CS303 : Operating Systems

Dr. Narendran Rajagopalan
Assistant Professor, Dept. of CSE, NIT Puducherry

July 30, 2019

Significance of the Course

- No Computer Science Course can exclude Operating Systems!!
- All Users using Phones, Laptops, computers invariably use it.
- Any applications designed, depends on the Operating System services!!
- This course distinguishes between any programmer and a CS graduate Programmer.

Text Books and References

Text Book

Operating Systems Concepts, 8th Edition, Silberschatz, Galvin and Gagne.

Reference Book

Operating Systems, 5th Edition, William Stallings.

Reference Book

Modern Operating Systems, Tanenbaum.

Reference for Laboratory Practice

Unix Shell Programming, Sumitaba Das.

- File System
- Secondary-Storage Structure
- I/O Systems

Part 1

INTRODUCTION TO OPERATING SYSTEMS

What is the use of an Operating System

- You have been using it without your knowledge.
- Avoids reinventing the Wheel!
- Provides User convenience!
- Efficient Resource utilization!
- Abstraction and Virtualization (Recently!)

Roles of an Operating System

- Acts like a Government-Provides environment for useful work.
- Resource Manager/Allocator - CPU time, memory space, file storage space, IO devices.
- Security Provider.
- Facilitator to applications.
- Examples include (For PCs) DOS, Windows, Unix, Linux, MAC OS, etc.,
- Examples for Mobile Phones Android, Symbion, Firefox OS, IOS, etc.,
- OS for Embedded Systems are designed to run without user inferference.

Operating System Definition

- No completely adequate definition.
- OS acts as an intermediary between the Computer Hardware and the User.
- The common functions of controlling and allocating resources are brought together as OS.
- A program that runs at all times is the OS.

Goals of Operating System

- Purpose of OS is convenience and efficiency.
- OS is large and complex, each piece performing well delineated tasks.
- Mainframe OS emphasized on optimized utilization of hardware.
- PC OS emphasized on user convenience.
- Mobile OS emphasize on energy efficiency.

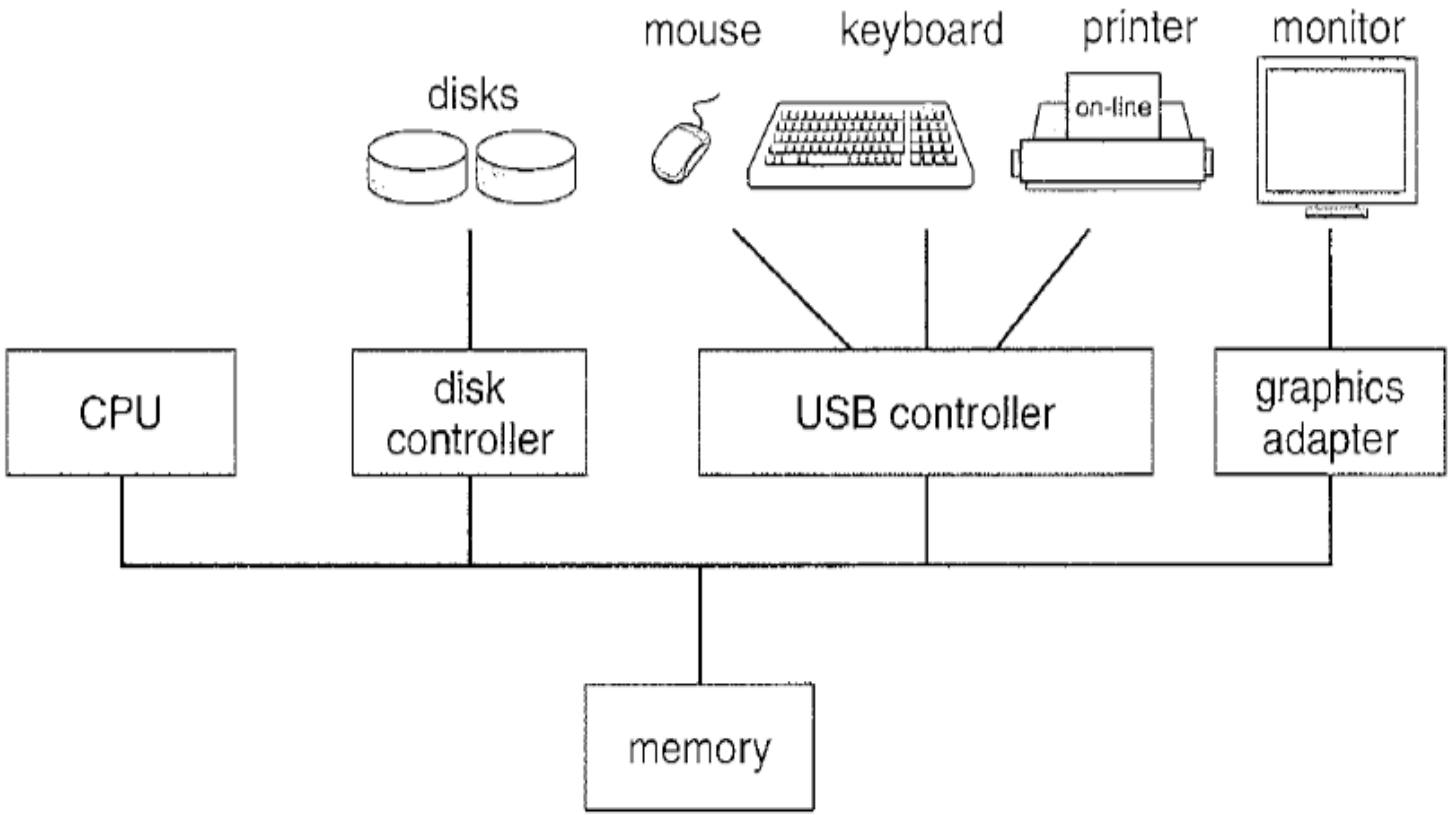
Components of a Computer System

- Hardware - Central Processing Unit, Memory and I/O devices.
- Operating System
- Application Programs - Word Processors, Spread Sheets, Compilers, Web Browsers, ...
- Users

Storage Definitions and Notations

- Bit - A zero or a One.
- Byte - Eight Bits.
- Word - Number of bits an Instruction operates on, X bit Architecture.
- Kilobyte - 1024 bytes, Megabyte - 1024^2, Gigabyte - 1024^3, Terabyte - 1024^4.

Modern Computer System



Modern Computer System

- Bootstrap Program
  - Initial Program stored in ROM or EEPROM.
  - Also known as Firmware.
  - Initializes all aspects of the System.
  - Locates and loads OS kernel into memory.
- OS starts executing the first Process called init.

Interrupts

- Interrupt - signals occurence of an event.
- Hardware Interrupt - sends a signal to CPU via system bus.
- Software Interrupt - through System call(monitor call).
- During interrupt, CPU stops its current operation and starts executing the Interrupt Service Routine.
- Interrupt Vector - Array of addresses indexed by a unique device number.

Symmetric Multiprocessing System

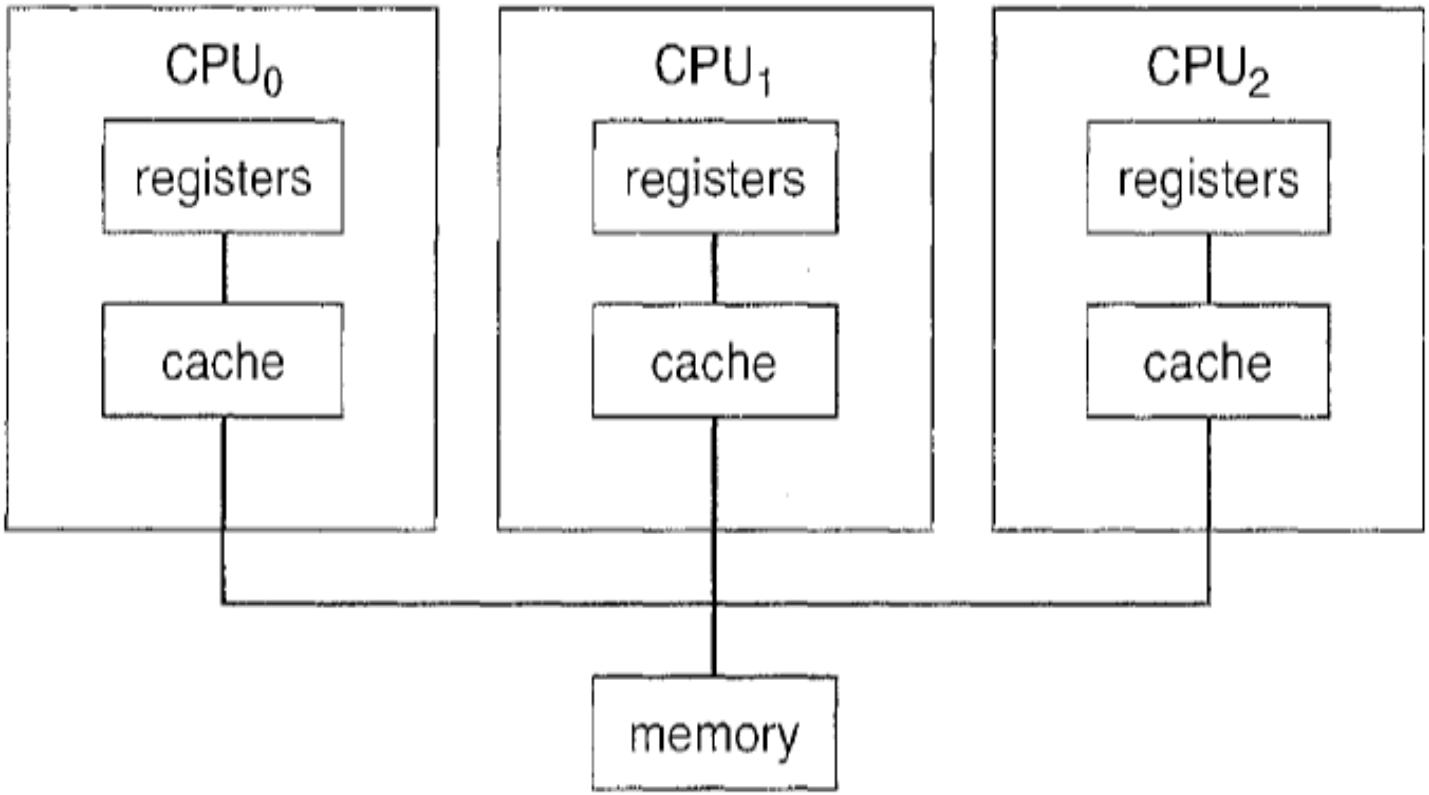
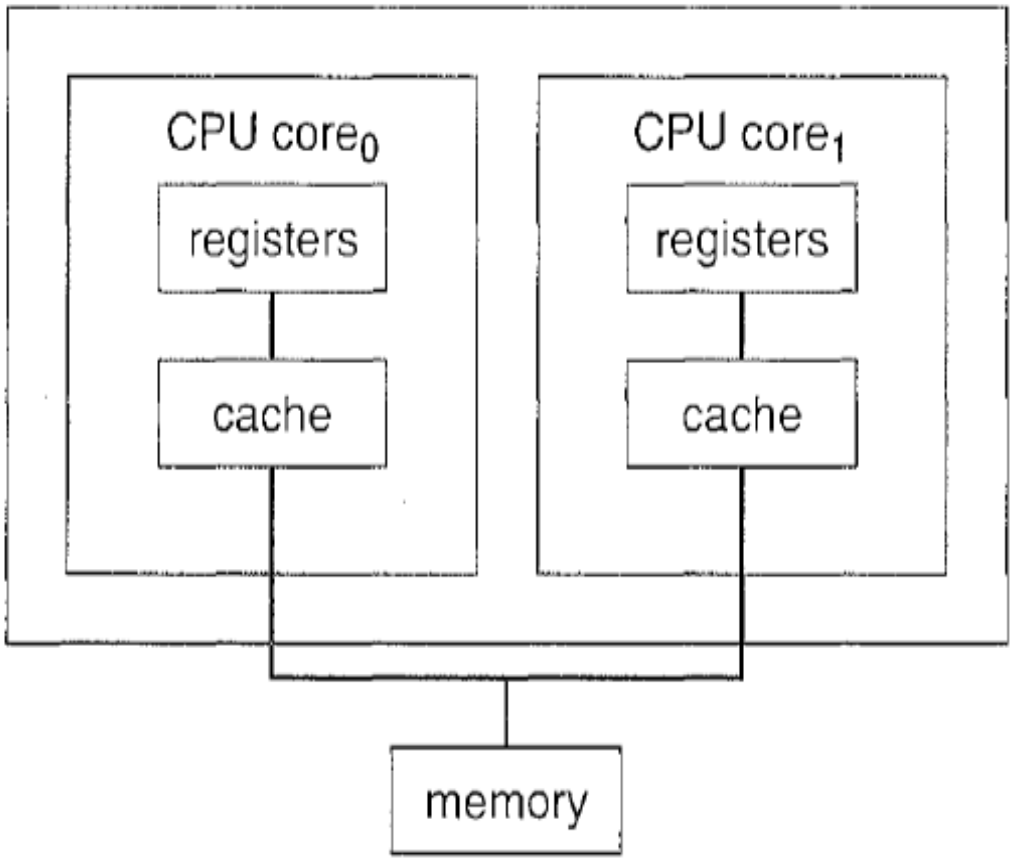


Figure 1.6 Symmetric multiprocessing architecture

Symmetric Multiprocessing System

- A multiprocessor System allows processes and resources such as memory to be shared dynamically among various processors.
- Difference between symmetric and asymmetric can be due to hardware or software(differences).
- UMA - Uniform Memory Access - Access to any RAM from any CPU takes the same amount of time.
- NUMA - Some parts of memory may take longer to access than other parts.
- OS minimizes the NUMA penalty through resource management.

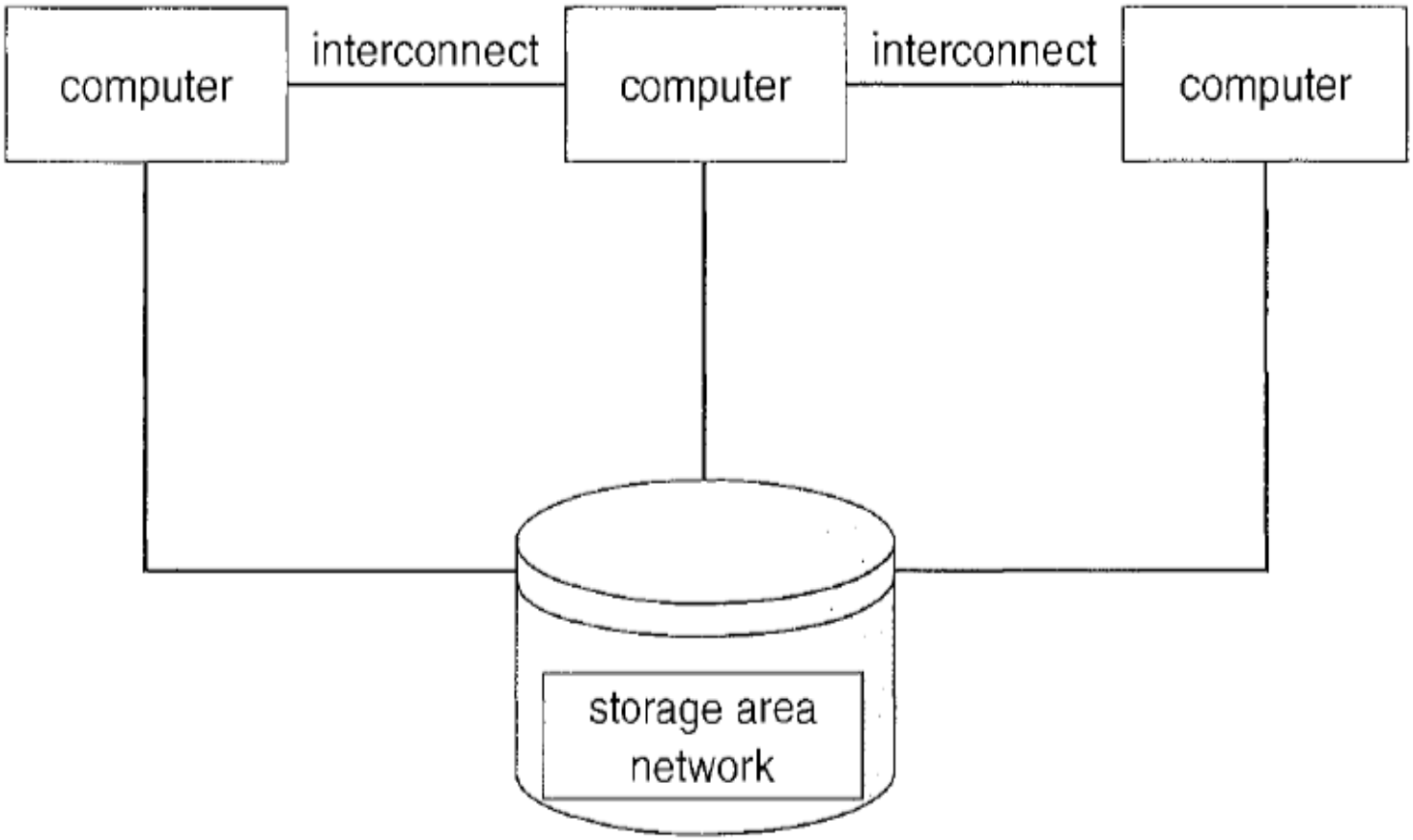
Dual Core Design System



Dual Core Design System

- Multiprocessor Chips - Multiple computing cores on a single chip.
- More efficient due to on-chip communication.
- One chip with multiple cores use less power than multiple single-core chips.
- Each core has its own register set and own local cache.
- Multicore systems are well suited for server systems.

Clustered Systems Design



Blade Servers and Clustered Systems

- Blade Servers - Multiple Processor boards, i/o boards and networking boards are placed in the same chassis.
- Each blade processor boards, boot independently with its own OS.
- Clustered Systems - Composed of two or more individual systems or nodes sharing storage through LAN.
- Clustering is used to provide high availability service(with redundancy/fault tolerant).

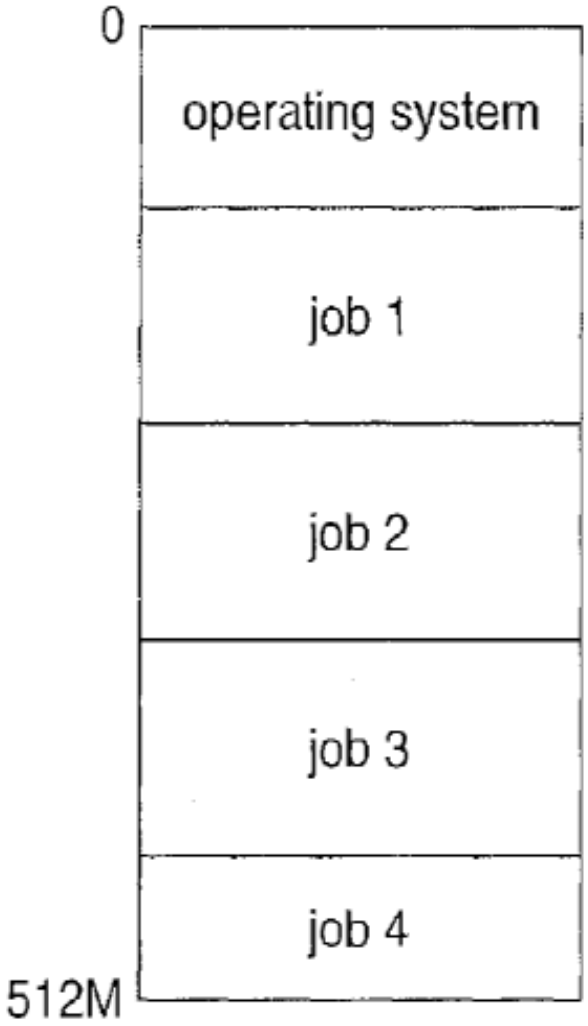
Blade Servers and Clustered Systems

- Asymmetric Clustering - One machine is in hot-standby mode(monitoring) while the other is running the applications
- Symmetric Clustering - Two or more hosts are running applications and are monitoring each other.
- Clusters can provide High-performance computing environment with higher computational power.
- Parallelization - Dividing a program into separate components that run in parallel on individual computers.
- Distributed Lock Manager - Access control and locking funtion to provide shared access to data in a clustered system.

Operating System Structure

- Multiprogramming - increases CPU utilization by organizing jobs so that the CPU always busy.
- OS keeps several jobs in memory simultaneously.
- The jobs are initially kept on disk in job pool.
- In multiprogramming environment various resources are utilized effciently.
- Time-sharing (multi-tasking) is a logical extension of multiprogramming.
- Time-sharing - CPU executes multiple jobs by switching among them.
- Response time should be short (less than 1 sec).
- A time shared OS uses CPU scheduling and multiprogramming.

Multiprogramming System Design



Operating System Structure

- Job Scheduling - The decision of choosing a job from job pool to be loaded into main memory.
- Memory Management - having several programs in memory requires memory management.
- CPU Scheduling - Deciding which process to run at any point in time.
- Swapping - to accomplish reasonable resonse time, processes are swapped in and out of main memory to the disk.
- Virtual Memory - Allows users to run programs larger than actual physical memory.
- Disk Management - File system management.
- Modern OS are interrupt driven.

Trap, Interrupts and Security

- Trap - (or exception) is a software generated interrupt caused either by an error or by a specific request from user program.
- Interrupt - Each type of interrupt determine what action to be taken.
- Protection - Dual-Mode Operation
- User Mode and Kernel Mode differentiated by mode bit.
- Privileged Instructions can be executed only in System mode.
- Eg. : I/O Control, Timer management and interrupt management.
- A System Call is treated by hardware as a software interrupt.
- Control passes to ISR and the mode bit is set to kernel mode.
- Timer - is used to interrupt a computer after a specified period.

Dual Mode of Operation Design

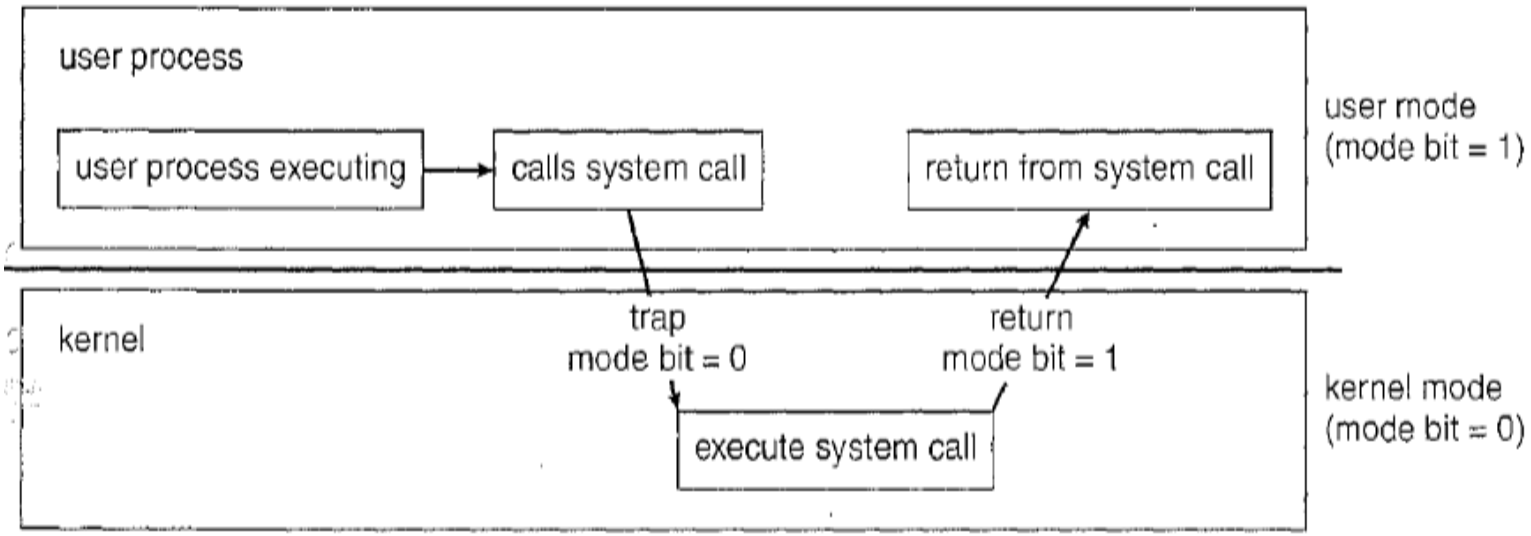


Figure 1.10 Transition from user to kernel mode.

Operating System Responsibilities

- Process Management.
- Memory Management.
- Storage Management.
- Mass Storage Management.
- Cache Management.
- I/O Management.

Operating System Responsibilities for Process Management

- Scheduling Processes and Threads on the CPU.
- Creating and deleting user and system processes.
- Suspending and resuming processes.
- Providing mechanisms for process synchronization.
- Providing mechanisms for process communication.

Operating System Responsibilities for Memory Management

- Allocating and deallocating memory space as needed.
- keeping track of free and memory-in-use and by whom.
- Deciding which process and data to move into and out of memory.

Operating System Responsibilities for Storage Management

- OS provides uniform logical view of information storage.
- A file is a collection of related information.
- Creating and deleting files.
- Creating and deleting directories to organize files.
- Primitives for manipulating files and directories.
- Mapping file onto secondary storage.
- Backing up files on stable storage media.

Operating System Responsibilities for Mass Storage Management

- OS services for disk management.
- Free space management.
- Storage allocation.
- Disk scheduling.
- Secondary Storage devices - Hard Drives.
- Tertiary Storage devices - Magnetic tape drives, CD, DVD.

Operating System Responsibilities for Caching

- The Cache - A faster storage system.
- Cache management is an important design issue.
- Cache Coherency - In a multiple processor environment, each processor may have its own cache, assurance that an update in one cache will reflect in all other caches.

Operating System Responsibilities for I/O Systems

- OS hides the peculiarity of specific hardware devices.
- I/O Subsystem consists of
- Memory management component - includes buffering, caching and spooling.
- A general device driver interface.
- Drivers for specific hardware devices.

Operating System Responsibilities for Security

- User IDs.
- Group IDs.
- Effective User IDs - privilege escalation.
- Distributed Systems - A collection of physically separate heterogeneous computer systems networked.
- Networked OS - includes a communication scheme that allows different processes on different computers to exchange messages.
- Real time Embedded OS - Embedded Systems run on them.
- Multimedia Systems
- Handheld Systems - Less memory, slow processor, small display screens.

Operating Systems History

- Batched OS - processed jobs in bulk with predetermined input.
- Interactive Systems, time sharing systems.
- Client Server Computing - Compute Servers and file servers.
- Peer-to-peer Computing - clients and servers are indistinguishable.
- Peer-to-peer computing avoids bottleneck unlike client server model.
- Discovery protocol - used to discover services provided by other peers.
- Peer-to-peer computing - Napster, gnutella, bittorent,

Open Source Operating Systems

- OS available in source code format.
- Linux, BSD Unix - Examples for Open Source OS.
- Richard Stallman - started GNU Project in 1983.
- Linus Torvalds - released Linux kernel to the open source community.
- Linux Flavours - RedHat, Suse, Debian, Fedora, Unbuntu, Kali.
-
- Peer-to-peer computing - Napster, gnutella, bittorent,

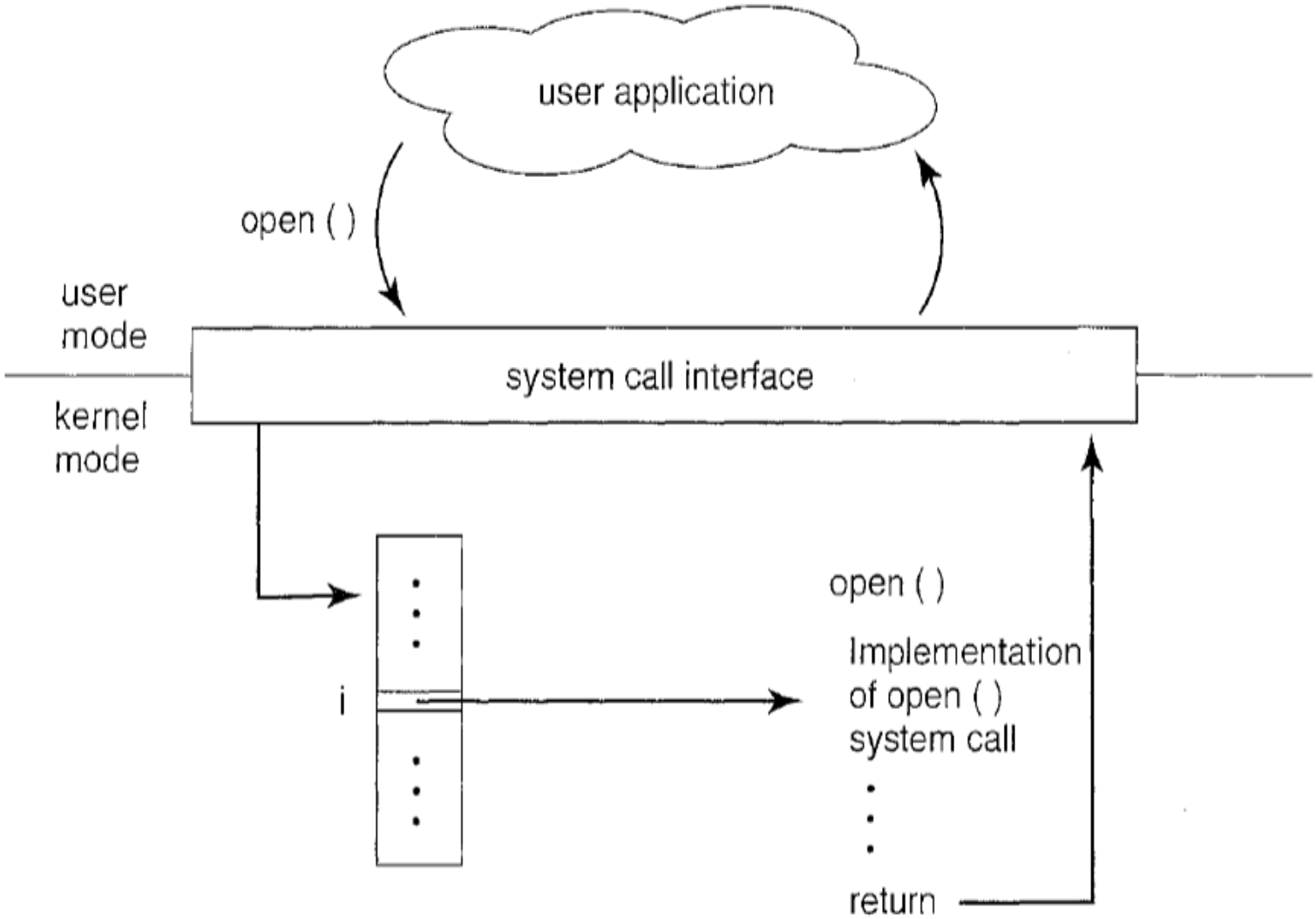
Operating Systems Services

- User Interface - Command Line Interface, GUI, Shells.
- Program Execution - Load program into memory and execute.
- I/O Operation - OS provides the means to do I/O operation.
- File System Manipulation - access permission management.
- Communications - via shared memory or message passing.
- Error Detection - OS should take appropriate steps for various types of errors.
- Resource Allocation, Accounting, Protection and Security.

System Calls

- **System calls** provide an interface to the services made available by OS.
- Generally available as routines written in C or C++.
- Low level tasks may be written using assembly language.
- Abstraction and Portability through **Application Programming Interface**(API).
- System Calls - Types - Process Control, File Manipulation, device manipulation, Communication, protection and Information maintenance.
- Parameter passing to OS - Using **registers**.
- Parameter passing to OS - Using **memory block** and passing the block address in a register.(Linux and Solaris)
- Parameter passing to OS - Using **System stack**.

API - System Call Design



Communication System Calls

- Two common models for IPC : **message-passing model** and **shared-memory model**.
- Message-passing model : Messages can be exchanged between the processes either directly or indirectly through **mailboxes**.
- Shared Memory Model : Processes use shared memory create and shared memory attach system calls.
- Protection System Calls include : set permission, and get permission.

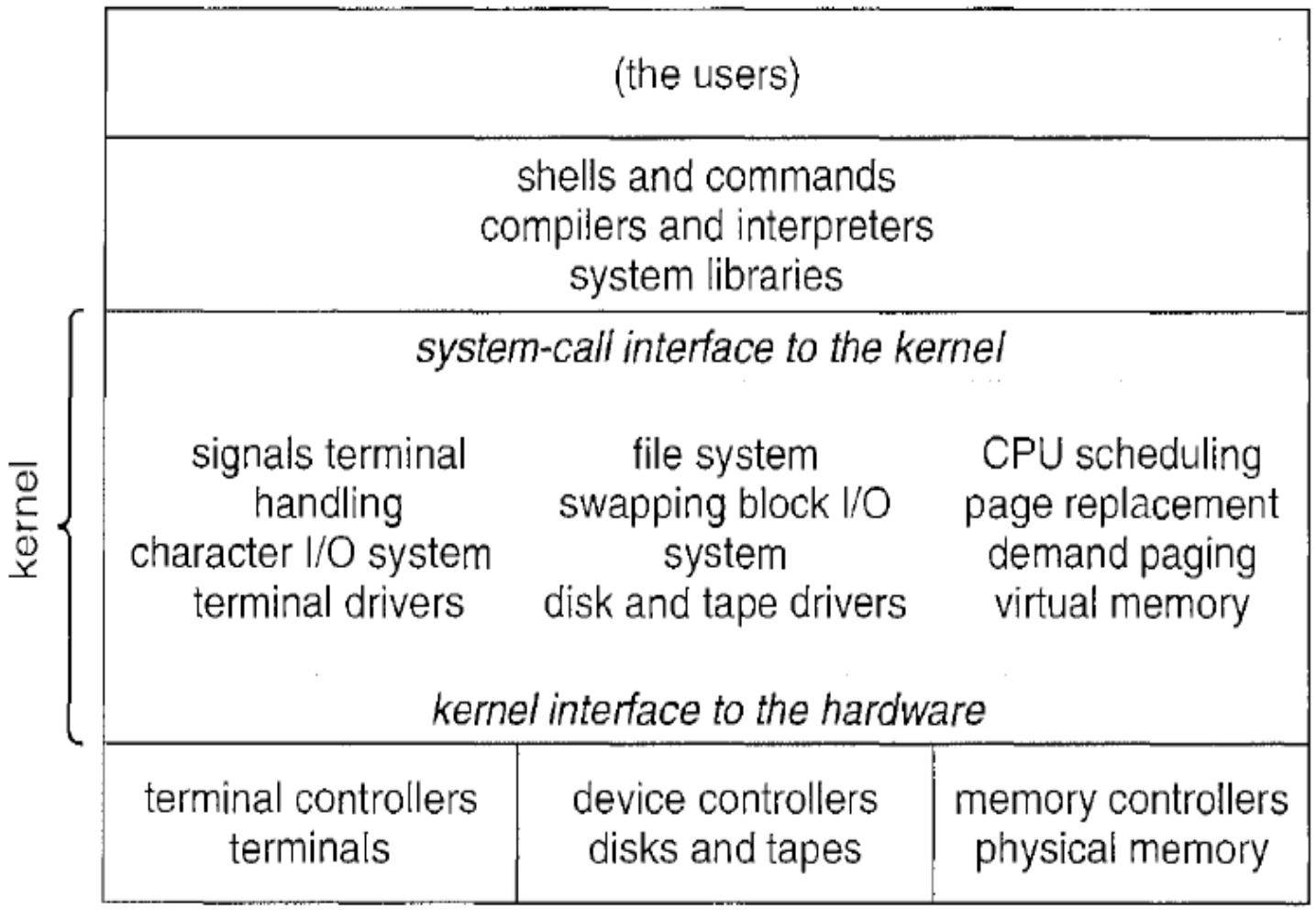
OS Design Goals

- User Goals : Convenience, ease to learn and use, reliable, safe and fast.
- System Goals : Efficiency, transparency.
- Mechanisms and Policies.
- **Mechanisms** : Determine **how** to do something.
- **Policies** : Determine **what** will be done.
- Example : Timer construct is a mechanism for ensuring CPU protection, how long the timer is set is a policy decision.
- Policies are likely to change across places and over time.

OS Implementation

- OS was primarily written in assembly language.
- Device drivers are written in assembly language.
- C or C++ is used to write OS recently.
- Better data structures and algorithms result in major performance improvement in OS.
- **Memory manager** and **CPU scheduler** are the most critical routines.
- In MS-DOS, the interfaces and levels of functionality are not well separated.
- In MS-DOS, No dual mode operation.
- Advantage of Layered Approach : simplicity of construction and debugging.
- Disadvantage of Layered Approach : Less efficient, defining layers,

Unix System Structure



OS Implementation

- **Micro kernel** Approach : structures OS by removing all nonessential from the kernel and implementing them as user or system level programs.
- Main functionality is to provide communication facility between client processes and services running in user space.
- Advantage : Easier portability, higher security.
- MAC OS X(Darwin) is an example of micro kernel approach.
- Modular Kernel Approach : kernel has a set of core components and links in additional services during boot time or run time.
- Dynamically Loadable Modules : commonly used in Linux, Solaris.
- Support for different file system can be added as loadable module.

Solaris Loadable Modules

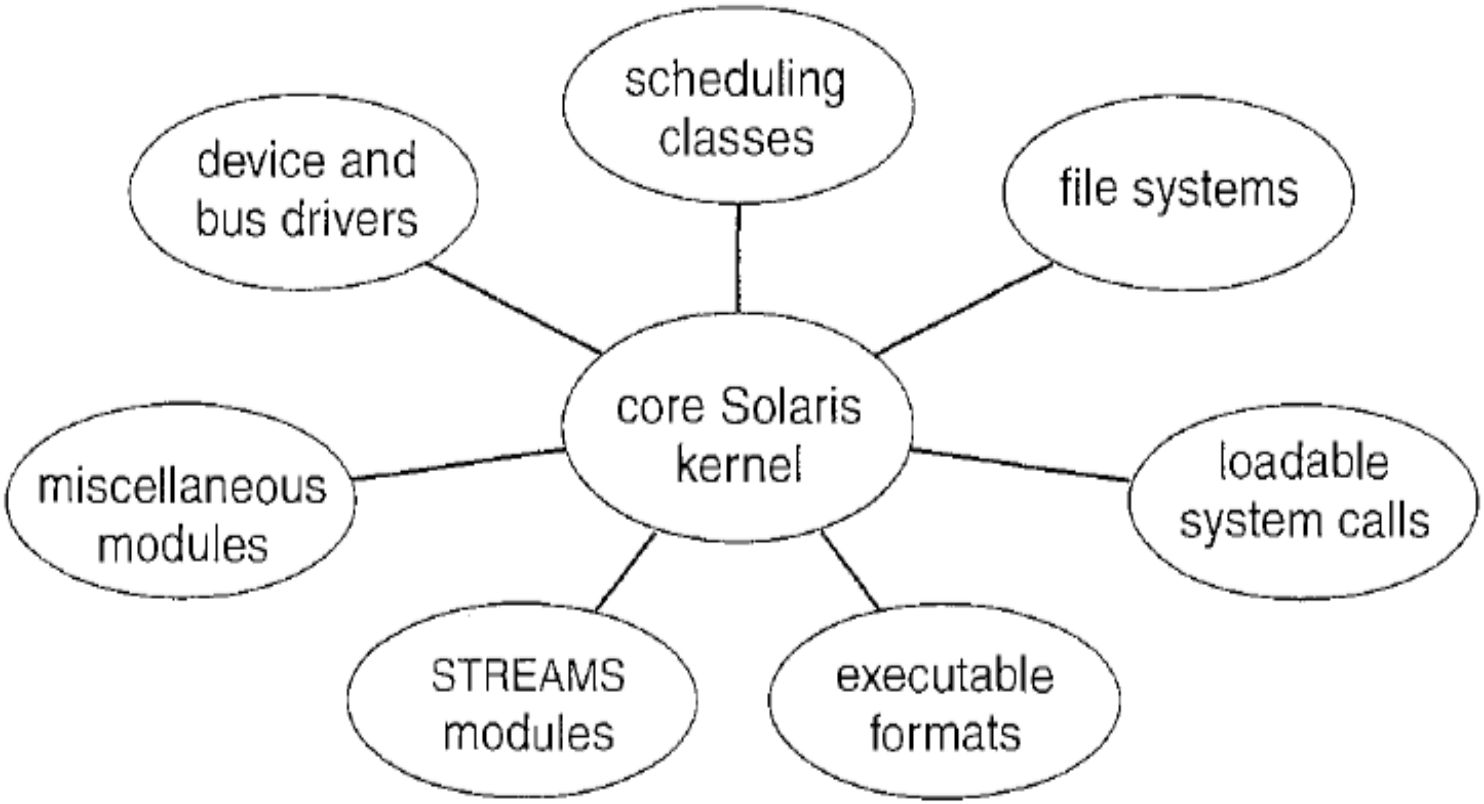
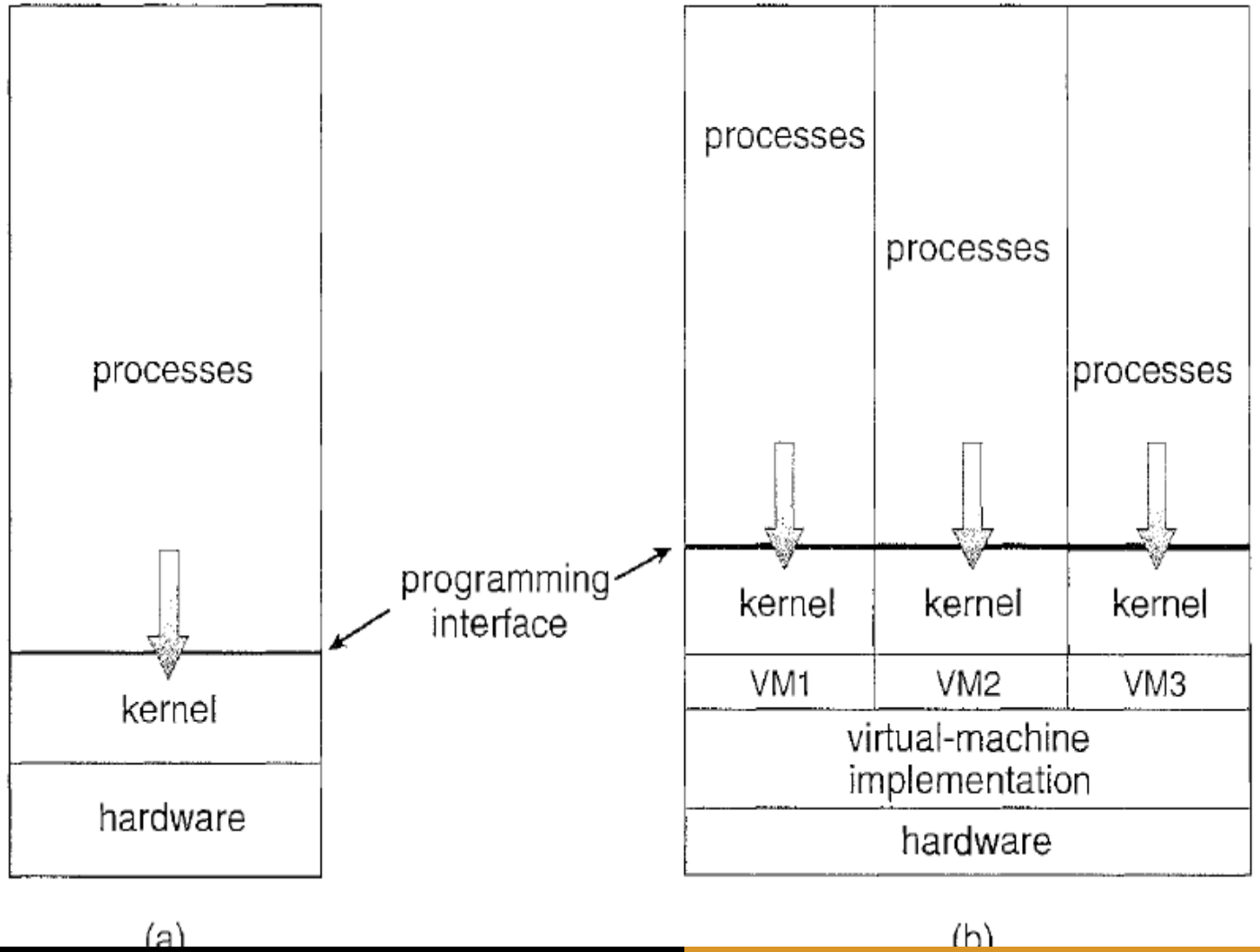


Figure 2.15 Solaris loadable modules

OS Implementation

- Virtual Machine : Layered Approach taken to its logical conclusion.
- Fundamental Idea : abstract the hardware of a single system into several different execution environment.
- Benefits : being able to share a single hardware instance.
- Benefits : Host system is protected from the virtual machines.
- Benefits : Aids in OS research and development.
- Examples : VMWare, Xen
- Simulation : Host system has one system architecture and the guest system was compiled for a different architecture.
- Disadvantage : Slow.
- Java Virtual Machine, Bytecode, garbage collection, just-in-time(JIT) compiler.

Virtual Machine Approach



OS Implementation

- Debugging : performance tuning by removing bottleneck.
- Process failures are written to a log file.
- A kernel failure is called a **crash**.
- **Crash dump** : memory state of a failed process.

Kerninghan's Law :

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

Bootstrap Program

- Runs diagnostics to determine the state of the machine.
- Initializes CPU registers, device controllers and main memory.
- Cellular Phones usually store the entire OS in ROM.
- Boostrap Program : Grub, lilo.

Part 2 A

PROCESS MANAGEMENT

Process

- OS is responsible for creation and deletion of user and system processes.
- OS is responsible for scheduling, synchronization, communication, deadlock handling.
- Process is a program in execution.
- A process includes program code as text section, program counter.
- A process includes process stack, register contents, data section, heap

Process in Memory

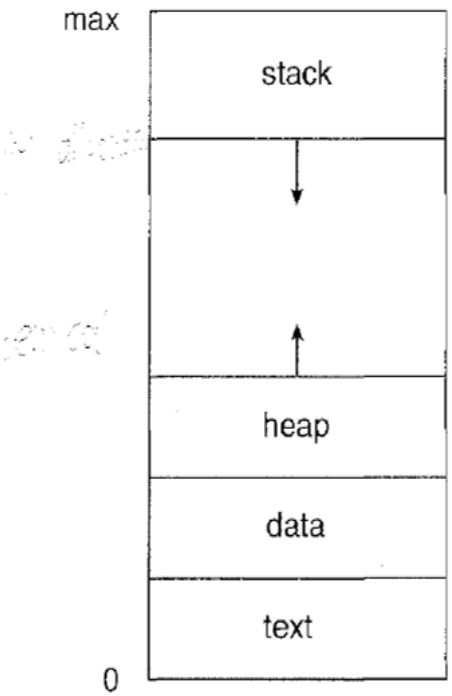


Figure 3.1 Process in memory.

Diagram of Process State

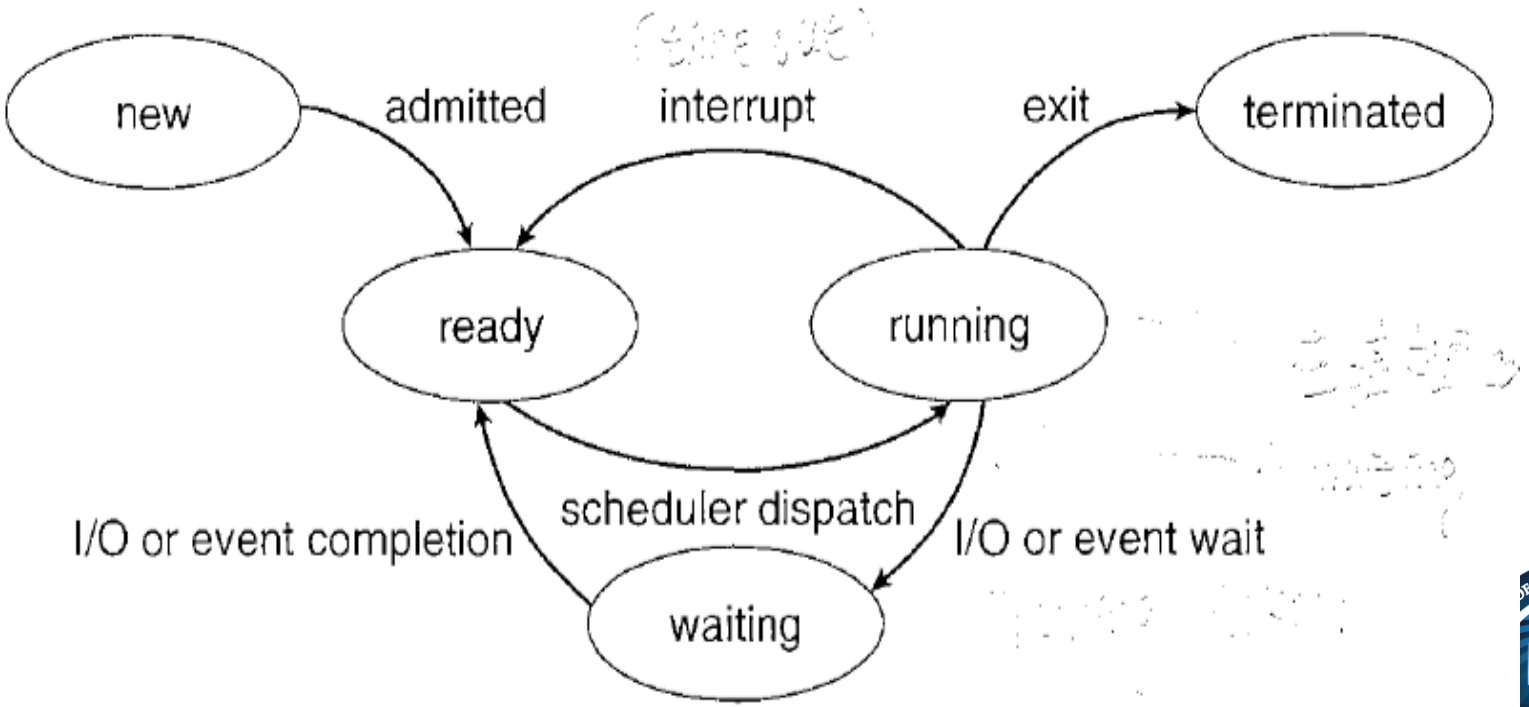


Figure 3.2 Diagram of process state.

Process State

- New : The process is being created.
- Running : Instructions are being executed.
- Waiting : The process is waiting for some event to occur.
- Ready : The process is waiting to be assigned to a processor.
- Terminated : The process has finished execution.

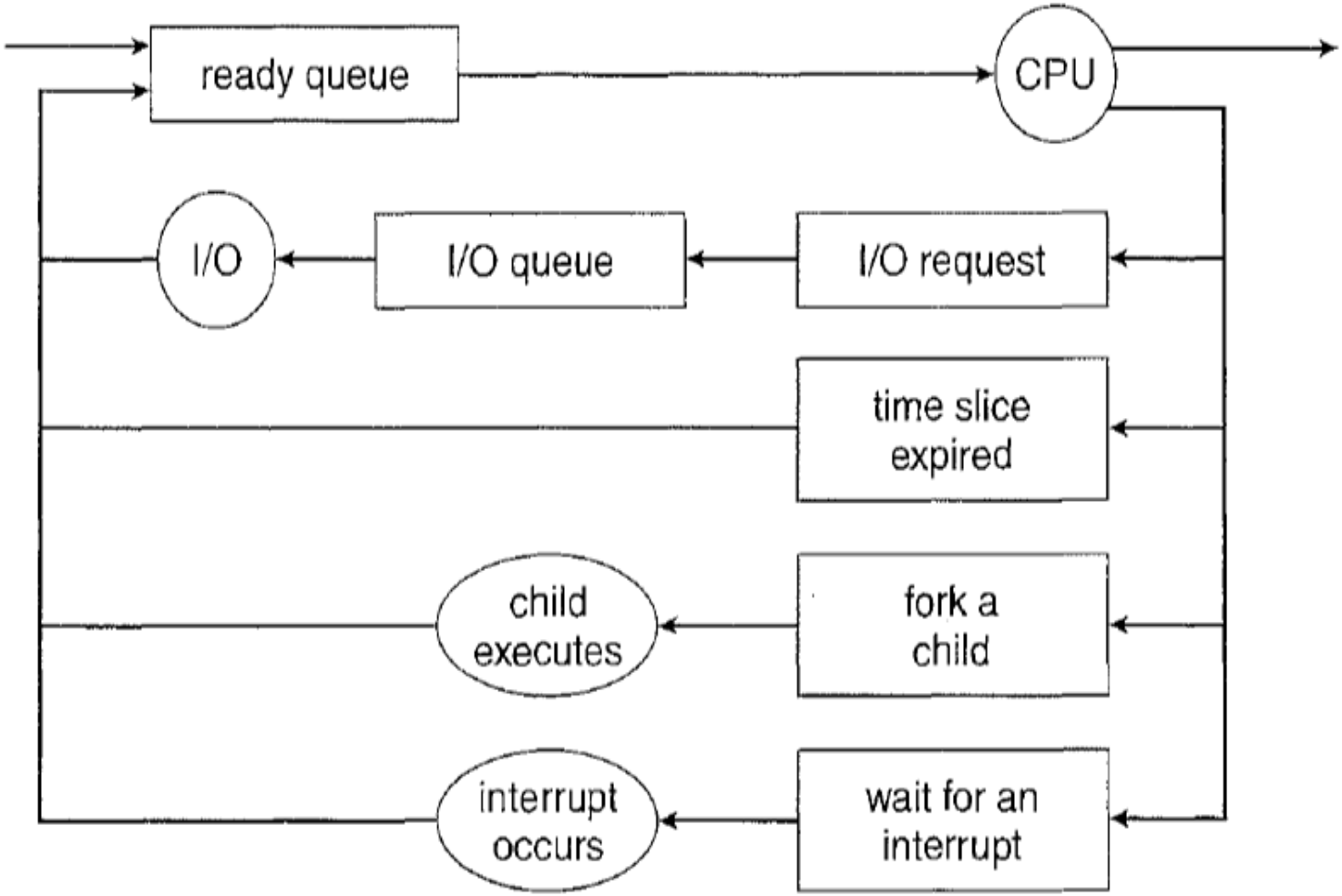
Process Control Block

- Each process is represented by a Process Control Block(PCB).
- PCB is a repository of process information.
- PCB contains : Process State, Program Counter, CPU Registers.
- PCB contains : CPU scheduling information(process priority, pointers to scheduling queues, other scheduling parameters).
- PCB contains : Value of base and limit registers, page table, segment table, etc.,.
- PCB contains : CPU time and real tme used, time limits, process ids.
- PCB contains : list of I/O devices allocated to the process, list of open files

Process Scheduling

- Objective of multiprogramming is to maximize CPU utilization.
- Objective of time sharing is to switch between processes frequently.
- Job Queue : consists of all processes in the system.
- Ready Queue : consists of processes residing in main memory.
- Device Queue : list of processes waiting for a particular I/O device.

Queueing Diagrams for Process Scheduling Rate



Linux Process Scheduling

- PCB in Linux is represented by task_struct C structure.
- In Linux, all active processes are represented using a doubly linked list task_struct.
- In Linux, 'current' is a pointer pointing to the executing process.

Scheduler

- Scheduler is a part of an OS which carries out the selection process.
- Long term scheduler or Job scheduler selects processes from the job pool and loads them into memory.
- Short term scheduler or CPU scheduler selects process for CPU allocation.
- Short term scheduler executes once every 100 ms.
- Long term scheduler executes less frequently.
- Long term scheduler controls the degree for multiprogramming.

Scheduler

- Stable degree of multiproramming if, avg rate of process creation = avg rate of process departure.
- I/O bound process spends more time doing I/O.
- CPU bound process spends more time on CPU.
- Long term scheduler should ensure a good combination of I/O bound and CPU bound processes.
- Time sharing systems may introduce medium term scheduler.
- Medium term scheduler reduces degree of multiprogramming by removing process from memory.
- Later the process is reintroduced to continue execution (Called as Swapping!).

Medium Term Scheduler

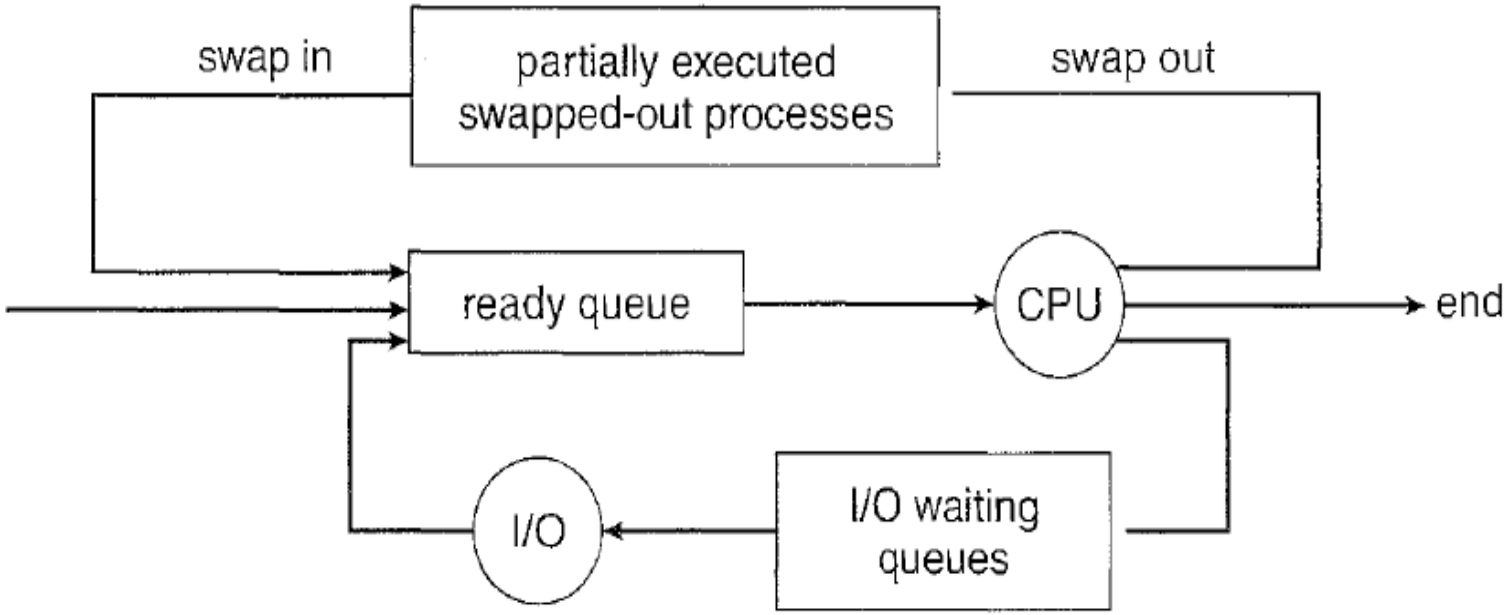


Figure 3.8 Addition of medium-term scheduling to the queueing diagram.

Context Switch

- When an interrupt occurs, the system needs to save the context of the running process.
- The context is represented in the PCB.
- Context Switch : saving the context of the current process and loading the saved context of a new process.
- Context switch time is overhead and should be reduced.
- Process Creation : A parent process creates a child process.
- Processes are identified by unique process identifier.
- Command : ps -el lists complete information for all processes in the system.

Process Creation

- A child process may be allotted resources by the OS or by the parent process.
- the parent's resources being partitioned among childen prevent resource exploitation.
- Parent and children may continue execution concurrently.
- Parent may wait for the children to complete.
- The address space of the child process is a duplicate of the parent process.
- fork() creates a child which returns 0 and at the parent return the pid of the child.
- exec() system call replaces the memory space of a program with a new program.

Process Creation

- A child process may be allotted resources by the OS or by the parent process.
- the parent's resources being partitioned among childen prevent resource exploitation.
- Parent and children may continue execution concurrently.
- Parent may wait for the children to complete.
- The address space of the child process is a duplicate of the parent process.
- fork() creates a child which returns 0 and at the parent return the pid of the child.
- exec() system call replaces the memory space of a program with a new program.
- wait() system call is used by the parent to wait for the child process to complete.

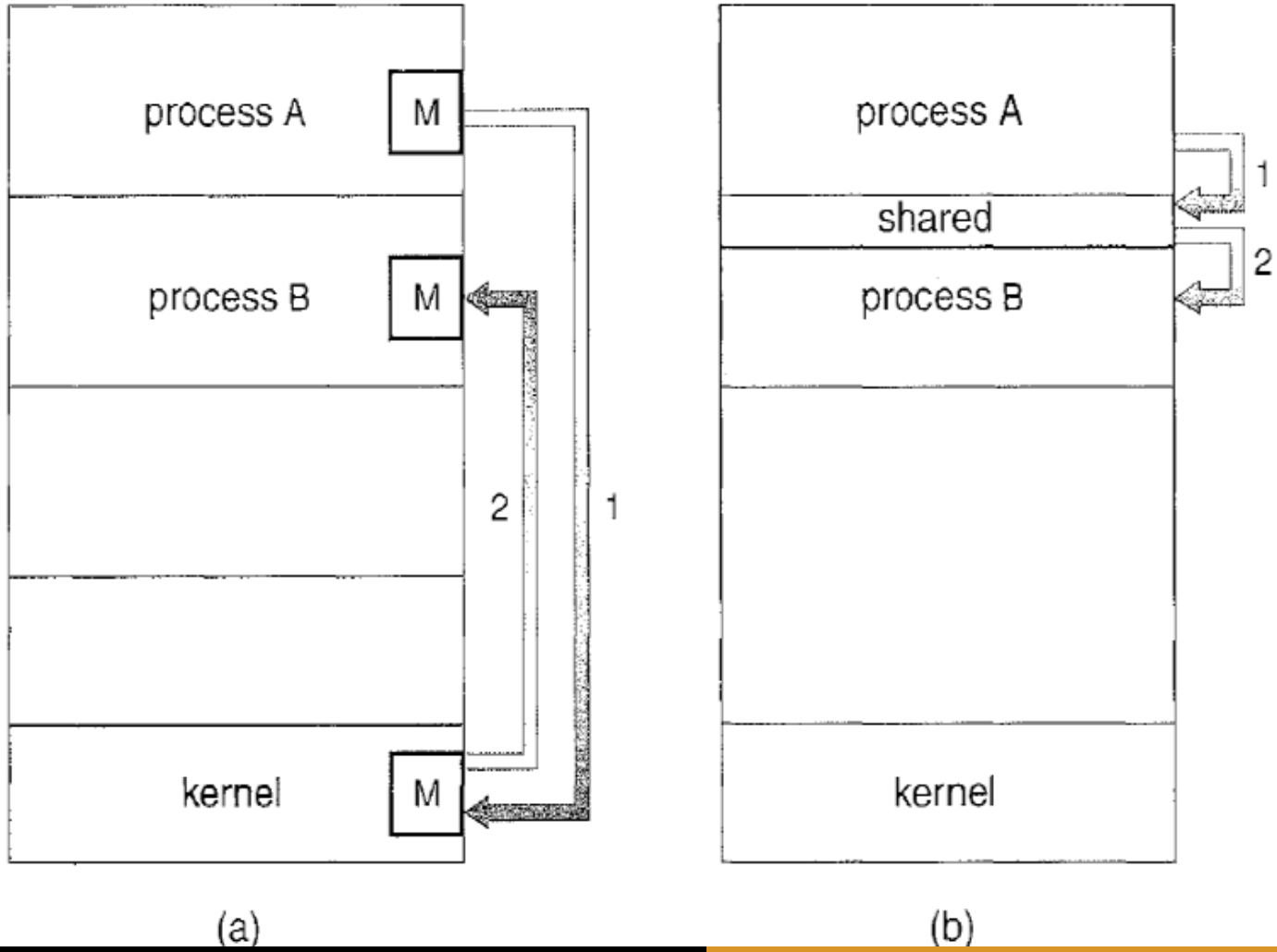
Process Termination

- A process requests the OS to terminate it with exit() system call.
- The process may return a exit status to the parent process, received through wait().
- Resources(virtual memory, open files, I/O buffers) of the process are deallocated.
- Cascading Termination : If a process terminates, all its children should be terminated.
- OS initiates cascading termination.
- But in Unix, if a parent terminates, the children are assigned init as the new parent.

InterProcess Communication

- Processes executing concurrently may be independent processes or cooperative processes.
- Independent Process : it cannot affect or be affected by other processes.
- Independent Process : does not share any data.
- Cooperating Process : It can affect or be affected by other executing processes.
- Reasons for cooperative processes : Information sharing, computation speedup, modularity and convenience.
- Two fundametal models for IPC : 1. Shared memory 2. Message passing.
- Message passing is useful for exchanging smaller amounts of data.
- Message passing is useful for inter-computer communication.

IPC Models


process A
M
process B
M
2
1
kernel
M
(a)
process A
1
shared
2
process B
kernel
(b)

InterProcess Communication : Shared Memory

- Shared memory allows maximum speed and convenience.
- Once shared memory is established, no assistance from the kernel is required.
- Message passing implemented using system calls, requires kernel intervention.
- Shared memory region resides in the address space of the process creating it.
- The processes are responsible for ensuring synchronization.
- Example : Producer - Consumer Problem.
- The container buffer is shared between the producer and consumer processes.
- The bounded buffer assumes a fixed buffer size.

InterProcess Communication : Message Passing Systems

- Allows processes to communicate and synchronize without sharing same address space.
- send(message), receive(message).
- Naming : Direct Communication, send(P,msg), receive(Q,msg).
- Symmetric Addressing : Sender and receiver name each other.
- Asymmetric Addressing : Sender names the receiver, receiver need not have to name the sender.
- send(P,msg), receiver(&id, msg).
- Indirect communication : communication via mailboxes or ports.
- send(msg,A), receive(msg,A) ; where A is mailbox identifier.
- A might be shared by more than two processes.
- mailbox might be owned by a process or by the OS.

InterProcess Communication : Synchronization

- Message passing may be either blocking(synchronous) or non-blocking(asynchronous).
- Blocking send : sender is blocked until receiver or mailbox receives the msg.
- Nonblocking send : sending process resumes after sending msg.
- Blocking receive : receiver is blocked until msg is received.
- Non-blocking receive : receiver retrieves a valid msg or null.
- With blocking send and blocking receive, the producer-consumer problem becomes trivial.
- Buffering : Zero capacity(sender must block untill receiver receives).
- Buffering : Bounded capacity(if the link is full, sender must block until space is available).

InterProcess Communication : Case Study

- POSIX shared memory.
- segment-id = shmget(IPC-PRIVATE, size, s-IRUSR/S-IWUSR).
- shared-memory = (char *) shmat(id, null, 0).
- Id represents the shared memory segment.
- Null allows the OS to choose the location where shared memory will be attached.
- 0 indicates for both reading and writing.
- shmat() returns a pointer to the beginning location in memory where the shared memory has been attached.
- shmdt(shared-memory) detaches the shared memory.
- shmctl() with IPC-RMID flag removes the shared memory from the system.
- Sockets is an example for IPC.

IPC : Remote Procedure Call(RPC)

- Allows a client to invoke a procedure on a remote host.
- Client side stub hides the intricate details of communiaction.
- A separate stub exists for each separate remote procedure.
- Parameter marshalling involves packaging the parameters into a form to be transmitted over a network.
- Marshalling involves representing data in External Data Representation(XDR) format.
- Pipe : acts as a conduit allowing two processes to communicate.
- Ordinary pipes are unidirectional.
- Communicating processes must have parent-child relationship.
- Unix treats pipes as a special type of file.

IPC : Named Pipes

- Named pipes or FIFO allow bidirectional communication.
- Created with a mkfifo() system call.
- Example use : ls — more.

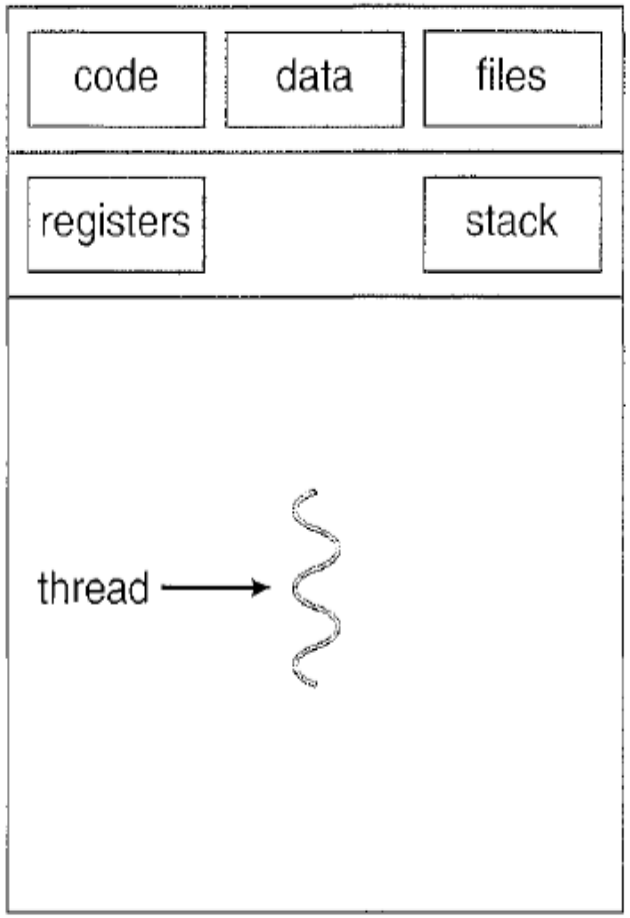
Part 2 B

Multithreaded Programming

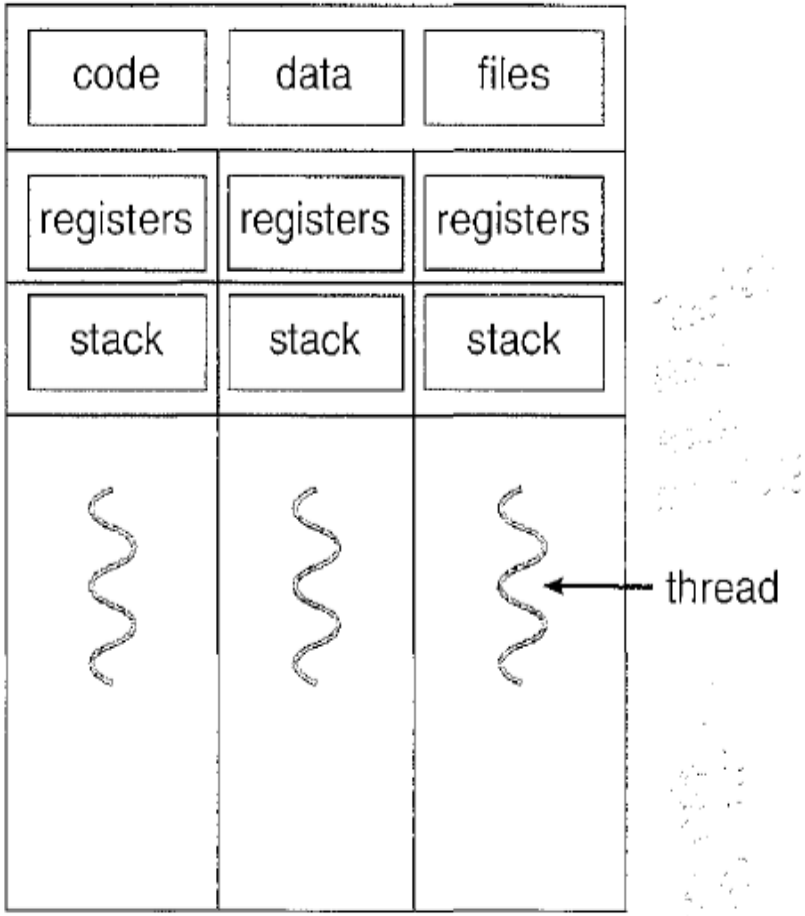
Basics of Multithreaded Programming

- A Thread is a basic unit of CPU utilization.
- Unique to a Thread : Thread Id, A program counter, A Register Set and a Stack.
- Shared with other threads : code section, data section, open files and signals, and other OS resources.
- A process can have multiple threads of control, performing multiple tasks concurrently.
- Example : Word processor may use a thread for GUI display and another thread for keystrokes.
- Example : A web server with many threads each serving a client.
- Benefits of Multithreading : Responsiveness, resource sharing, economy, scalability.
- In solaris, Process creation = 30 * thread creation, context

Representation of a Multithreaded Process



single-threaded process



multithreaded process

Multicore Programming

- Multithreaded Programming efficiently uses the multiple cores for improved concurrency.
- Challenge : Design application programs to take advantage of multicore systems.
- Challenges : Dividing activities : find parts of code that can run in parallel.
- Challenges : Balance : ensure that tasks perform equal work of equal value.
- Challenges : Data splitting : Data accessed and manipulated bt the tasks must be divided to run on separate cores.
- Challenges : Data dependency.
- Challenges : testing and debugging is inherently difficult than signle threaded applications.

Types of Threads

- User Level Threads : Support for threads is provided at user level.
- Kernel Level Threads : Support for threads is provided at the kernel level.
- User threads to kernel thread relationships : Many-to-One Model - if one thread makes a blocking sys call, all threads are blocked.
- One-to-one Model - Each user thread is mapped to a kernel thread.
- Linux and Windows - implements the one-to-one model.
- Many-to-many Model - multiplexes many user level threads to a small or equal number of kernel level threads.
- Thread support : POSIX Thread Library pthreads.
- *** Does fork() duplicate all threads of a process? ***

Concepts of Threads

- Thread cancellation : The task of terminating a thread before completion.
- Asynchronous Cancellation : One thread immediately terminates the target thread.
- Deferred Cancellation : The target thread checks periodically whether it should terminate.
- Asynchronous Cancellation can lead to deadlocks since it may hold some resource.
- In deferred cancellation, the target thread checks a flag to determine it should be cancelled.
- Cancellation Points : points at which threads can be cancelled safely.

Signal Handling

- A signal is used in UNIX systems to notify a process that a particular event has occurred.
- Synchronous Signals : Signals delivered to the process that performed a specific operation(divide by zero).
- Asynchronous Signals : A signal generated by a nevent external to a running process (eg., ¡ctrl-C¿).
- A signal may be handled by to possible ways : a default signal handler, a user-defined signal handler.
- In a multi-threaded environment,
- Deliver the signal to the thread to which the signal applies.
- deliver the signal to every thread in the process.
- Deliver the signal to certain threads in the process.
- Assign a specific thread to receive all signals for the process.

Multithreading Concepts

- Thread Pool : To create a number of threads at process startup and place them into a pool waiting for work to be assigned.
- Thread Pool eliminates thread create overhead.
- LWP(Light Weight Process) is a datastructure between user and kernel threads.
- LWP appears as a virtual processor of which a user thread can be scheduled.
- Kernel threads run on physical processors and when blocked also blocks LWP.
- Linux doensnot distinguish between process and thread.
- It only uses clone-fs, clone-vm,clone-sighand and clone-files flags to specify the sharing of resources, thus making it equivalent to threads.

Exercise 1

Quiz

Which of the following components of program state are shared across threads in a multithreaded process?

a. Register values
b. Heap memory
c. global variables
d. Stack memory

Exercise 2

Analysis and Discussion

Consider a multiprocessor system and a multithreaded program written using the many-to-many threading model. Let the number of user-level threads in the program be more than the number of processors in the system. Discuss the perfomance implications of the following scenarios.

a. The number of kernel threads allocated to the program is less than the number of processors

b. The number of kernel threads allocated to the program is equal to the number of processors

c. The number of kernel threads allocated to the program is greater than the number of processors, but less than the number of user level threads.

Exercise 3

Programming Exercise

Matrix Multiplication Project using multi-threading

Part 2 C

Process Scheduling

Process Scheduling Concepts

- CPU Bursts vs I/O Bursts.
- Short term Scheduler or CPU Scheduler selects processes from the ready queue whenever CPU becomes idle.
- The ready queue can be implemented as a FIFO queue, a priority Queue, a tree or a linked list.
- CPU-scheduling decisions take place under :
- 1. When Process switches from running state to the waiting state(due to I/O request)
- 2. When Process switches from running state to ready state due to interrupt.
- 3. When a process switches from waiting state to ready state.
- 4. When a process terminates.
- If scheduling takes place under 1 and 4 only, then the scheduling scheme is non-preemptive or cooperative, otherwise

Process Scheduling Concepts : Dispatcher

- Dispatcher : A module that gives control of the CPU to the process selected by the short-term scheduler.
- Functions involve : Switching context, switching to user mode and loading Instruction Pointer.
- Dispatcher is invoked during every process switch.
- Dispatch Latency : time taken to stop a process and start another running.
- Metrics for evaluation of Scheduling algorithms : CPU utilization(%),
- throughput(number of processes completed per unit time), Turnaround Time(Time of submission to time of completion),
- Waiting time(sum of periods spent waiting in the ready queue.), Response time : is the time taken to respond.

Scheduling Algorithms : FCFS

- First-come first-served Scheduling(FCFS).
- Average waiting time is quite long.
- Gantt Chart : bar chart representation of a schedule.
- Suffers from Convoy Effect(All processes wait for one big process to get off the CPU).
- FCFS is non-preemptive and and can result in lower CPU and I/O utilization due to Convoy Effect.
- Disastrous for time-sharing systems.

Scheduling Algorithms : SJF

- Shortest Job First Scheduling(SJF).
- CPU is assigned to the process with smallest next CPU burst.
- SJF is proverbally optimal, it gives the minimum average waiting time for a given set of processes.
- Practical Challenge : knowing the next CPU request(burst).
- SJF scheduling is used frequently in long-term scheduling.
- Preemptive SJF acheduling is also called Shortest-Remaining-Time-First scheduling.
- Next CPU burst is predicted as an exponential average of the measured lengths of previous CPU bursts.

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

Exercise 4

Scheduling Exercise

Given processes with arrival time and burst time, calculate the average waiting time, turn around time and response time.

Table : Process Details

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Scheduling Algorithms : Priority Scheduling

- CPU is allocated to the process with highest priority.
- Equal priority processes are scheduled in FCFS order.
- SJF is a special case of Priority Scheduling, where priority is the inverse of next CPU burst.
- Priority is indicated between 0 - 7 or 0 - 4095.
- Priorities can be defined internally or externally.
- Internally defined priority use measurable quantity like time limits, memory requirements, number of open files, ratio of average I/O burst to average CPU burst.
- External Priorities are set by criteria outside the OS, importance of the process, monetory factor, political, etc.,
- Priority scheduling can be preemptive or nonpreemptive
- Disadvantage of Priority Scheduling is starvation or indefinite blocking.

Exercise 5

Scheduling Exercise

Given processes with burst time and priority, calculate the average waiting time, turn around time and response time for Priority Scheduling, it is assumed all processes arrive at time 0.

Table : Process Details

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Scheduling Algorithms : Round-Robin Scheduling

- Designed specially for time-sharing systems.
- Time slice or time quantum is defined as a small unit of time.
- The average waiting time under RR policy is often long.
- The performance of RR algorithm depends heavily on the size of the time quantum.
- If time quantum is large, it is FCFS, and if time quantum is small, it is time-sharing.
- Time quantum should be relatively more than context switch time(overhead[10%]).

Exercise 6

Scheduling Exercise

Given processes with burst time, calculate the average waiting time, turn around time and response time for Round-Robin Scheduling, it is assumed all processes arrive at time 0.

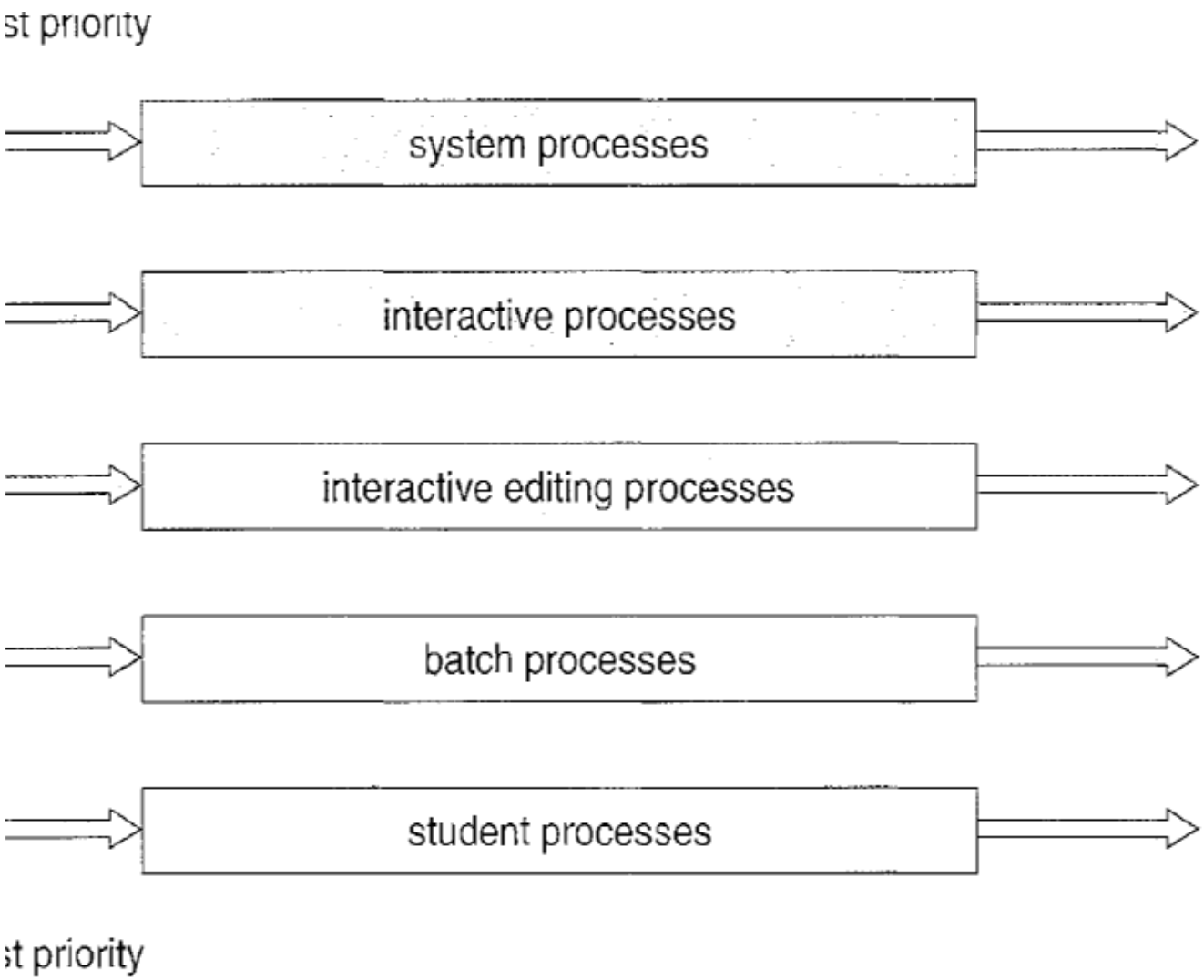
Table : Process Details

Process	Burst Time
P_1	24
P_2	3
P_3	3

Scheduling Algorithms : Multilevel Queue Scheduling

- Processes can be classified into **foreground and background processes** with different response time.
- The ready queue is partitioned into several queues with different priorities.
- Each queue has itso own scheduling algorithm.
- Foreground queue might use RR scheduling.
- Background queue might use FCFS scheduling.
- Scheduling among the queues might follow fixed priority preemptive scheduling.
- **Starvation** is an issue.

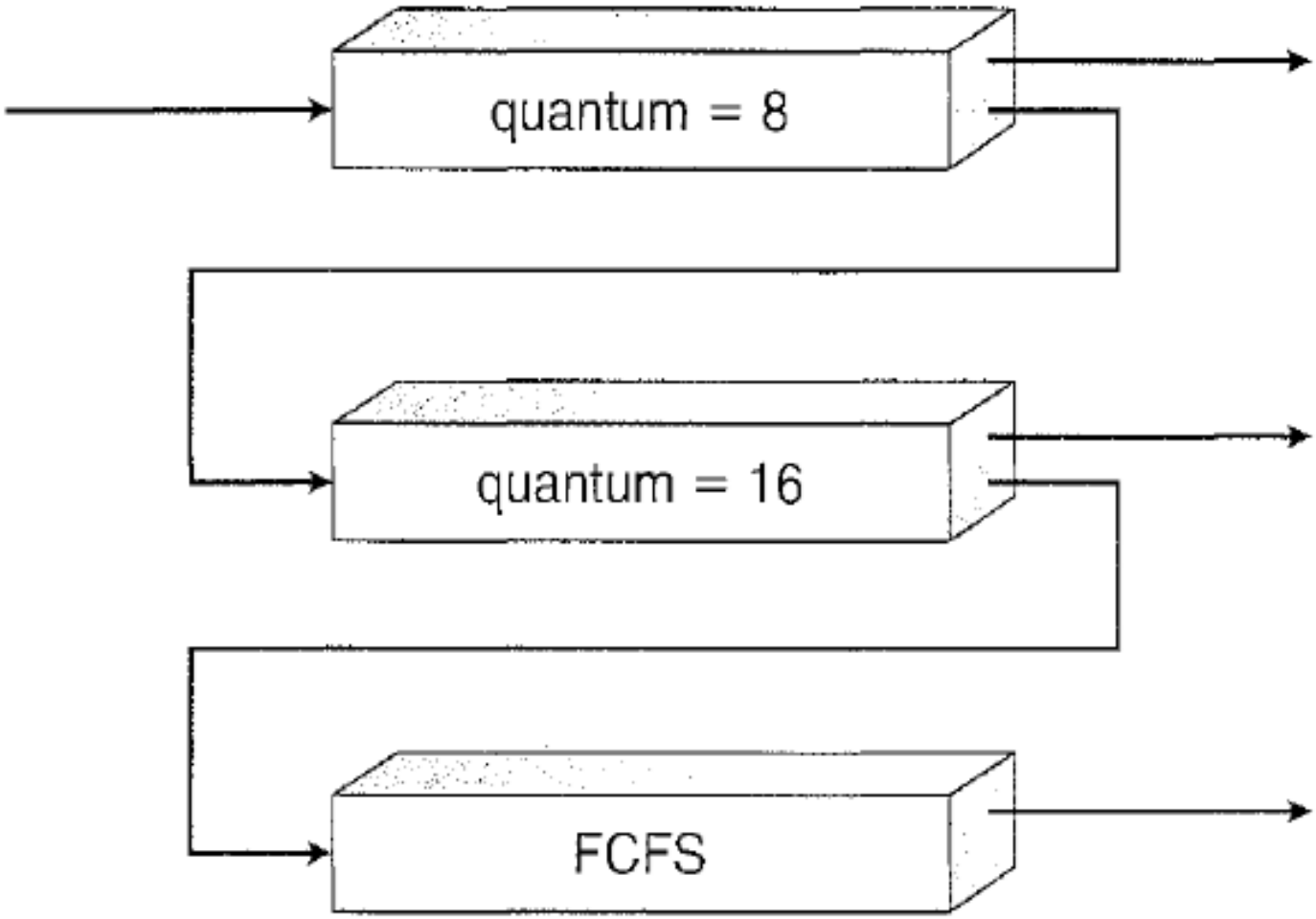
Multi-Level Queue Scheduling



Scheduling Algorithms : Multilevel Feedback Queue Scheduling

- Allows a process to move between Queues.
- If a process uses too much CPU time, it is moved to lower priority queue.
- A process that waits too long in a lower priority queue is moved to higher priority queue.
- This process of aging prevents starvation.
- Parameters : The number of Queues, Scheduling algorithm for each queue.
- Parameters : When to upgrade/degrade a process to a higher/lower priority queue.
- Parameters : Which queue a process will enter when that process needs service.
- Most general CPU scheduling algorithm.

Multi-Level Queue Scheduling



Contention Scope

- User level threads to run on an available LWP, known as process-contention scope(PCS).
- To decide which kernel thread to schedule onto a CPU, the kernel uses system-contention scope(SCS).
- Systems using the one-to-one model uses SCS (Windows XP, Solaris and Linux).
- PCS is done as per priority, the scheduler selects the runnable thread with highest priority.
- User level thread priorities are set by the programmer.

Multiple-Processor Scheduling

- Asymmetric multiprocessing : All scheduling decisions, I/O processing and system activities are handled by a single processor, other processors execute only user code.
- Symmetric multiprocessing : each processor is self scheduling.
- All modern OS support SMP. (Windows XP, Solaris and Linux).
- **Processor Affinity** : If a running process from one processor is migrated to another processor, the cache memory of the old processor should be invalidated and repopulated in the cache of new processor.
- To avoid the cost of invalidating and repopulating caches, most SMP systems try to avoid migration of processes between processor, this is known as processor affinity.
- Linux provides system call to support **hard affinity** allowing a

Multiple-Processor Scheduling

- **Load balancing** attempts to keep the workload evenly distributed across all processors in a SMP system through push and pull migration.
- **Push migration** a specific task periodically checks the load on each processor and evenly distributes the load by moving(pushing) processes from overloaded to idle or less-busy processors.
- **Pull migration** occurs when an idle processor pulls a waiting task from a busy processor.
- Push and pull migration(both) are implemented in parallel on load-balancing systems.
- Linux runs its load balancing algorithm every 200ms(push migration) or whenever the run queue for a processor is empty(pull).

Multiple-Processor Scheduling

- In a multicore processor systems, when a processor accesses memory, it waits for a significant amount of timefor the data to be available, this situation is called **memory stall** due to cache miss.
- To solve memory stall, multithreaded processor cores are implemented with two or more hardware threads assigned to each core.
- Coarse grained multithreading : A thread executes on a processor until a long-latency event(memory stall) occurs. The processor must switch to another thread.
- FIne grained multithreading switches between threads at the boundary of an instruction cycle.
- In a virtualization environment, the virtualized OS is at the mercy of the virtualization software.

Multithreaded multicore System

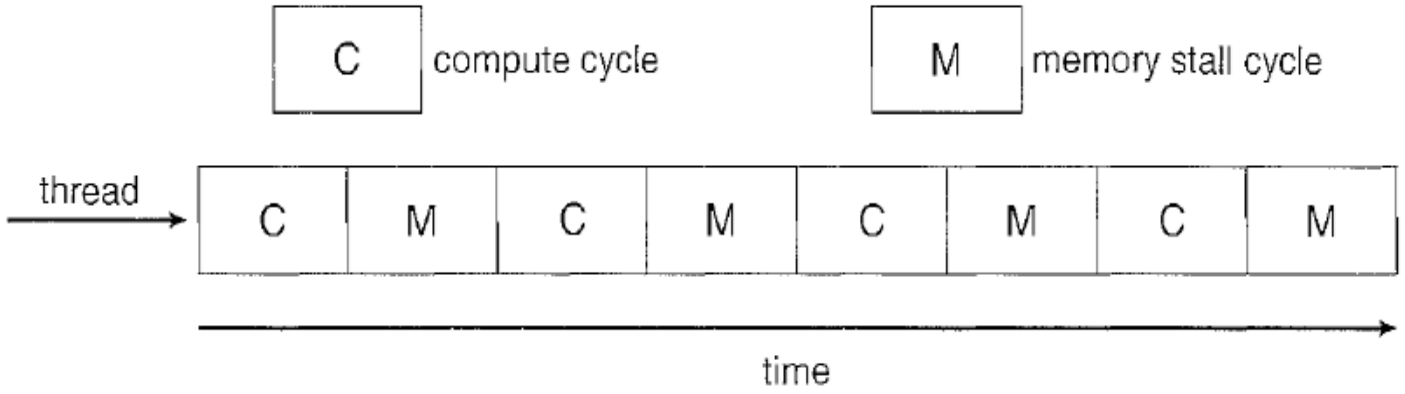
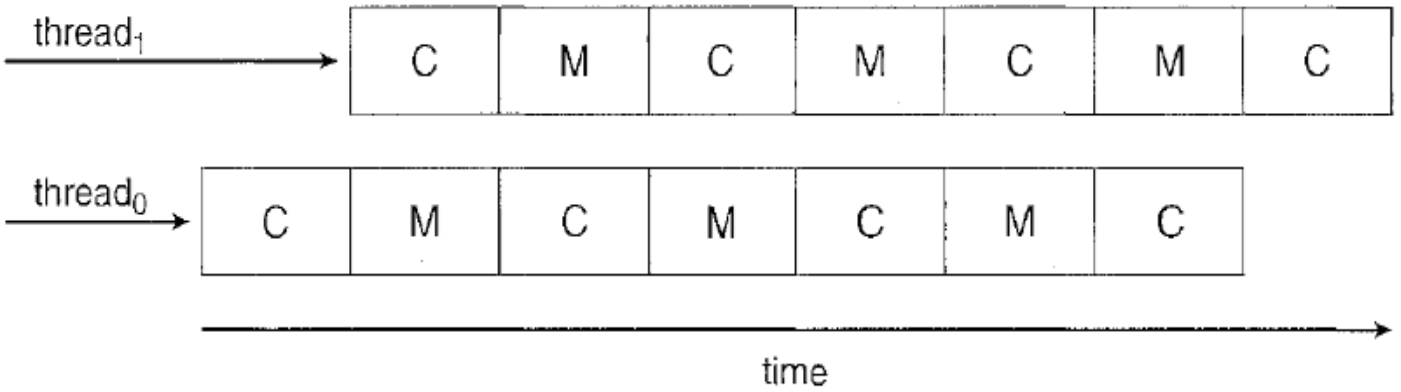


Figure 5.10 Memory stall.



Exercise 7

Scheduling Exercise

Consider the following set of processes, with the length of the CPU burst given in milliseconds :

Table : Process Details

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

Exercise 7

Scheduling Exercise

Consider the following set of processes, with the length of the CPU burst given in milliseconds :

- The processes are assumes to have arrived in the order P_1, P_2, P_3, P_4, P_5 all at time 0.
- Draw Gantt charts for FCFS, SJF, nonpreemptive priority and RR(quantum $= 1$)
- What is the turn around time of each process in each algorithm?
- What is the waiting time for each process in each algorithm?
- Which of the algorithms results in minimum average waiting time?

Part 2 D

Process Synchronization

Process Synchronization

- **Concurrent access** to shared data may result in data inconsistency.
- Bounded Buffer Producer - consumer Problem
- Code for Producer Process :

```
while (true)
{
   produce an item in nextProduced;
   while ( counter == BUFFER_SIZE)
            no op;
   buffer[in] = nextProduced;
   in = (in + 1) % BUFFER_SIZE;
   counter++;
 }
```

Process Synchronization

- Code for Comsumer Process :

```
while (true)
{
   while ( counter == 0)
          no op;
   nextConsumed = buffer[out];
   out = (out + 1) % BUFFER_SIZE;
   counter--;
   consume the item in nextConsumed;
 }
```

Process Synchronization

- The code for Producer and Consumer Processes are correct separately.
- But may not function correctly, when executed concurrently.
- If the shared variable counter is 5.
- Execution of "counter++" and "'counter–" may result in 4, 5 and 6!
- But the correct value should be 5.
- **Race Condition** : Several processes access and manipulate the same data concurrently and the outcome of the execution depends on the order of access.

Process Synchronization

- **Critical Section** : Each process has a segment of code in which the process may be changing common variables.
- When one process is executing in its critical section, no other processes is to be allowed to execute in its critical section.
- **Critical-section Problem** is to design a protocol that the processes can use to cooperate.
- **Entry Section** : Each process must request permission to enter its critical section.
- The critical section may be followed by an exit section.
- The remaining code is the **remainder section**.

General Structure of a Process

```
do {
    entry section
            critical section
    exit section
    remainder section
  } while(true);
```

General Structure of a Process

- A critical section problem must satisfy the following three requirements :
- **Mutual Exclusion** : No two processes can be in their critical sections concurrently.
- **Progress** : Selection of a process wishing to enter the critical section, cannot be postponed indefinitely.
- **Bounded Waiting** : There exists a limit or bound 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.
- It should be ensured that the OS is free from Race Condition.

Critical Section Problem

- Two general approaches used to handle critical sections in OS :
- **Preemptive Kernel** : Allows a process to be preempted while it is running in kernel mode.
- **Non-preemptive Kernel** : Doesnot allow a process running in kernel mode to be preempted.
- Non-preemptive kernel is free from race condition on kernel data structures, as only one process is active in the kernel at a time.
- Preemptive kernels are difficult to build for SMP architectures.
- Preemptive kernel is suitable for real-time programming and is more responsive.

Peterson's Solution

- A classic software based solution to the critical-section problem.
- Peterson's solution is restricted to two processes(P_0 and P_1.
- The two processes share, int turn; boolean flag[2].
- turn indicates the process id, allowed to execute in its critical section.
- if flag[i] is true, it indicates that P_i is ready to enter its critical section.
- To enter the critical section, P_i sets flag[i] = true and turn = j allowing the other process to CS if it is ready.
- Even if both processes wish to enter CS at the same time, turn will be either i or j.

General Structure of Process(Pi) in Peterson's Solution

```
do {
   flag[i] = TRUE;
   turn = j;
   while ( flag[j] && turn = j ) no op;
   critical section
   flag[i] = FALSE;
   remainder section
  } while(true);
```

General Structure of Process(Pi) in Peterson's Solution

- The above solution satisfies the critical section problem requirements :
- **Mutual Exclusion** : No two processes can be in their critical sections concurrently.
- **Progress** : Selection of a process wishing to enter the critical section, cannot be postponed indefinitely.
- **Bounded Waiting** : There exists a limit or bound 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.

Synchronization Hardware : Solution to the critical section problem using locks

```
do {
     acquire lock
   critical section
     release lock
   remainder section
  } while(true);
```

- Critical regions are protected by **locks**.
- Disabling interrupts, can solve race condition but it cannot be used in a multi-processor environment.

Synchronization Hardware : Solution to the critical section problem using locks

- Special hardware support to test and modify the content of a word or to swap the contents of two words **atomically**.
- TestAndSet and Swap instructions are uninterruptible.

```
boolean TestAndSet(bool *target) {
   bool rv = *target;
   *target = TRUE;
   return rv;
 }
void Swap(bool *a, bool *b) {
   bool temp = *a;
   *a = *b;
   *b = temp;
```

Synchronization : Solution to the critical section problem using Semaphores

- A semaphore S is an integer variable, apart from initialization, is accessed only through two standard atomic operations wait called P and signal also called V.
- Counting Semaphore : the value can range over an unrestricted domain.
- Binary semaphore : the value can range only between 0 and 1 and also known as Mutex locks.
- Spin Lock : Busy waiting wastes CPU cycle, if a semaphore waits wasting CPU cycles while waiting for the lock, it is called as spin lock.

Synchronization : Solution to the critical section problem using Semaphores

```
boolean wait(S)
{
   while S <=0
   ; // no-op
   S--;
}
  \
 signal(S)
{
    S++;
}
```

Semaphore Implementation

```
typedef struct {
        int value;
        struct process *list;
}
wait(semaphore *S) {
        S->value--;
        if(S->value < 0) {
                add this process to S->list;
                block();
        }
}
```

Semaphore Implementation

```
signal(semaphore *S) {
        S->value++;
        if(S->value <=0) {
            remove a process P from S->list;
            wakeup(P);
         }
       }
```

Semaphore Implementation

- block operation suspends the process that invokes it.
- wakeup$P$ operation resumes the execution of a blocked process P.
- The negative value of a semaphore denotes the number of processes waiting on that semaphore.
- The positive value denotes the number of instances of the resource available for service.

Deadlocks and Starvation

- A situation where two or more processes are waiting indefinitely for an event to occur is called **deadlock**.
- Misuse of semaphore can cause deadlock.
- The two semaphores S and Q are initialized to 1.
- P_0 : wait(S); wait(Q); .... signal(S); signal(Q);
- P_1 : wait(Q); wait(S); .... signal(Q); signal(S);

Deadlocks and Starvation

- **Indefinite Blocking or Starvation** is a situation in which processes wait indefinitely within the semaphore.
- **Prority Inversion** : Three processes L, M and H with priorites in the order L ¡ M ¡ H. If H requires resource R held by L, but waits for L to finish, but M becomes runnable preempting process L. A process with lower priority(M) has affected how long process H must wait for L to relinquish R.
- Priority inversion occurs only in systems with more than two priorities.
- **Priority inheritance Protocol** : All processes accessing resources needed by a higher priority process inherit the higher priority until they are finished with the resources in question.

Classic Problems of Synchronization

- Bounded Buffer Problem.
- mutex semaphore provides mutual exclusion for access to the buffer pool and is initialized to the value 1.
- The empty and full semaphores count the number of empty and full buffers.
- The semaphore empty is initialized to N.
- The semaphore full is initialized to 0.

Bounded Buffer Problem : Producer Process

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

Bounded Buffer Problem : Consumer Process

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

Readers-Writers Problem

- Two or more readers can access the shared data simultaneously.
- Every writer should get access to the data exclusively.
- The structure of a writer process is explained below.

```
do {
        wait(wrt);
        ....
        // writing is performed
        ....
        signal(wrt);
    } while(TRUE);
```

Reader Process : Readers-Writers Problem

```
do {
        wait(mutex);        readcount++;
        if (readcount == 1)
         wait(wrt);
        signal(mutex);
        ....        // reading is performed        ....
        wait(mutex);        readcount--;
        if(readcount == 0)
         signal(wrt);
        signal(mutex);
     } while(TRUE);
```

Dining Philosophers Problem

- Five philosophers spend their lives thinking and eating.
- The philosophers share a circular table.
- The table is laid with five single chopsticks.
- A philosopher gets hungry and tries to pick up the two chopsticks that are closest to her.
- Each chopstick is represented with a semaphore.
- The solution can lead to deadlock if each pick up one chopstick.
- Allow a philosopher to pick up her chopsticks only if both chopsticks are available.
- An odd philosopher picks up first her left chopstick and then her right chopstick, whereas an even philosopher picks up her right chopstick and then her left chopstick.

Dining Philosophers Problem

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

Monitor

- Monitor is an Abstract Datatype(ADT) encapsulating private data with public methods to operate on that data.
- Synonymous to objects, a monitor encapsulates variables along with the bodies of procedures or functions that operate on those variables.

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

Part 2 E

Deadlock

Deadlock

- A process requests resources, if the resources are not available at that time, the process enters a waiting state, and stays there indefinitely, this is **deadlock**.
- When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone.
- A system consists of a finite number of resources to be distributed among a number of competing processes.
- A process may utilize a resource in only the following sequence :
- Request- may have to wait if being in use. Use- Use the resource Release- Process releases the resource.
- Examples request and release device, open and close file, and allocate and free.

Deadlock Characterization

- A deadlock situation can arise if the following four conditions hold simultaneouslyin a system :
- **Mutual Exclusion** : Atleast one resource must be held in non-sharable mode, if a process requests that resource, the requesting process must be delayed until the resource has been released.
- **Hold and Wait** : A process must be holding one resource and waiting to acquire additional resources that are currently held by other process.
- **No preemption** : Resources cannot be preempted; a resource can be released only voluntarily by the process holding it.
- **Circular Wait** : A set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes must exist such that P_0 is waiting for a resource held by P_1 and P_n is waiting for a resource held by P_0.

Resource Allocation Graph

- Deadlocks described with directed graph called system resource-allocation graph.
- A graph consists of a set of vertices V and set of edges E.
- The set of vertices V is partitioned into different types of nodes P = { P_1, P_2,..., P_n}, the set of all active processes in the system and R = {R_1, R_2, ..., R_m} set of all resource types in the system.
- $P_i \rightarrow R_j$ signifies P_i has requested an instance of resource R_j, called as request edge.
- A directed edge $R_j \rightarrow P_i$, is called an assignment edge.
- Each process P_i is represented as a circle amd each resource type R_j as a rectangle with a dot for each instance of the resource.

Resource Allocation Graph

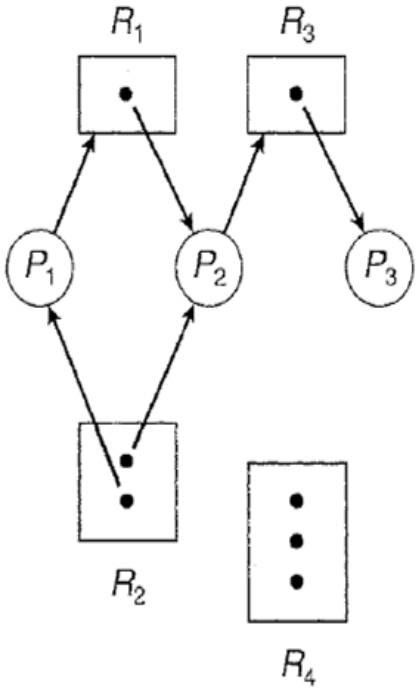


Figure 7.2 Resource-allocation graph.

Methods for handling Deadlock

- Deadlock can be dealt in one of the three ways :
- 1. Use a protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlocked state.
- 2. Allow the system to enter a deadlocked state, detect it and recover.
- 3. Ignore the problem and pretend that deadlocks never occur in the system.
- Third solution is used by most OS UNIX and Windows.
- To ensure deadlocks never occur, the system can use either a deadlock prevention or a deadlock-avoidance scheme.

Methods for handling Deadlock

- Deadlock Prevention provides a set of methods for ensuring that atleast one of the necessary conditions cannot hold.
- Deadlock avoidance requires that OS be given in advance additional information concerning which resources a process will request and use.
- When deadlocks occur, system performance deteriorates and eventually stops working needing maunal restart.
- To ensure deadlocks never occur, the system can use either a deadlock prevention or a deadlock-avoidance scheme.

Deadlock Prevention

- Deadlock Prevention : Mutual Exclusion - Cannot deny mutual exclusion, since some resources are intrinsically non-sharable.
- Hold and Wait - One protocol requires each process to request and be allocated all its resources before it begins execution and holds on to the resources for the entire duration.
- Hold and Wait - Another protocol allows the process to request resources only when it has none.
- Disadvantage : low resource utilization and starvation.
- No Preemption - If a process holding some resource requests another resource that cannot be allocated, then the held resources are preempted and released.
- Circular Wait - Each resource type is assigned a unique number, and a process can request resources only in an

Deadlock Avoidance

- Each process declares the maximum number of resources of each type that it may need.
- With this apriori information, an algorithm can ensure that the system will never enter deadlocked state.
- Safe State : A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock*safesequence*.
- Resource Allocation Graph for deadlock avoidance : A resource request*edge* can be granted*convertedtoresourceassignmentedge* only if resource-allocation graph doesnot form a cycle.

Deadlock Avoidance : Safe State

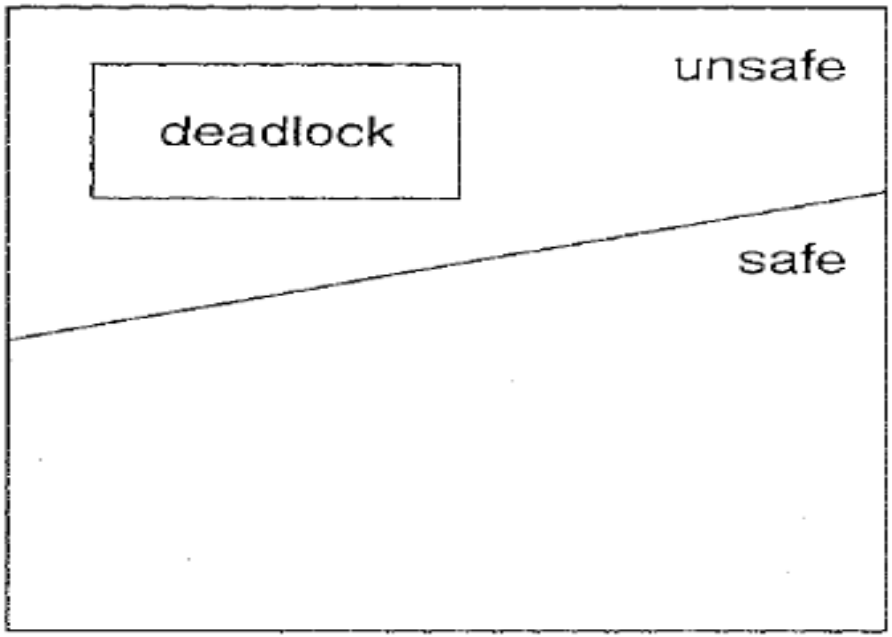


Figure 7.5 Safe, unsafe, and deadlocked state spaces.

Deadlock Avoidance : Banker's Algorithm

- A new process declares the maximum number of instances of each resources in the system.
- For any request for resources, the resources are allocated only if the allocation leaves the system in safe state.
- Data Structures used : n : number of processes, m : number of resource types :
- Available : Available[j] = k means k instances of resource type R_j are avilable.

Deadlock Avoidance : Banker's Algorithm

- Max[i][j] = k means process P_i may request at most k instances of resource type R_j.
- Allocation[i][j] equals k, then process P_i is currently allocated k instances of resource type R_j.
- Need[i][j] equals k, then process P_i may need k more instances of resource type R_j to complete its task.
- Safety Algorithm : finding whether the system is in a safe state.

Exercise 8

Theorem Orange.

Find whether the System is in Safe State using Banker's Algorithm

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

Deadlock Detection

- If a system doesnot employ deadlock-prevention or deadlock avoidance, then deadlock may occur.
- Then the system must provide an algorithm that examines the state of the system to determine whether deadlock has occurred.
- An algorithm to recover from the deadlock.
- If resources are of single instances, then wait-for graph derived from resource-allocation graph can be derived.
- If the wait-for graph contains a cycle, then deadlock exists in the system.
- Wait-for graph not applicable for multiple instances of graph.

Deadlock Detection

- For multiple-instances of resource types, the algorithm is similar to Banker's algorithm is used.
- How often deadlock detection algorithm be run?
- At every resource request is too costly.
- Ideally once per hour or whenever CPU utilization drops below 40%.
- After Deadlock is detected, process termination, resource preemption.
- Process termination :abort all deadlocked processes. Abort one process at a time until the deadlock is eliminated.
- Resource Preemption should be gracefully done avoiding instability and starvation.

Part 3

Memory Management

Part 3a

Memory Management Strategies

Basic Hardware

- CPU can operate on register contents at the rate of few ops/clock tick.
- Memory access via memory bus may take many CPU cycles.
- Cache is a fast memory buffer used to normalize the speed difference.
- Process Memory protection through base register and limit register.
- The processes on the disk that are waiting to be brought into memory form the input queue.

Memory Protection for a Process

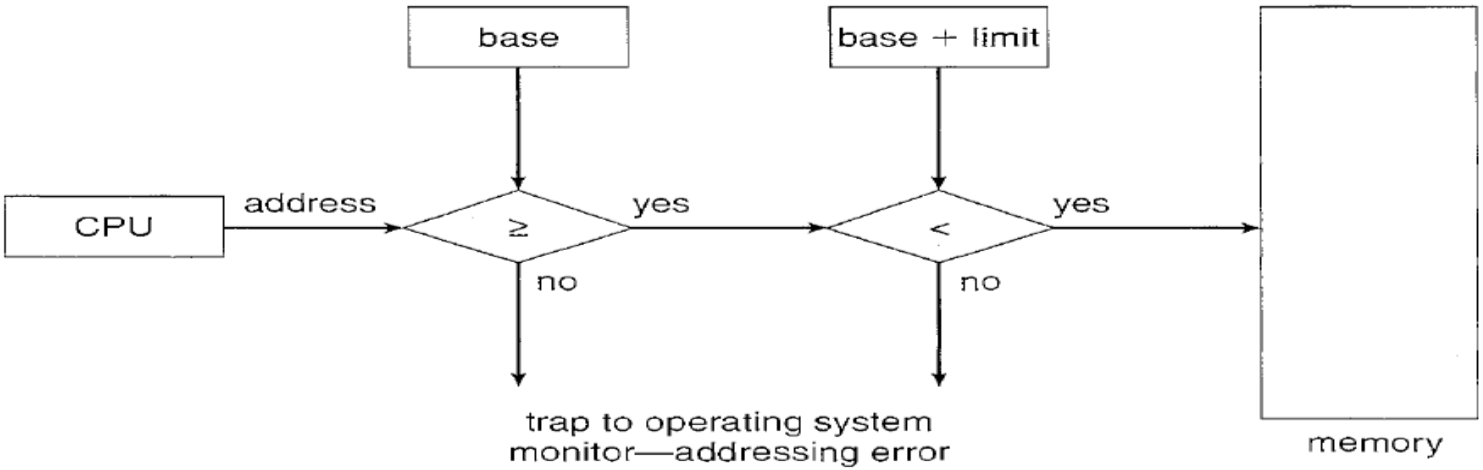


Figure 8.2 Hardware address protection with base and limit registers.

Address Binding

- A compiler will bind the symbolic addresses to relocatable addresses.
- Linkage editor or loader will bind the relocatable addresses to absolute addresses.
- Binding can be compile time or load time.
- Compile time binding : The memory address where the process will reside is decided at compile time(absolute code generated). To change the stating location, the code should be recompiled.
- Load time binding : Relocatable code is generated at compile time, final binding is delayed until load time.

Logical versus Physical Address Space

- Logical Address : An address generated by the CPU.
- Physical Address : address loaded into memory-address register and seen by the memory unit.
- Compile-time and load-time address-binding methods generate identical logical and physical addresses.
- Execution-time address binding scheme results in differing logical and physical addresses, logical address called as virtual address.
- Memory-Management Unit(MMU) : Run-time mapping from virtual to physical addresses.
- The base register is called the relocation register and every address generated is added to it.

Dynamic Relocation

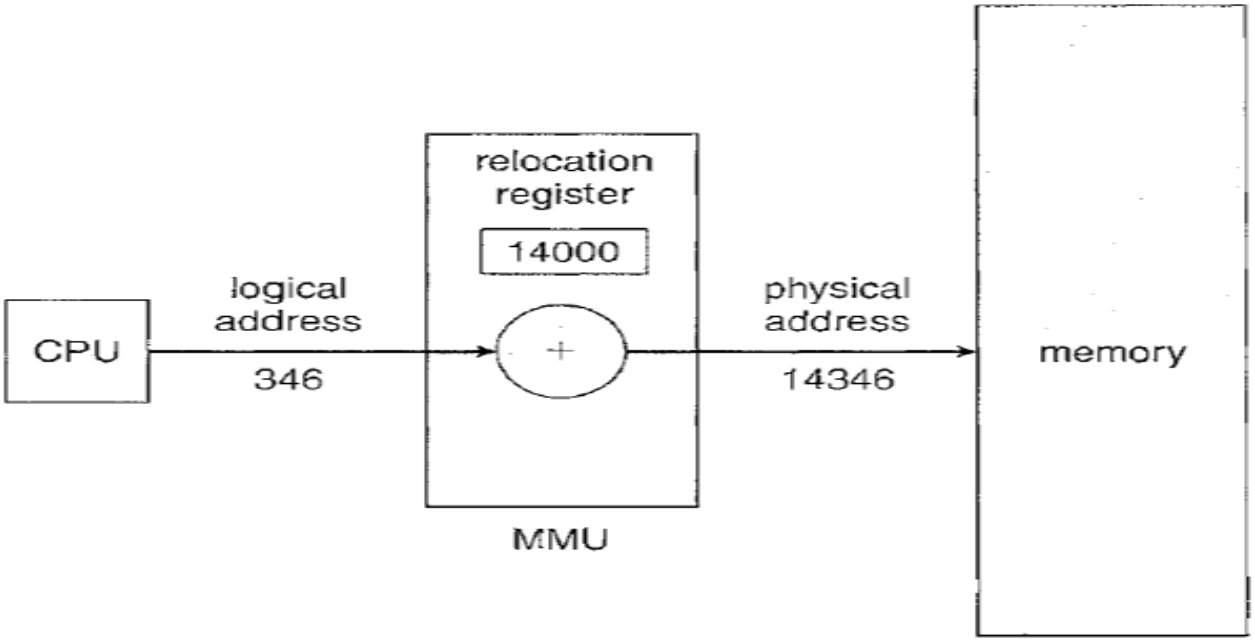


Figure 8.4 Dynamic relocation using a relocation register.

Dynamic Loading and Dynamic Linking

- Dynamic Loading : For optimal memory utilization, a routine is not loaded until it is called. All routines are kept on disk in relocatable load format.
- Static linking : System language libraries are treated like any other object module and are combined by the loader into binary program image.
- Dynamic Linking : linking is postponed until execution time, used with system libraries.
- A stub(indicating how to locate the routine) is included in the image for each library-routine reference.

Dynamic Loading and Dynamic Linking

- All processes that use a language library execute only one copy of the library code.
- Dynamic linking requires help from the OS, since the needed routine is in another process's memory space.
- Swapping : A process may temporarily be swapped out of memory to the secondary storage.
- Roll-out and Roll-in : swapping out lower priority process for a higher priority process.

Contiguous Memory Allocation

- Memory is divided into two partitions : one for resident OS and one for user process.
- OS might be in either low memory or high memory.
- In contiguous memory alloaction, each process is contained in a single contiguous section of memory.
- Memory Mapping and Protection is achieved with Relocation register and limit register.
- The MMU maps the logical address dynamically by adding the value in the relocation register.
- Transient OS Code : Code that comes and goes out of memory on need basis(eg., Device drivers).

Contiguous Memory Allocation

- **Fixed Partition Scheme** : Memory is divided into fixed size partitions and allocated to each process.
- Number of partitions limit the degree of multiprogramming.
- **Variable-partition Scheme** : OS keeps a table indicating the variable size and start address for each partition.
- A Hole : One large block of available memory.
- When a process needs memory, the system searches for hole large enough for the process.
- If the hole is too large, it might be split and allocated.

Memory Allocation Strategies

- Dynamic Storage Allocation : First fit, best-fit and worst-fit strategies.
- First fit and Best fit strategies suffer from external fragementation.
- 50-percent Rule : Statistical analysis revelas that with first-fit, for every N allocated blocks, 0.5 N blocks are lost to fragmentation.

Fragmentation Strategies

- External Fragmentation : There is enough memory space, but not contiguous.
- Internal Fragmentation : The difference between memory allocated to a process and the memory requested which can be slightly lesser.
- Solution to external fragmentation is compaction which is possible only if relocation is dynamically done at execution time.
- Other solutions are Paging and Segmentation.

Paging

- Paging is a memory management scheme the permits the physical address space of a process to be non-contiguous.
- Paging avoids external fragmentaion and compaction.
- Paging is implemented by breaking physical memory into fixed-sized blocks called **frames**.
- The logical memory is broken into same size blocks called **pages**.
- Every address generated by the CPU(logical address) is divided into two parts : a page number(p) and a page offset(d).
- The page number is the index into the page table which contains frame number as its value.

Paging Hardware

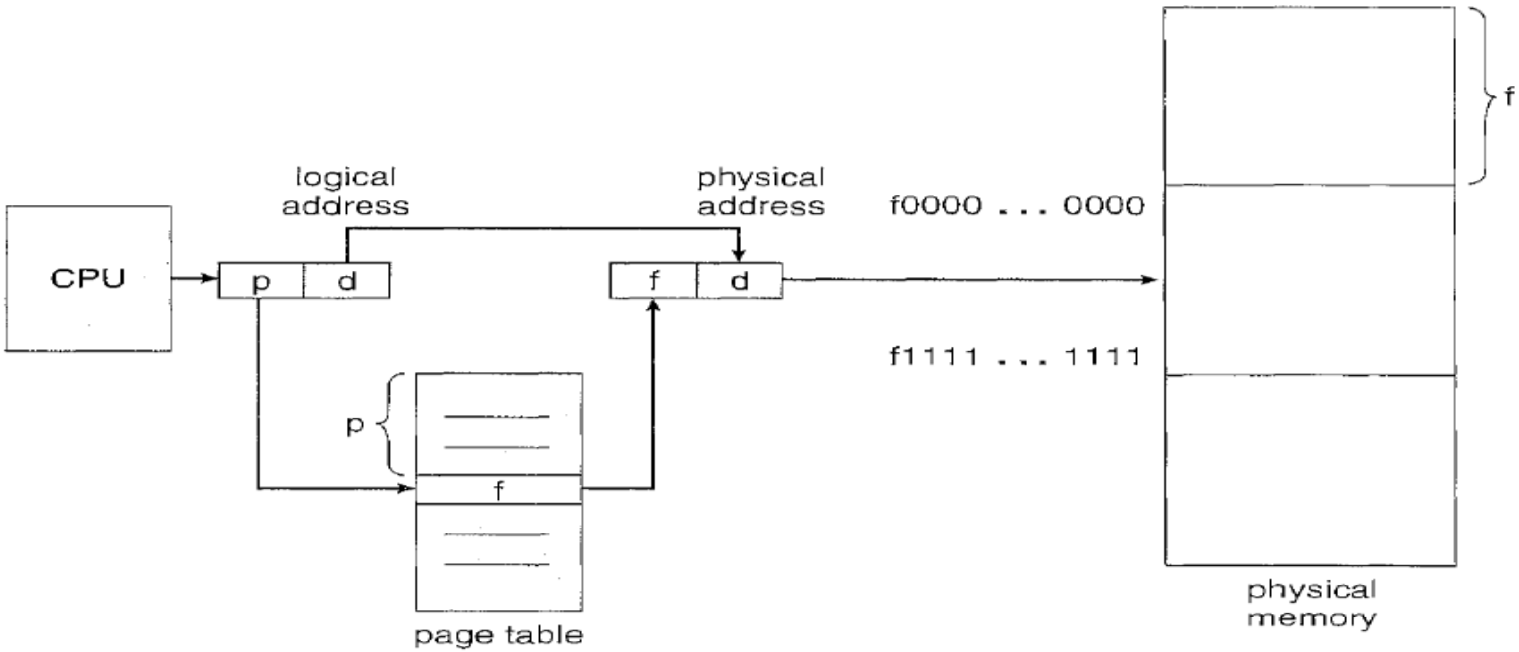


Figure 8.7 Paging hardware.

Paging

- The size of a page defined by hardware is between 512 Bytes and 16 MB depending upon architecture.
- The page table is implemented as a dedicated set of registers.
- Every address is divided into two parts : page number(p) and page offset(d).
- The page number(p) is an index into the page table.
- If logical address is 2^m and page size is 2^n, then higher-order $m - n$ bits designate page no, and n low-order bits designate page offset.
- Paging eliminates external fragmentation but suffers from internal fragmentation.

Paging Model

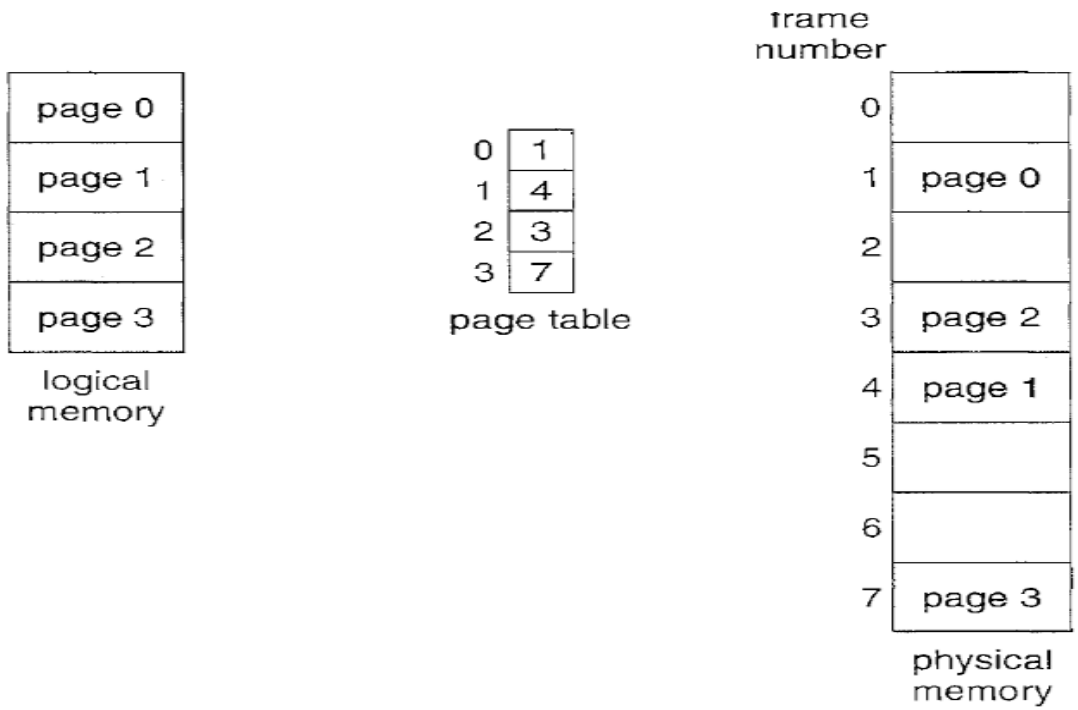


Figure 8.8 Paging model of logical and physical memory.

Paging

- **Frame Table** - datastructure used for allocation details of physical memory.
- The frame table has one entry for each physical page frame.
- Paging increases context-switch time.
- A page table is allocated for each process.
- A pointer(Page Table Base Register) to the page table is stored in PCB.
- Since every memory access needs to access the page table and then the actual content, the speed is reduced by half.

TLB Paging

- Translation look-aside buffer(TLB) a fast lookup hardware cache is used.
- In case of a **TLB miss**, a memory reference to the page table must be made.
- TLB entries for OS are wired down.
- **Hit Ratio** : % times a particular page number found in TLB.
- If 20 ns is the TLB access time, 100ns is the memory access and 80% is the hit ratio then the access latency is $EffectiveAccessTime = 0.8 * 120 + 0.2 * 220 = 140ns$

TLB Supported Paging Model

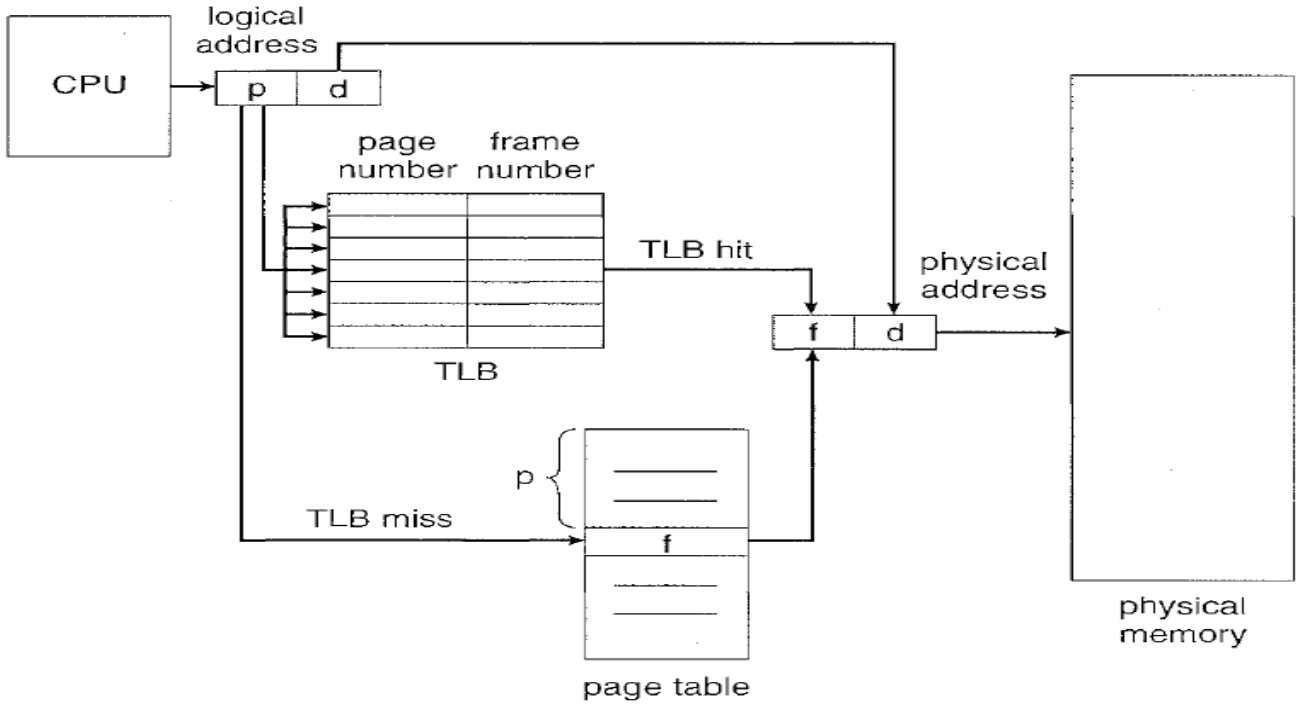


Figure 8.11 Paging hardware with TLB.

Paging

- Advantage of paging is sharing common code.
- Reentrant code is non-self modifying code, it never changes during execution.
- Example : The code(pages) for the editor can be shared among several users, with their own private copy of data(pages).
- Hierarchical Paging : The page table itself becomes large, then the page table is divided into smaller pieces.
- Two level Paging : The page table is also paged.
- A 64-bit architecture requires seven levels of paging.

Sharing Pages

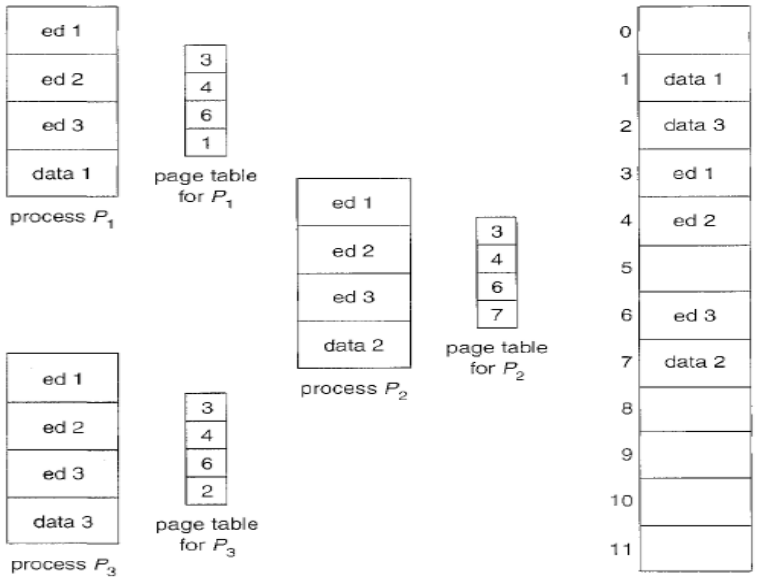


Figure 8.13 Sharing of code in a paging environment.

Two Level Paging

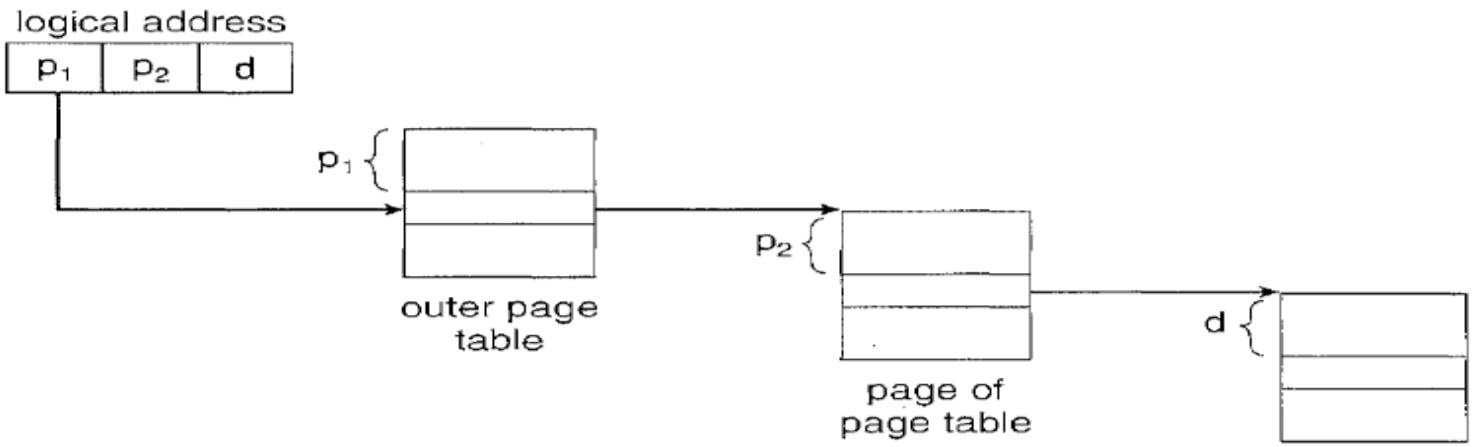


Figure 8.15 Address translation for a two-level 32-bit paging architecture.

Two Level Paging

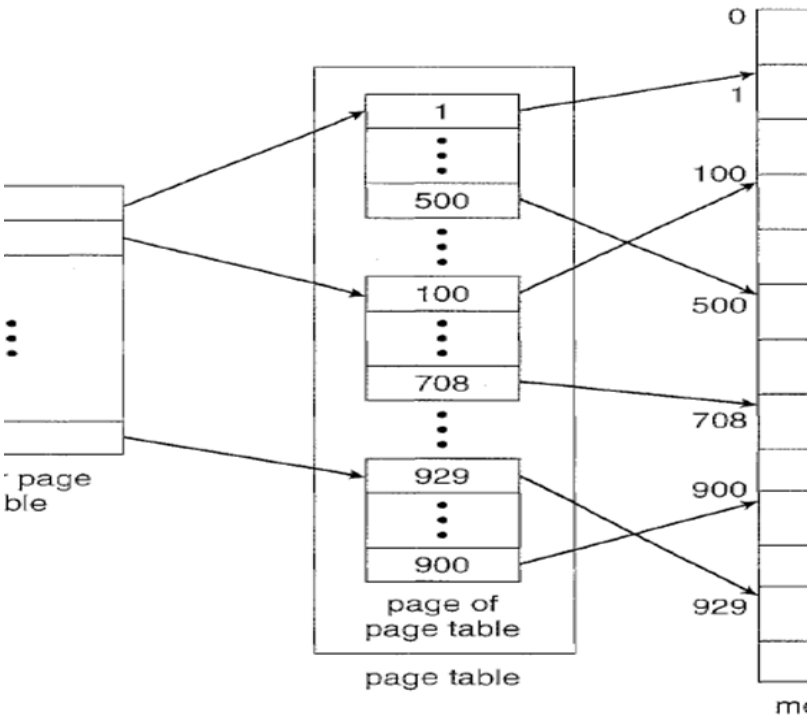


Figure 8.14 A two-level page-table scheme.

Hashed Page Tables

- Hashed Page Table is a common approach for handling address spaces larger than 32 bits.
- The virtual page number in the virtual address is hashed into the hash table.
- Each entry in the hash table contains a linked list of elements that hash to the same location.
- Each element consists of 1. The virtual page number, 2. The value of the mapped page frame, 3. A pointer to next element.
- Clustered Page Tables : Each entry in the hash table refers to several pages.

Inverted Page Tables

- An inverted page table has one entry for each real page of memory.
- Each entry consists of the virtual address of the page stored in that real memory location.
- Only one page table is in the system, and it has only one entry for each page of physical memory.
- An address-space identifier is stored in each entry of the page table which ensures mapping of logical page to corresponding physical page frame.
- <process-id, page-number, offset>

Inverted Page Tables

- Since inverted page table is sorted by physical address, but lookups occur on virtual addresses, whole table might need to be searched for a match.
- Inverted page tables have difficulty implementing shared memory,since every physical page is mapped to one virtual page entry.

Inverted Page Table

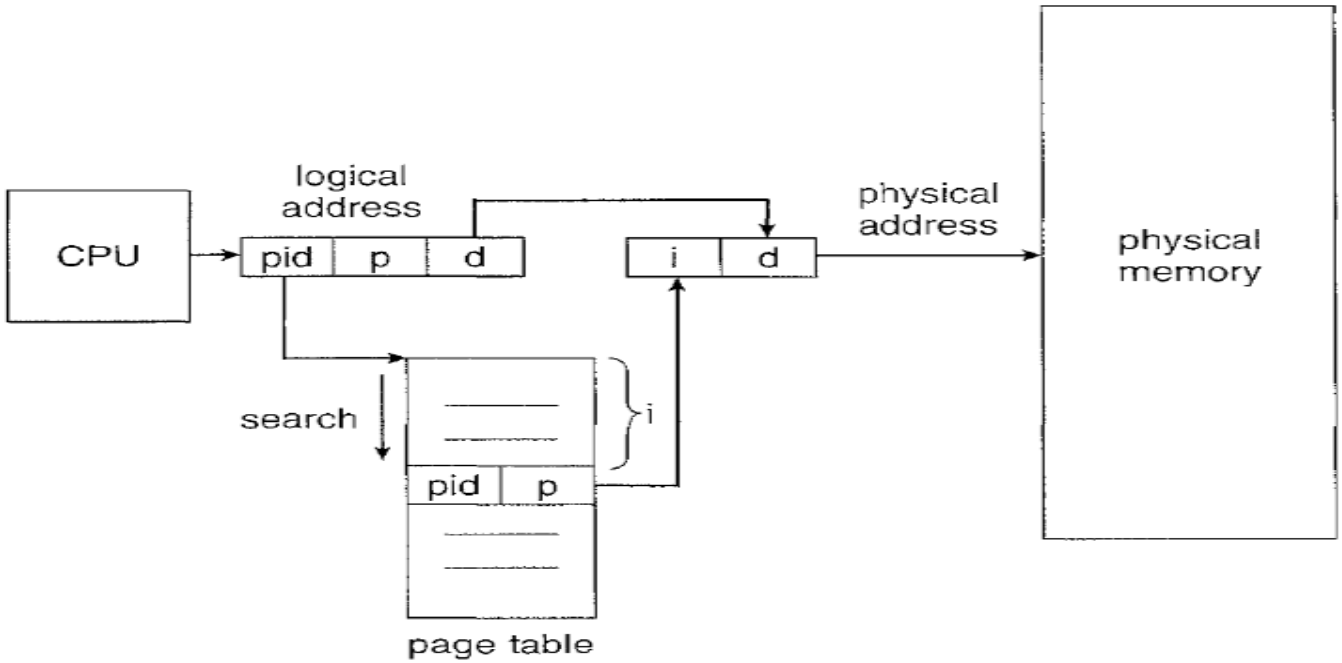


Figure 8.17 Inverted page table.

Segmentation

- Users prefer to view memory as a collection of variable-sized segments.
- A logical address space is a collection of segments.
- <segment-number, offset>
- Each entry in the segment table has a segment base and a segment limit.
- Pentium supports segment size of 4 GB and max number of segments per process is 16K.
- Pentium supports page size of 4KB or 4MB.

Part 3b

Virtual Memory Management

Virtual Memory

- Virtual memory involves the separation of logical memory from physical memory as perceived by users.
- Programmer need not worry about the size of physical memory available.
- Virtual Address space of a process refers to the logical view of how a process is stored in memory.
- Demand Paging : Load pages only as they are needed.

Virtual Memory

- Lazy Pager : swaps a page into memory only upon need.
- When a process is swapped in, the pager guesses which pages will be used, and brings only those pages into memory.
- Valid-invalid bit distinguish between pages in memory and pages on the disk.

Page Fault

- Page fault : Access to a page marked invalid, causes a trap to the OS.
  1. An internal table of the process is checked to determine the reference as valid or invalid.
  2. If the reference was invalid, the process is terminated.
  3. if it was a valid reference, but page is not in memory, it is paged in.
  4. A free frame is identified.
  5. A disk operation to read the desired page into the newly allocated frame.
  6. After the disk read is complete, the internal table and the page table is updated to indicate that page is in memory.
  7. The instruction that was interrupted by the trap is restarted.

Pure Demand Paging

- Pure Demand Paging : Page faults happen until all required pages are in memory.
- Pure Demand Paging : Never bring a page into memory until it is required.
- $Effective\_Access\_Time = (1 - p) * ma + p * page\_facult\_time$
- Where p = Probability of Page fault, ma = memory access(200ns), $page\_fault\_time(8ms)$.

Pure Demand Paging

- fork() - Creates copy of Pages
- Copy On Write - Pages are shared until modified.
- vfork() - Efficient way of process creation used in Unix.

Page Replacement Algorithms

- During Page Replacement - if no frame is free, a frame not currently being used is freed by writing its contents to swap space.
- The page table is updated to indicate that the page is no longer in memory.
- The page fault time doubles - since two pages are transferred.
- Modify Bit or Dirty Bit : Indicates whether a page is modified.
- Major aim of Page replacement algorithm is reducing page faults.

Page Replacement Algorithms

- FIFO : Simplest Page-Replacement Algorithm.
- FIFO associates time with each page when it is brought into memory.
- With the Reference string : 1,2,3,4,1,2,5,1,2,3,4,5
- Number of page faults increases from three frames to four!!
- This result is known as Belady's anomaly - Page fault rate increases with increase of allocated frames.

Optimal Page Replacement Algorithm

- Requires future knowledge of the reference string.
- will only replace pages that are not required in near future.
- Difficult to implement similar to SJF CPU scheduling algorithm.

LRU Page Replacement Algorithm

- Its performance is between FIFO and Optimal Page replacement algorithm.
- Replace the page that has not been used for the longest period of time.
- It uses the recent past as an approximation of the near future.
- Counter or logical clock or stack can be used to implement LRU.

Second-Chance Page Replacement Algorithm

- A reference bit is maintained for each page representing whether it is used in the previous time period.
- If the value is 0, the page is replaced.
- If the value is 1, the page is given a second chance by clearing the reference bit.
- If all bits are set, it performs like FIFO.
- Can be implemented with a circular queue.

Enhanced Second-Chance Page Replacement Algorithm

- Reference bit and modify bit as ordered pair.
- (0,0) : Neither used recently nor modified. Best page to replace.
- (0,1) : Not recently used but modified. Not quite as good.
- (1,0) : recently used but clean. Might be used again soon.
- (1,1) : recently used and modified.

Allocation Algorithms

- Equal Allocation : Equal number of frames are allocated to each process.
- Proportional Allocation : Number of frames allocated depends on the size of process.
- Proportional Allocation based upon priority of processes.
- Page-replacement algorithms are classified into global replacement and local replacement.

Allocation Algorithms

- Global Replacement allows a process to select a replacement frame from the set of all (global)frames.
- Local Replacement : each process selects from only its own set of allocated frames.
- With global replacement algorithm, a process cannot control its own page fault rate.

Thrashing

- Thrashing : High paging activity, spending more time on paging than on executing.
- CPU utlization increases with degree of multiprogramming.
- But when multiprogramming is increased beyond a threshold, it causes thrashing.
- Thrashing can be limited with local allocation instead of global allocation.

Thrashing

- Working Set : The most recent 't' page references form a working set.
- By studying working set, appropriate number of frames can be allocated thus preventing thrashing.
- The page fault Rate can be measured and should be within upper and lower threshold limits to avoid thrashing.

Part 4

Storage Management

Part 4a

File System

File System Introduction

- File System consists of, a collection of files and Directory Structure.
- Storage media : magnetic disks, magnetic tapes, optical disks.
- OS provides a uniform logical view of information storage.
- OS abstracts from the physical properties of its storage devices.
- File : is defined as alogical storage unit.
- Data cannot be written to secondary storage unless thay are within a file.

File Attributes

- Name : Symbolic file name in human readable form.
- Identifier : Unique tag usually a number, identifies the file within the file system.
- Type : This info is needed for systems that support different types of files.
- Location : pointer to the device and to the location of the file on that device.
- Size : The current size of the file and the maximum allowed size are included in this attribute.
- Protection : Access-control information determines who can do reading, writing, executing and so on.
- Time, Date and user identification : These information for creation, last modification and last use are recorded.

File Structure

- The information about all files is kept in the directory structure residing on secondary storage.
- A file is an abstract data type.
- Operations on a File : Creating a File, Writing a file, Reading a file, Repositioning within a file, Deleting a file, truncating a file.
- OS uses two levels of internal tables : A per-process table and a system-wide table.
- Each entry in the per-process table inturn points to a system-wide open-file table.

File Structure

- Open file is associated with File Pointer, File-open-count, Disk location of the file, access rights, file-locking.
- The file types are represented as extensions. .com, .exe, .bat,
- UNIX system uses crude magic number stored at the beginning of some files to indicate type of file, executable, shell scripts, etc.,
- If the OS supports five different file structures, then the size of the OS becomes large.
- All OS must support atleast executable file type.

Internal File Structure

- The disk space is always allocated in blocks.
- A file with 1949 Bytes would be allocated 4 blocks of 512 each.
- The space wasted in the last block is called internal fragmentation.
- File Access Methods : Sequential Access, Direct Access, Indexed Access.

Directory and Disk Structure

- A disk is partitioned into partitions or quarters.
- Each quarter or partition can hold a file system.
- Any entity containing a file system is generally known as a volume.
- A directory is a symbol table that translates file names into their directory entries.
- Operations on a Directory : Search for a file, create a file, delete a file, list a directory, rename a file, tranverse the file system.

Directory Structure

- Single Level Directory : All files are contained in the same directory.
- Single Level Directory : Unique file name constraint is difficult to maintain.
- Two Level Directory : Each user has his own user file directory(UFD).
- Two Level Directory : File names within each UFD should be unique.
- Two Level Directory : UFD isolates users disallowing sharing of file.

Directory Structure

- Master File Directory(MFD) is the root.
- Tree Structured Directories : Tree is the most common directory structure and prohibits sharing of files or directories.
- Path names are of two types : Absolute and relative.
- Acyclic Graph is a natural generalization of the tree-structures directory scheme allowing sharing of files or directories.
- Shared files and subdirectories can be implemented using Link.

Directory Structure

- A Link is effectively a pointer to another file or subdirectory.
- OS ignores these links when traversing directory trees preserving acyclic structure.
- Deletion of a Directory is driven by policy : In Windows you cannot delete a directory unless it is empty.
- In Linux, with appropriate options, you can delete recursively.
- Deletion of a file when a link is referring to the file, deletes the file and leaves the link dangling in both linux and windows.

Directory Structure

- Keep count of the number of references, deleting a link decrements the count.
- When count is 0, the file is deleted, Unix uses this approach for hardlinks.

File System Mounting

- A file system must be mounted before it can be available to processes on the system.
- Mount point is the location within the file structure where the file system is to be attached.
- At boot time, all file systems are identified and mounted in Linux, Windows, etc.,
- Remote File Systems : FTP allowed files to be transferred between machines.
- Distributed File Systems(DFS) : remote directories are visible from a local machine.

File System Mounting

- WWW allows to access the remote file through browser.
- Consistency semantics specify how multiple users of a system are to access a shared file simultaneously.
- Access Control : Users classification into Owner, Group and Others.
- Each file can be password protected.
- Access Control List Precedence with clear ay conflist that arises.

File System Implementation : File System Structure

- I/O transfers between memory and disk are performed in units of blocks.
- Each block has one or more sectors.
- Sector Size varies from 32 Bytes to 4,096 Bytes.
- Usual size of a sector is 512 Bytes.
- The file-organization module knows about files their logical blocks as well as physical blocks.

File System Implementation : File System Structure

- The logical file system manages metadata information and file-system structure.
- The logical file system maintains file strucure via File Control Block(Inode)
- CD-ROMs are written in ISO 9660 format.
- Windows support FAT,FAT32 and NTFS file system formats.
- Linux supports Extended File System ext2 and ext3.

File System Implementation : File System Structure

- Disk Structure consists of : A boot control block(optional)
- A Volume Control Block(super block) : number on blocks, size of block, free block count, etc.,
- A Direcory Structure to organize file.
- A per-file FCB contains details about file.
- in-memory mount table contains information about each mounted volume.

File System Implementation : File System Structure

- in-memory directory structure cache holds the recently accessed directory information.
- System-wide open file table contains a copy of the FCB of each open file.
- Per-process open-file table contains a pointer to the appropriate entry in the system-wide open-file table.
- Directory implementation can be as a linear list of file names with pointer to the data blocks, or using Hash table.
- Allocation Methods : Contiguous, Linked and Indexed.
- Free Space Management : Bit Vector, Linked List, Grouping

Part 4b

Secondary-Storage Structure

Part 4c

I/O Systems