6-2-24
Lec-1

Ch-1 Lec-1

Operating system
↳ এটা এমন system যেটা operate করতে পারে
↳ Computer user and hardware এর মধ্যে interface/interaction
↗ main purpose

Operating system functions:
" " services:
* " " components:

1. Users
2. Application softwares →
3. System "
4. Hardware
↳ নিজে run করে output নিয়ে আসে

* OS Examples:

1. Process Management :
↳ single unit processor

2. I/O device management
3. File Management
4. Network "
5. Main Memory " → Primary memory
6. Secondary "
7. Security "
8. Command interpreter system

* Why do learn OS?

⇒

System software → Operating system

Application "

*OS component: 8 টি

File Management

↳ Creation & deletion

Lists, stacks, queues

* Cache memory তে কোন File থাকলে তা তাড়াতাড়ি open হয়।
Recent File।

1. Linux → Dual boot
Windows → gpt or mbr ← Disk partition type
Pendrive ↓ iso File ubuntu boot

7-2-24

Lec-2

Lecture-2

* Types of Operating System:

↳ service এর উপর Depend করে Network OS.

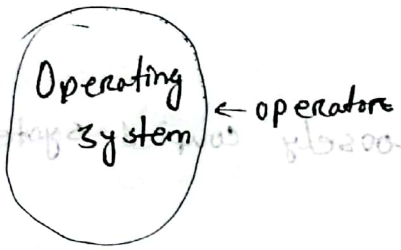
Distributed Network → আলাদা আলাদা Computer, এদের OS

→ acts the manager of all the resources

1. Batch OS:

↳ similar type এর task বা কাজ perform করার জন্য

→ " jobs → same requirement and group



Adv, ~~Disadv.~~

↳ processor knows how long the job

↳ Multiple users

↳ idle time very less

↳ easy to manage Large work repeatedly

Disadv.

↳ best experienced

↳ hard to debug

↳ costly

↳ wait for an unknown time if any job fails

↳ idle time বেশি

2. Time-sharing OS / Multitasking system:

↳ time কে share করে OS কাজ করে

↳ time wise CPU কে ভাগ করে

→ One by one sequentially

Adv.

↳ idle time can be reduced

Disadv.

↳ security and integrity of programs and data ensure করা যায়না

Ex → Unix

3. Distributed OS / Loosely coupled system:

↳ বেশি used

↳ অনেকগুলো computer এর মাঝে connect করার জন্য

↳ Processors differ in size and function

Adv.

↳ All systems are independent

↳ resources being shared, computation highly fast and durable

↳ Load

↳ Delay data processing reduces

Disadv.

4. Network OS / tightly coupled system
↳ Linux server
↳ Network এর provide করে এই OS.

Adv.
↳ Highly stabled
Hardware change করলে Windows এর মতো effect হয় না

Ex: Linux, MacOS, UNIX

5. Real-time OS:
↳ Process and response time
↳ small

Ex: like missile systems;

Adv.

Disadv.
↳ Complex algorithm

OS services:

1. Program execution
↳ Load main memory → a program
↳ Execute program
↳ Handle . .
↳ Process synchronization
↳ . . communication
↳ deadlock deadlock handling

2. I/O operation:

3. File system manipulation

* Communication
  - ↳ 2 methods implemented
    - ↳ 1. Shared Memory
    - 2. Massage Passing

* Error handling:
  - ↳ error ~~dect~~ detect and correction

* Resource Management:
  - ↳ Allocate → → Disallocate

* Protection:

15-2-24

Lec-3

* OS Properties:

1. Batch processing → sequence of commands, programs and data
2. Multitasking → number of jobs in memory and executes
   ↳ order of submission
3. Multiprogramming ↳ output spool
   ↳ যদি printing এত সময় লাগে
   ↳ Process synchronization
4. Interactivity
5. Real Time system
6.
7.

executes complete

Adv. disadv.

→ simultaneously

2. Multitasking: একসাথে অনেকগুলো job executed হচ্ছে

Time-sharing system

context switching
↳ দ্রুত switching হয় বুঝা যায় না switching হচ্ছে

→ 1ns, 1ns

↳ CPU scheduling

↳ CPU utilization বাড়ানোর জন্য

3. Multiprogramming:

* process in time → program

↳ sharing the processor

↳ 2 or more programs → concurrently

Multithread → একটা process কে ছোট ছোট part/unit এ ভাগ করা
→ একটা task এর ভিন্ন ভিন্ন অংশ

Multiprocessor → core i7 → 7টা process একসাথে simultaneously কাজ করে।

↳ time save করে,

Round-Robin algorithm: time-share করা হয়,

Illustrate → Diagram

4. Interactivity:

↳ user & interaction

5. Real Time System: → hard/soft

↳ embedded system

→ Robotics

→ ensure correct performance

6. Distributed Environment:

↳ multiple independent CPUs and processors

↳ CPU Buffer

* Diff. 32bit & 64-bit OS → H.W.

BIOS is Record

↳ Boot

1. Kernel Mode

19-2-24
Lec-4

* Microkernel in OS:

↳ Software and hardware এর মাঝে Bridge তৈরি করে

↗

Kernel

↳ specific কাজ করার জন্য বলে

↳ Create new file
read file
write "

Figure:

Kernel process → internal

User process → user mode ↳ execution → যখন কোন User কাজ করে → user address space

Kernel " → mode → internal এর OS এর মাঝামাঝি → Kernel " "

Primary memory
↓
secondary memory
↓
Harddisk

* Virtual Memory Shaduling

↑ secondary memory

* System call OS:

↳ interface between a process and OS.

User mode → User process Executing → S.C →

Kernel mode →

Excute call

result

* Why do we need System Calls in OS?
↳ Reading and writing

Types:

1.
2.
3.
4.

* Bus time: কোনো একটা complete হতে প্রয়োজনীয় সময় যাবে → শুরু থেকে শেষ

wait()
↳ একটা process থেকে আরেকটা process complete/execute হওয়ার জন্য wait করে।

fork(), kill(), exit()
↳ একটা কাজ করলে terminate হয়
→ child process runs concurrently

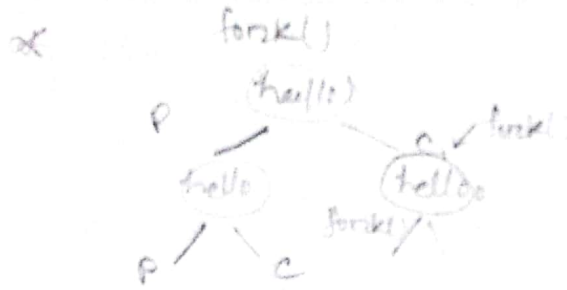
* Windows, Linux → catagories → দুইভাবে

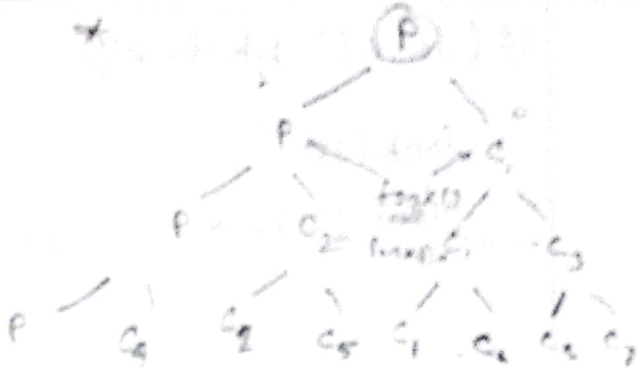
→ -ve : Child ~~not created~~ process unsuccessful

child তৈরি call করে → 0 : New child created, returned value হিসেবে দিবে সেটা

+ve : Parent থেকে call করে
↳ -ve হলে child থেকে শুরু হবে
↳ +ve " Parent " " "

fork() link 2nd example





0 → if
1 → else

0 → right side check mate

fork() || fork()

1 → left

0 || 0
1 || 0

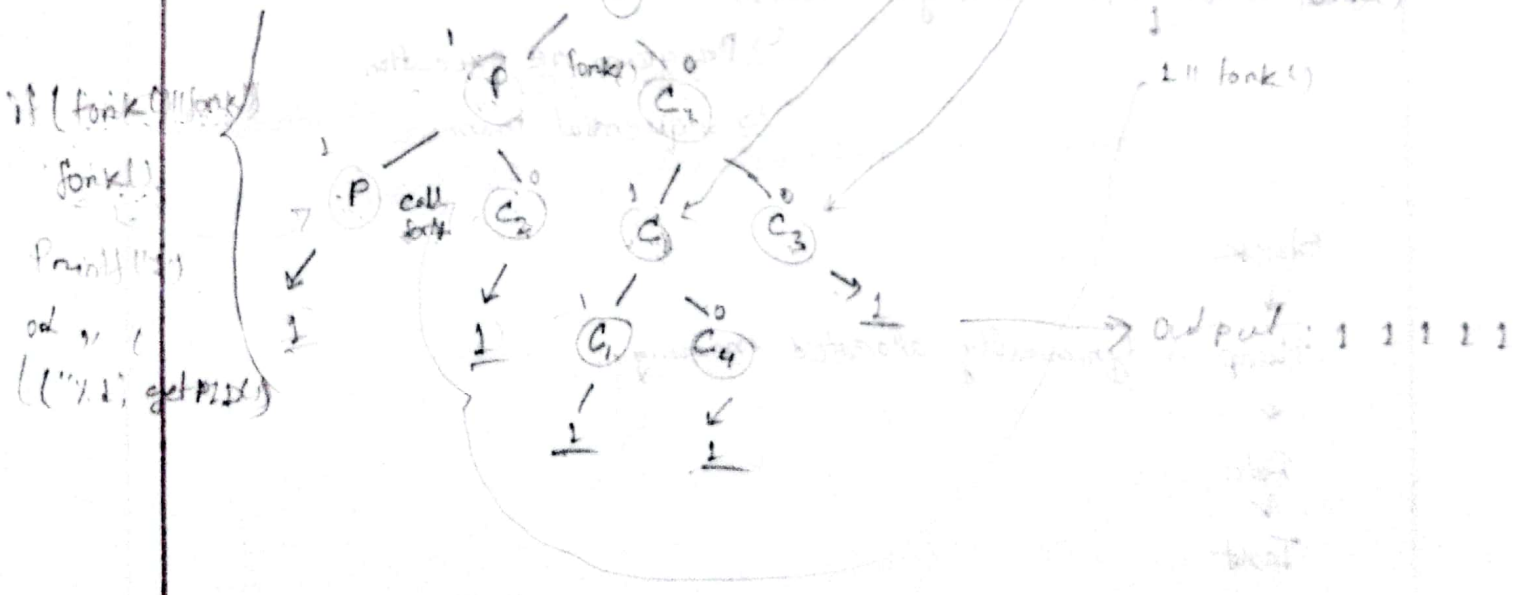
$2^n = 8 = 2^3$

$2^n - 1 = 8 - 1 = 7$

3 → fork() ka 3 bar call

n = system call

0 || 0 → 0
0 || 1 → 1
0 || fork()
fork() || fork()
1
1 || fork()



H.W: fork() && fork()

27-2-24

Lec-5

```
if (fork() && fork())
    fork();
print(1);
```

[illegible] Running → Process

↳ Program in execution

↳ sequential manner

Stack
↓
Heap → dynamically allocated memory
↓
Data
↓
Text

Ready State:

* Process Life Cycle:

1. New State → Secondary memory
2. Ready " → Primary
3. Run " → Primary
4. Terminate " → Secondary

→ 3. Wait state → Primary

* Degree of Multiprogramming
* Long term scheduler (LTS)
  ↳ New থেকে Ready state এ Data নিয়ে আসে

* PCB → Process Control Block → PID থাকে
  ↑ প্রত্যেকটা Process এর একটা PCB থাকে
  ↳ Process শেষ হলে PCB ও শেষ

Time quantum: Parallel অবস্থায় কাজের কমানো time দেয়, starvation দূর করার জন্য

* Suspend Ready → VIP কে কাজ করার জন্য
* " Wait → I/O completion হলে যাওয়া suspend wait
  ↳ Ready state এসে পড়ে

Round Robin CPU scheduling Algorithm

28-2-24

Lec-6

1. Long term or Job scheduler:

↳ careful selection → I/O and CPU bound Process

↓

secondary and main memory এর মধ্যে

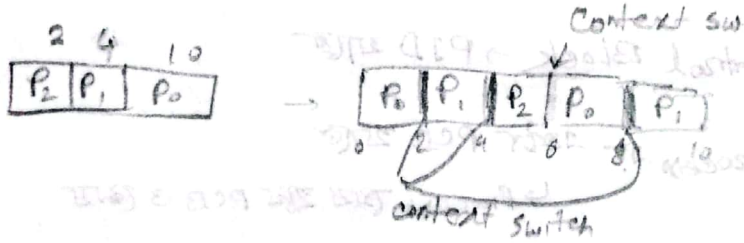
Task Manager → Running → CPU bound processes

2. CPU scheduler or

↳ ready to run state

↳ dispatcher

* Context switching → কাজ complete না হওয়া পর্যন্ত বারবার নিজেকে switch করবে।



3. Medium term scheduler:

↳ suspend, wait state

↳ swaping → main memory to disk or vice versa

High Priority

↳ CPU scheduler

1. I/O scheduler: I/O operations

2. Real-time ও : Critical task

* ~~Threa~~ Threads in OS:

Thread → একটি process এর ছোট ছোট অংশ

↳ light weight process

↳ small memory

Program counter (PC):

Ex → formatting text

Processing unit

* Need of Thread:

Process Vs Thread

Thread → same memory space → interdependent → system call এর প্রয়োজন নেই।

Process → different " " → ~~Mobile Ro~~ independent

↳ Thread এর সুবিধা

Types of Thread:

1. Kernel level thread
2. User " "

↳ OS not recognize

↳ user Level thread blocking operation → whole process blocked

↳ Kernel দেখে একটি thread ~~process~~ মনে হয় but multiple thread ঠিকমত আসছে

2. Kernel

↳ যে Multiple thread কে একটা process মনে করে,

একটা thread kernel এ কিছু block হলে তখন আমি thread এর উপর কোনো effect হবে না

* Component of Threads
* Benifit " "

29-2-24
Lec-7

Multithreading Models & Hyperthreading:

3 ways-

1. Many to one Model → কোনো একটা systemএর block কে call করে তাহলে আমাদের Automatic Block হয়ে যায়
2. One to one " ↳ solution Multiprocessor এগুলোর problem হবে,
3. Many to Many " ↳ concurrently, parallel execution possible

↓

↳ ≤ হবে যাতে user and Kernel thread.

↳ " " specific হবে যাতে,

↳ Multiprocessor এ parallel execution হবে

↳ একটা user thread এর জন্য একটা Kernel thread আসবে,

↳ problem → Overhead

↳ performance degrade হবে

* Hyperthreading / Simultaneous Multithreading (SMT)

Number Of Cores → Physical

" Logical Processors → Logical

CPU scheduling

Why do we need CPU scheduling?

⇒

CPU scheduling Algorithm

1. Pre-emption Algorithm → shortest Round Robin
2. Non " " → shortest, longest

→ shortest remaining Time First → Preemptive SJF
→ longest " " " → LJF

(i) Non-preemptive SJF
(ii) " LJF

Ready তে আসার পর থেকে complete হতে যে সময় লাগে or TAT.

5-2-24
Lec - 8

CPU Scheduling Terminologies

Arrival time: ready queue তে আসলো Process কখন

↳ কত time এ আসছে

↳ যদি 5th time এ ready queue তে আসে

Arrival time – 5ms

CPU Idle time: 1st 5ms time → Arrival এর আগের

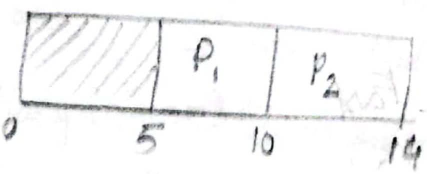
Burst time / Execution time: execution হতে কত time লাগছে

* Arrival time same হলে PID দেখে কাজ করতে দিবে

Complition time: প্রত্যেকটি process CPU এর কোন time এ শেষ হবে

Preemption, Non-preemption

↳ First in First out



↳ Completion time

→ 6ms এ আসছে

P_2 → Arrival time → 6ms

Burst time → 4ms → 10 to 14

Complition time → 14ms

Turn Around time → 14 - 6 = 8ms

Response time: 10 ms − 6 ms = 4 ms ← পাবের step এর

↓ আগের step এর completion time

↳ পাবের step এর Arrv. time

Waiting time: = Trunaround time − burst time

14 − 10 = 4 ms

First Come First Serve (FCFS): → Non-prumption

↳ Arrival time ← Crutenda

Process	Arrival time	Burst time
P_1	0	4
P_2	1	3
P_3	2	1
P_4	3	2
P_5	4	5

Avg trunaround time ✱

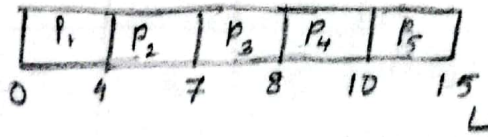
Avg Waiting time

↑

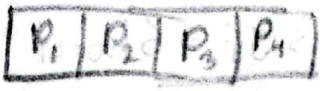
Algorithm ভালো বা খারাপ

→ Avg TAT = 6.8, Avg waiting time = 3.8

Gantt Chart: Bar chart এর ছবিটি version



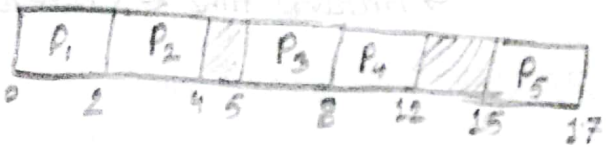
↳ CT



↑ ready queue

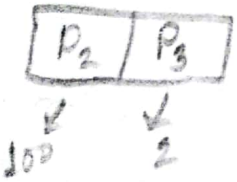
Process	AT	BT	CT	TAT	WT	RT
P_1	0	2	2	2	0	0
P_2	1	2	4	3	1	1
P_3	5	3	8	3	0	0
P_4	6	4	12	6	2	2
P_5	15	2	17	2	0	0

Gantt chart:



4-5 : 1 ms ← Idle time

Disadvantage: convey effect



TAT → 102 2 (1 412

~~* SJF → Short Jump First~~

* SJF = Short Job First

6-3-24
Lec-9

Disadvantages → Convoy effect

SJF → যার Arrival time কম আগে যে আগে যাবে

↳ Shortest Job First

↳ 1. Primptive SJF বলে
2. Non-Primptive { Shortest Remaining Time First
↳ Context switching করা যাবে না

Disadv. → প্রত্যেকটার Process এর Burst time আগে জানতে হবে
→ Starvation হতে পারে

Avg. Waiting time & Avg. Turnaround time যত কম তত ভালো।

SJF এর
Criteria → Burst time

Non-premptive SJF : →WT, RT same হবে Always

CT-AT
→TAT-BT

PID	AT	BT	CT	TAT	WT	RT
P_1	0	4	4	4	0	0-0 0
P_2	1	3	10	9	6	7-1= 6
P_3	2	1	5	3	2	4-2= 2
P_4	3	2	7	4	2	5-3= 2
P_5	4	5	15	11	6	10-4= 6
				Avg = 6.2	Avg = 3.2	

Ready queue

P_1	P_2	P_3	P_4	P_5

0 1 2 3 4 5

7

Gannt chart :

P_1	P_3	P_4	P_2	P_5

0 4 5 7 10 15

→ ms

→ Gannt chart এ time হবে AT দেখে

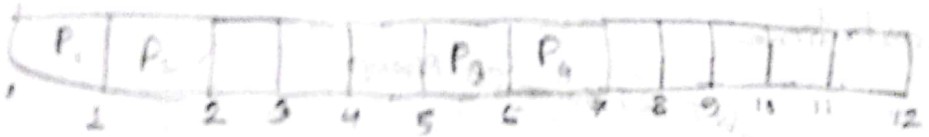
P_1 শেষ হওয়ার পর Burst time check করবে।
↳ short করবে

কাজ complete হবে Ready queue থেকে

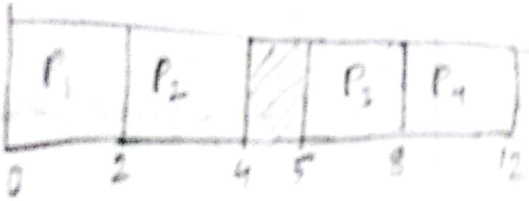
PID	AT	BT	CT	TAT	WT	RT
P_1	0	2	2	2	0	0-0=0
P_2	1	2	4	3	1	2-1=1
P_3	5	3	8	3	0	5-5=0
P_4	6	4	12	6	2	8-6=2
				Avg 3.5	Avg 0.75	

↳ Burst time sorting ✗ ∴ FCFS & SJF same

Ready Queue:



Gantt chart:



→ CPU time instance

Preemptive SJF (SRTF): → WT, RT same হতে হবে না

Time quantum = 1 ms

PID	AT	BT	CT	TAT	WT	RT
P_1	0	4	5	5	1	0
P_2	1	3	10	9	6	7-1 = 6
P_3	2	1	3	1	0	2-2 = 0
P_4	3	2	7	4	2	5-3 = 2
P_5	4	5	15	11	6	10-4 = 6
				Avg = 6	Avg = 3	

Ready Queue:

P_1	P_2	P_1	P_3	P_1	P_4	P_5	P_1	P_4	P_2	P_2	P_5	P_5	P_5	P_5
4	3	3	1	2	2	5	1	1	2	1	4	3	2	1

Gantt Chart:

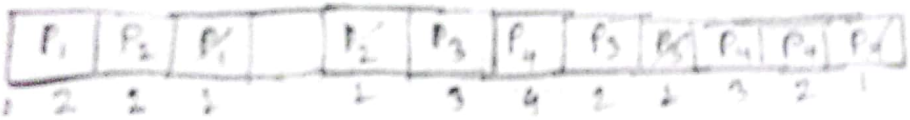
P_1	P_1	P_3	P_1	P_1	P_4	P_4	P_2	P_2	P_2	P_5	P_5	P_5	P_5	P_5
0–1	1–2	2–3	3–4	4–5	5–6	6–7	7–8	8–9	9–10	10–11	11–12	12–13	13–14	14–15

P_5
10–15

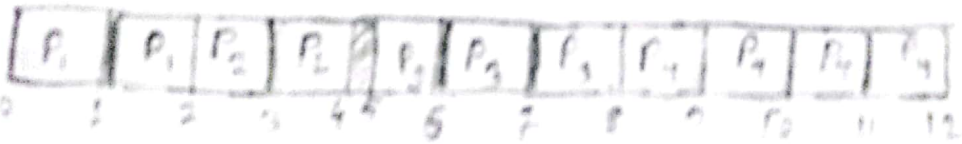
* Compare করলে বেশি switching হবে। Load এর টাইম বেশি
" হয় না

PID	AT	BT	CT	TAT	WT	RT
P_1	0	2	2	2	0	0
P_2	1	2	4	3	1	1
P_3	5	3	8	3	0	0
P_4	6	4	12	6	2	2

Ready Queue:



Gantt Chart:



* [illegible] Algorithm [illegible] Adv. Disadv. diff/ Comparison

7-3-24

Lec-10

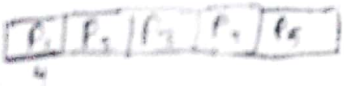
Longest Job First → Preemptive

↳ Non - "

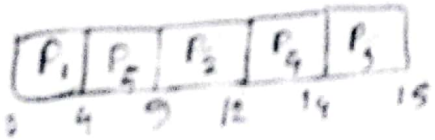
Non-preemptive:

PID	AT	BT	CT	TAT	WT	RT
P_1	0	4	4	4	0	0
P_2	1	3	12	11	8	7-1 = 6
P_3	2	1	15	13	12	14-2 12
P_4	3	2	14	11	9	12-3 = 9
P_5	4	5	9	5	0	4-4 = 0
				Avg: 8.8	Avg: 5.0	

Ready queue



Gantt chart

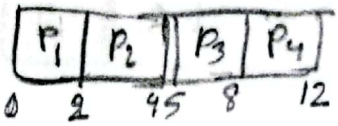


Non-preemptive LJF:

PID	AT	BT	CT	TAT	WT	RT
P_1	0	2	2	2	0	0
P_2	1	2	4	3	1	1
P_3	5	3	8	3	0	0
P_4	6	4	12	6	2	2
				Avg=	Avg=	

Ready queue:

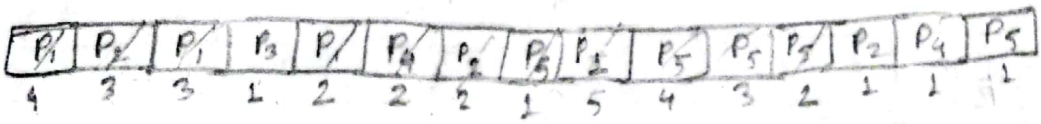
Gantt chart:



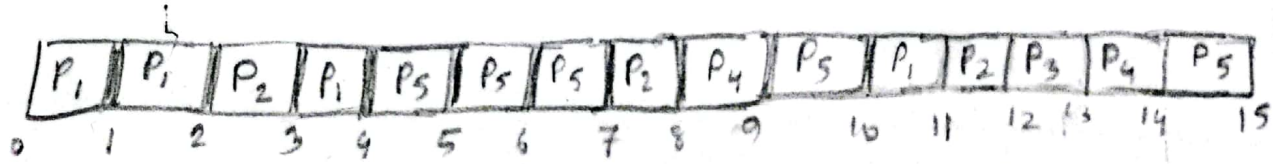
Preemptive LJF:

PID	AT	BT	CT	TAT	WT	RT
P_1	0	4	11	11	7	0-0 =0
P_2	1	3	12	11	8	2-1 =1
P_3	2	1	13	11	10	12-2 =10
P_4	3	2	14	11	9	8-3 =5
P_5	4	5	15	11	6	4-4 =0

Ready queue:



Gantt chart:



* Convoy effect

↳ waiting time বেড়ে যাওয়া

→ context switching এর কারণে CPU degrade করে

preemptive → Popularity

Round Robin Algorithm:

→ Adv. CPU performance বাড়ায় → Context switching হয় / Disadv.

Quantum
↳ time slice use করে
↳ but fixed time

→ CPU অনুযায়ী Process কে নির্দিষ্ট time এ

Time slice/Quantum = 2ms ← Maximum 2ms

Ready queue

4	3	1	2	2	5	1	3	1
P_1	P_2	P_3	P_1	P_4	P_5	P_2	P_5	P_5

0

or P_5 (3)

Gantt → CPU time

P_1	P_2	P_3	P_1	P_4	P_5	P_2	P_5	P_5

0 2 2 4 5 7 9 11 12 14 15

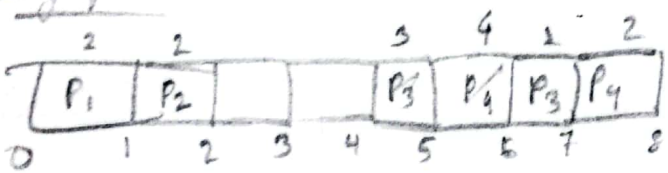
or P_5 (12 – 15)

PID	AT	BT	CT	TAT	WT	RT
P_1	0	4	7	7	3	0 − 0 = 0
P_2	1	3	12	11	8	2 − 1 = 1
P_3	2	1	5	3	2	4 − 2 = 2
P_4	3	2	[illegible]	6	9	7 − 3 = 4
P_5	4	5	15	11	6	9 − 4 = 5

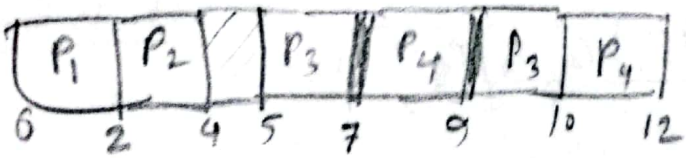
PID	AT	BT	CT	TAT	WT	RT
P_1	0	2	2	2	0	0-0 =0
P_2	1	2	4	3	1	2-1 =1
P_3	3	3	10	5	2	5-5 =0
P_4	6	4	12	6	2	7-6 =1
				Avg = 4	Avg = 1.25	

Time slice = 2

Ready queue:



Gantt:



Priority:

1. Lowest value Highest Priority → Priority column no extra
2. Highest " Lowest "

* Priority A time slice 2 but priority base queue

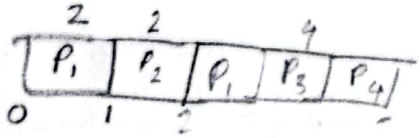
Priority Based Scheduling Algorithm:

LVHP

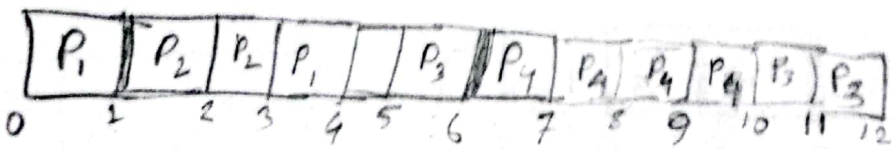
PID	AT	BT	Priority	CT	TAT	WT	RT
P_1	0	2	2	4	4	2	0-0=0
P_2	1	2	1	3	2	0	1-1=0
P_3	5	3	4	12	7	4	5-5=0
P_4	6	4	3	10	4	0	6-6=0
					4.25	1.5	

Preemptive

Ready queue



Gantt:

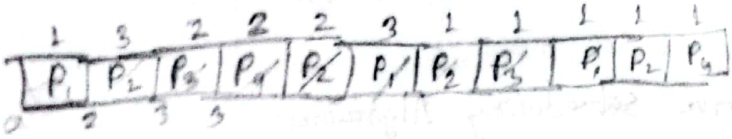


Mix Burst time: (Criteria - Priority

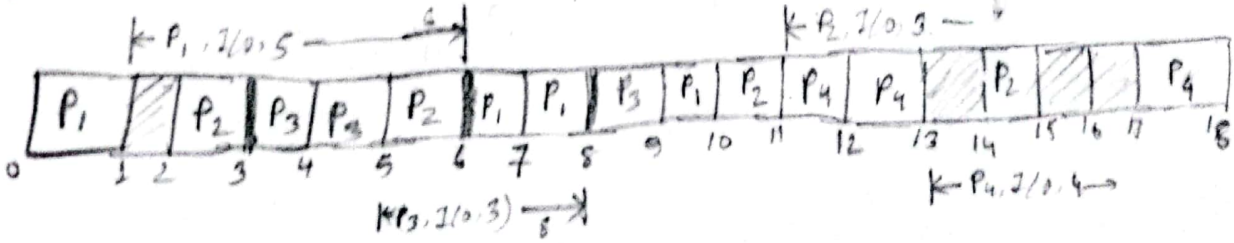
Mode - Preemptive, Lowest No - Highest Priority)

PID	AT	Priority	CPU	I/O	CPU	CT	TAT	WT	RT
P_1	0	2	2	5	3	10	10	10-(2+5+3) = 0	0-0=0
P_2	2	3	3, 2	3	1	15	13	13-(3+3+1) = 6	2-2=0
P_3	3	1	2	3	1	9	6	6-(2+3+1) = 0	3-3=0
P_4	3	4	2	4	2	18	15	15-(2+4+2) = 8	11-3=8
							11	3.75	

Ready queue:



Gantt chart:



~~Multilevel Queue Scheduling:~~

VIP process → System process

user " → interactive "

Similar type " → Batch " → Background ?

Priority → High → System
Midium → Interactive
Low → Batch

VIP
Interactive
Batch → আলাদা আলাদা Queue use করে → Multilevel Queue

↳ আলাদা আলাদা Algorithm follow করতেও পারে আবার এক " ও " " পারে

1. Fixed Priority preemptive scheduling Algorithm

↳ এদের time শেষ না হওয়া পর্যন্ত অন্য কেউ সুযোগ পাবে না

2. Time slicing:

Multilevel Feedback Queue Scheduling

* Background Process → Batch Process

↳ Queue - 3 → Interactive

↳ Queue - 1 → System ..

↳ " - 2 → Interactive ..

priority.

*CPU-Bound process

→ I/O ..

Ex:

Q_1	2
Q_2	7
Q_3	12
Q_4	17
Q_5	2

14-3-24
Lec-13

Process synchronization:

↳ কেন দরকার?

2 জন user একসাথে login করলে একজনকে Allow করবে
↳ Multiple session একসাথে run করতে দিবে না

*How process synchronization in OS works?

↳ একই time 2 Process Read & Write করতে পারে

Elements/Section in a program:

1. Entry section
2. Critical " → main → allow only one process
3. Exit " ↳ Read or write
→ আগের process শেষ হওয়ার পর নতুন আরেকটা process shared করলে এটা মুক্ত হবে
4. Remainder " → Code এর যে অংশটা উপরের এই section না পড়লে তা এই section এ আসবে

Race Condition:

Multiple process একসাথে run করতে চাইলে → risk

→ একই Data access করলে ~~Execution হবে~~ result depend করবে Execution এর order এর উপর depend করে

*Multiple thread execution হলে,

Solution:

Critical section এ একই সময়ে একটি process-কে Allow করবে

* Critical section problem:

wait() function

↳ handle করে কখন কোন process run হবে

* Requirements of Synchronization:

1. Mutual exclusion: একটি process running হওয়ার সময় critical section এ অন্য process কে Allow করবে না। → ২টি process এর মধ্যে mutually decision নি কোন process আগে execute হবে।

2. Progress: Remaining section এ যে process/thread decide করবে কোন process টি আগে হবে Execution, Critical section এ।

3. No Starvation/Bounded Waiting:

↳ নির্দিষ্ট সময় wait করে critical section এ যাবে

Starvation → infinite time wait

Solution of Critical Section Problem:

* Critical part execution করার আগে Remainder part execution হবে কী problem?

→ Remaining এ Write কে আগে দিয়ে তারপর Read/ Template করবে

Critical section এ execution শেষ হলে Remaining process গুলো execution হবে

*A Boolean flag:

True: Process access করতে পারবে

False: " " " না

initial → False থাকবে

↳ False থেকে true হলে Access করবে,

19-3-24
Lec - 14

IN → কোনটায় count হবে

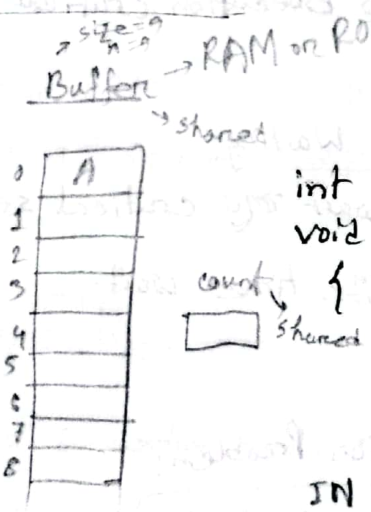
→ size Fixed

Procedure Consumer Problem:

→ Best case

Consumer Code

```
int count;
void consumer(void)
{ int itemC
  while (1)
  { while (count==0);
    itemC = buffer[out]
    out = (out+1)%n
    count = count - 1
  }
```



→ Best case

Procedure Code

```
int count = 0
void prodedure(void)
{ int itemp
  while(1)
  { Produce_item(item P)
    while(count==n);
    buffer[in] = item P;
    in = (in+1)%n   → division Method
                       ↳ circularly
    count = count+1
```

item = A
↳ IN

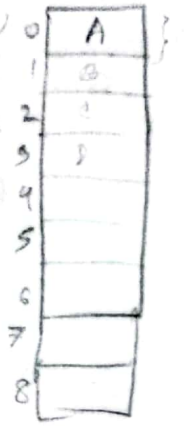
IN
নির্দেশ
initially
IN
0
count
0

4
2

1. Load R_c, m[count]
2. Decr. R_c
3. Store, m[count], R_c

1. Load R_p, m[count]
2. Incr. R_p
3. Store m[count], R_p

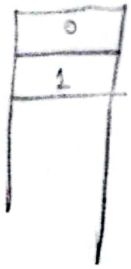
Critical section: buffer, count



while (count == n); → buffer full → condition true → infinite loop

while (count == 0) → buffer empty → condition true →

Register



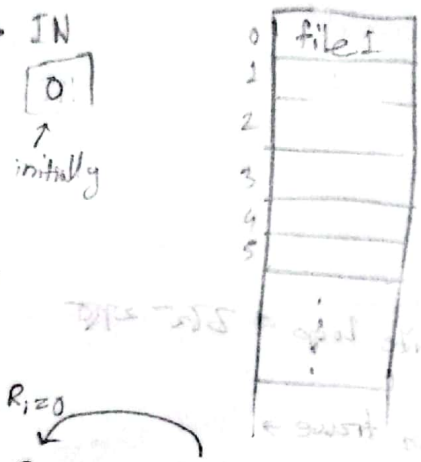
* Synchronization problem

Printer Spooler Problem:

1. Load R_i, m[IN]
2. Store SD[R_i], "File_Name"
3. Increment R_i
4. Store m[IN], R_i

Best case

Spool direction



P_1 → file 1

Best case

$R_i = 0$

$R_1 \leftarrow m[0]$

$R_i = 1$

$IN = 1$

$P_1 \rightarrow I_1, I_2, I_3$ | P_2 I_1 I_2 I_3 I_4 | P_1 I_4

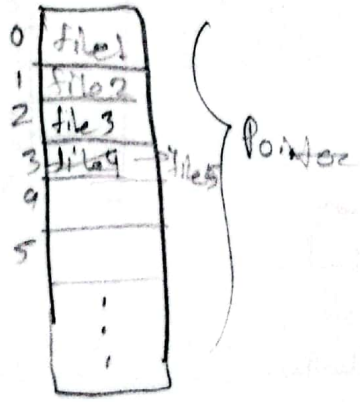
Context Switching

Spool pointer

Synchronization problem 2nd

Data loss

IN [3]



1. $R_2 = m[3]$
2. store m[3] → file 5
3. $R_2 = 4$
4. store $R_4 = 4 \rightarrow m[4]$ → IN [4]

P_1 = file 4

P_2 = file 5

1. R_1 = Load m[3]
2. store m[3] file 4
3. $R_1 = 4$

IN [4]

Race condition → একাধিক spool এর মধ্যে হতে চায়

↳ Race condition দূর করতে synchronize হয়

↳ Semaphore

Semaphore:

↳ integer value → concurrent process থেকে Race around condition দূর করা হয়

↗

+ve value থাকে বাকি

বাকি, -ve আসলে

হবে না

Binary, counting

↓ ↓

(0 and 1) (−∞ to +∞)

Program এ 4 টি section:

1. Entry, → P( ), down, wait
2. Critical → একটির program critical section এ থাকে চায়
3. Exit → V( ), up, signal, post, release
4. Remainder

Critical section

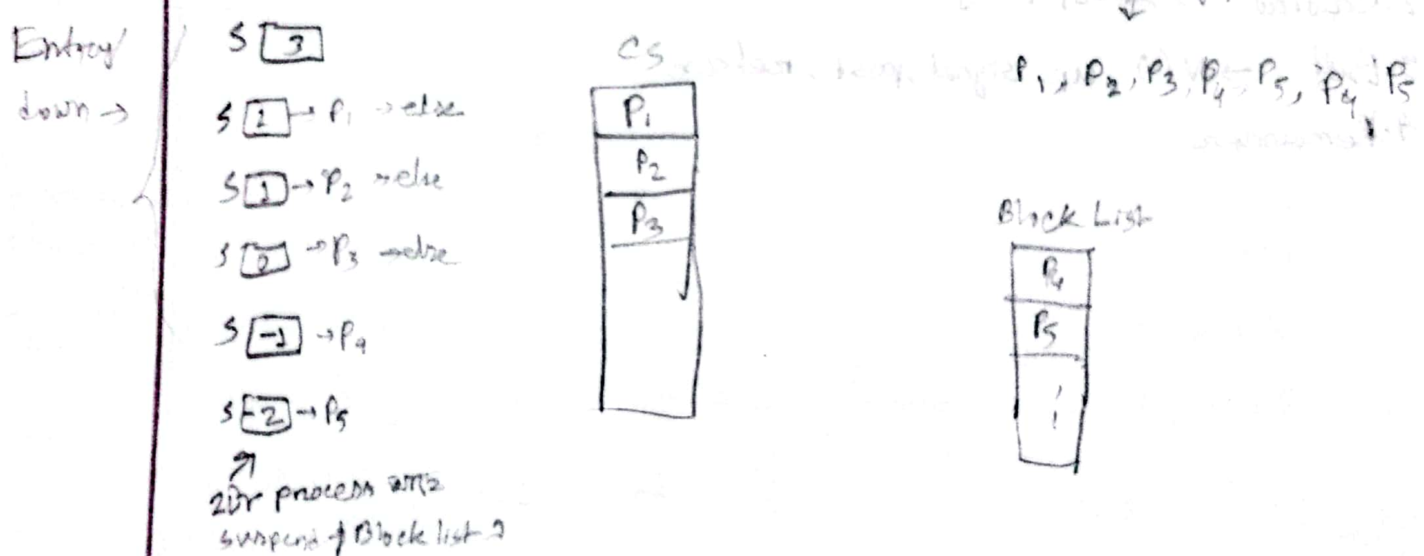
Entry code ↓ CS ↑ Exit code

```
down(Semphore s)
{
  s.value = s.value - 1
  if (s.value < 0)
  {
    put process(PCB) in
    suspended, sleep();
  }
  else
    return; / put process in cs
}
```

```
up(Semaphore s)
{
  s.value = s.value + 1;
  if (s.value <= 0)
  {
    Select a process from
    suspend list
    wake up()
  }
}
```

* S [0] Zer critical section ...

* S [3] ... process ...



Deadlock

S = 2

S = -1 → P_4 → Entry

Ready Queue

P_4 P_5

Entry

Semaphore

S = 0 → P_5 → Entry

S = 1

CS

P_1

P_2

P_3

P_4

Semaphore → +ve → process Excess

-ve → suspend process

0 → Block list & CS

* S = 5

6P

3V

When will be the value of S after these operations?

P V

$9 \rightarrow 9-6=3+3=6$

21-3-24
Lec-15

Binary Semaphore:

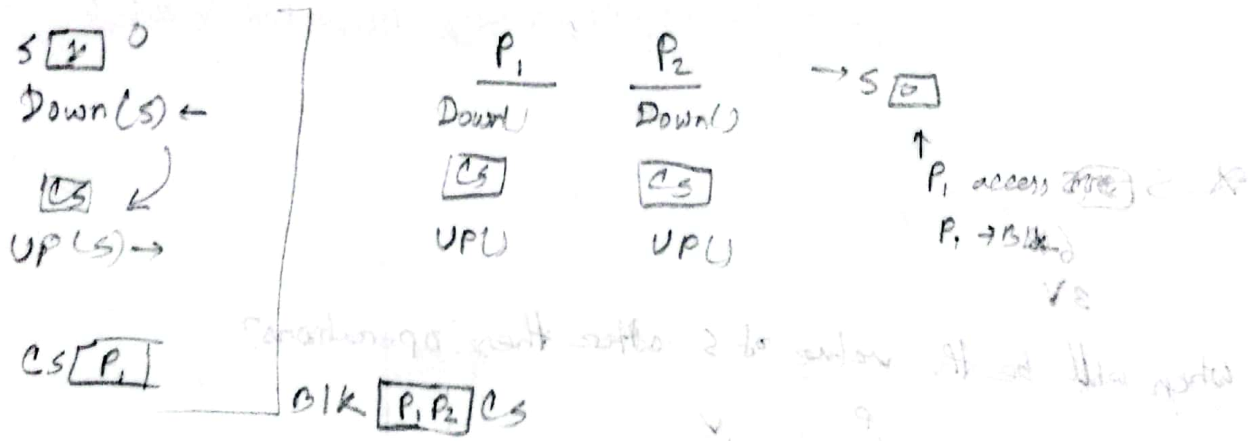
S → 0
S → 1

0 দিয়ে শুরু করলে ঢুকতে পারবে না process তাই রাখি initially 0.

```
Down (Semaphore S)
{ if (S.value == 1)      S[1]
   { S.value = 0
   }
  else                   S[0]
   block process, sleep().
}
```

```
Up (Semaphore S)
{ if (suspend list empty)      S[0]
   { S.value = 1;      → value কে up করলে always 1 করে ← 0/1 থাকে
   }
  else                         S[1]
   select a process from
   block list, wake up()     ↳ Ready Queue তে যাবে
}
```

Semaphore 0



* Binary Semaphore S[0] First এ থাকলে, সব process block List এ চলে যাবে, Down() select করার; carefully select করতে হবে তাই

CS → Shared → Synchronization problem হয় তাই
Down, Up → দিয়েছি

* Mutual exclusion
2টি process syn.

S [1]
↓
P1 → S [0] (Down) করে দিবে
CS → ঢুকে যাবে
Up করলে S [1] করে দিবে

* একটি process block list এ যাবে
আর " CS → যাবে
Mutual exclusion হবে

P1	P2
UP()	Down()
CS	CS
Down()	UP()

S [1]

(P1 – P9)
P(S) —— Entry
C(S)
V(S) → Exit

P10
V(S) ← Entry
[CS]
V(S) → P(S)

↳ Max → 3টি
↳ Max → একটি process যাবেই
↖ Worst case

Min → 1 টি Always

P10
P(S) → Block List
CS
P(S)

* Maximum কয়টি process CS এ থাকতে পারে? → 10টি
১০টি process CS এ ঢুকবে
S [1]

P1 → S [0]

CS [P1, P] → অন্য কোন process ঢুকতে পারবে না CS এ, P2 - P9 → block list এ চলে যাবে
↓ S [0]

P10 → S [1]

CS [P1, P10]

P2 → S [0]

CS [P1, P10, P2]

[P1, P10, P2]
↓
S [1]

S → 1, 0, 1, 0, 1, 0, 1, 0,

CS [P1, P10, P2, P3, P10, P4, P10, P5]

$S \rightarrow 1_1 0_1 1_1 0$

$\boxed{P_1 | P_{10} | P_2}$ → Maximum → 3 Or