Binita Khan -19301145

Lecture-01

19301145

* Why we use operating system?

⇒ Intermidiary & negotiative between hardware and user.

* What is an operating system?

A program that acts as an intermediary between a user of a computer and the computer hardware.

Goals →

(i) Execute user programs and make solving user problems easier and make the computer system convinient to use.

(ii) Use the computer hardware in an efficient manner.

Primary functions :

(I) Processes - management
(II) Storage - memory management (Ram)
(III) Data - file management
(IV) Input/Adput devices - i/o management
(V) Network management
(VI) Protection & security.

Date

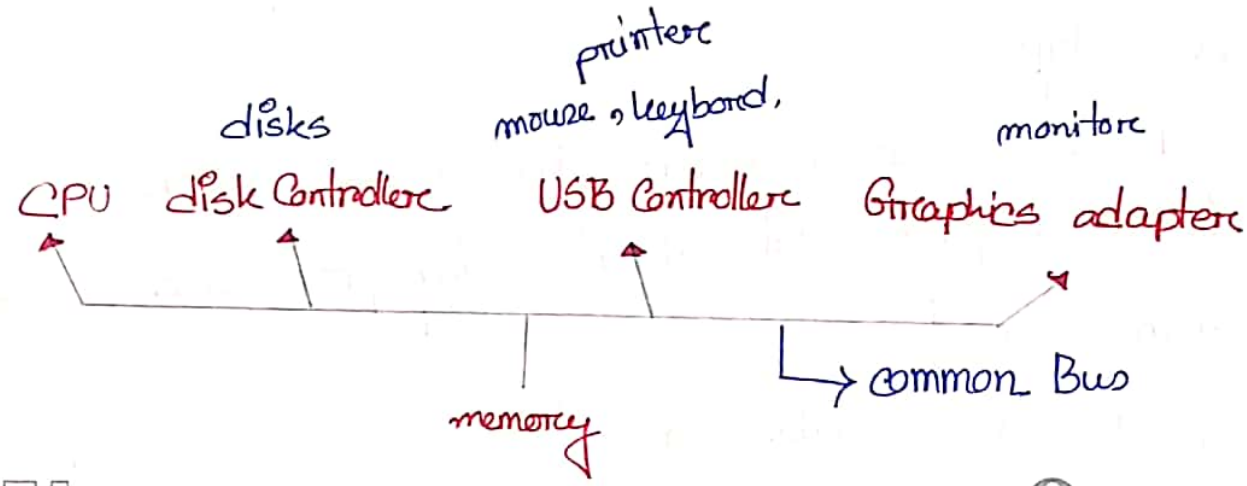
Computer system can be devided into four components—

Hardware :
provide basic computing resources. [CPU, memory, I/O dev

Operating System :
Controls and coordinates use of hardware among various applications and users.

Application Program :
Define the ways in which the system resources are used to solve the computing problems of the users.
Ex- word processors, compilers, web browsers, database systems, video games.

Users :
People, Mechine, other computers.



19301145

Computer System Operation :

a. One or more CPUs, device controllers connect through common bus providing access to shared memory.

b. Connects Concurrent execution of CPUs and devices for memory cycles.

c. To ensure orderly access to shared memory, a memory controller is provided whose function is to synchronize action to the shared memory.

Computer System Operation :

a. I/O devices and the CPU can execute concurrently.

b. Each device controller is in charge of a particular device type.

c. Each device controller has a local buffer. [store data in a buffer while it is being processed to/~~from~~ transferred]

d. CPU moves data from/to main memory to/from local buffers.

e. Device controllers informs CPU that it has finished it's operation by causing an interrupt.

Device Drivers & Controllers:

a. OS has a device drivers to each device controller.

b. Device drivers understand the device controllers and provide a uniform interface.

Bootstramp Program:

Bootstramp program it is loaded at powers-up or reboot

a. Typically stored in ROM or EPROM, generally knows as firmware.

b. Initializes all aspects of system.

c. Loads operating system kernal and starts execution.

19301145

Lecture – 1.2

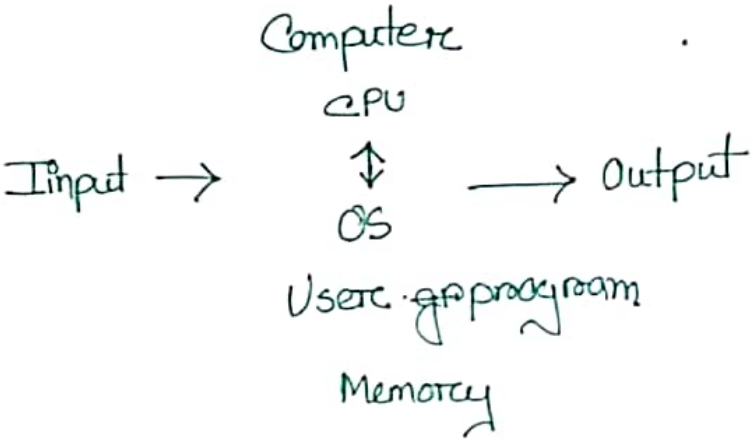
19301145

Types of O.S →

(i) Simple Batch System
(ii) Multi programming system
(iii) Time – Sharing system
(iv) Parallel System
(v) Distributted system
(vi) Real – Time System

Simple Batch System :

- Automatically transfer control from one Job to another
- Common i/o devices are card reader and tape drivers.
- Users prepare a Job which consisted of program, input data and control instruction.



Job execution instead of progra execution.

Program
↓
Input Data
↓
Control Instruc -ion

Job

Date

19301145

Limitation :

(i) Not interactive

(ii) Memory is limited

(iii) CPU Utilization is poor

(iv) Speed mismatch between I/O devices & CPU.

Note : Mechanical I/O device were very slow than the CPU. Even a slow CPU working in micro second rate and difference of speed between the CPU & I/O devices may be 3 orders of magnitude or more.

User prepares his job on an off-line device like punch card and submits it to the computer operator To speed up processing, job like similar needs are batched together and run as a group.

Advantages :

(i) It saves the time that was wasted earlier for each individual process in context switching from one enviornment to another environment.

(ii) No manual intervention is needed.

Disadvantages:

(i) Executing a series of non-interactive jobs all at one time.

(ii) The output is obtained only after all the jobs are executed.

(iii) Thus, priority can not be implemented if a certain job has to be executed on an urgent basis.

Multi-Programming Operating System:

- Multi programming needed for efficency or maximize utilization.
- Multi programming means more than one process in main memory which are ready to execute.
- Single user can not keep CPU & and I/O devic busy at all times.
- Multi programming organizes job (cod/data)

so CPU always has to execute.

- A subset of total jobs in system is kept in memory.
- Process generally requires CPU time and I/O time, when CPU has to wait (for I/O for example) OS switches to another job and idea will continue.

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

Advantages:

(i) High CPU utilization

(ii) It appears that many programs are ~~allo~~ alloted CPU almost concurrently.

(iii) Response time is shorter

Disadvantages:

(i) CPU scheduling is required.

(ii) To accomodate several jobs in memory, memory management is essential.

Multi-programmed systems provide an enviornment in which the various system resources [like- CPU, memory, and peripheral devices] are utilized effectively but they donot provide for users interactions with the computer system.

Lecture : 1.3

What is process?

⇒ A process is actually a program in execution.

Advantages :

(i) Increased throughput

(ii) Economy of scale

(iii) Increased realibility

19301145

Date

Lecture – 14

19301145

Storage Device Hierchy:

(1) Register
(2) Cache Memory
(3) Main Memory
(4) Electronic Disk
(5) Magnetic Disk
(6) Optical Disk
(7) Magnetic tapes.

* Storage system organized in hierchy
  (1) Size (↑)
  (2) Speed / Access time (↓)
  (3) Cost (↓)
  (4) Volality

* Caching: Information in use copied from slower to faster storage temporarily.

19301145

Operating System Services:

1. User Interface
2. Program execution
3. I/O operations
4. File System Manipulation
5. Communication
6. Error Detection
7. Resource Allocation
8. Accounting
9. Protection & Security.

'Week-02'

Introduction & O.S Structures

Two types of interrupt:

(i) Hardware generates interrupt
(ii) Software errors handled by exeption or trap.

[Modern O.S are interrupt driven]

With sharing many process could be adversely by a bug in one program. Sol: Since the operating system and the user's program share the software & hardware resources of the computer system, a properly designed OS must be ensure that an incorrect program can not run and also can not cause other programs to execute incorrectly.

19301145

Date

Hardware Protection :

* This approach taken by most computer system to provide hardware support that allows us to differentiate among various modes of execution.
* Dual Mode operation allows OS to protect itself and and other system components.
* User Mode (1) and Kernel/Monitor/System Mode (0)
* Mode Bit provided by hardware.

System Call :

* A system call is a way for programs to interact with the operating system.
* A computer program makes a system call when it makes a request to the operating system's kernel. System call provides the services of the operating system to the user programs via Application program Interface (API).
* Programming interface to the services provided by the OS
* Routines typically written in a high level language [C or C++]

19301145

Process

Process: The process is the unit of work in a modern time-sharing system.

*The call of activities of CPU :→

Job → Batch System

User Program / Task → Time sharing System

These activities are called Process

* The term 'jobs' and process are used almost interchangeably.

* A process is a program that is in execution. But it is more than the programs code. Program code is known as "text section" of a process.

Beside the code of the program, it contains →

1. Program Counter & Register
2. Stack
3. Data section
4. Heap

Program is the Blueprint

19301145

*States of Process:

(i). New
(ii). Running
(iii). Waiting
(iv). Ready
(v). ~~Term~~ Terminated

PCB [Process Control Block]: PCB is data structure to store information of a process. Such →

(1). Process State
(2). Program Counter
(3). CPU register
(4). CPU scheduling information
(5). Memory Management information
(6). Accounting Information
(7). I/O status Information

Week-03

Process Scheduling

The job of the operating system is to execute processes in the computer. All the processes stored in the memory and those processes comes from memory into the processor of the operating system. The processors are executed in the process
-or.

Scheduling Queue

1. Stores the processes in different steps of OS.
2. Different queues are ~~men~~ maintained in different steps. [The processes follow some steps to be
3. executed. And the normal stage of the processes are stored in the memory (secondary). After that, those processes come into the the primary memory, then those processes goes to processors or the CPU. So, processes passes several steps to be executed.

Processes → ~~Primary~~ Secondary Memory → Primary Memory → CPU/ Main Processors → Ready to execute

all processes

3. Job queue (Secondary Memory) → Ready Queue (Main Memory) → CPU → Device Queue

waiting to be executed

19301145

Schedulers :

Two type of schedulers are available.

1. Long term → pick from Job q to ready q
2. Short term → pick from ready to CPU [More frequently executed]
3. Medium term (Swapping)

*Long term degree of multiprogramming

Long term scheduler select (CPU bound / I-O bound) process wisely to make waiting time shorter.

Swapping reduce the degree of multiprogramming.

Context Switch

Two steps

① Storing currently executed process
② Restoring the next process context to execute.

19301145

Week-06

CPU Scheduling

[Exponential Average]

* SJF depends on brust time.

Exponential Averaging:

1. t_n = actual length of n^{th} CPU Brust
2. τ_{n+1} = predicted value for CPU Brust
3. α, $0 \leq \alpha \leq 1$
4. $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$

* Commonly α set to $1/2$.
* Preemptively version called Shortest Remaining time first.

"Priority Scheduling"

* priority number must be integer.
* smaller integer = higher Priority
* problem → Starvation
* Solution → Aging

Date

Round Robin:

* time quantam q (10–100) miliseconds
* No process waits more than (n-1) q time units.
* q large → FIFO
  q small → q must be large with respect to context switch, otherwise overhead is too high.

Multilevel Queue:

Ready queue is partitioned into separate ques →

(I) Foreground (Interactive)
(II) Background (Batch)

Highest Priority → System Process
Interactive Process
Interactive Editing Process
Batch Process
Lowest Priority → Student Process

* Multilevel feedback queue.

19301145

CPU Schedulers

[Extended]

* Non preemptive Vs Preemptive

⇒ Non preemptive → if any process is occurring, then it will continue until it's being finish.

⇒ Preemptive → if any process arrives with heighest priority, current process will leave it's the process and give chance the heighest one.
[If pause, problem may occure]

Scheduling Criteria:

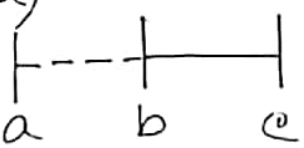
(I) CPU utilization → keep cpu busy

(II) Throughput → process that complete their execution per time unit.

(III) Turnaround time → amount of time to execute a particular process (c − a)

(IV) Waiting time (b − a)

(V) Response time (b − a)

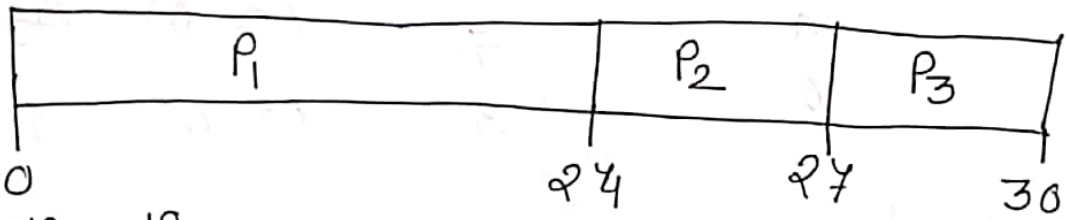


First Come first Serve (FCFS):

Date

Brust time: Number of time ~~are~~ (CPU time) required by a process.

Process	Burst Time	W.T	T.A
P_1	24	0-0 = 0	24-0=24
P_2	3	24-0 = 0	27-0=27
P_3	3	27-0 = 0	30-0=30



Avg Waiting time

$$\frac{0+24+27}{3} = 17$$

Avg Trwion Around time

$$\frac{24+27+30}{3} = 27$$

* Convoy effect

Shortest Job First (SJF):

two schemes: (i) Non preemptive [SJF]

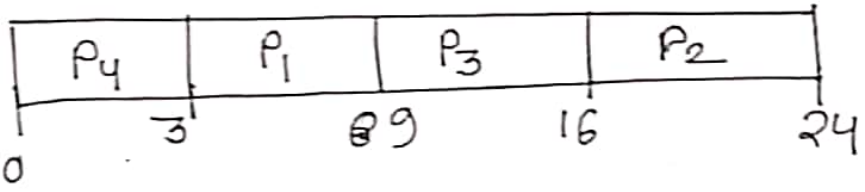
(ii) Preemptive [SRTF]

* SJF is optimal

Example [Non Preemptive]

Process	Arrival Time	Burst Time	WT
P_1	0	6	(3-0) = 3
P_2	0	8	(16-0) = 16
P_3	0	7	(9-0) = 9
P_4	0	3	(0-0) = 0

Scheduling Chart



T.A
(9-0) = 9
(24-0) = 24
(16-0) = 16
(3-0) = 3

$$\text{Avg WT} = \frac{3+16+9+0}{4} = 7$$

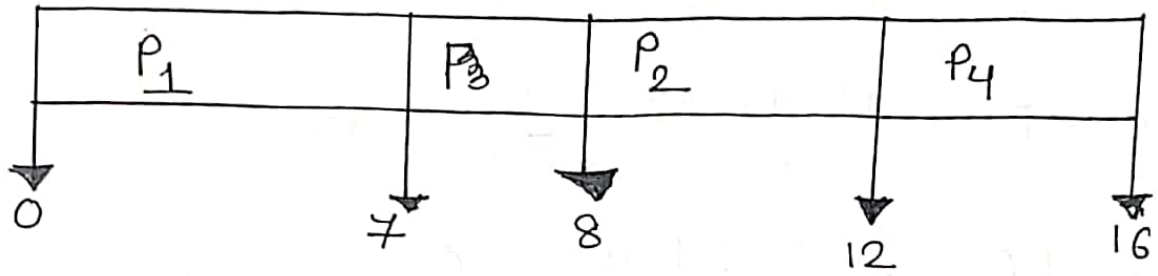
$$\text{Avg T.A} = \frac{9+24+16+3}{4} = 13$$

Example -2

Date :

Pn	A.T	B.T	W.T	T.A
✓ P_1	0	7	(0-0) = 0	7-0 = 7
P_2	2	4	(8-2) = 6	12-2 = 10
✓ P_3	4	1	(7-4) = 3	8-4 = 4
P_4	5	4	(12-5) = 7	16-5 = 11

Scheduling Chart

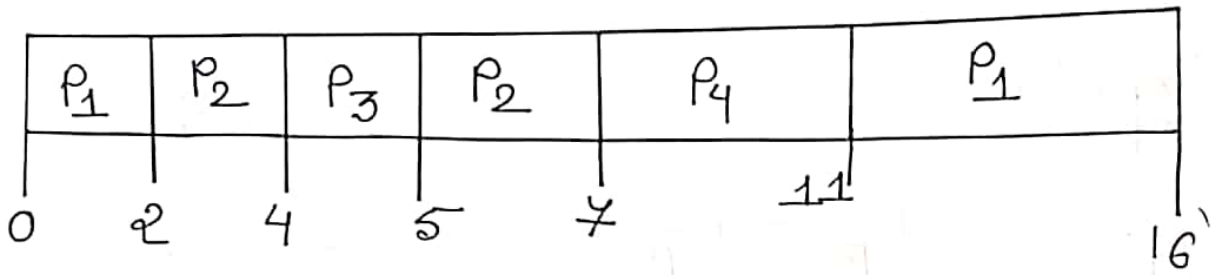


$$\text{Avg WT} = \frac{0+6+3+7}{4} = 4$$

$$\text{Avg T.A} = \frac{7+10+4+11}{4} = 8$$

Example → Preemtive SJF :

P	Ar.T	B.T	WT	T.A
P_1	0	~~4~~ 5 0	(0-0) + (11-2) = 9	16-0 = 16
P_2	2	~~4~~ 2 0	(2-2) + (~~7~~ 5-4) = 1	~~4~~ 7-2 = ~~8~~ 5
P_3	4	~~1~~ 0	(4-4) = 0	5-4 = 1
P_4	5	~~4~~ 0	(7-5) = 2	11 - ~~7~~ 5 = ~~4~~ 6



$$\text{Avg WT} = \frac{9+1+0+2}{4} = \frac{12}{4} = 3$$

$$\text{Avg T.A} = \frac{16+5+1+6}{4} = \frac{26}{4} = 6.5$$

Priority Scheduling :

(i) Integere

(ii) smallest int ≡ heighest priority

(iii) priority ∝ $\frac{1}{\text{predicted next CPU broust time}}$

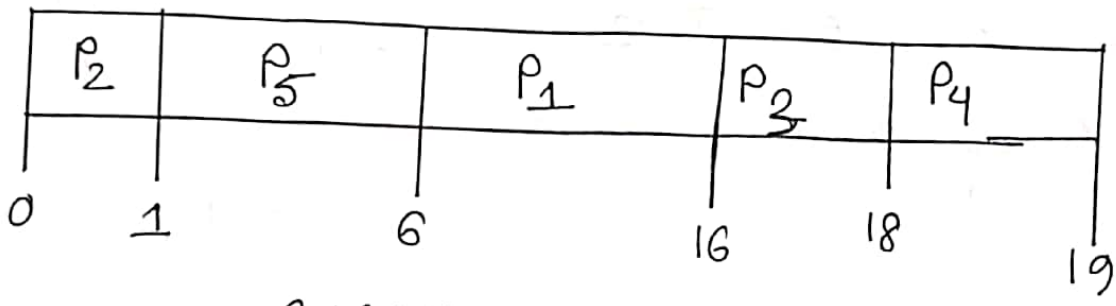
(iv) Priority (Internally or Externally)

Example (Priority Scheduling)

arv + turn around compu

Pn	B.T	Prio	W.T	T.A
P_1	10	3	(0 − 6) = 6	16 − 0 = 16
P_2	1	1	(0 − 0) = 0	1 − 0 = 1
P_3	2	4	16 − 0 = 16	18 − 0 = 18
P_4	1	5	18 − 0 = 18	19 − 0 = 19
P_5	5	2	(1 − 1) = 0	6 − 0 = 6

Scheduling



$$\text{Avg WT} = \frac{6+0+16+18+1}{5} = 8.2$$

$$\text{Avg T.A} = \frac{16+1+18+19+6}{5} = \cancel{9.6}\ 12$$

Problem

① Starvation → low priority process may never execute

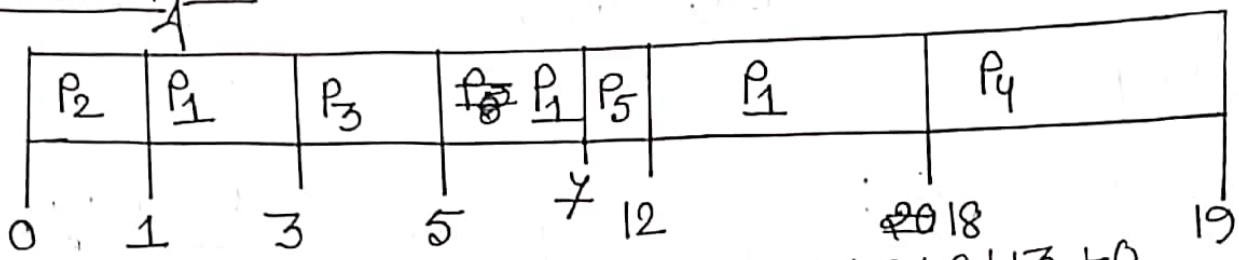
Solve

① Aging → as time processes increase the priority of the process

Example → Priority (Preemtive) :

Pro	AT	BT	Prio	WT	T.A
✓ P_1	0	~~10~~ 8 6	4	(1-0) + (5-3) + (12-7) = 8	18-0 = 18
✓ P_2	0	1	1	(0-0) = 0	1-0 = 1
✓ P_3	3	2	3	(3-3) = 0	5-3 = 2
P_4	5	1	5	18-5 = 13	19-5 = 14
✓ P_5	7	5	2	7-7 = 0	12-7 = 5

Scheduling —



Average Waiting time = $\frac{8+0+0+13+0}{5}$

$= \frac{21}{5} = 4.2$

Average Turnaround time = $\frac{18+1+2+14+5}{5}$

$= 8$

19301145

Date

Round Robin

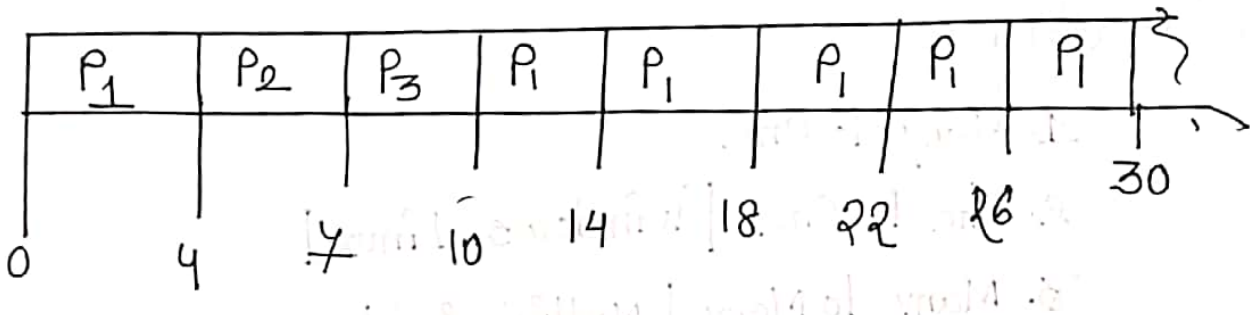
(i) Time quantum = q

(ii) If n process, q time quantum then,
each process gets $1/n$ of CPU time.
No process will wait more than $(n-1)q$ t.u

Example

	B.T	WT	T.A
P_0	~~8~~ 4 ~~0~~	~~30-0~~ $(0-0)+(10-4)=6$	$30-0=30$
P_1	~~24~~ ~~20~~ ~~16~~ ~~12~~	$4-0=4$	$4-0=4$
P_2	3	$7-0=7$	$10-0=10$
P_3	3		

The Giroantt chart



Avg TW $= \frac{(6+4+7)}{3} = \frac{17}{3}$

Avg T.W $= \frac{7+10}{3} = \frac{30+7+10}{3} = \frac{47}{3}$

THREAD

* Difference between task parallelism and data parallelism is, different task is occuring in a /for a particular data in different core is task parallelism but in same task is occuring for a particular data in a different/ multiple core is known ~~do~~ as data parallelism.

Data parallelism → Deviding the data for a single task.

Task parallelism → Same data in multiple core for multiple task.

Amdahl's Law:

$$\text{Speedup} \leq \frac{1}{S + \frac{1-S}{N}}$$

P → parallel
S → serial
N → processing course.

Multithreading:

1. Many to One
2. One to One [Windows, Linux]
3. Many to Many [Multiplexing]

Two Level Multithreading Model → Special type of second level Model [1 to 1, M, L]

19301145

Date :

Pthreads

1. User Level / Kernel Level
2. POSIX Standard API → thread creation + Synchronization
3. Specification, not implementation

Java Threads

1. Managed by JVM
2. Underlying OS
3. Can be created by —
   i. Extending thread class
   ii. Implementing on runnable interface
   [Standard Practice]

Implicit Threads

Methods →

1. Thread pull
2. Fork Join
3. Open MP
4. Grand Central Dispatch
5. Intel Threading Building Blocks.

19301145

19301145

Process Synchronization

Producer-Consumer Problem:

(I) Modify the algorithm to remedy this deficiency - add an integer variable as counter, initialized as to 0.

(II) add new item in buffer → counter++

(III) remove a item from buffer → counter--

In machine Language:

c++:

$r_1 = c$

$r_1 = r_1 + 1$

$c = r_1$

c--:

$r_2 = c$

$r_2 = r_2 - 1$

$c = r_2$

If these two process works parallely then it might be messed as it is shared variable.

To avoid this situation one process has to wait until another is on process and this system is known as process synchonization.

19301145

Critical Section

Critical section : Segment of code of each process, which may change common variables, update a table write a file and so on. Three section —

(I) Entry section — Implementing critical section execution
(II) Exit section — existing from critical section
(III) Remainder section - Remaining code of the program

Solution :

(I) Mutual exclusion
(II) Progress
(III) Bounded waiting

Two general approach to handle critical section in OS.

(I) Preeptive kernel allows a process to be preempted while it is running in kernel mode. [Free from race condition]

(II) Non-preemtive kernel : a kernel mode process will run until it exits kernel mode, blocks or volunatri-ly yeilds control of CPU [Free from race condition]

19301145

* Peterson Solution (Software Based)

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

* Each statement takes 2ms to execute.
* Context switch will occure after 6ms.

* Test and Set (Hardware Based)

(i) Executed automatically.

(ii) Implemented by initializing a boolean variable "lock" to false.

a. *target → rv
b. check and modify
c. *target → true (update true from false)
d. return → rv

↓
other keeps in waiting

* Compare & Swap (Hardware Based)

(I) Declare a global variable "lock" initialize it 0.

(II) Process invokes this 1, so other process keeps in waiting

(III) 3 variables.

a. * Value
b. expected (int)
c. New_value.

* MUTEX Lock (Hardware Based)

(1) Simplest one (Mutual Exclusion)

(2) Acquire lock before entering CS [acquire() function]

(3) Release the lock after exiting CS [release()]

(4) Variable "available" indicates the lock's availablity.

(5) Acquare = false ; Release = True.

if available = true if available = false

SEMAPHORE:

(1) S is an integer variable
2. Two atomic (standard) operations → wait(), signal()
3. if, $S \leq 0$ then other process should wait
4. if not, $S > 0$ then S--
5. signal (S) → S++ (No other job)

* Two types of semaphore.
  - (a) Counting semaphore (1-∞/n)
  - (b) Binary semaphore (0 or 1) [Similar to MUTEX Lock]

(*) It use to control access to given resource of finite number of intances.

(*) Solve various synchronization problems

a. Counting semaphore
(i) S=n [S initialize to number of resources / processes]
(ii) Each process get 1. S = S-1 (get)
(iii) Release, S=S-1 [Call signal)
(iv) S=0 means all resources are used
(v) process will block resources until it S=0.

b. Binary Semaphore :

Date :

1. Execute parallaly but we have to make sure that S_1 can not execute before S_2 or vice versa.

2. a variable 'sync' – 0 (initialize) to mentain (1)

Problems :

(I) Deadlock : If two processes waits for other to complete or two process wait it indefinitely for an event.

(II) Starvation : Process wait indefinitely within the semaphore. And, if we remove processes from the list associated with a semaphore in LIFO (Last in 1st out) order.

Semaphore Implementation :

* list
* 3 part
  a. Defination of semaphore (value, list)
  b. " " Wait (block())
  c. " " Signal (wakeup)

Block → remove from CPU
Wakeup → back to CPU

19301145

Application of Semaphore.

Bounded Buffer Problem:

(I) Shared fixed data or size buffer used as a queue.

(II) Producers produce piece of data from the buffer.

(III) Consumer: removes data from buffer.

Concern 1: producer won't produce data when buffer is full.

Concern 2: Consumer won't remove data when the buffer is empty.

Deadlock occurs in this case (May occur)

* If producer can't give an wake up call to the consumer.

* It could be solved by with semaphore.

(I) Count Full (return the number of available space)

(II) Count Emply (return available space)

* wait function এর কাজ হচ্ছে semaphore এর value যদি <0 হয় তবে সে প্রসেসকে wait করাবে। অর্থাৎ, process টি কে busy waiting / list এ add করবে।

Reader-Writers problem :

Two types of variation—

(I). First variation (No readers will kept waiting unless any writers got the permisson)

(II) Second variation (Once a writers is ready, ASAP it will start the work)

"May occure starvation"

1st case → writers may starve

2nd case → readers may starve

Solution — Semaphore

2 synchonization

(I) Among readers

(II) Among readers & writers.

Resource Allocation Graph

P to R → request edge
R to P → Access edge

* Easy to find deadlock
  (I) Cycle → posibleity of deadlock
  (II) No cycle → No deadlock

* If processes of a resource has further request then there must be a deadlock otherwise the cycle hasn't deadlock.

Method to Handle Deadlock:

(I) Prevention
(II) Avoidance
(III) Detection and recovery

Deadlock Prevention

① Mutual exclusion
② Hold and wait (low utilization of resources, starvation)
③ No preeption
④ Circular wait

We have to break atleast one condition to prevent deadlock.

19301145

Date

Banker's Algorithm

* Need = Allocation – ~~nee~~ Available
* Max = (total intance – total Allocation) of a ~~certai~~ particular resource.

Two arrays

(I) finish → initially false

(II) Work → initally work = available

Algorithm :

```
if finish[i] == false && Need <= work {
    Work = Work + Allocation;
    finish[i] = true
} else {
    wait
}
```

~~Req~~ Resource – Req Algo (P_i)

(I) $P_i \leq$ Need (true)

(II) $P_i \leq$ Availabe (true)

if (I) && (II) both are true

Available = Available – P_i

Allocation = Allocation + ~~need~~ P_i

Need = Need – P_i

ii, after these processes we can assume that we can find safe sequence then we can add pi.

Deadlock Detection & Recovery

① Neither prevented nor avoided → deadlock occurs

System must provide

a. Deadlock algorithm

b. Recovery scheme

* if finish has any false in it's array then that have deadlock, otherwise it won't be in deadlock state.

Recovery:

a. Abort one or more process to break the circular wait.

b. preempt some resources

Date

Main Memory

1. CPU access → Main memory + Registers
2. Stream of address + read request or, address + data & write req

Range = base to base + limit

if, (add ≥ base) && (add < base + limit)

then access the memory

* Only os can operate base & base + limit process can not.

Address Binding :

HDD – secondary Memory

HDD → Memory → CPU

Symbolic → Relocatable → Absolute

Stages

(I) Compile time. (Symbolic to Absolute)

(II) Load time (Relocatble must)

(III) Execution time (Base & limit register)

19301145

Logical Vs Physical Address Space

Logical address → generate by CPU.
also refered as virtual address.

Physical address → address seen by the memory unit (Absulate address).

Logical address + Physical address → same in compile time & load time binding scheme but different in execution time

Memory Management Unit (MMU)

* Logical to Physical (Logical add + Base register)
* Logical address bound to physical address.

Dynamic Loading

* Routine is not loaded until it called.
* All routine keeps on disk (Secondary mem)
* Implemented through program design.
* OS can help by providing Libraries to implement dynamic loading.

19301145

Paging

Date

(i) Devide physical memory fixed size blocks — frames
(ii) Devide logical memory fixed same size blocks — pages

To run a program of size N pages, need to find N free frames and load program.

page table (logical to physical

Address translation scheme

1. Page number (p) : an index to page table which contains base address of each page in physical memory. (m-n)

2. Page offset (d) : combined with based address to define the physical memory address that is sent to the memory address. (n)

For given logic address space 2^m and size 2^n.

** Page count starts from (0-q). to represent q in binary
= page bit. or 2^{m-n}

Therefore, offset = ~~(m-q)~~ ~~(m-n)~~ (total bit - q)

Example

32 byte memory and 4 byte page.

∴ Total page

here,

$2^m = 2^5$ $\quad 2^n = 2^2$

$\therefore m = 5 \quad \Rightarrow n = 2$

Page number $= (5-2) = 3$ (MSB 3 bit) or $2^3 = 8$ page possible

and offset = LSB 2 bit.

* ~~Page~~ if we reduce page size then entity of page table will increase.
* Worst case fragmentation = 1 frame − 1 byte
* Average fragmentation = $\frac{1}{2}$ frame size

Date :

Page table

* In this scheme every data/instruction access requires two memory access.
  a. page table
  b. data/ instruction

* Associative memory or translation look-aside buffers (TLBs) helps ~~to~~ to 1 memory access

* Associative Memory
  * parallel search
  * Address translation (p, d)
    - If p is an associative register, get frame out - (1)
    - Else, get frame from page table in memory - (2)

TLB
  (I) 1st search in TLB
  (II) Not found - page table - memory

Effective Access Time :
  Associate look up = ε time unit (<10% of memory access time)
  Hit ratio = α

Example

19301145

① $\alpha = 80\%$, $\varepsilon = 20ns$, TLB search $= 100ns$

$\therefore EAT = \frac{80}{100} \times 100 + 0.2 \times 200 = 120ns$

$= (80 + 40)ns$ miss 0.2 ∈ 2-time execution

$= 120ns$

② $\alpha = 99\%$, $\varepsilon = 20ns$, TLB $= 100ns$

$\therefore Eat = \frac{99}{100} \times 1 \left(\alpha/100 \times TLB \right) + \left(\frac{100 - \alpha}{100} \times 2 \times TLB \right)$

$= \frac{99}{100} \times 100 + \frac{1}{100} \times 200$

$= 99 + 2$

$= 101 ns.$

— X —

By —

Binita Khan Shaikal

19301145