Chapter 3
Introduction

Chapter-3
Process

Program vs Process:

OS abstraction for execution

Program	Process
1. Program is a set of instruction	1. A process is a program in execution
2. A program is a ~~possibe~~ passive entry	2. A process is an active entry
3. Program has a longer lifespan	3. Process has a limited lifespan
4. It does not require any resources.	4. Process hold resources such as CPU, memory, address, disk, I/O etc.
5. It resides on a secondary memory	5. It resides on primary memory.

Why process is an active entity?

Ans: A process is sometimes referred as active entity as it resides on the primary memory and leaves the memory if the system is rebooted.

⇒ Several processes may related to same program.

Example: We can run multiple instances of a notepad program. Each instance is referred as process.

~~Why~~

A program is referred to as passive entity as it resides on a secondary memory.

CamScanner

→ CPU state
→ address space
→ resource

Structure of Process/Address space of process:

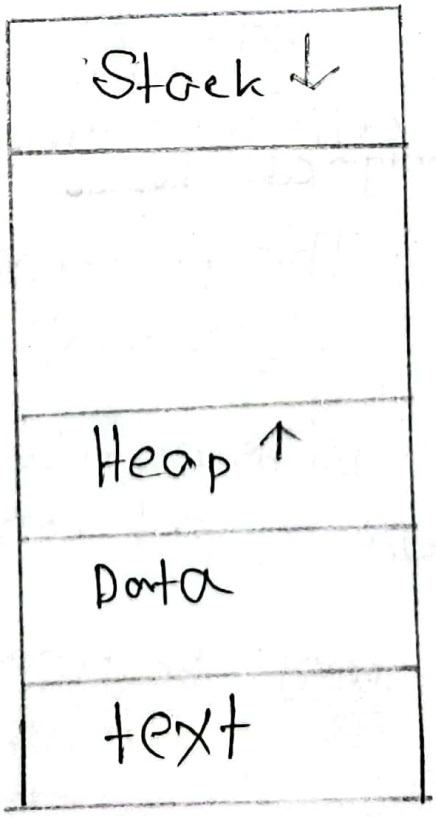
A process consists of-

→ a process stack → contains temporary data → function parameter, → return address, → local variable

→ heap → contains memory that is dynamically allocated during process run time

→ data → contains global variable

→ text section/code section → includes current activity as represented by the value of the program.



Process-state

The state of the process is defined ~~by~~ in part by the current activity of that process.

~~App~~ A process has following 5 states:

1. **New**: The process is being created.

2. **Running**: Instructions are being executed

3. **Waiting**: The process is waiting for some event to occur
   - → I/O operation
   - → reception of signal

4. **Ready**: Process is waiting to be assigned to a processor.

5. **Terminated**: The process is finished execution.

⇒ Only one process is can be running on any processor at any instant.

⇒ Many processor may be ready and waiting.

CamScanner

⇒ two more States are=
→ Suspend waiting
→ Suspend ready.

Process State diagram:

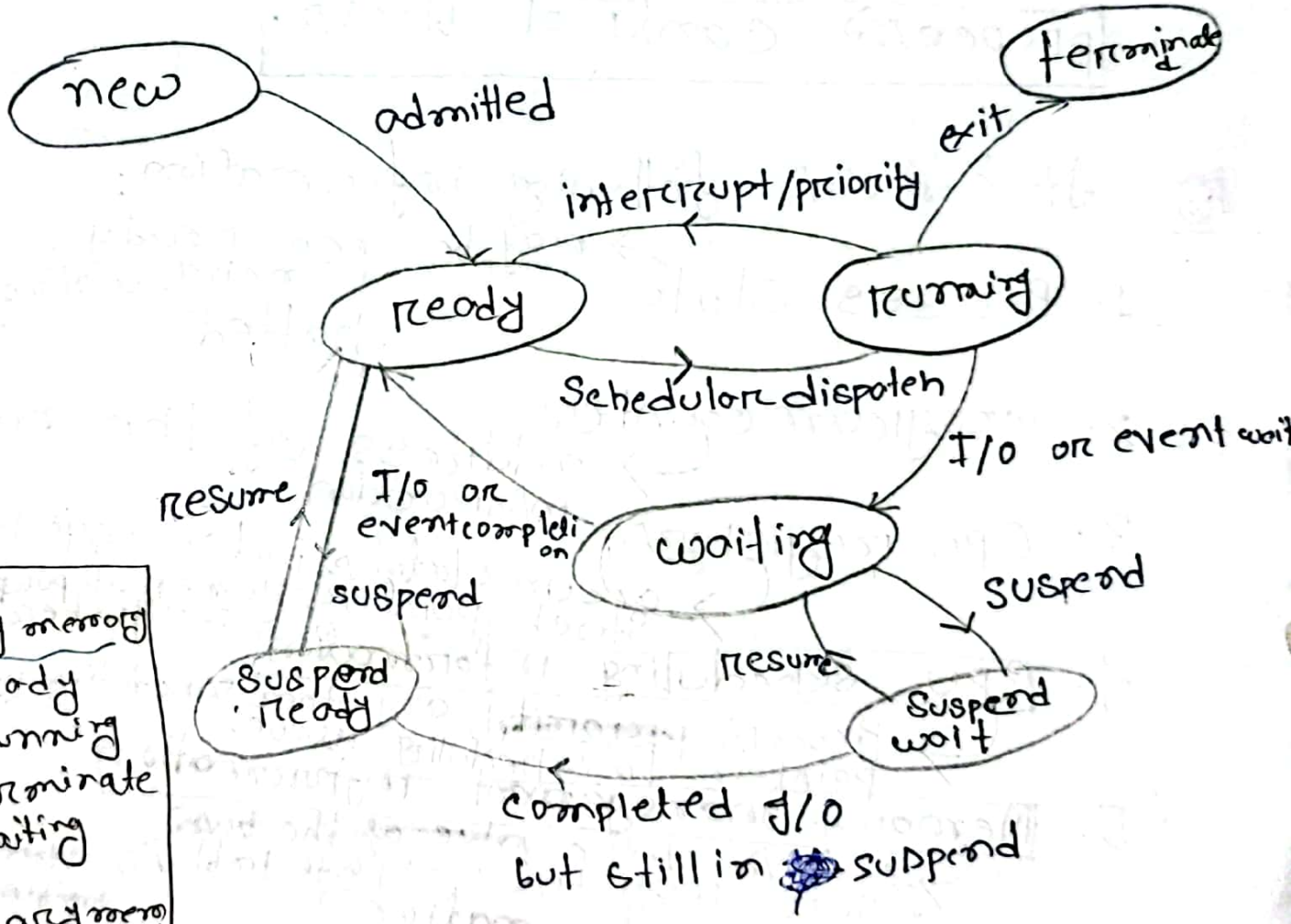


Fig: Diagram of process state.

Primary memory
→ ready
→ running
→ terminate
→ waiting

Secondary memory
→ new
→ suspend wait
→ suspend ready

☆☆☆

Process Control Block (PCB)

To implement the process model, the operating system maintains a table called the process table with one entry per process. Process table is also known as process control blocks.

It contains following information:

1. Process state → may be new, ready, running, waiting, halted
2. program counter → address of the next instruction
3. CPU registers → accumulators, index register, stack pointer, general purpose register
4. CPU scheduling information → Process priority, any other scheduling parameter, pointer to scheduling queue
5. ☆ Memory management information → items as the value of the base, limit register → page tables/segment tables
6. Accounting information → amount of CPU → time limits, account number → Job/process number.
7. I/O status information → listed I/O devices allocated to the process

process state
process number
program counter
register
memory limits
list of open files
...

Fig: Process control Block (PCB)

Attribute of PCB:

- → Process ID
- → Process State
- → Program counter
- → priority
- → ~~Gen~~ CPU registers
- → list of I/O devices
- → list of open files

} I/O status information

CamScanner

→ process priority
→ pointer to scheduling queue
→ scheduling ~~pointer~~ parameter

} cpu scheduling information

→ limit registers
→ page tables / segment tables
→ memory system used by operating system

} memory management information

→ amount of cpu
→ real time used
→ time limits
→ account numbers
→ job / process numbers.

} Accounting information.

→ protection.

* PCB serves as the repository for any information that may vary from process to process.

Fundamental task of OS:
→ must allocate resource
* → protect resource from other process
→ enable synchronization.

How → using PCB
describing process state

CamScanner

Scheduling queues

There are three (3) types of scheduling queue:

i. Job queue: It consists of all processes in the system. When processors enter the system they put into a job queue

ii. ready queue: The processes which are residing in main memory, ready to execute are kept here.
This queue is generally stored as a linked list. → header contains pointers to the first and final PCBs in the list.

iii. device queue: The list of processes waiting for a particular I/O devices is

called a device queue.
Each device has its own device queue.

Scheduling diagram:

A common representation of process scheduling is a queuing diagram.

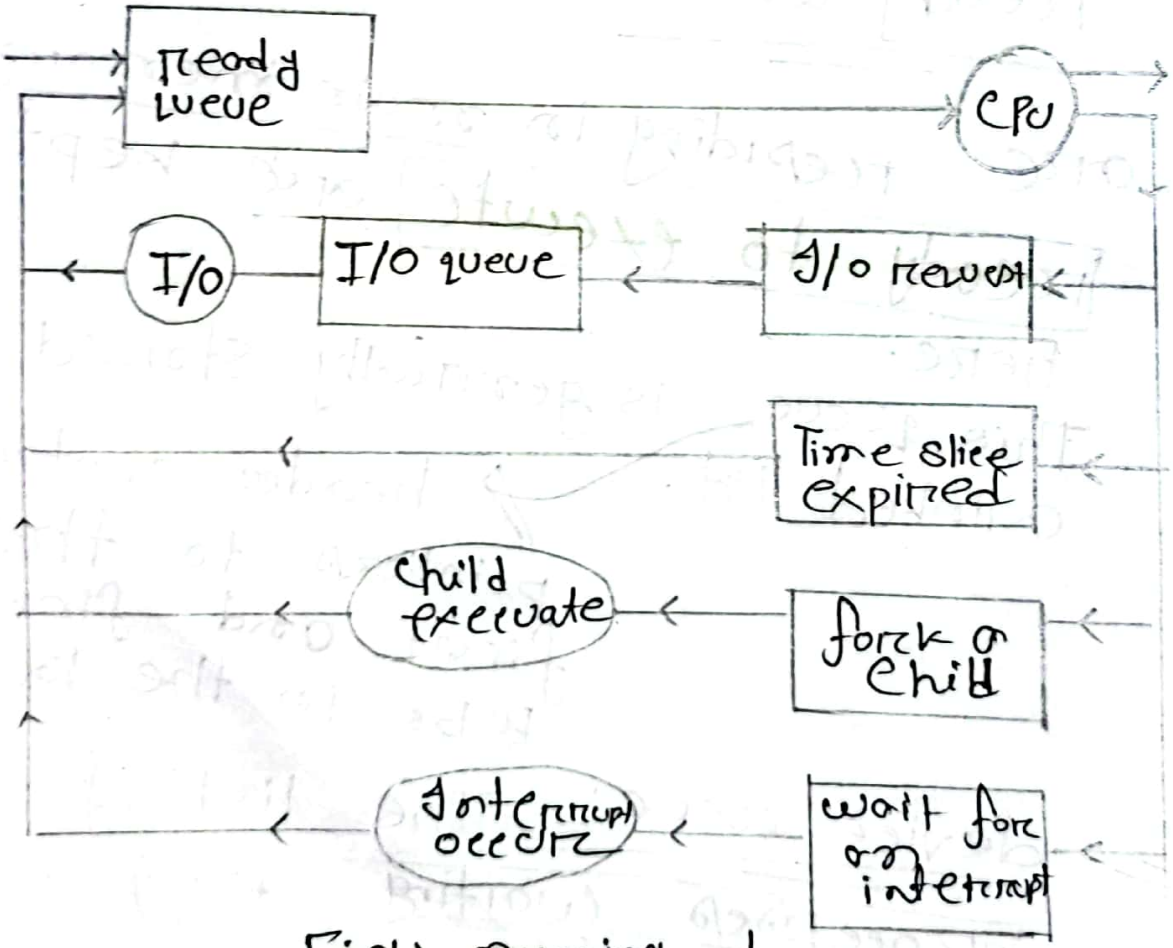


Fig: Queuing diagram

A new process is initially put in the ready queue.

It waits there until it is ~~exe~~ selected for execution or dispatched.
দ্রুত প্রেরণ

Once the process is allocated CPU and executing and following events could occur:

→ The process could issue an I/O request and then placed in I/O queue

→ The process ~~could~~ could create a new child and wait for child termination

→ It could be removed forcibly from CPU as a result of an interrupt and then put back in the ready queue ~~block~~

Scheduler

3 types

1. long-term scheduler / job scheduler
2. Short-term scheduler / CPU scheduler
3. medium term scheduler

Process

Generally 2 types

→ I/O-bound process

→ CPU-bound process

I/O-bound process:

An I/O-bound process is one that spends more of its time doing I/O than it spends doing computation.

(I/O time > CPU time)

CPU-bound process:

A CPU-bound process, in contrast, (তুলনামূলকভাবে) generates I/O requests infrequently, using ~~using~~ more of its time doing computations.
(CPU time > I/O time)

~~Function~~

Scheduler name	Function
Long term scheduler/ Job scheduler	→ select process from job pool (Secondary memory) → load them into main memory for execution → execute much less frequently → control the degree of multiprogramming. → it can afford to take more time to decide which process should be selected for execution → controls the degree of multiprogramming

context switching

Scheduler name	function
	→ It select a good process mix of I/O bound and CPU-bound process
Short term scheduler → also called dispatcher	→ select from among the processes that are ready to execute. → allocate the CPU to one of them → select a new process for the CPU frequently → execute for only a few milliseconds before waiting for an I/O request → increases the system performance

Degree of multiprogramming:

Number of processes that can be present in the ready state at maximum is called degree of multiprogramming.

CamScanner

Concept of medium term schedular/Swapping:

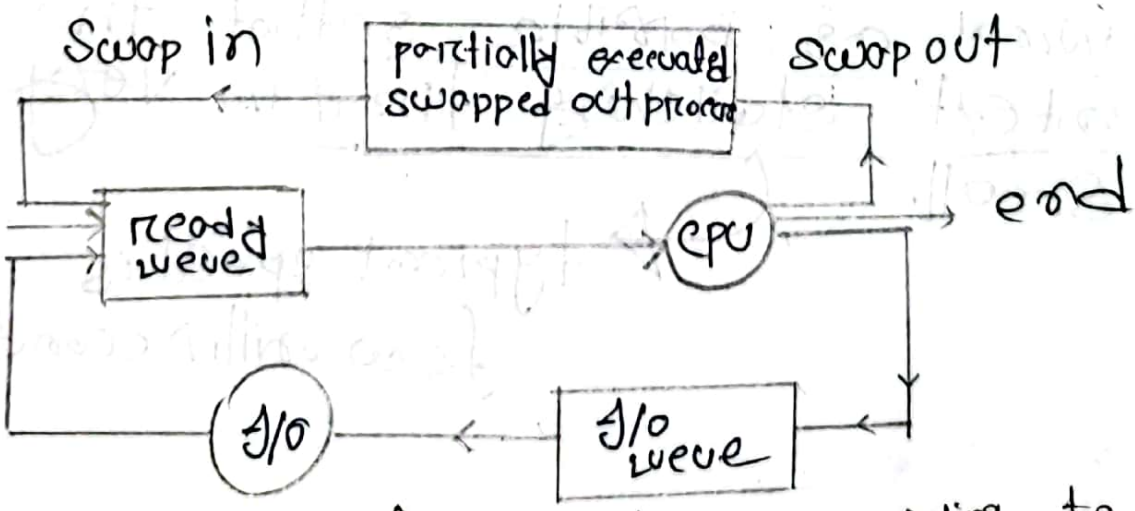
Sometimes it can be advantageous to remove a process from memory and thus reduce the degree of multiprogramming.

Later the process can be reintroduced into memory and its execution can be continued where it left off.

This scheme is called swapping.

→ necessary to improve process mix

→ a change in memory requirements.



The queuing of medium term scheduling to the queuing diagram

Context switching:

Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process.
This task is known as context switching.

When a context switch occurs the kernel saves the context of the old process in its PCB and loads the saved context of the new process schedule to run.

Size of the context should be as ~~small~~ minimal as possible so that the context switching time is very small.

→ typical speed is few millisecond.

Steps of context switch

- → interrupt the running process or thread
- → Store the context in PCB
- → run OS scheduler to decide what to do next
- → flash cash if required
- → load the context from PCB to the process
- → begin executing the new process or thread.

CamScanner

Medium term scheduler:

→ is a part of swapping

→ So it is also called process swapping scheduler.

→ swapping from main memory to disk or vice versa

main memory ⇌ disk

→ works for suspending and resuming the process

In state diagram queue

Secondary memory contain

→ Suspend wait
→ Suspend ready
→ new

} State

Main memory / primary memory contain

→ ready
~~remaining~~
→ running
→ waiting
→ terminate

} State

CS CamScanner

CPU switching from process to process:

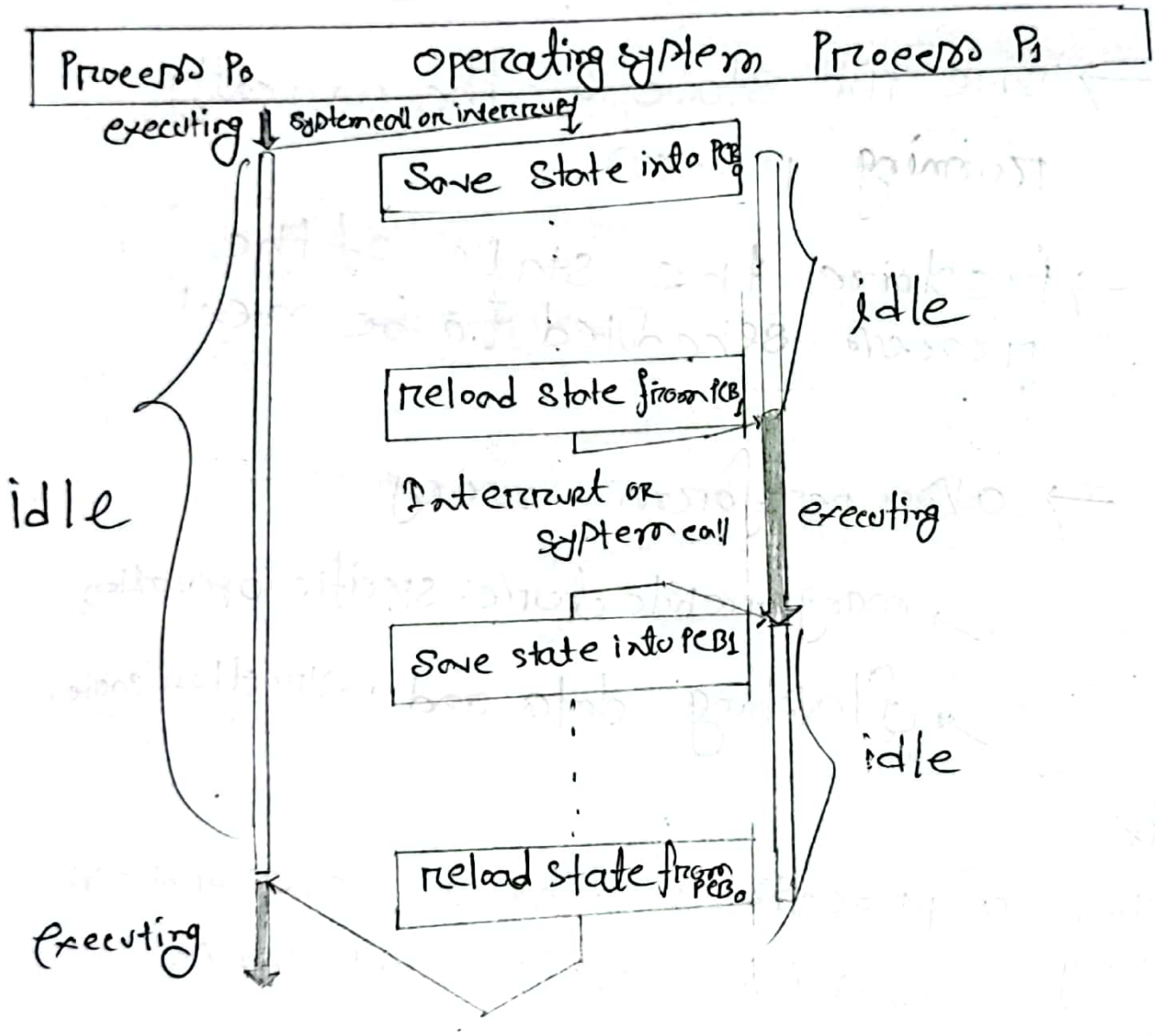


Fig: CPU switch from process to process

Kernel action to context switch:

- → Save the state of the currently running process
- → restore the state of the process scheduled to be next

→ also perform ~~many~~

- → many architecture-specific operation,
- → flushing data and instruction caches

When a process creates a new process using fork() operation following is happened

→ Shared memory segments are shared between the parent process & newly forked process.

→ Copies of the stack and heap are made for the newly created process.

CamScanner

Process creation

During execution a process may create several new processes.

→ Creating process is called parent process

→ new " is " children "

→ in this way there form a tree of process

→ Child process need certain resources → cpu time
→ memory file
→ files
→ I/O devices

Resource Sharing:

→ parent and children share all resources

→ Child share subset of the parent resources

possibilities for execution

When a process creates a new process, 2 possibilities for execution exist

→ The parent continuous to execute concurrently with its children

→ The parent waits until some or all

of its children have terminated.

2 Address space possibilities for new process

→ child process is a duplicate of the new process

2 possibilities for address space

→ Child process have a new program loaded into it.

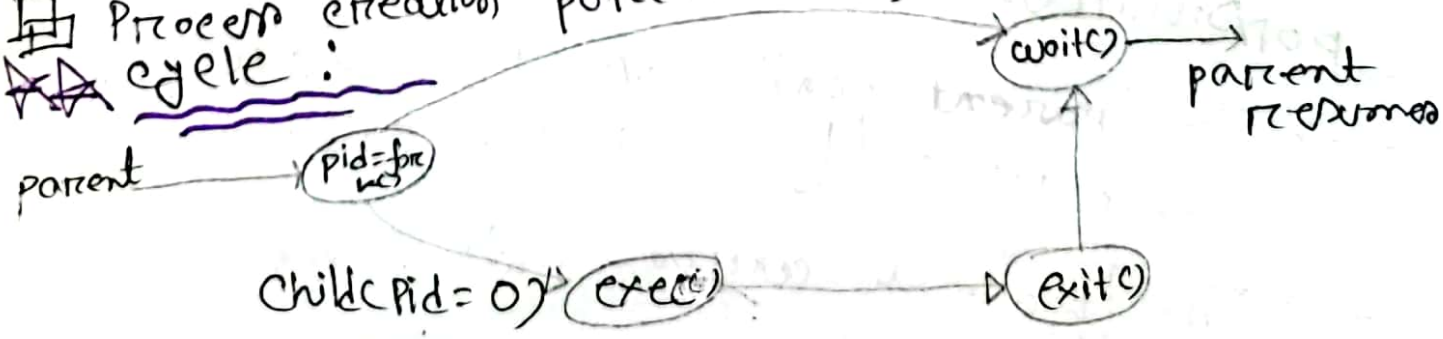
Pid :

In most operating systems identify the processes according to a unique process identifier, which is an integer number, called pid or process id.

→ for root process pid = 1.

→ for parent (pid > 0)

Process creation cycle :



Process creation in UNIX operating system:

⇒ **fork()**:

A new process is created by using fork() system call

⇒ Pid = 0 for child process (new process)

⇒ pid > 0 for parent process

⇒ **exec()**:

It replaces the process's memory space with a new program

⇒ **exec()**:

→ loads a binary file into memory

⇒ parent wait → When to execute the children

→ or nothing else to do

wait():

parent move itself into ready queue until the termination of the child.

CamScanner

execvp( ): replaces the calling process image with new process image.

getpid( ): to know the id of process

~~exit( ): to terminate~~

Process termination

→ exit(1): OS delete process by using the exit( ) system call

→ all the resource of the process are deallocated (physical & virtual memory, open files, I/O buffers)

Cascading termination:

* If a process terminates, all its children must also terminate

Chapter-1
Introduction

☆ Operating System:

is a system software that Operating System control the hardware and co-ordinates its use among the various applications or programs for the various users.

An operating system is similar to government. Like a government it performs no useful function by itself. It simply provides an environment within which other program can do useful work.

Computer system (4 components)

- hardware
- operating system
- application programs
- users

CamScanner

Modes of Computer:

Most computers have 2 modes of operation

→ Kernel mode/supervisor mode

→ User mode.

Kernel mode:

Operating system is most fundamental piece of software and runs in kernel mode (also called supervisor mode).

In this mode, it has complete access to all the hardware and can execute any instruction the machine is capable of executing

It has

→ complete access to all the hardware

→ can execute any instruction the machine is capable of executing

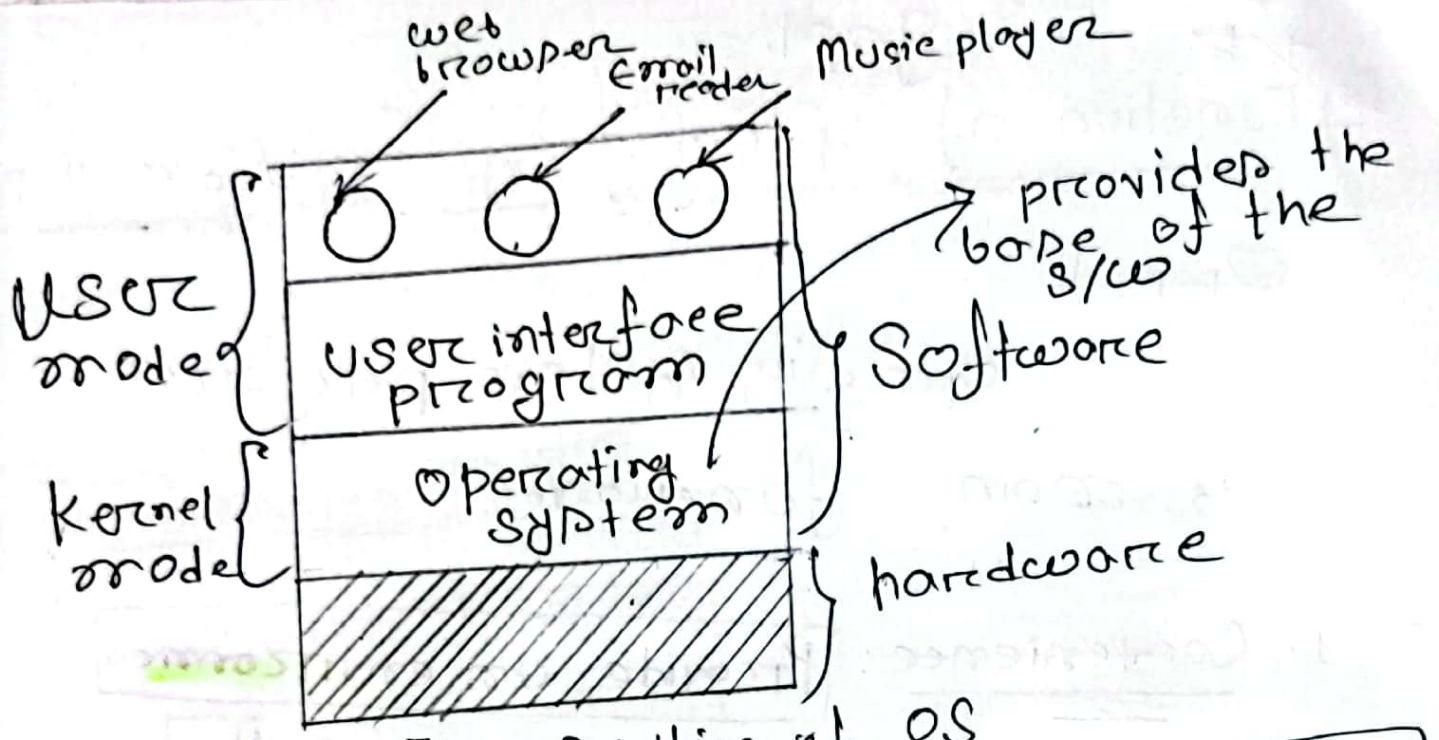


Fig: Position of OS

ser mode

→ only a subset of machine instruction is available

→ those instruction that affects control of the machine or do I/O are forbidden to user mode program.

verything running in the kernel mode is clearly part of the operating system. but some programs running outside it are arguably also part of it. or atleast closely associated with it.

☆☆ OS goal.

Function of operating system: / 3 main purpose

~~Operati~~

Operating system perform 3 main functions:

1. ~~Convenience~~: Provide an environment for a computer user to execute programs on computer hardware in a convenient and efficient manner. ☆

2. Allocate the seperate resource of computer as needed to solve the problem given. Allocation should be as efficient as possible

3. An operating system as a control program.
   → manages the execution of user program to prevent errors and improper use of the computer

→ management of the operation and control of I/O devices.

Types of operating System:

→ Batch System.
→ Multiprogramming
→ Time sharing system.
→ Multiprocessor System
→ Distributed operating system.
→ Network Operating System.
→ [illegible] system

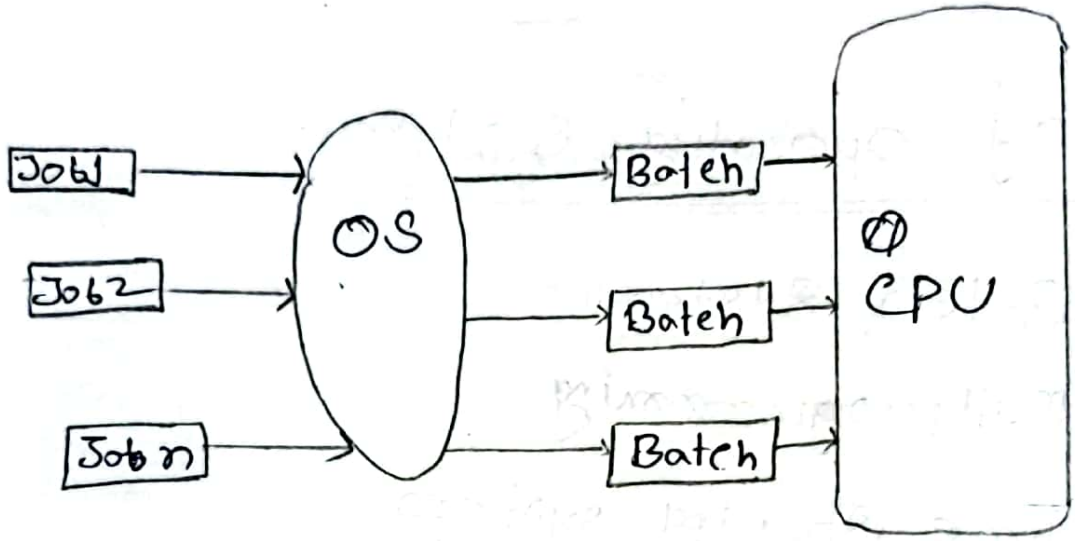
OS provide service such as

→ protection
→ Scheduling
→ Memory management
→ File system
→ Synchronization
→ virtualization
→ inter process communication
→ User Interface

stem don't directly with the computer

on operator which similar jobs having [illegible]ire ment and group batches. It is responsi[illegible] operator to sort

the jobs with similar needs.



Advantages:

→ Multiple user can share the batch system

→ The idle time of batch system is very less

→ It is easy to manage large work repeatedly in batch system

Disadvantages:

→ The computer operator should be well know with batch system

→ Batch system are hard to debug

→ It is sometimes costly

→ The other Jobs will have to wait for an unknown time if any job fails.

Example: payroll system, Bank Statement

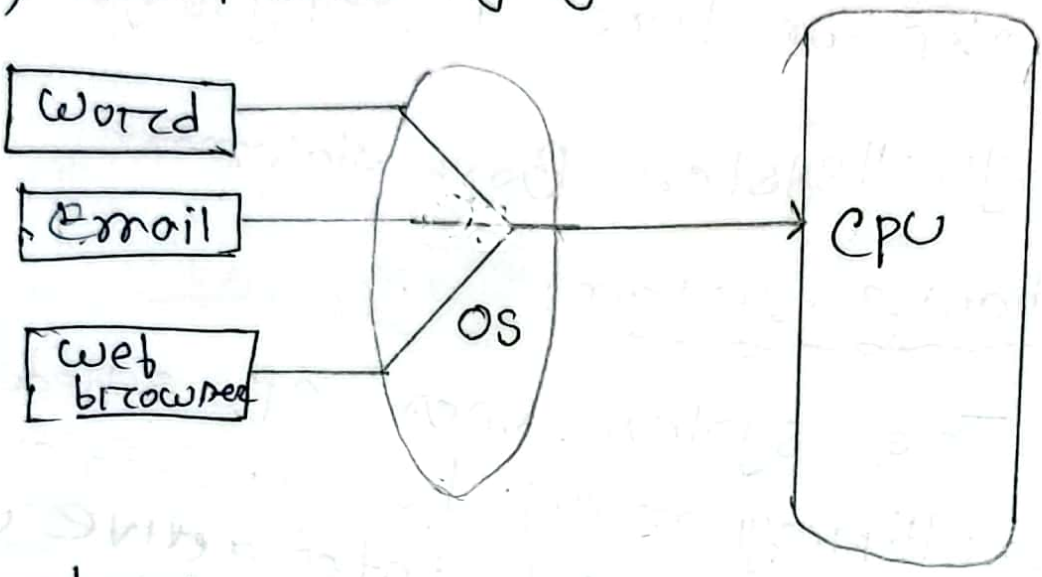
Time Sharing System:

This system uses CPU scheduling and multiprogramming to provide ~~ehonomi~~ economical interactive use of a system. The CPU switches rapidly to one user to another. Instead of having a job defined by card images, each program reads its next control card from terminal and output is normally printed immediately to the screen. Each task has given some time to execute, so that all the tasks work smoothly. The time that each ~~time~~ task ~~to~~ gets to execute is called quantum. After this time

interval is over OS switches over to next task. Also known as Multitasking system.

→ Execution of multiprogramming system

→ virtual memory system control its time.



Advantages:

→ Each task gets an equal opportunity

→ Less chances of duplication of S/W

→ cpu idle time can be reduced.

Disadvantages:

→ Reliability problem

→ Data communication problem

→ One must have to take care of security

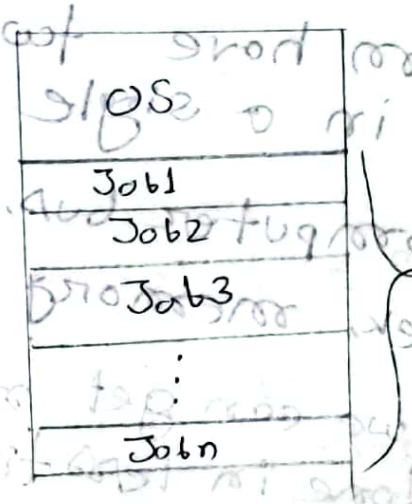
Example: MULTICS, CTSS,

ultiprogramming System

→ Sharing the processor, when two or more programs reside in memory at a time

→ It increases CPU scheduling by organizing job.

→ keeps ~~several~~ several job in memory



→ set of jobs is a subset of jobs kept in the job pool

→ monitor the state of all active programs.

ontages

→ high and efficient CPU utilization

→ It's idle time is less than batch system.

Did advantages:

→ cpu scheduling is required

→ To accommodate many jobs in memory, memory management is required.

Multiprocessor System:

→ such system have two or more processors in a single CPU.

→ Shares computer bus, sometimes, the clock, memory and peripheral device.

Advantages:

3 main advantages.

1) Increased throughput → we can get more work done in less time.

2) Economy of scale → If functions can properly be distributed among several processors

3) Increased reliability → failure of one processor will not halt the system, only slow it down

→ It can cost less than equivalent multiple single processor, because the peripheral mem. sto

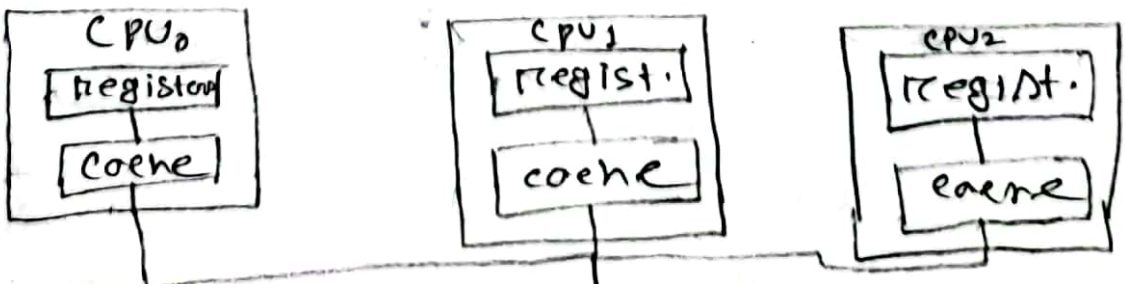
Classification:

2 types

→ Asymmetric multiprocessing

→ Symmetric "

Asymmetric	Symmetric.
→ each processor is assigned a specific task.	→ each processor performs all tasks within the operating system
→ A boss processor control the system → it define boss-worker relationship	→ no-boss-worker relationship
	→ each processor has its own set of registers, local cache → all processors share physical memory



Distributed System :

A distributed system is a collection of physically separate possibly heterogeneous computer systems that are networked to provide user with access to the various resources. Independent system process their own memory unit and CPU. These are referred as loosely coupled system or distributed system.

CamScanner

The major benefit of working with these types of ~~os~~ operating system is that, it is always possible that one user can access the file or s/w which are not actually present on his system but on some other system connected within this network. i.e. remote access is possible.

Advantages:

→ Load on host computer reduces

→ computation is highly fast and durable.

→ delay in data processing ~~de~~ reduces.

→ failure of one will not affect the other network as all systems are independent from each other

→ Electronic mail increases the data exchanges speed.

CamScanner

Disadvantages:

→ Failure of the main network will stop the entire communication

→ systems are not readily available as they are very expensive

Example: LOCUS

Network operating System:

System that provides features such as file sharing across the network along with communication scheme that allows different processes on different computers to execute messages.

A computer running a network operating system act autonomously from all other computers on the network. A distributed operating

System provide less autonomous environment.

Example:

windows server 2003,
windows server 2008,
UNIX, Linux, Mac, OSX, BED etc.

Client - Server system:

Real time system ★★

This systems are characterized by having time as a key parameter.

Real time systems are used when there are time requirement is very strict like missile system, air traffic control system, robots etc.

Real time systems are two types
→ Hard real time system
→ Soft real time system

CamScanner

Hard real time system:

If the action absolutely must occur at a certain moment (or within certain range) we have a hard real time system.

→ Time constraints are very strict.

→ Even the shortest ~~possible~~ possible delay is not acceptable.

Example: industrial process control, military, avionics (electronic system used in aircraft) (অভিজ্ঞতা)

Soft real time system:

This OSs are for application where time constraint is less strict. system in which missing an occasional deadline, while not desirable is ~~acct~~ acceptable and does not cause any permanent damage.

Example: Digital audio or multimedia system, Digital telephone.

CamScanner

System call :

A system call is a programmatic way in which a computer program request a service from the kernel of the operating system it is executed on.

Mechanism of read system call:

Any ~~cpu~~ ~~signal~~ single cpu computer can execute only one instruction at a time. If a process is running a user program in user mode and needs a system service, such as reading a data from file it has to execute trap instruction to transfer control to the operating system. Then it carries out the system call and returns control to the instruction following the system call.

CamScanner

System call of a read data from a file file is performed in a series of steps.

read (fd, buffer, nbytes)

- **Step-1-3** the calling program first pushes the parameters onto the stack.

Step-4: perform the actual call to the library procedure (written in assembly language)

Step 5: puts the system call number in a place where the operating system expect it such as register

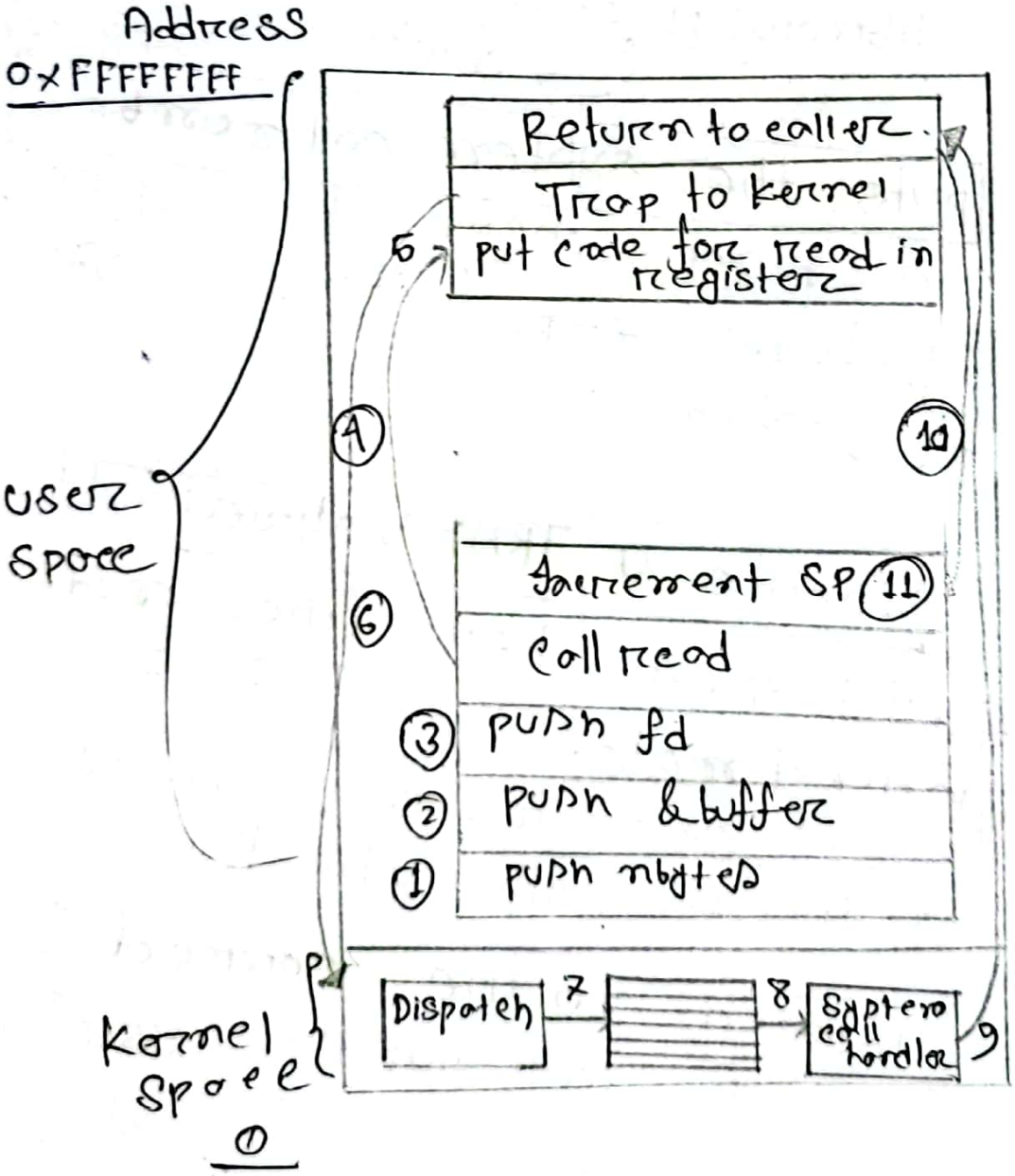
Step-6: Execute a TRAP instruction to switch from ~~ker~~ user mode to kernel mode.
Start execution at a fixed address within the kernel

Step7: dispatch to the correct system call handler usually via a table of system call handler

CamScanner

Step 8: System call handler runs

Step 9: Once system call handler has completed its work, control may be returned to the user space-library procedure



Step-10 : This proceedure then returns to the user program in a usual way proceedure.

Step-11 : user program has to clean up the stack.

TRAP instruction:

It is actually fairly similar to the proceedure call instruction, in the sense that instruction following it is taken from a distinct location and return address is saved on the stack for use later.

CamScanner

Trap instruction Vs Procedure call

trap instruction	procedure call
1. It switches into Kernel mode	1. The procedure call instruction does not change the mode
2. Rather than giving a relative or absolute address where the procedure is located, the TRAP instruction can't jump to a single fixed location	2. Depending on the architecture it either jump to a single fixed location.

Dual Mode Operation:

There are two modes of operation:

1. user mode
2. Kernel mode (Supervisor mode, system mode, privileged mode).

To seperate mode of operation a bit called mode bit is added to the hardware of the computer. It helps to identify current mode of operation (1 for user mode and 0 for kernel mode).

~~When~~ User mode: When the user application requests for a service from the operating system ~~are an interrupt~~ via system call. the system must transition from user to Kernel to fulfill the request.

CamScanner

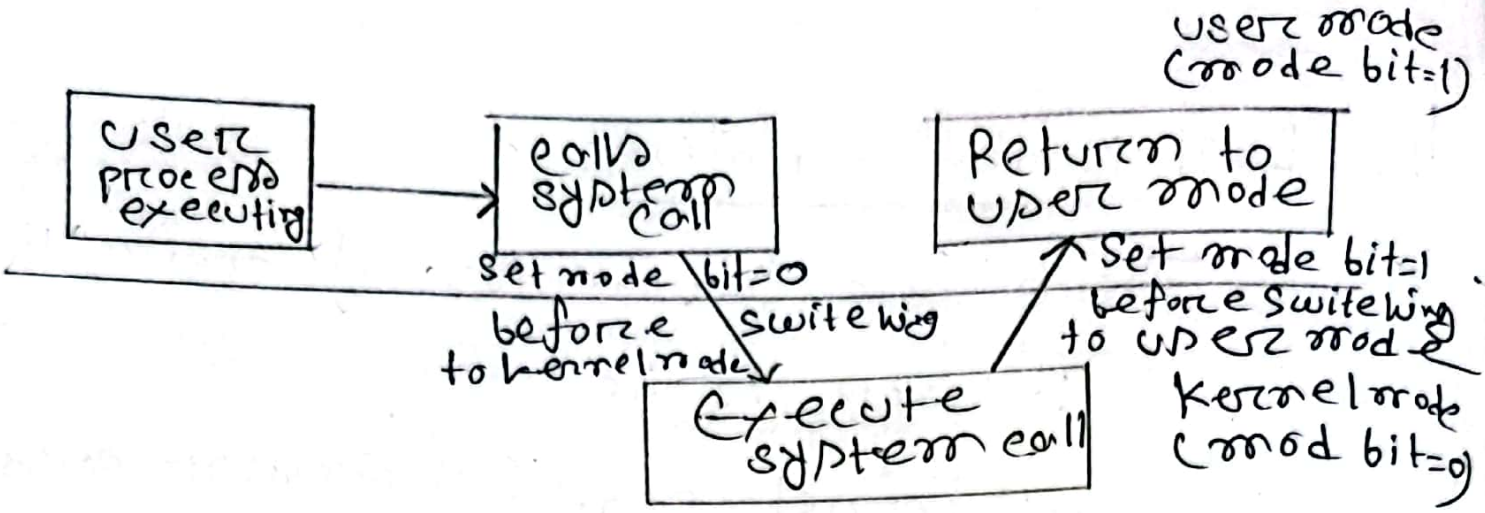


Fig: Transition from user mode to kernel mode.

Kernel mode:

When system boots than hardware starts in kernel mode and when operating system is loaded then it start application in user mode. To protect the hardware, we have privileged instructions which executes only in kernel mode.

Whenever a ~~trap~~ user attempt to run priviliged instruction in user mode then it will treat instruction as illegal and trap to OS. Some trap instructions are

→ Handeling interrupt

→ Input output management

→ Change frequency of clock

→ To switch from user mode to kernel mode.

→ Reset processor

→ Halt a processor

→ invalid TLB entries.

→ memory address mapping

→ perform I/O operation

Structure of / Operating System:

★★

Kernel: Kernel is the core part of an operating system which manages system resources. It also act like a bridge between application and hardware of the computer.

→ Contain previliged instruction as well as other instructions.

provide core features

→ process management

→ Memory management

→ Device management

I/O communication ⟷ Interrupt handling

Structure of operating System

→ Microkernel.

→ Monolithic system.

→ Layered system.

→ Virtual Machine Based.

→ client-server system

CamScanner

→ exokernel

→ peer-to peer

→ Hybrid

→ distributed.

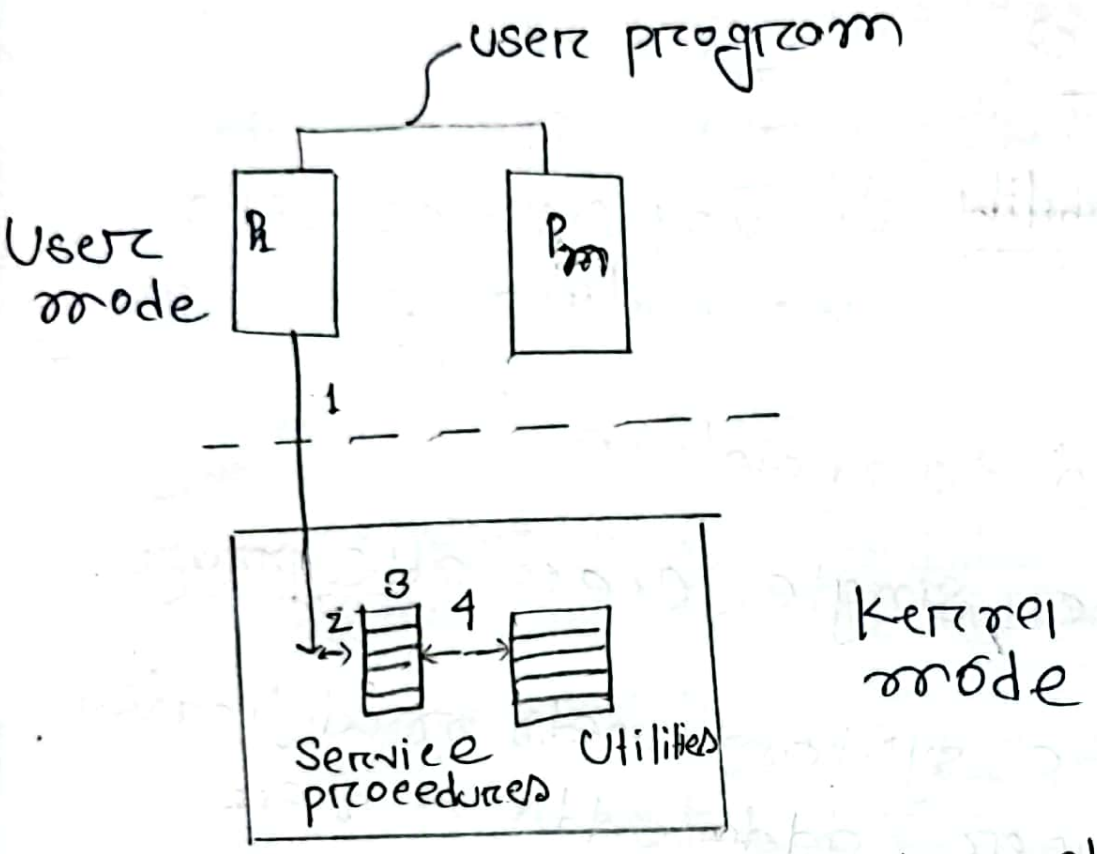
Monolithic:

→ The entire operating system runs as a single program in kernel mode.

→ The operating system is written as a collection of procedures, linked together into a single large executable binary program.

⇒ Each procedure in the operating system is free to call any other one. Hence thousand of procedure can call each other without restriction

CS CamScanner



1. System call (User → Kernel Mode)
2. ~~Cho~~. Check parameter
3. Call service routine
4. ~~Je~~ Service routine call utilities

Return to the user.

Advantages :

→ Shorted kernel space
→ Good performance
→ Easy to implement
→ Low overheads

Disadvantages:

→ Instability: If any service fails the entire system crashes

→ hard to maintain

→ One single executable program

→ entire system needs modification if user ~~adds~~ adds a new service.

→ large OS size

Example: BSD, Unix, windows

Layered System:

1. Components are divided into layers
   → grouping similar components
2. Interaction among layers
   → The outer layer → request service
   → The inner layer → provide service

Example: THE, MULTICS

CamScanner

THE system:

It consists of 6 layers

Layer	Function
5	The operator
4	User program
3	Input/Output management
2	Operator-process communication
1	Memory & drum management
0	Processor allocation and multiprogramming

MULTICS system (5 layers)

⇒ Described as having a series of concentric ring. (having common center)

⇒ Inner one is more privileged than outer ones.

Hardware
Kernel
Libraries
Shell
Application

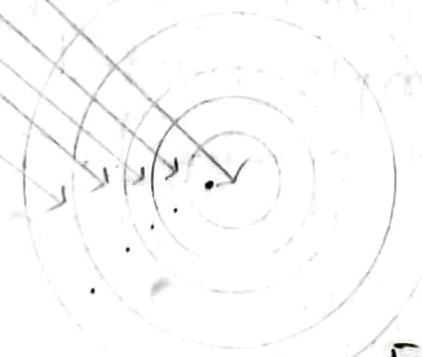


Fig: MULTICS

Advantages:

→ Layered ~~system~~ Abstraction

Disadvantages:

→ All the layers are linked together into a single executable program

Microkernel (N-Kernel)

☆ The basic idea behind the microkernel design is to achieve high reliability by spliting the operating system up into small, well-defined modules.

☆ Only the microkernel runs in kernel mode and the rest ~~runs~~ run as a relatively powerless ordinary user processes

☆ User service and kernel service are implemented in different address space. Thus also ~~redu~~ reduces the size of kernel and size of operating system as well

Example: MINIX 3,
K42,
L4,
ONX,
PikeOS
Symbian

Structure of MINIX3:

It is only about 3200 lines of C and 800 lines of assembler for very low level function.
→ catching interrupt
→ switching process.

User mode:
3 layers
→ device driver → lowest layer
→ servers
→ user process

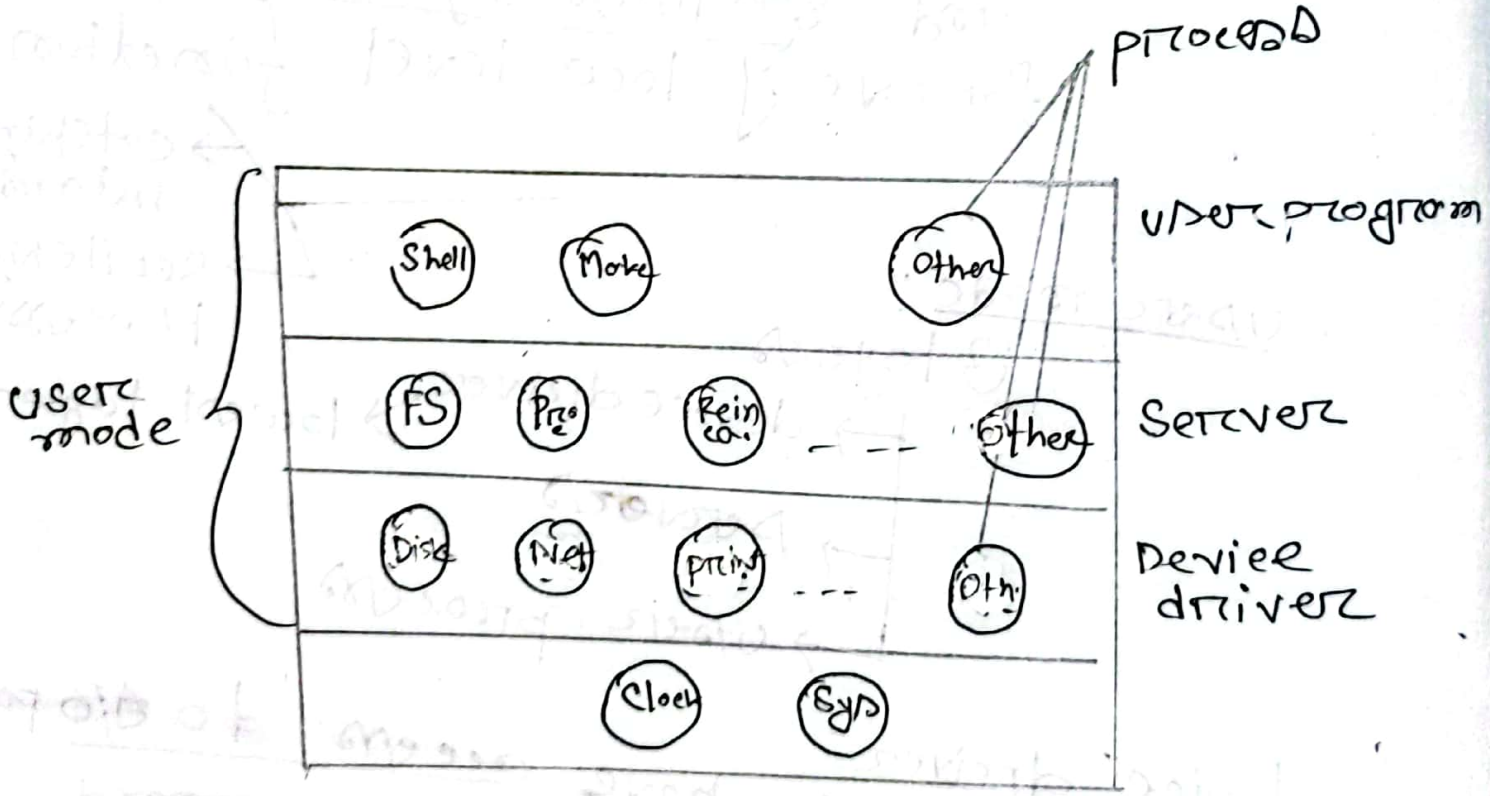
device driver:
1. Don't have access to I/O port
2. Can't issue I/O command directly.
3. Only packet the value to be written at a port.

Servers

1. File server → manages file system
2. Process server → create, destroys and manages process
3. Reincarnation server → do most of the work of the operating system

রিইনকারনেশন

→ check if other server and driver are functioning correctly



ernel mode:

~~Clock~~ 1. System task

→ catch interrupt

→ Switch process

→ Scheduler process

→ Manage inter process communication

→ provide other 35 system call service

2. Clock task:

Contain driver for processor clock in order to assist scheduler

Distributed server:

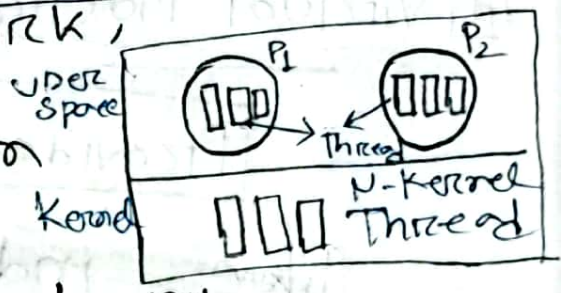
1. The Bullet server
   → for file storage
2. The directory server
   → for file naming
   → Map from name to capabilities
3. The Replication server
   → for fault tolerance and performance
4. The Run server
   → manages the processor pool
5. Boot server
   → Ensure that ~~ser~~ servers are up and running.
   → discovered that server have crashed

CamScanner

Structure of Amoeba OS

It is ~~micro~~ N-kernel based operating system.

It offers multithreaded program and a remote procedure call (RPC) mechanism for communication between threads, potentiality across the network,



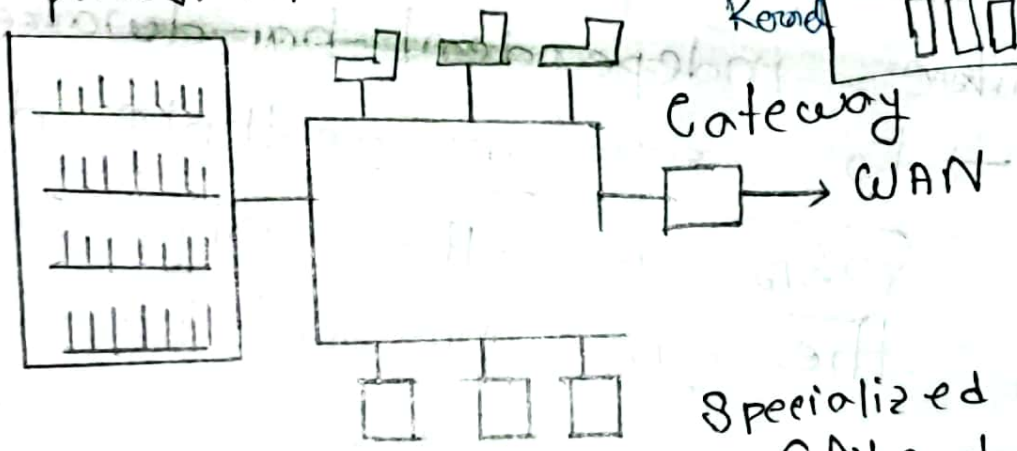


Fig: Amoeba

It has four basic components

1. Workstation → each user has a workstation running windowsX or UNIX emulation
2. Processors pool → ~~which are~~ dynamically allocated to users as required
3. Specialized servers → file, directory, database etc.

4. Gateway: Components are connected to each other by fast LAN and to the WAN by a gate way

Virtual Machine

Provide an interface that looks like independent hardware.

→ to ~~mu~~ run multiple different OSs simultaneously on the same physical hardware.

→ Each OS believes that it has access to and control over own CPU, RAM, I/O, devices, etc.

Example: VM/370, Java VM, VMW, IBM, OS/360

CS CamScanner

VM/370 Structure

~~Based on an astute observation:~~ (তীক্ষ্ণ দৃষ্টি)

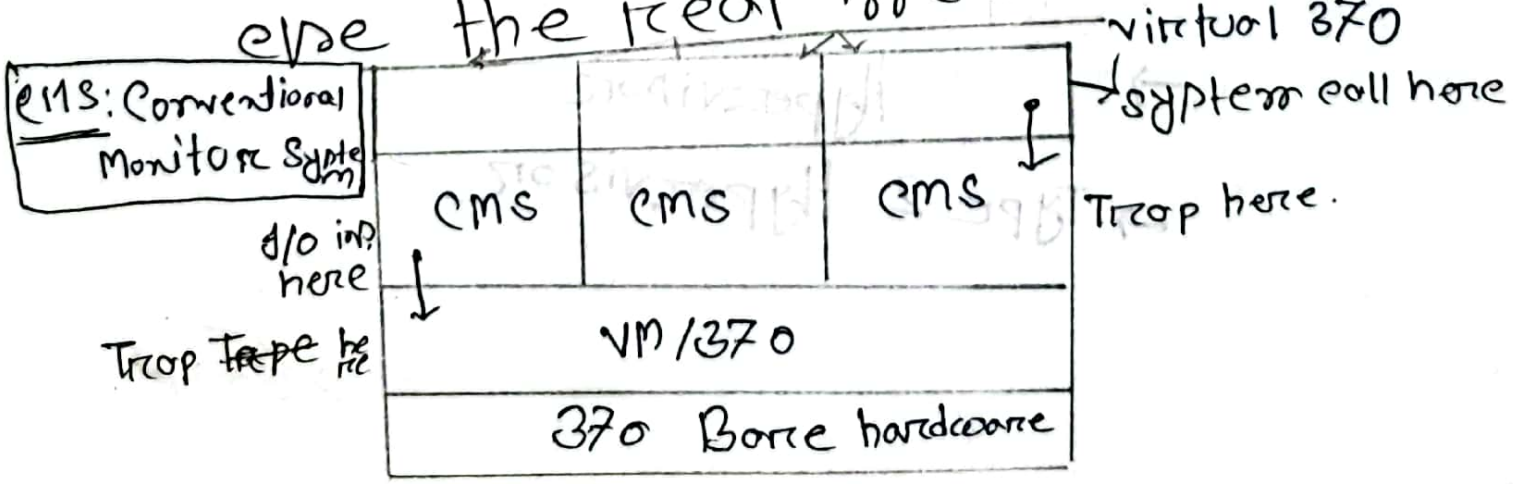
Based on an astute observation:

a time sharing system provides
→ multiprogramming
→ extended machine

Heart of the system known as the virtual machine monitor run on the bare hardware. ~~and does~~

Upper layer has several virtual machines

This virtual machines are not extended machines with files and other nice features. They include kernel/user mode, I/O, interrupts and everything else the real machine has.



OS need to protect usage of

→ **CPU**: use of cpu by any user process.

→ **I/O**: prevent user process from ~~from~~ from access illegal I/O or perform illegal I/O command

→ **Memory**:

- prevent any user process from accessing another's data
- prevent any user process from access and manipulate OS's code and data structure.

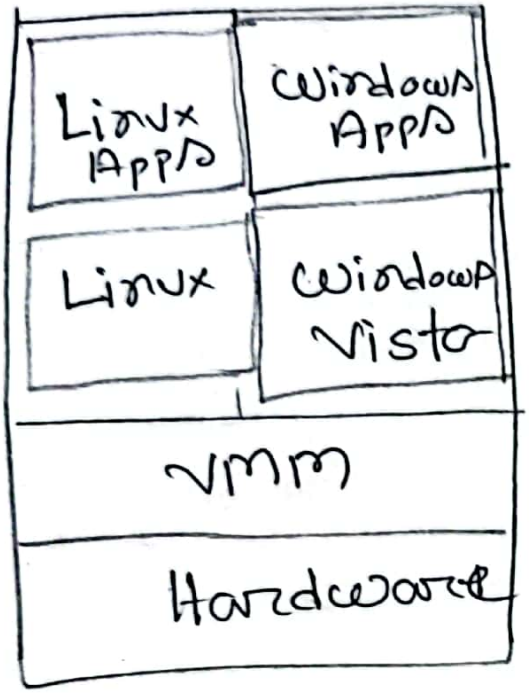
Hypervisor

2 types

→ Type 1 Hypervisor

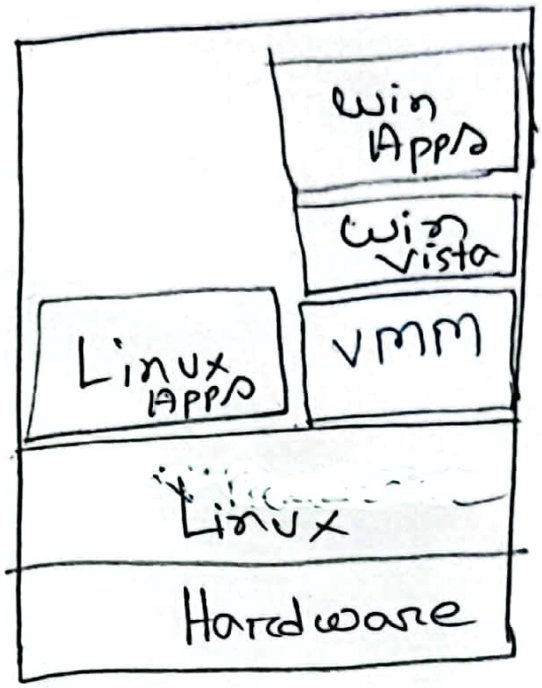
→ Type 2 Hypervisor

CamScanner



Type-1 Hypervisor

VMM run on hardware



Type-2 Hypervisor

VMM as an application

Chapter- 6
CPU Scheduling

CPU Scheduling:

It is the task of selecting a waiting process from the ready queue and allocating the CPU to it.

CPU is allocated to the selected process by the dispatcher.

Who: Short term scheduler / CPU scheduler

Ready queue can be implemented as

- FIFO
- a priority queue
- a tree
- an unordered ~~lined~~ linked list

When?

CPU scheduling can take place 4 circumstance/situations

process switches from

1. Running state → Ready state (when interrupt occurs)
2. Running state → waiting state (→ for I/O request; → " invocation of wait)
3. waiting state → ready state (→ at completation of I/O)
4. termination

Dispatcher:

model / program that gives control of the CPU to the process selected by the short term scheduler.

Function of dispatcher:

→ switching context

→ switching to user mode

→ jumping to the proper location

of in the user program to ...
that program.

☆☆

Dispatch latency:

Time it takes for the dispatcher to stop one process and start another running

Functional component of scheduler:

3 component

→ Enqueuer → adds ~~off~~ pointer reference to the PCB as per the desired data structure or ready queue

→ Dispatcher → implements the scheduling algorithm to pick the next process to run

→ Context switcher → load the selected process into the CPU

CamScanner

Criteria of CPU scheduling algorithm

CPU utilization:

→ to keep the CPU as busy as possible

→ it range from 0 to 100 percent

→ in real system, it should range from 40 to 90 percent

Throughput: (Process/time)

Number of processes that are ~~computed~~ completed per time unit is called throughput.

Turnaround time

How long it takes to execute that process.

Interval from the submission ~~time~~ of a process to the time of completion.

TAT = CT − AT

TAT = BT + CWT

time:

It is sum of the periods spent waiting in the ready queue

$$WT = TAT - BT$$

Response time:

Time from the submission of a request until the first response is produced.

Waiting time:

Time in which a process is ready to be executed

Burst time:

CPU time of a process.

We want to ~~minimize~~

maximize the
- CPU utilization
- throughput

minimize the
- waiting time
- response time
- turnaround time

Completation: time:

Time at which the process finishes.

Scheduling algorithm:-

2 types

→ preemptive → running process can be interrupted when a high priority process enter the queue and is allocated to CPU

→ non-preemptive → running process can't be interrupted by any other process.

Example:

→ FCFS

→ SJF

→ ~~Non~~ preemptive SJF / SRTS → Shortest Remaining time Scheduling

→ priority scheduling

→ Round robin

→ ~~preemptive~~ "

→ Preemptive priority scheduling

→ multilevel queue

→ " feedback " scheduling

CamScanner

Consider the following table

Process no	AT	BT	Priority
P_1	0	18	1
P_2	10	10	0
P_3	6	5	2
P_4	8	25	3

0 is highest priority

1 Gantt chart for FCFS

P_1	P_3	P_4	P_2

0 18 23 ~~31~~ 48 ~~41~~ 58

P.No	AT	BT	TAT	WT	CT
P_1	0	18	18	0	18
P_2	10	10	48	38	58
P_3	6	5	17	12	23
P_4	8	25	40	15	48

∴ Average turnaround time $\left(\frac{18+48+17+40}{4}\right) = 20.75$

Average waiting time $\frac{0+38+12+15}{4}$

$= 16.25$ Ans

For SJF (non-preemptive)

P_No	AT	BT	CT	TAT	WT
P_1	0	18	18	18	0
P_2	10	10	33	23	13
P_3	6	5	23	17	12
P_4	8	25	58	50	25

Gantt chart

P_1	P_3	P_2	P_4

0 18 23 33 58

average turnaround time $\left(\frac{18+23+17+50}{4}\right)$

$= 27$

average waiting time $= \left(\frac{0+13+12+25}{4}\right)$

$= 12.5$

For SJF (Preemptive)

P-No	AT	BT	CT	TAT	WT
P_1	0	18	33	33	15
P_2	10	10	21	11	1
P_3	6	5	11	5	0
P_4	8	25	58	50	25

Gantt chart:

P_1	P_3	P_2	P_1	P_4

0 6 11 21 33 58

∴ Average turnaround time

$$\left(\frac{33+11+5+50}{4}\right)$$

$= 24.75$

Average waiting time $= \left(\frac{15+1+0+25}{4}\right)$

$= 10.25$ Ans

For priority scheduling

P_No	AT	BT	Priority	CT	TAT	WT
P_1	0	18	1	18	18	0
P_2	10	10	0	28	18	8
P_3	6	5	2	33	27	22
P_4	8	25	3	58	50	25

Gantt chart

P_1	P_2	P_3	P_4

0 18 28 33 58

∴ Average turnaround time,

$$= \left(\frac{18+18+27+50}{4}\right)$$

$$= 28.25$$

Average waiting time $= \frac{08+22+25}{4}$

$$= 13.75$$

A preemptive SJF has smallest average turnaround time and

average waiting time, so preemptive SJF is better. ~~for~~

Consider the following table

TQ = 4ms

P-Id	AT	BT
P_1	0	17
P_2	1	5
P_3	2	4
P_4	3	3
P_5	4	7

For preemptive SJF

P-Id	AT	BT	CT	TAT	WT
P_1	0	17 16	36	36	19
P_2	1	5 1	6	5	0
P_3	2	4	13	11	7
P_4	3	3	9	6	3
P_5	4	7	20	16	9

P_1	P_2	P_4	P_3	P_5	P_1

0 9 13 20 36

Average turnaround time $\left(\frac{36+5+11+6+16}{5}\right)$

$= 14.8$

Average waiting time $= \left(\frac{19+0+7+3}{5}\right)$

$= 7.6$

For RR

TQ = 4 ms

Process	AT	BT	~~TAT~~ CT	TAT	WT
P_1	0	17, 13, 9	36	36	19
P_2	1	5, 1	24	23	18
P_3	2	4	12	10	6
P_4	3	3	15	12	9
P_5	4	7, 3	27	23	~~20~~ 16

P_1	P_2	P_3	P_4	P_5	P_1	P_2	P_5	P_1	P_1	P_2

0 4 8 12 15 19 23 24 27 31 35 36

$P_1, P_2, P_3, P_4, P_5, P_1, P_2, P_5, P_1, P_1, P_1$

Ready queue

Average waiting time $= \left(\frac{12+18+9+6+20}{5}\right)$

$= 14.4$

Average turnaround time $= \frac{36+23+10+12+23}{5}$

$= 20.8$

Consider the following table

Process	AT	BT
1	0	8
2	1	4
3	2	9
4	3	5

TQ = 4

Sol^n 1 for FCFS

Process	AT	BT	CT	TAT	WT
1	0	8	8	8	0
2	1	4	12	11	7
3	2	9	21	19	10
4	3	5	26	23	18

CamScanner

P_1	P_2	P_3	P_4

0 8 12 21 26

$\therefore$ Average waiting time $= \frac{0+7+10+8}{4}$

$= 8.75$

II Preemptive SJF

Process	AT	BT	~~TAT~~ CT	TAT	WT
1	0	8,7	17	17	9
2	1	4	5	4	0
3	2	9	26	24	15
4	3	5	10	7	2

P_1	P_2	P_4	P_1	P_3

0 1 5 10 17 26

$\therefore$ Average waiting time $= \frac{9+0+15+2}{4}$

$= 6.5$

CS CamScanner

iii RR (TQ = 4)

P.no	AT	BT	CT	TAT	WT
1	0	8,4	20	20	12
2	1	4	8	7	3
3	2	25,1	26	24	15
4	3	5,1	25	22	17

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_3

0 4 8 12 16 20 24 25 26

$P_1, P_2, P_3, P_4, P_1, P_3, P_4, P_3$

Ready queue

Average waiting time $= \frac{12+3+15+17}{4}$

$= 11.75$ Ans

Characteristics of FCFS

→ It focuses on arrival time

→ nonpreemptive algorithm

→ average waiting time under the FCFS policy is quite long

→ troublesome (বিরক্তিকর) for time sharing system

→ has convey effect

→ can be both preemptive & non-preemptive

Characteristics of SJF (~~nonpreemptive~~)

→ It focuses on burst time

Advantages:

→ maximum throughput

→ minimum waiting time

Disadvantages:

→ Starvation to large jobs

→ It is not implementable because BT of processes can't be known ahed

~~Round robi~~

Characteristics of Round Robin: → preemptive

→ has a ready queue & time quantum

Disadvantages is selection of TQ

→ If time quantum is too large,
→ degenerates to FCFS
→ starvation will be happened

→ If time quantum is too small,
→ context switching time becomes excessive.

Advantages

→ less starvation that SJF & priority scheduling

→ each process gets (1/n) of the CPU time.

→ Average waiting time is often long

Convey effect: Short process, behind long process.

Or, Short process wait for long process completation, is called convey effect

Characteristic of priority scheduling

→ has Stravation problem
indefinite blocking

→ can be preemptive ~~or~~
~~and~~ or non preemptive

→ It is special case of
SJF algorithm.

A process that is ready to run but waiting for the CPU can be considered blocked or starvation

Solution of starvation:

Both Priority scheduling and SJF
~~has~~ It may suffer from
starvation problem.

Starvation:

Solution: Aging is a technique to prevent Starvation.

→ involves increasing the priority of process gradually

Aging:

Mechanism in which every lower priority process's priority has to be increased after a fixed interval of time.

Multilevel scheduling:

→ processes are easily classified into different groups

→ for different types of process different scheduling algorithm is used.

→ it partions the ready queue into separate queue.

CS CamScanner

→ each queue has own scheduling algorithm

→ It is preemptive

Example:

Processes can be 2 types

→ foreground (interactive) → use RR scheduling, given 80% of cpu time

→ background (batch) → use FCFS, 20% of cpu time

NB: If an interactive editing process entered the ready queue, while batch process was running, the bat " " would be preempted

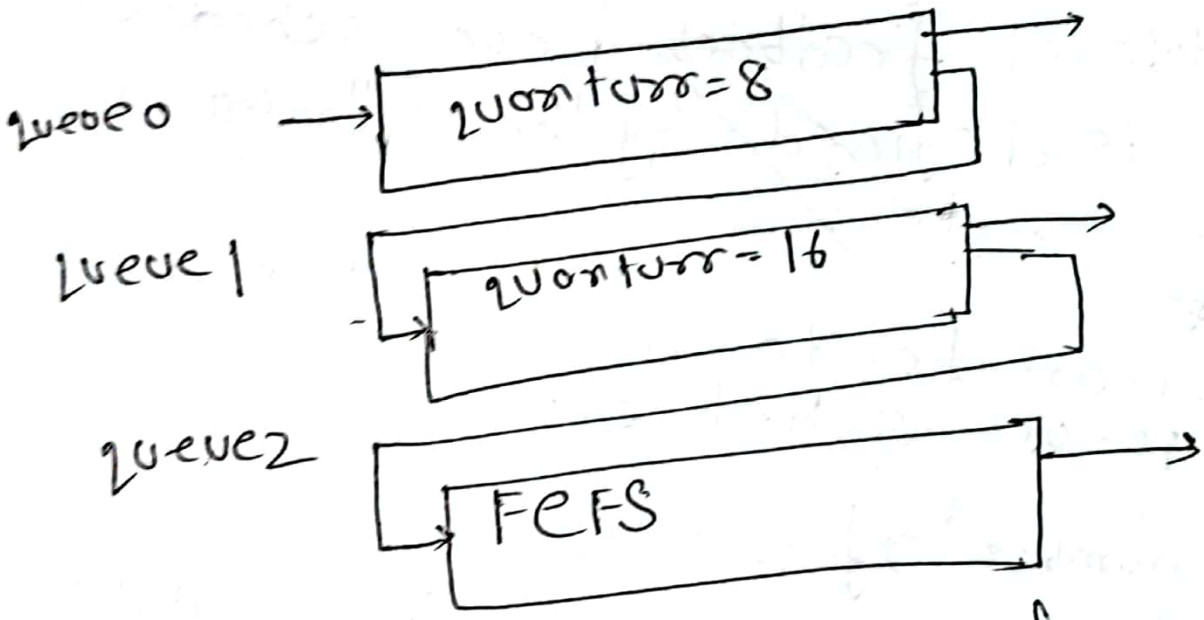
→ System
→ Interactive
→ Interactive editing
→ Batch
→ Student

Process

Multilevel feedback queue scheduling

→ Seperate processes according to the characteristics of their CPU bursts.

→ Allows a process to move from one queue to another.



→ process that arrives for queue 1 will preempt a process in queue 2

→ when ~~operation~~ higher priority queue is empty lower priority queue will be executed

→ a process having higher burst time will be moved to the lower priority queue.

→ last queue apply FCFS
→ Previouser arrive " RR

→ most complex algorithm

☐ Multilevel feedback queue scheduler is defined by the following parameter:

Parameter to define feedback queue scheduler:

→ number of queue
→ scheduling algorithm of each queue
→ when to upgrade a process to a higher priority queue
→ when to demote a process to a lower priority queue
→ used to determine which queue.

CamScanner

☆☆☆

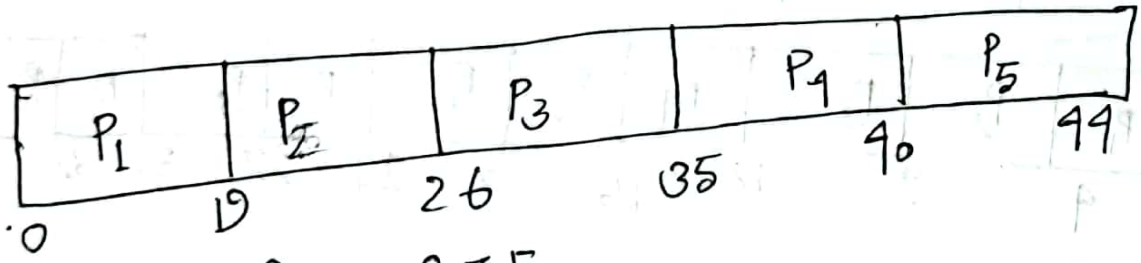
Consider the following table

NB : SJF, RR, priority তুলনা আছে

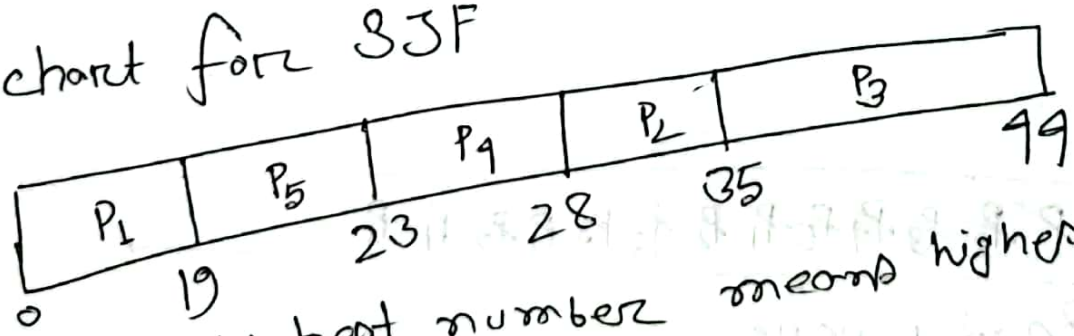
Process	BT(ms)	Priority
P_1	19	3
P_2	7	2
P_3	9	1
P_4	5	4
P_5	4	5

Sol^n : i Assume that all process at time zero

Gantt chart of FCFS



Gantt chart for SJF



- Assuming highest number means highest priority.

Gantt chart for priority scheduling

P_1	P_5	P_4	P_2	P_3

0 10 23 28 35 44

Assume lowest number means highest priority,

Now

P_1	P_3	P_2	P_4	P_5

0 10 28 35 40 44

Gantt chart for Round Robin

Time Slice = 4 ms

P_1	P_2	P_3	P_4	P_5	P_1	P_2	P_3	P_4	P_1	P_3	P_1	P_1

0 4 8 12 16 20 24 27 31 32 36 37 41 4

~~P_1~~ ~~P_2~~ ~~P_3~~ ~~P_4~~ ~~P_5~~ ~~P_1~~ ~~P_2~~ ~~P_3~~ ~~P_4~~ ~~P_1~~ ~~P_3~~ ~~P_1~~ ~~P_1~~

Ready queue

ii

Computation of turn around time & average waiting time

For FCFS

Process	BT	CT	WT	TAT
P_1	19	19	0	19
P_2	7	26	19	26
P_3	9	35	26	35
P_4	5	40	35	40
P_5	4	44	40	44

$\therefore$ Average waiting time $= \left(\frac{0+19+26+35+40}{5}\right)$

$= \frac{120}{5}$

$= 24$ Ans

Average turn around time $= \left(\frac{19+26+35+40+44}{5}\right)$

$= 164/5$

$= 32.8$ Ans

For SJF

Process	BT	CT	WT	TAT
P_1	19	19	0	19
P_2	7	35	28	35
P_3	9	44	35	44
P_4	5	28	23	28
P_5	4	23	19	23

Average waiting time $= \frac{(0+28+35+23+19)}{5}$

$= 21$ Ans

Average turnaround time $= \frac{19+35+44+28+23}{5}$

$= 29.8$ Ans

For round robin

Process	BT	CT	WT	TAT
P_1	19	44	25	44
P_2	7	27	20	27
P_3	9	37	28	37
P_4	5	32	27	32
P_5	4	20	16	20

Average waiting time $= \left(\frac{25+20+28+27+16}{5}\right)$

$= 23.2$ Ans

Average turnaround time $= \left(\frac{44+27+37+32+20}{5}\right)$

$= 32$ Ans

CS CamScanner

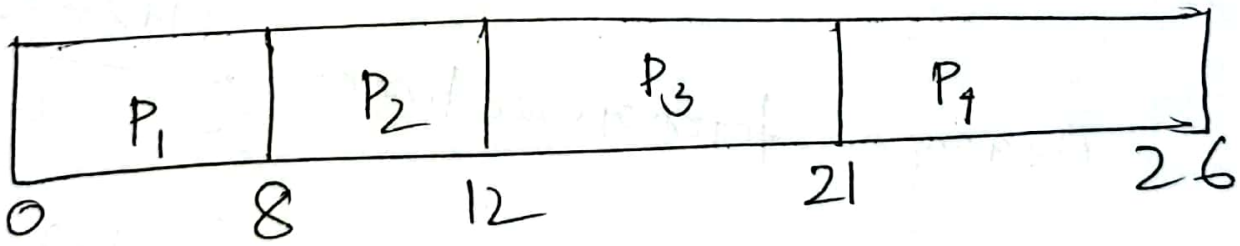
Consider the following table

Process	AT	BT
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

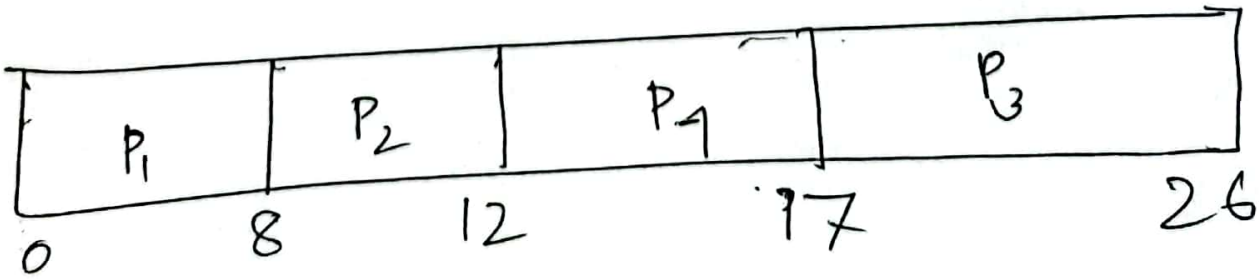
Sol^n

1

Gant chart for FCFS



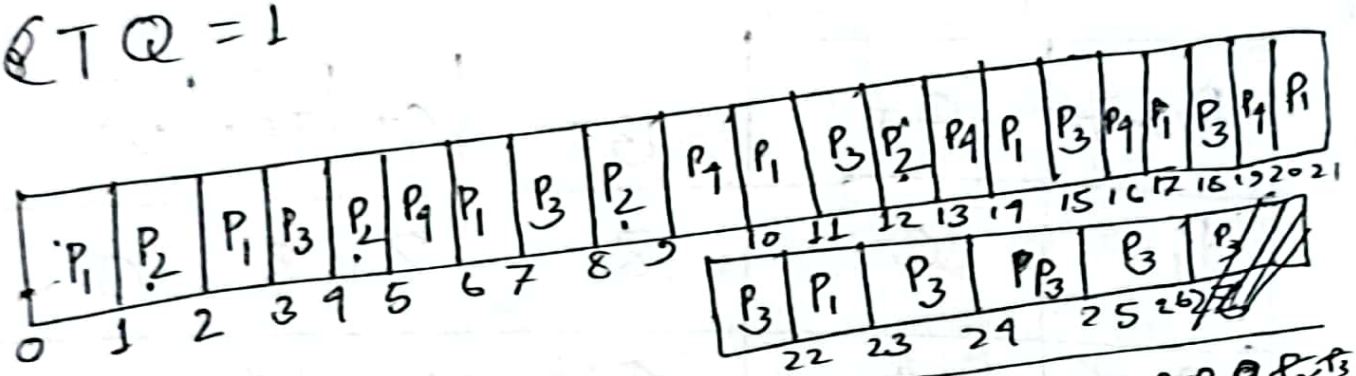
Gantt chart for SJF



CamScanner

For Round Robin

TQ = 1



~~P1 P2 P1 P3 P2 P4 P1 P3 P2 P4 P1 P3 P2 P4 P1 P3 P4 P1 P3 P4 P1 P3 P1 P3 P3 P3~~

Ready queue

For FCFS

Process	AT	BT	CT	TAT	WT
P_1	0	8	8	8	0
P_2	1	4	12	11	7
P_3	2	9	21	19	10
P_4	3	5	26	23	18

Average waiting time $= \left(\frac{0+7+10+18}{4}\right)$

$= 8.75$ Ans

" turnaround " $= \left(\frac{8+11+19+23}{4}\right)$

For SJF

Process	AT	BT	CT	TAT	WT
P_1	0	8	8	8	0
P_2	1	4	12	11	7
P_3	2	9	26	25	16
P_4	3	5	17	14	11

Average waiting time $= \left(\frac{0+7+16+11}{4}\right)$

$= 8.5$ Ans

" turnaround " $= \left(\frac{8+11+25+14}{4}\right)$

$= 14.5$ Ans

CamScanner

For RR

Process	AT	BT	CT	TAT	WT
P_1	0	8	23	23	15
P_2	1	4	13	12	8
P_3	2	9	26	24	15
P_4	3	5	20	17	12

Average waiting time $= \left(\frac{15+8+15+12}{4}\right)$

$= 12.5$

" turn around time $= \left(\frac{23+12+24+17}{4}\right)$

$= 19$

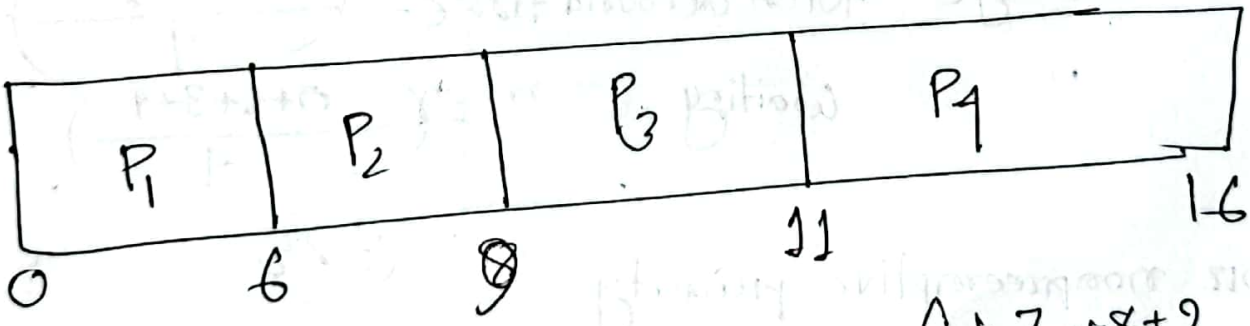
Ans III As we see from (ii) that average waiting time and average turn around time of SJF is very small. So SJF is best for this scheduling

2017-2018)

For FCFS

Process	AT	BT	TAT	CT	WT	RT
P_1	0	6	6	6	0	0
P_2	2	3	7	9	4	4
P_3	3	2	8	11	6	6
P_4	7	5	9	16	4	4

Gantt chart for FCFS



Average turnaround time $= \frac{6+7+8+9}{4}$

$= 7.5$

Average waiting time $= \frac{0+4+6+4}{4}$

$= 3.5$

For SJF

Process	AT	BT	CT	TAT	WT	RT
P_1	0	6	6	6	0	0
P_2	2	3	11	9	6	6
P_3	3	2	8	5	3	3
P_4	7	5	16	9	4	4

Gantt chart of SJF

P_1	P_3	P_2	P_4

0 6 8 11 16

∴ Average turn around time $= \left(\frac{6+9+5+9}{4}\right) = 7.2$

" waiting " $= \left(\frac{0+6+3+4}{4}\right)$

$= 3.25$

For nonpreemptive priority

Process	AT	BT	CT	TAT	WT	RT	Priority
P_1	0	6	6	6	0	0	5
P_2	2	3	9	7	4	4	3
P_3	3	2	16	13	11	11	4
P_4	7	5	~~16~~ 14	7	2	2	2

let 0 means highest priority

For SJF

Process	AT	BT	CT	TAT	WT	RT
P_1	0	6	6	6	0	0
P_2	2	3	11	9	6	6
P_3	3	2	8	5	3	3
P_4	7	5	16	9	4	4

Gantt chart of SJF

P_1	P_3	P_2	P_4

0 6 8 11 16

∴ Average turn around time $= \left(\frac{6+9+5+9}{4}\right) = 7.2$

" waiting " $= \left(\frac{0+6+3+4}{4}\right)$

$= 3.25$

For nonpreemptive priority

Process	AT	BT	CT	TAT	WT	RT	Priority
P_1	0	6	6	6	0	0	5
P_2	2	3	9	7	4	4	3
P_3	3	2	16	13	11	11	4
P_4	7	5	~~16~~ 14	7	2	2	2

let 0 means highest priority

ntt chart of nonpreemptive priority

P_1	P_2	P_4	P_3

0, 6, 9, 14, 16

∴ Average turnaround time $= \left(\frac{6+7+13+7}{4}\right)$
$= 8.25$

" waiting time $= \left(\frac{0+4+2+11}{4}\right)$
$= 4.25$

or preemptive priority
0 means highest priority

Process	AT	BT	priority	CT	TAT	WT	RT
P_1	0	6	5	16	16	10	0
P_2	2	3	3	5	3	0	0
P_3	3	2	4	7	4	2	4
P_4	7	5	2	12	5	0	0

P_1	P_2	P_2	P_3	P_4	P_1

0, 2, 3, 5, 7, 12, 16

Average waiting time $= \frac{(10+2+0+0)}{4} = 3$

" turnaround time $= \left(\frac{16+3+4+5}{4}\right) = 7$

For Round Robin with TQ

Process	AT	BT	TAT	WCT	WT	RT
P_1	0	6	11	11	5	0
P_2	2	3	4	6	1	1
P_3	3	2	5	8	3	3
P_4	7	5	9	16	4	4

Ready queue

P_1 P_2 P_3 P_1 P_4 P_4

Gantt chart

P_1	P_2	P_3	P_1	P_4	P_4

0 3 6 8 11 14 16

∴ Average waiting time $= \left(\frac{5+1+3+4}{4}\right) = 3.25$

" turnaround time $= \left(\frac{11+4+5+9}{4}\right) = 7.25$

Among all scheduling algorithms preemptive priority scheduling ~~should~~ takes small average waiting time, ~~and~~ small turnaround time and small ~~turnar~~ response time.

So I choose preemptive priority scheduling.

Chapter-4
thread

Thread:

A thread is a flow control within a process. It is a sequential execution stream within a process

→ basic unit of CPU utilization

→ process ~~is a~~ has a single thread of control

→ It comprises / includes
- thread ID
- a program counter
- a register set
- a stack

→ If a process has multiple threads of control, it can perform more than one task at a time.

→ two thread in a same process share
- ~~text~~
  - → code section
  - → data section
  - → other OS resources
  - files

Example:

A web browser might have one thread display images or text while another thread retrieves data from the network.

A word processor may have a thread for displaying graphics, another " " responding to keystrokes from the user and a third for performing spelling and grammar checking in the background.

Most operating system kernels are now multithread. Each ~~the~~ thread performs specific task.

- → managing devices
- → " memory
- → interrupt handling

Process vs thread

Process	thread
1. Program in execution	1. Lightweight process or part of it
2. Do not share memory	2. Shares memory with each other
3. Heavy weight	3. Lightweight
4. Require more time to create.	4. Require less time
5. It takes more time to context switch	5. It takes ~~mo~~ less time to context switch
6. Less efficient as compared to the process in the context of communication	6. Enhances efficiency in the context of communication
7. Require ~~less~~ more termination time	7. Require less termination time
8. Inter process communication is expensive	8. Inter process communication is comparatively less expe

Benefit of multithread:

1. Responsiveness
   - allow a program to continue even if
     - → part of it is blocked
     - → or performing a lengthy operation.
2. Resource sharing
   - → Share the memory and resource of the process.
3. Economy
   - → thread share the resources of the process to which it belong.
   - → it is economical to create and context-switch thread
4. Scalability
   - → Can be running in parallel on different processing core.
   - → Single threaded process can be run on only one processor

CamScanner

types of thread:

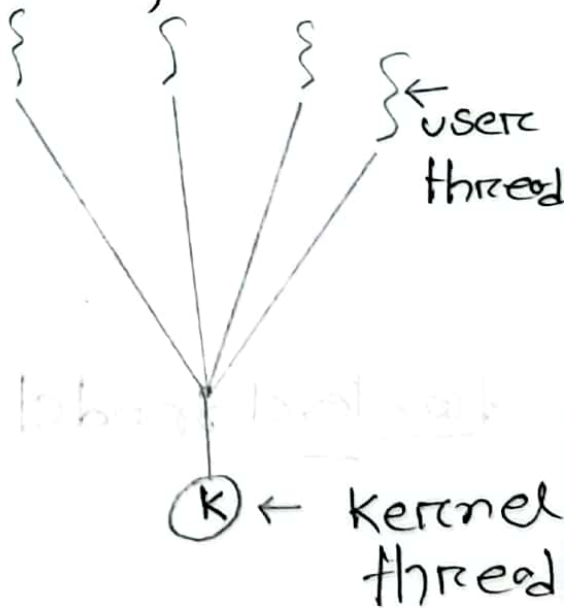
1. User label
2. Kernal label

Many relationship may exist between user level thread and multi level thread

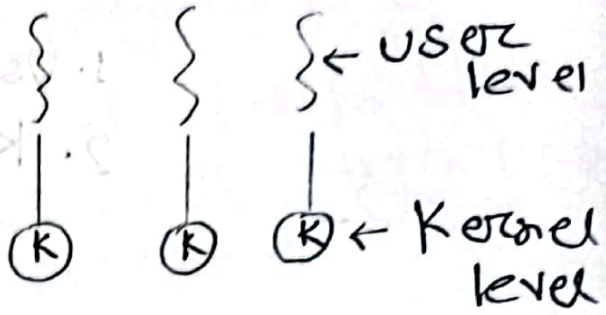
types of relationship:

- → Many to one → no concurrency facility
- → One to one → concurrency "
- → Many to ~~one~~ many → two level

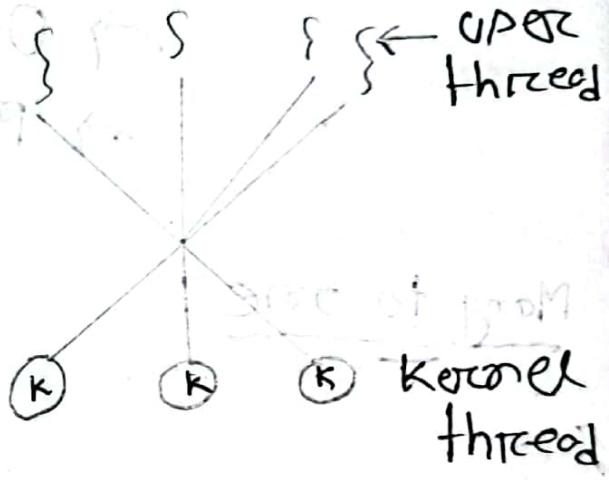
Many to one:



One to one



Many to many



two level model

Chapter-3

Interprocess communication:

Classification of process:

2 types

→ Independent process

→ co-operating process

Independent process:

Process that can not affect or^n affected by other process executing in the system.

~~Any~~ → process can't affect other process

→ " can't be affected by " "

→ " does n't share data with other process

Co-operating process:

→ process can affect other process

→ " " can be affect by other process

→ process share data with other process

Inter process Communication:

A mechanism which allows processes

→ to communicate each other

→ to synchronize their action

2 ways

1. Shared memory
2. message passing

CamScanner

Shared memory:

In the shared memory model

→ a region of memory is shared by cooperating processes is establishe

→ exchange information by reading and writing data to the shared region

Message passing:

→ communication takes place by means of message exchange between the cooperating processes.

Which one is better?

Message passing is useful for exchange smaller amounts of data. Message passing is also easier to implement in a distributed system than shared memory.

CamScanner

Shared memory can be faster than message passing since message-passing systems are typically implemented using system calls and thus require more time-consuming task of kernel ~~ex~~ intervention.

In shared memory system, system calls are required only to establish shared memory organs. Once shared memory is established no assistance from the kernel is required.

NB: Shared memory suffer from cache coherency issue which arise as shared data migrate among the several caches

Message passing System:

Provides a mechanism to allow processes to communicate and to synchronize their action without sharing the same address space.

It provides at least two operations:

→ Send (message)

→ receive (message)

~~message sent by a process~~

message can be

→ either fixed insize

→ or variable in " "

If process P and Q want to communicate they must send message to and receive message from each other. There is a communication link between them.

CamScanner

Several methods for logically implementing a link and the send()/receive() operations

1) Direct or indirect communication
2) Synchronous or asynchronous "
3) Automatic or explicit buffering.

Direct communication	Indirect communication
1) processes explicitly name the recipient or sender	1) message are send to and receive from mailbox or ports
2) Send (P, message) — Send a message to process P.	2) send (A, message) — Send a message to mailbox A.
3) receive (Q, message) — receive a message from process Q.	3) receive (A, message) — Receive a message from mailbox A
→ link is established automatically that used to communicate. Every pair of process → link associated with exactly two processes	→ link is established between a pair of processes only if the both members of the pair have shared mailbox

Direct	Indirect
→ between each there exist exactly one link	→ link may be associated with more than two processes → between each pair of communication process a number of ~~process~~ links may exist

Mail Box:

Direct communication:

it employs

→ symmetry in addressing → both the sender process and receiver must name the other to communicate.

→ asymmetry " " → only the sender names the recipient

the recipient doesn't names the sender.

primitives are defined as

Send (P, message) → send a message to process P

receive (id, message) → Receive a message from any process.

Mailbox :

Mailbox may be

→ owned by a process

→ " by the operating system.

→ part of the address space

→ distinguish between ~~the~~ we

→ the owner

→ the user

→ operating system provide mechanism

1. Create a ~~message~~ new mailbox

2. Send and receive messages through the mailbox

3. Delete a mailbox.

CamScanner

Synchronization:

message passing may be

→ blocking / synchronous

→ nonblocking / asynchronous

Blocking send: Sending process is blocked until the message is received by receiving process

~~Nonblocking send~~

Nonblocking send:

Sending process sends the message and resumes operation

Blocking receive: Receiver blocks until a message is available.

Nonblocking receive: Receiver retrieves either a valid message or a null.

CamScanner

Buffering

Message exchanged by a communicating process reside in a ~~temporary memory~~

[temporary queue]

→ implemented in 3 ways

1. Zero capacity : → has maximum length of zero
   → can't have any message waiting in it.
2. Bounded capacity : → has finite length of n
   → atmost n messages can reside in it
3. Unbounded capacity :

   ~~has length i~~

   → length is potentially infinite

   → any number of message can wait in it

Pipe:

→ pipe allow two processes to communicate.

→ first IPC mechanisms in early UNIX system

Two common types of pipe

→ ordinary pipes

→ named pipes.

Ordinary pipes

Allow two process to communicate in standard ~~procedure~~ producer - consumer fashion

→ ~~procedure~~ producer write one end of the pipe.

→ consumer read from the other end

→ ordinary pipes are unidirectional.

→ two pipes are used for two way communication.

CamScanner

Constructing pipe

~~pipe~~
constructed by using function

pipe(int fd[ ]) → create a pipe accessed through file decriptors int fd[]

fd[2] is the standard error
fd[0] is the read-end
fd[1] is the write-end

Operation:
Pipe can be accessed using ordinary

→ read() · system call
→ write() " "

close (fd[n]) – used to close unused end of the pipe.

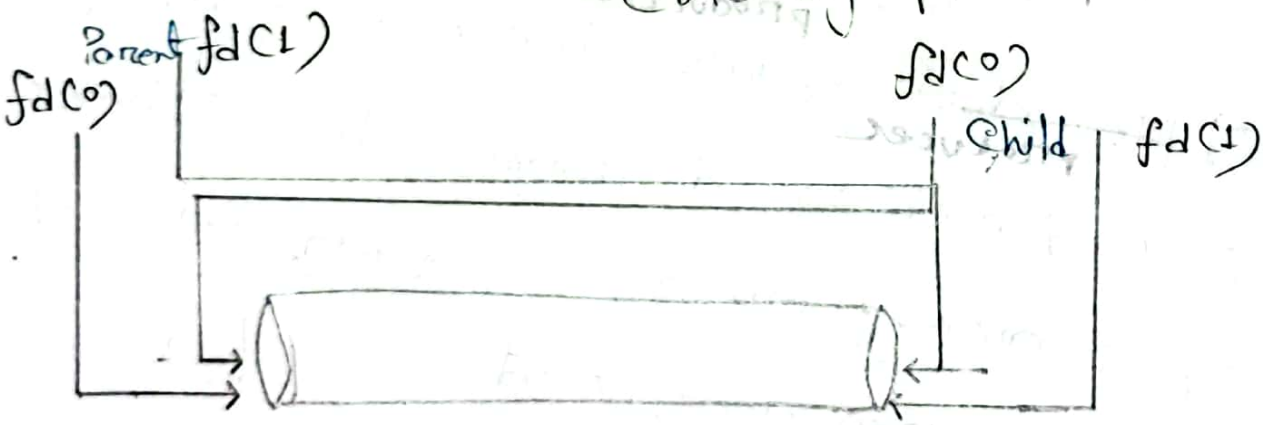


Fig: File descriptors for an ordinary pipe

"Heaven's Light is Our Guide"

RAJSHAHI UNIVERSITY OF ENGINEERING & TECHNOLOGY

RUET

II

Name : Md Ismail

Subject : CSE-3201 (Operating System) Year : ○1st ○2nd ◉3rd ○4th

Roll No : 1503094 Semester ○1st ○2nd ○3rd ○4th ○5th ◉6th ○7th ○8th

Department CE EEE ME CSE ETE IPE URP GCE MTU ARCH

190 PAGE

CamScanner

Chapter-8

Main Memory

* To ~~pe~~ be executed the program must be brought into memory and placed within a process.

Memory hierarchy:

Separates computer storage into a hierarchy based on response time.

It consists ~~ve~~ of

Volatile
- → few megabyte of very fast expensive volatile cache memory
- → few gigabyte of medium speed, medium priced, volatile main memory
- → few terabyte of slow, cheap, nonvolatile disk storage

Memory manager:

Part of the operating system that manages the memory hierarchy.

Function of memory manager:

→ keep track on which part of the memory are in use

→ allocate memory to process, when they need it

→ deallocate it, when they are done.

Address Binding:

It is a process of mapping from one address space to another address space

3 types:

→ Compile time address binding

→ loading time " "

→ execution " " "

Something:

→ address in the source program are generally symbolic

→ Compiler typically binds these symbolic addresses to relocatable addresses

→ the linkage editor or loader, in turn binds the relocatable addresses to absolute address.

Compile time:

If it is known that, where the process will resides in memory, then absolute code can be generated at compile time

load time:

If it is not known that compile time that where the process will reside, then compiler must generate relocatable code.

In this case final binding is delayed until load time.

CS CamScanner

Execution time:

If the process can be moved during its execution from one memory segment to another, binding ~~to~~ must be delay until run time.

Logical address space:

The set of all logical address generated by a program is a logical address space.

Physical address space:

The set of all physical address corresponding to these logical address is physical address space.

CamScanner

Logical address Vs physical address:

Logical address	physical address
1. It is the virtual address generated by CPU	1. A location in memory unit
2. Set of all logical addresses generated by CPU in reference to a program is referred as logical address space.	2. Set of all physical address mapped to corresponding logical addresses is referred as physical Address space
3. User can view the logical address of a program	3. The user can never view physical address of program
4. User uses the logical address to access physical address.	4. The user can't directly access physical address
5. generated by CPU	5. Computed by MMU. or seen by MMU

MMU :

The run time mapping from virtual to physical addresses is done by a device called memory management unit (MMU)

Dynamic loading :

→ it is a memory allocation technique

→ A routine is not loaded until it is not called.

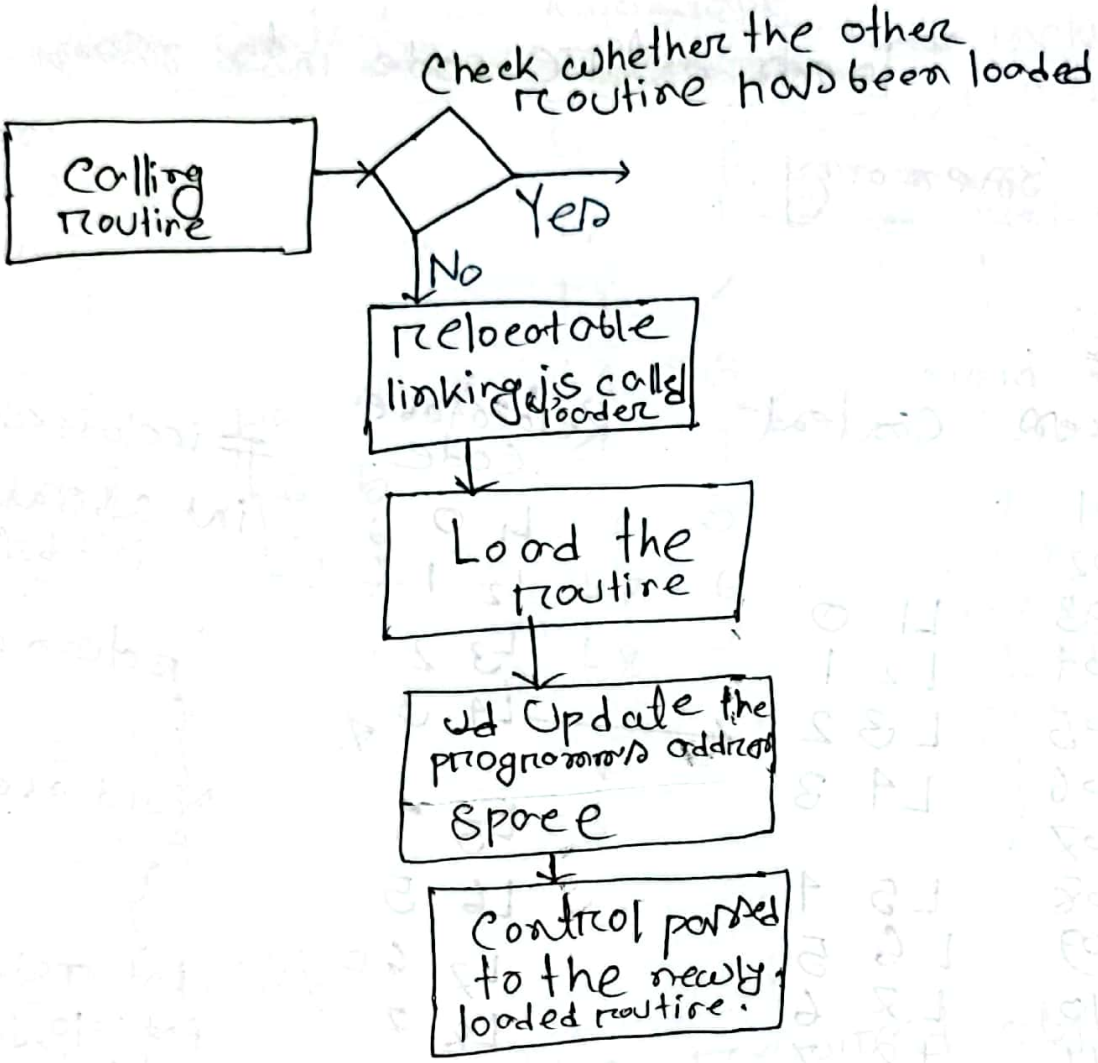
→ All routines are kept on disk in a relocatable load format.

→ the main program is loaded into memory and executed

→ when a routine needs to call another routine, the calling routine first check whether the other routine has been loaded.

→ If not, the relocatable linking loader is called to load the desired routine into memory and to update the programs address table.

→ control is passed to the newly loaded routine.



Advantages:

→ capable of better memory space utilization

→ doesn't require special support from the operating system.

→ useful in handling infrequent occuring cases like error routine

→ Speed is better.

→ unused routine is never loaded.

Static loading

loads entire code into main memory.

Exaple:

Address	Content	Relocatable code
0001		L1 0
0002		L2 1
0003	L1 0	L3 2
0004	L2 1	L4 3
0005	L3 2	L5 4
0006	L4 3	L6 5
0007		L7 6
0008	L5 4	L8 7
0009	L6 5	
0010	L7 6	
0011	L8 7	
0012		
0013		
0014		

Saved to Hard Drive

```
# include<Stdio.h>
int sub(int a,
        int b){
  ---
  return a-b;
}
void abc(){
  ---
}
int main(){
int i=10, j=20
int k;
k=i+j;
print(k);
}
```

Source Code

Example Of Dynamic loading

Address	Content	Relocatable Code	
0001		L_1	0
0002		L_2	1
0003		L_3	2
0004		L_4	3
0005			
0006	L_5 4	L_5	4
0007	L_6 5	L_6	5
0008	L_7 6	L_7	6
0009	L_8 7	L_8	7
0010			
0011			
0012			
0013			
0014			

```
# include<stdio.h>
int sub(int a, int b){
  - - - -
  return a-b;
}
void cube(){
  - - -
}
int main(){
  int i=10, j=20, k;
  k=i+j;
  print(k);
}
```

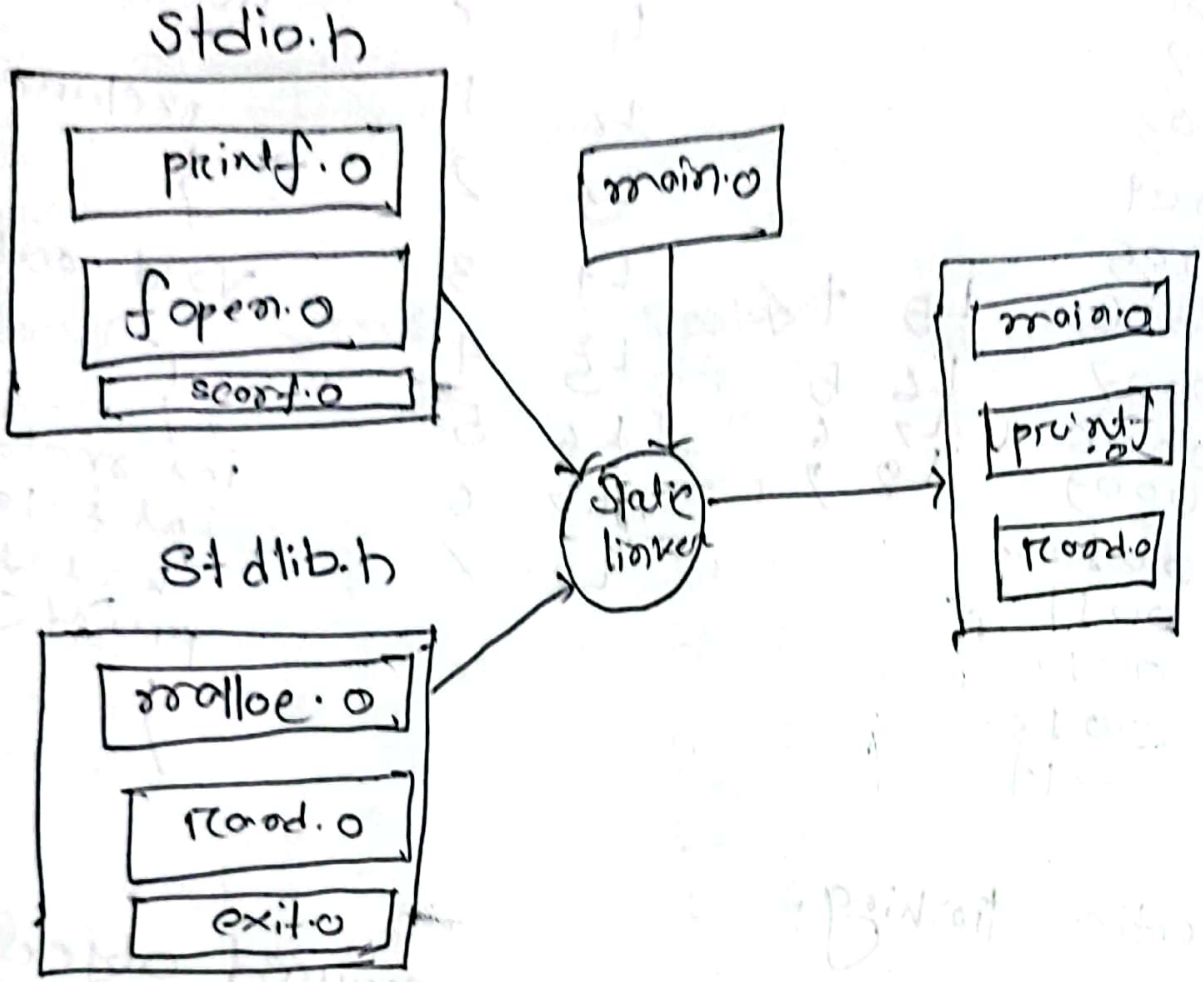
Static linking:

→ copied the required object code from library and put them into the executable file.

→ done at compile time

fig

Block diagram of static linking/example



Dynamic linking:

→ a stub is include in the image for each library routine reference.

→ Stub is a small piece of code

→ how to locate the appropriate memory-resident library

→ how to load the library if the routine is not already present.

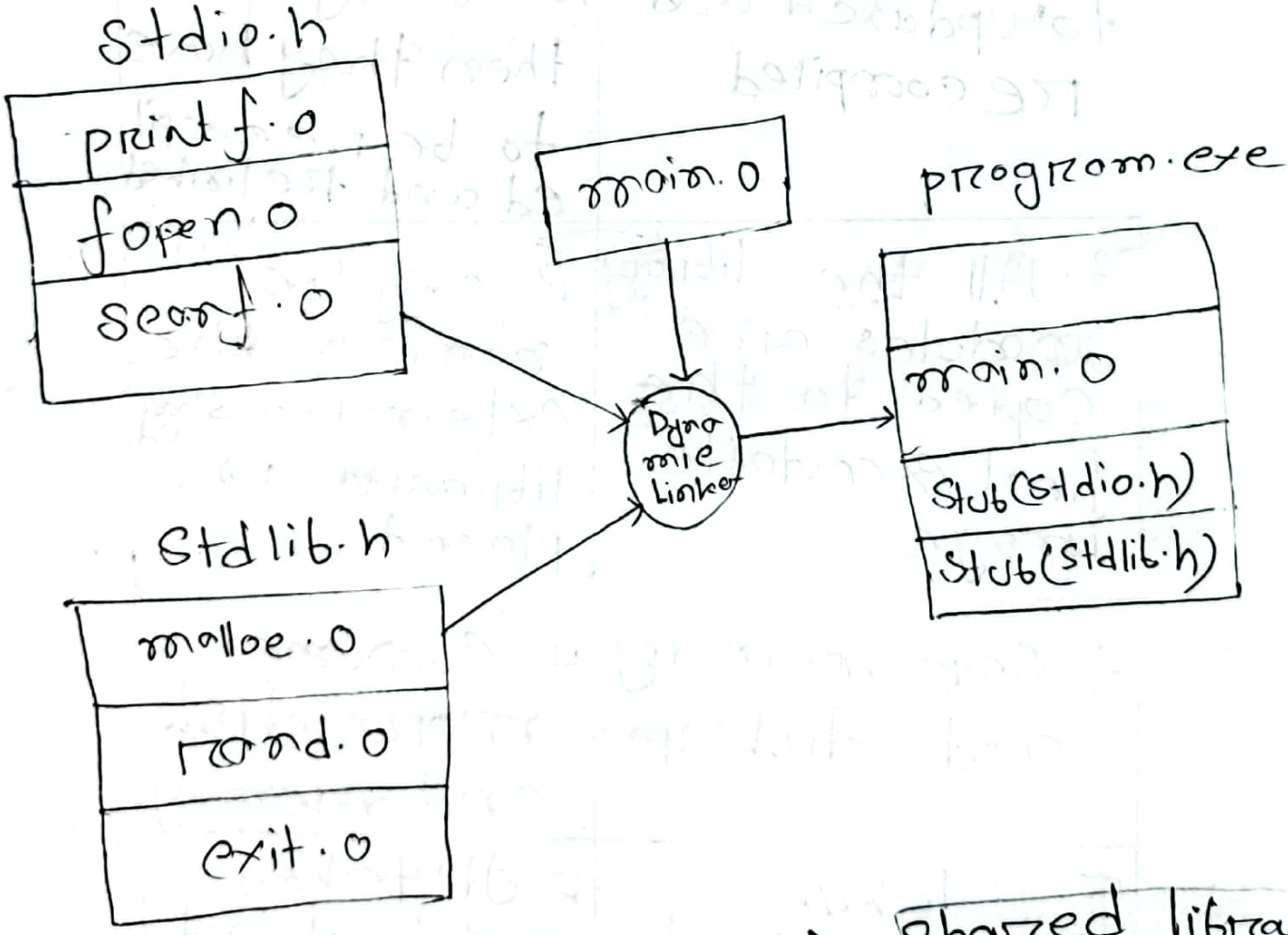
→ Stub is executed

→ Checks to see whether the needed routine is already in memory.

→ If not program loads the entire routine into memory.

→ Stub replaces itself with the address of the routine

→ execute the routine.



→ system is also known as shared libraries.

Dynamic linking vs Static linking

Dynamic linking	Static linking
1. Done at run time by OS	1. Done by the compiler by the linker
2. Only a single module needs to be updated and recompiled	2. If external source program is changed then they have to be recompiled and relinked
3. All the library modules are copied to the final executable image	3. Only the names of the external or shared libraries is placed
4. Save memory and disk space	4. Consume more disk and memory
5. It takes variable load time	5. It take constant load time.
6. Slower execution time	6. Faster execution time

Backing store is commonly a fast disk

PTO

A process can be temporarily removed out of memory to backing store and then brought back into the memory for continued execution. This is known as Swapping.

Ex:- Multiprogramming environment with a Round Robin CPU Scheduling.

Swapping procedure:

When priority a ready queue of processes

→ CPU scheduler decides to execute a process.

→ Dispatcher is called to run the selected process.

→ Dispatcher check whether the scheduled process is present in the main memory

→ If not and if there is no free memory, it swap out a process

→ Swap in the desired process.

→ It then reload registers ~~and~~
→ transfer control to the selected process.

NB context switching time of swapping is fairly high.

Backing store:

→ it is commonly a fast disk
→ must be large enough to accommodate copies of all memory images for all users
→ provide direct access to this memory.

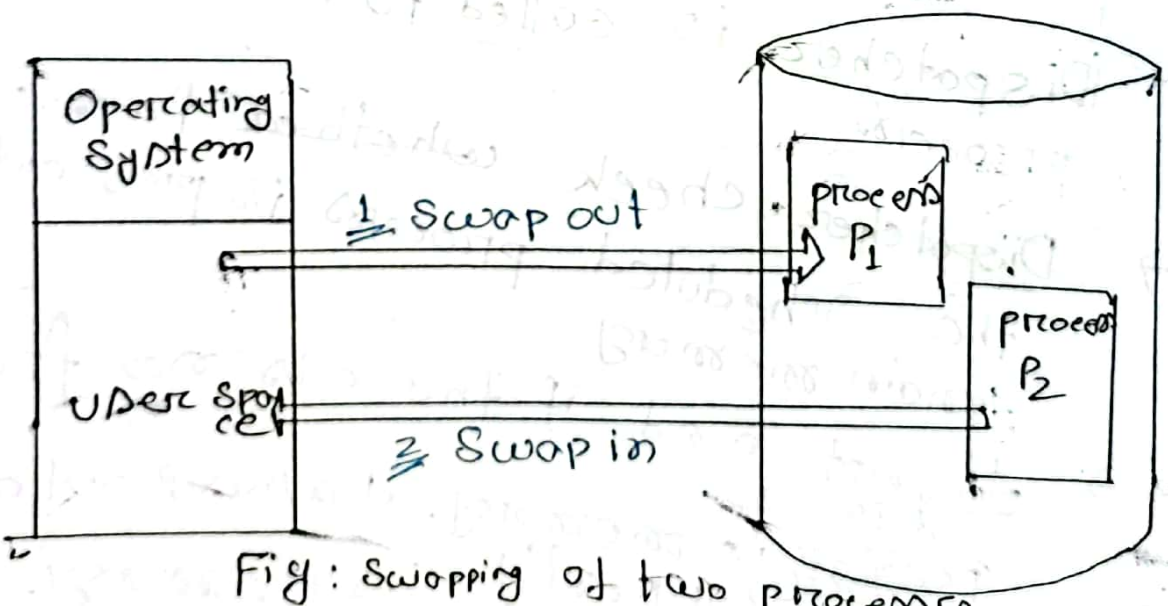


Fig: Swapping of two processes

Memory Allocation

2 types
- → Contiguous — [→ fixed sized partitioning / → variable " "]
- → Non contiguous

Contiguous memory allocation :

Each process is contained in a single section of memory that is contiguous to the section containing the next process.

Non-contiguous memory allocation:

Each process is origin the scattered memory blocks of different location in memory.

Memory is devided into 2 partitions

1. High memory → for user process
2. low memory → for operating system

Why operating system resides in a low memory:

→ Operating system can place in high memory or low memory

→ It depends on interrupt vector.

→ As interrupt vector often is in low memory, operating system place in low memory.

Memory protection:

We need to prevent a process from ~~overlapping memory it does not own~~.

→ prevent to overlap other's memory space

→ prevent to overlap operating system.

Memory protection in contiguous memory allocation

1. CPU scheduler select a process for execution
2. The dispatcher loads the relocation and limit register with the correct value
3. Address is checked against these registers.
4. Operating system and other users program protected from data being modified by running process.

relocation register:

contains the value of smallest physical address

limit register:

contains the range of logical address.

Each logical address must fall within the range specified by the limit register

NB: The MMU maps the logical address dynamically by adding the value in the relocation register.

CamScanner

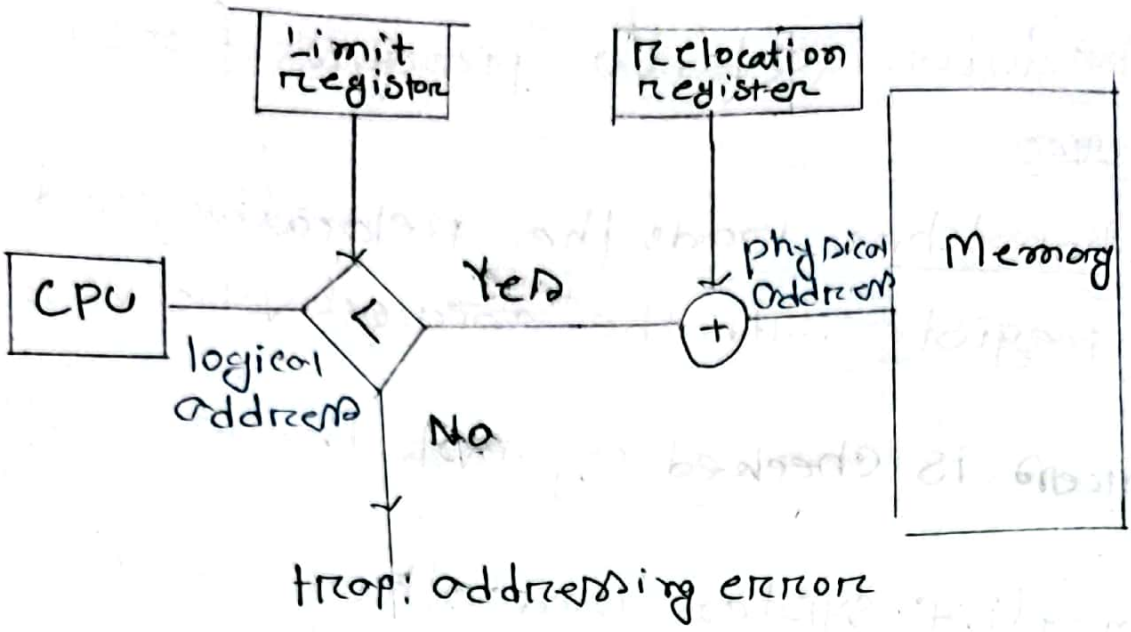


Fig: Hardware support for relocation and limit register

Partioning memory:

2 types

1. fixed-sized partition
2. variable-sized partition

fixed sized partition: → degree of multiprogramming is bound by the number of partition.

→ divide memory into several fixed size partition

→ each partition may contain exactly one process

Thus degree of multiprogramming is bounded by the number of partitions

ble- sized partition:

→ OS keeps a table indicating which parts of memory are available and which parts of memory are occupied

→ Hole: All memory are available for process initially and is considered one large block of available memory, is called hole.

memory allocation ~~sta~~ strategies:

1. First fit → allocate the first hole that is big enough
   → searching can start either at the beginning of the hole or, where the previous first fit search ended

2. Best fit → produces smallest leftover hole → Allocate the smallest hole.

3. Worst fit → produces the largest leftover hole. → Allocate the largest hole.

Search entire list unless the list is ordered by sized.

Fragmentation:

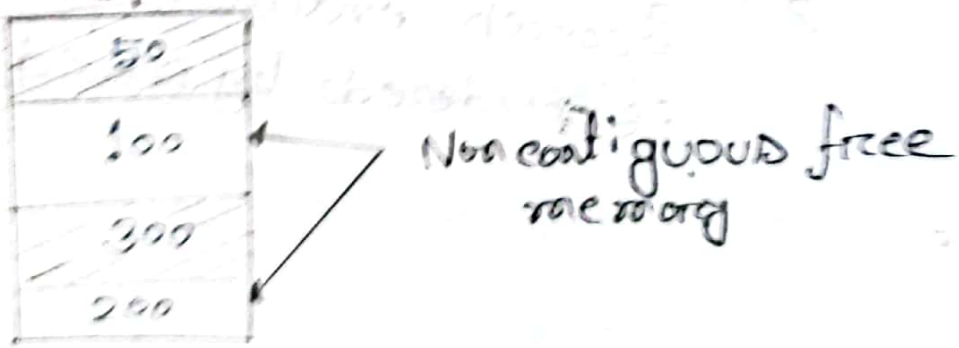
It is a phenomenon in which Storage space is used inefficiently, reducing capacity or performance and often both.

Classification of fragmentation:

2 types
→ External fragmentation
→ Internal fragmentation.

External fragmentation:

It exists when there is enough total memory space to satisfy a request but the available spaces are not contiguous.



Internal fragmentation:

If memory allocated to a process is slightly bigger than the process. It leads to internal fragmentation.

internal fragmentation

→ break the physical memory into fixed size blocks

→ allocate memory in units based on block size

Solution of internal fragmentation:

can be reduced by effectively designing the smallest partition but not large enough for the process.

Solution of External fragmentation:

→ compaction

→ segmentation permit logical address space of the process to be noncontiguous.

→ paging

Allocation techniques can also be combined.

→ Shuffle all the memory contents so as to place all free memory together in one large block.

→ possible only if relocation is dynamic, i.e. it can be done in execution time.

Internal vs External

Internal fragmentation	External fragmentation
1. When the memory allocated to the process is slightly larger than the memory requested by a process	1. Occurs when there is enough total space but they are not contiguous
2. Memory must be partitioned into variable sized blocks and allocate the best fit block to the process	2. Compaction, paging, Segmentation
3. Occurs when fixed sized partitions is used	3. when variable size memory space are allocated

Chapter- 9
Virtual memory

Virtual memory:

It involves separation of logical memory from physical memory as perceived by users.
অবহিত হওয়া

→ allows an extremely large virtual memory to be provided for programmer, when smaller physical memory is available.

→ programmer no longer need to worry about the amount of physical memory.

Virtual address space:

→ refers to the logical view of how a process is stored in memory.
→ All

Address space:

range of memory addresses available to process.

Benefits of virtual memory:

→ In process address space, space between stack & heap is a part of the virtual memory address.

~~but we the~~

Here heap grow upward

Stack grow downward.

Sparse address space:

Virtual address space that includes holes are known as sparse address space.

~~Spbe~~

Sparse address space is beneficial as holes can be filled up with stack and heap segment grow dynamically.

Virtual memory allows files and memory to be shared by two or more processes through page sharing.

Demand paging :

→ load the pages only if they are needed during program execution.

→ Similar to paging system with swapping where processes reside in secondary memory.

→ When we want to execute, we swap it into memory.

→ A lazy swapper is used.

→ It never swaps a page unless that page will be needed

Swapper vs pager

→ Swapper manipulate entire process where pager concerned with the [illegible] of a process

→ term pager is used for demand paging.

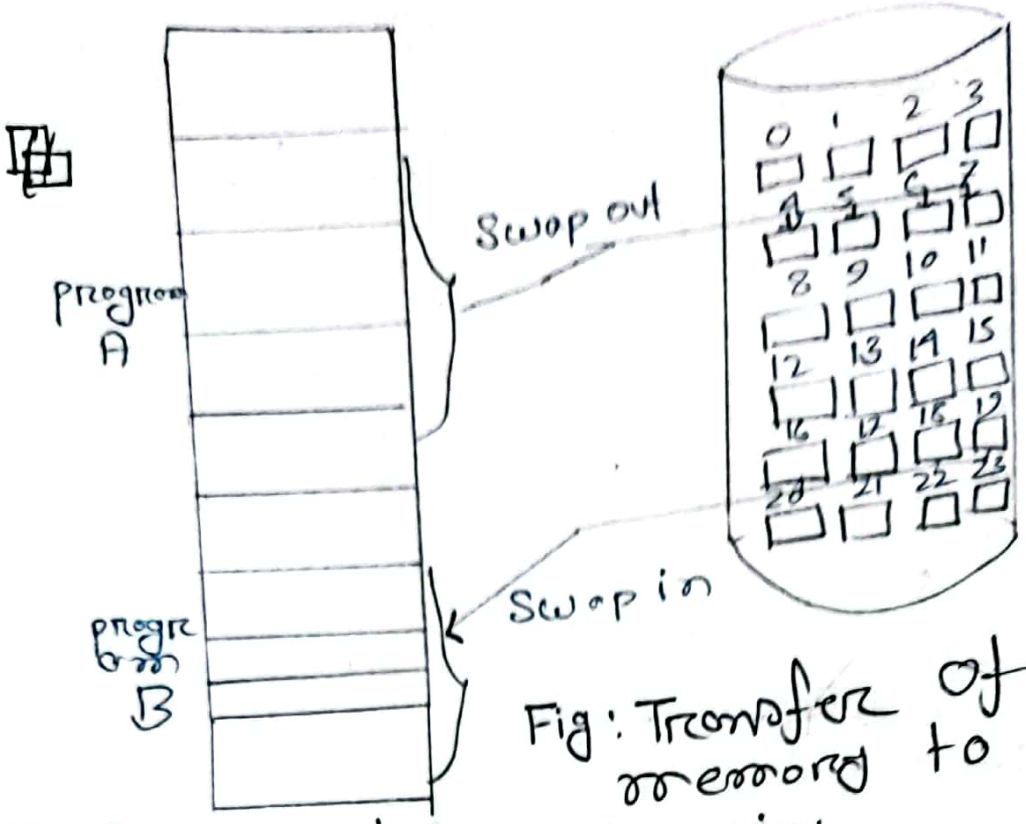


Fig: Transfer of a paged memory to contiguous disk space

Process of demand paging:

→ When a process is needs to be swapped in for execution, pager guesses which pages will be used before the process is swapped out again.

→ Valied-invalied bit scheme can be used in page table, ~~to make~~, to make whether the pages associated with the process are on memory or not.

Valied: pages is both legal and main in memory.

invalied: pages is either not legal or is legal but not present in main memory.

→ during execution whose valied invalied bit in the page table is marked as invalied then it causes page fault.

→ ~~moki~~ an invalied page has no effect if the process never attempt to access that page.

CamScanner

~~Procedure for handeling~~

Handeling page fault:

→ Check the internal table

→ if the reference was invalied, terminate the process

→ if the reference was valied, but the referred page have not yet brougth in, then page it in.

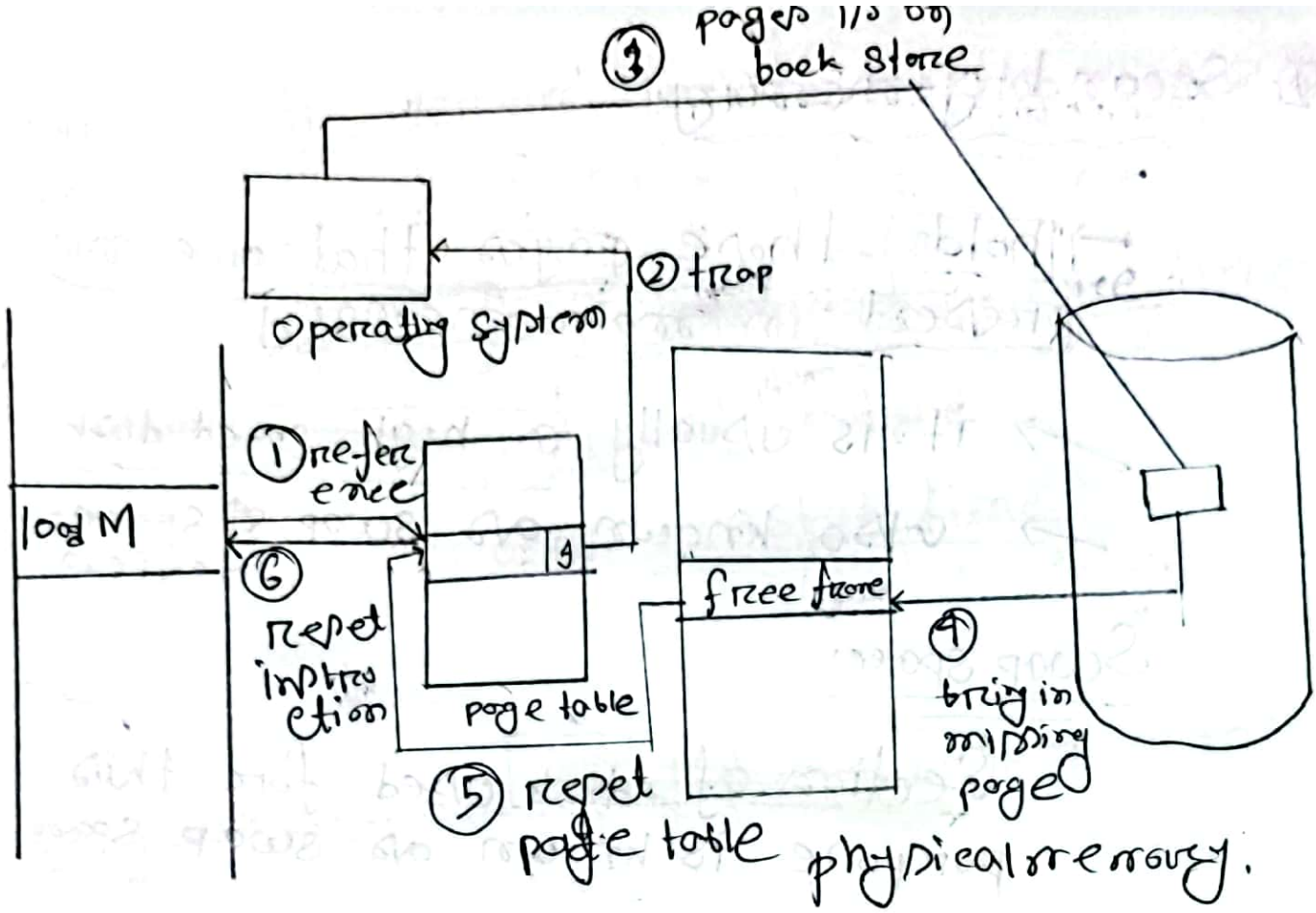
→ free frame is located

→ Schedule a disk operation, to read the desired page into newly allocated frame.

→ after completation of disk read, the page table is modified

→ Restart the instruction that was interrupted by the trap.

* in extreme case, we can start executing a process with no pages in memory.



Hardware to support demand paging:

→ Same as hardware for → paging
→ swapping

page table:

→ has the ability to mark an entry invalied through a valied-invalied bit or, a physical value of protection bits.

Secondary memory:

→ (holds) those pages that are not present in main memory

→ it is usually a high-speed disk.

→ also known as swap ~~space~~ device

Swap space:

Section of disk used for this purpose is known as swap space. → swapping

Performance of demand paging:

~~effective odd~~
effective access time,

$$= (1-P) \times ma + P \times \text{page fault time}$$

where,

P = probability of page fault

ma = memory access time

NB: ma ranges from (10 to 200) ns

Effective access time ∝ page fault time.

- Page fault service time majorly consists of:
  - → page fault interrupt service time
  - → page read in time
  - → process restart time

- Page fault causes the following sequence to occur:
  1. Trap to the operating system
  2. Save the user registers & process state
  3. determine that the interrupt was a page fault?
  4. Check the page reference was legal.
  5. determine the location of the page on disk.
  6. Issue a read from the disk to free frame.

CamScanner

6.
- → wait in the queue
- → wait for device seek and/or latency time
- → begin the transfer of the page to a free frame

7. While waiting, allocate the CPU to some other user
8. Receive an interrupt from the disk
9. Save the registers and process state for the other user.
10. Determine that the interrupt was from the disk.
11. Correct the page table and other tables.
12. Wait for the CPU to be allocated to this process again.
13. All registers, table and state are restore and then resume the interrupt instruction.

CS CamScanner

Page replacement:

When need:

When no frame is free to page in the desired page from the back store on disk.

It involves:

→ Locating a frame in memory which is not currently in use and then freeing it by paging out its content

→ Change the page table.

→ Page in the desired page into the freed frame

→ Reset the page table.

Modification of page-fault service routine to include page replacement

→ find the location of the desired page on the disk

→ Find a free frame

→ if there is a free frame, use it

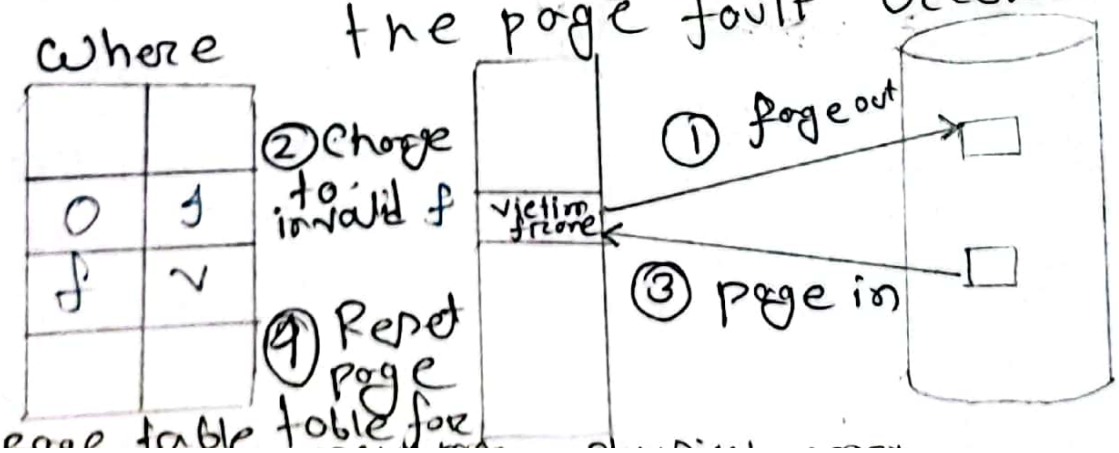
→ if no free frame use page replacement algorithm

→ write the victim frame to the disk.

→ Read the desired page into the newly freed frame

→ Change the page & frame table

→ Continue the user process from where the page fault occurred.



NB: If no frames are free,

two page transfers are required.

→ one in

→ one out

~~If effectively doubles the page fault~~

If effectively doubles the

→ page fault service time

→ increase the effective access time

~~Classification of P~~

Page replacement algorithm:

→ FIFO page replacement

→ Optimal page replacement

→ LRU page replacement

CamScanner

FIFO page replacement:

→ assign a time to each page

→ Oldest page is chosen to replace

→ FIFO queue is used for recording time.

Optimal page replacement:

→ replaces the page that will not used for the longest period of time

→ difficult to implement.

→ as it requires future knowledge of reference string

LRU page replacement:

→ page that has not been used for longest period of time.

CamScanner

Belady's anomaly:

For some page replacement algorithm, the page fault rate may increase as the number of allocated frame increases.

page fault rate $\propto$ number of allocated frame.

Advantages of FIFO page replacement

→ Easy to understand
→ Easy to implement

Disadvantages of FIFO page replacement:

→ System need to keep track of each frame
→ suffer from Bady's anomaly.
→ not very effective
→ it could contain a heavily used variable that was initialized early.

CamScanner

→ it needs future knowledge
→ difficult to implement
~~→ twice as good as FIFO~~

Adv. of Optimal page replacement

→ lowest page fault rate
→ never suffer from Belady's anomaly
→ twice as good as FIFO replacement.

Advantages of LRU:

→ Never suffer from Belady's anomaly
→ amenable to full scall analysis

Disadv. of LRU:

→ require substantial h/w assistance

LRU can be implemented
→ by using counter
→ by " stack

Pure demand paging:

Never brings a page into memory until it is required

Find the page fault by using

- FIFO,
- LRU and
- OPR

7 0 1 2 0 3 0 4 2 3 0 4 2 0 3 1 2 3 7 2 1 0 7

frame size = 3

Sol^n:

1. FIFO

7 0 1 2 0 3 0 4 2 3 0 4 2 0 3 1 2 3 7 2 1 0 7

7	7	7	1	2	2	4	4	4	0	0	0	3	3		3	2	2				
	0	0	0	3	3	3	2	2	2	4	4	4	1		1	1	0				
		1	1	1	0	0	0	3	3	3	2	2	2		7	7	7				

∴ Number of page fault = 17

Number of page hit = 6

CamScanner

ii LRU

7 0 1 2 0 3 0 4 2 3 0 4 2 0 3 1 2 3 7 2 1 0 7

7	7	7	2		2		4	4	4	0	0	0		0	0	2		2		2	2	7
	0	0	0		0		0	0	3	3	3	2		2	1	1		7		7	0	0
		1	1		3		3	2	2	2	4	4		3	3	3		3		1	1	1

Number of page fault = (17+1) = 18

Number of page hit = 5

iii OPR

7 0 1 2 0 3 0 4 2 3 0 4 2 0 3 1 2 3 7 2 1 0 7

7	7	7	2		2		2			2				2	2			2			0	
	0	0	0		0		4			4				3	3			7			7	
		1	1		3		3			0				0	1			1			1	

Number of page fault = 11

Number of page hit = 12

Find out the page fault for the following reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

by using FIFO,
LRU
OPR

frame size = 3

Sol^n

FIFO:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2		2	2	4	4	4	0			0	0		7	7	7
	0	0	0		3	3	3	2	2	2			1	1		1	0	0
		1	1		1	0	0	0	3	3			3	2		2	2	1

Number of page fault: 15

OPR

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2		2		2		2		2					7		
	0	0	0		0		4		0		0					0		
		1	1		3		3		3		1					1		

Number of page fault = 9

CamScanner

LRU:

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2		4	4	4	0			1		1		1		
	0	0	0		0		0	0	3	3			3		0		0		
		1	1		3		3	2	2	2			2		2		7		

Number of page fault: 12

Determine the number of page fault by using FIFO, LRU, OPR

1 2 3 4 2 4 1 5 6 6 2 1 2 3 7 6 3 2 1 2 3 6

frame size = 4

Sol^n:

LRU:

1	2	3	4	2	4	1	5	6	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1				1	1		1			1	1	6			6			
	2	2	2				2	6		6			6	7	7			1			
		3	3				5	5		5			3	3	3			3			
			4				4	4		2			2	2	2			2			

Number of page fault = 11

Ans

FIFO

1	2	3	4	1	4	1	5	6	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1				5	5		5	5		3	3	3		3	1		1	
	2	2	2				2	6		6	6		6	7	7		7	7		3	
		3	3				3	3		2	2		2	2	6		6	6		6	
			4				4	4		4	1		1	1	1		2	2		2	

∴ Number of page fault = 14 Ans

OPR:

1	2	3	4	2	4	1	5	6	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1				1	1						1	1						
	2	2	2				2	2						2	2						
		3	3				3	3						3	3						
			4				5	6						7	6						

∴ Number of page fault = 8 Ans

CamScanner

Determine the page fault

1, 2, 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

frame size = 4

by using: i) LRU

ii) OPR

iii) FIFO

Solⁿ i LRU

1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

1	1	1	1			1	1				1	1	6			6			
	2	2	2			2	2				2	2	2			2			
		3	3			5	5				3	3	3			3			
			4			4	6				6	7	7			1			

Number of page fault = 10

II OPR

1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

1	1	1	1			1	1				7				1				
	2	2	2			2	2				2				2				
		3	3			3	3				3				3				
			4			5	6				6				5				

Number of page fault = 8

iii) FIFO

1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

1	1	1	1			5	5	5	5	3	3	3		3	1		1		
	2	2	2			2	6	6	6	6	7	7		7	7		3		
		3	3			3	3	2	2	2	2	6		6	6		6		
			4			4	4	4	1	1	1	1		2	2		2		

Number of page fault = 14

Q For the following determine page fault.

1, 2, 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

by using FIFO, ~~FI~~ LRU, OPR.

Frame size = 5

Sol^n:

FIFO

1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

1	1	1	1			5	6		6	6	6	6							
	2	2	2			2	2		1	1	1	1							
		3	3			3	3		3	2	2	2							
			4			4	4		4	4	3	3							
						5	5		5	5	5	7							

Number of page fault = 10

CamScanner

LRU

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1			1	1				1	1							
	2	2	2			2	2				2	2							
		3	3			3	6				6	6							
			4			4	4				3	3							
						5	5				5	7							

Number of page fault = 8

OPR

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1			1	1					1							
	2	2	2			2	2					2							
		3	3			3	3					3							
			4			4	6					6							
						5	5					7							

Number of page fault = 7

1B : Technique:

FIFO: ফ্রেমের ভিতরের সবগুলি backwardly দেখতে থাকব, যেটা আগে আসবে তাকে Replace করব

LRU: ফ্রেমের ভিতরে যেগুলো আছে সেগুলোর sequence দেখব, i.e. কোন যে আগে এসেছিল,

OPR: ফ্রেমের ভিতরে যেগুলো আছে সেগুলোর ভবিষ্যতে কে সবার শেষে এসেছে, তাকে Replace করব।

→ যদি কেউ না থাকে তাকে আগে Replace করব

→ যদি একের অধিক না থাকে তাহলে যে বেশি সময় ধরে ছিল তাকে Replace করব।

CamScanner

Chapter- 8 (2)
Main memory

Segmentation:

Segementation is a memory management scheme that supports programmer view of memory.

→ logical address space is a collection of segments

→ Each segment has name and length.

→ programmer specifies each address by two options

1. Segment name
2. offset within the segment

Segment are numbered and are referred to by a segment number.

→ logical address consists of two tuple

<segment number, offset>

Segment table:

A table stores the information about all such segments and is called segment table.

→ It maps two dimensional user defined address into one dimensional physical address

→ ~~Segme~~

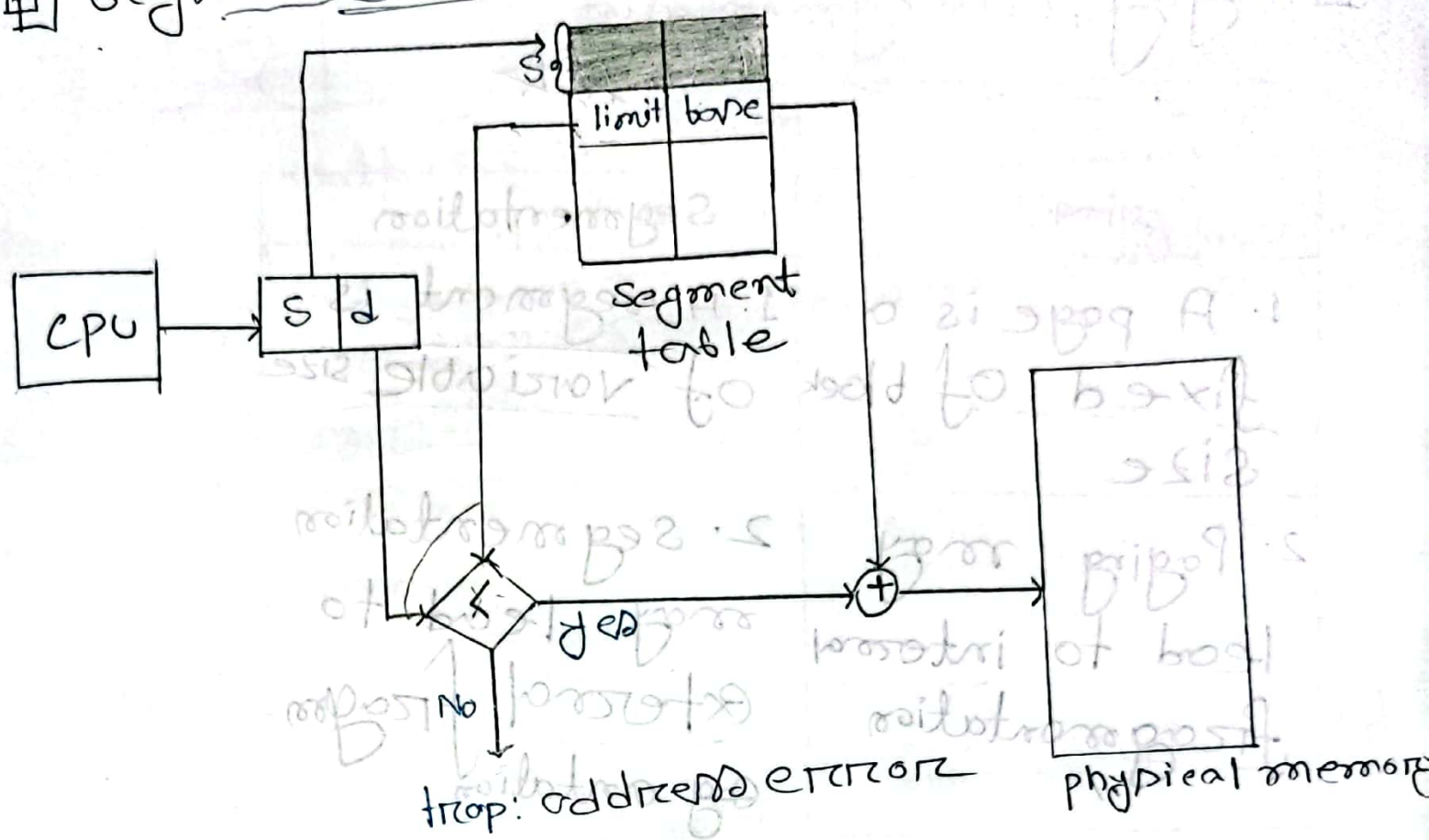
→ 2 entry → contains the starting physical address

- Segment base
- segment limit → specifies the length of the segment

C compiler create separate segment for the following

1) The code - Code segment
2) Global variable - [global constant object (Code segment)
   [" non- " " (Data segment)
3) The heap - Heap segment
4) The stack - Stack segment
5) The standard-C library

Segmentation Hardware:



logical address consists of 2 parts
segment number s
segment offset d

- Segment number is used as an index to the segment table.
- d must be between 0 and segment limit
  If not trap to the operating system

Paging vs Segmentation ★★

Paging	Segmentation
1. A page is a fixed of block size	1. A segment is of variable size
2. Paging may lead to internal fragmentation	2. segmentation may lead to external fragmentation
3. The hardware decides the page size	3. Segment size is specified by the user
4. page table contains base address of each page	4. segmentation table contains segment number and offset (segment length)
5. Address is divided by CPU into a page number, offset	5. user specifies each address by 2 quantities segment number, limit

Paging:

It is a memory management scheme that allows a process to be stored in a non-contiguous manner.

Paging method:

frame: breaking physical memory into fixed-sized blocks called frames.

page: breaking logical memory into block of some fixed size called pages.

→ pages are load into any available memory frame

→ backing store is divided into fixed-sized block → same as the memory frame

CamScanner

Address generated ~~by~~ the CPU is ~~de-vi~~ divided into

CPU generated address is divided into 2 parts

1) a page number (p)
2) a page offset (d)

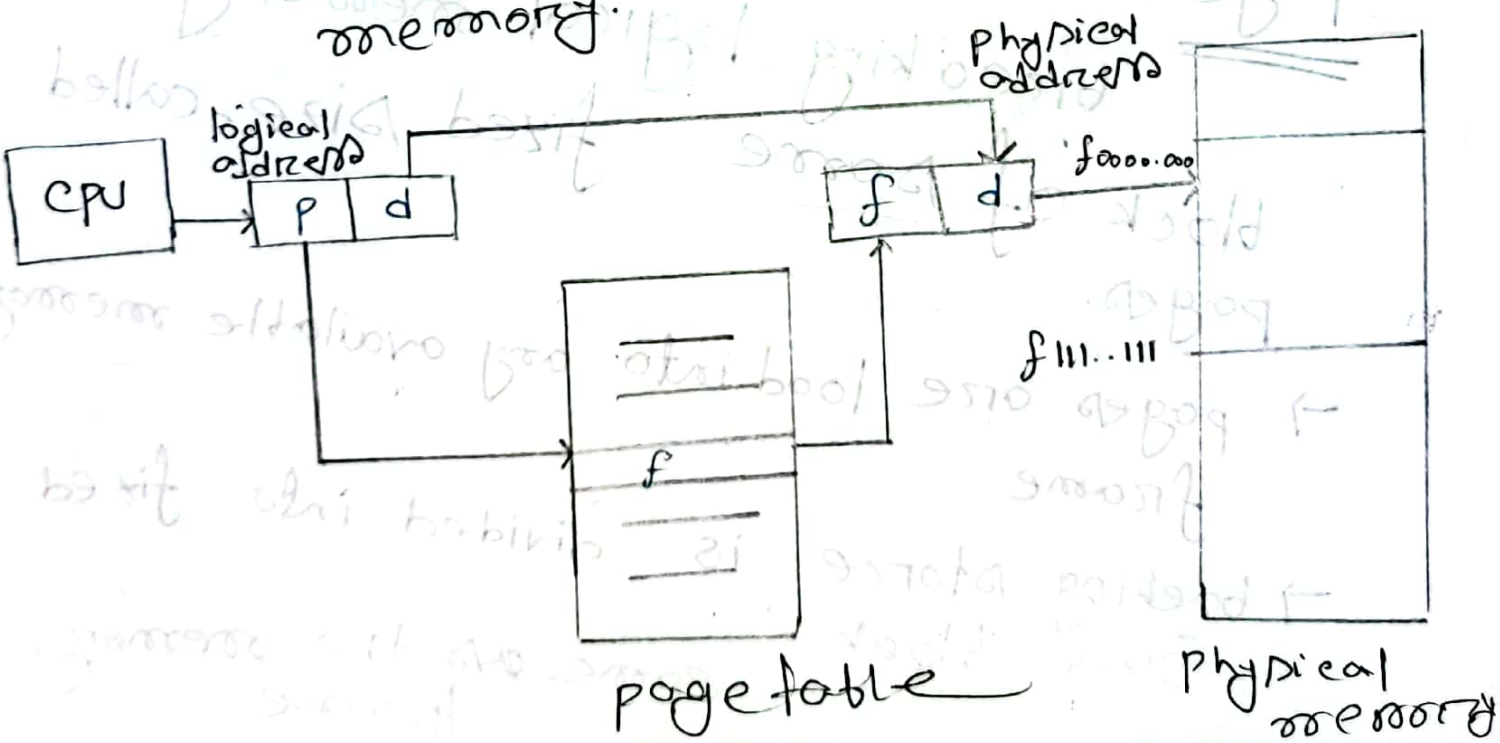
→ used as an index of page table

page table:

→ contains the base address of each page in physical memory.

~~→ pag~~

→ base address is combined with page offset to define the physical memory.



Why size of the page is power of 2?

Ans

→ Page size is defined by the hardware.

→ varying between $\boxed{512 \text{ bytes and } 1GB}$.

Page size is power of 2 as a page size makes the translation of a logical address into a

→ page number
→ page offset

Let,

size of the logical address space $= 2^m$

" of the page $= 2^n$

page number = high order $(m-n)$ bit of logical address

page offset = low order n bits.

page number	page offset
p	d

→ P is an index into the page table

→ d " the displacement within the page.

Example

let, $m = 4$ $(4-2) = 2$

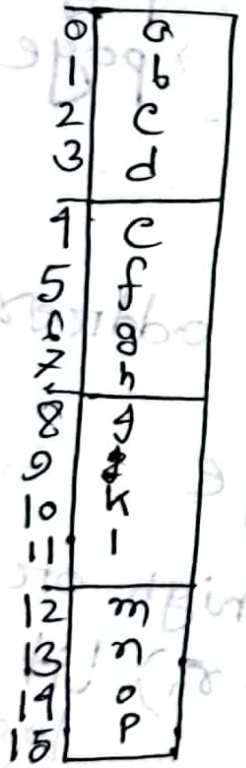
$n = 2$

physical memory = 32 bytes

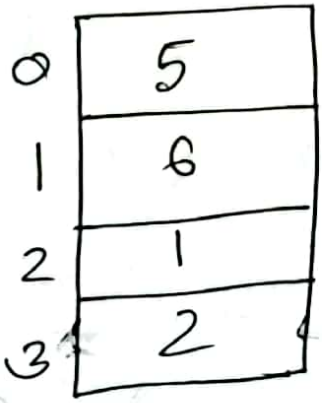
∴ logical address ~~of page size~~ = 4 ~~bit~~ bit

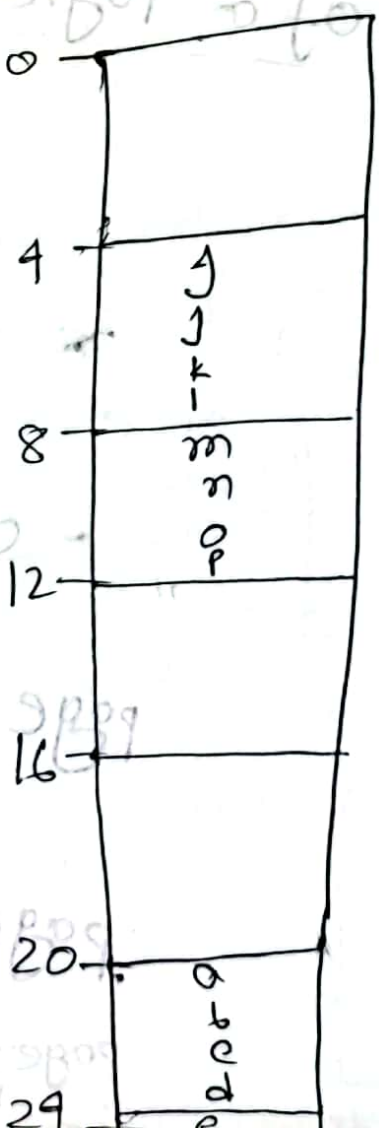
∴ " " range = 0000 - 1111

∴ page size = $4 = 2^2$



logical address





logical address 0 = 0000

∴ page number = 00 = 0

~~offset~~ offset = 00 = 0

From page table.

page 0 is in frame 5

∴ physical address = (5×4) + 0 = 20

logical address 4 = 0100

page number = 01 = 1

" offset = 00 = 0

∴ From page tabe page 1 is in frame = 6

∴ physical address = (6×4) + 0 = 24

logical address = 10 = 1010

page number = 10 = 2

" offset = 10 = 2

∴ From page tale page 2 is in frame 1

∴ physical address = (1×4) + 2 = 4+2 = 6

Ans

Paging hardware

H/w implementation can be done several way.

One way:

→ page table is implemented as a set of dedicated registers.

Should be built with very high speed logic to make the paging-address translation efficient.

→ CPU dispatcher load this register.

→ usage of register is satisfactory only if page table is small.

~~Another~~

another way:

If the page table contain large number of entries then we can use TLB (translation-look-aside buffer)

→ It is a special, small fast lookup hardware cache.

→ It is a associative high speed memory

~~TLB has 2 entry,~~

→ Each entry in TLB has 2 parts
1. a key (or tag)
2. a value

→ When this memory is used, then an item is compared with all tags simultaneously.

→ If the item is found corresponding value is returned.

→ The search is fast.

→ A TLB lookup in modern hardware is part of the instruction pipeline.

CamScanner

→ to be able to execute the search

→ TLB must be kept small.

→ typically between 32 and 1024 entries.

Mechanism:

→ TLB is used with page tables.

→ Contains ~~the~~ only a few of the page entries.

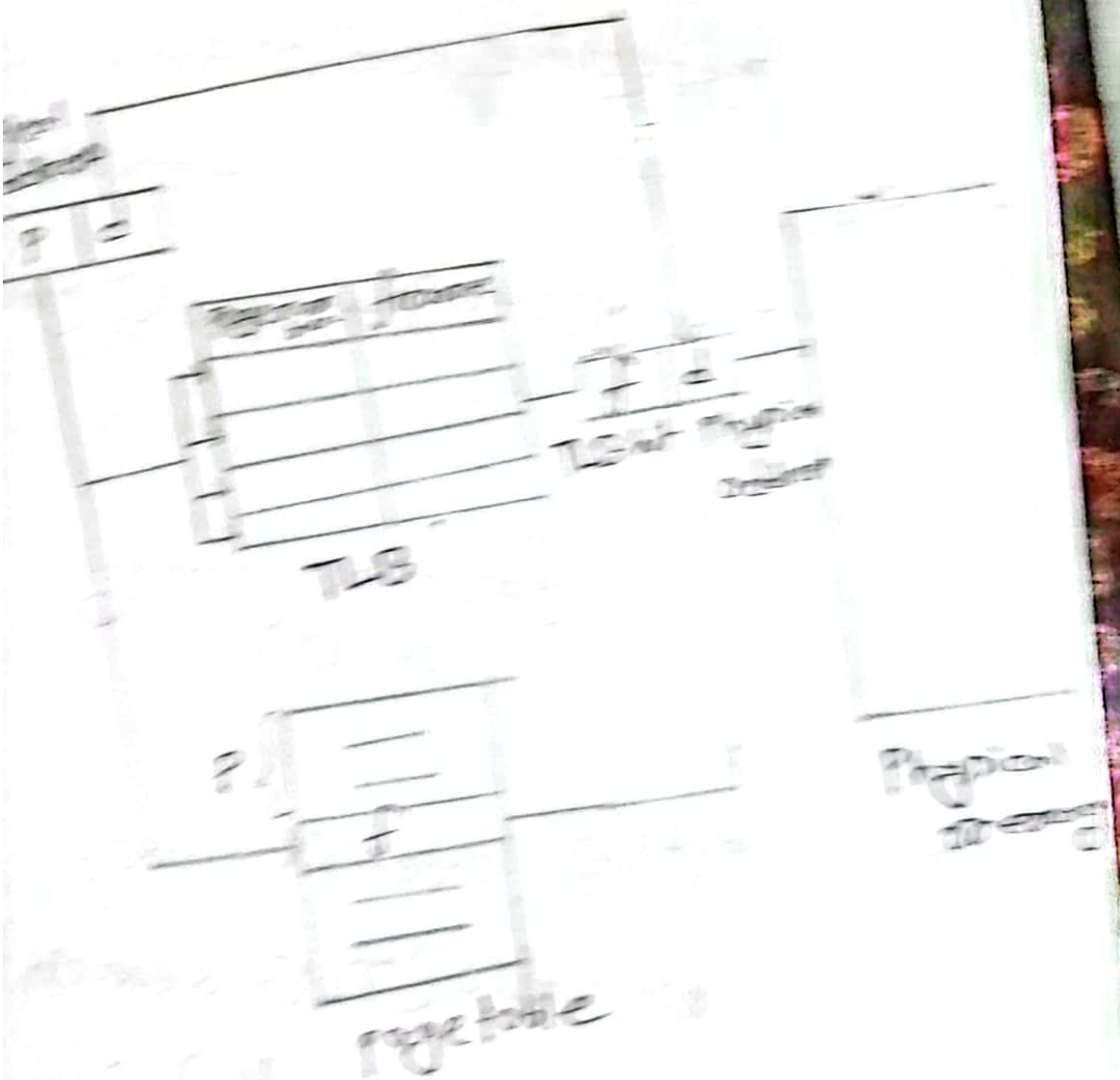
~~when~~ → When logical address is generated by the CPU, its page number is presented to the TLB.

TLB hit: If the page number is found, its frame number is immediately available and is used to access memory.

TLB miss: If the page number is not found, a memory reference to the page table must be made.

full of entries

an existing entry must be replaced



Percentage of times that the number of interest in food is called hit radio.

Formula

TLB access time = e

TLB hit ratio = x

TLB miss " = $(1-x)$

$\therefore$ Effective access time = hit ratio $(e+m)$ + miss ratio $(e+m+m)$

$$= x(e+m) + (1-x)(e+m+m)$$

for page table access

for main memory access

* effective access time =

hit ratio (TLB search time + memory access time)

+ (1 − hit ratio) (TLB search time + 2 memory access time)

Memory protection in paging:

accomplished by protection bits associated with each frame.

These bits are kept in the page table

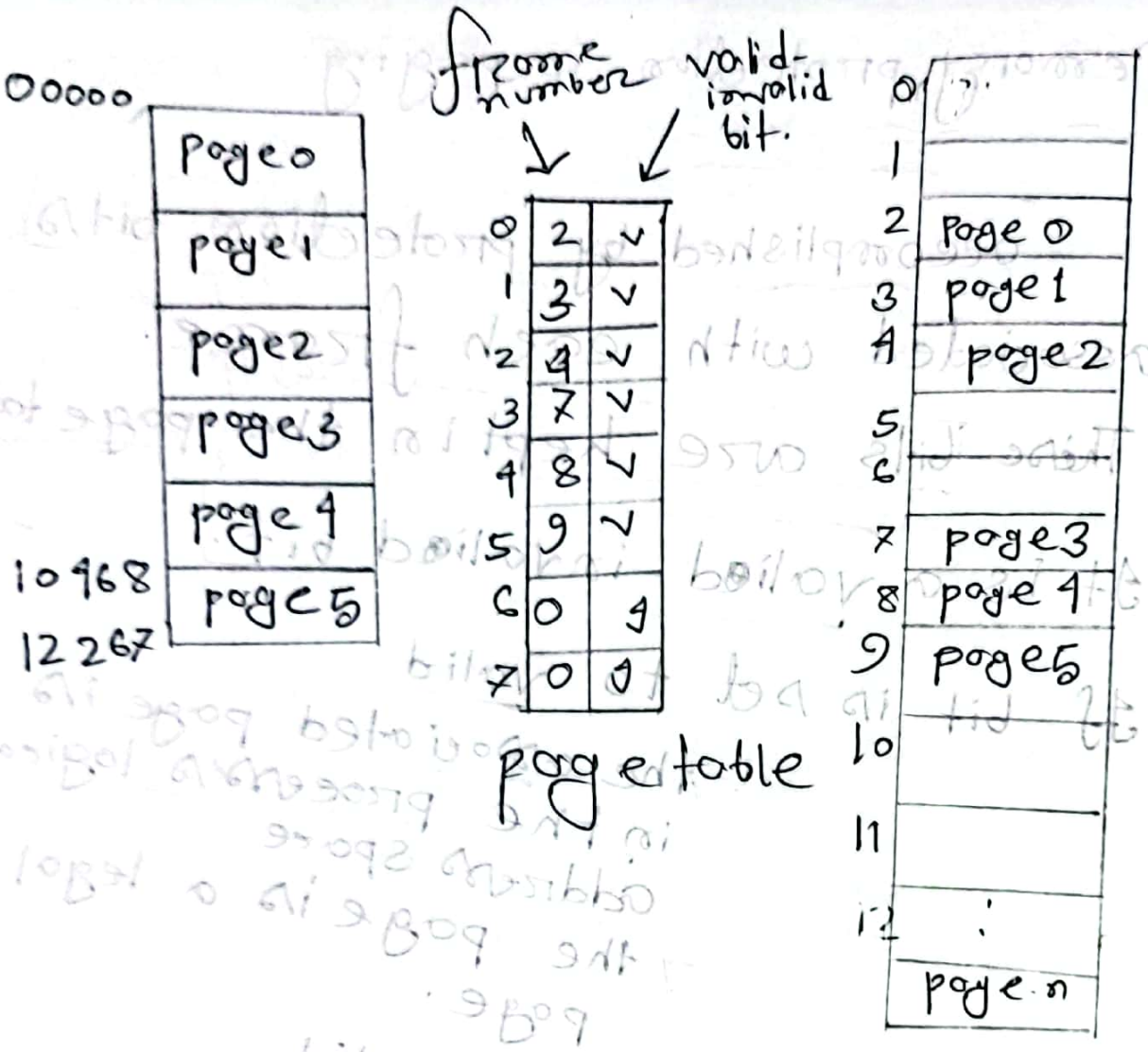
It is a valid-invalid bit.

If bit is set to valid
→ the associated page is in the process's logical address space
→ the page is a legal page.

If bit is set to invalid
→ page is not in the process's logical address space.
→ illegal addresses are trapped by use of valid-invalid bit.

☆ We can create h/w to provide
read-only
read-write
execute only protection.
by providing separate protection bit of each kind.

CamScanner



PTLR: Page Table Length Register used to indicate the size of the page table.

Shared page:

Common code is shared.

It is important in a time sharing environment. Compilar, window system, run-time libraries, database system also shared.

Example:

3 process sharing 3 page editor.

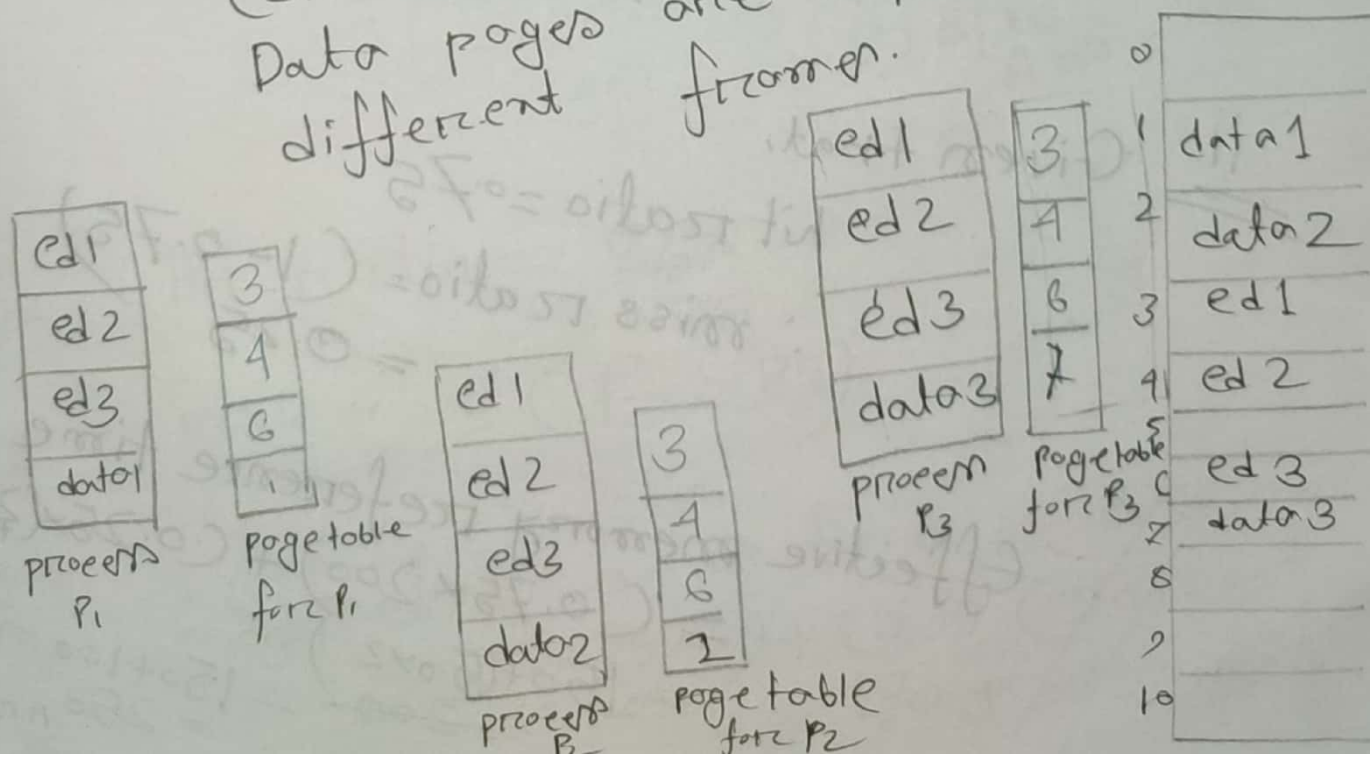
Each process has its own data page.

Reentrant code is non-self modifying code.

→ it never changes during execution.

Each user's page table maps onto the same physical copy of the editor.

Data pages are mapped onto different frames.



~~Given~~ Effective Access time

i Given that,

memory reference time = 200 ns

paged memory reference =)

paged memory reference

= access time for page table

+ access the word in memory

= (200 + 200) ns

= 400 ns

Ans

ii Given that,

hit ratio = 0.75

∴ miss ratio = (1 - 0.75)

= 0.25

∴ Effective memory reference time

= (0.75 × 200) + (0.25 × (200 + 200))

= 150 + (50 × 2) ns

= ~~200~~ = 150 + 100 ns

= 250 Ans

Given that,

memory access time = 200 ns

access to the page table = 200 ns

" to " TLB = 20 ns.

i ~~Effective~~ Hit ~~Hig~~ ratio = 95%

= 0.95

∴ miss " = (1 − 0.95)

= 0.05

~~memory~~ Effective access time = (0.95 × (200 + 20))

+ 0.05 (200 + 20)

= 209 + 21 = ~~(190 + 11) ns~~

= 230 ns Ans = 201 ns Ans

ii Hit ratio = 85%

= 0.85

∴ miss " = (1 − 0.85)

= 0.15

∴ Effective access time = (0.85 × 200)

= (0.85 × 220) + (0.15 (200 + 20)

170 + 33 = 203 ns Ans

Segment table (logical address → physical address

Segment	Base	Length
0	200	600
1	2100	14
2	70	100
3	1518	570

i 0, 330

∴ physical address

= (200 + 300)

= 500

Ans

ii 2, 12

∴ physical address

= (~~2100~~

= (70 + 12)

= 82

Ans

iii 3, 390

Physical address

= (1518 + 390)

= 1908 Ans

CamScanner

(logical address → Physical ")

Segment table

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

a. 0, 430

Physical address = (219 + 430)

= 649 Ans

b. 1, 10,

Physical address = (2300 + 10)

= 2310 Ans

c. 2, 500.

Illegal reference, trap to the operating system

d. 3, 400

Physical address = (1327 + 400)

= 1727 Ans

CamScanner

e 4.112

Illegal reference trap to the operating system.

Given that,

Partition no	memory size
1	100KB
2	500KB
3	200KB
4	300KB
5	600KB

process,

$P_1 \rightarrow 212KB$
$P_2 \rightarrow 417KB$
$P_3 \rightarrow 112KB$
$P_4 \rightarrow 426KB$

first fit

$P_1 \rightarrow 212KB$: partition 2 (500KB)
$P_2 \rightarrow 417KB$: partition 5 (600KB)
$P_4 \rightarrow 426KB$: must wait
$P_3 \rightarrow 112KB$: partition 2 (500-212=288)

CamScanner

Best fit

$P_1 \rightarrow 212KB$: partion 4 (300 KB)
$P_2 \rightarrow 417KB$: partition 2 (500KB)
$P_3 \rightarrow 112KB$: partition 3 (200KB)
$P_4 \rightarrow 426KB$: partition 5 (600KB)

Worst fit:

$P_1 \rightarrow 212$ KB : partition 5 (600KB)
$P_2 \rightarrow 417$ KB : partition 2 (500KB)
$P_3 \rightarrow 112$ KB : partition 4 ~~(300KB)~~ 388K = (600 − 212)
$P_4 \rightarrow 426$ KB : must wait

Best fit turns out to the best.

$m = n = \log_2 G$
$n = \frac{10}{\log_2 2} = 1$
$m = \log_2 8$

Given that,

Number of page = 64
page size = frame size = 1024 word
Number of frame = 32

∴ Number of bits in logical address

$$= \left\lceil \frac{\log(64 \times 1024)}{\log 2} \right\rceil$$

$$= \frac{\log 65536}{\log 2}$$

$= 16$ Ans

b. Number of bits in physical address $= \left\lceil \frac{\log(32 \times 1024)}{\log 2} \right\rceil$

$$= \left\lceil \frac{\log 32768}{\log 2} \right\rceil$$

$= 15$ Ans

Given that,

logical address space $= 64$ bit

page size $= 4KB$

$= 2^2 \times 2^{10}$

$= 2^{12}$

Given that,

memory access time = 100 ns

TLB search time = ~~200ns~~ 10ns

memory access for page table = 200 ns

I hit ratio = 90%
= 0.9

~~miltr~~ miss ratio = (1 − 0.9)
= 0.1

∴ EAT = 0.9 × (200+10) + 0.1 (200+10+200)
= 189 + 41
= 230 ns Ans

II hit ratio = 96%
= 0.96

miss ratio = (1 − 0.96)
= 0.04

EAT = (200+10) 0.96 + 0.04 (200+200+10)
= 201.6 + 16.4
= 218 ns Ans

CamScanner

Chapter- 9 (2)
Virtual Memory

FIFO

1	2	3	4	5	3	4	1	6	7	8	7	8	9	7	8	9	5	4	5	4	2
1	1	1	1	5			5	5	5	8			8				8	8			2
	2	2	2	2			1	1	1	1			9				9	9			9
		3	3	3			3	6	6	6			6				5	5			5
			4	4			4	4	7	7			7				7	4			4

Number of page foult = 13

OPR

1	2	3	4	5	3	4	1	6	7	8	7	8	9	7	8	9	5	4	5	4	2
1	1	1	1	1				6	6	8			8					8			8
	2	2	2	5				5	5	5			5					5			5
		3	3	3				3	7	7			7					4			4
			4	4				4	4	4			9					9			9

Number of page foult = 11

~~RRU~~

LRU

1	2	3	4	5	3	4	1	6	7	8	7	8	9	7	8	9	5	4	5	4	2
1	1	1	1	5			5	5	6	7			6				5	5			5
	2	2	2	2			1	1	1	1			9				9	9			9
		3	3	3			3	3	6	6			7				7	4			4
			4	4			4	4	4	8			8				8	8			2

LRU

1	2	3	4	5	3	4	1	6	7	8	7	8	9	7	5	9	5	4	5	2
1	1	1	1	5	3	3	5	6	6	6			6				5	5		5
	2	2	2	2		4	1	1	1	1			9				9	9		9
		3	3	3			3	3	3	7			7				7	4		
			4	4			4	4	4	8			8				8	8		

Given that,

effective access time = 200 ns

page fault time = 8 ms

memory access time = 100 ns

replaced to be modified = 20 ms

" " " = 70%

∴ page fault = (1-.70)

= 0.30

$$\therefore 200\times10^{-9} = \{(1-P)(100\times10^{-9})\} + (0.7P \times 20\times10^{-3})$$

$$+ (0.3P \times 8\times10^{-3})$$

$$= (1-P)(10^{-7}) + (1.4\times10^{-2})P$$

$$+ (2.4\times10^{-3})P$$

$$= 10^{-7} - 10^{-7}P + (1.4\times10^{-2})P$$

$$+ (2.4\times10^{-3})P$$

$$= 10^{-7} + 1.639\times10^{-2}P$$

$$\Rightarrow 10^{-7} = 1.639\times10^{-2}P$$

$$\therefore P = 0.000006$$

CamScanner

Chapter- 5
Process Synchronization

Race Condition:

A situation where several process access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called race condition.

Critical section:

Each process has a segment of code called critical section, in which the process may be changing common variables, updating a table, writing a file and so on.

☆ When one process is executing in its critical section, no other process is allowed to execute in its critical section.

CamScanner

do {

entry section

critical section

exit section

remainder section

} while (true);

Fig : General ~~solution~~ structure of ~~criti~~ a typical process pi.

Solution of critical section problem

Problem must satisfy the following sections:

1. Mutual exclusion

If process P_i is executing in its critical section, then no other process can be executing in their critical section.

2. Progress:

If no process is executing in its critical section and some process wishes to enter their critical section then.

→ Only those process that are not executing in their reminder section can participate in deciding which will enter its critical section

→ this section can't be postponed indefinitely.

CamScanner

Bounded waiting:

There exists a bound or limit on the number of times that other process are allowed to enter their critical section, after a process has made a request to enter its critical section and before the request is granted.

Synchronization process:

- → Peterson's Solution
- → Mutex Lock Solution
- → Semaphore
- → Monitor

Peterson's Solution:

→ restricted to two processes that alternate execution between their crit

~~alt~~ alterna

Peterson's Solution:

→ classical software base solution

→ restricted to two processes alternate execution between their critical section and reminder section.

CamScanner

Structure

```
do {
    flag[i] = true;
    turn = j;
    while (flag[j] && turn==j);
        critical section.
    flag[i] = false;
        remainder section
} while (true).
```

Mutex lock

→ Short for mutual exclusion

2 functions

→ 1) acquire() function → acquire the lock

2) release() " → release the lock

→ has a boolean variable → identifies if the lock is available or not

available

defination of acquire()

```
acquire() {
    while (!available);
    available = false;
}
```

defination of release()

```
release() {
    ~~while~~ available = true;
}
```

~~Soluti~~
~~Structure of mutex lock~~:

Structure:

```
do {
    [acquire lock]
        critical section
    [release lock]
        reminder section
} while (true);
```

Problem of mutex lock:

→ it requires busy waiting

→ ~~it prev~~

→ it results in a spinlock

→ spinlock waste useful cpu cycle, while busy waiting on the while loop.

```
acquire (){
    [while(!available);]
    available = false;
}
```

Advantages

If locks are expected to be held for very short period, then mutex lock is advantageous → as no context switch is required.

CamScanner

Semaphore:

→ Semaphore S is an integer variable, apart from initilization.

→ accessed only through two standard atomic operation

wait ()

signal ()

```
wait (S) {
    while (S<=0);
        S--;
    }

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

Code of the producer process

```
while (true) {
    ~~while (BufferSize~~
    while (counter == BufferSize);
        buffer[in] = next_produced;
        in = (in+1) % BufferSize;
        counter++;
}
```

Code of the consumer

```
while (true) {
    while (counter == 0);
    next_consumed = buffer[out];
    out = (out+1) % BufferSize;
    counter--;
}
```

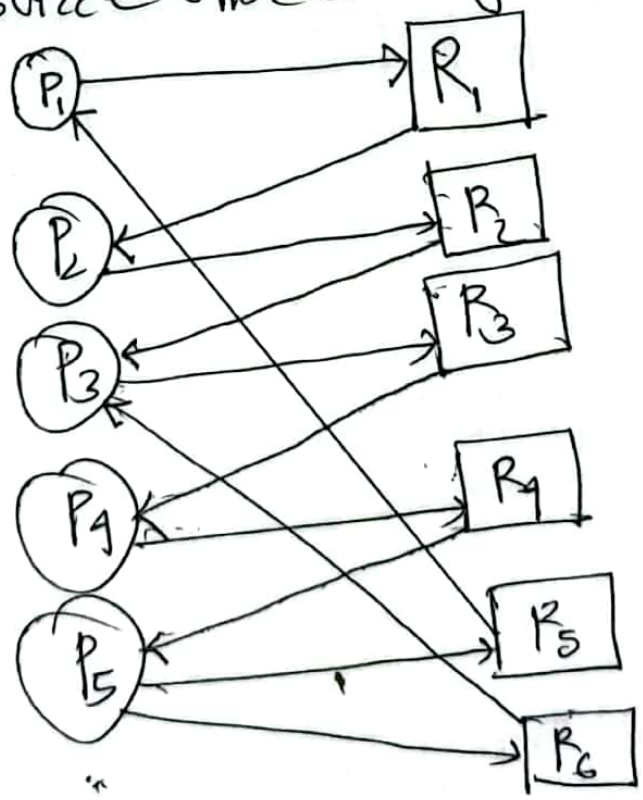
Chapter- 7 (3)
Deadlock

$R = \{R_1, R_2, R_3, R_4, R_5, R_6\}$

$P = \{P_1, P_2, P_3, P_4, P_5\}$

$E = \{P_1 \to R_1, R_1 \to P_2, P_2 \to R_2, R_2 \to P_3,$
$P_3 \to R_3, R_3 \to P_4, P_4 \to R_4, R_4 \to P_5,$
$P_5 \to R_5, R_5 \to P_1, P_5 \to R_6, R_6 \to P_3\}$

1 Resource allocation graph



Deadlock-banker

Process	Allocation A	B	C	D	Max A	B	C	D	Available A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

a. Need matrix : Max - Allocation

Process	A	B	C	D
P_0	0	0	0	0
P_1	0	7	5	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	6	4	2

b. Available = [1 5 2 0]

~~Step-1 AD Available~~

Let finish vector

Process	State
P_0	false
P_1	false
P_2	false
P_3	false

Step-1

As Finish[P_0] = false and Need$_{P_0}$ ≤ Available

∴ New available [1 5 2 0] + [0 0 1 2]

= [1 5 3 2]

∴ P_0 is allocated resource.

∴ Finish[P_0] = true

Step-2

As Need P_2 ≤ Available and Finish[P_2] = false

∴ New available = [1 3 5 3 2] +

[1 3 5 4]

= [2 8 8 6]

∴ P_2 is allocated resource.

∴ Finish[P_2] = true

Step-3

As Need P_3 ≤ Available and Finish[P_3] = false

∴ Available = [2 8 8 6]

+ [0 6 3 2]

= [2 14 11 8]

∴ finish[P_3] = true

Step-4

As Need$_{P_4}$ ≤ Available and finish[P_4] = false

Available = [2 14 11 8] + [0 0 1 4]

= [2 14 12 12]

finish [P4] = true

Step-5 Need $P_1 \leq$ Available and ~~Availa~~

~~Available~~ finish [P_1] = false

Available = [2 14 12 12] + [1 0 0 0]

= [3 14 12 12]

∴ finish [P_1] = true.

∴ Safe sequence,

< P_0, P_2, P_3, P_4, P_1 >

	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			

I. Need matrix = Max - Available

	Need		
	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

Ans

II. Available = [3 3 2]

Let finish[i] = false, for i = 0, 1, 2, 3, 4

Step-1

finish[P_1] = false and Need$_{P_1}$ ≤ Availa

∴ Available = [3 3 2] + [2 0 0]

= [5 3 2]

finish[P_1] = true.

Step-2

finish[P_3] = false and

Need$_{P_3}$ ≤ Available

∴ Available = [5 3 2] + [2 1 1]

1 4 3]

finish [P3] = true.

3 Need P4 ≤ Avail and finish [P4] = false.

∴ Available = [7 4 3] + [0 0 2]

= [7 4 5]

finish [P4] = true.

P-4 Need P0 ≤ Available & finish P0 = false

∴ Available = [7 4 3] + [0 1 0]

= [7 5 3]

∴ finish [P0] = true

P-5 Need P2 ≤ Available & finish P2 = false

∴ Available = [7 5 3] + [2 0 0]

= [9 5 3]

finish [P2] = true.

∴ As finish [i] = true i = 0, 1, 2, 3, 4

So system is safe.

Safe sequence

< P1, P3, P4, P0, P2 >

	Max Need				Allocation				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	6	0	1	2	4	0	0	1	3	2	1	1
P_1	1	7	5	0	1	1	0	0				
P_2	2	3	5	6	1	2	5	4				
P_3	1	6	5	3	0	6	3	3				
P_4	1	6	5	6	0	2	1	2				

Soln

1. Resources of type A $= (4+1+1)+(3)$
$= 9$

" of " B $= (1+2+6+2)+(2)$
$= 13$

" of " C $= (5+3+1)+(1)$
$= 10$

" of " D $= (1+4+3+2)+(1)$
$= 11$

II Need matrix

	A	B	C	D
P_0	2	0	1	1
P_1	0	6	5	0
P_2	1	1	0	2
P_3	1	0	2	0
P_4	1	4	4	4

III Let, finish[i] = ~~true~~ false

Step 1: Need $P_0 \leq$ Available

$\therefore$ Available $= [3\ 2\ 1\ 1] + [4\ 0\ 0\ 1]$

$= [7\ 2\ 1\ 2]$

$finish_{P_0} = true$

Step-2

Need$P_2 \leq$ Available

$\therefore$ Available $= [7\ 2\ 1\ 2] + [1\ 2\ 5\ 4]$

$= [8\ 4\ 6\ 6]$

$finish_{P_2} = true.$

Step-3 Need $P_3 \leq$ Available

$\therefore$ Available $= [8\ 4\ 6\ 6] + [0\ 6\ 3\ 3]$

$= [8\ 10\ 9\ 9]$

$finish_{P_3} = true$

CamScanner

Need $P_4 \leq$ Available

$\therefore$ Available $= [8 \ 10 \ 9 \ 9] + [0 \ 2 \ 1 \ 2]$

$= [8 \ 12 \ 10 \ 11]$

finish P_4 = true

Step-5 Need $P_1 \leq$ Available

$\therefore$ Available $= [8 \ 12 \ 10 \ 11] + [1 \ 1 \ 0 \ 0]$

$= [9 \ 13 \ 10 \ 11]$

finish P_1 = true

$\therefore$ AD finish $[i]$ = true for $i = 0, 1, 2, 3, 4$

$\therefore$ System is safe.

Safe sequence,

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

IV P_4 arrives for additional resources of $(1, 2, 0, 0)$

$\therefore$

$\therefore$ Request $P_4 \leq$ Need P_4 and Available $\geq$ Request

$\therefore$ Available = Available − Request

$= [3\ 2\ 1\ 1] - [1\ 2\ 0\ 0]$

$= [2\ 0\ 1\ 1]$

~~Need =~~ Need (Need$_i$ − Request$_i$)

	A	B	C	D
P_0	2	0	1	1
P_1	0	6	5	0
P_2	1	1	0	2
P_3	1	0	2	0
P_4	0	2	4	4

Allocation (Allocation$_i$ + Request$_i$)

	A	B	C	D
P_0	4	0	0	1
P_1	1	1	0	0
P_2	1	2	5	4
P_3	0	6	3	3
P_4	1	4	1	2

finish [i] = false , i = 0 ... 4

Step-1

$Need_{P_0} \leq Available$

$\therefore Available = [2\ 0\ 1\ 1] + [4\ 0\ 0\ 1]$

$= [6\ 0\ 1\ 2]$

$finish_{P_0} = true$

Step-2

$Need_{P_2} \leq Available$

$\therefore Available = [6\ 0\ 1\ 2] + [1\ 2\ 5\ 4]$

$= [7\ 2\ 6\ 6]$

$finish_{P_2} = true$

Step-3

$Need_{P_3} \leq Available$

$Available = [7\ 2\ 6\ 6] + [0\ 6\ 3\ 3]$

$= [7\ 8\ 9\ 9]$

$finish_{P_3} = true$

Step-4

$Need_{P_4} \leq Available$

$\therefore Available = [7\ 8\ 9\ 9] + [1\ 4\ 1\ 2]$

$= [8\ 12\ 10\ 11]$

$\therefore finish_{P_4} = true$

Step-5

$Need_{P_1} \leq Available$

$Available = [8\ 12\ 10\ 11] + [1\ 1\ 0\ 0]$

$= [9\ 13\ 10\ 11]$

$\therefore finish_{P_1} = true$

CamScanner

system is safe

safe sequence,

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

∴ request is granted.

Ans

→ access physical address by using logical address

* Memory management technique
  - → segmentation → segment table
  - → paging → page table

→ segmentation

S	d

← logical address

* paging

CPU

page | p | d | ← logical address

frame | f | d | ← physical address

↓ MMU

} fixed size

* logical address → m bit
* physical " → n bit

p	d

high order (m-n) low order n

logical address → physical address
→ from segment table
→ " page "

~~* TBI~~

* TLB → cache memory → $\boxed{\text{\# page} \mid \text{value}}$

* TLB hit ratio = x
* TLB miss " = $(1-x)$
* effective access time = ~~$(1-P)$ ma + P~~ $x(e+m) + (1-x)(e+2m)$

e → TLB access time

m → main memory access time

$2m$ → page table access time

* TLB hit time

TLB miss time