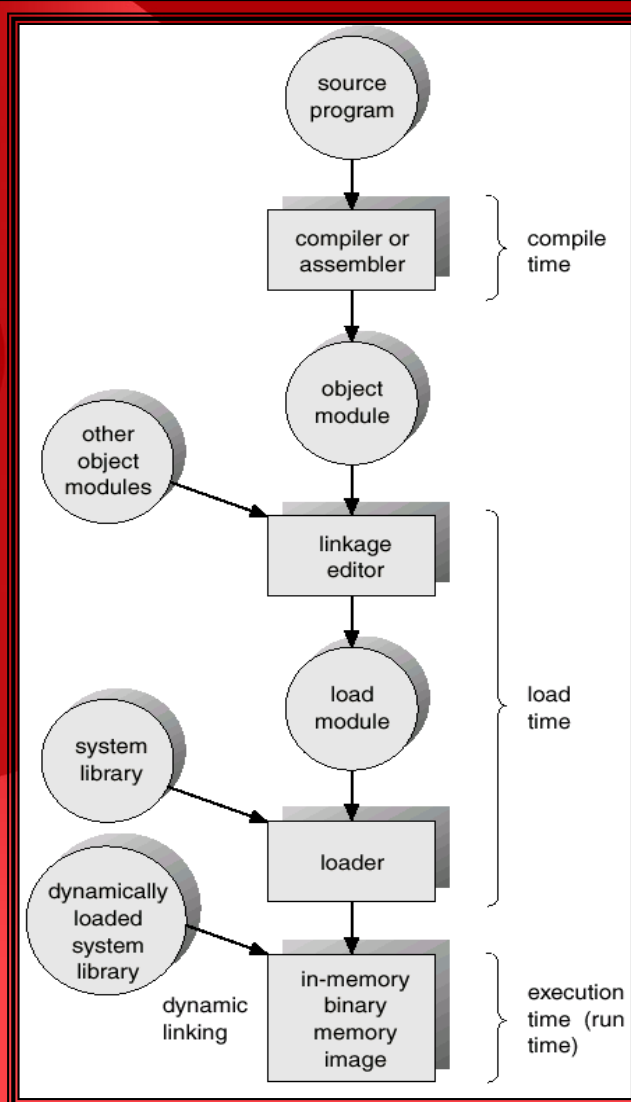


Execute

7 August 2021

© Copyright Virtual University of
Pakistan





7 August 2021

Address Binding

- Binding instructions and data to memory addresses
- Compile time
- Load time
- Execution time

7 August 2021

Address Binding

- **Compile time:** If you know at compile time where the process will reside in memory, the **absolute code** can be generated. Process must reside in the same memory region for it to execute correctly.

7 August 2021

© Copyright Virtual University of
Pakistan



Address Binding

- **Load time:** If the location of a process in memory is not known at compile time, then the compiler must generate **re-locatable code**. In this case the final binding is delayed until load time. Process can be loaded in different memory regions.

7 August 2021

© Copyright Virtual University of
Pakistan



Address Binding

- **Execution time:** If the process can be moved during its execution from one memory region to another, then binding must be delayed until run time. Special hardware must be available for this to work.

7 August 2021

© Copyright Virtual University of
Pakistan



Logical and Physical Addresses

- **Logical address:** An address generated by the process/CPU; refers to an instruction or data in the process
- **Physical address:** An address for a main memory location where instruction or data resides

7 August 2021

© Copyright Virtual University of
Pakistan



Logical and Physical Address Spaces

- The set of all logical addresses generated by a process comprises its **logical address space**.
- The set of physical addresses corresponding to these logical addresses comprises the **physical address space** for the process.

7 August 2021

© Copyright Virtual University of
Pakistan



Logical and Physical Address Spaces

- The run-time mapping from logical to physical addresses is done by a piece of the CPU hardware, called the **memory management unit (MMU)**.

7 August 2021

© Copyright Virtual University of
Pakistan



Example

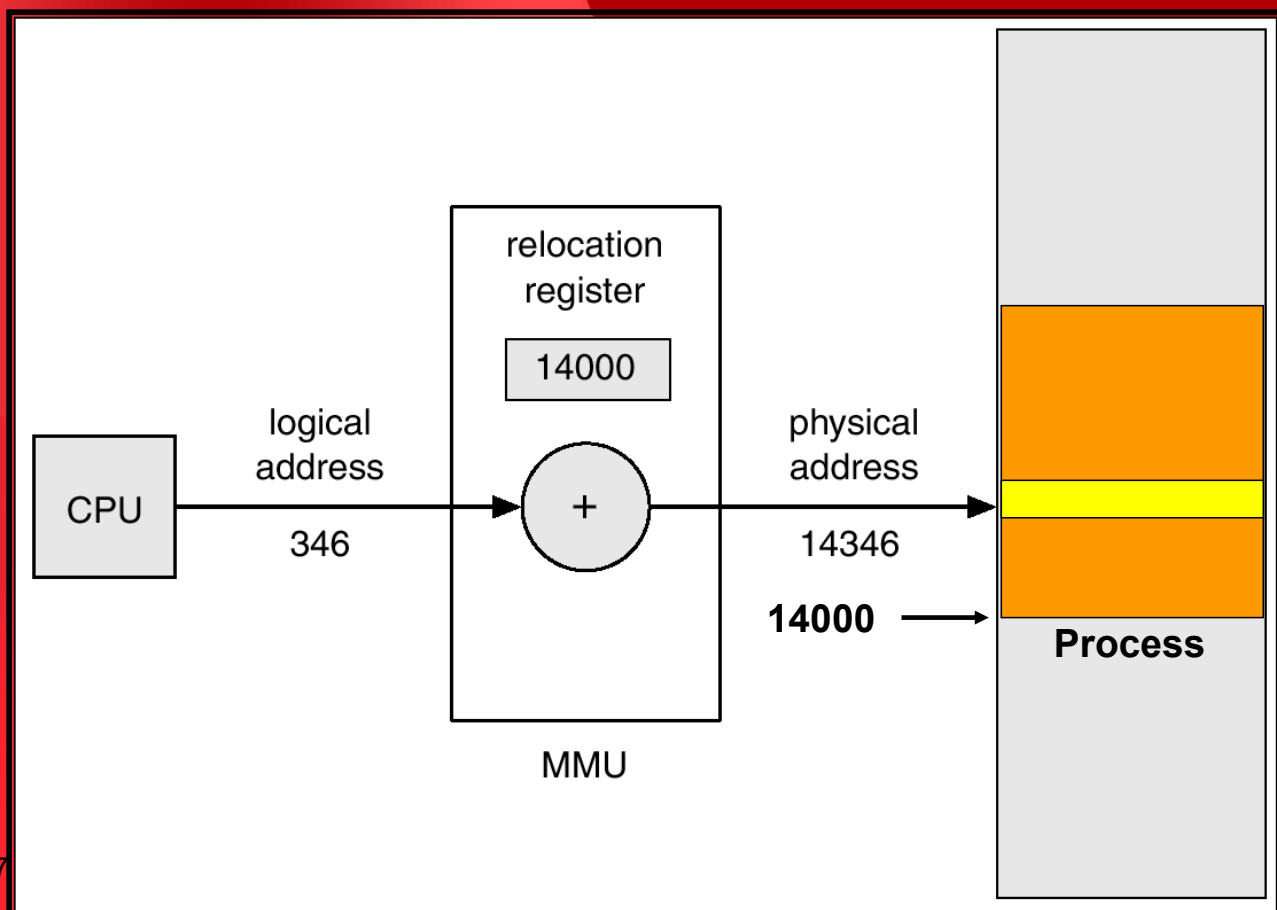
- The base register is called the **relocation register**.
- The value in the relocation register is added to every address generated by a user process at the time it is sent to **memory**.

7 August 2021

© Copyright Virtual University of
Pakistan



Example



Example

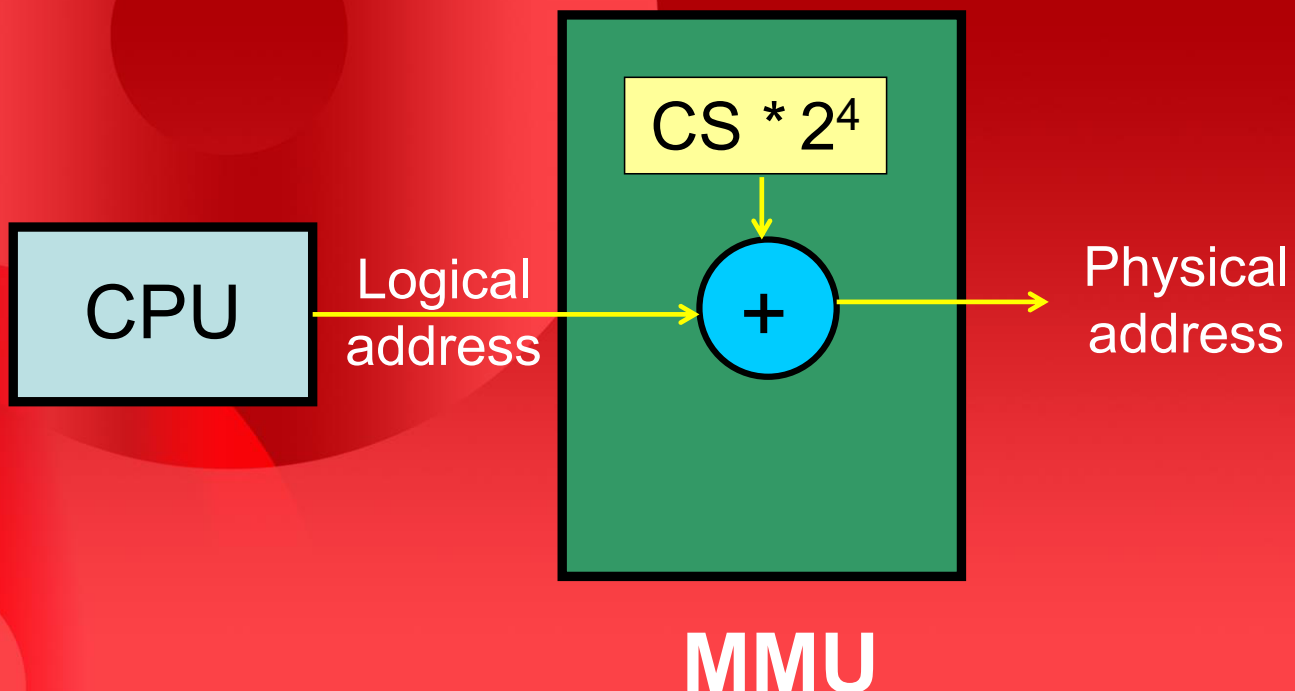
- In i8086, the logical address of the next instruction is specified by the value of **instruction pointer (IP)**. The physical address for the instruction is computed by shifting the **code segment register (CS)** left by four bits and adding IP to it.

7 August 2021

© Copyright Virtual University of
Pakistan



Example



7 August 2021

© Copyright Virtual University of
Pakistan



Example

- Logical address (16-bit)

IP = 0B10h

CS = D000h

- Physical address (20-bit)

$CS * 2^4 + IP = D0B10h$

- Sizes of logical and physical address spaces?

7 August 2021 © Copyright Virtual University of Pakistan



Dynamic Loading

- With dynamic loading, a routine is not loaded into the main memory until it is called.
- All routines are kept on the disk in a re-locatable format.
- The main program is loaded into memory and is executed

7 August 2021

© Copyright Virtual University of Pakistan



Dynamic Loading

- **Advantages**

- Potentially less time needed to load a program
- Potentially less memory space needed

- **Disadvantage**

- **Run-time activity**

7 August 2021

© Copyright Virtual University of
Pakistan



Dynamic Linking

- In **static linking**, system language libraries are linked at compile time and, like any other object module, are combined by the loader into the binary image

7 August 2021

© Copyright Virtual University of
Pakistan



Dynamic Linking

- In **dynamic linking**, linking is postponed until run-time.
- A library call is replaced by a piece of code, called **stub**, which is used to locate memory-resident library routine

7 August 2021

© Copyright Virtual University of
Pakistan



Dynamic Linking

- During execution of a process, stub is replaced by the address of the relevant library code and the code is executed
- If library code is not in memory, it is loaded at this time

7 August 2021

© Copyright Virtual University of
Pakistan



Dynamic Linking

■ Advantages

- Potentially less time needed to load a program
- Potentially less memory space needed
- Less disk space needed to store binaries

7 August 2021

© Copyright Virtual University of
Pakistan



Dynamic Linking

■ Disadvantages

- Time-consuming run-time activity, resulting in slower program execution

■ gcc compiler

- Dynamic linking by default
- `-static` option allows static

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- What is memory management
- From source code to execution
- Address binding
- Logical and physical address spaces
- Dynamic loading
- Dynamic linking

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 30

Syed Mansoor Sarwar

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 31

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Dynamic linking
- Overlays
- Swapping
- Contiguous storage allocation
- MFT and MVT
- Recap of lecture



Review of Lecture 30

- What is memory management?
- Source to execute
- Address binding
- Logical and physical address spaces
- Dynamic loading and linking

Dynamic Linking

- In **static linking**, system language libraries are linked at compile time and, like any other object module, are combined by the loader into the binary image

Dynamic Linking

- In **dynamic linking**, linking is postponed until run-time.
- A library call is replaced by a piece of code, called **stub**, which is used to locate memory-resident library routine

Dynamic Linking

- During execution of a process, stub is replaced by the address of the relevant library code and the code is executed
- If library code is not in memory, it is loaded at this time

Dynamic Linking

- **Advantages**

- Potentially less time needed to load a program
- Potentially less memory space needed
- Less disk space needed to store binaries

Dynamic Linking

- **Disadvantages**

- Time-consuming run-time activity, resulting in slower program execution

- **gcc compiler**

- Dynamic linking by default
- `-static` option allows static linking

Overlays

- Allow a process to be larger than the amount of memory allocated to it
- Keep in memory only those instructions and data that are needed at any given time

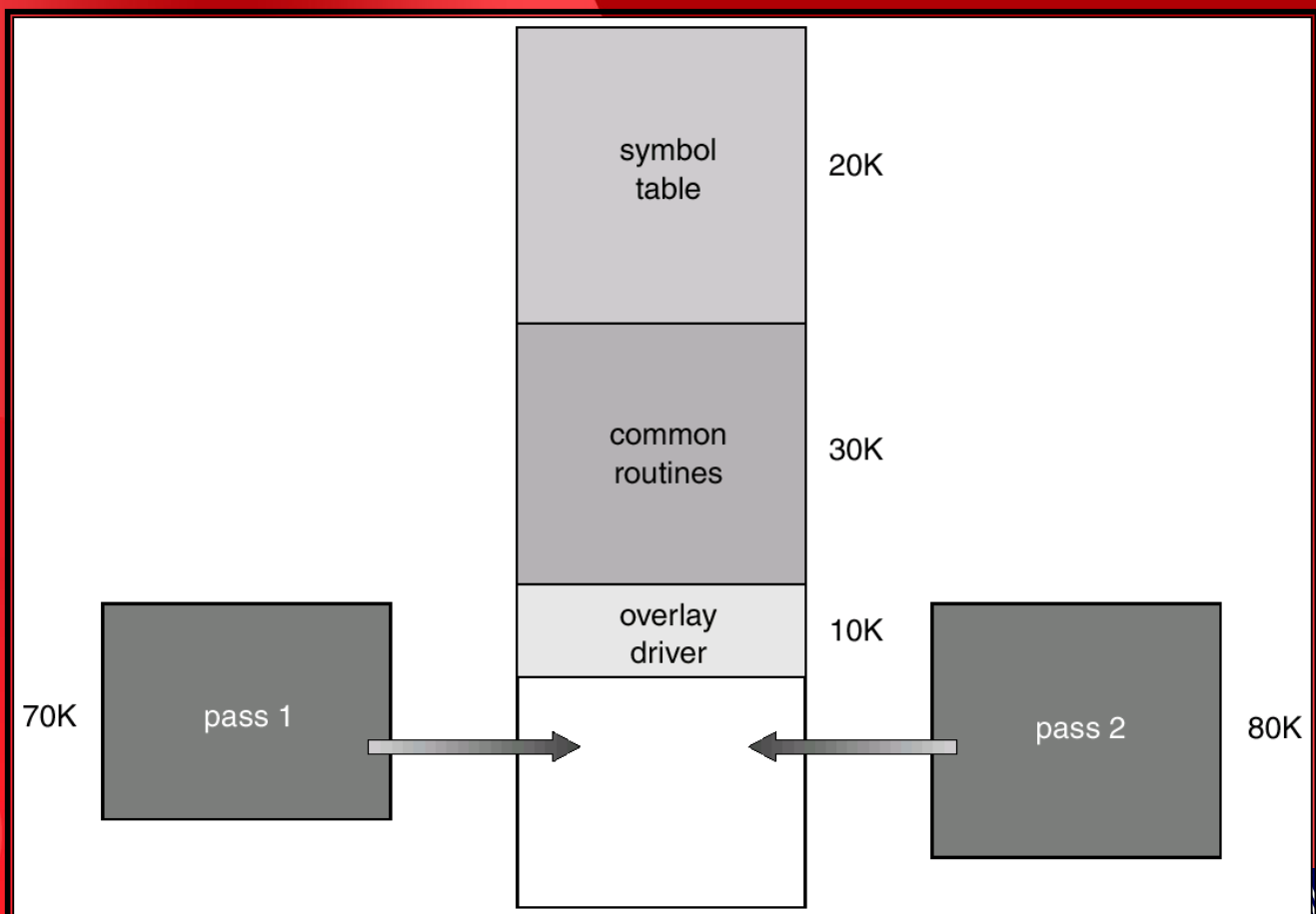
Overlays

- When other instructions are needed, they are loaded into the space occupied previously by instructions that are no longer needed
- Implemented by user
- Programming design of overlay structure is complex and not possible in all cases

Overlays Example

- 2-Pass assembler/compiler
- Available main memory: 150k
- Code size: 200k
 - Pass 1 70k
 - Pass 2 80k
 - Common routines 30k
 - Symbol table 20k

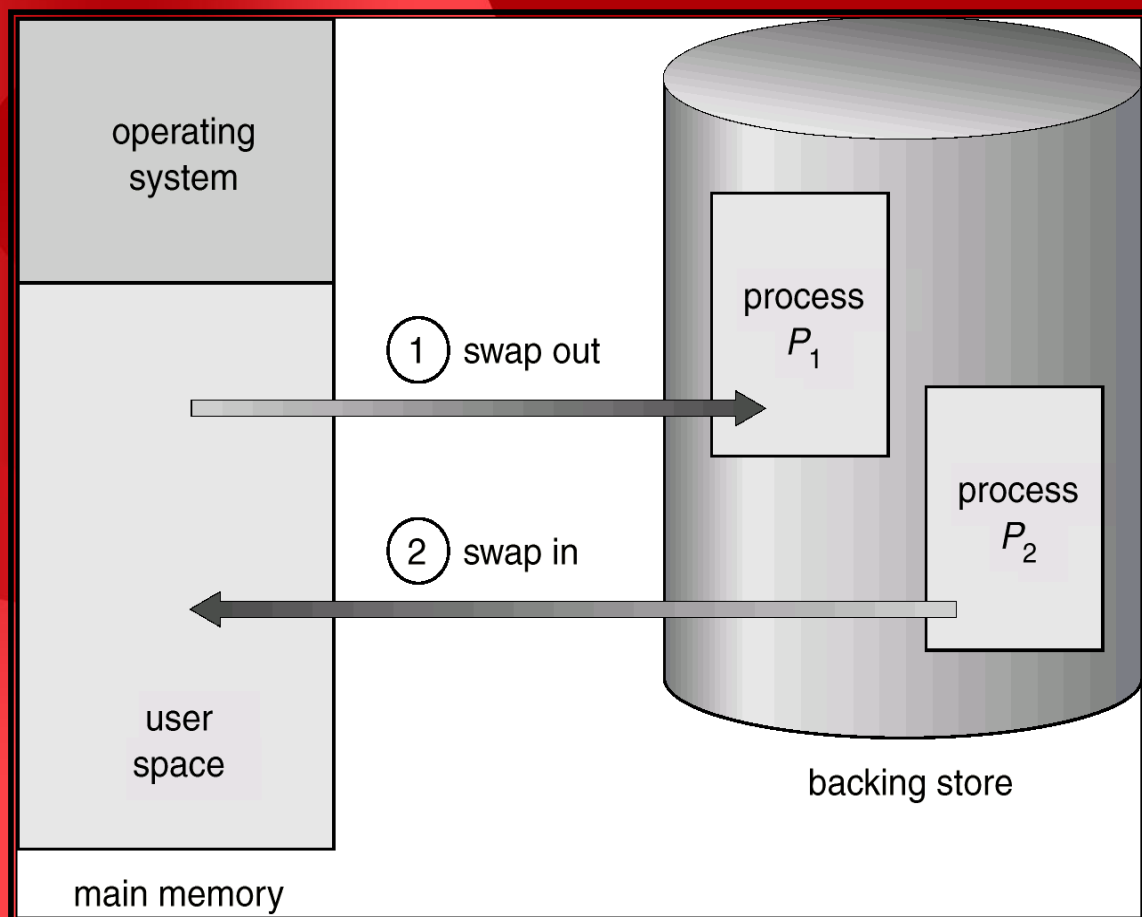
Overlays Example



Swapping

- Swap out and swap in (or roll out and roll in)
- Major part of swap time is transfer time; the total transfer time is directly proportional to the *amount* of memory swapped
- Large context switch time

Swapping



Cost of Swapping

- Process size = 1 MB
- Transfer rate = 5 MB/sec
- Swap out time = $1/5$ sec
= 200 ms
- Average latency = 8 ms
- Net swap out time = 208 ms
- Swap out + swap in = 416 ms

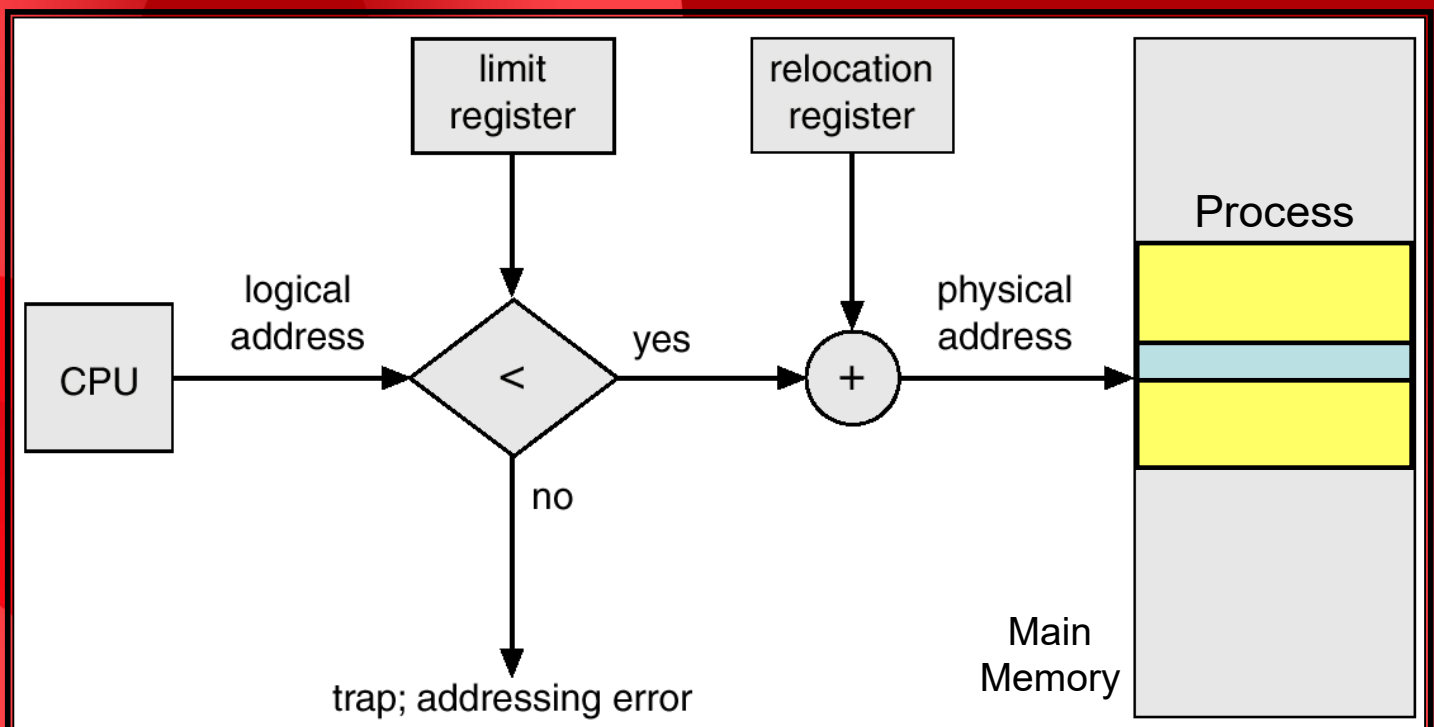
Issues with Swapping

- Quantum for RR scheduler
- Pending I/O for swapped out process
- User space used for I/O
- Solutions
 - Don't swap out processes with pending I/O
 - Do I/O using kernel space

Contiguous Allocation

- Kernel space, user space
- A process is placed in a single contiguous area in memory
- **Base** (re-location) and **limit** registers are used to point to the smallest memory address of a process and its size, respectively.

Contiguous Allocation



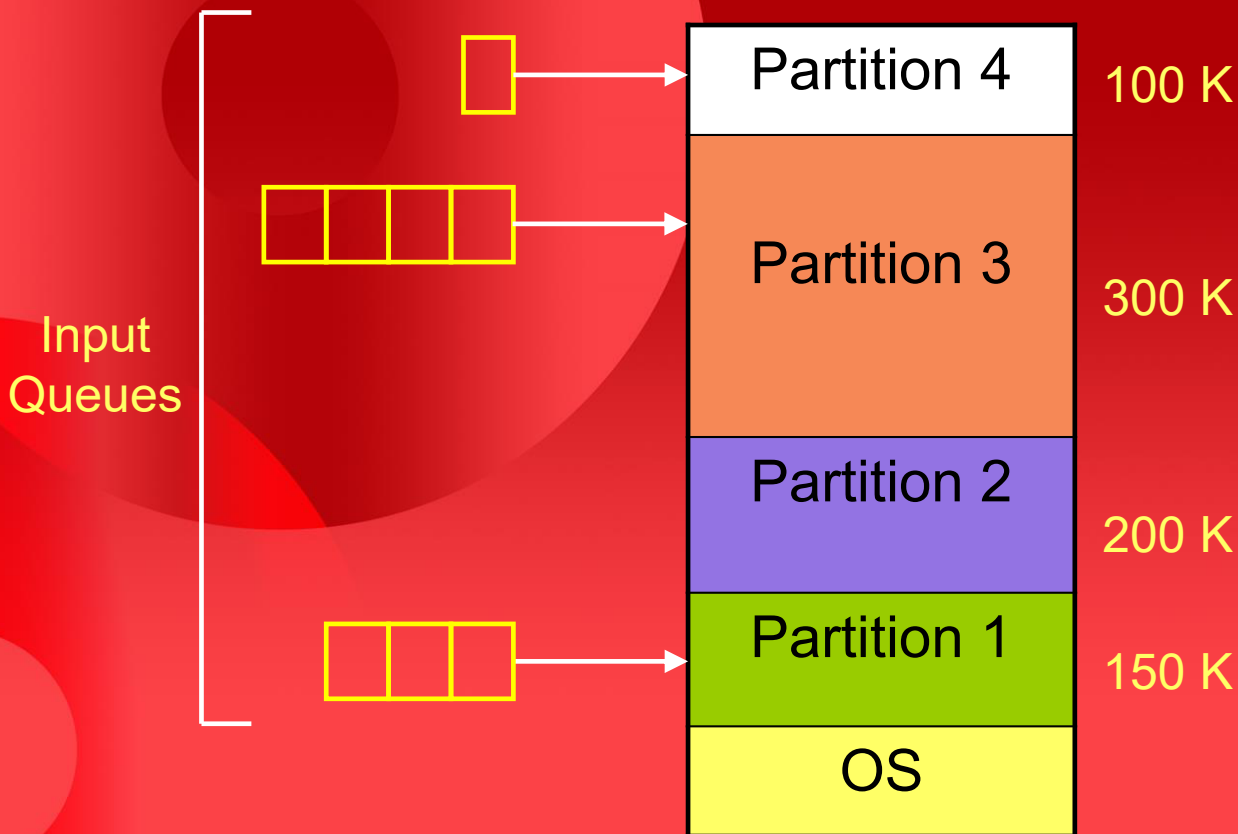
MFT

- Multiprogramming with fixed tasks (MFT)
- Memory is divided into several fixed-size partitions.
- Each partition may contain exactly one process/task.

MFT

- Boundaries for partitions are set at boot time and are *not* movable.
- An input queue per partition
- The degree of multiprogramming is bound by the number of partitions.

MFT



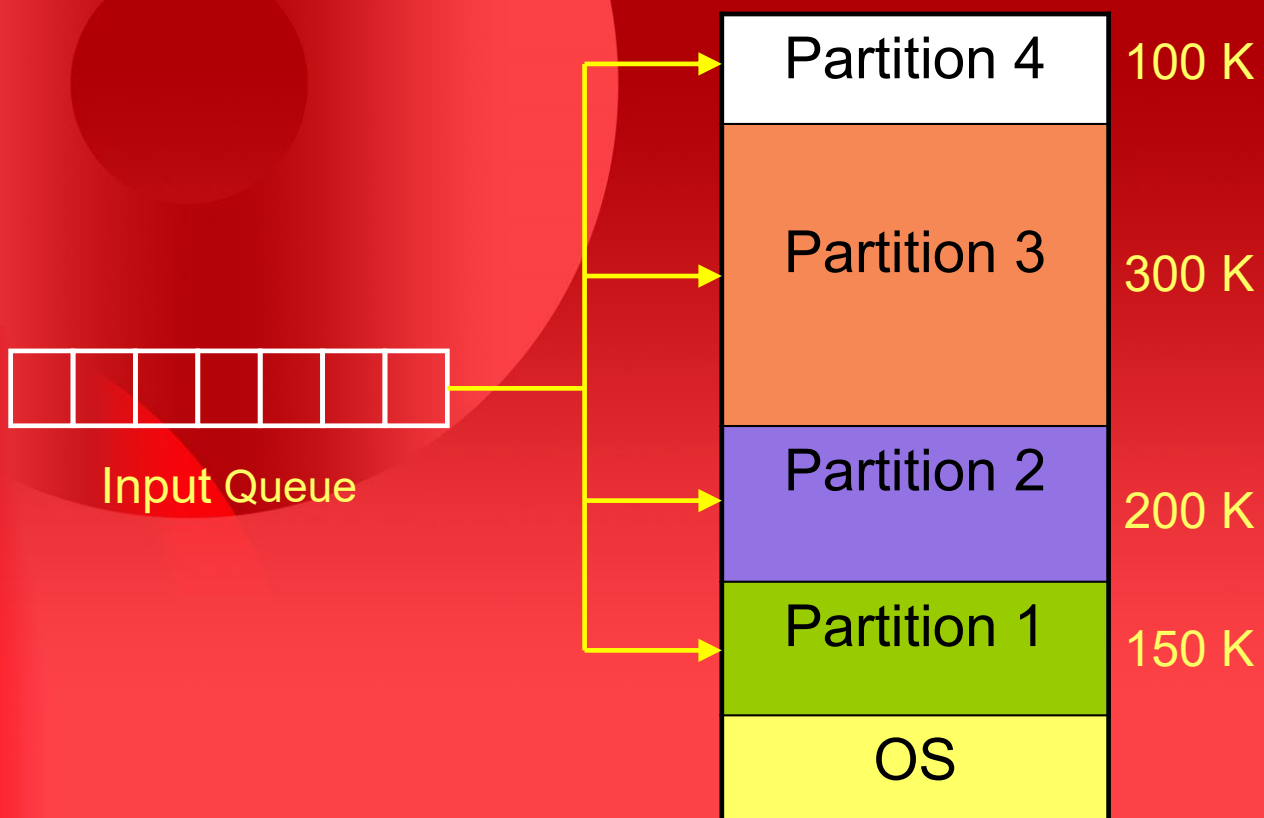
MFT With Multiple Input Queues

- Potential for wasted memory space—an empty partition but no process in the associated queue
- Load-time address binding

MFT With Single Input Queue

- Single queue for all partitions
 - Search the queue for a process when a partition becomes empty
 - *First-fit, best-fit, worst-fit* space allocation algorithms

MFT With Single Input Queue



MFT Issues

- Internal fragmentation—wasted space *inside* a fixed-size memory region
- No sharing between processes.
- Load-time address binding with multiple input queues

Recap of Lecture

- Dynamic linking
- Overlays
- Swapping
- Contiguous storage allocation
- MFT

Operating Systems

Lecture 31

Syed Mansoor Sarwar



Operating Systems

Lecture 32

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- MVT
- Paging
- Addressing and logical to physical address translation
- Page table implementation
- Recap of lecture

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 31

- Dynamic linking
- Overlays
- Swapping
- Contiguous storage allocation
- MFT

7 August 2021

© Copyright Virtual University of
Pakistan



MVT

- Multiprogramming with variable tasks (MFT)
- Both the number and size of partitions change with time
- A process still resides in one region but can be of any size up to the size of the user space and can change with time.

7 August 2021

© Copyright Virtual University of
Pakistan



MVT

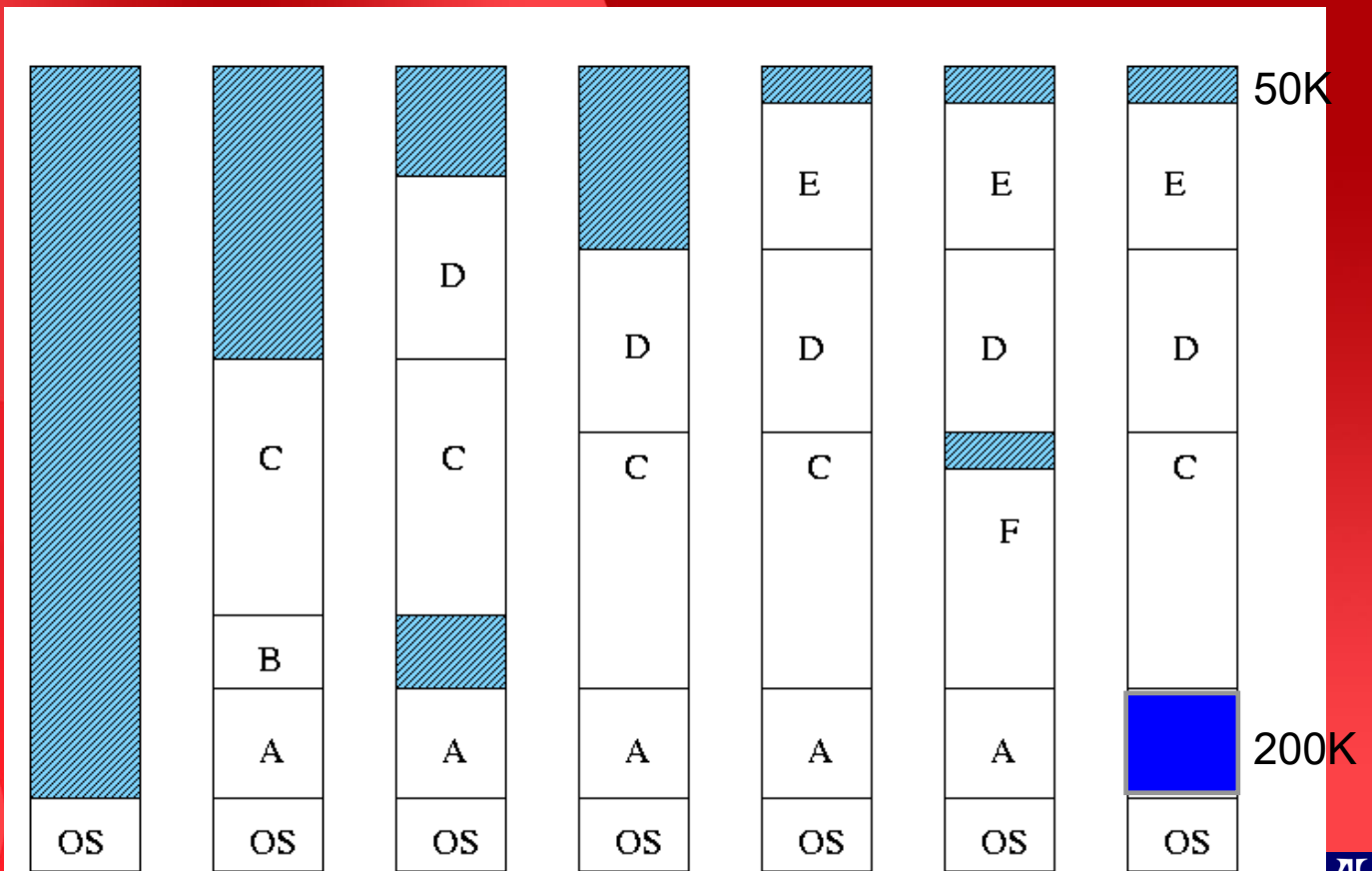
- Jobs can move from one partition to another
- Dynamic address binding
- No internal fragmentation
- Introduces **external fragmentation**

7 August 2021

© Copyright Virtual University of
Pakistan



MVT



External Fragmentation

- External Fragmentation refers to the state of memory space when total amount of unused memory space exists to satisfy a request but this memory space is not contiguous.

External Fragmentation

- **Compact** main memory (shuffle processes around to put free space in one contiguous area)
- Slows down execution of currently running processes

7 August 2021

© Copyright Virtual University of
Pakistan



Paging

- Divide physical memory into fixed-sized blocks, called frames
- Divide logical memory into blocks of the same size, called pages
- Size of a page is a power of 2
- Typical page sizes: 1K – 16K

7 August 2021

© Copyright Virtual University of
Pakistan



Paging

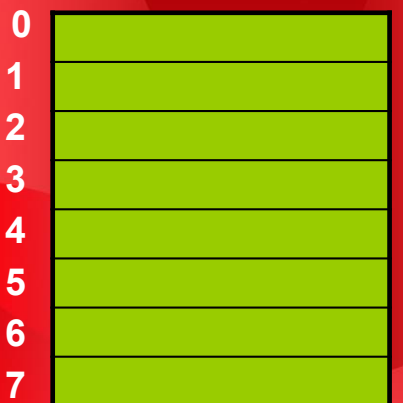
- Keep track of all free frames.
- To run a program of size n pages, find n free frames and load program pages into these frames.
- Keep track of a program's pages in the main memory by using the page table.

7 August 2021

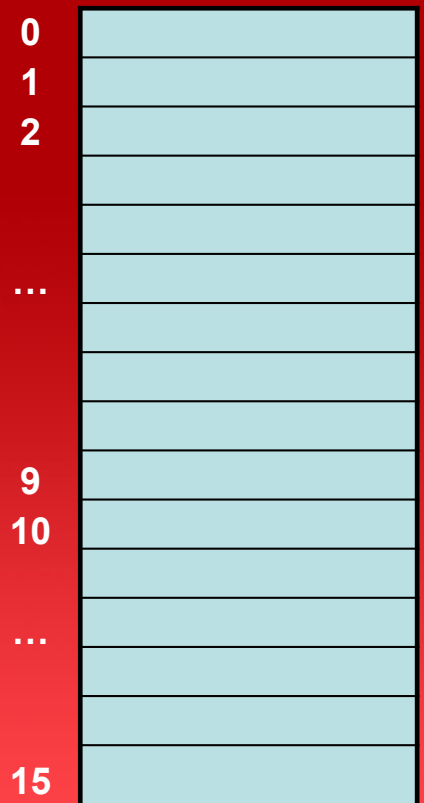
© Copyright Virtual University of
Pakistan



Paging



Process Address
Space



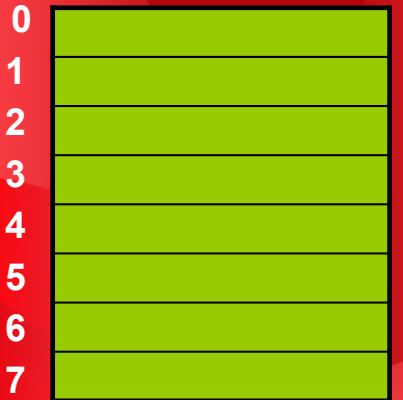
Physical Address
Space

7 August 2021

© Copyright Virtual University of
Pakistan



Paging

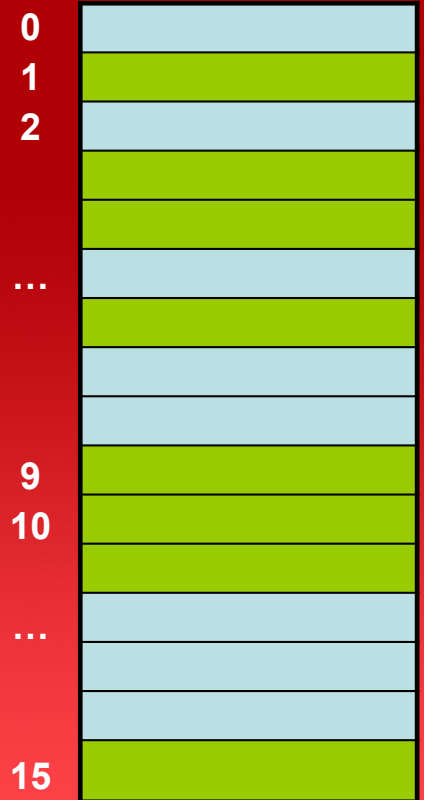


Process Address Space

Page

0	3
1	9
2	11
3	...
...	
7	1

Page Table



Physical Address Space



7 August 2021

© Copyright Virtual University of Pakistan

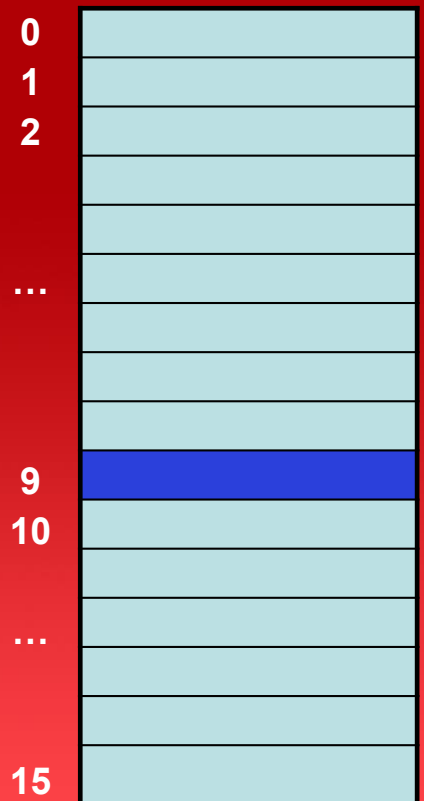
Paging



Process Address Space

0	
1	9
2	
3	...
...	
7	

Page Table



Physical Address Space



7 August 2021

© Copyright Virtual University of Pakistan

Paging

- Logical address space of a process can be noncontiguous; a process page is allocated a frame whenever the latter is available.
- Set up a page table to translate logical addresses to physical addresses.
- Internal fragmentation

7 August 2021

© Copyright Virtual University of

Pakistan



Addressing in Paging

- Logical address: (p, d)

where, p is page number and d is offset within the page

- p is used to index the process page table; page table entry contains the frame number, f , where page p is loaded

7 August 2021

© Copyright Virtual University of

Pakistan



Addressing in Paging

- Physical address of the location referenced by (p, d) is computed by appending d at the end of f , as shown below:

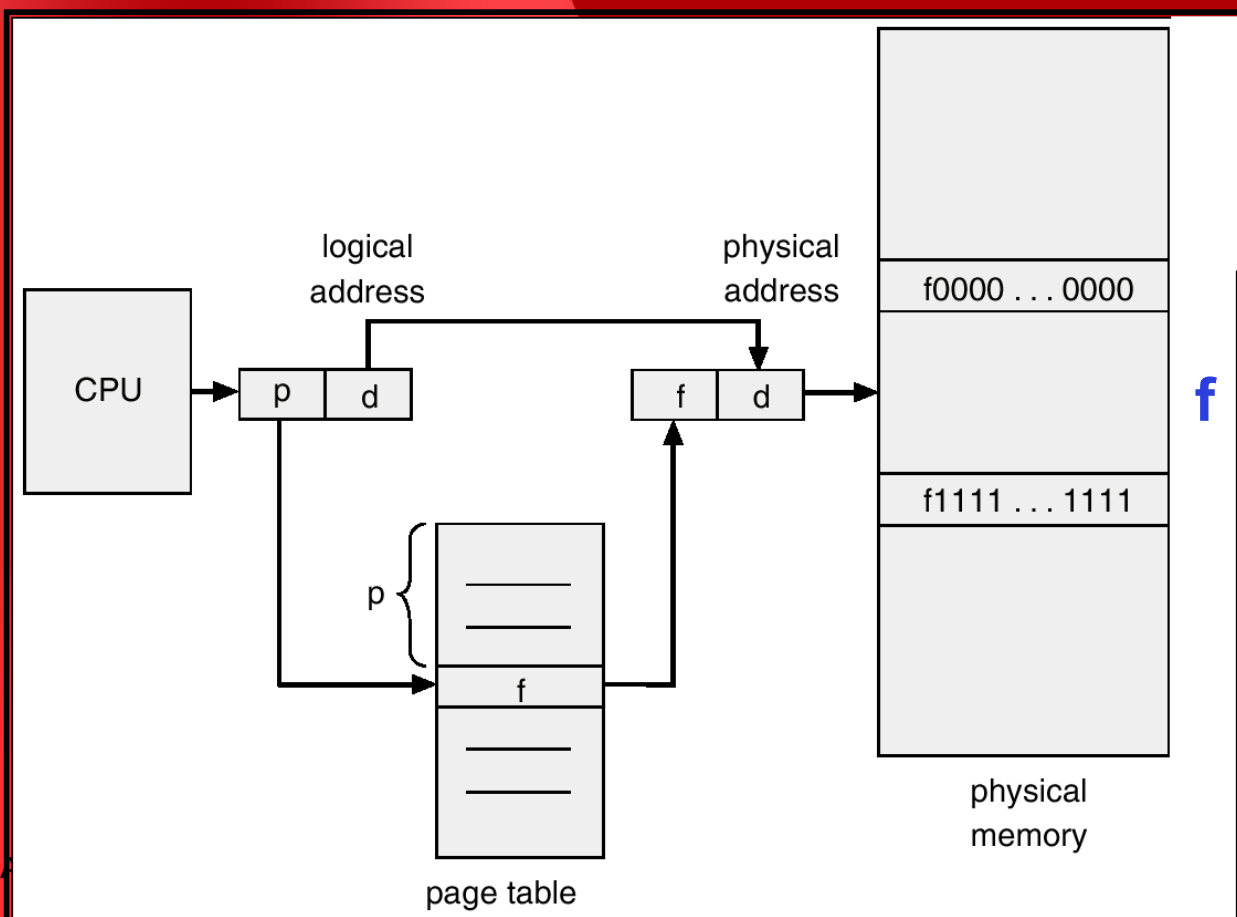


7 August 2021

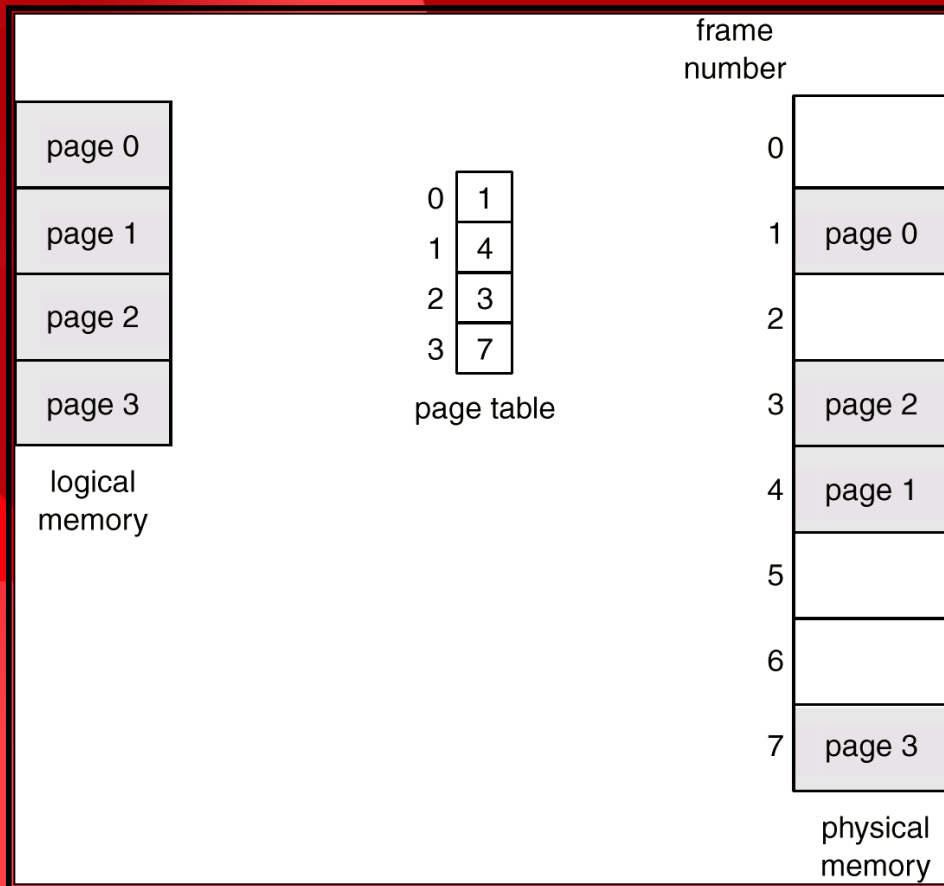
© Copyright Virtual University of
Pakistan



Address Translation



Addressing Example

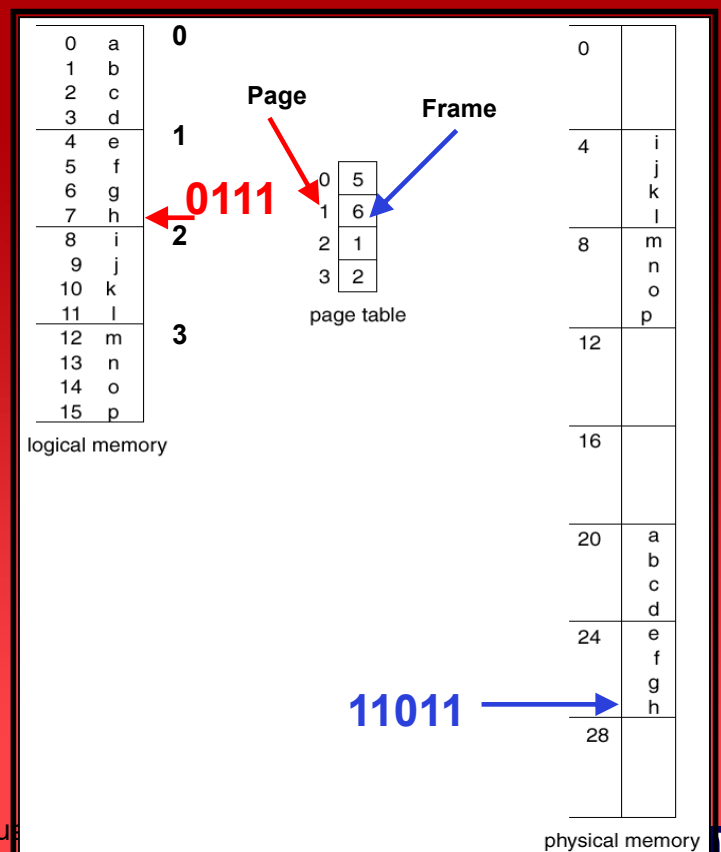


7 August 2021



Paging Example

- Page size = 4 bytes
- Process address space = 4 pages
- Physical address space = 8 frames
- Logical address: (1,3)
= 0111
- Physical address: (6,3)
= 11011



7 August 2021

© Copyright Virtus
Pakistan



Addressing in Paging

- Logical address space of 16 pages of 1024 words each, mapped into a physical memory of 32 frames.
- Logical address size?
- Physical address size?
- Number of bits for p, f, and d?

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- MVT
- Paging
- Logical and physical addresses
- Address translation

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 32

Syed Mansoor Sarwar

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 33

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Addressing and logical to physical address translation
- Examples: Intel P4 and PDP-11
- Page table implementation
- Performance of paging
- Protection in paging

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 32

- MVT
- Paging
- Addressing and address translation in paging

7 August 2021

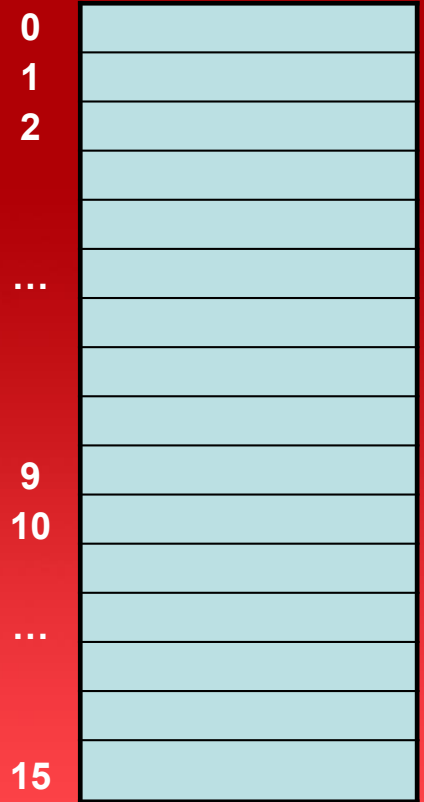
© Copyright Virtual University of
Pakistan



Paging



Process Address
Space



Physical Address
Space

7 August 2021

© Copyright Virtual University of
Pakistan



Paging

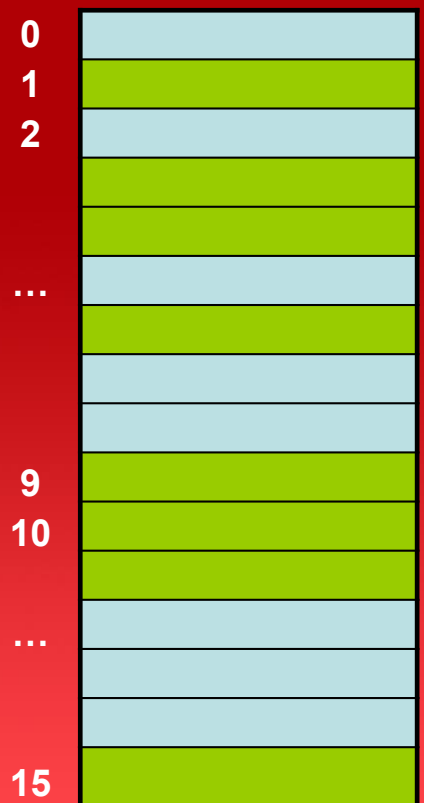


Process Address
Space

Page

0	3
1	9
2	11
3	...
...	
7	1

Page Table



Physical Address
Space

7 August 2021

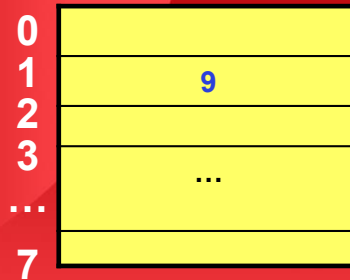
© Copyright Virtual University of
Pakistan



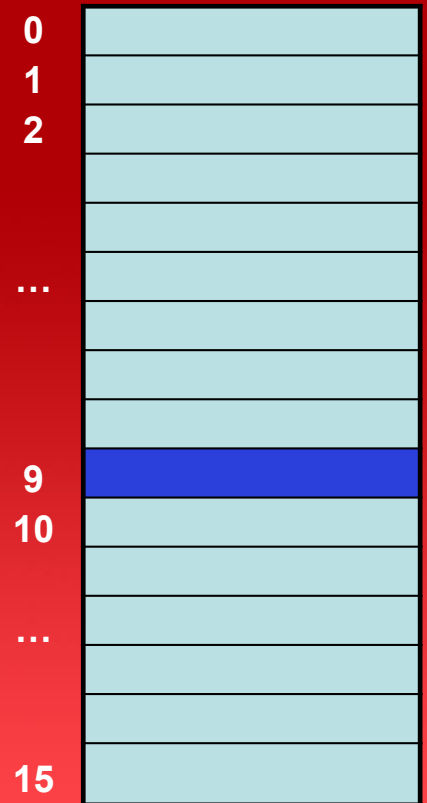
Paging



Process Address Space



Page Table



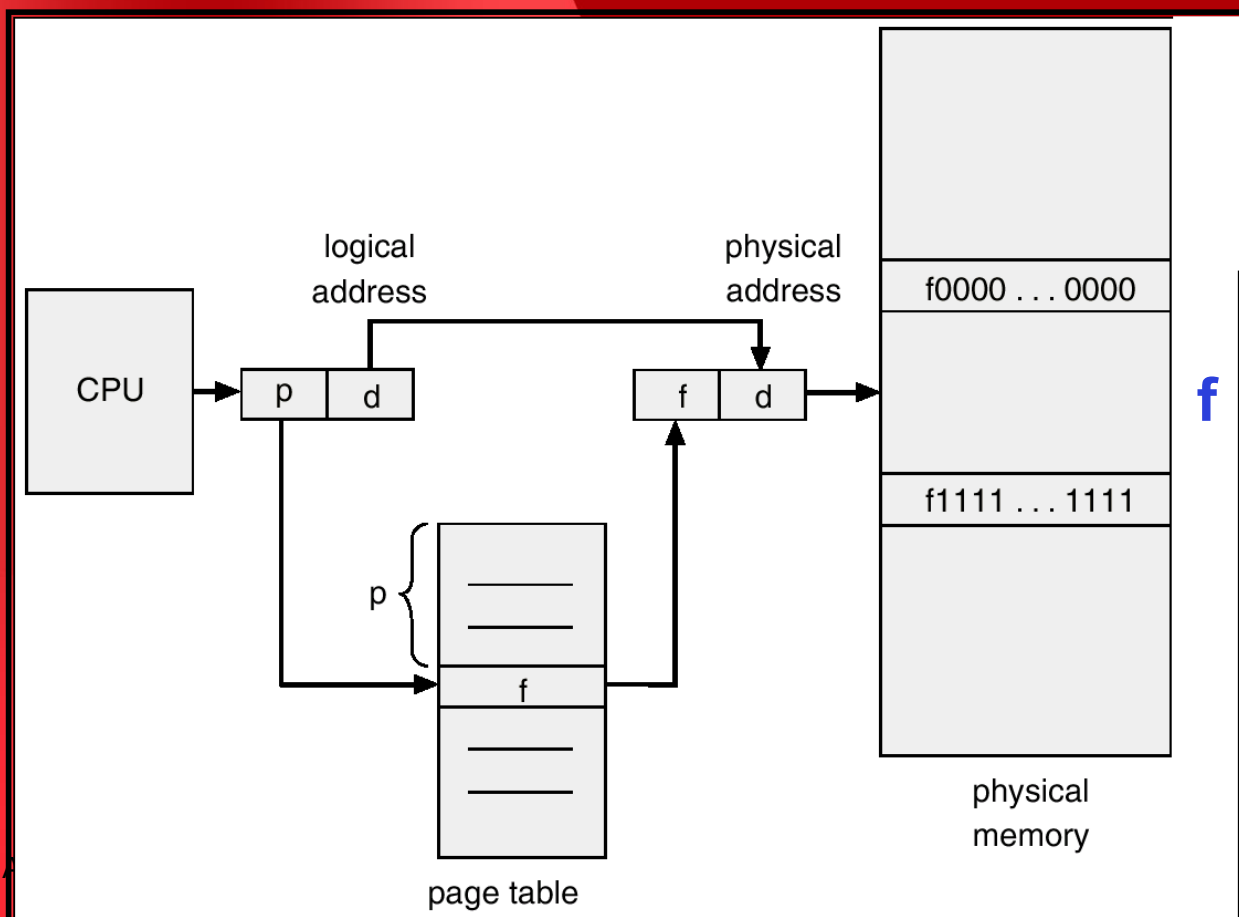
Physical Address Space

7 August 2021

© Copyright Virtual University of Pakistan

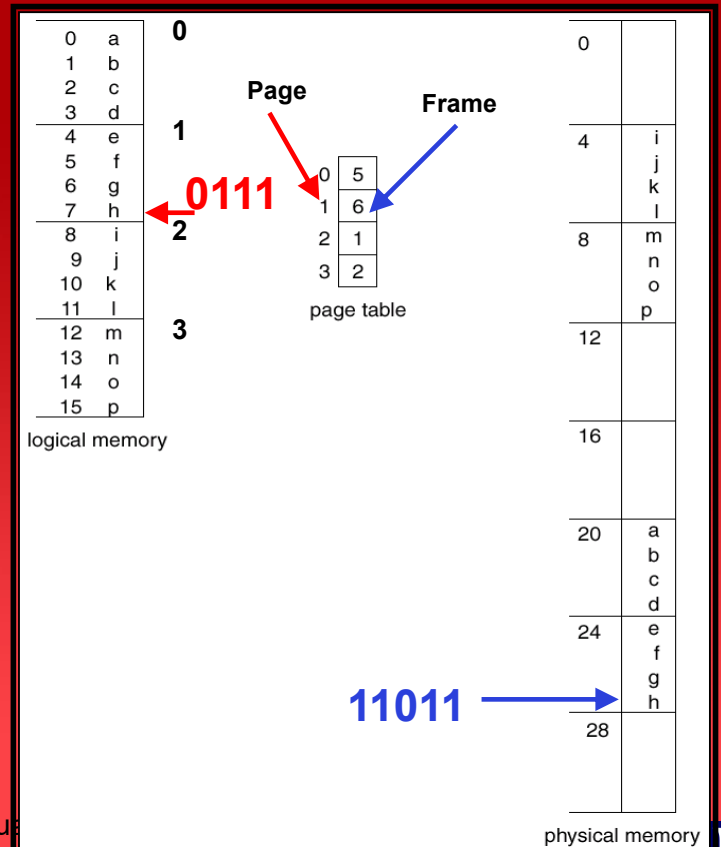


Address Translation



Paging Example

- Page size = 4 bytes
- Process address space = 4 pages
- Physical address space = 8 frames
- Logical address: (1,3)
= 0111
- Physical address: (6,3)
= 11011



7 August 2021

© Copyright Virtual University of Pakistan

physical memory



Addressing in Paging

- Logical address space of 16 pages of 1024 words each, mapped into a physical memory of 32 frames.
- Logical address size?
- Physical address size?
- Number of bits for p, f, and d?

7 August 2021

© Copyright Virtual University of Pakistan



Addressing in Paging

- No. of bits for **p** = ceiling $[\log_2 16]$ bits
= 4 bits
- No. of bits for **f** = ceiling $[\log_2 32]$ bits
= 5 bits
- No. of bits for **d** = ceiling $[\log_2 2048]$ bits
= 11 bits

7 August 2021

© Copyright Virtual University of
Pakistan



Addressing in Paging

$$\begin{aligned}\text{Logical address size} &= |p| + |d| \\ &= 4 + 11 \\ &= 15 \text{ bits}\end{aligned}$$

$$\begin{aligned}\text{Physical address size} &= |f| + |d| \\ &= 5 + 11 \\ &= 16 \text{ bits}\end{aligned}$$

7 August 2021

© Copyright Virtual University of
Pakistan



Page Table Size

- Page table size = $NP * PTES$

where NP is the number of pages in the process address space and PTES is the page table entry size (equal to $|f|$ based on our discussion so far).

Page table size = $16 * 5 \text{ bits}$

7 August 2021

© Copyright Virtual University of
Pakistan

= 16 bytes



Paging in Intel P4

- 32-bit *linear* address
- 4K page size
- Maximum pages in a process address space = $2^{32} / 4K$
- Number of bits for $d = \log_2 4K$
= 12
- Number of bits for $p = 32 - 12$

7 August 2021

© Copyright Virtual University of
Pakistan



Paging in PDP-11

- 16-bit logical address
- 8K page size
- Maximum pages in a process address space = $2^{16} / 8K = 8$
- $|d| = \log_2 8K = 13$ bits
- $|p| = 16 - 13 = 3$ bits
- $|f| = ?$ bits

7 August 2021

© Copyright Virtual University of
Pakistan



Another Example

- Logical address = 32-bit
- Process address space = 2^{32} B
= 4 GB
- Main memory = RAM = 512 MB
- Page size = 4K
- Maximum pages in a process address space = $2^{32} / 4K = 1M$

7 August 2021

© Copyright Virtual University of
Pakistan



Another Example

- $|d| = \log_2 4K = 12 \text{ bits}$
- $|p| = 32 - 12 = 20 \text{ bits}$
- No. of frames = $512 \text{ M} / 4 \text{ K}$
= 128 K
- $|f| = \text{ceiling} [\log_2 128 \text{ K}] \text{ bits}$
= $17 \text{ bits} \approx 4 \text{ bytes}$
- Physical address = $17 + 12 \text{ bits}$



Implementation of Page Table

- In CPU registers
- OK for small process address spaces and large page sizes
- Effective memory access time ($T_{\text{effective}}$) is about the same as memory access time (T_{mem})

Implementation of Page Table

- Keep page table in the main memory
- Page table base register (PTBR)
- $T_{\text{effective}} = 2T_{\text{mem}}$
- $T_{\text{effective}}$ is not acceptable

7 August 2021

© Copyright Virtual University of
Pakistan



Implementation of Page Table

- Use a special, small, fast lookup hardware, called translation look-aside buffer (TLB)
- Typically 64–1024 entries
- An entry is (key, value)
- Parallel search for key; on a hit, value is returned

7 August 2021

© Copyright Virtual University of
Pakistan



Implementation of Page Table

- (key,value) is (p,f) for paging
- For a logical address, (p,d), TLB is searched for p. If an entry with a key p is found, we have a hit and f is used to form the physical address. Else, page table in the main memory is searched.

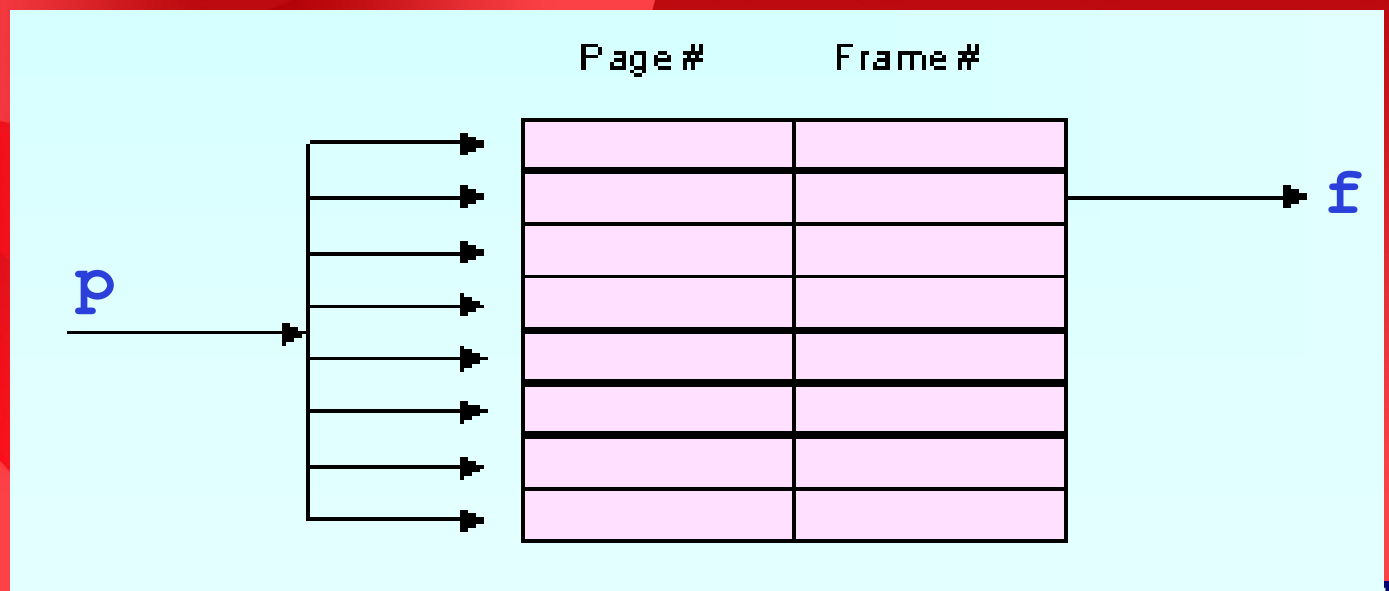
7 August 2021

© Copyright Virtual University of
Pakistan



TLB

Logical address: (p, d)



Implementation of Page Table

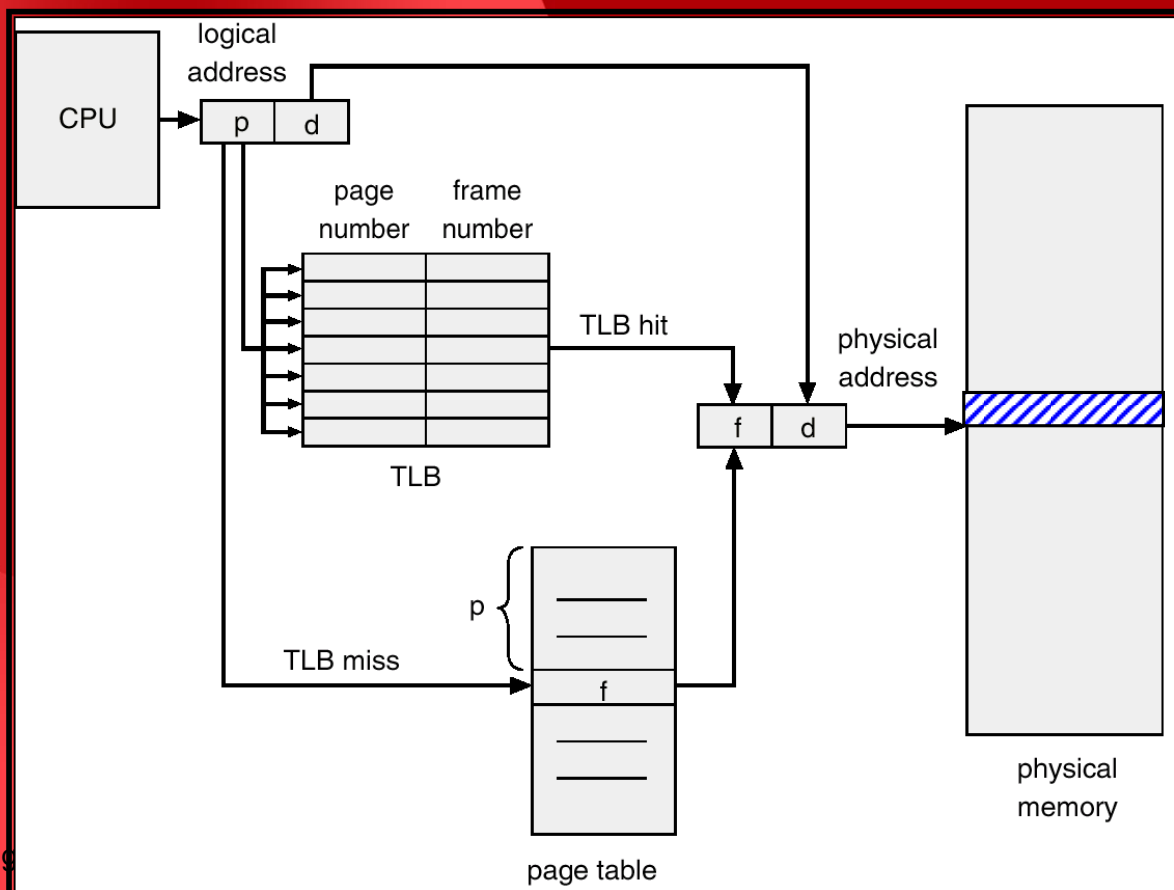
- The TLB is loaded with the (p,f) pair so that future references to p are found in the TLB, resulting in improved hit ratio.
- On a context switch, the TLB is flushed and is loaded with values for the scheduled process.

7 August 2021

© Copyright Virtual University of
Pakistan



Paging Hardware



7 Aug



Performance of Paging

- $T_{\text{effective}}$ on a hit = $T_{\text{mem}} + T_{\text{TLB}}$
- $T_{\text{effective}}$ on a miss = $2T_{\text{mem}} + T_{\text{TLB}}$
- If HR is hit ratio and MR is miss ratio, then

$$T_{\text{effective}} = \text{HR} (T_{\text{TLB}} + T_{\text{mem}}) + \text{MR} (T_{\text{TLB}} + 2T_{\text{mem}})$$

Example

- $T_{\text{mem}} = 100 \text{ nsec}$
- $T_{\text{TLB}} = 20 \text{ nsec}$
- Hit ratio is 80%
- $T_{\text{effective}} = ?$

$$\begin{aligned} T_{\text{effective}} &= 0.8 (20 + 100) \\ &\quad + 0.2 (20 + 2 \times 100) \\ &= 140 \text{ nanoseconds} \end{aligned}$$

Example

- $T_{\text{mem}} = 100 \text{ nsec}$
- $T_{\text{TLB}} = 20 \text{ nsec}$
- Hit ratio is 98%
- $T_{\text{effective}} = ?$

$$\begin{aligned} T_{\text{effective}} &= 0.98 (20 + 100) \\ &\quad + 0.02 (20 + 2 \times 100) \\ &= 122 \text{ nanoseconds} \end{aligned}$$

Recap of Lecture

- Logical and physical addresses
- Address translation
- Intel P4 example
- PDP-11 example
- Implementation of page table
- Performance of paging

Operating Systems

Lecture 33

Syed Mansoor Sarwar

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 34

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Protection in paging
- Structure of the page table
 - Multi-level paging
 - Hashed page tables
 - Inverted page table
- Sharing in paging

August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 33

- Addressing and logical to physical address translation
- Examples: Intel P4 and PDP-11
- Page table implementation
- Performance of paging

Example

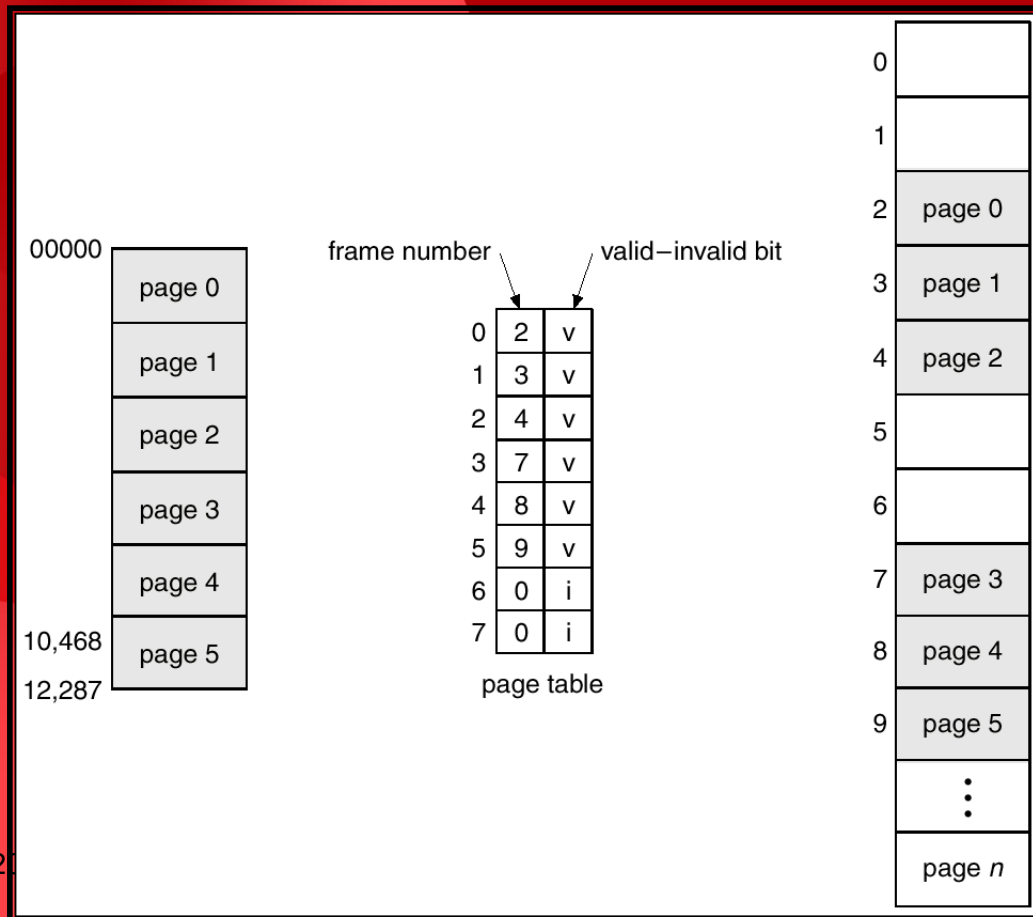
- $T_{\text{mem}} = 100 \text{ nsec}$
- $T_{\text{TLB}} = 20 \text{ nsec}$
- Hit ratio is 98%
- $T_{\text{effective}} = ?$

$$\begin{aligned} T_{\text{effective}} &= 0.98 (20 + 100) \\ &\quad + 0.02 (20 + 2 \times 100) \\ &= 122 \text{ nanoseconds} \end{aligned}$$

Protection

- Memory protection in paging is achieved by associating protection bit with each page table entry
- Valid/Invalid bit (v)—page in the process address space or not
- Read, write, and execute bits (r, w, x)

Protection



7 August 2021

Structure of the Page Table

- Hierarchical / Multilevel Paging
- Hashed Page Table
- Inverted Page Table

Multilevel Paging

- Logical address = 32-bit
- Page size = 4K bytes (2^{12} bytes)
- Page table entry = 4 bytes
- Maximum pages in a process address space = $2^{32} / 4K = 1M$

7 August 2021

© Copyright Virtual University of
Pakistan



Multilevel Paging

- Maximum pages in a process address space = $2^{32} / 4K = 1M$
- Page table size = 4M bytes
- This page cannot fit in one page $\Rightarrow$ Page the page table

7 August 2021

© Copyright Virtual University of
Pakistan



Multilevel Paging

- Page table needed for keeping track of pages of the page table—called the outer page table or page directory
- No. of pages in the page table is $4M / 4K = 1K$
- Size of the outer page table is $1K * 4 \text{ bytes} = 4K \text{ bytes} \Rightarrow$ outer page will fit in one page

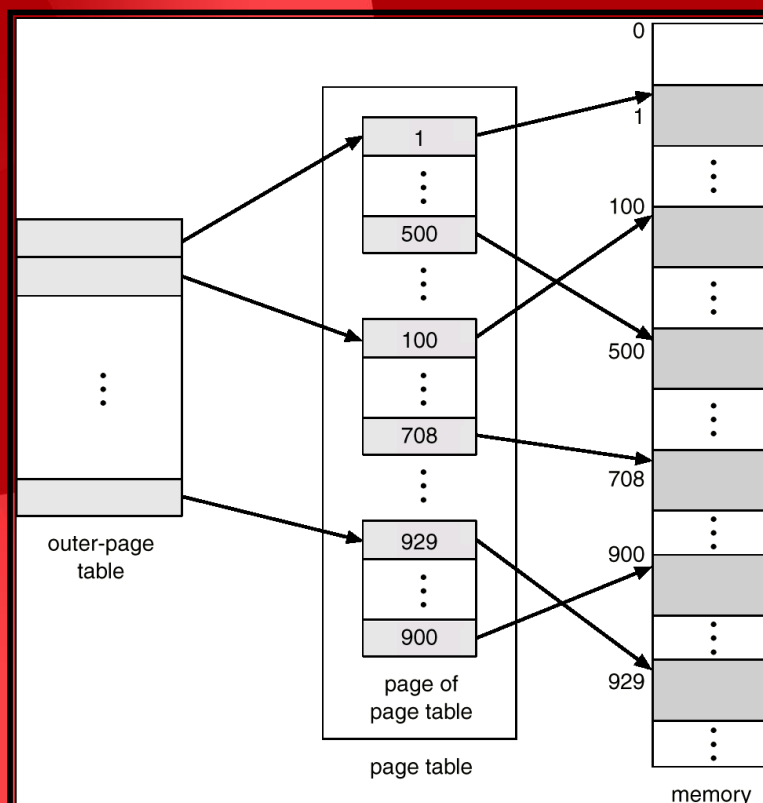
7 August 2021

© Copyright Virtual University of
Pakistan



Multilevel Paging

- 2-level paging



7 August 2021



Multilevel Paging

- Addressing and address translation
 - 32-bit logical address: p_1 used to index the outer page table and p_2 to index the inner page table
- page number page offset

p_1	p_2	d

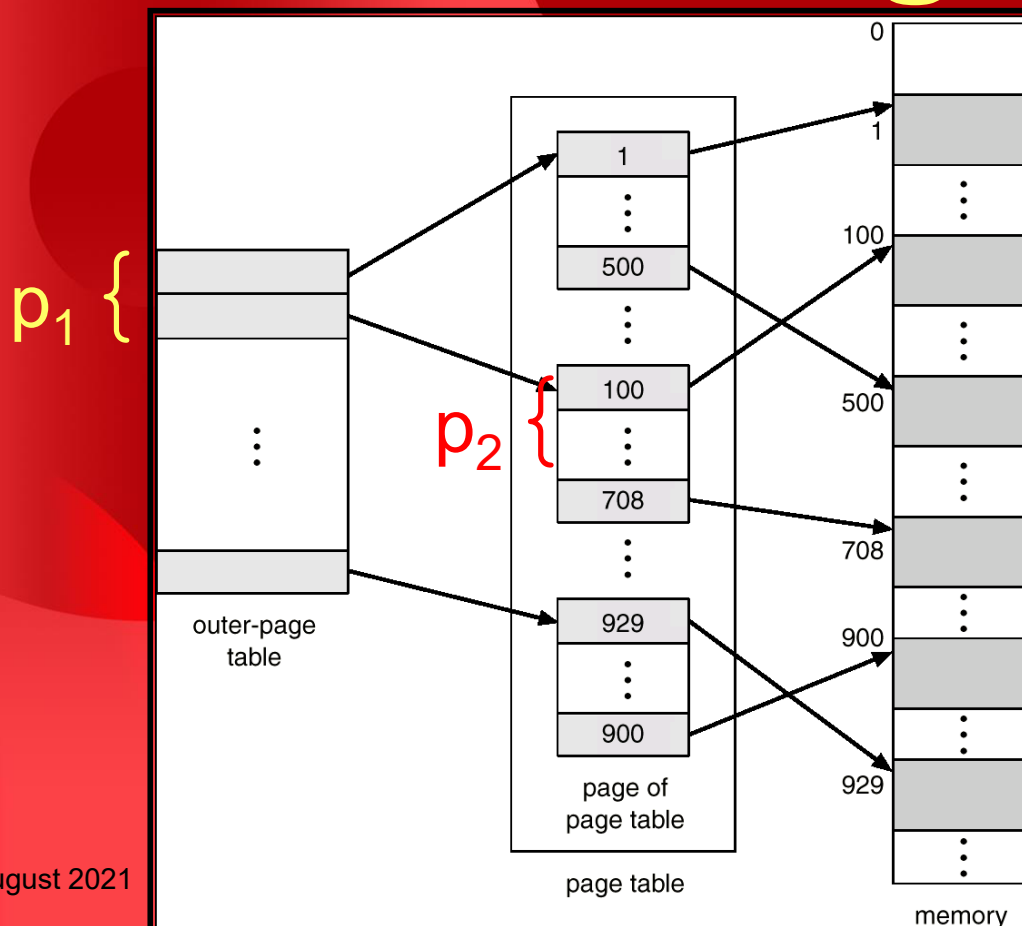
7 August 2021 10

© Copyright Virtual University of
Pakistan

12



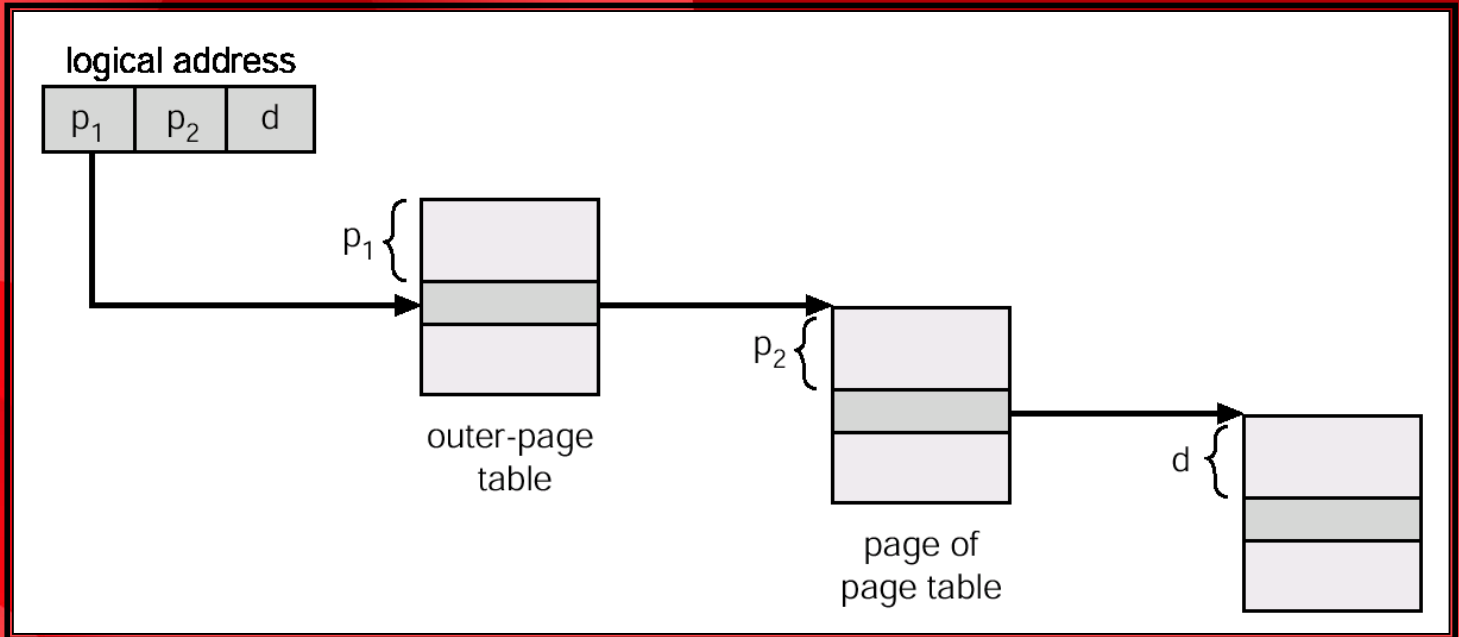
Multilevel Paging



7 August 2021



Multilevel Paging



7 August 2021

© Copyright Virtual University of
Pakistan



DEC VAX

- Logical address = 32 bits
- Page size = 512 bytes = 2^9 bytes
- Process address space divided into four equal sections
- Pages per section = $2^{30} / 2^9$
 $= 2^{21} = 2M$
- Size of a page table entry = 4 bytes

7 August 2021

© Copyright Virtual University of
Pakistan



DEC VAX

- Bits needed for page offset
 $= \log_2 5^{12} = 9$ bits
- Bits to specify a section $= \log_2 4 = 2$ bits
- Bits needed to index page table for a section $= \log_2 2^{21} = 21$ bits
- Size of a page table $= 2^{21} * 4 = 8 \text{ MB}$

7 August 2021

© Copyright Virtual University of
Pakistan



DEC VAX

- 8 MB page table is paged into 8MB / 512 = 2 K pages
- Size of the outer page table (2K * 4 = 8 KB) is further paged, resulting in 3-level paging per section

section page number page offset

s	p	d

7 August 2021

© Copyright Virtual University of
Pakistan



2

21

9

More Examples

- 32-bit Sun SPARC supports 3-level paging
- 32-bit Motorola 68030 supports 4-level paging
- 64-bit Sun UltraSPARC supports 7-level paging – too many memory references needed for address translation

7 August 2021

© Copyright Virtual University of
Pakistan



Hashed Page Table

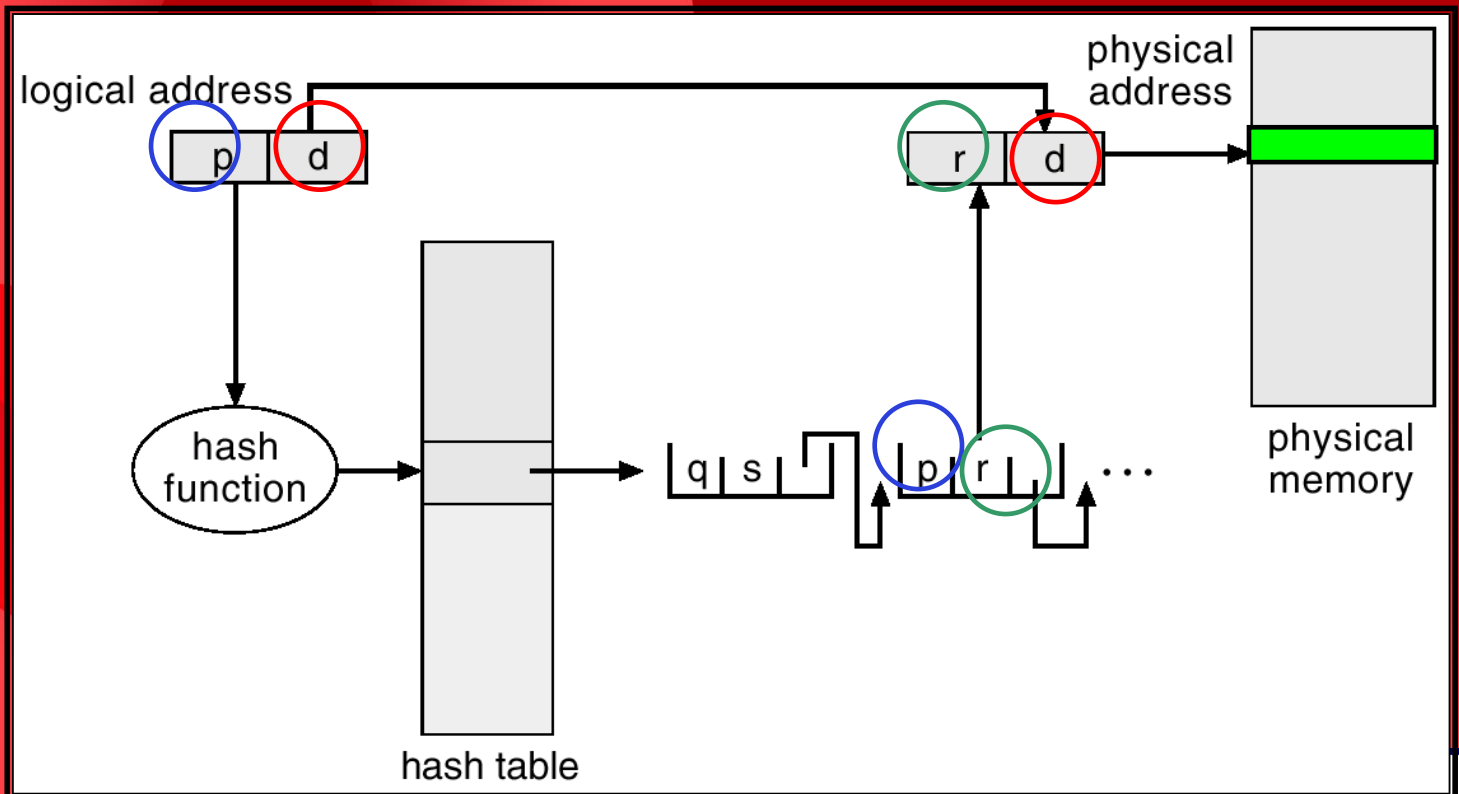
- A common approach to handle address spaces larger than 32 bits
- Usually open hashing is used
- Each entry in the linked list has three fields: page number, frame number for the page, and pointer to the next element – (p, f, next)

7 August 2021

© Copyright Virtual University of
Pakistan



Hashed Page Table



Inverted Page Table

- One entry per real page in memory
- Page table size is limited by the number of frames (i.e., the physical memory) and not process address space
- Each entry in the page table contains (pid, p)

Inverted Page Table

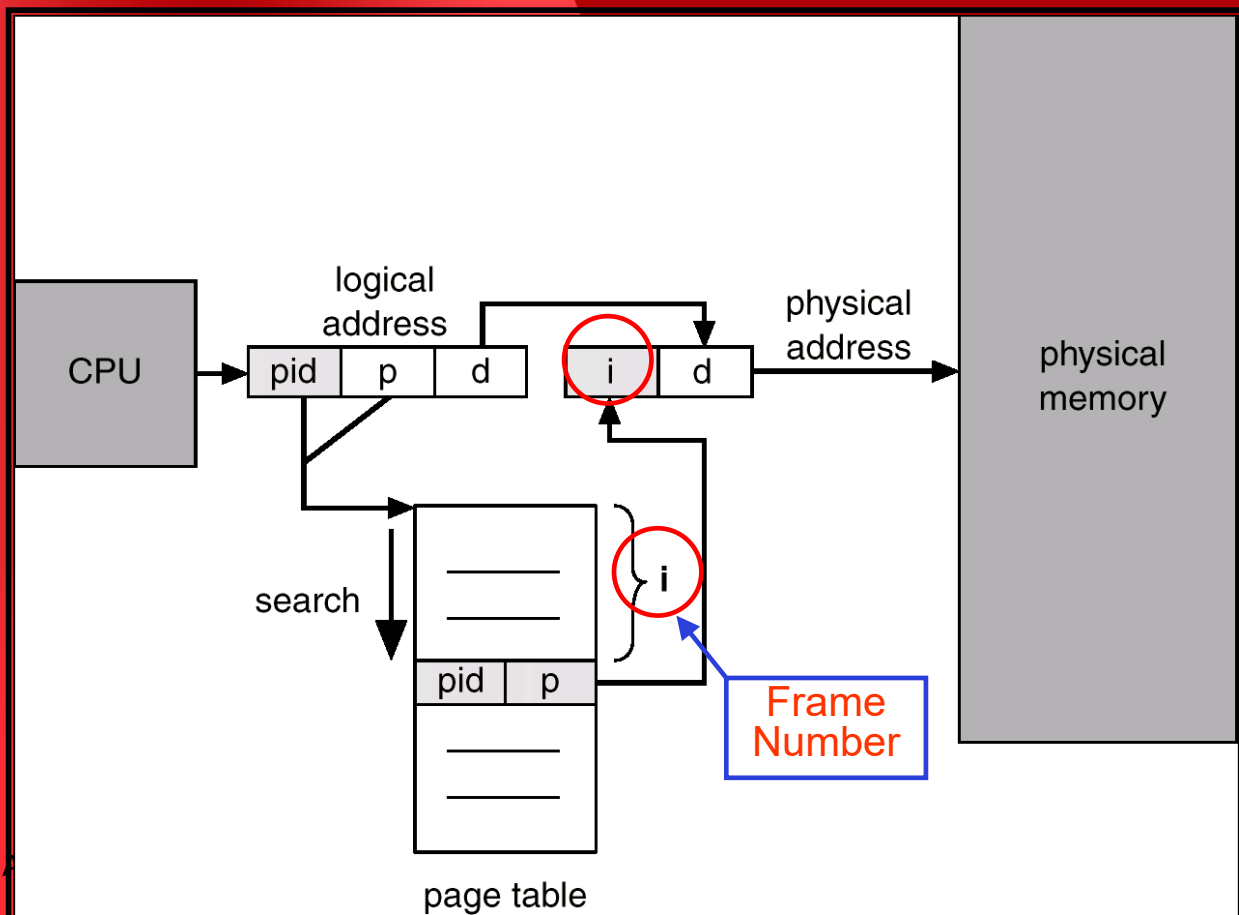
- If a page 'p' for a process is loaded in frame 'f', its entry is stored at index 'f' in the page table
- We effectively index the page table with frame number; hence the name inverted page table
- Examples: 64-bit UltraSPARC and PowerPC

7 August 2011

© Copyright Virtual University of
Pakistan



Inverted Page Table



7/8



Recap of Lecture

- Structure of the page table
 - Multi-level paging
 - Hashed page tables
 - Inverted page table

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 34

Syed Mansoor Sarwar

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 35

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Sharing in paging
- Segmentation
- Logical to physical address translation
- Hardware support needed
- Protection and sharing



Review of Lecture 34

- Performance of paging
- Protection in paging
- Structure of the page table
 - Multi-level paging
 - Hashed page tables
 - Inverted page table

7 August 2021

© Copyright Virtual University of
Pakistan



Shared Pages

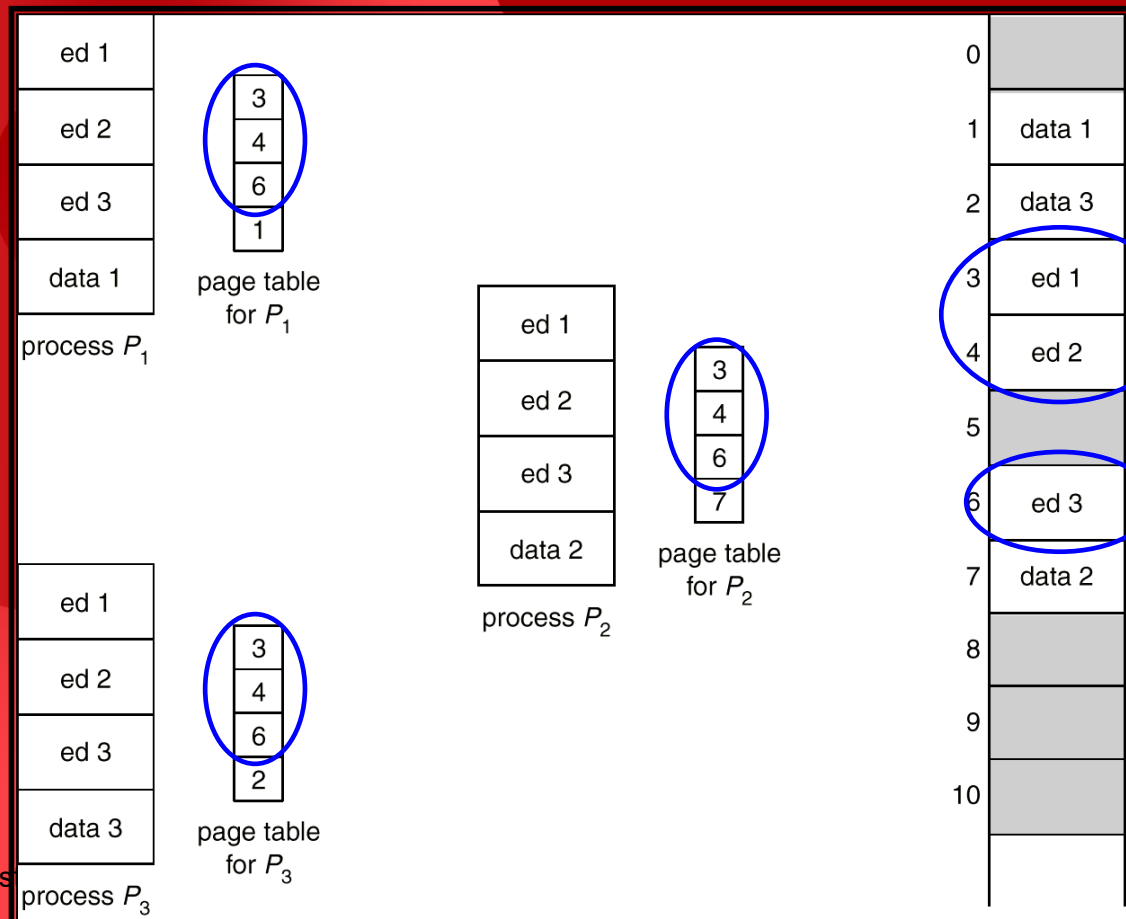
- Reentrant (read-only) code pages of a process address space can be shared
- Shared code appears in the same logical address space of all processes
- Multiple invocations of `pico` or `gcc`

7 August 2021

© Copyright Virtual University of
Pakistan



Shared Pages



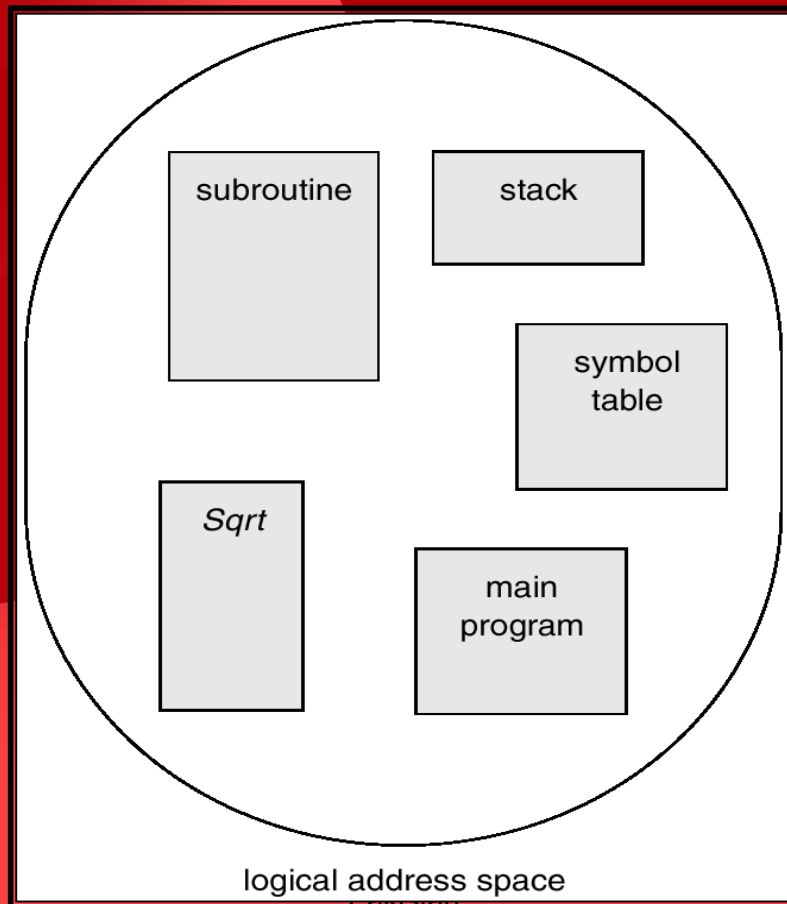
7 August 2021

Segmentation

- A memory management scheme that supports programmer's view of memory.
- A segment is a logical unit such as: main program, procedure, function, method, object, global variables, stack, symbol table
- A program is a collection of segments

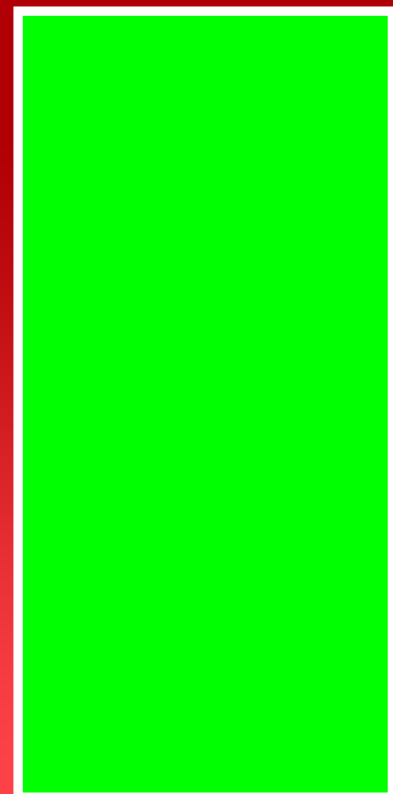
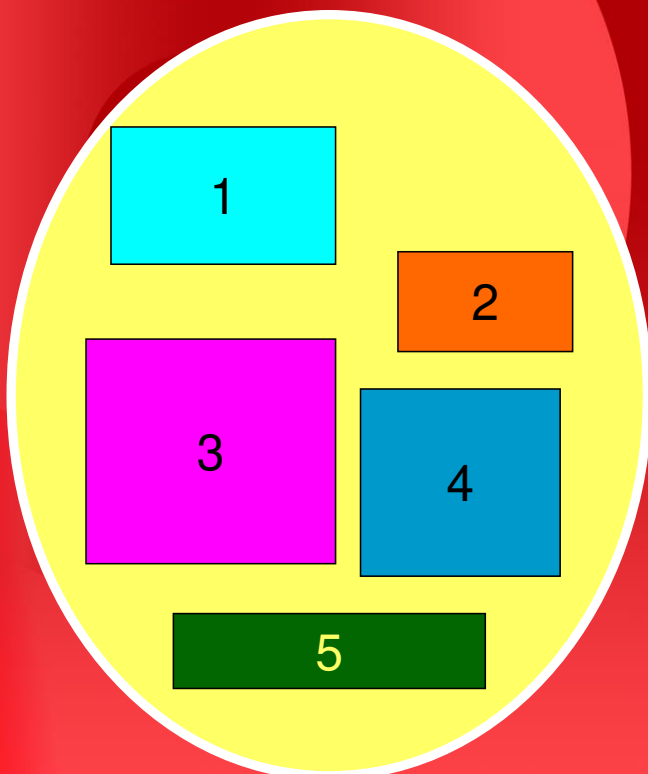
7 August 2021

Segmentation



7 August 2021

Segmentation



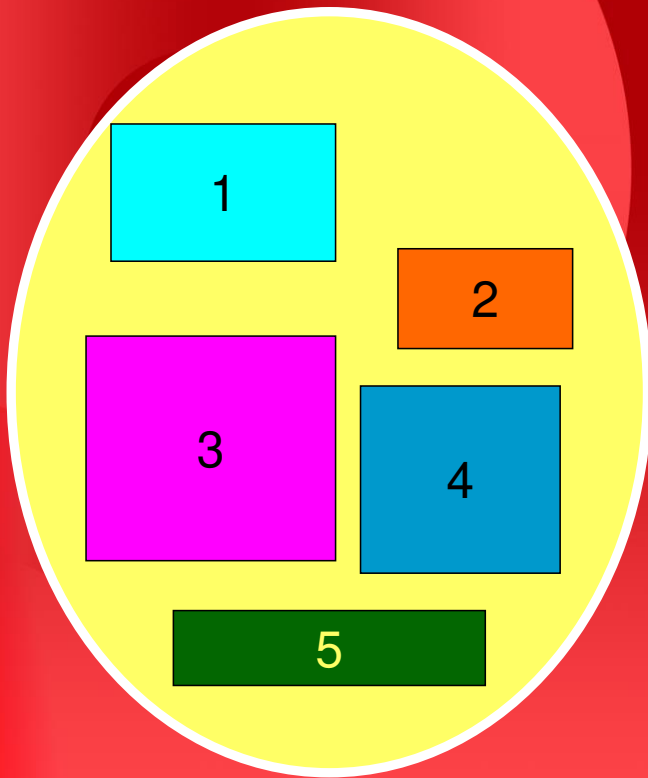
7 August 2021

logical memory

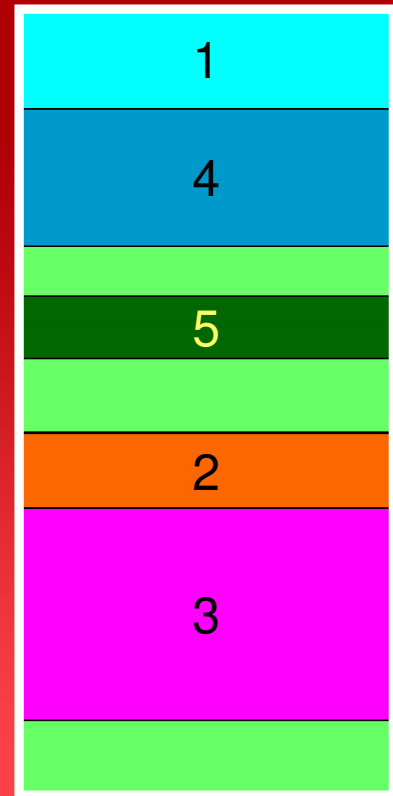
© Copyright Virtual University of
Pakistan

physical memory

Segmentation



logical memory



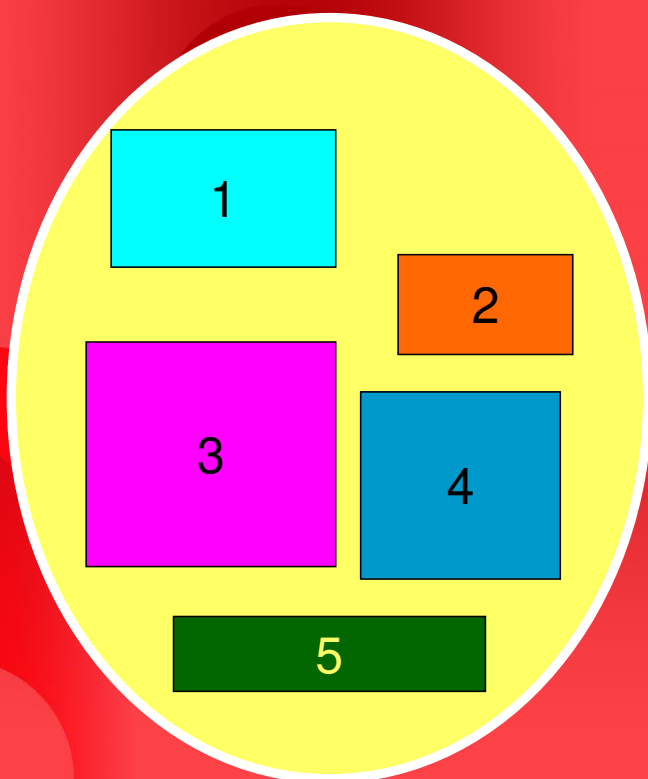
physical memory

7 August 2021

© Copyright Virtual University of
Pakistan



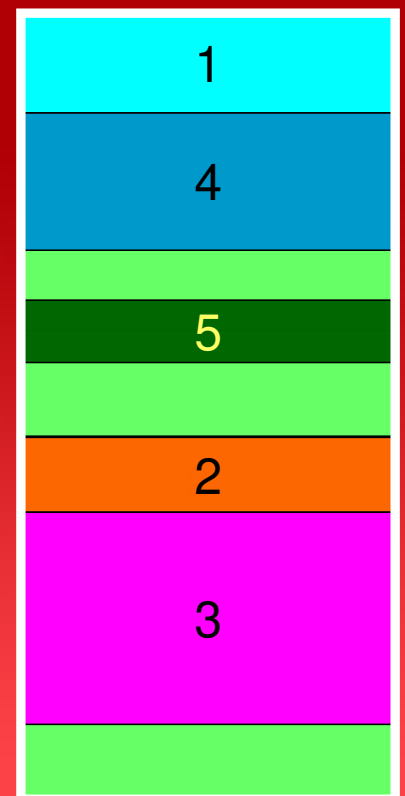
Segmentation



logical memory



segment
table



physical memory

7 August 2021

© Copyright Virtual University of
Pakistan



Segmentation

- Logical address consists of a two tuple:
 $\langle \text{segment-number, offset} \rangle$
- Segment table – maps two-dimensional logical addresses to physical addresses

7 August 2021

© Copyright Virtual University of
Pakistan



Segmentation

- Each segment table entry has:
 - base – contains the starting physical address where the segments reside in memory.
 - limit – specifies the length of the segment.

7 August 2021

© Copyright Virtual University of
Pakistan



Segmentation

- Segment-table base register (STBR) points to the segment table's location in memory.
- Segment-table length register (STLR) indicates number of segments used by a program
- Segment number **s** is legal if

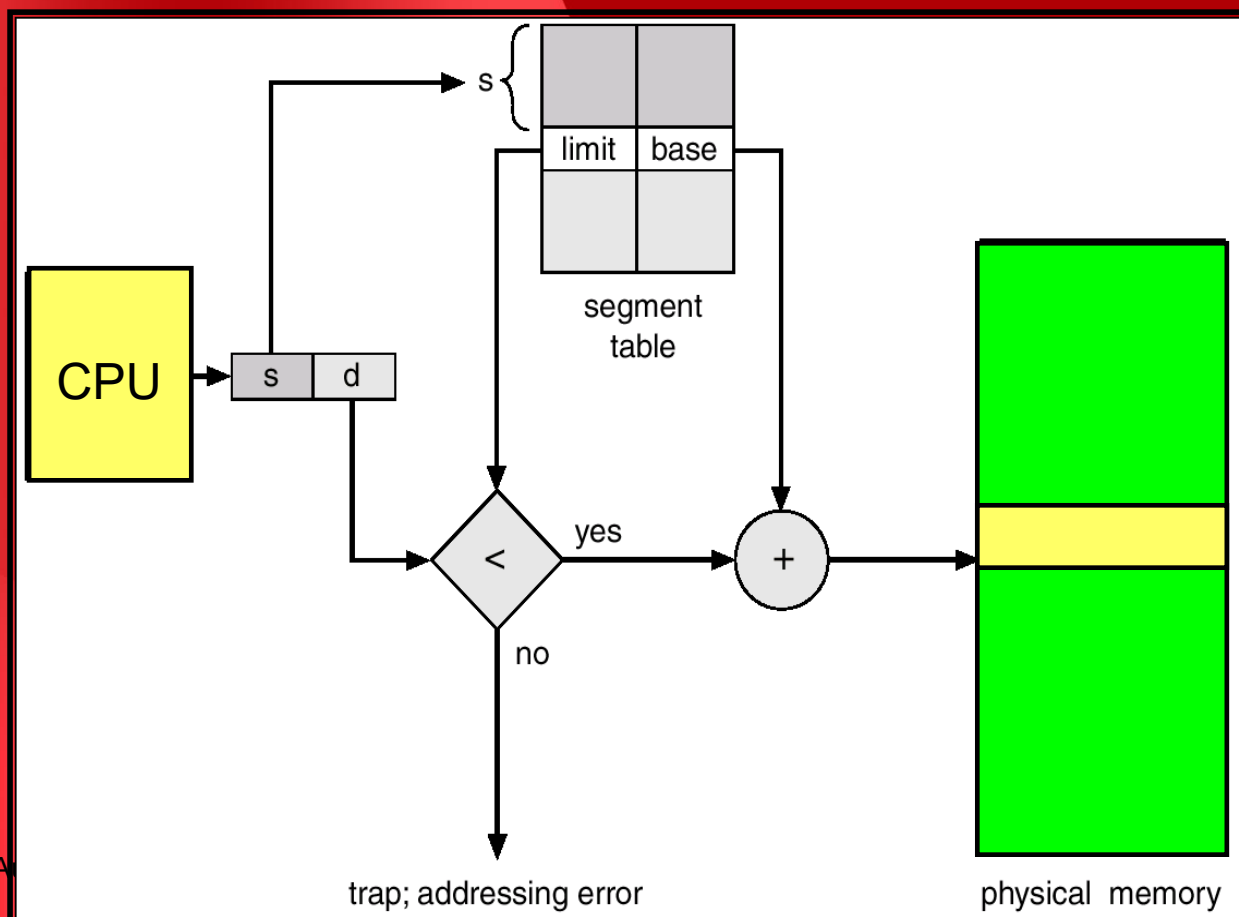
$$s < \text{STLR}$$

7 August 2021

© Copyright Virtual University of
Pakistan



Segmentation



7 A



Segmentation Architecture

- Relocation
 - Dynamic
 - By segment table
- Sharing
 - Shared segments
 - Same segment number

7 August 2021

© Copyright Virtual University of
Pakistan



Segmentation Architecture

- Dynamic Storage Allocation
 - First fit
 - Best fit
 - Worst fit
 - External fragmentation

7 August 2021

© Copyright Virtual University of
Pakistan



Segmentation Architecture

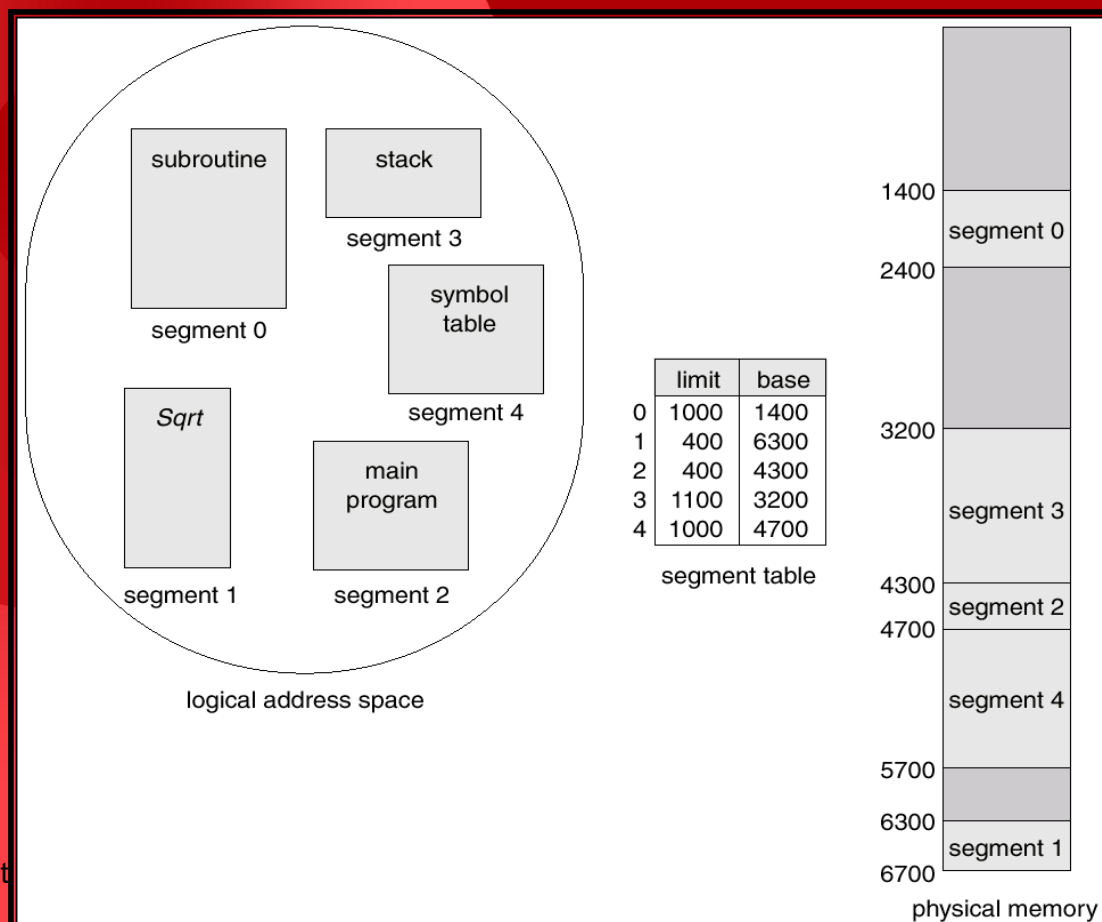
- Protection: Bits are associated for this purpose with each entry in segment table:
 - Validation bit = 0 $\Rightarrow$ illegal segment
 - Read, write, execute bits

7 August 2021

© Copyright Virtual University of Pakistan



Example



7 August



Address Translation

- Logical and Physical Addresses
 - (2, 399) – PA: $4300 + 399 = 4699$
 - (4, 0) – PA: $4700 + 0 = 4700$
 - (4, 1000) $\Rightarrow$ trap
 - (3, 1300) $\Rightarrow$ trap
 - (6, 297) $\Rightarrow$ trap

7 August 2021

© Copyright Virtual University of
Pakistan



Sharing of Segments

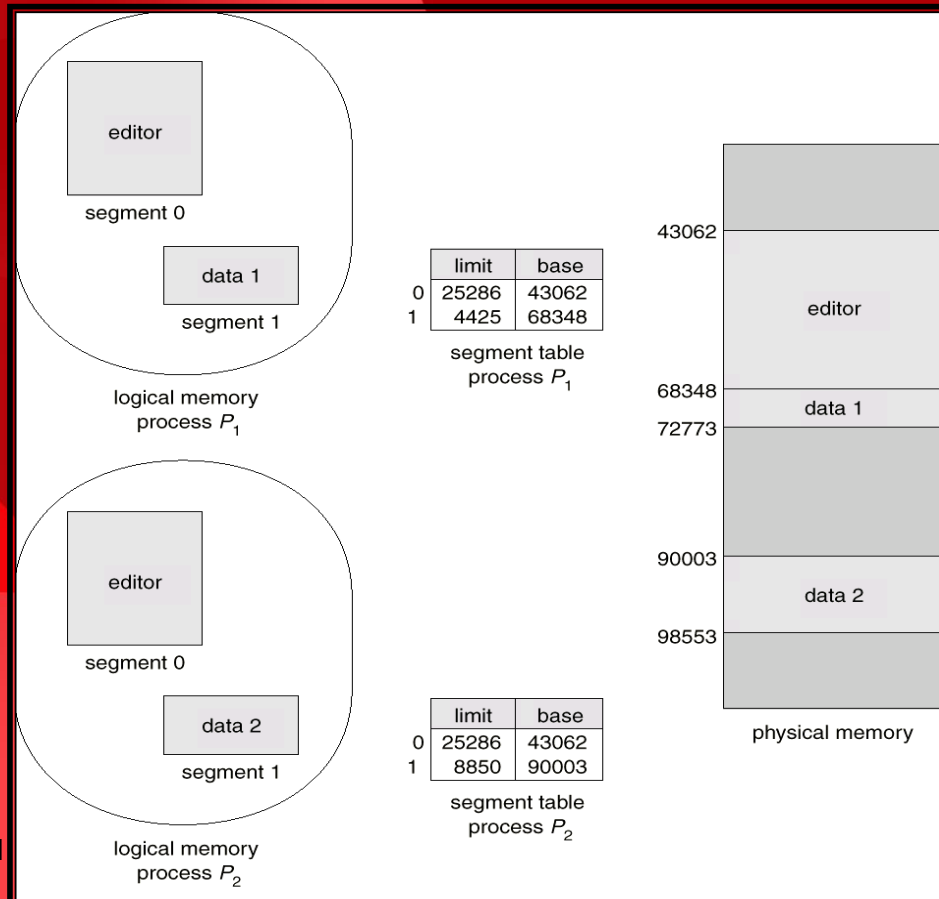
- Sharing at the segment level and not at the fixed-size page level
- Sharing at the code or data level
- Segment table of multiple processes point to the same segment

7 August 2021

© Copyright Virtual University of
Pakistan



Sharing of Segments



7 August 2021

Issues with Segmentation

■ External Fragmentation

Total memory space exists to satisfy a space allocation request for a segment, but memory space is not contiguous.

7 August 2021

Issues with Segmentation

- Reduce external fragmentation by compaction
 - Shuffle segments to place free memory together in one block.
 - Compaction is possible *only* if relocation is dynamic, and is done at execution time.

7 August 2021

© Copyright Virtual University of
Pakistan



Issues with Segmentation

- I/O problem
 - Latch job in memory while it is involved in I/O.
 - Do I/O only into OS buffers
- Very large segments $\Rightarrow$ page program segments—paged segmentation

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Sharing in paging
- Segmentation
- Logical to physical address translation
- Hardware support needed
- Protection and sharing
- Issues with segmentation

August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 35

Syed Mansoor Sarwar

Operating Systems

Lecture 36

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Paged segmentation
- Examples of paged segmentation: MULTICS under GE 345 and OS/2, Windows, and Linux under Intel CPUs

Review of Lecture 35

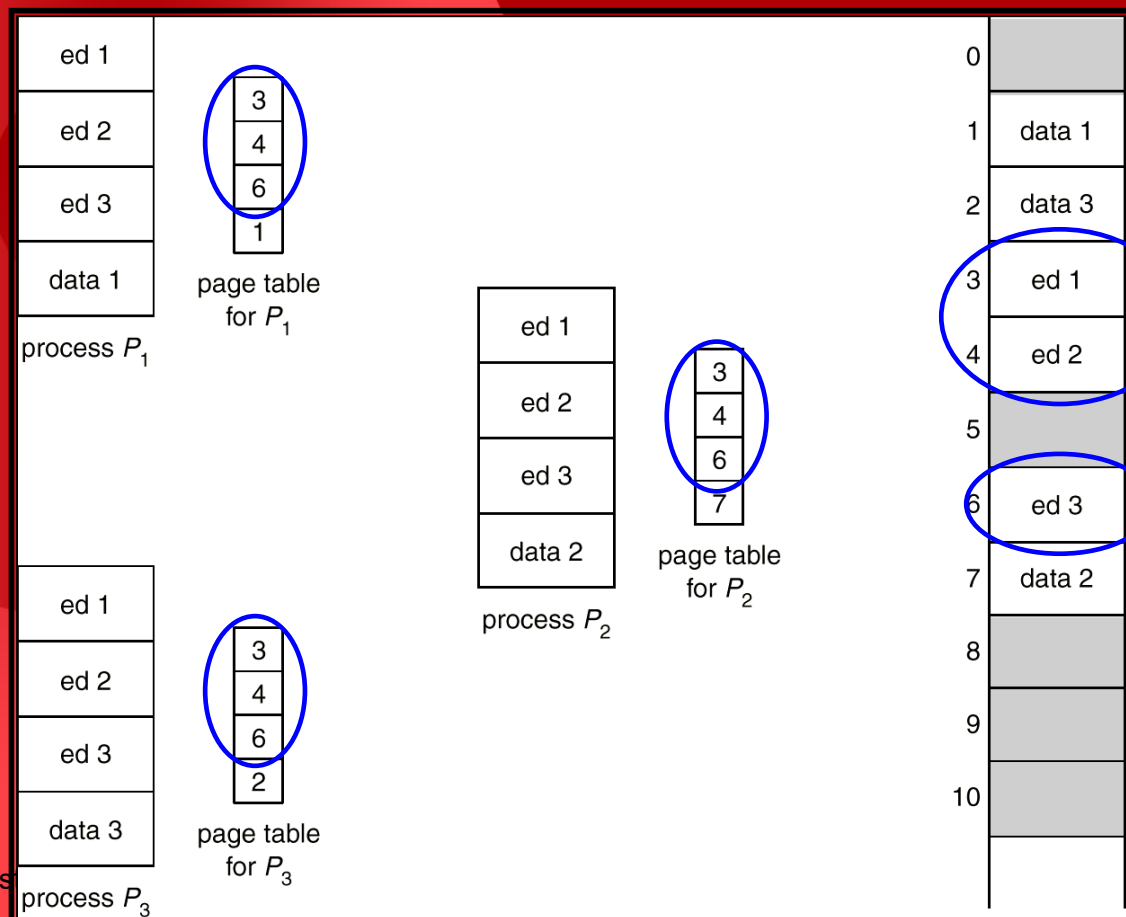
- Sharing in paging
- Segmentation
- Logical to physical address translation
- Hardware support needed
- Protection and sharing
- Issues with segmentation

7 August 2021

© Copyright Virtual University of
Pakistan



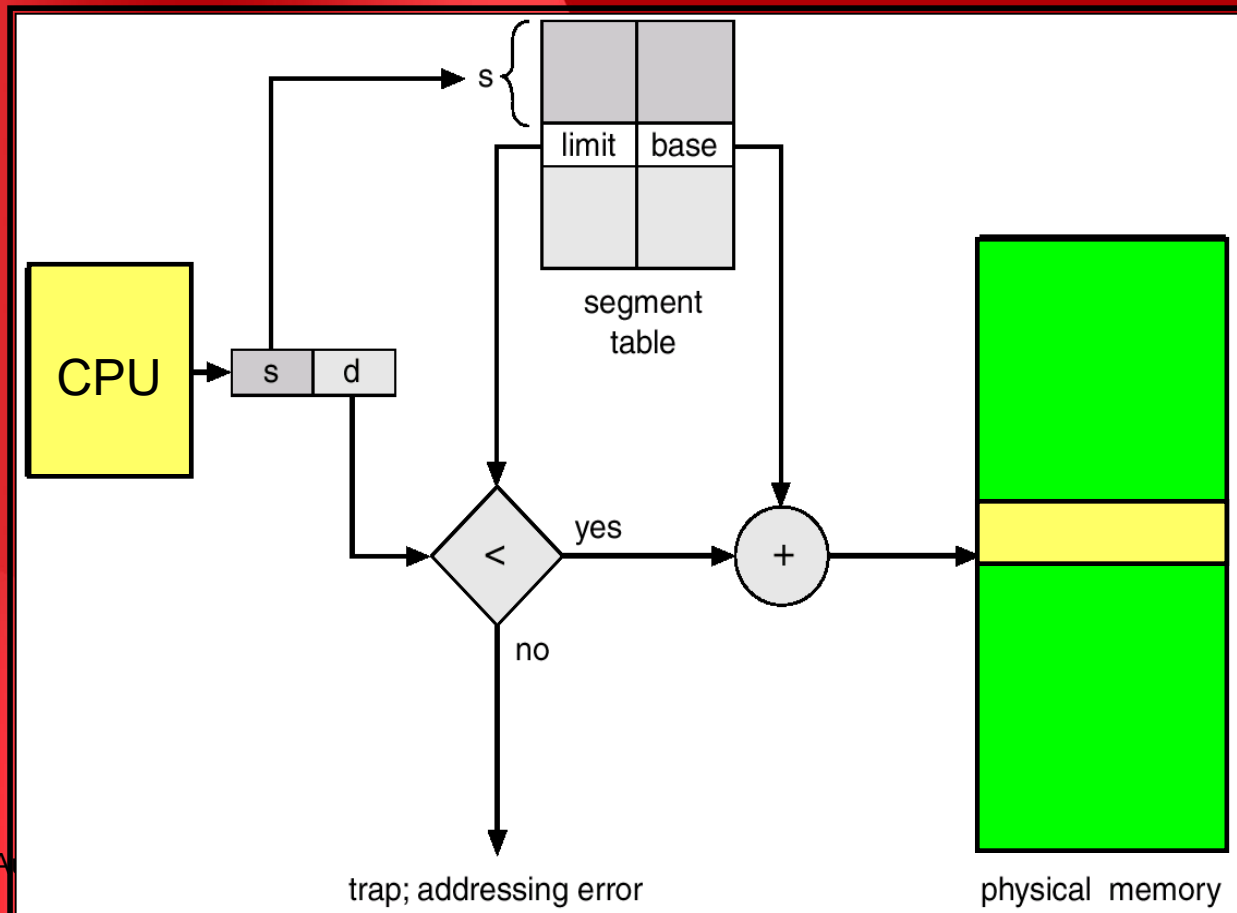
Shared Pages



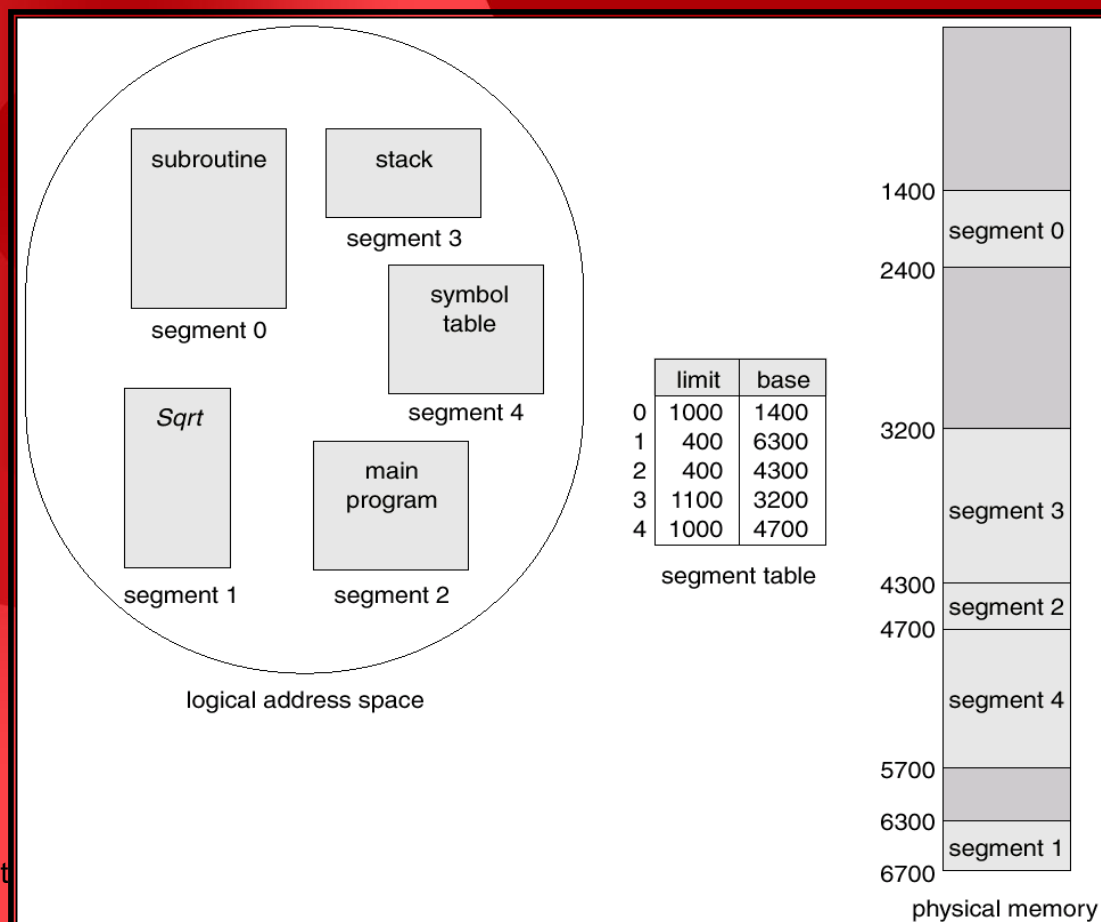
7 August



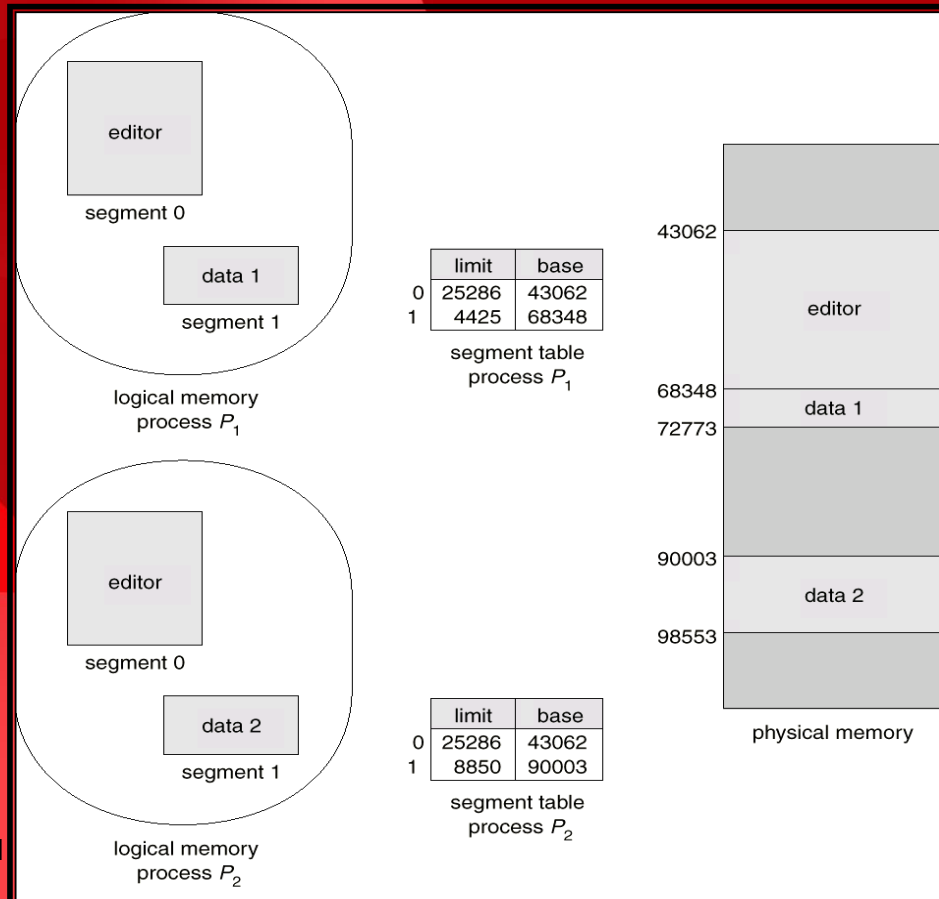
Segmentation



Example



Sharing of Segments



7 August 2021

Protection

- Associate valid/invalid bit with each segment table entry to indicate if the referenced segment is part of the process address space or not
- Read, write, and execute bits to define legal operations on a segment

7 August 2021

Paged Segmentation

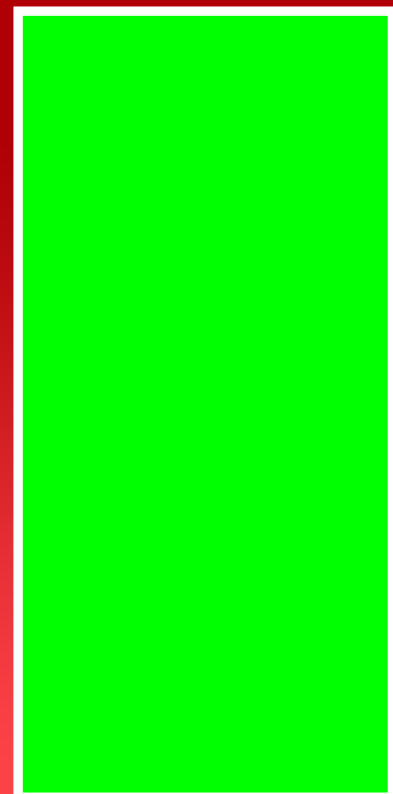
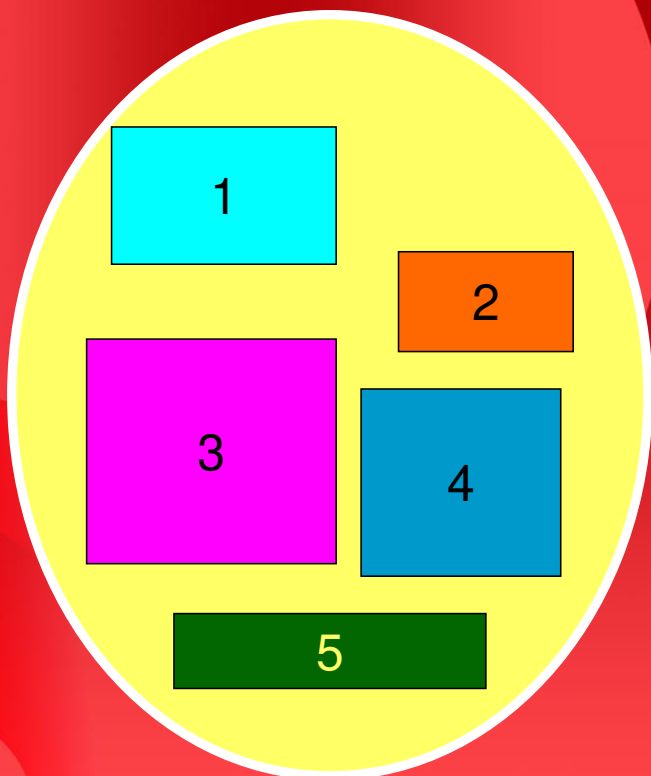
- Divide every segment in a process into fixed size pages
- Need for a page table per segment
- CPU's memory management unit must support both segmentation and paging

7 August 2021

© Copyright Virtual University of
Pakistan



Paged Segmentation

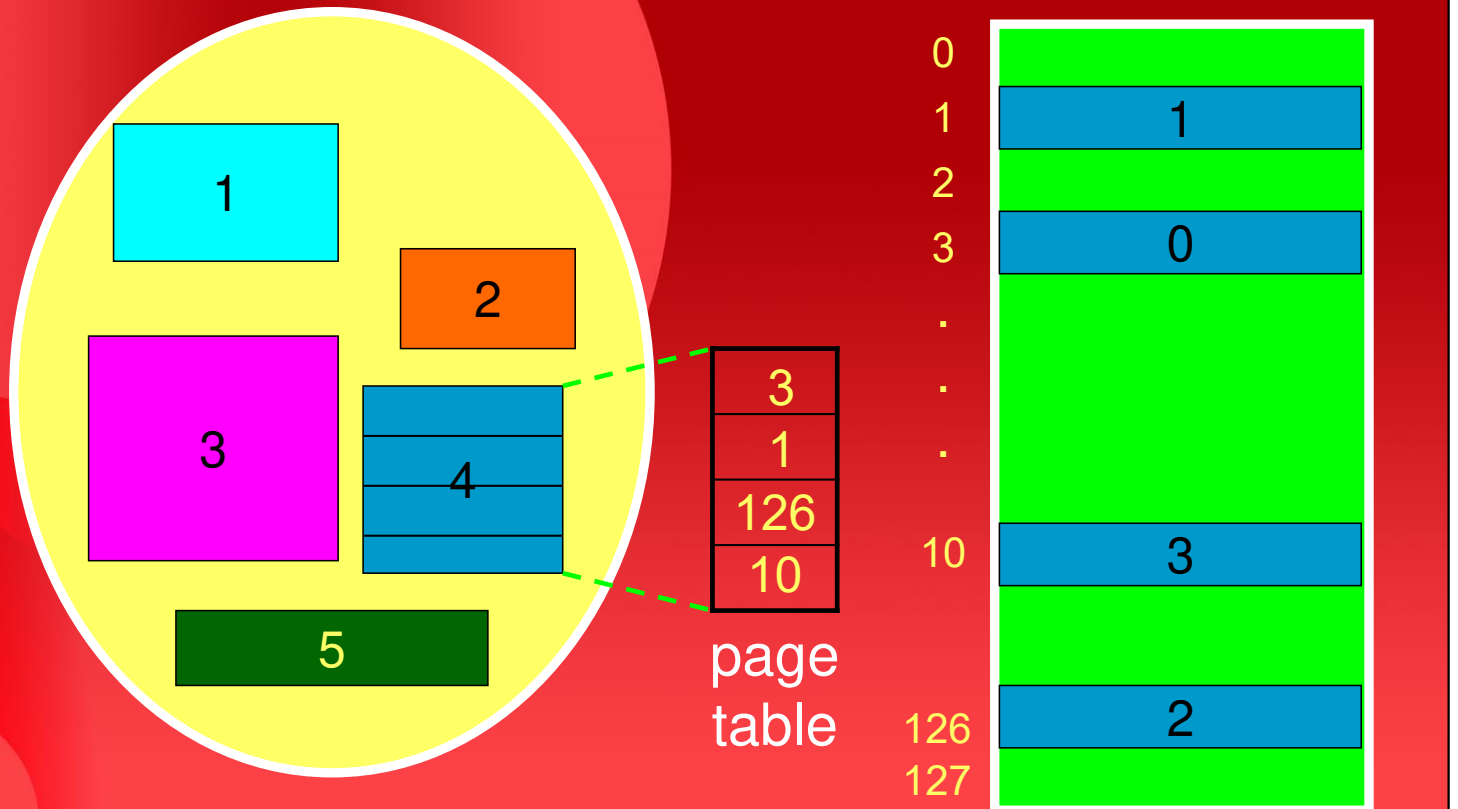


7 August 2021

© Copyright Virtual University of
Pakistan



Paged Segmentation



7 August 2021

© Copyright Virtual University of Pakistan

physical memory



Paged Segmentation

- Logical address is still $\langle s, d \rangle$, with s used to index the segment table
- Each segment table entry consist of the tuple $\langle \text{segment-length}, \text{page-table-base} \rangle$
- The logical address is legal if $d < \text{segment-length}$

7 August 2021

© Copyright Virtual University of Pakistan



Paged Segmentation

- Segment offset, d , is partitioned into two parts: p and d' , where p is used to index the page table associated with segment, s , and d' is used as offset within a page

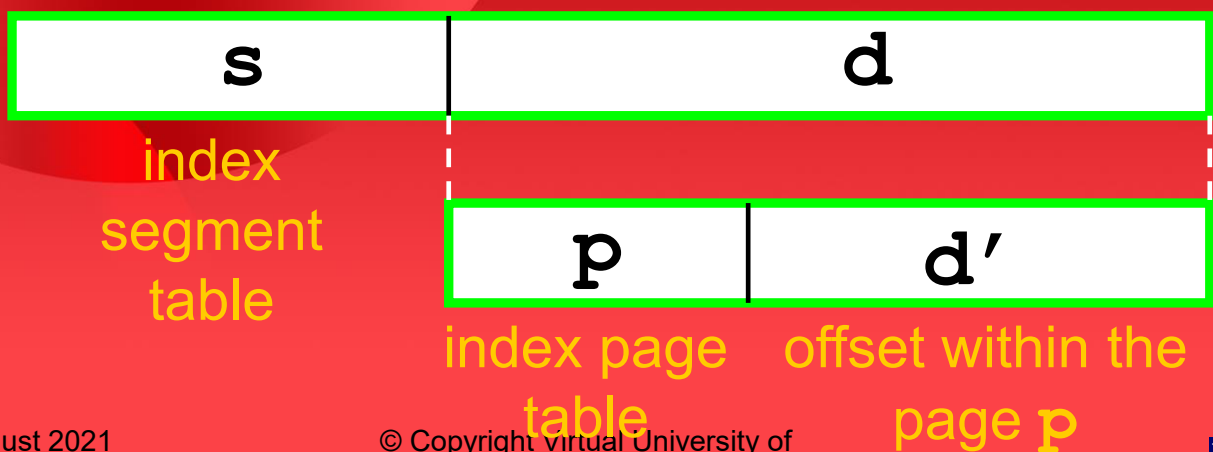
7 August 2021

© Copyright Virtual University of Pakistan



Paged Segmentation

- p indexes the page table to retrieve frame, f , and physical address (f, d') is formed

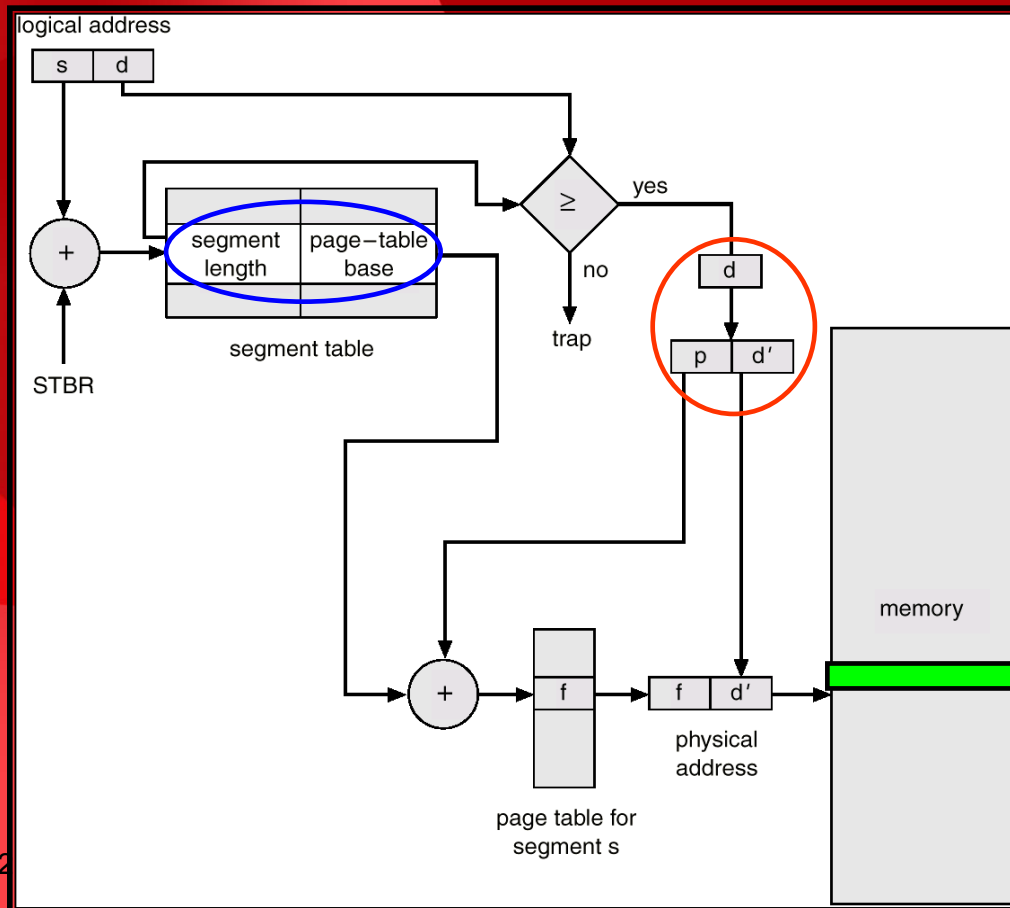


7 August 2021

© Copyright virtual University of Pakistan



Paged Segmentation



7 August 2021



MULTICS Example

- GE 345 processor
- Logical address = 34 bits
- Page size = 1 KB
- s is 18 bits and d is 16 bits
- Size of p and d' , largest segment size, and max. number of segments per process?

7 August 2021

Copyright Virtual University of
Pakistan



MULTICS Example

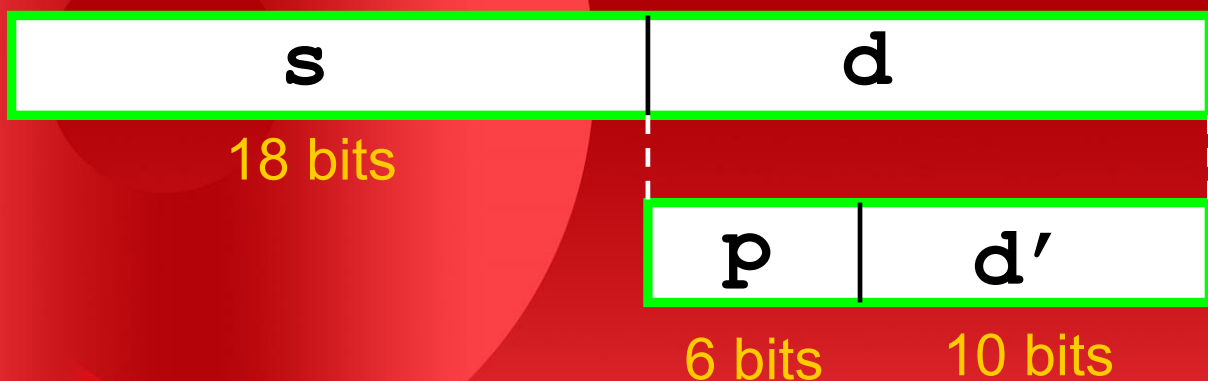
- Largest segment = 2^d bytes
= 2^{16} bytes
- Maximum number of pages per segment = $2^{16} / 1 \text{ K} = 64$
- $|p| = \log_2 64 \text{ bits} = 6 \text{ bits}$
- $|d'| = \log_2 1 \text{ K} = 10 \text{ bits}$
- Maximum number of segments per process = $2^s = 2^{18}$

7 August 2021

© Copyright Virtual University of
Pakistan



MULTICS Example



MULTICS Example

- Consider a process with its segment 15 having 5096 bytes. The process generates a logical address (15,3921).
- Is it a legal address?
- How many pages does the segment have?
- What page does the logical address refer to?

7 August 2021

© Copyright Virtual University of
Pakistan



MULTICS Example

- Is it a legal address? **Yes**
- How many pages does the segment have?

$$\text{ceiling}[5096/1024] = 5$$

- What page does the logical address refers to?

$$\text{ceiling}[3921/1024] = 4$$

(i.e., page number 3)

7 August 2021

© Copyright Virtual University of
Pakistan



MULTICS Example

- What are the value of d' and the physical address if page number 3 (i.e., the fourth page) is in frame 12?

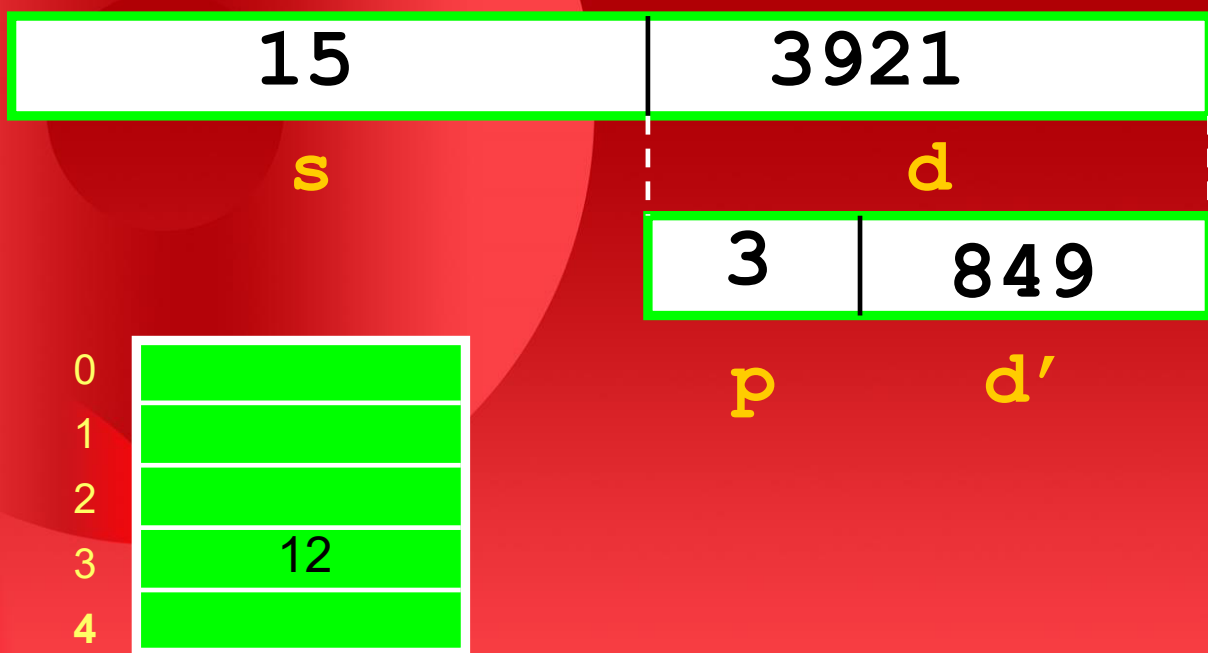
$$d' = 3921 - 3 \times 1K = 849$$

$$\begin{aligned}\text{Physical address} &= 12 \times 1K + 849 \\ &= 13137\end{aligned}$$

Pakistan

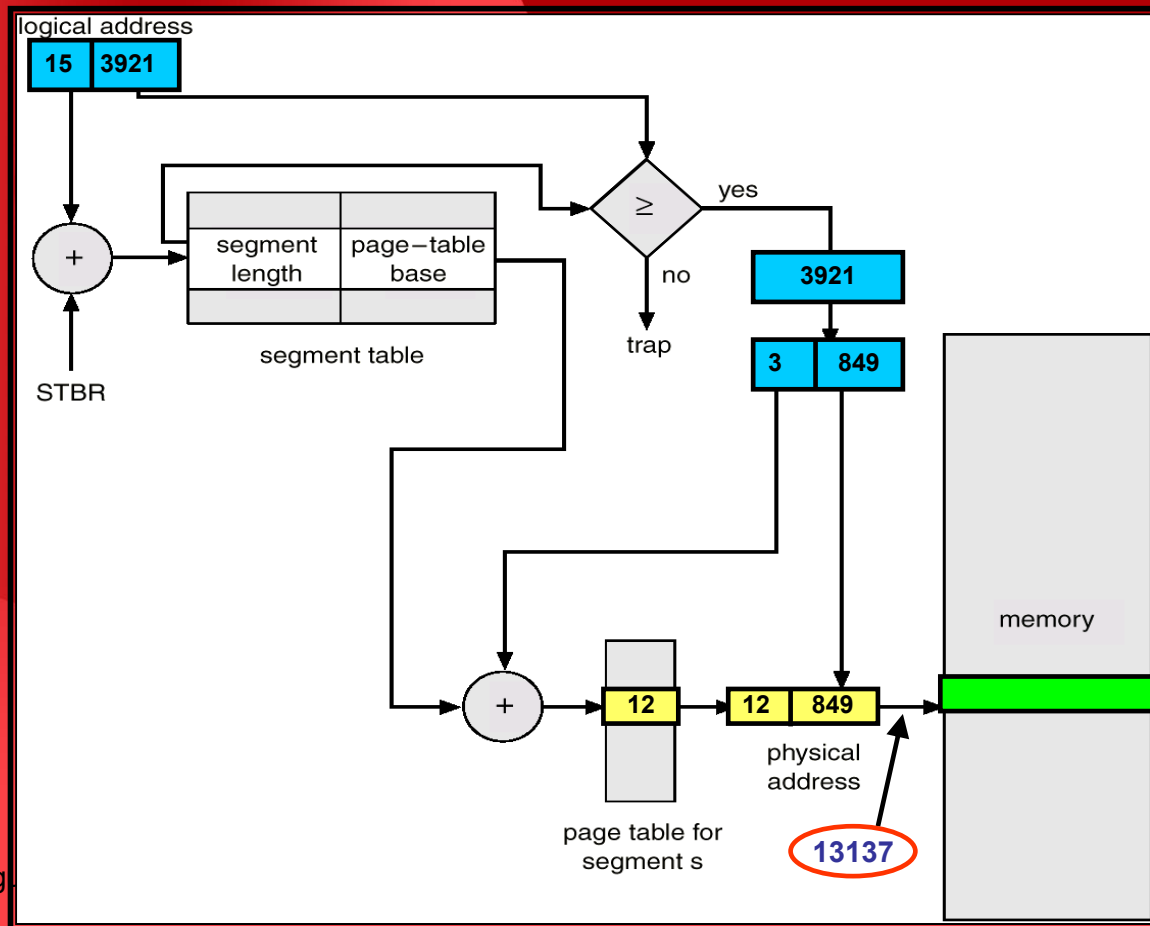
VUI
Virtual University

MULTICS Example



page table for
segment 15

MULTICS Example



7 Aug



Intel 80386 Example

- IBM OS/2, Microsoft Windows, and Linux
- Paged segmentation with two-level paging
- Logical address = 48 bits
- 16-bit selector and 32-bit offset
- Page size = 4 KB

7 August 2021

© Copyright Virtual University of
Pakistan



Intel 80386 Example

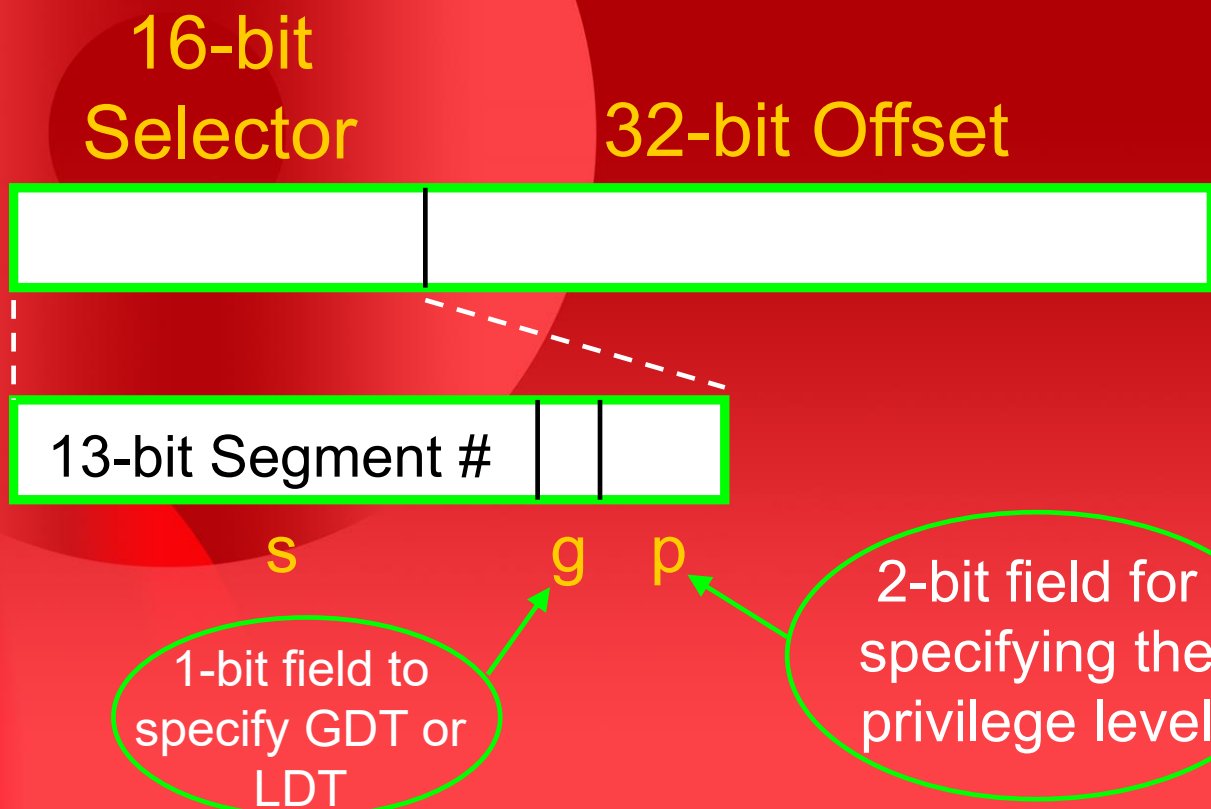
- 4-byte page table entry
- 32-entry TLB, covering $32 \times 4\text{K}$ (128 KB) memory ... TLB Reach

7 August 2021

© Copyright Virtual University of
Pakistan



Intel 80386 Example



7 August 2021

© Copyright Virtual University of
Pakistan



Intel 80386 Example

■ Real Mode

20-bit physical address is obtained by shifting left the Selector value by four bits and adding to it the 16-bit effective address

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Paged segmentation
- Examples of paged segmentation: MULTICS under GE 345 and OS/2, Windows, and Linux under Intel CPUs

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 36

Syed Mansoor Sarwar

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 37

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Memory Management in Intel 80386
- Virtual Memory
- Demand Paging
- Page Fault
- Performance of Demand Paging

7 August 2021

■ Process Creation

© Copyright Virtual University of
Pakistan



Review of Lecture 36

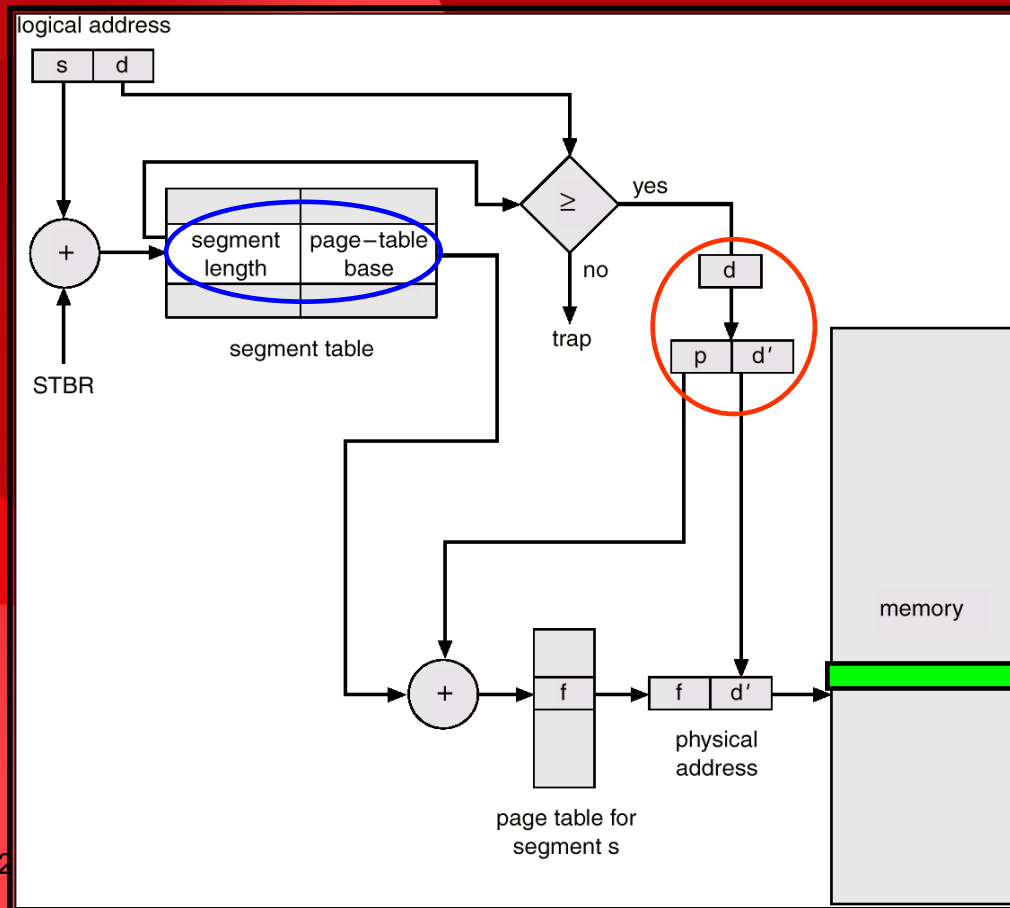
- Paged segmentation
- Examples of paged segmentation: MULTICS under GE 345 and OS/2, Windows, and Linux under Intel CPUs

7 August 2021

© Copyright Virtual University of
Pakistan

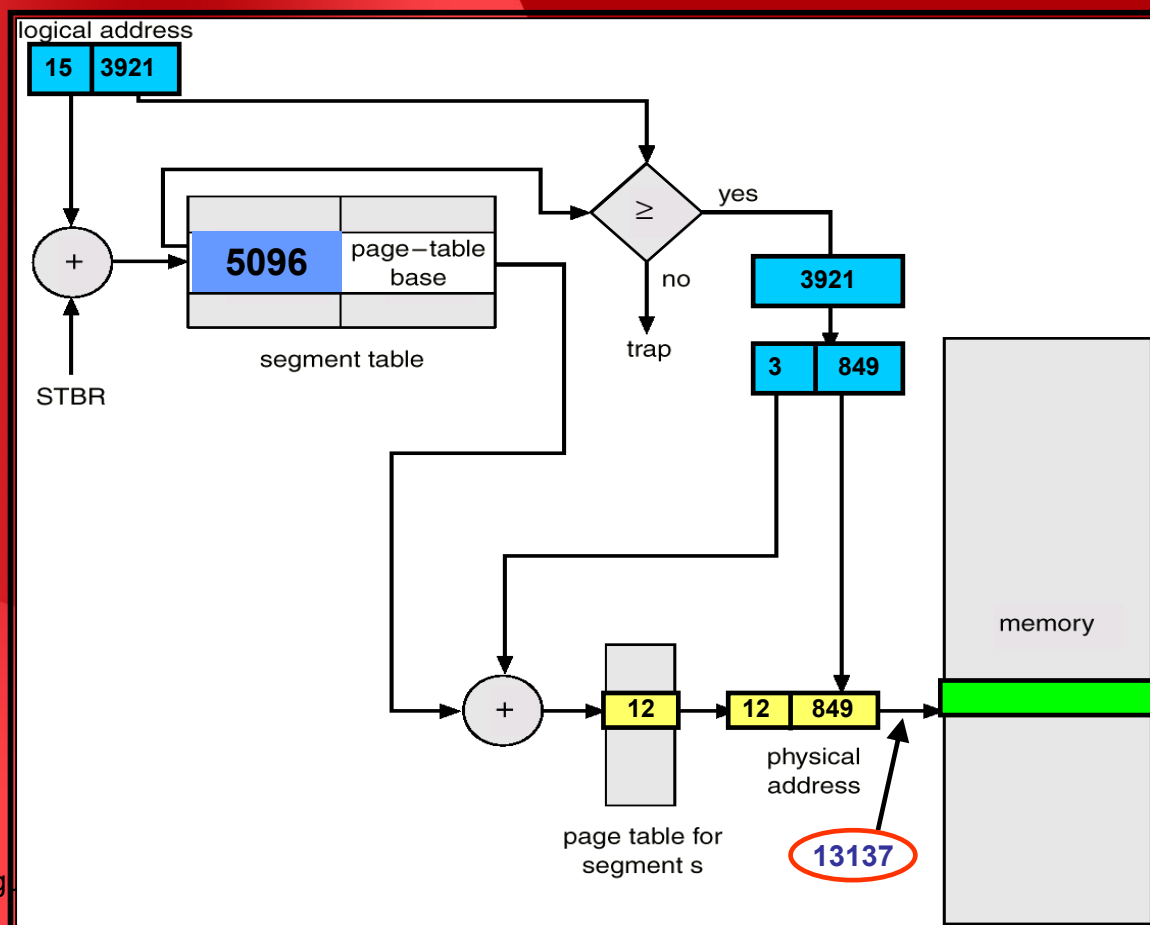


Paged Segmentation



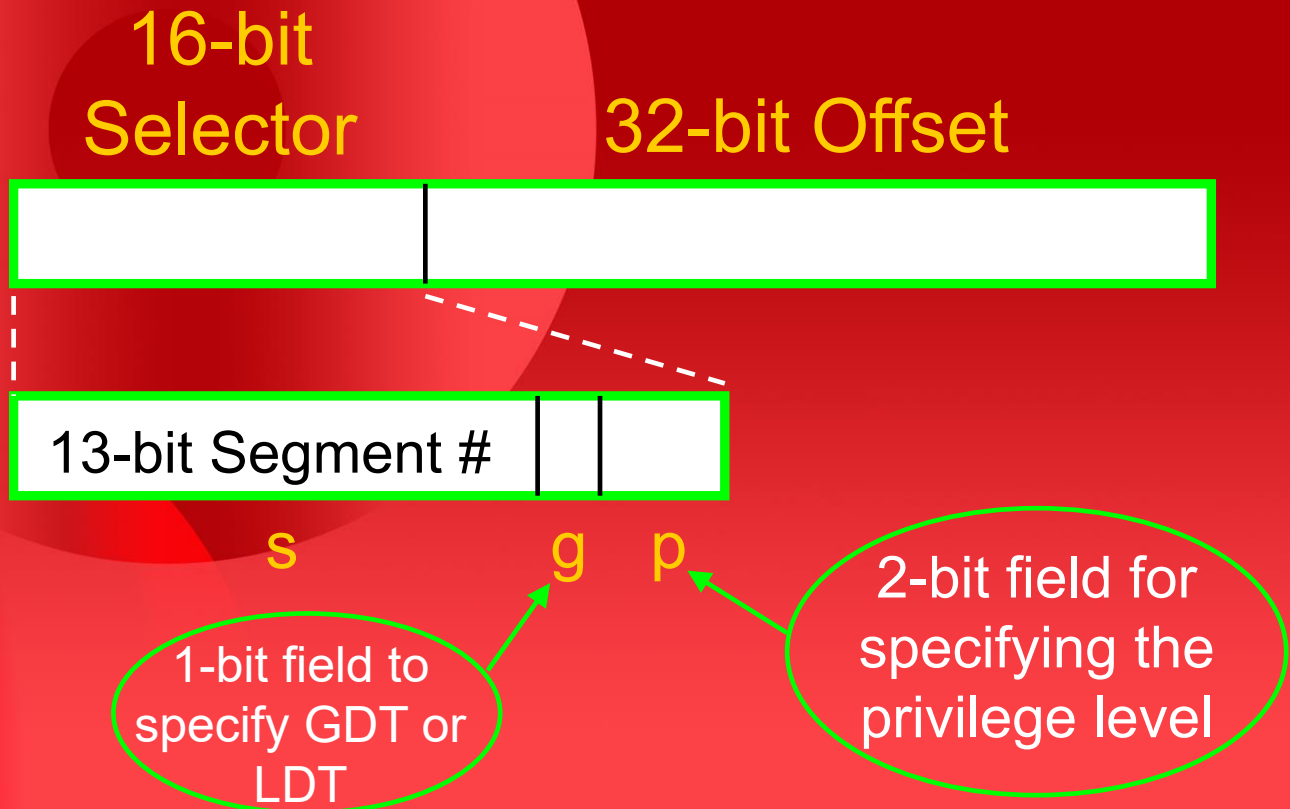
7 August 202

MULTICS Example



7 Aug

Intel 80386 Example



7 August 2021

© Copyright Virtual University of
Pakistan



Intel 80386 Example

■ Protected Mode

- 2^{48} bytes virtual address space
- 2^{32} bytes linear address space
- Max segment size = 4 GB
- Max segments / process = 16K
- Six CPU registers allow access to six segments at a time

7 August 2021

© Copyright Virtual University of
Pakistan



Intel 80386 Example

▪ Protected Mode

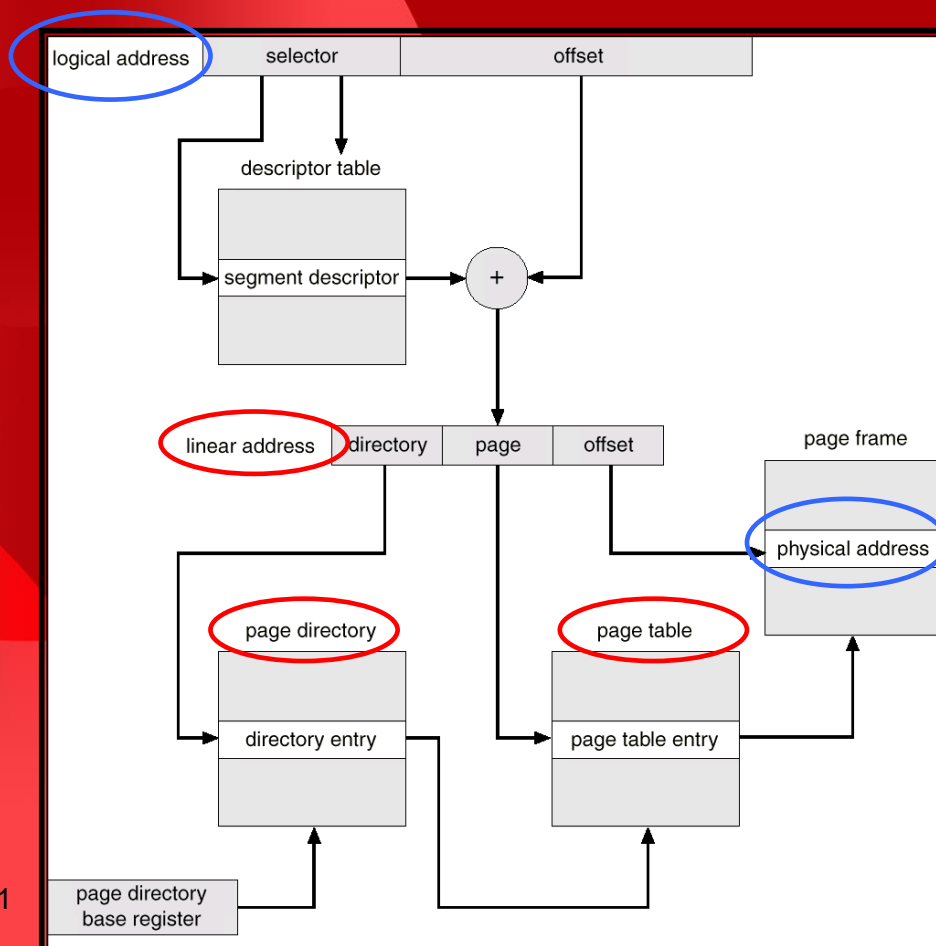
Selector is used to index a segment descriptor table to obtain an 8-byte segment descriptor entry. Base address and offset are added to get a 32-bit linear address, which is partitioned into **p1**, **p2**, and **d** for supporting 2-level paging.

7 August 2021

© Copyright Virtual University of
Pakistan



Intel 80386 Example



7 August 2021



Fundamentals

- **Virtual Memory:** A technique that allows a process to execute in the main memory space which is smaller than the process size
- Only a part of a process needs to be loaded in the main memory for execution

7 August 2021

© Copyright Virtual University of
Pakistan



Fundamentals

- Sharing of address space among several processes
- Efficient process creation—`fork()` and `vfork()`
- Memory mapped files
- Support needed for virtual memory

7 August 2021

© Copyright Virtual University of
Pakistan



Fundamentals

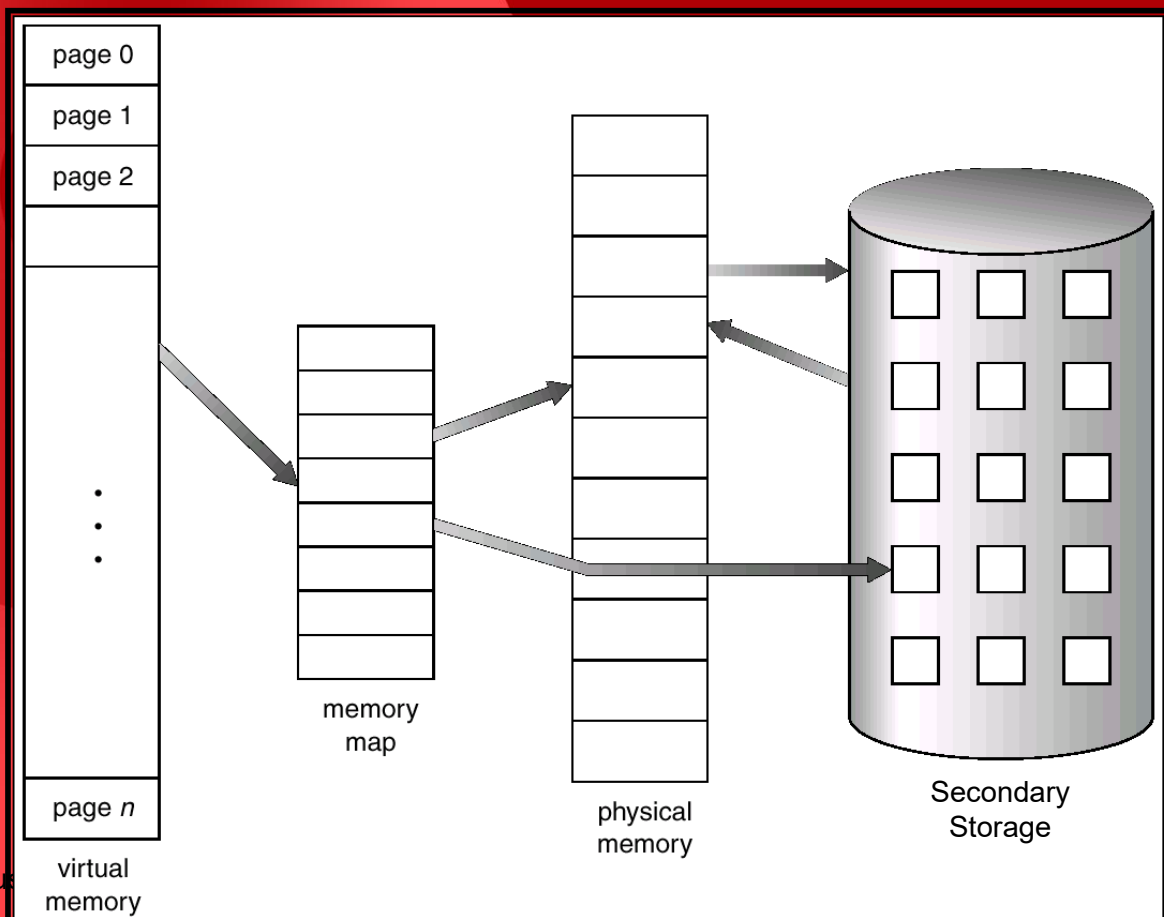
- Virtual memory can be implemented via:
 - Demand paging
 - Demand segmentation

7 August 2021

© Copyright Virtual University of
Pakistan



The Basic Idea



7 August



Demand Paging

- Bring a page into memory only when it is needed
 - Potentially less I/O needed
 - Potentially less memory needed
 - Faster response
 - Higher degree of multiprogramming

7 August 2011

© Copyright Virtual University of
Pakistan



Demand Paging

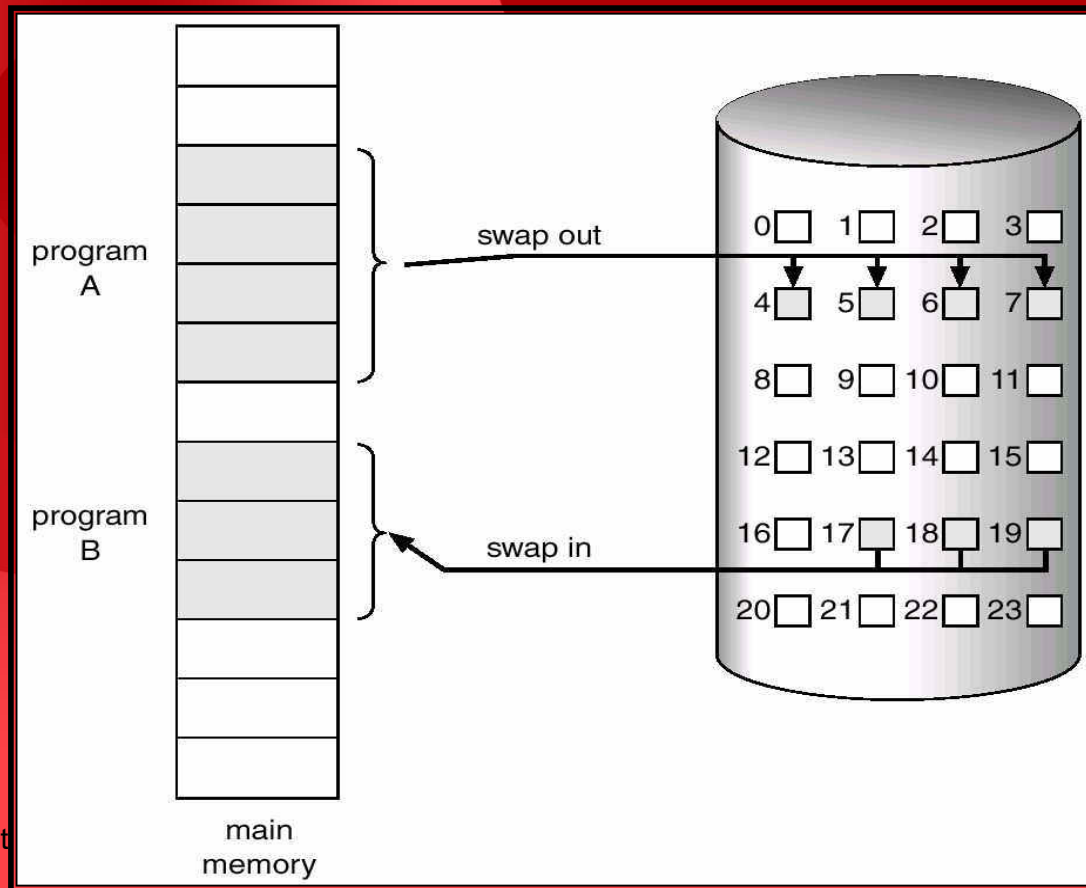
- Page is needed $\Rightarrow$ a reference is made to it
 - Invalid reference $\Rightarrow$ abort
 - Not-in-memory
 - $\Rightarrow$ **page fault**
 - $\Rightarrow$ bring page to memory
 - No free frame $\Rightarrow$ **swapping (out and in)**

7 August 2011

© Copyright Virtual University of
Pakistan



Swapping



7 August

Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (1 $\Rightarrow$ in-memory, 0 $\Rightarrow$ not-in-memory)
- Initially valid–invalid bit is set to 0 on all entries

Valid-Invalid Bit

- During address translation, if valid–invalid bit in page table entry is 0 $\Rightarrow$ **page fault**.

Frame #	valid-invalid bit
	1
	1
	1
	1
	0
⋮	
	0
	0

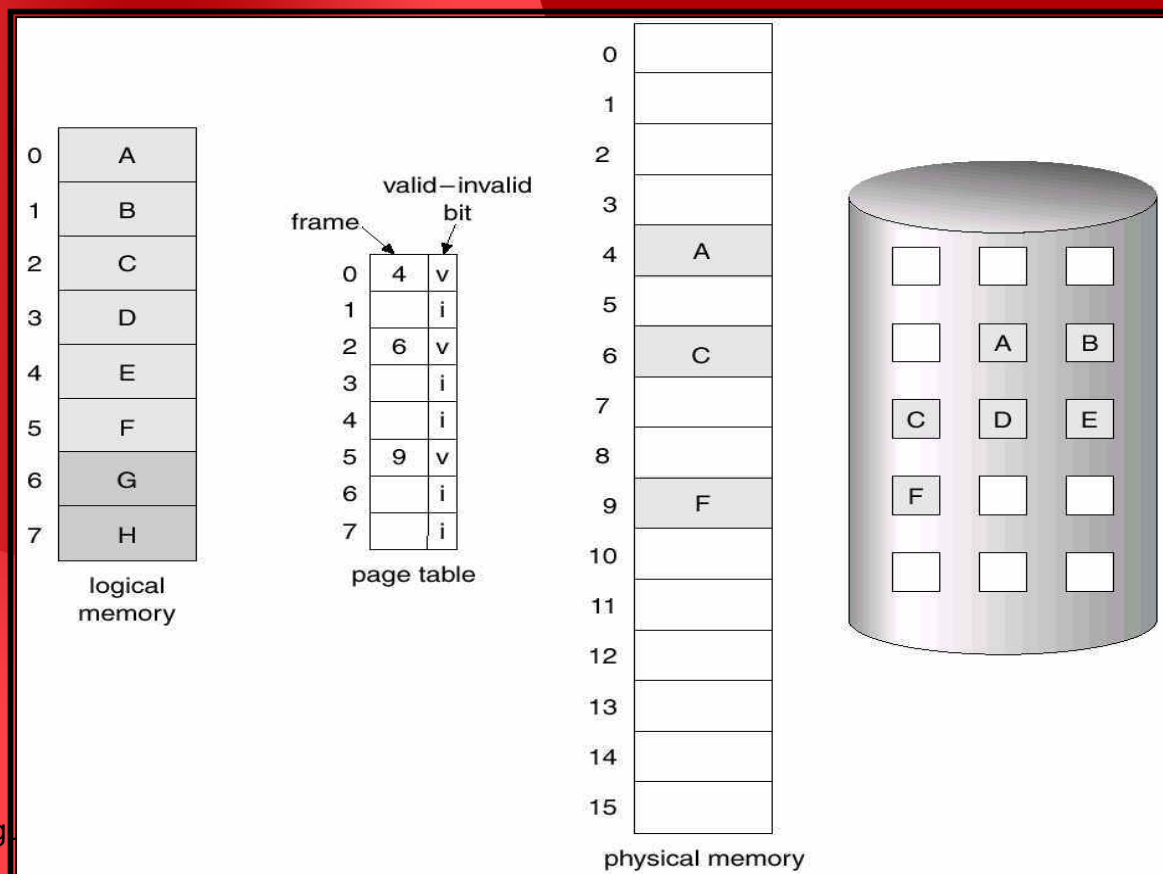
7 August 2021

© Copyright Virtual University of Pakistan

page table



Demand Paging



7 Aug



Page Fault

- If there is ever a reference to a page, first reference will trap to OS $\Rightarrow$ page fault
- OS decides
 - Invalid reference $\Rightarrow$ trap to OS $\Rightarrow$ abort process
 - Just not in memory $\Rightarrow$ page fault $\Rightarrow$ service page fault

7 August 2021

© Copyright Virtual University of
Pakistan



Page Fault

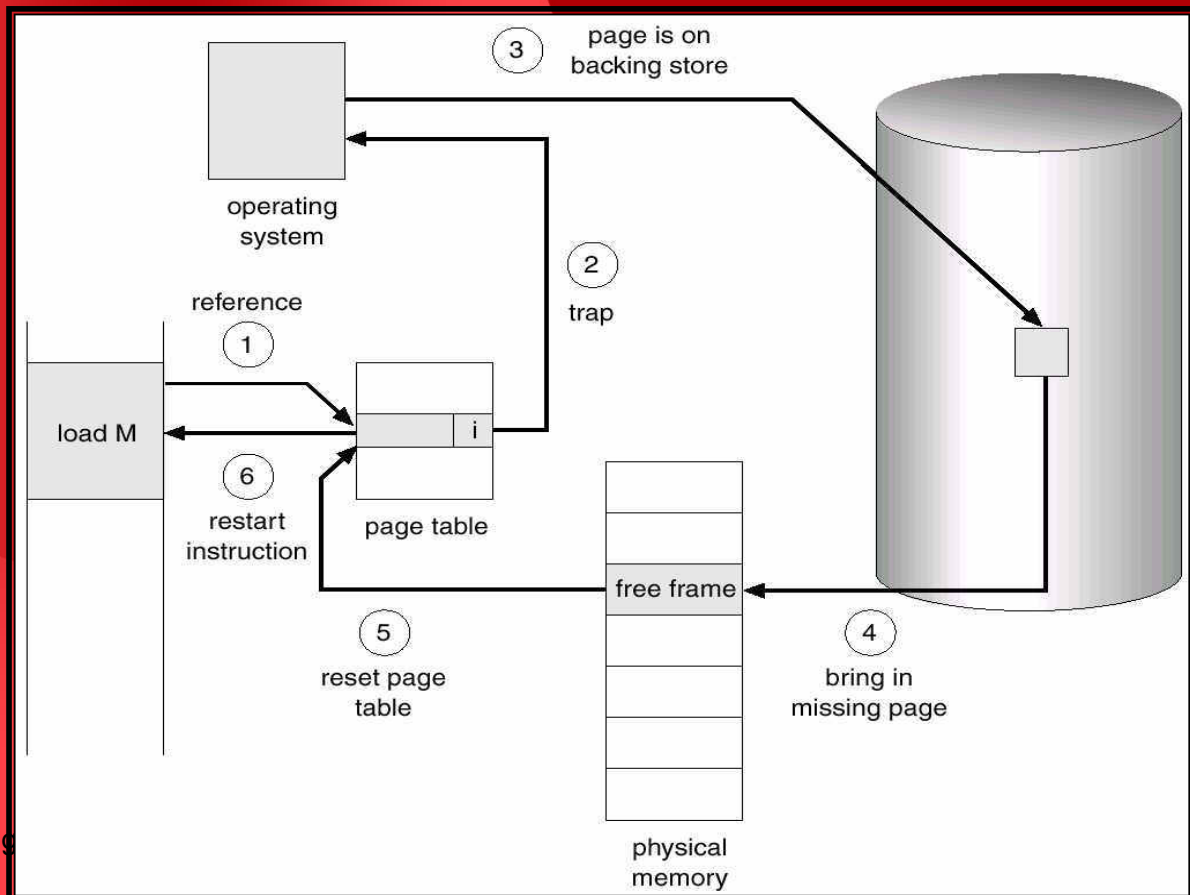
- Allocate an empty frame
- Locate the desired page on disk
- Swap in the desired page into the newly allocated frame.
- Store the frame number in the appropriate page table entry
- Reset tables; set valid/invalid bit to 1
- Restart instruction

7 August 2021

© Copyright Virtual University of
Pakistan



Servicing a Page Fault



Restarting Instruction

- Problems with restarting instructions that cause page faults
 - Auto increment/decrement location
 - Block move

Block Move

Destination
String



0

1

Source
String

2

3

⋮

7 August 2021

© C



Block Move

Destination
String



0

1

Source
String

2

3

⋮

7 August 2021

© C



Performance of Demand Paging

- Page Fault Rate $0 \leq p \leq 1.0$
 - if $p = 0$ no page faults
 - if $p = 1$, every reference is a fault

7 August 2021

© Copyright Virtual University of
Pakistan



Performance of Demand Paging

- **Effective Access Time (EAT)**
$$\text{EAT} = (1 - p) \times \text{memory access} + p (\text{page fault service time})$$

7 August 2021

© Copyright Virtual University of
Pakistan



Page Fault Service

- Trap to OS
- Context switch
- Locate the vector for the given trap
- Check that the page reference was legal and determine the location of the desired page on the disk
- Locate a free frame
- Issue a disk read into this frame

7 August 2021

© Copyright Virtual University of
Pakistan



Page Fault Service

- While waiting for disk read to complete, schedule another process
- Interrupt from the disk controller indicating completion of disk read
- Correct the page table (frame number, in-memory bit, etc.)
- Put process in the ready queue
- Restart process with the instruction that caused page fault

7 August 2021

© Copyright Virtual University of
Pakistan



Example

- Memory access time = 100 nanosec
- Page fault service time = 25 millisec
- $T_{\text{effective}} = (1 - p) \times 100 + p (25 \text{ milli})$
 $= (1 - p) \times 100 + p (25000000)$
 $= 100 + 24999900 \times p$
- If one access out of 1000 causes a page fault, effective access time is 25 microseconds, a slowdown by a factor of 250.

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Background
- Demand Paging
- Page fault
- Performance of Demand Paging

Operating Systems

Lecture 37

Syed Mansoor Sarwar

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 38

Syed Mansoor Sarwar



Agenda for Today

- Review of previous lecture
- Performance of Demand Paging
- Process Creation
- Memory Mapped Files

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 37

- Memory Management in Intel 80386
- Virtual Memory Concept
- Hardware Support
- Demand Paging
- Page Fault
- Performance of Demand Paging

7 August 2021

© Copyright Virtual University of
Pakistan



Intel 80386 Example

16-bit
Selector

32-bit Offset



13-bit Segment #

s

g

p

1-bit field to
specify GDT or
LDT

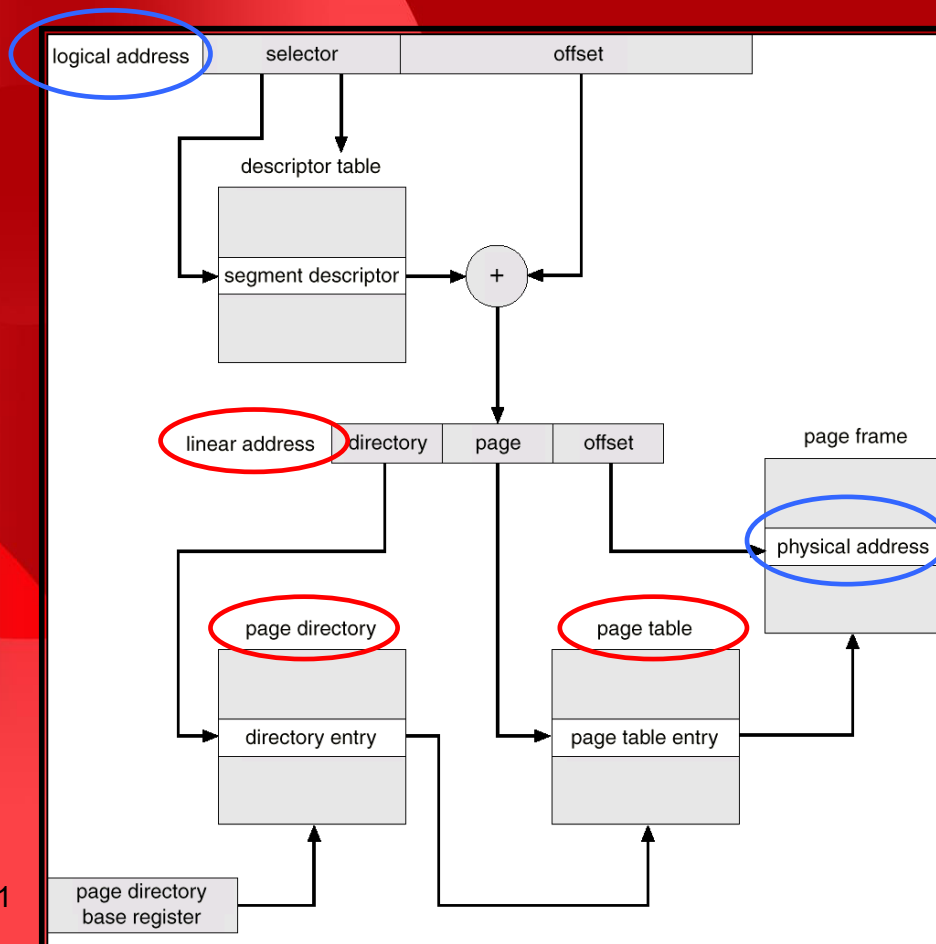
2-bit field for
specifying the
privilege level

7 August 2021

© Copyright Virtual University of
Pakistan



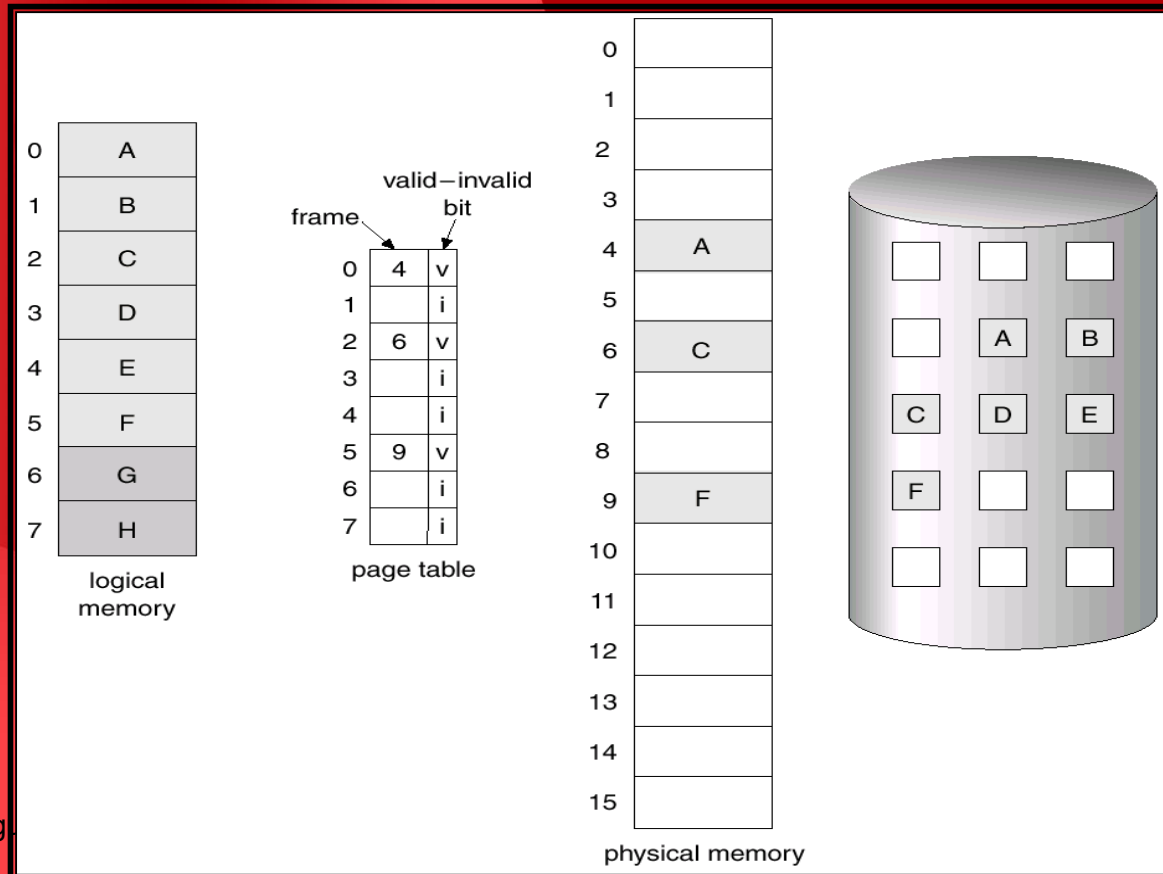
Intel 80386 Example



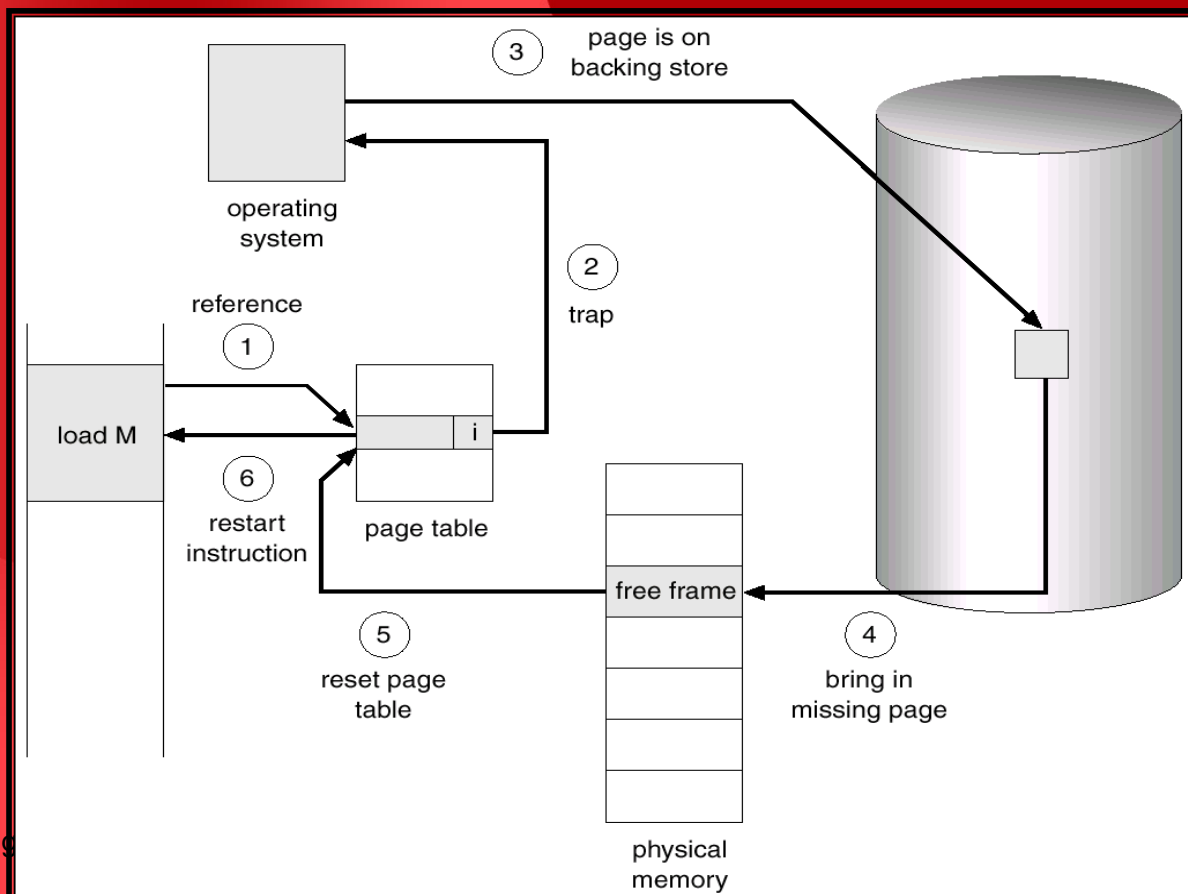
7 August 2021



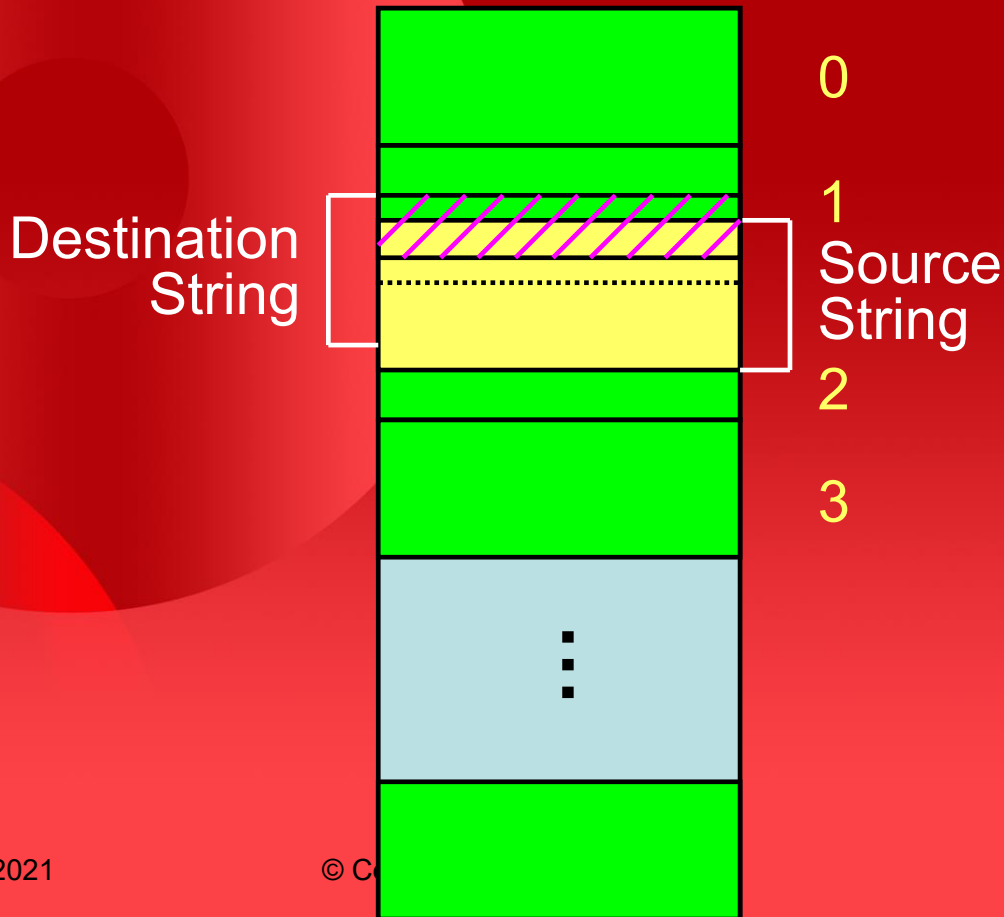
Demand Paging



Servicing a Page Fault



Block/String Move



7 August 2021

© C



Performance of Demand Paging

- **Effective Access Time (EAT)**

$$\text{EAT} = (1 - p) \times \text{memory access time} + p (\text{page fault service time})$$

7 August 2021

© Copyright Virtual University of Pakistan



Example

- Memory access time = 100 nanosec
- Page fault service time = 25 millisec
- $T_{\text{effective}} = (1 - p) \times 100 + p (25 \text{ milli})$
 $= (1 - p) \times 100 + p (25000000)$
 $= 100 + 24999900 \times p$
- If one access out of 1000 causes a page fault, effective access time is 25 microseconds, a slowdown by a factor of 250.

7 August 2021

© Copyright Virtual University of
Pakistan



Example

- If we want less than 10 percentage degradation in effective memory access time then we have the following inequality
 $110 > 100 + 25000000 \times p$
 $10 > 25000000 \times p$
 $p < 0.0000004$
- This means we can allow only one page fault every 2,500,000.

7 August 2021

© Copyright Virtual University of
Pakistan



Page Fault and No Free Frame

■ Page Replacement

- Replacement algorithm (minimize number of page faults)
- Performance with replacement
- Victim selection
 - Criteria
- Local vs. global

7 August 2021

© Copyright Virtual University of Pakistan



Another Example

- Effective memory access is 100 ns
- Page fault overhead is 100 microseconds = 10^5 ns
- Page swap time is 10 milliseconds = 10^7 ns
- 50% of the time the page to be replaced is “dirty”
- Restart overhead is 20 microseconds = 2×10^4 ns

7 August 2021

© Copyright Virtual University of Pakistan



Another Example

Effective access time =

$$\begin{aligned} & 100 \times (1-p) + (10^5 + 2 \times 10^4 \\ & \quad + 0.5 \times 10^7 + 0.5 \times 2 \times 10^7) \times p \\ & = 100 \times (1-p) + 15,120,000 \times p \end{aligned}$$

7 August 2021

© Copyright Virtual University of
Pakistan



What is a Good Page Fault Rate?

- For the previous example suppose p is 1%, then EAT is
$$\begin{aligned} & = 100 \times (1-p) + 15,120,000 \times p \\ & = 151299 \text{ ns} \end{aligned}$$
$$\Rightarrow \text{a slowdown of } 151299 / 100 = 1513$$

7 August 2021

© Copyright Virtual University of
Pakistan



What is a Good Page Fault Rate?

- For the luxury of virtual memory to cost only 20% overhead, we need
$$120 > 100 \times (1-p) + 15,120,000 \times p$$
$$120 > 100 - 100p + 15,120,000p$$
$$p < 0.00000132$$

⇒ Less than one page fault for every 755995 memory accesses!

7 August 2021

© Copyright Virtual University of
Pakistan



Process Creation

- `fork()`
 - Exact copy of parent's memory image
 - Copying parent's pages (address space)
- Child may call `exec()` immediately ⇒ copying parent's address space is not needed

7 August 2021

© Copyright Virtual University of
Pakistan



Process Creation

- “Copy-on-write” $\Rightarrow$ child share's parent's address space and pages which may change are marked “copy-on-write”
- When a process tries to modify a page, it is copied and inserted in the page table for the process
- Used in Windows 2000, Solaris 2, and Linux

7 August 2021

© Copyright Virtual University of
Pakistan



Process Creation

- `vfork()`
- Parent is suspended and child uses parent's address space
- Pages altered by the child process are visible to parent
- Intended to be used when child process calls `exec()` immediately after its creation

7 August 2021

© Copyright Virtual University of
Pakistan



Linux Implementation

- Shared pages are marked readonly after `fork()`
- If either process tries to modify a shared page, a page fault occurs and the page is copied.
- The other process (who later faults on write) discovers it is the only owner; so no copying takes place.

7 August 2021

© Copyright Virtual University of
Pakistan



Memory-Mapped Files

- Memory-mapped file I/O allows file I/O to be treated as routine memory access by mapping a disk block to a page in memory.
- A file is initially read using demand paging. A page-sized portion of the file is read from the file system into a physical page. Subsequent reads/writes from/to the file are treated as ordinary memory accesses.

7 August 2021

© Copyright Virtual University of
Pakistan



Memory-Mapped Files

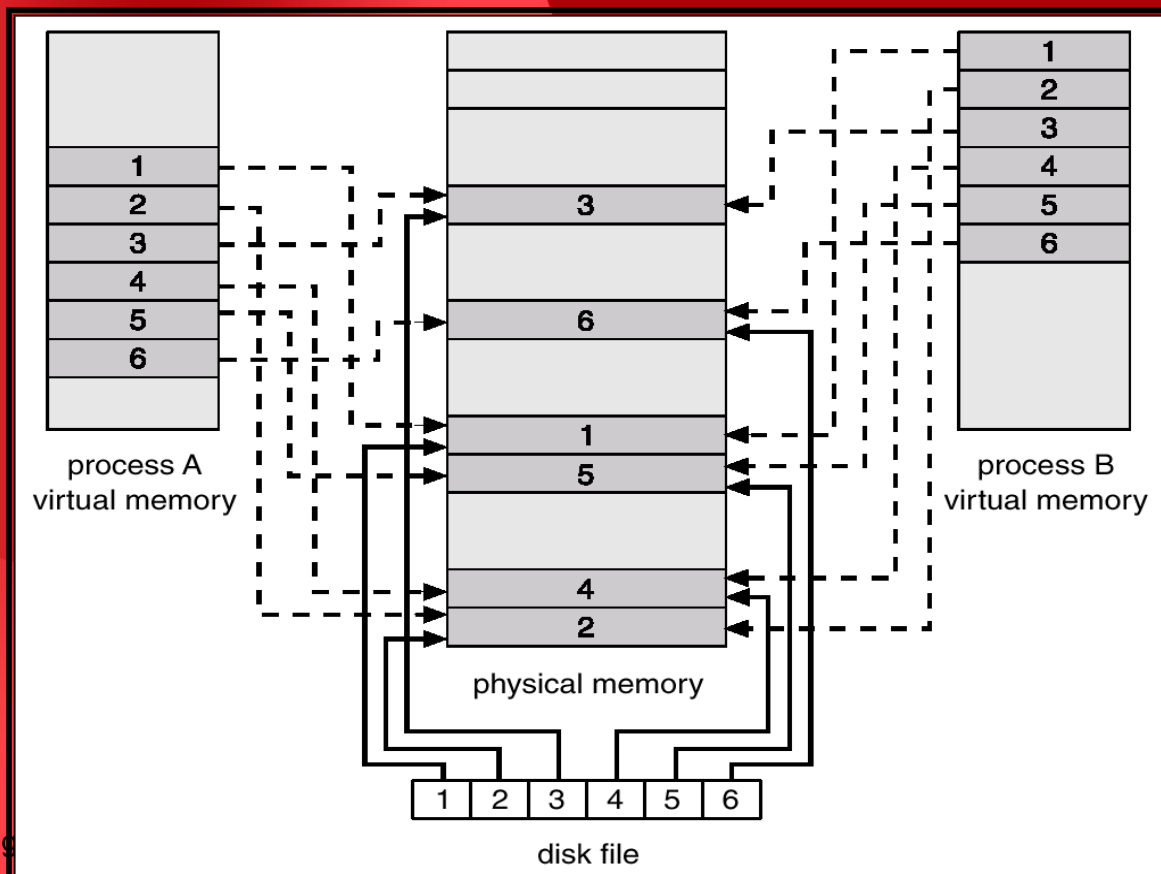
- Simplifies file access by treating file I/O through memory rather than `read()` `write()` system calls.
- Also allows several processes to map the same file allowing the pages in memory to be shared.
- Some operating systems only provide memory-mapping through a system call (`mmap()`). Others treat all file I/O as memory mapped files.

7 August 2021

© Copyright Virtual University of
Pakistan



Memory-Mapped Files



7 Aug



mmap () System Call

- “Normal” File I/O

```
files = open(...);  
lseek(...);  
read(files, buf, len);  
/* use data in buf */
```

- File I/O with mmap ()

```
files = open(...)  
address = mmap((caddr_t) 0, len, (PROT_READ |  
PROT_WRITE), MAP_PRIVATE, files, offset);  
/* use data at address */
```

7 August 2021

© Copyright Virtual University of
Pakistan



Memory-Mapped Files in Solaris 2

- If file is opened as memory mapped: it maps the file to the address space of the process.
- If the file is opened non memory-mapped, Solaris 2 opens it as a memory-mapped file, mapping it to the kernel address space

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Performance of Demand Paging
- Process Creation
- Memory Mapped Files

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 38

Syed Mansoor Sarwar

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 39

Syed Mansoor Sarwar



Agenda for Today

- Review of the previous lecture
- Page Replacement
- Page Replacement Algorithms
 - First-In-First-Out (FIFO)
 - Optimal Replacement
 - Least Recently Used (LRU)
- Belady's Anomaly
- Stack Replacement Algorithms



Review of Lecture 38

- Performance of Demand Paging
- Process Creation
- Memory Mapped Files

7 August 2021

© Copyright Virtual University of
Pakistan



Performance of Demand Paging

- **Effective Access Time (EAT)**
$$\text{EAT} = (1 - p) \times \text{memory access time} + p (\text{page fault service time})$$

7 August 2021

© Copyright Virtual University of
Pakistan



Example

- Memory access time = 100 nanosec
- Page fault service time = 25 millisec
- $T_{\text{effective}} = (1 - p) \times 100 + p (25 \text{ milli})$
 $= (1 - p) \times 100 + p (25000000)$
 $= 100 + 24999900 \times p$
- If one access out of 1000 causes a page fault, effective access time is 25 microseconds, a slowdown by a factor of 250.

7 August 2021

© Copyright Virtual University of
Pakistan



Example

- If we want less than 10 percentage degradation in effective memory access time then we have the following inequality
 $110 > 100 + 25000000 \times p$
 $10 > 25000000 \times p$
 $p < 0.0000004$
- This means we can allow only one page fault every 2,500,000.

7 August 2021

© Copyright Virtual University of
Pakistan



Process Creation

- “Copy-on-write”—child shares parent’s address space and is given its own copy of “copy-on-write” pages when it tries to modify them
- `vfork()` —child shares parent’s address space; useful when child invokes `exec()` immediately after its creation

7 August 2021

© Copyright Virtual University of
Pakistan



Memory-Mapped Files

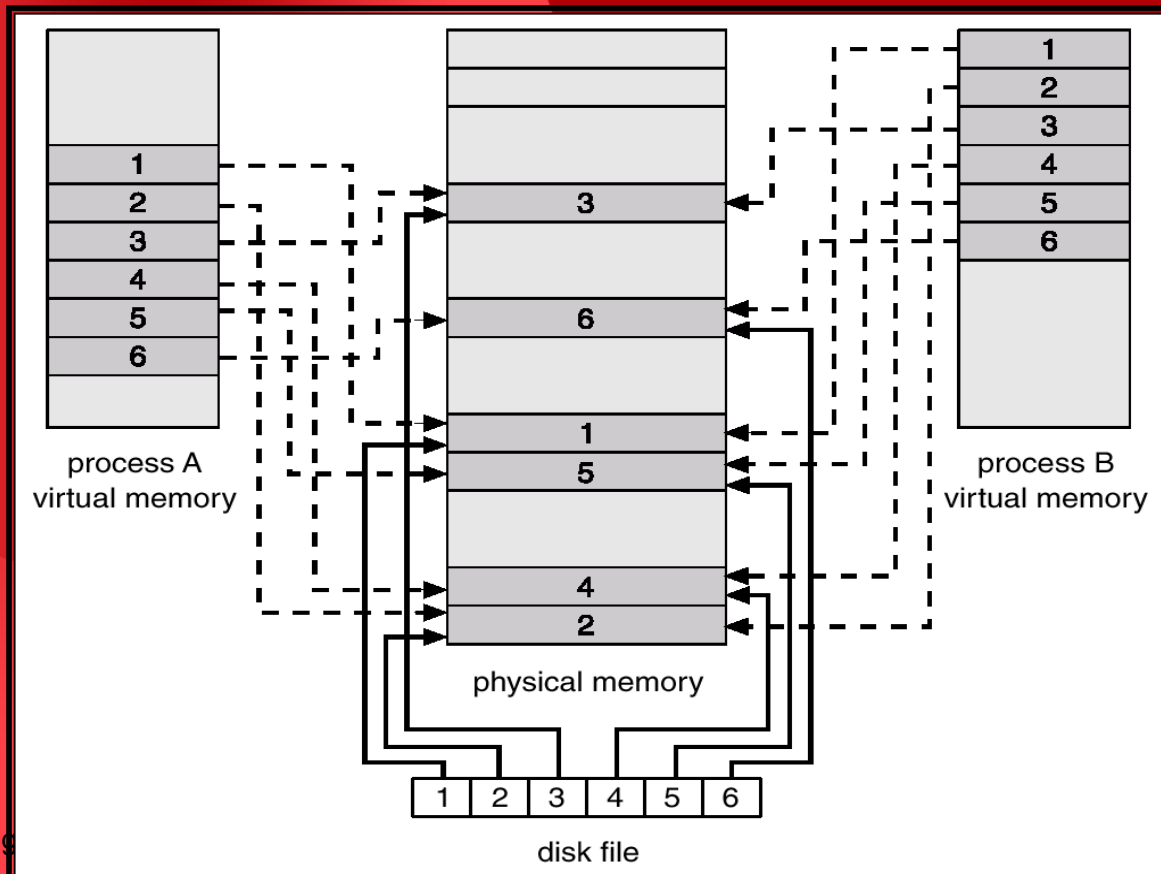
- Memory-mapped file I/O allows file I/O to be treated as routine memory access by mapping a disk block to a page in memory.
- Automatically or by using the `mmap()` system call on Solaris

7 August 2021

© Copyright Virtual University of
Pakistan



Memory-Mapped Files



Page Replacement

- If no free frame is available on a page fault, replace a page in memory to load the desired page
- Page-fault service routine is modified to include page replacement.

Page Replacement

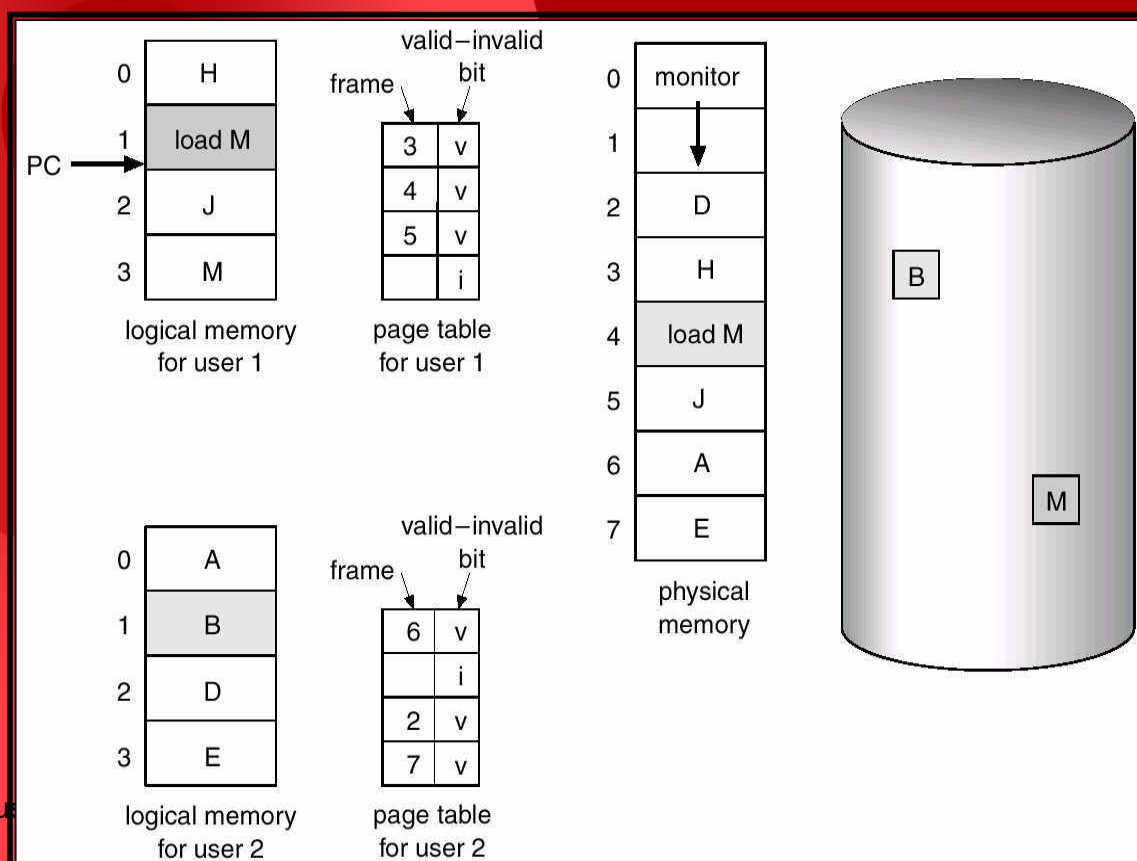
- Use modify (dirty) bit to reduce overhead of page transfers – only modified pages are written to disk.
- Set by hardware when data is written to a page
- Checked by OS at page replacements

7 August 2021

© Copyright Virtual University of
Pakistan



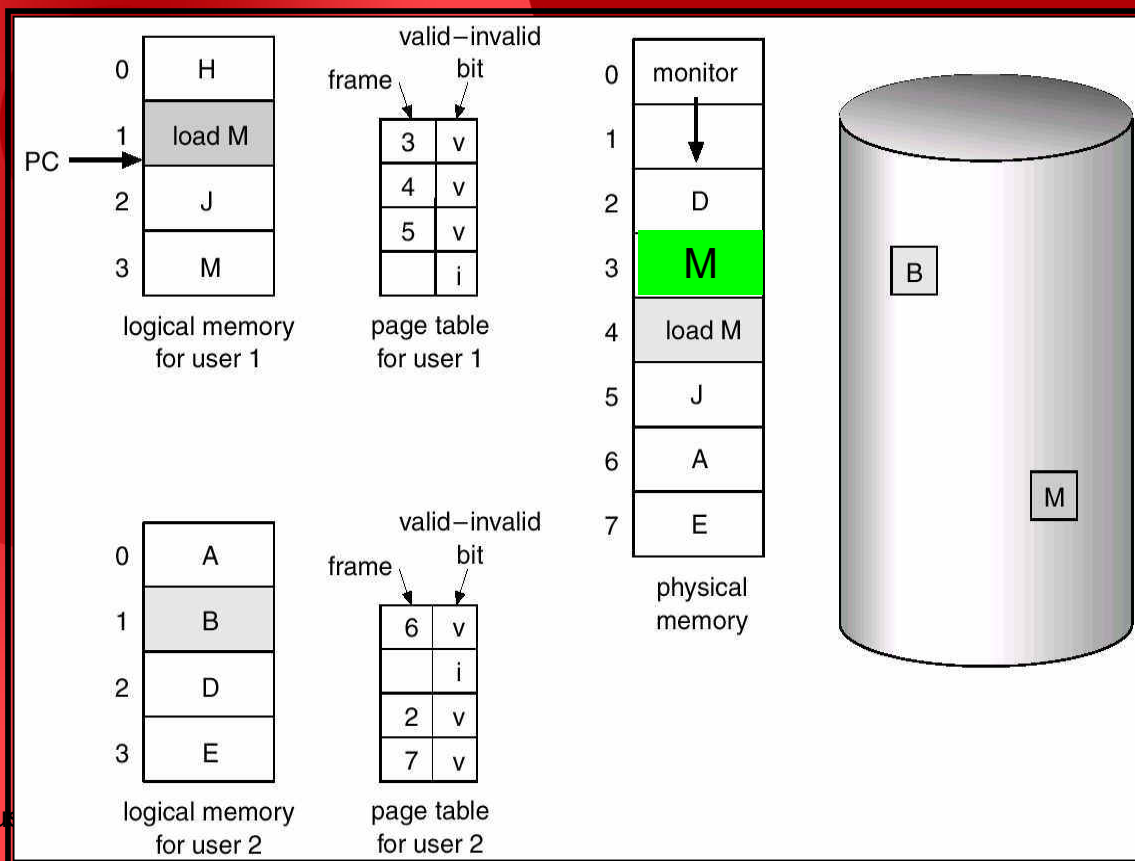
Page Replacement



7 August

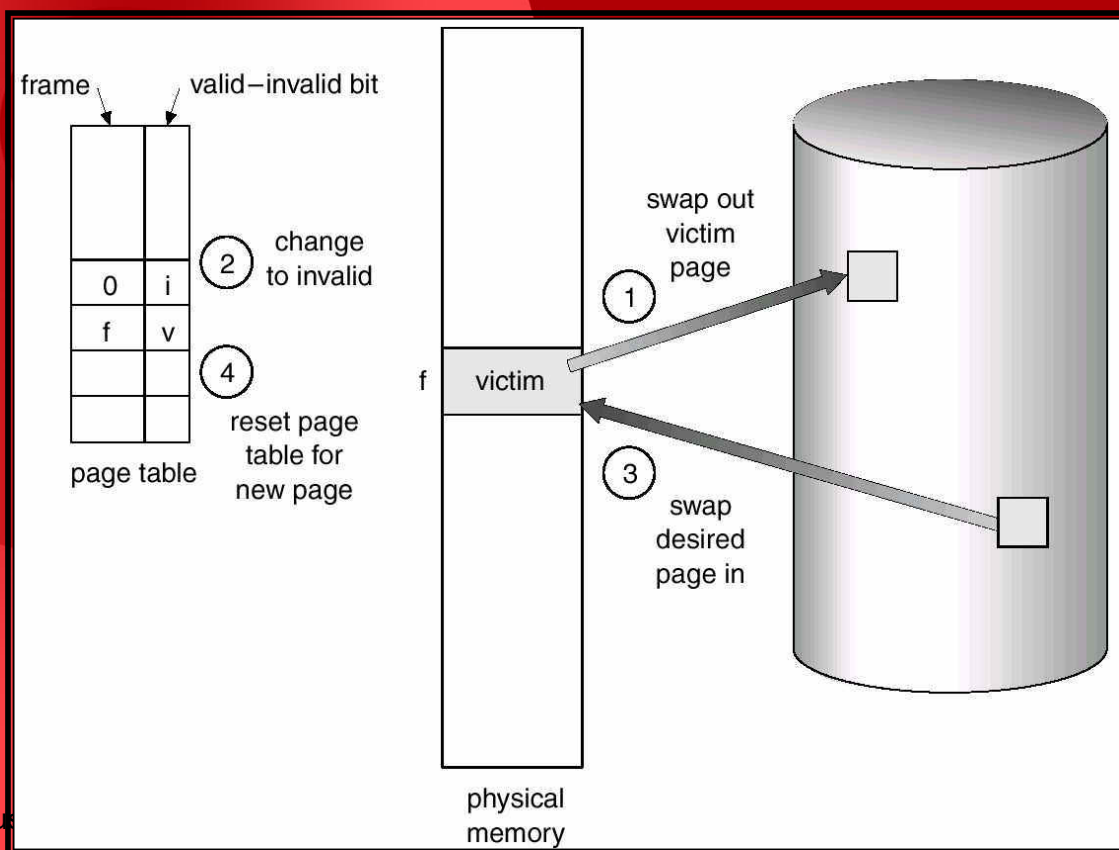


Page Replacement



7 August

Page Replacement



7 August

Page Replacement Algorithms

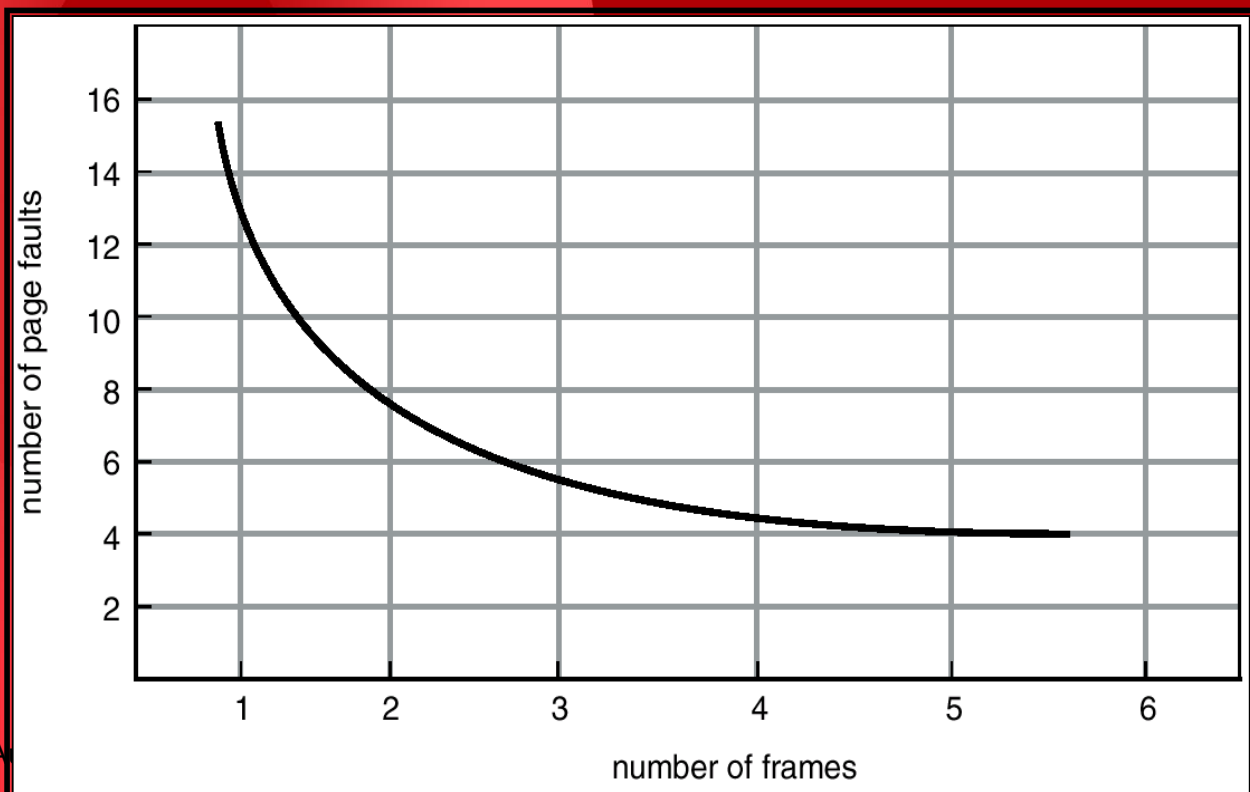
- Want lowest page-fault rate.
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.

7 August 2021

© Copyright Virtual University of
Pakistan



Page Faults vs. Number of Frames



FIFO Algorithm

- Number of frame allocated = 4

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 5, 4, 5, 4, 4

1	5	5	5	5	8	8	8	8
2	2	1	1	1	1	9	9	9
3	3	3	6	6	6	6	5	5
4	4	4	4	7	7	7	7	4

7 Aug

Optimal Algorithm

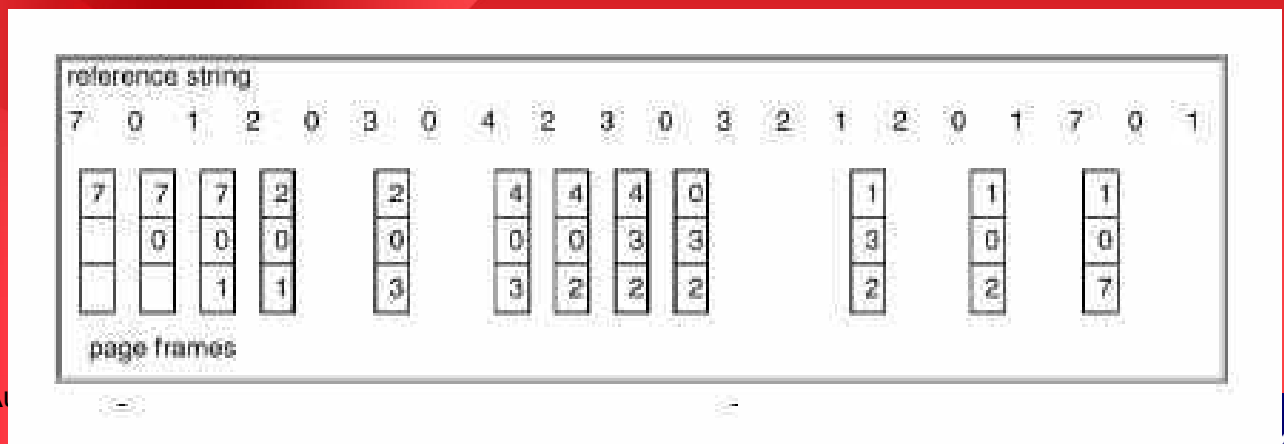
- Replace the page that will not be used for the longest period of time.

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 5, 4, 5, 4, 4

1	1	6	6	8	9	9
2	5	5	5	5	5	5
3	3	3	7	7	7	7
4	4	4	4	4	4	4

Least Recently Used (LRU) Algorithm

- Replace the page that hasn't been used for the longest period of time.



LRU Example

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 5, 4, 5, 4, 4

1	5	5	6	6	6	6	5	5
2	2	1	1	1	1	9	9	9
3	3	3	3	7	7	7	7	4
4	4	4	4	4	8	8	8	8

LRU Implementation

■ Counter-Based Implementation

- Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
- When a page needs to be replaced, choose the one with minimum counter value.

7 August 2021

© Copyright Virtual University of
Pakistan



LRU Implementation

■ Stack-based Implementation

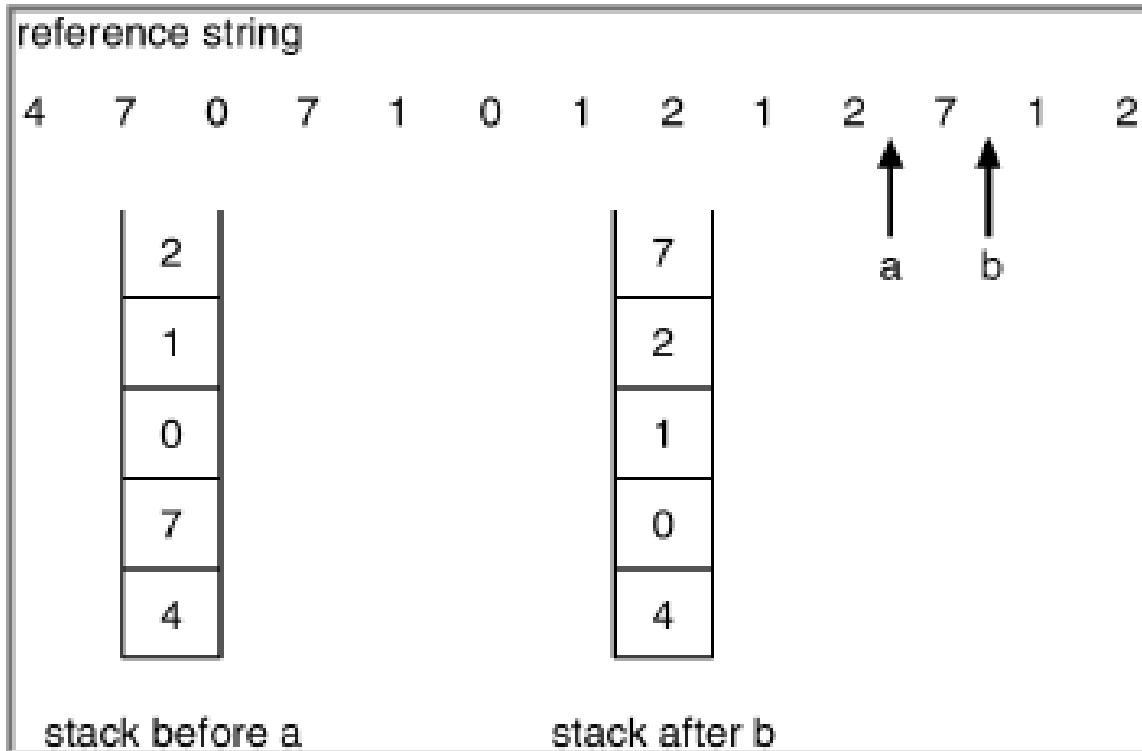
- Keep a stack of page numbers. Whenever a page is referenced, remove it from its current location and push it onto the stack. The LRU page is at the bottom of the stack.
- Doubly-linked list implementation of stack requires 6 pointer changes.

7 August 2021

© Copyright Virtual University of
Pakistan



LRU Implementation



Recap of Lecture

- Page Replacement
- Page Replacement Algorithms
 - First-In-First-Out (FIFO)
 - Optimal Replacement
 - Least Recently Used (LRU)

Operating Systems

Lecture 39

Syed Mansoor Sarwar

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 40

Syed Mansoor Sarwar



Agenda for Today

- Review of the previous lecture
- Belady's Anomaly
- Page Replacement Algorithms
 - Least Frequently Used (LFU)
 - Most Frequently Used (MFU)
 - Page Buffering Algorithm
- Allocation of Frames
- Minimum Number of Frames
- Thrashing

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 39

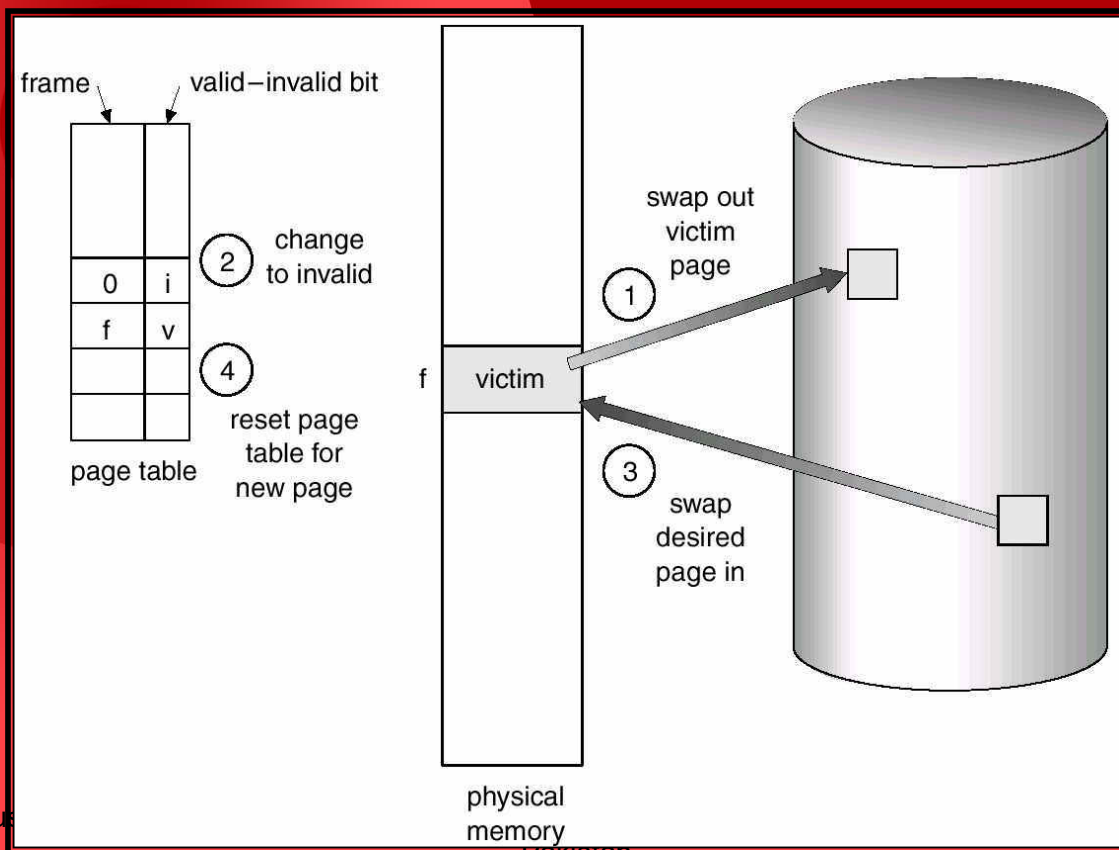
- Page Replacement
- Page Replacement Algorithms
 - First-In-First-Out (FIFO)
 - Optimal Replacement
 - Least Recently Used (LRU)

7 August 2021

© Copyright Virtual University of
Pakistan



Page Replacement



Page Replacement Algorithms

- First-In-First-Out (FIFO)
- Optimal
- Least Recently Used (LRU)
 - Counter-based implementation
 - Stack-based implementation

FIFO Algorithm

- Number of frames allocated = 3
- Reference string:
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Number of page faults = 9

1	1	1	4	4	4	5	5	5
	2	2	2	1	1	1	3	3
		3	3	3	2	2	2	4

7 August 20

Pakistan

FIFO Algorithm

- Number of frames allocated = 4
- Reference string:
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Number of page faults = 10

1	1	1	1	5	5	5	5	4	4
	2	2	2	2	1	1	1	1	5
		3	3	3	3	2	2	2	2
			4	4	4	4	3	3	3

7 August

FIFO Algorithm

■ Belady's Anomaly

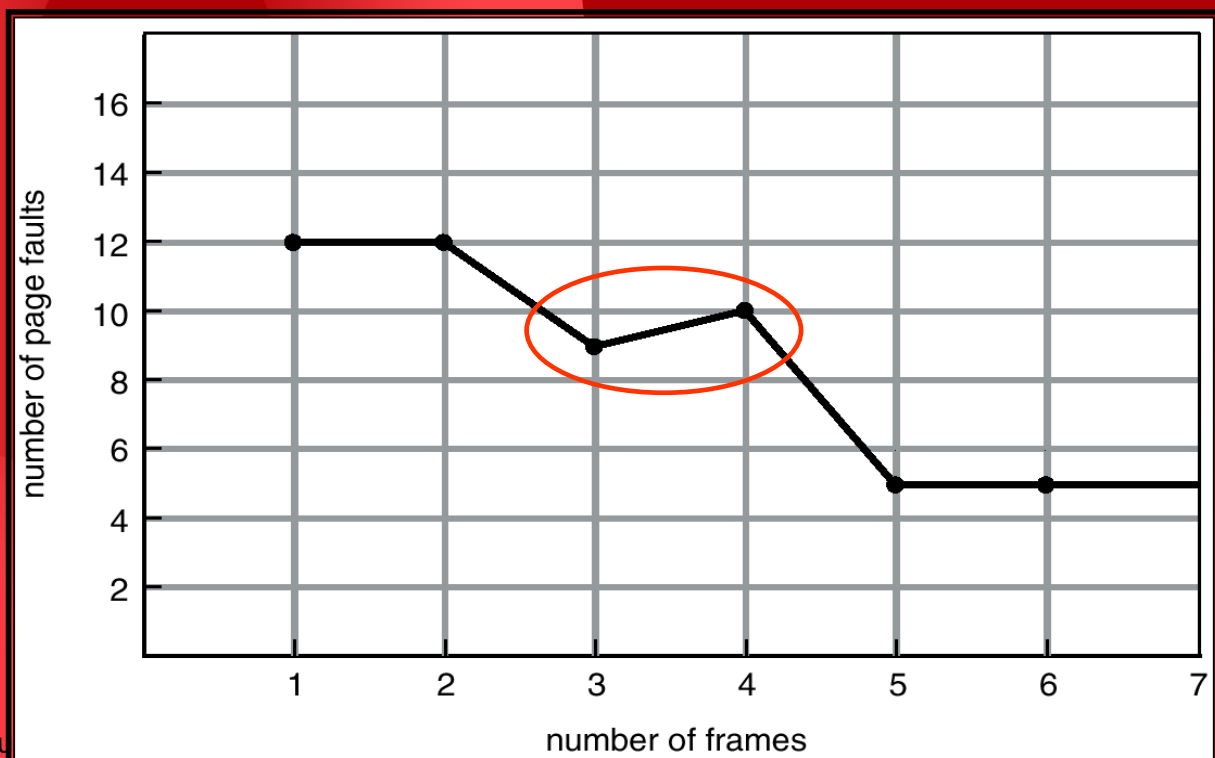
“For some page replacement algorithms, the page fault rate may increase as the number of allocated frames increases.”

7 August 2021

© Copyright Virtual University of
Pakistan



Belady's Anomaly



7 Au

Pakistan



Stack Replacement Algorithms

- A class of page replacement algorithms with the following property:

Set of pages in the main memory with n frames is a subset of the set of pages in memory with $n+1$ frames

- Do not suffer from Belady's Anomaly
- Example: LRU

7 August 2021

© Copyright Virtual University of
Pakistan



LRU Algorithm

- Number of frames allocated = 3
- Reference string:
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Number of page faults = 10

1	1	1	4	4	4	5	3	3	3
	2	2	2	1	1	1	1	4	4
		3	3	3	2	2	2	2	5

7 August

Pakistan



LRU Algorithm

- Number of frames allocated = 4
- Reference string:
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Number of page faults = 8

1	1	1	1	1	1	1	5
	2	2	2	2	2	2	2
		3	3	5	5	4	4
			4	4	3	3	3

7 August 2021



LRU Approximation Algorithm

- **Use of reference bit**
 - With each page associate a bit, initially = 0
 - When page is referenced bit set to 1.
 - Replace the one which is 0 (if one exists). We do not know the order, however.

7 August 2021

© Copyright Virtual University of
Pakistan



Least Frequently Used

- **Least Frequently Used (LFU)**
 - Replace the page with the smallest reference count.
 - Based on the **locality of reference** concept—least frequently used page is not in the **current locality**

7 August 2021

© Copyright Virtual University of
Pakistan



Most Frequently Used

- **Most Frequently Used (MFU)**
 - The page with the smallest count was probably just brought in and has yet to be used; it will be in the **locality** that has just started.

7 August 2021

© Copyright Virtual University of
Pakistan



Page Buffering Algorithm

- OS may keep a pool of free frames
 - A process in need can be given a frame quickly
 - Victims are selected and free frames are added to the pool in the background

7 August 2021

© Copyright Virtual University of
Pakistan



Page Buffering Algorithm

- OS may keep a list of modified pages
 - In the background, the modified pages are written to disk
 - Less dirty pages in the system at page replacements
 - Used together with FIFO replacement in the VAX/VMS operating system

7 August 2021

© Copyright Virtual University of
Pakistan



Local vs Global Replacement

- If process P generates a page fault
 - Select for replacement one of its frames.
 - Select for replacement a frame from a process with lower priority number.

7 August 2021

© Copyright Virtual University of
Pakistan



Allocation of Frames

- Each process needs a **minimum number of frames** so that its execution may be guaranteed on a given machine .
- **Example:** MOV X,Y
 - Instruction is 6 bytes long (16-bit offsets) and might span 2 pages.
 - 2 pages to handle **from**.
 - 2 pages to handle **to**.

7 August 2021

© Copyright Virtual University of
Pakistan



Allocation of Frames

Instruction

X

Y



7 August 2021

Allocation of Frames

- Three major allocation schemes:
 - **Fixed allocation**
Free frames are equally divided among processes
 - **Proportional Allocation**
Number of frames allocated to a process is proportional to its size
 - **Priority allocation**
Priority-based proportional allocation

7 August 2021

Frame Allocation

- Number of free frames = 64
- Number of processes = 3
- Process sizes

P1 = 10 pages

P2 = 40 pages

P3 = 127 pages

- Fixed allocation

64/3 = 21 frames per process and one put in the free frames list

7 August 2021

© Copyright Virtual University of
Pakistan



Frame Allocation

- Proportional Allocation

- s_i = Size of process P_i

- $S = \sum s_i$

- m = Number of free frames

- a_i = Allocation for $P_i = (s_i / S) * m$

- $a_1 = (10 / 177) * 64 = 3$ frames

- $a_2 = (40 / 177) * 64 = 14$ frames

- $a_3 = (127 / 177) * 64 = 45$ frames

- Two free frames are put in the list of free frames

7 August 2021

© Copyright Virtual University of
Pakistan



Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
 - Low CPU utilization
 - Operating system thinks that it needs to increase the degree of multiprogramming.
 - Another process added to the system—serious problem!

7 August 2021

© Copyright Virtual University of
Pakistan



Thrashing

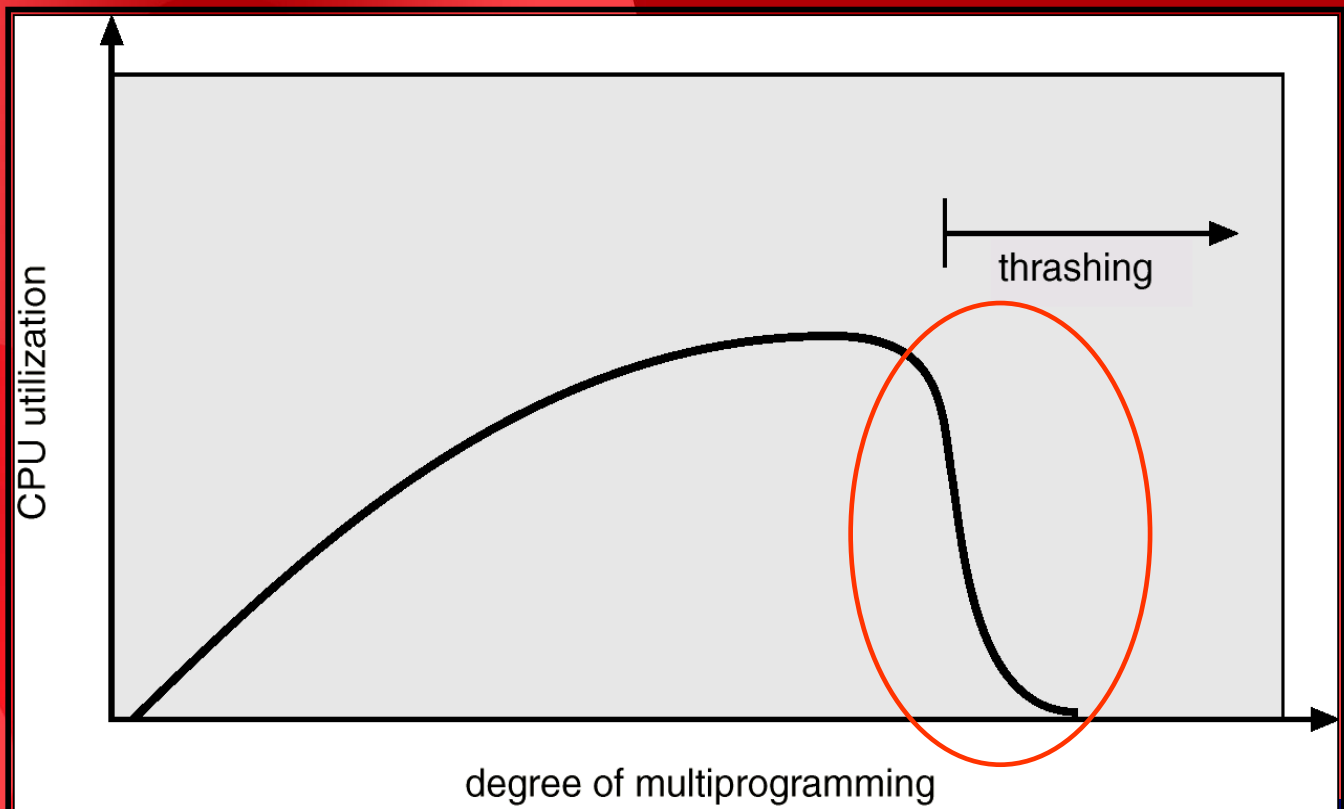
- A process is **thrashing** if it is spending more time paging (i.e., swapping pages in and out) than executing
- Thrashing results in severe performance problems
 - Low CPU utilization
 - High disk utilization
 - Low utilization of other I/O devices

7 August 2021

© Copyright Virtual University of
Pakistan



Thrashing



Pakistan

VUI
Virtual University

Thrashing

- To stop thrashing, the degree of multiprogramming needs to be reduced
- Local page replacement prevents thrashing to spread among several processes
- However, a thrashing process consumes more resources

Recap of Lecture

- Belady's Anomaly
- Page Replacement Algorithms
 - Least Frequently Used (LFU)
 - Most Frequently Used (MFU)
 - Page Buffering Algorithm
- Allocation of Frames
- Minimum Number of Frames
- Thrashing

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 40

Syed Mansoor Sarwar

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 41

Syed Mansoor Sarwar



Agenda for Today

- Review of the previous lecture
- Thrashing
- The Working Set Model
- Page Fault Frequency Model
- Other Considerations
 - Prepaging
 - Page size
 - Program structure
- Examples



Review of Lecture 40

- Belady's Anomaly
- Page Replacement Algorithms
 - Least Frequently Used (LFU)
 - Most Frequently Used (MFU)
 - Page Buffering Algorithm
- Allocation of Frames
- Minimum Number of Frames
- Thrashing

7 August 2021

© Copyright Virtual University of
Pakistan



Allocation of Frames

- Each process needs a **minimum number of frames** so that its execution may be guaranteed on a given machine .
- **Example:** MOV X,Y
 - Instruction is 6 bytes long (16-bit offsets) and might span 2 pages.
 - 2 pages to handle **from**.
 - 2 pages to handle **to**.

7 August 2021

© Copyright Virtual University of
Pakistan



Allocation of Frames

Instruction

X

Y



7 August 2021

Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
 - Low CPU utilization
 - Operating system thinks that it needs to increase the degree of multiprogramming.
 - Another process added to the system—serious problem!

7 August 2021

Thrashing

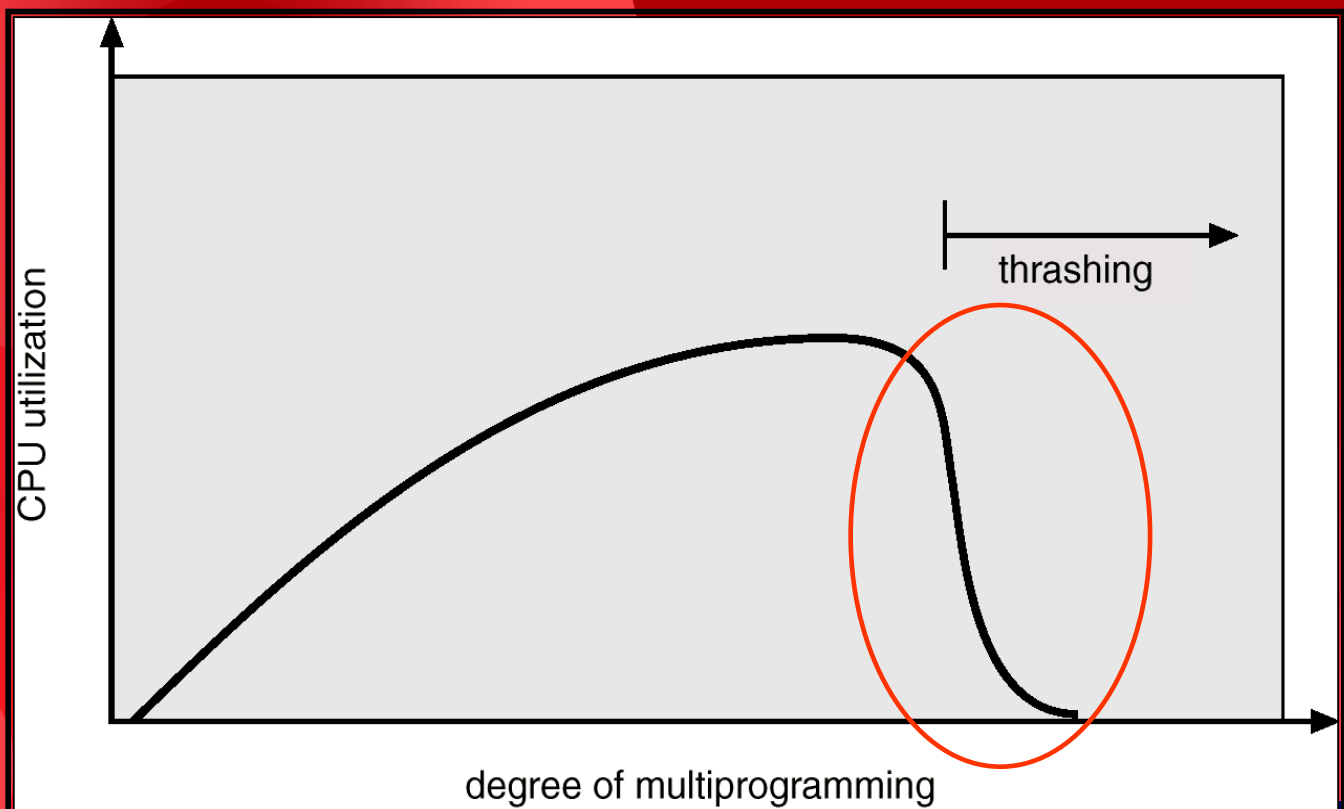
- A process is **thrashing** if it is spending more time paging (i.e., swapping pages in and out) than executing
- Thrashing results in severe performance problems
 - Low CPU utilization
 - High disk utilization
 - Low utilization of other I/O devices

7 August 2021

© Copyright Virtual University of
Pakistan



Thrashing



Pakistan



Thrashing

- To stop thrashing, the degree of multiprogramming needs to be reduced
- Local page replacement prevents thrashing to spread among several processes
- However, a thrashing process consumes more resources

7 August 2021

© Copyright Virtual University of
Pakistan



Locality of Reference

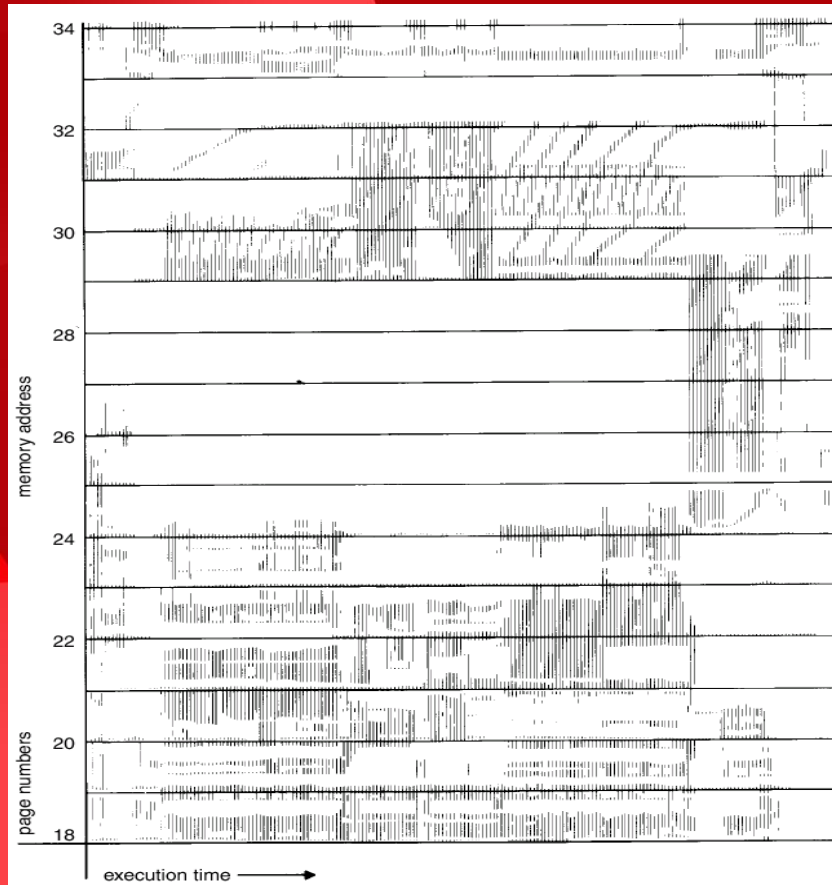
- Why does paging work?
Locality of Reference Model
 - The set of pages a process accesses actively, is known as its locality
 - Process migrates from one locality to another
 - Localities may overlap

7 August 2021

© Copyright Virtual University of
Pakistan



Process Localities



7 August 2021



Working Set Model

- **Working Set Window (Δ)**
A fixed number of page references, e.g., 10,000 references
- **Working Set Size of P_i (WSS_i)**
Total number of pages referenced in the most recent Δ

7 August 2021

© Copyright Virtual University of
Pakistan

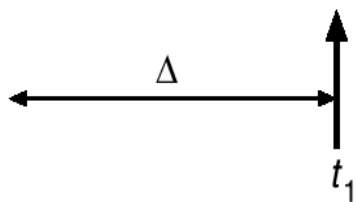


Working Set Model

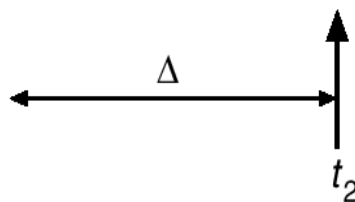
Let $\Delta = 10$ references

page reference table

... 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 ...



$WS(t_1) = \{1, 2, 5, 6, 7\}$



$WS(t_2) = \{3, 4\}$

7 August 2021

© Copyright Virtual University of
Pakistan



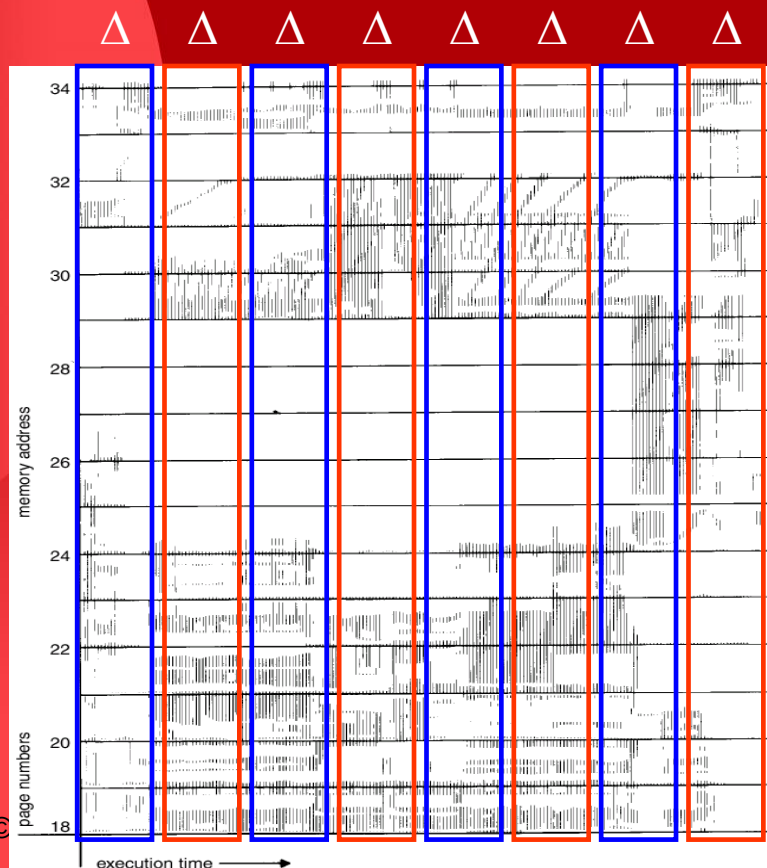
Process Localities

The first two and last localities are:

$L1 = \{18-26, 31-34\}$

$L2 = \{18-23, 29-31, 34\}$

Last = $\{18-20, 24-34\}$



7 August 2021

©



Working Set Model

- If Δ is too small, it will not encompass an entire locality
- If Δ is too large it will encompass several localities
- If $\Delta = \infty \Rightarrow$ will encompass entire program

7 August 2021

© Copyright Virtual University of
Pakistan



Working Set Model

- If D is the total demand of frames in the system, then
$$D = \sum WSS_i$$
- If m is the total number of frames in the system, then
$$D > m \Rightarrow \text{Thrashing}$$
- Policy: if $D > m$, then suspend one of the processes

7 August 2021

© Copyright Virtual University of
Pakistan



Keeping Track of the Working Set

- Approximate with a **fixed interval timer** and a **reference bit**
- Example: $\Delta = 10,000$ references
 - Timer interrupts every 5000 references
 - Keep in memory 2 bits for each page
 - Whenever a timer interrupts, copy all reference bits and set their values to 0
 - After Δ references, if one of the bits in memory = 1 $\Rightarrow$ page in the working set

7 August 2021

© Copyright Virtual University of
Pakistan



Keeping Track of the Working Set

- Why is this not completely accurate?
- Improvement: Use 10 reference bits and interrupt every 1000 references

7 August 2021

© Copyright Virtual University of
Pakistan



Page-Fault Frequency

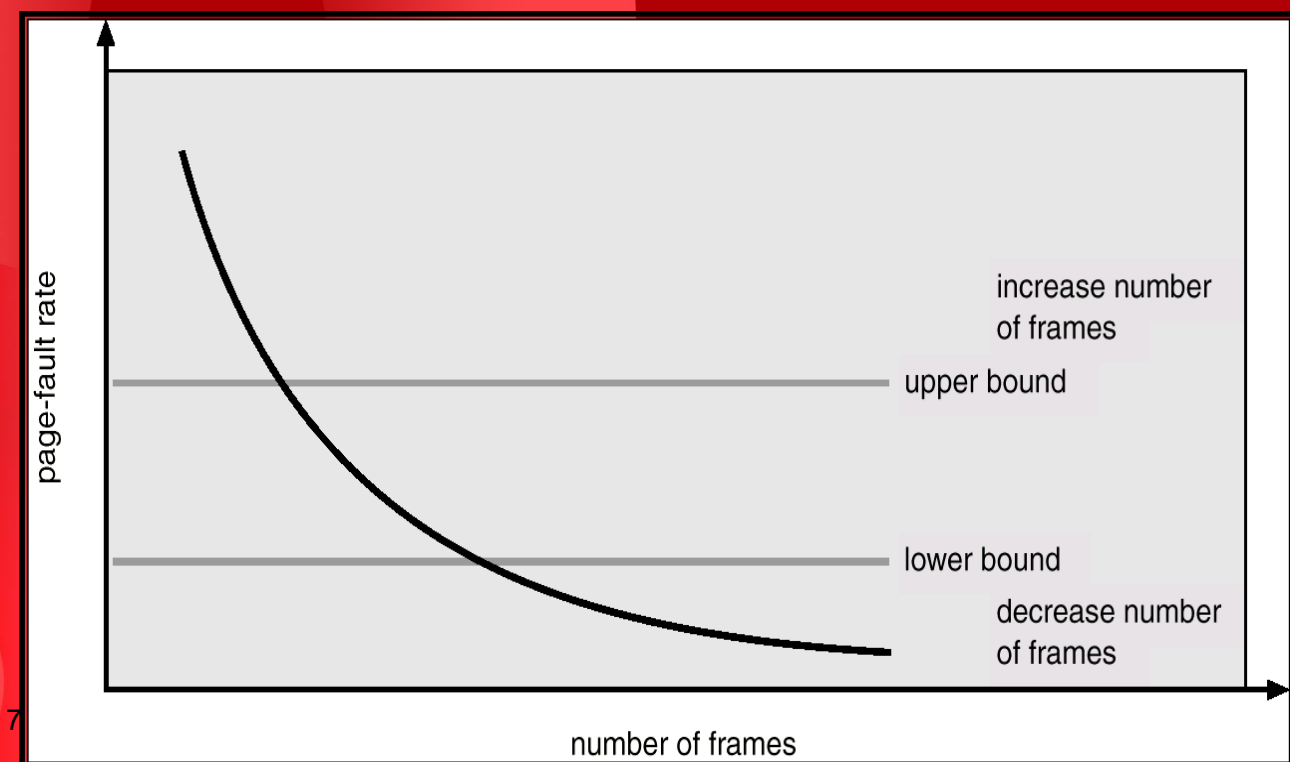
- Another method to control thrashing.
- OS keeps track of the upper and lower bounds on the page-fault rates of processes
- If page-fault rate falls below the lower limit, the processes loses frames. If page-fault rate goes above the upper limit, process gains frames.

7 August 2021

Copyright Virtual University of
Pakistan



Page-Fault Frequency Scheme



Other Considerations

- Prepaging
- Page size selection
 - Internal fragmentation
 - Page table size
 - Locality
 - I/O overhead
 - Reduced with small page size because locality improves

7 August 2021

© Copyright Virtual University of
Pakistan



Other Considerations

- Program structure
 - `int A[1024][1024];`
 - Matrix is stored in **row-major order**
 - Each row is stored in one page
 - Program 1 for (j = 0; j < 1024; j++)
for (i = 0; i < 1024; i++)
A[i,j] = 0;

7 August 2021

© Copyright Virtual University of
Pakistan



■ **1024 x 1024** page faults

Other Considerations

- **Program structure**

- `int A[1024][1024];`
- Matrix is stored in **row-major order**
- Each row is stored in one page
- Program 2 `for (i = 0; i < 1024; i++)
for (j = 0; j < 1024; j++)
A[i,j] = 0;`

- **1024 page faults**

7 August 2021

© Copyright Virtual University of
Pakistan



Example System

- A demand paging system with the following utilizations:
 - CPU = 20%
 - Paging disk = 97.7%
 - Other I/O devices = 5%
- Which of the following will improve CPU utilization?
 - Install a faster CPU
 - Increase degree of multiprogramming
 - Decrease degree of multiprogramming
 - Install more main memory

7 August 2021

© Copyright Virtual University of
Pakistan



Example

- Which of the following programming techniques and structures are “good” for a demand paged environment? Which are bad? Explain your answer.
 - Stack
 - Hash table
 - Sequential search
 - Binary search
 - Indirection
- Vector operations

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Thrashing
- The Working Set Model
- Page Fault Frequency Model
- Other Considerations
 - Prepaging
 - Page size
 - Program structure

7 August 2021

© Copyright Virtual University of
Pakistan



Examples

Operating Systems

Lecture 41

Syed Mansoor Sarwar

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 42

Syed Mansoor Sarwar



Agenda for Today

- Review of the previous lecture
- File Concept
- File Types
- File Operations
- Access Methods
- Directories
- Directory Operations
- Directory Structure

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 41

- Thrashing
- The Working Set Model
- Page Fault Frequency Model
- Other Considerations
 - Prepaging
 - Page size
 - Program structure
- Examples

7 August 2021

© Copyright Virtual University of
Pakistan



The File Concept

- Contiguous logical address space
- Types:
 - Data
 - Numeric
 - Character
 - Binary
 - Program

7 August 2021

© Copyright Virtual University of
Pakistan



File Structure

- **None** - sequence of words, bytes
- **Simple record structure**
 - Lines
 - Fixed length
 - Variable length
- **Complex Structures**
 - Formatted document
 - Relocatable load file

7 August 2021

© Copyright Virtual University of
Pakistan



File Structure

- Who decides?
 - Operating system
 - DOS/Windows
 - .exe, .com, .bat, ...
 - Program
 - UNIX/Linux

7 August 2021

© Copyright Virtual University of
Pakistan



File Attributes

- Name
- Type
- Location
- Size
- Protection
- Owner
- Time and date created
- Time and date last updated
- Read/write pointer value

7 August 2021

© Copyright Virtual University of
Pakistan



Where are Attributes Stored

- In the directory structure, as part of the **directory entry** for the file—DOS, Windows
- In a separate data structure; in UNIX/Linux this structure is known as the **inode** for the file

7 August 2021

© Copyright Virtual University of
Pakistan



Directory Entry

- DOS/Windows: File name and attributes
- UNIX/Linux: File name and inode number.
 - Name can be up to 255 characters in BSD UNIX compliant systems
 - Inode number—used to access file's inode

7 August 2021

© Copyright Virtual University of
Pakistan



Directory Entry

■ DOS/Windows

File Name	Date	...	FAT Pointer

■ UNIX/Linux

File Name	Inode Number

7 August 2021

© Copyright Virtual University of
Pakistan



File Operations

- Create — `creat`
- Open — `open`
- Write — `write`
- Read — `read`
- Reposition within file — `lseek`
- Delete — `unlink`
- Truncate — `creat` with a special flag
- Close — `close`

7 August 2021

© Copyright Virtual University of
Pakistan



File Types: Extensions

file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

7 August 2021



File Types in UNIX

- Ordinary file
- Directory
- Block-special file
- Character-special file
- Link file (`ln -s`)
- FIFO (`mkfifo` or `mknod`)
- Socket (in BSD-compliant systems—`socket`)

7 August 2021

© Copyright Virtual University of
Pakistan



File Operations

■ Sequential Access

- read next
- write next
- reset

■ Direct Access

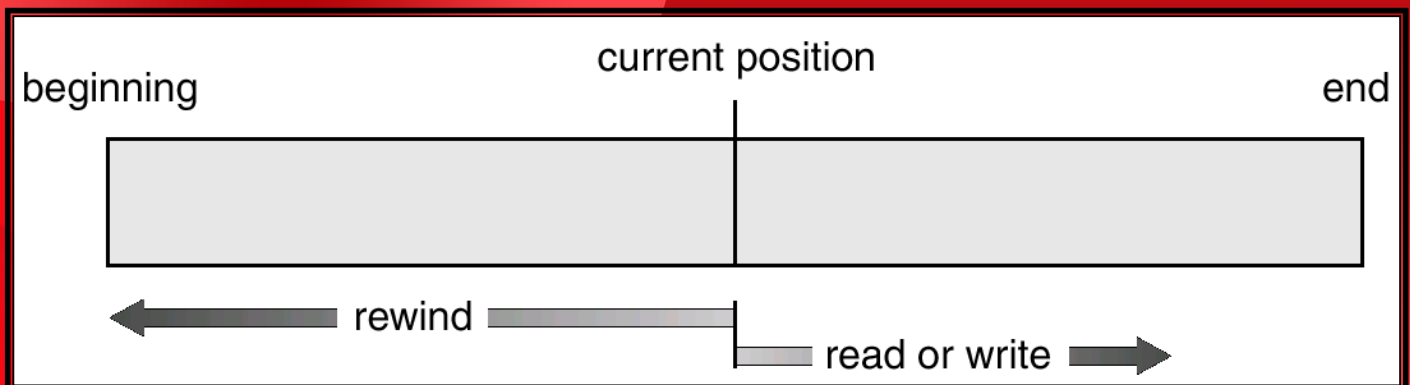
- read n (n = relative block number)
- write n
- position to n
 - read next
 - write next

7 August 2021

© Copyright Virtual University of
Pakistan



Sequential Access File



7 August 2021

© Copyright Virtual University of
Pakistan



Sequential Access on a Direct Access File

sequential access	implementation for direct access
<i>reset</i>	<i>cp = 0;</i>
<i>read next</i>	<i>read cp;</i> <i>cp = cp+1;</i>
<i>write next</i>	<i>write cp;</i> <i>cp = cp+1;</i>

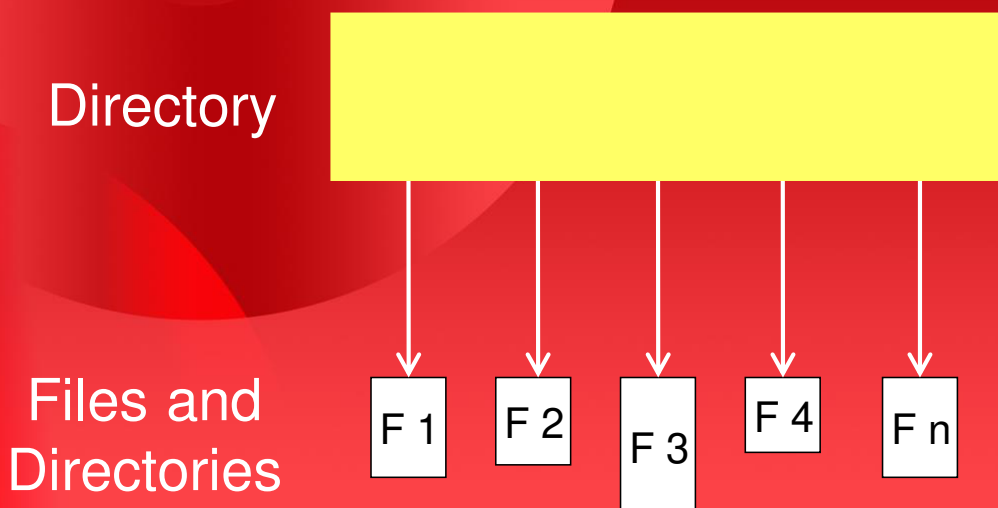
7 August 2021

© Copyright Virtual University of
Pakistan



Directory Structure

- A collection of directory entries



7 August 2021

© Copyright Virtual University of
Pakistan



Directory Operations

- Create — `mkdir`
- Open — `opendir`
- Read — `readdir`
- Rewind — `rewinddir`
- Close — `closedir`
- Delete — `rmdir`
- Change Directory — `cd`
- List — `ls`
- Search

7 August 2021

© Copyright Virtual University of
Pakistan



Directory Structure

- **Efficient Searching**
- **Naming** — convenient to users
 - Two users can have same name for different files
 - The same file can have several different names
- **Grouping** — logical grouping of files by properties, (e.g., all Java programs, all games, ...)

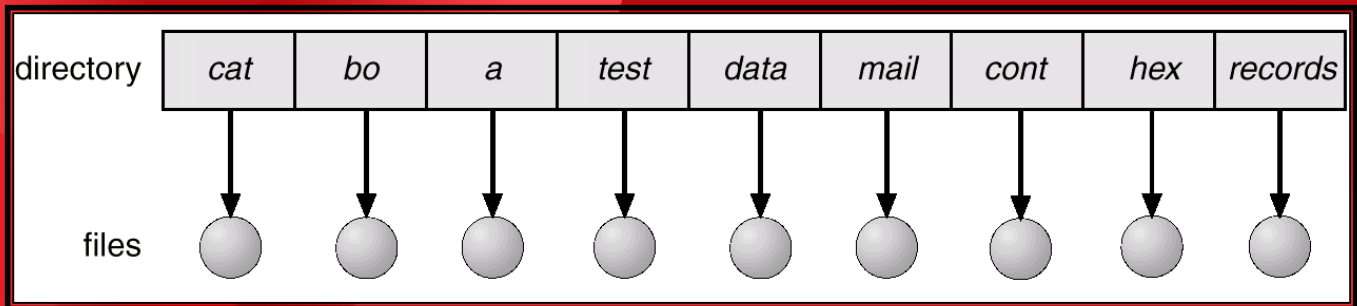
7 August 2021

© Copyright Virtual University of
Pakistan



Single-Level Directory

- A single directory for all users.



- Naming problem
- Grouping problem

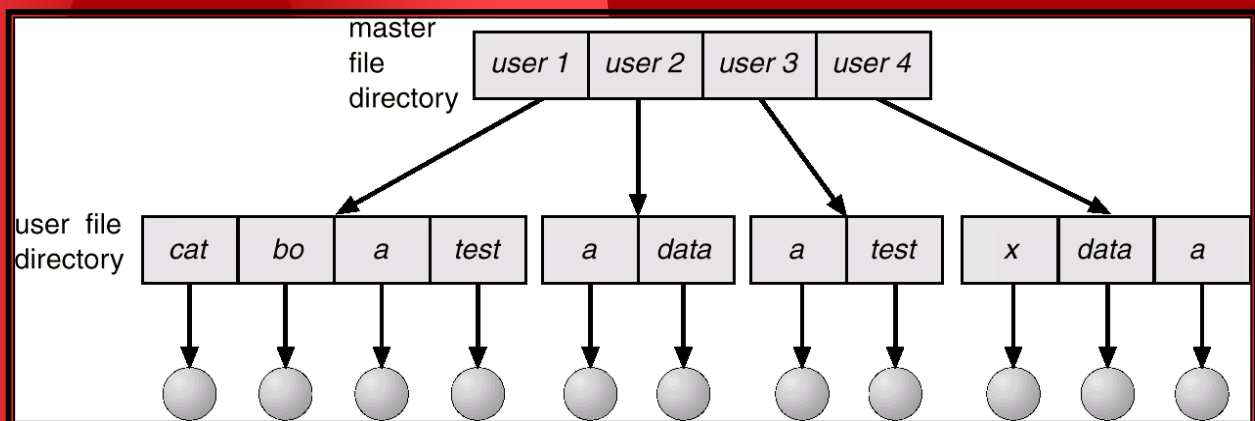
7 August 2021

© Copyright Virtual University of
Pakistan



Two-Level Directory

- Separate directory for each user



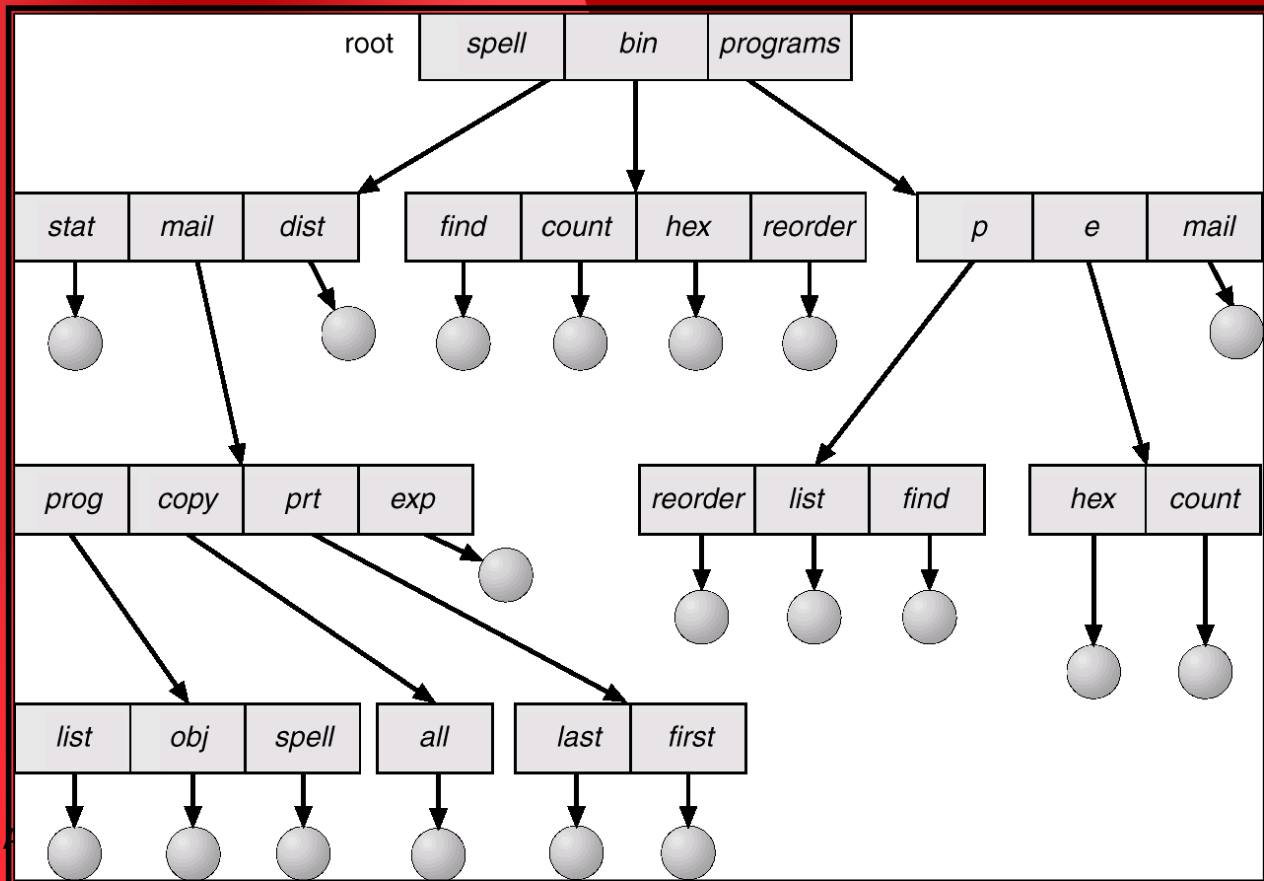
- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

7 August 2021

© Copyright Virtual University of
Pakistan



Tree Directory



Tree Directory

- Efficient searching
- Grouping Capability
- Root directory
- Home directory
- Current/working directory

Tree Directory

- UNIX / Linux
 - Root directory (/)
 - Home directory
 - ~, \$HOME, \$home
 - cd ~
 - cd
 - Current/working directory (.)
 - pwd
 - Parent of Current Directory (..)

7 August 2021

© Copyright Virtual University of
Pakistan



Tree Directory

- Absolute Pathname
 - Starts with the root directory
 - /etc, /bin, /usr/bin,
/etc/passwd,
/home/students/ibraheem
- Relative Pathname
 - Starts with the current directory or a user's home directory
 - ~/courses/cs604, ./a.out

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- File Concept
- File Types
- File Operations
- Access Methods
- Directories
- Directory Operations
- Directory Structure

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 42

Syed Mansoor Sarwar

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 43

Syed Mansoor Sarwar



Agenda for Today

- Review of the previous lecture
- Directory Structures
- Links in UNIX/Linux
- File System Mounting
- File Sharing



Review of Lecture 42

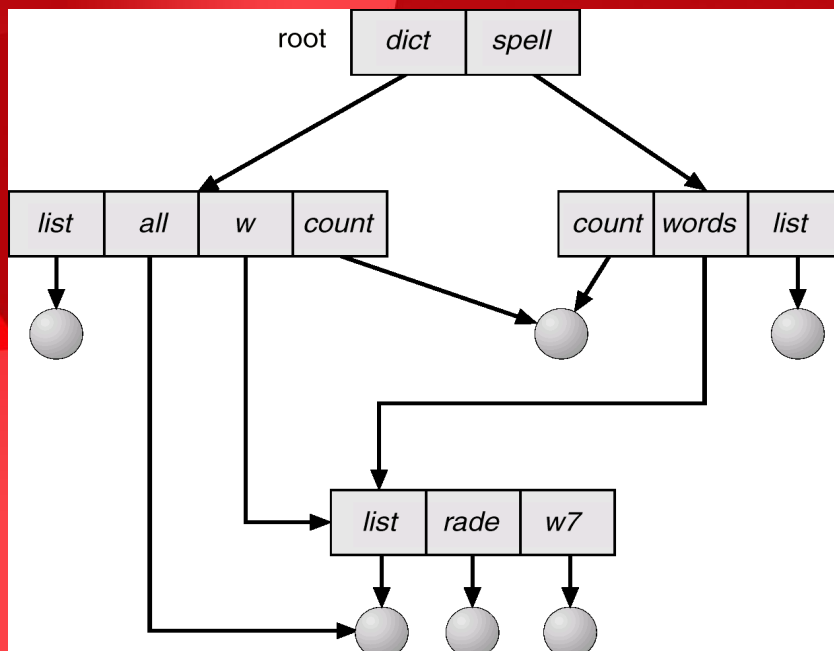
- Review of the previous lecture
- File Concept
- File Types
- File Operations
- Access Methods
- Directories
- Directory Operations
- Directory Structure

© Copyright Virtual University of
Pakistan



Acyclic-Graph Directories

- Have shared subdirectories and files.



7 August 2021

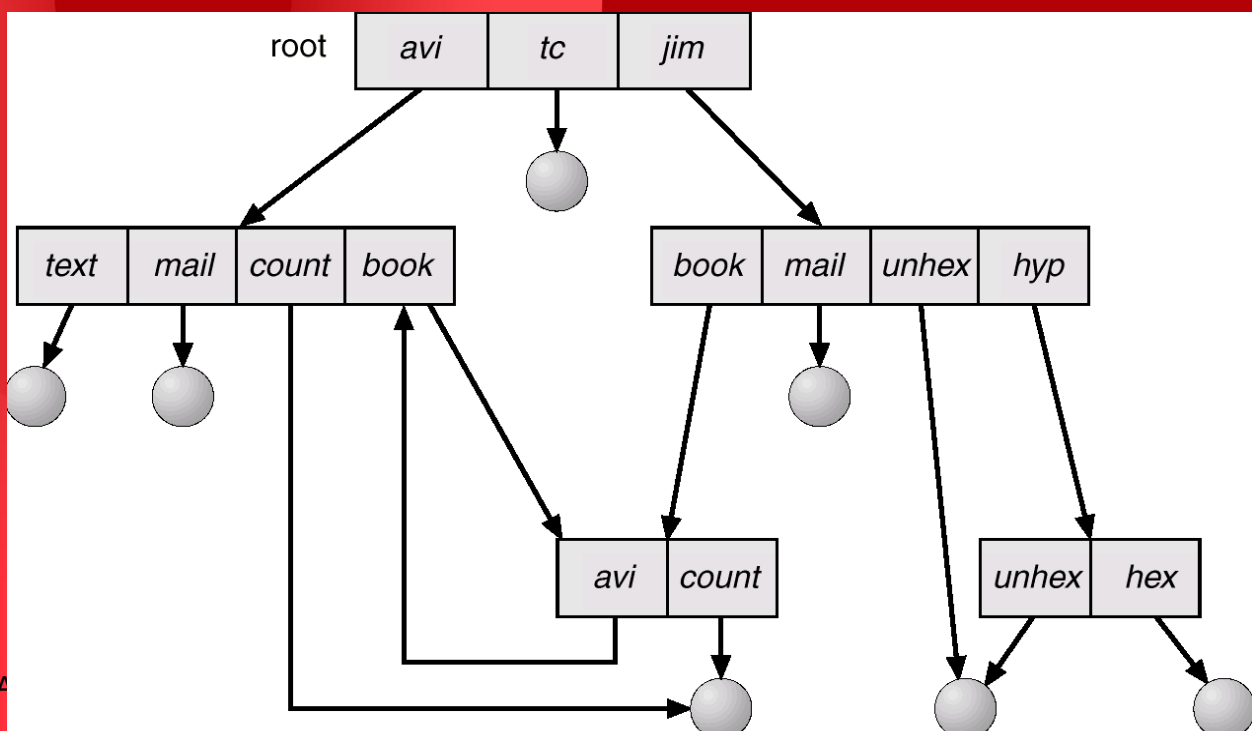


Acyclic-Graph Directories

- Two different names (**aliasing**)
- If *dict* deletes *count* $\Rightarrow$ **dangling pointer**
- Solutions:
 - **Backpointers**, so we can delete all pointers
 - **Link count solution**

7 August 2011 © Copyright Virtual University of Pakistan

General Graph Directory



General Graph Directory

- How do we guarantee no cycles?
 - Allow only links to files not subdirectories
 - Every time a new link is added use a **cycle detection algorithm** to determine whether it is OK
 - **Garbage collection**

7 August 2021

© Copyright Virtual University of
Pakistan



Links in UNIX

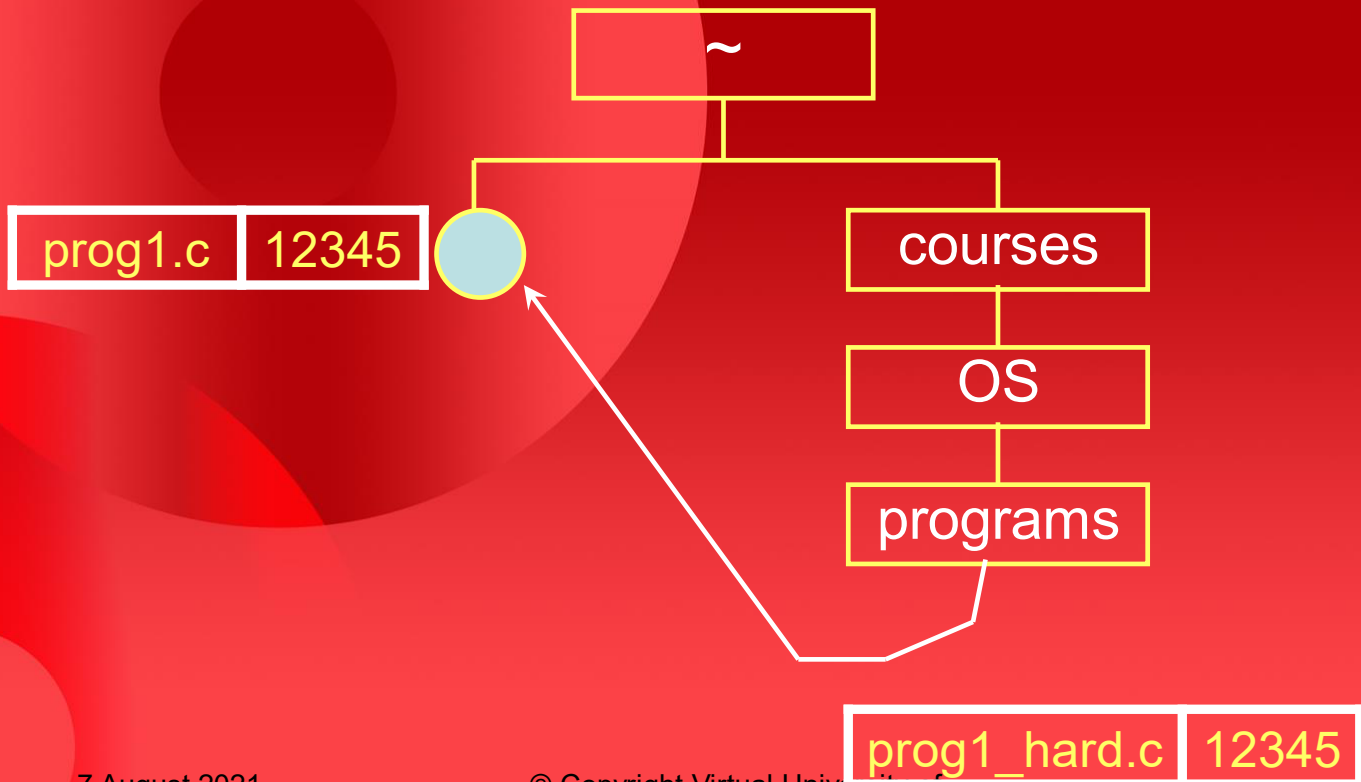
- Two types of links:
 - **Hard links**
 - **Soft (symbolic) links**
- The `ln` command
 - `ln [options] existing-file new-file`
 - `ln [options] existing-file-list directory`
- Examples:
 - `ln -/prog1.c ~/courses/OS/programs/prog1_hard.c`
 - `ln -s ~/courses/OS/programs/prog2.c ~/prog2_soft.c`

7 August 2021

© Copyright Virtual University of
Pakistan



Hard Links

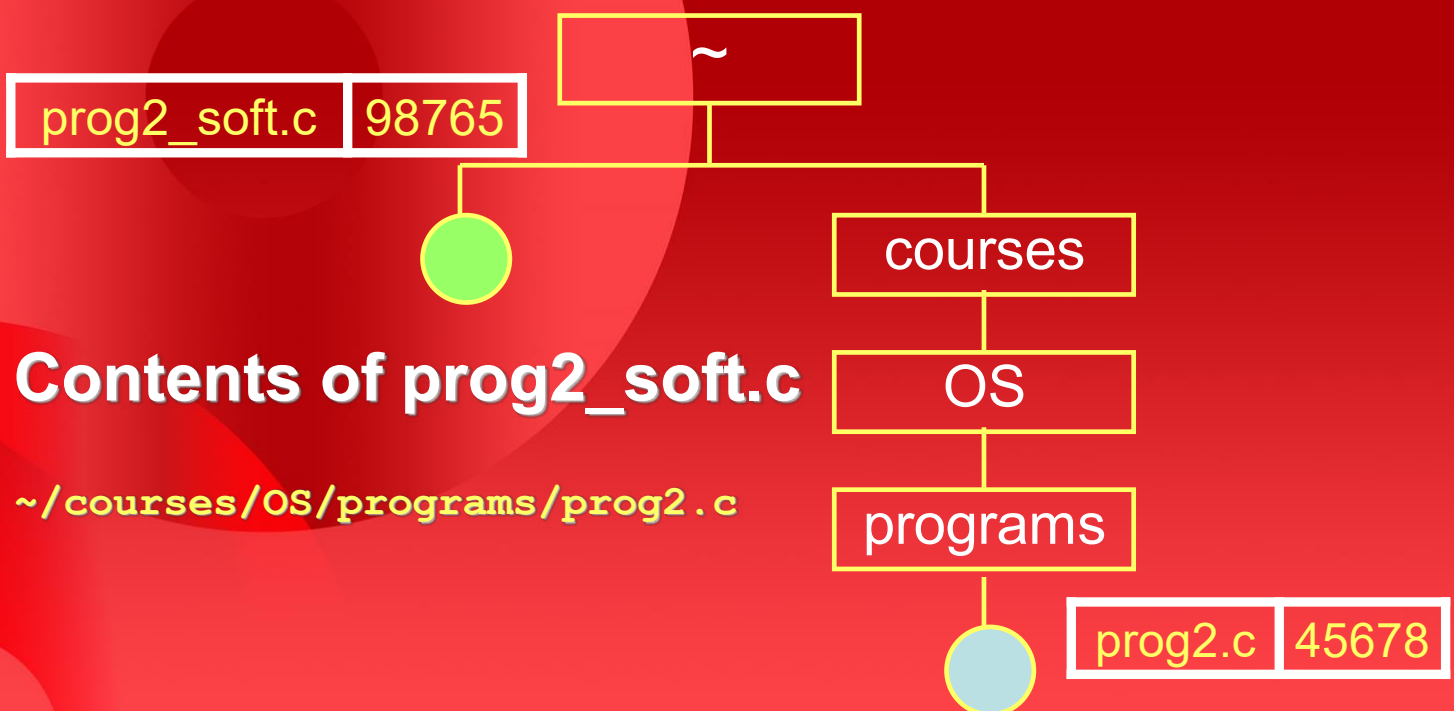


7 August 2021

© Copyright Virtual University of
Pakistan



Soft Links



Contents of prog2_soft.c

~ / courses / OS / programs / prog2 . c

7 August 2021

© Copyright Virtual University of
Pakistan



Links in UNIX

■ Hard links

- A directory entry for the existing file is created—there is still only one file
- Both entries have the same inode number
- Link count is incremented by one in the inode for the file
- No hard links to directories
- No hard links across file systems
- File is removed from the file system only if its hard link count is 0

7 August 2021

© Copyright Virtual University of
Pakistan



Links in UNIX

■ Soft (symbolic) links

- A file of type 'link' is created, which contains the pathname for the existing file as specified in the `ln` command
- The existing file and the new (link) files have different inode numbers
- When the existing file is removed, you have a 'dangling pointer' to it in the link file
- May have soft links to directories
- May have soft links across file systems

7 August 2021

© Copyright Virtual University of
Pakistan



File System Mounting

- A file system is best visualized as a tree, rooted at /. /dev, /etc, /usr, and other directories in the root directory are branches, which may have their own branches, such as /etc/passwd, /usr/local, and /usr/bin.

7 August 2021

© Copyright Virtual University of
Pakistan



File System Mounting

- Filling up the root file system is not a good idea, so splitting /var from / is a good idea.
- Another common reason to contain certain directory trees on other file systems is if they are to be housed on separate physical disks, or are separate virtual disks, or CDROM drives.

7 August 2021

© Copyright Virtual University of
Pakistan



File System Mounting

- **Mounting** makes file systems, files, directories, devices, and special files available for use at a particular location
- **Mount point** is the actual location from which the file system is mounted and accessed.

7 August 2021

© Copyright Virtual University of
Pakistan



File System Mounting

- You can mount a file or directory if you have access to the file or directory being mounted and write permission for the mount point
- Types of mounts:
 - **Remote mount**
 - **Local mount**

7 August 2021

© Copyright Virtual University of
Pakistan



File System Mounting

- **Remote mounts** are done on a remote system on which data is transmitted over a telecommunication line
- **Local mounts** are mounts done on your local system

7 August 2021

© Copyright Virtual University of
Pakistan



Mounting in UNIX

- All files accessible in a Unix system are arranged in one big tree, the file hierarchy, rooted at /. These files can be spread out over several devices.
- The `mount` command serves to attach the file system found on some device to the big file tree. Conversely, the `umount` command will detach it again.

7 August 2021

© Copyright Virtual University of
Pakistan



Mounting in UNIX

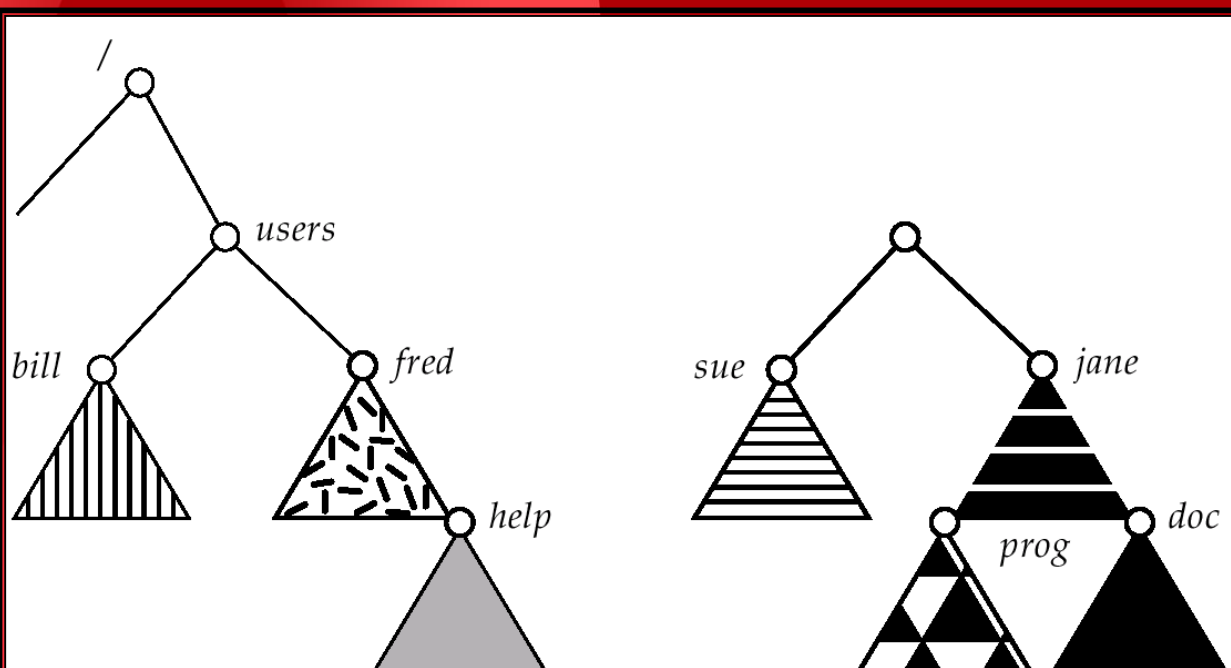
- **mount -t** *type device dir*
- Tells the kernel to attach the file system found on *device* (which is of type *type*) at the directory *dir*
- The previous contents (if any) and owner and mode of *dir* become invisible
- As long as this file system remains mounted, the pathname *dir* refers to the root of the file system on *device*.

7 August 2021

© Copyright Virtual University of
Pakistan



Mounting in UNIX

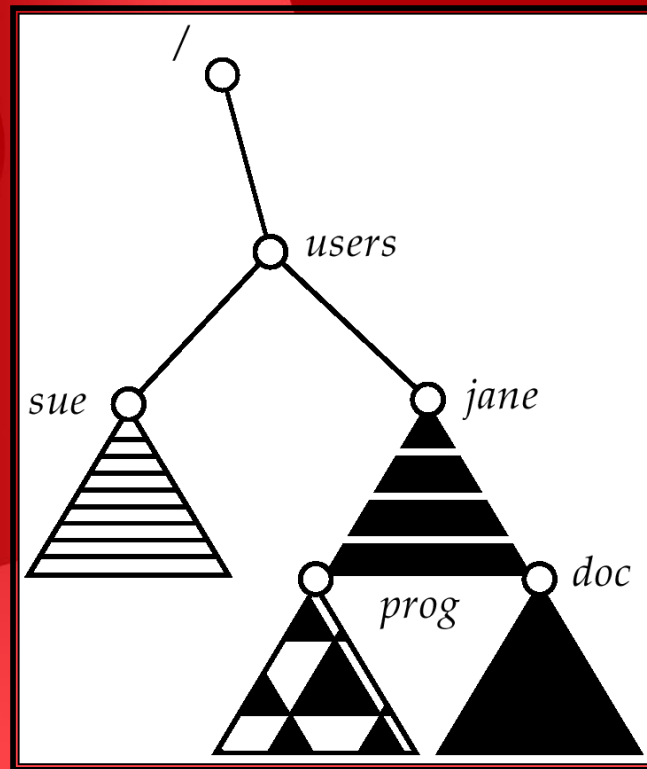


Existing Tree

Unmounted Filesystem



Mounting in UNIX



7 August

New Tree after Mounting Filesystem



File System Space Usage

On SuSE Linux

\$ df

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
/dev/hda3	74837584	11127072	59908892	16%	/
tmpfs	257948	12	257936	1%	/dev/shm
/dev/hda1	19976	6960	11985	37%	/boot
inbox:/var/spool/mail	66602516	3319996	59899232	6%	
/var/spool/mail					
upfile1a:/usr1.a	230044816	70533680	147825456	33%	/usr1.a
upfile2a:/usr2.a	230044816	118228296	100130840	55%	/usr2.a
upibma:/usr3.a	16713676	804252	15039103	6%	/usr3.a
upfile4a:/usr4.a	230044816	14594384	203764752	7%	/usr4.a
\$					

7 August 2021

© Copyright Virtual University of
Pakistan



File System Space Usage

On Solaris 2

```
$ df -v
```

Mount	Dir	Filesystem	blocks	used	free	%used
/		/dev/dsk/c0t12d	7557677	2484225	4997876	34%
/proc		/proc	0	0	0	0%
/etc/mnttab		mnttab	0	0	0	0%
/dev/fd		fd	0	0	0	0%
/var/run		swap	510103	22	510081	1%
/tmp		swap	683241	173160	510081	26%
/oldexport		/dev/dsk/c0t8d0	4668856	4229110	393058	92%
/export/ho		/dev/dsk/c0t12d	23684712	21714309	1733556	93%

```
$
```

7 August 2021

© Copyright Virtual University of
Pakistan



File Sharing

- Sharing of files on multi-user systems is desirable.
- Sharing may be done through
 - **Duplicating files**
 - **Common login** for members of a team
 - Setting appropriate **access permissions**
 - **Common groups** for members of a team
 - **Links**

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Directory Structures
- Links in UNIX/Linux
- File System Mounting
- File Sharing

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 43

Syed Mansoor Sarwar

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 44

Syed Mansoor Sarwar



Agenda for Today

- Review of the previous lecture
- File Sharing
- File Protection
- In-Memory Data Structures
- Space Allocation Techniques
 - Contiguous, Linked, Index
 - UNIX Allocation
 - File Allocation Table (FAT)



Review of Lecture 43

- Directory Structures
- Links in UNIX/Linux
- File System Mounting

7 August 2021

© Copyright Virtual University of
Pakistan



File Sharing

- Sharing of files on multi-user systems is desirable.
- Sharing may be done through
 - **Duplicating files**
 - **Common login** for members of a team
 - Setting appropriate **access permissions**
 - **Common groups** for members of a team
 - **Links**

7 August 2021

© Copyright Virtual University of
Pakistan



Protection

- File owner/creator should be able to control
 - What can be done
 - By whom
- Types of access
 - Read
 - Write
 - Execute
 - Append
 - Delete
 - List

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX Protection

- Mode of access: **read**, **write**, and **execute** (r, w, x)
- Three classes of users: **owner**, **group**, and **others**

		r	w	x
a) Owner access: 7	⇒	1	1	1
b) Group access: 6	⇒	1	1	0
c) Public access: 1	⇒	0	0	1

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX Protection

- Ask system administrator to create a **group** and add some users to the group
- For a particular file or subdirectory, set appropriate **access permissions** for proper protection

7 August 2021

© Copyright Virtual University of
Pakistan



Default Permissions

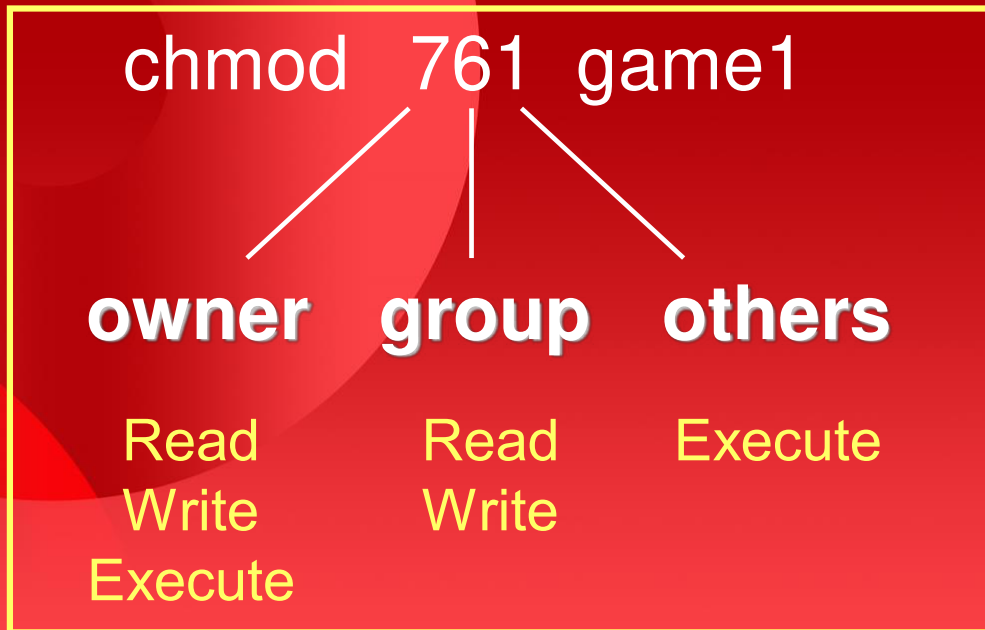
- 777 for executable files and directories
- 666 for text files
- Can be changed with the **umask** command

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX Protection

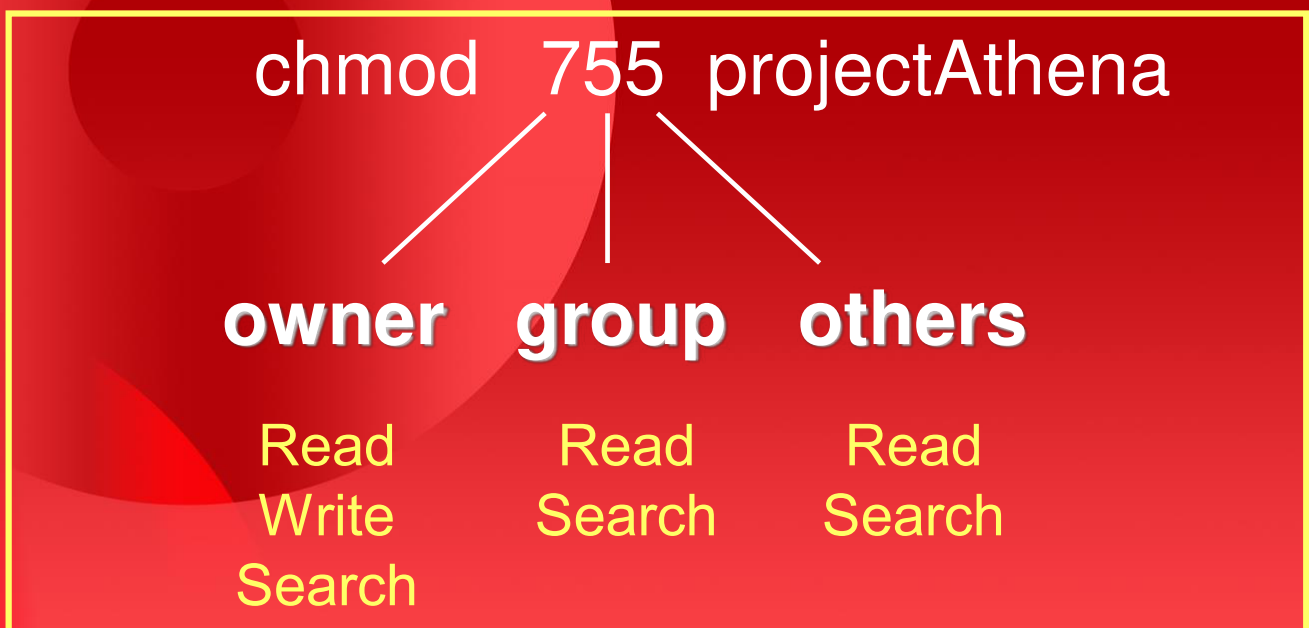


7 August 2021

© Copyright Virtual University of
Pakistan



UNIX Protection



7 August 2021

© Copyright Virtual University of
Pakistan



UNIX Protection

■ Sample commands

```
chmod 700 ~
```

```
chmod 744 ~/file
```

```
chmod 755 ~/directory
```

```
ls -l ~
```

```
ls -ld ~
```

```
ls -l prog1.c
```

```
ls -ld ~/courses
```

7 August 2021

© Copyright Virtual University of
Pakistan



UNIX Protection

- The **umask** Command
- Sets default permissions on newly created files and directories as **(default permissions – mask value)**

```
umask
```

```
umask 022
```

```
touch temp1
```

```
ls -l temp1
```

7 August 2021

© Copyright Virtual University of
Pakistan



File Control Block

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks

■ UNIX inode

7 August 2021

© Copyright Virtual University of
Pakistan



In-Memory Data Structures

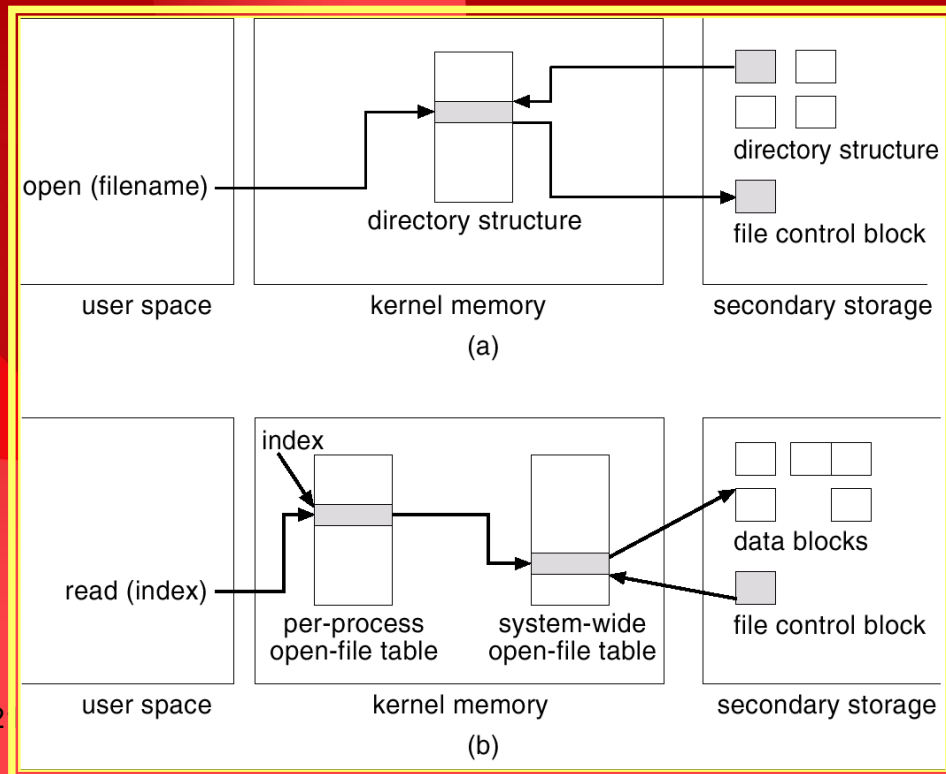
- The following upper-level data structures needed for file system support.
- Mapping directory entry for the file to the file control block at the time of opening a file.
- Mapping used while reading or writing a file

7 August 2021

© Copyright Virtual University of
Pakistan



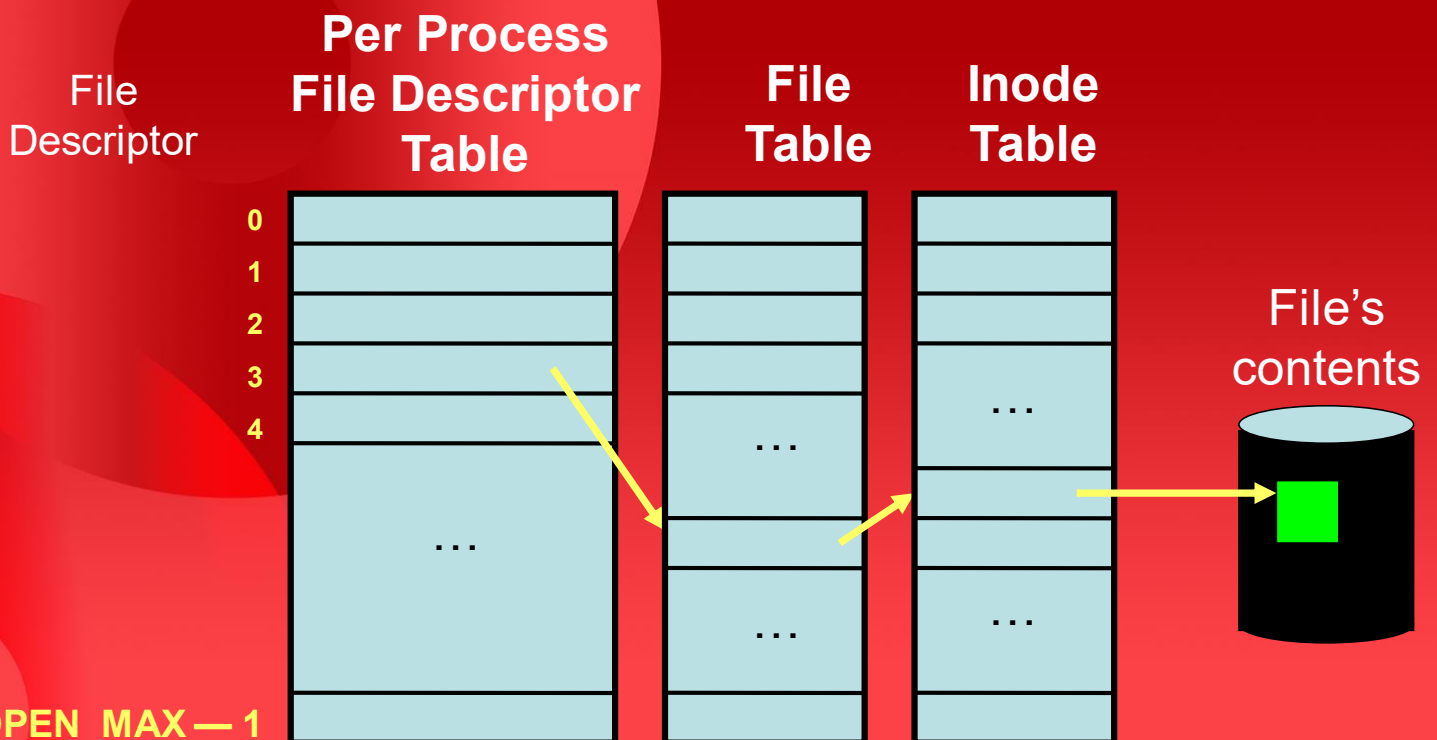
In-Memory Data Structures



7 August 2021



File Descriptor to File Contents



OPEN_MAX — 1
7 August 2021

© Copyright Virtual University of Pakistan



Space Allocation Methods

- **Contiguous allocation**
- **Linked allocation**
- **Indexed allocation**

7 August 2021

© Copyright Virtual University of
Pakistan



Contiguous Allocation

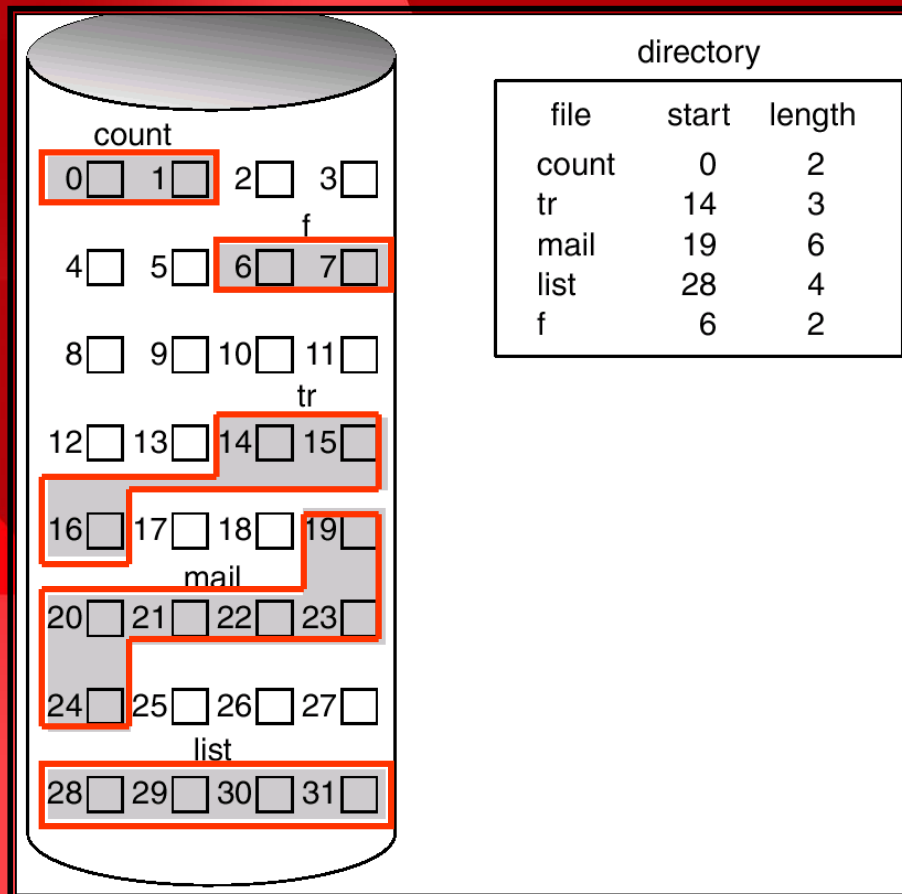
- Each file occupies a set of contiguous blocks on the disk
- Best-fit, first-fit, or worst-fit algorithm
- Directory entry contains starting block number and file size (in blocks)
- Good sequential and random access
- External fragmentation
- User needs to declare file size
- Expensive file growth

7 August 2021

© Copyright Virtual University of
Pakistan



Contiguous Allocation



7 August 2021



Linked Allocation

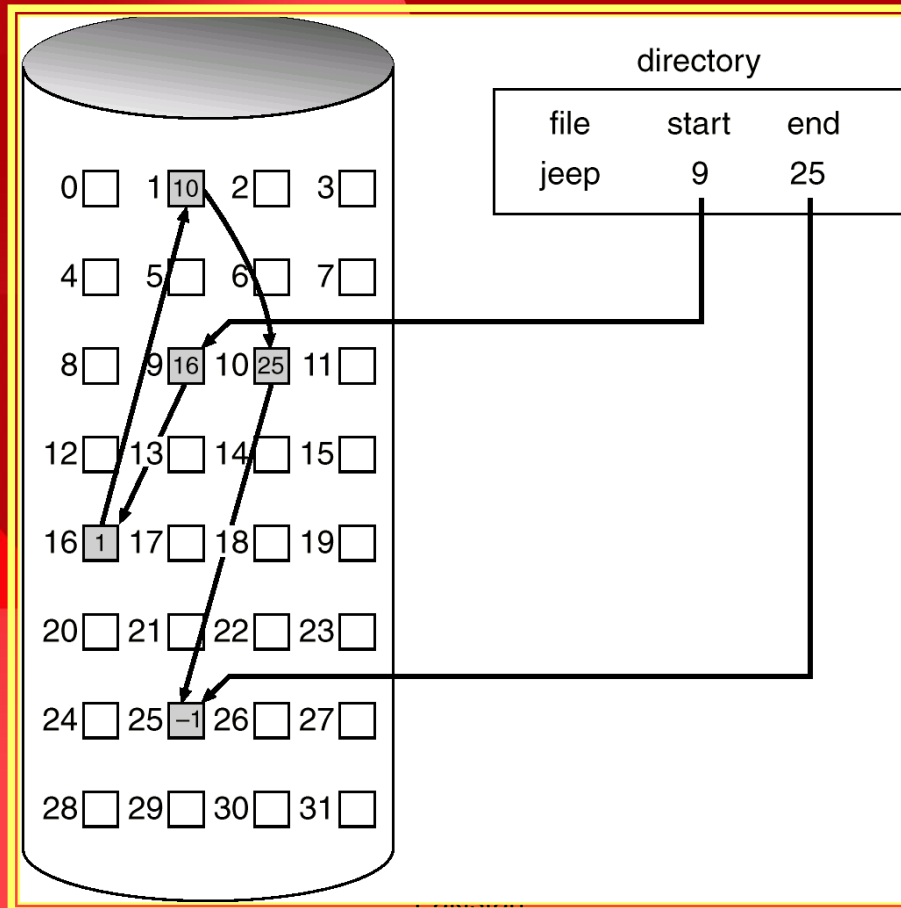
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.
- Simple – need only starting address
- No wastage of space
- No random access

7 August 2021

© Copyright Virtual University of
Pakistan



Linked Allocation

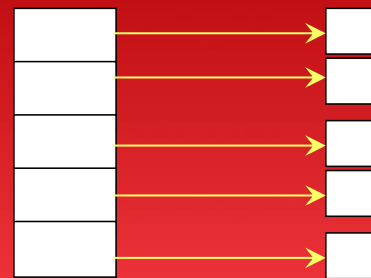


7 August 2021



Index Allocation

- Brings all pointers together into the **index block**
- Logical view



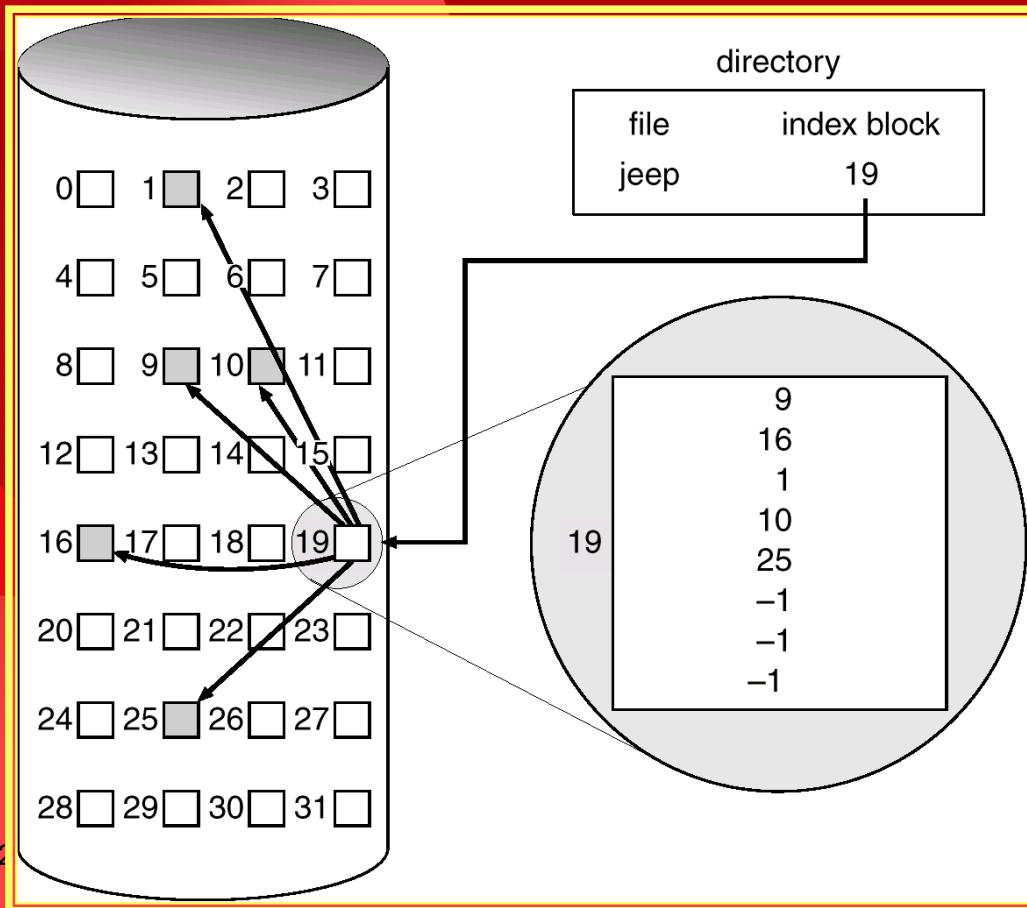
Index
table

Data
blocks

7 August 2021



Index Allocation



Recap of Lecture

- File Sharing
- File Protection
- In-Memory Data Structures
- Space Allocation Techniques
 - Contiguous, Linked, Index

Operating Systems

Lecture 44

Syed Mansoor Sarwar

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 45

Syed Mansoor Sarwar



Agenda for Today

- Review of the previous lecture
- Space Allocation Techniques
- Free Space Management
- Disk Structure and Scheduling
 - Access time and disk bandwidth
 - Disk scheduling algorithms (FCFS, SSTF, Scan, C-Scan, Look, C-Look)
- Course Recap
- Future Directions

7 August 2021

© Copyright Virtual University of
Pakistan



Review of Lecture 44

- File Sharing and Protection
 - Read, write, and execute permissions
 - User, group, and others
 - `chmod`, `ls -l`, `ls -ld`, `umask`, `ls -i`
- Per Process File Descriptor Table
- Space Allocation Techniques
 - Contiguous, Linked, Index

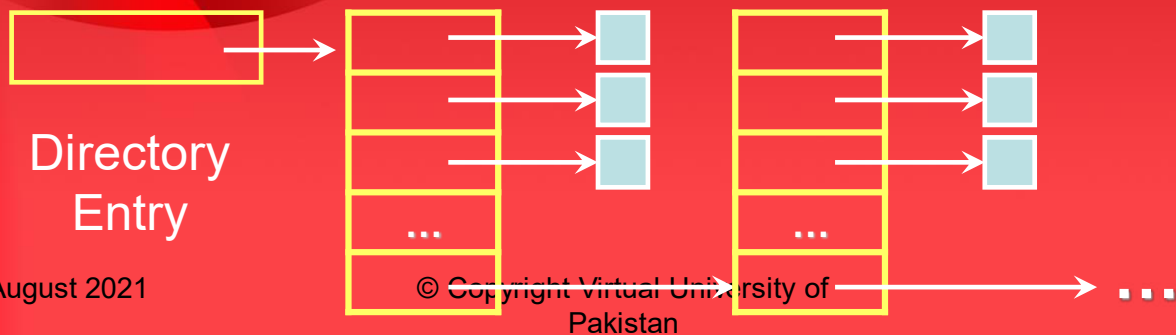
7 August 2021

© Copyright Virtual University of
Pakistan



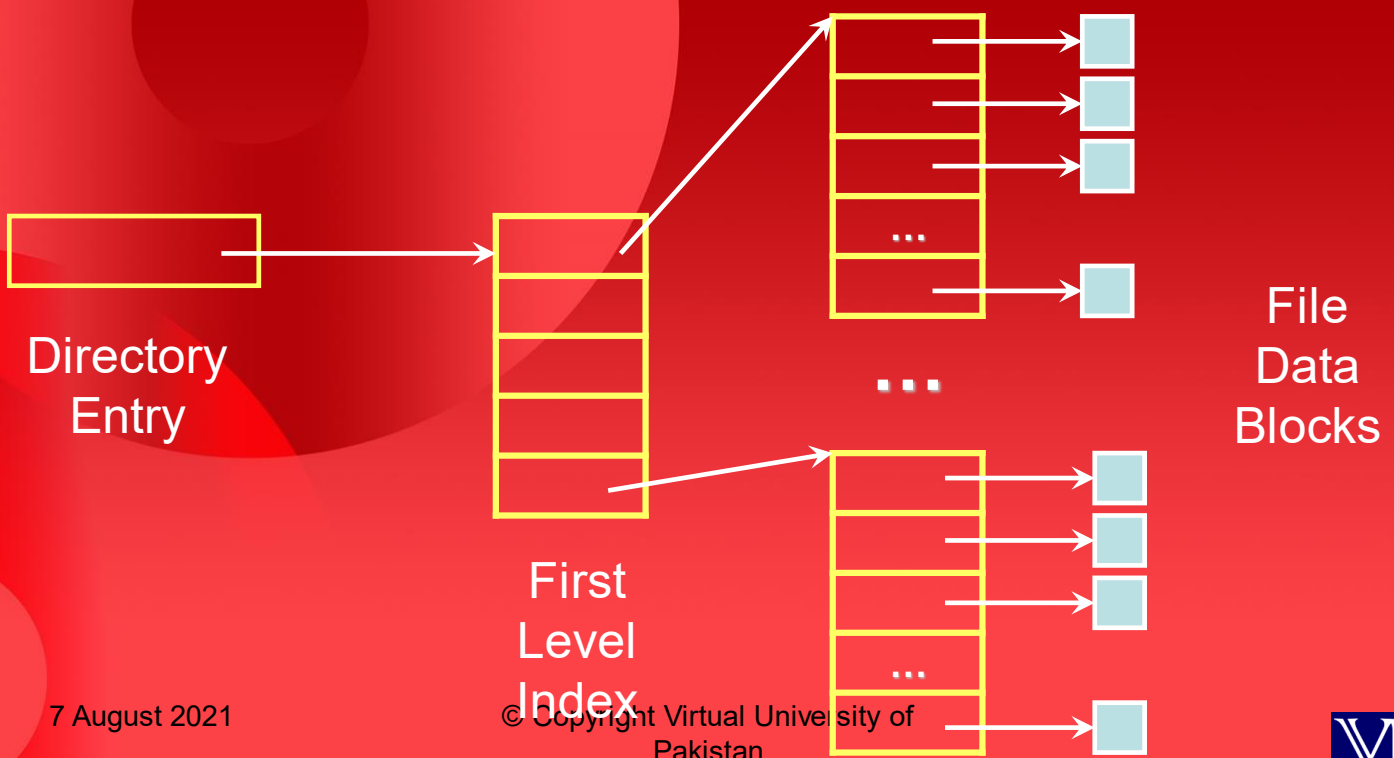
Index Allocation

- Need index table
- Random access
- **Linked scheme** – Linked list of index blocks

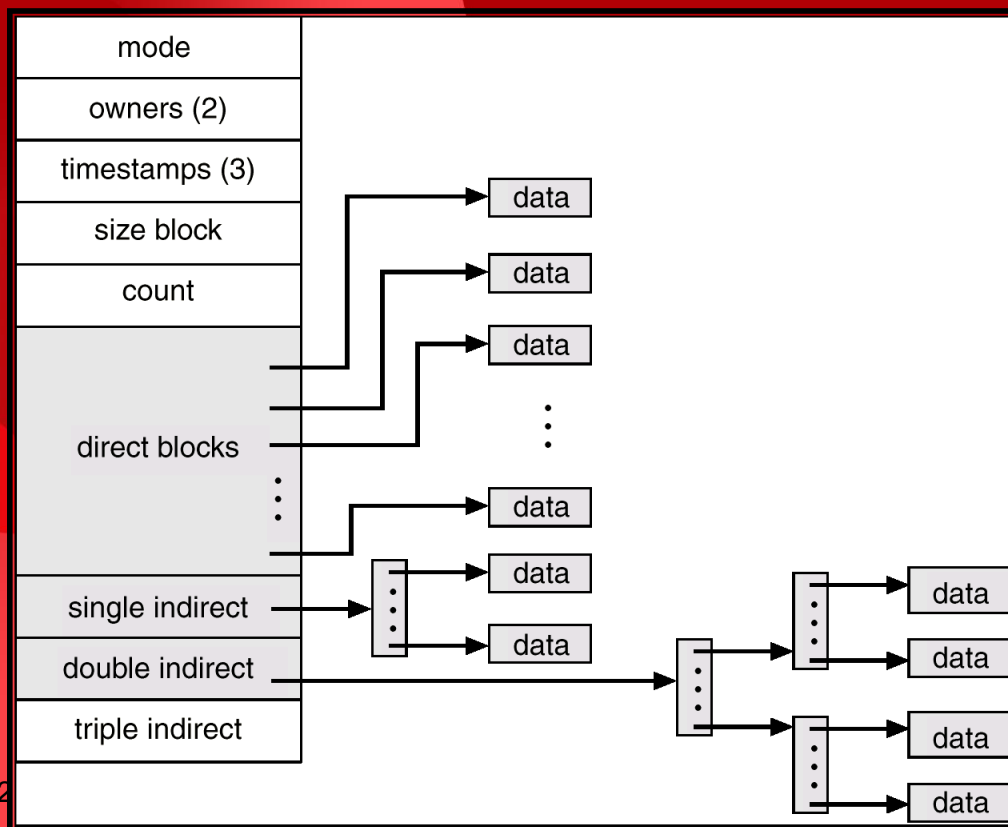


Index Allocation

- **Two-level index table**



UNIX Allocation

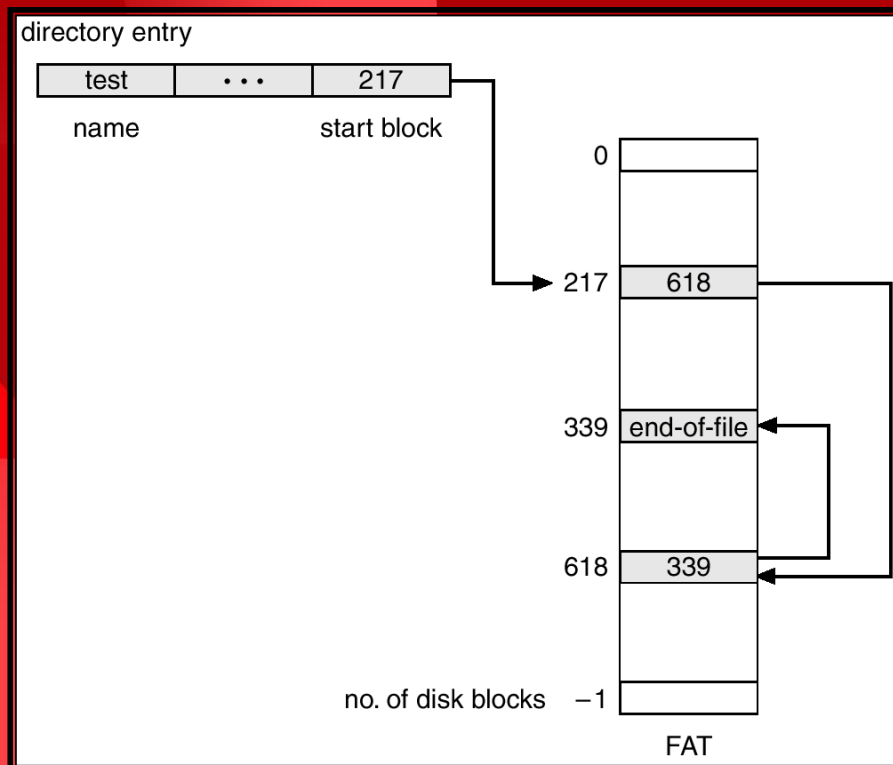


7 August 2021

UNIX Allocation

- 4 KB block
- 4-byte disk pointers
- 10 direct pointers to file blocks
- Maximum file size?
- Amount of space needed to store pointers?

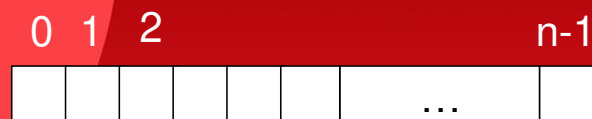
File Allocation Table (FAT)



7 August 2021

Free-Space Management

■ Bit vector



$$\text{bit}[i] = \begin{cases} 0 \Rightarrow \text{block}[i] \text{ is free} \\ 1 \Rightarrow \text{block}[i] \text{ is occupied} \end{cases}$$

■ Block number calculation

(number of bits per word) * (number of 0-value words) +
offset of first 1 bit

7 August 2021

Free-Space Management

- Example for **overhead of bit map**

Block size = 4 KB = 2^{12} bytes

Disk size = 40 GB = $40 * 2^{30}$ bytes

Overhead = $40 * 2^{30} / 2^{12} = 40 * 2^{18}$ bits
= $40 * 32$ KB = 1280 KB

- Easy to get contiguous files

7 August 2021

© Copyright Virtual University of
Pakistan



Free-Space Management

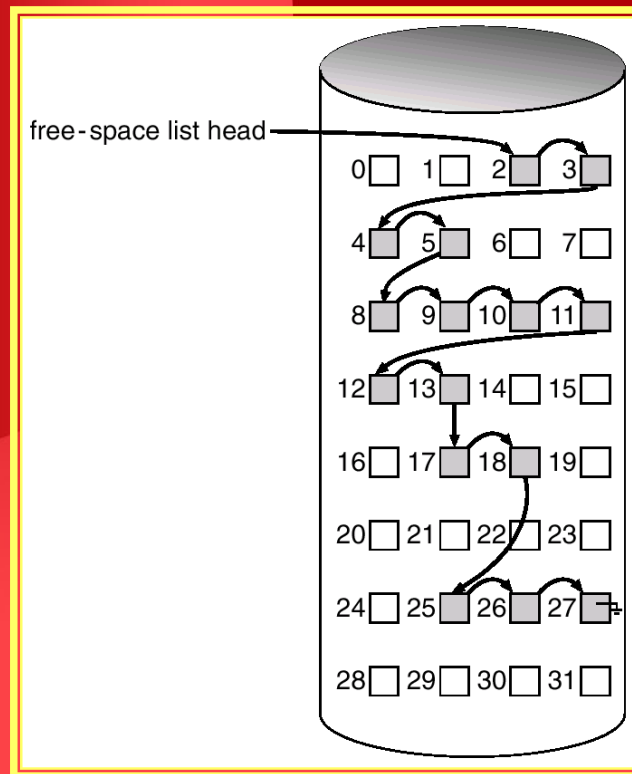
- **Linked list** (free list)
 - Cannot get contiguous space easily
 - Overhead (number of pointers used to maintain free space)

7 August 2021

© Copyright Virtual University of
Pakistan



Free-Space Management



7 August 2021



Free-Space Management

■ Grouping

- First (n-1) pointers to free blocks and last pointer to another such block

■ Counting

- An entry contain address of the first free block and number of free contiguous blocks that follow the first block—good for contiguous allocation

7 August 2021

© Copyright Virtual University of Pakistan



I/O Operations

- Number of I/O operations (inserting, deleting, and reading a file block) needed for the various allocation schemes
- Assumptions
 - Directory, Bit-map, and index blocks are in the main memory
 - Worst-case and best-case scenarios

7 August 2021

© Copyright Virtual University of
Pakistan



I/O Operations

- File size of 100 blocks
- Insert a block after the 50th block
- Read 50th block
- Insert 101st block
- Delete 50th block
- Assumptions
 - Directory, Bit-map, and index blocks are in the main memory
 - Worst-case and best-case scenarios

7 August 2021

© Copyright Virtual University of
Pakistan



Secondary Storage Management

File Management System

I/O System

Secondary Storage Management System

- Maintains the file system, its directories, and keeps track of free secondary storage space
- Provides device drivers to control transfer of data between memory and secondary storage devices
- Optimizes the completion of I/O tasks by employing algorithms to facilitate efficient disk usage

Disk Structure

- Disk drives are addressed as large 1-dimensional arrays of **logical blocks**, the smallest units of transfer (usually 512 bytes).
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially.
 - Block 0 is the first sector of the first track on the outermost cylinder.
 - Mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

Disk Scheduling

- **Disk Access Time** has two major components
 - **Seek time** is the time for the disk heads to the desired cylinder (track)
 - **Rotational latency** is the time waiting for the disk to rotate the desired sector under the disk head
- Want to minimize seek time

7 August 2021

© Copyright Virtual University of
Pakistan



Disk Scheduling

- **Disk bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer
- We can improve access time and bandwidth by scheduling the servicing of disk I/O request in a good order.

7 August 2021

© Copyright Virtual University of
Pakistan



Disk Scheduling

- Several algorithms exist to schedule the servicing of disk I/O requests. These include:
 - **First-Come-First-Serve (FCFS)**
 - **Shortest Seek Time First (SSTF)**
 - **Scan**
 - **Circular Scan (C-Scan)**
 - **Look**
 - **C-Look**

7 August 2021

© Copyright Virtual University of
Pakistan



FCFS

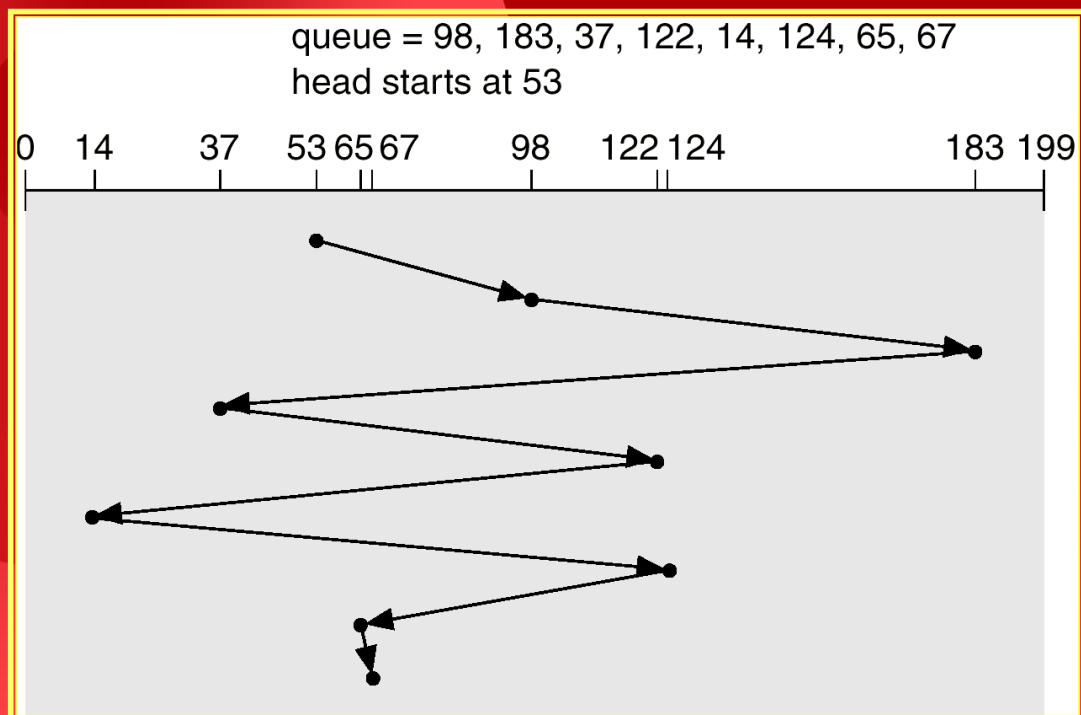


Illustration shows total head movement of 640 cylinders.

7 August 2021

© Copyright Virtual University of
Pakistan



Shortest Seek Time First (SSTF)

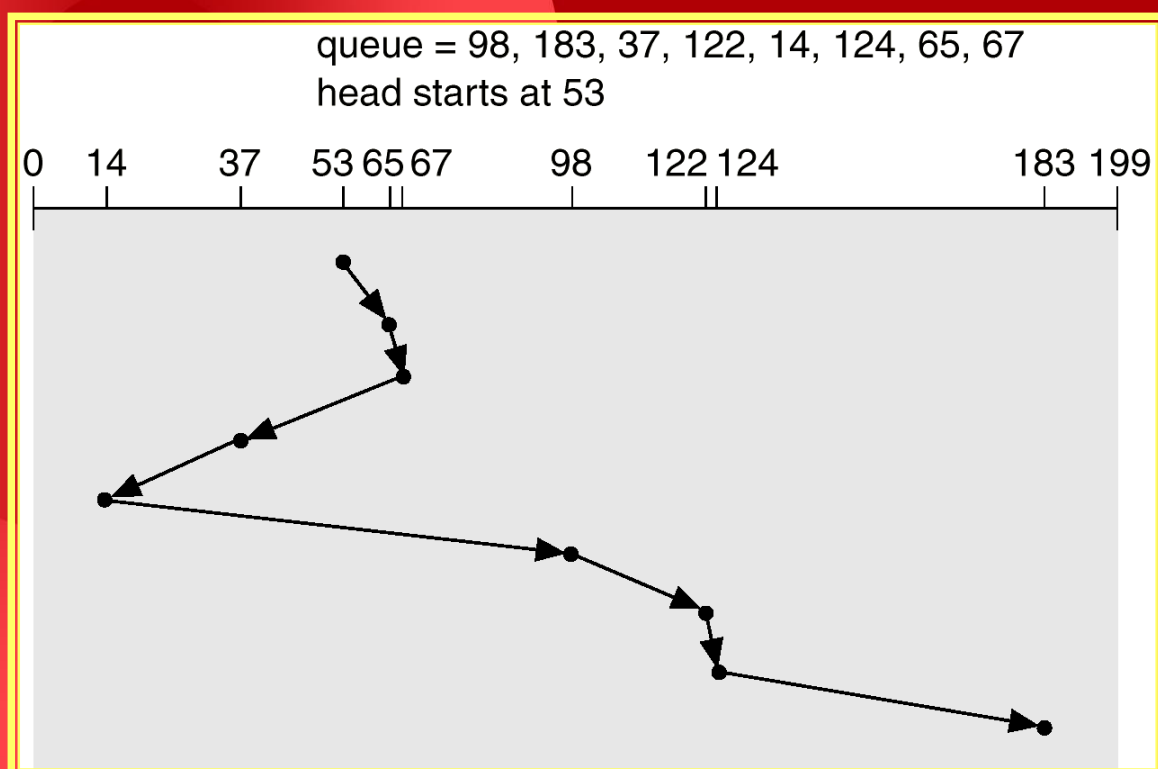
- Selects the request with the minimum seek time from the current head position.
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.
- Illustration shows total head movement of 236 cylinders.

7 August 2021

© Copyright Virtual University of Pakistan



SSTF



7 August 2021

© Copyright Virtual University of Pakistan



SCAN

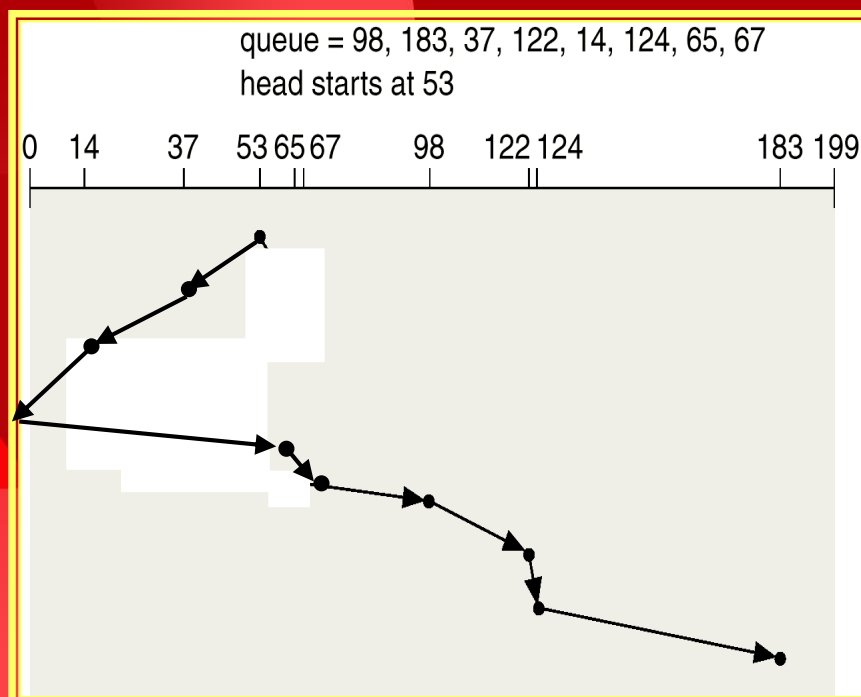
- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues. Also known as the **elevator algorithm**
- Total head movement is 208 cylinders

7 August 2021

© Copyright Virtual University of
Pakistan



SCAN



7 August 2021

© Copyright Virtual University of
Pakistan



LOOK

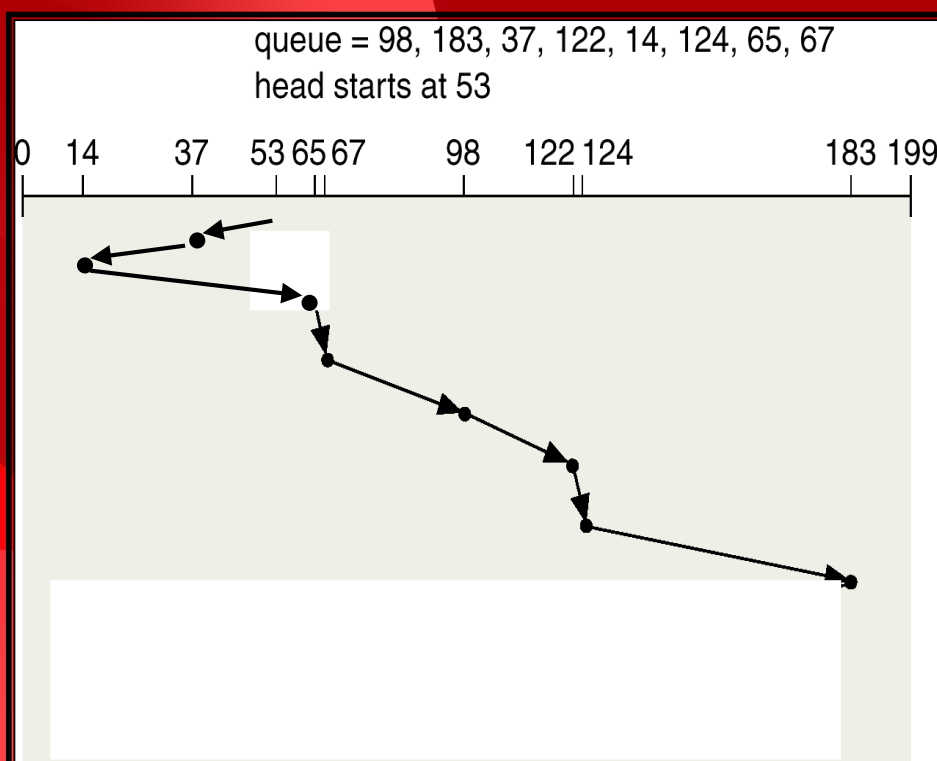
- Version of SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, serving requests while going in the other direction.
- That is, it looks for a request before continuing to move in a given direction.

7 August 2021

© Copyright Virtual University of
Pakistan



LOOK



7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Lecture

- Space Allocation Techniques
- Free Space Management
- Disk Structure and Scheduling
 - Access time and disk bandwidth
 - Disk scheduling algorithms (FCFS, SSTF, Scan, C-Scan, Look, C-Look)

7 August 2021

© Copyright Virtual University of
Pakistan



Recap of Course

- OS services and structures
- Processes and threads
- Process scheduling
- Concurrent processes
- Deadlocks in computer systems
- Memory management
- Virtual memory
- File system interface
- File system implementation
- Secondary storage management

7 August 2021

© Copyright Virtual University of
Pakistan



Future Directions

- Advanced Operating Systems
- System Programming
- Linux Kernel Programming

7 August 2021

© Copyright Virtual University of
Pakistan



Operating Systems

Lecture 45

Syed Mansoor Sarwar

7 August 2021

© Copyright Virtual University of
Pakistan



www.vuCtn.com

Good Luck

Regards : Zarva Chaudhary

Admin:

* Zarva Chaudhary *

* Chaudhary Moazzam *

* Laiba Maki *