Lecture – 01

19-01-2020

Q. What is an operating system from a user view and system view?

control program / controller → resource allocator

OS – system software that works as an intermediary betn resources and applications.

OS works as a controller from user view.

- Types of OS, adv, disadv

main aim – accomodate and provide all the given services (multi-tasking environment)

Types of OS

1. Single user single task OS / Single user Single process
2. Batch OS (one procen multiple instruction)
   → needed for efficiency (job scheduling)
3. Multiprogramming OS
4. Multiprocessing OS (Parallel System) [more than one CPU]
   multiple CPU
5. Multi-tasking OS / Time-sharing OS
6. Multithreading OS
7. Time-sharing OS
8. Distributed OS
9. Cloud OS

- An OS provides a basis for application programs.

An operating system is a system software that manages computer hardware, software resources, and provides common services for computer programs.

Classification of OS

- **Multi-user:** allows two or more users to run programs at the same time.
- **Multiprocessing:** supports running a program on more than one CPU.
- **Multitasking:** allows more than one program to run concurrently
- **Multithreading:** allows diff parts of a single program to run concurrently.
- **Real time:** responds to input instantly. (UNIX is not real-time)

Section: 11

Lab-01

md. muzahidul. islam. nahi @ g. bracu. ac. bd
ahmed. zohair. nasim @ g. bracu. ac. bd.

dktga2d

Linux → access to root files and directories;
developers' choice;
less restrictions compared to Windows / ~~user~~
open source, portable

1 $pwd → print current working directory
Shows the path direction from root
Root is the base of the Linux file system.
(prints the path of the working directory, starting from root)

2. ~~$fs~~ $ls → lists the files in the current directory
Shows list of files ^in ~~directories~~ directory

3. $cd change directory
takes you to a command directory (changes to another directory)
e.g.: $cd Downloads

4. $mkdir
makes a ^new directory
e.g. $mkdir abc

$rmdir
used to delete empty directory

owner – group – users

~~Read – 1~~
~~Write – 2~~
~~Execute – 4~~

5. $rm
removes both empty and filled directory
e.g. $rm Downloads

6. $touch create empty files
used for creating files
e.g. $touch new.txt

7. $cp
used for copying files from one directory to another
Directory path should be clearly mentioned
cp works same as mv, except it copies the original files instead of moving them.

8. $ mv
used for moving (cut and paste) files from one directory to another
Also used for renaming files.
moves a file to a new location
(also used to rename files)

$ ls
⇒ abc.txt
$mv abc.txt xyz.txt

⇒ $ls
⇒ xyz.txt

9. $locate
used for locating files
e.g. $locate new.txt
e.g. $ locate *new *abc*
Result: /home/new abc.txt

1 2 4
E W R

o-g-u

Execute — 1
Read — 4
Write — 2

- $ chmod 700 ese.txt
  $ ls -l

- commands and filenames in the terminal are case-sensitive.

Linux is a GUI based OS with a shell.

- Ubuntu is an OS
- Linux is the best-known and most-used open source OS.

Linux is software that sits underneath all of the other software on a computer, receiving requests from those programs and relaying these requests to the computer's hardware.

- Linux is open source software
- customizable
- secure

read (r), write (w), and execute (x)

The ln command creates links.

Lecture – 02

21-01-2020

Software → Application software
→ Systems software (interacts with hardware)

Hardware

Middleware

Firmware

- Booting
- API
- Kernel

Application software language → instructions → [Middleware] compiler, assembler, text editor

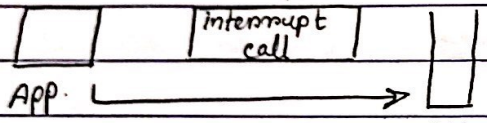
Kernel / OS language.

instructions → [Firmware]

Hardware

Kernel: Segment of codes that operates the OS

knock the CPU to inform that it is needed to do some work.



Buffer - temporary storage

interrupt call for a CPU request

Q. Why is cache needed?

- CPU knocks main memory after seeing the interrupt call.

cache - recently used / accessed data storage

- cache - frequently used / accessed or recently accessed.

So CPU now knocks cache first and then main memory.

- cache is small; costly because more speed than the RAM.

Secondary storage – CD, DVD, pendrive main memory – fixed-size memory

Storage → addressing scheme
helps to store and access exactly

magnetic tapes → Sequential access

Sectorization → specify storing এত smooth, accessing তত easier.

backstorage → main memory
[swap out]

PCB তে সবকিছু info dynamically store হতে থাকে

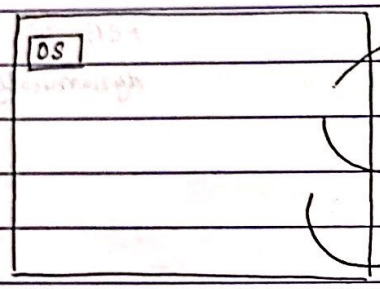
- Swap in, Swap out
- Timesharing environment
- interrupt handling

Q. Virtual box install ⇒ How does dual-mode work here?
(multiple mode)

- OS Functionality → OS Services [lecture-01 ~~fg-2~~ upto slide-20]
- System ~~wait~~ call
- Booting

parallel execution'e concurrency नहीं.
↳ eg. distributing OS → multiple req

System Calls → Programmatic way of calling the OS/Kernel.



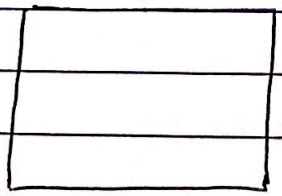
System works in two modes
↓ user mode
↓ kernel mode

execute, exit, read, write

Application Software { } System...

API translator

Java ⇒ machine-indepent, i.e. OS independent
↳ extra software
JVM
middle-man



- API translates high-level language to a System Call.

- API - Compiler / Library

Linux terminal'e direct invoke, তখন API লাগছে না!

System Boot

off → • no data in main memory
• kernel in ROM (secondary storage)

• OS main
↳ Kernel (particular segment of mode)

CPU by-default code
↳ firmware (hardware code)

• OS itself is a software

• re-boot
• system call and booting system

Process Management

Chapter (3)

program database ⇒ ~~PBM~~ PCB (Process Control Block)

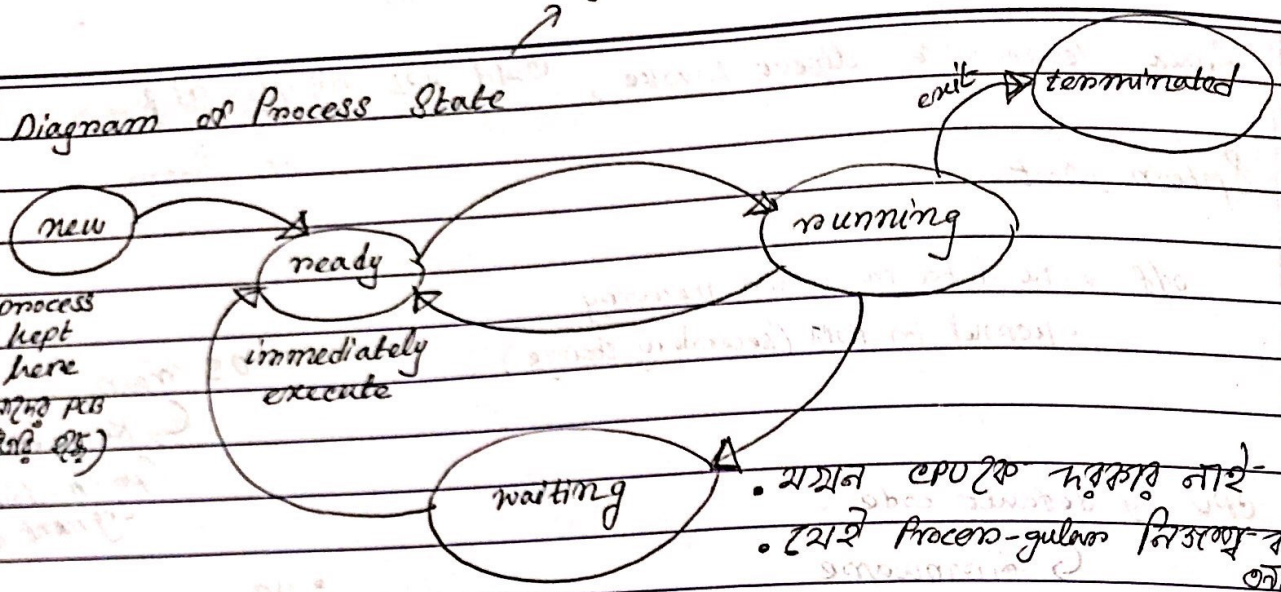
প্রত্যেক Processing'er shathe একটা PCB থাকবে

PCB jokhon jekhane jacche, shathe shathe update hocche.

PCB is a data structure that contains info of the process related to it.
It also defines the current state of the OS.

5 states

Diagram of Process State



- যখন CPU কে দরকার নাই
- যেই process-gulan নিজস্ব কাজ করছে

- 10 টি no. nille, user mode'e কাজ করছি

- consideration parameter depends on the algorithm.

- Process State'e info changeable.
- PCB আপডেট gets updated.

- temporary data in CPU registers

update PCB হলে, we can tell কোথায় process টা আছে।

a b input → waiting
a b * multiply → running

		States
0th s	P_0 → arrive	P_0 ⇒ new
2 s	P_1 → Arrive	P_1 ⇒ new
3 s	P_0 → immediately run	P_0 ⇒ ~~running~~ ready
5 s	P_0 → running	P_0 ⇒ running
7 s	P_1 → sleep / interrupted	P_1 ⇒ waiting / ready
8 s	P_3 → Arrive	P_3 ⇒ new
10 s	P_0 → End of execution	P_0 ⇒ terminated
11 s	P_3 → Processing the input	P_3 ⇒ waiting
12 s	P_1 → Start executing	P_1 ⇒ running

P_0 ⇒ ~~new running~~ ready running

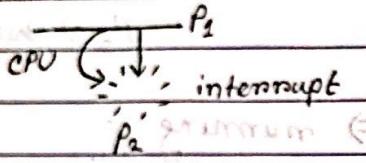
end of taking input ⇒ ready

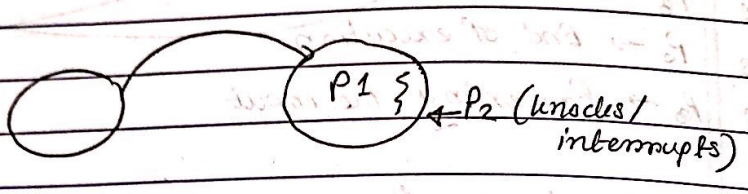
Lecture – 04 ~~Lab 02~~ Jan 28, 2020

- Ping
- PCB – Process Control Block database

new (so it's in main memory)

* CPU Switch from process to process
* context switch





যে interrupt করল, তাকে নিয়ে কাজ করছিল → Context Switch

(probably because P2 has a higher priority in the hierarchy)

* Process Scheduling

* Context Switch occurs in Running only *

Process State → (5)

Process Scheduling →

↳ queue maintained, because continuously process created.

Q. Suspended waiting used when?

- Suspend Ready, Suspend Waiting

We use midterm scheduler for interrupt-handling, swapping, contextswitch.

• OS uses various schedulers for Process Scheduling.

#

Scheduler → LTS, STS, MTS

mostly used →
- short-term scheduler (CPU allocate ready to running process)
- long-term " (new theke ready queue'teh nie asha)
- midterm → " (interrupt-handling) for interrupt call running e thaj theke, backup theke front-run e nie asha (Context switch)

(If I/O time > Execution time)
- I/O Bound process (execute time'r cheye beshi spend time I/O teh)
- CPU Bound process (CPU'te beshi time spend time)

(Execution time > I/O time)

interrupted process Ready'te jae.

5th - fork a child ⇒ Ready

fork a child: Create a process from another process (creating a child process)

~~Queing~~ Queue

Queueing diagram (ready → running)

Parent - Child
- Parallel executing
- Cascading process
  - if parent process terminated, automatically the child process being executed
- Shared → inheritance
  - Sharing is of 3 types (given in Process Creation slide)

Process cascading – if parent process dropped / killed, child process dropped / killed too.

if parent executes all files w/o child, the child also gets killed.

Lab 02

touch cse 321.sh

./filename.sh

./cse 321.sh

chmod +x cse 321.sh

#!/bin/bash

echo "Hello"

./cse 321.sh

var_name=value

double quotation ⇒ execute → variable's value

Single quotation ⇒ not executed, read as a string.

$variable

expr 1+1

' ' = no substitution

" " = subst. allowed

-n string (not null) ⇒ True

-z string (null)

if( )[( )$num1( ) -gt( )$num2( )]; ( ) then

( ) → space

যতগুলো line of arg আছে, সবগুলো read করবে। ⇒ $*

1. ls

2.
```
echo "enter num1"
read num1
echo "enter num2"
read num2
sum = $(( $num1 + $num2 ))
```

- ls command shows a directory listing.
- ls -l → list in long format.
The long option lists filename, sizes, permissions, and other info.

ls -a ⇒ shows hidden dotfiles.

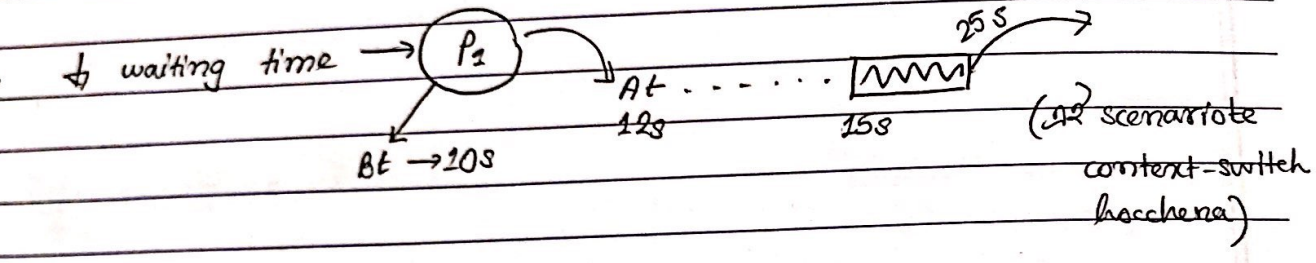
user	group	everyone
r w x	r w x	r w x
4 2 1	4 2 1	4 2 1
7	7	7

directories and files
handle file permissions differently.

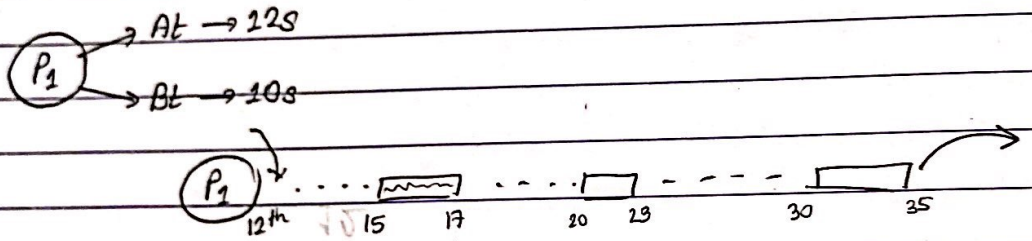
CPU Scheduling Criteria

→ CPU utilization → usability increase ↑

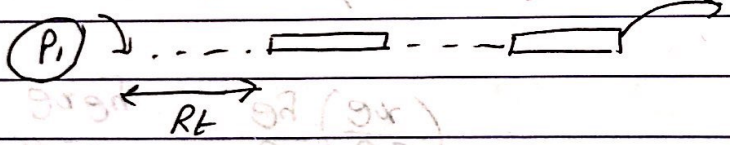
→ Throughput → outcome in per unit time in terms of total capacity ↑

→ ↓ waiting time → 

(12 scenario to context-switch hocchena)



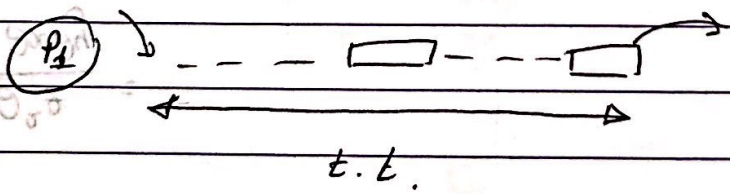
→ ↓ Response time



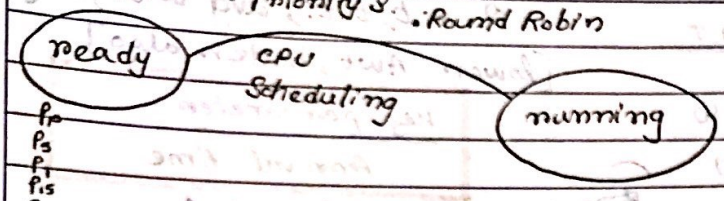
→ ↓ Turnaround time

wt + Bt

new theke shuru hoy



• FCFS • SJF • Multi-level
• Priority S. • Round Robin



P_0
P_3
P_1
P_5
P_2

- যত কম waiting time, তত efficient
- Throughput ↑, efficiency of algo ↑
- burst time = execution time

FCFS

Process	B.t	A.t
P_0	90	90
P_1	10	220
P_2	100	40
P_3	50	0
P_4	5	50
P_5	3	100

- Preemptive → allow context-switch
- Non-preemptive → context-switch not allowed

FCFS always nonpreemptive

Timesharing, RR always preemptive (কোন কোড কাজ করছে, so context-switch)

Lecture

09-02-2020

Q. w.t, t.t., avg w.t., avg.t.t. [lower Awt, better algo]

Key parameter: Arrival time

t.t = C - A

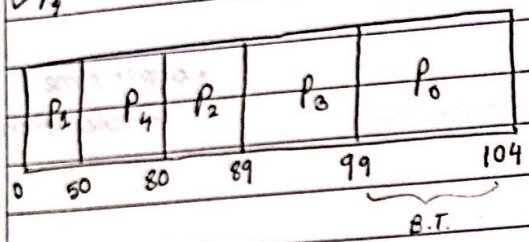
w.t = T - B

- waiting time
- turnaround time
- response time

FCFS

Non-preemptive

Process	B.T.	A.T.
P_0	5	80
P_1	50	0
P_2	9	9
P_3	10	40
P_4	30	4



P_1 w.t. = ~~99 - 80 = 19~~

w.t. → P_0 w.t. = 99 - 80 = 19

P_1 w.t = 0 - 0 = 0

P_2 w.t. = 80 - 9 = 71

P_3 w.t = 49 (start) (A.T.)

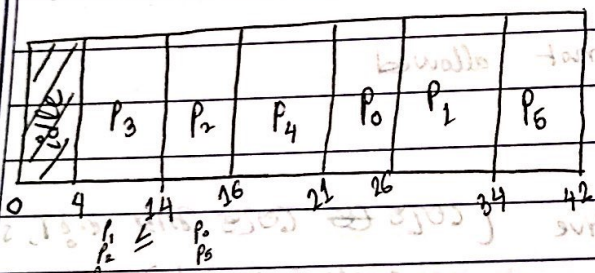
P_4 w.t. = 46

avg w.t. = $\frac{19+0+71+49+46}{5}$

SJF

Non-preemptive

Process	A.t.	B.t.
P_0	15	5
P_1	10	8
P_2	8	2
P_3	4	10
P_4	6	5
P_5	16	8



Shortage Job First

Key parameter: Burst Time

* non-preemptive scheduling

w.t. = r.t

r.t. CPU কতক্ষন সময় পেলাম

turnaround time — t.t = w.t. + B.T

w.t. B.T.

P_0 → 19 + 5 = 24

P_1 → 0 + 50 = 50

~~(or) t.t = C - A~~

S — L → completion time

↳ arrival time

P_0 → 104 - 80 = 24

[C.t. - A.T.]

* process-2 আগে arrive করতে হবে *

If, P_6 4 2 [2122 burst time কম, so use P_6 before P_4]

FCFS nonpreemptive, so process will run to completion.
↳ due to this nature, convoy effect may occur.

(Since AWT higher comparatively, so FCFS is poor in performance)

FCFS

	A.t.	B.t
P_0	0	100
P_1	1	2
P_2	2	4

Convoy Effect

$P_0 \to 0$ $\quad P_0 - 2$
$P_1 \to 99$ $\quad P_1 - 0$
$P_2 \to 98$ $\quad P_2 - 2$
65.6

choto processgula jokhon boro process'er behind'e ashche → Convey effect which is reqd to be solved

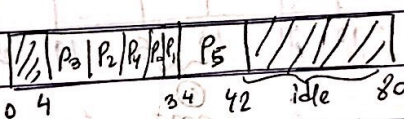
FCFS may suffer from the Convoy Effect if the burst time of the first job is the highest among all.

FCFS disadv: Convey effect

SJF

P_4 w.t. = 16 − 6 = 10s

P_6	80	5



Convoy Effect → FCFS
↳ short process waiting behind long process
↳ low priority process waiting behind higher priority process

increases waiting time

Starvation → Priority Scheduling (w.t. beshi hole)

↳ (solve by Aging)
↳ priority increase kore
increase priority after a periodic time

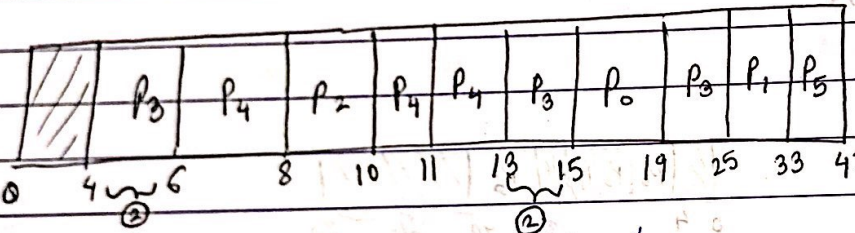
* Preemptive (SJF)

SRTF

Process	A.t.	B.t.
~~P_0~~	15	~~4~~
P_1	10	8
~~P_2~~	8	2
P_3	4	$10-2=8-2=6$
~~P_4~~	6	$5-2=3-1=2$
P_5	11	8

Preemptive - Shortest first depends on the arrival of next.

First take note of A.T. then others.



(2) P_4, P_3-8, P_1-8, P_5-8 context switch

(2) switch

w.t. ≠ r.t.

P_0 w.t. → 15 − 15 = 0

P_1 w.t. →

P_2 w.t. →

way 1

P_3 w.t. → 19 − (2) − (2) − A.t.

3 blocks = 19 − 2 − 2 − 4

= 11s

way 2

C.t − B.t − A.t

w.t. = C.t − A.t − B

P_4 w.t → 11 − 1 − 2 − 6

= 2s

~~Rt = (C.T.) - At~~ (1st)

Response time = initial − At

t.t = w.t + B.t.

(t.t.) (on)

t.t. = C.T. − A.T.

(r.s.) $P_3 = 0$

Q. P_4

(i) w.t. (ii) r.t. (iii) t.t

Ans: (i) w.t. = 11 − 1 − 2 − 6

= 2s ✓

(ii) r.t = 0 ✓

(6 − 6)

प्रश्न w.t. is my r.t.

(iii) t.t. = w.t + B.t

= 2 + 5

= 7s ✓

First'e arrive was (s) = 6s

= 6s

A.t

∴ 6 − 6 = 0s

s (At)

response time = initial − At

r.t.

highest number = highest priority

Priority

preemptive / non-preemptive

Process	A.t.	B.t.	Priority
P_0	15	4	2
P_1	10	8-1=7	46
P_2	8	2	3
P_3	4	10-2=8	1 ✓
P_4	6	5-2=3-2=1	5 ✓
P_5	11	8	1

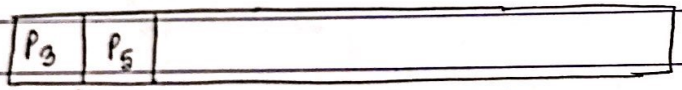
২য় Priority বেশি, তার প্রাধান্য বেশি।

* low no., high priority

1 → high

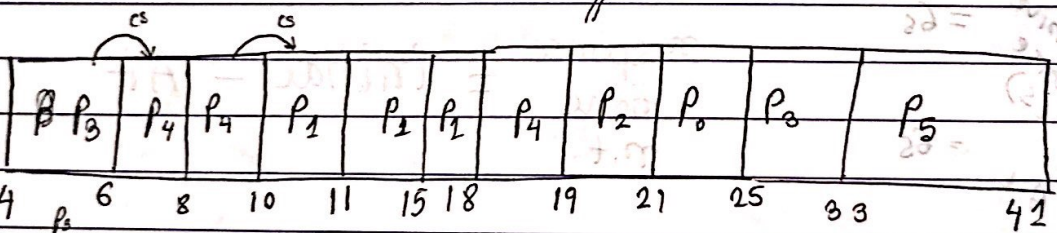
• B.T. not considered here

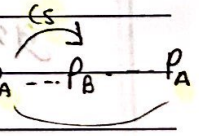
non-P



P_1
P_2
P_4
P_5

Context Switch = 2





preemptive

P_3
P_4
P_2
P_1

H.W Q. avg wt, avg t.t, avg r.t

Q. • Context-Switch? (w.t. কমানো যায়)

কিন্তু functionality bar bar korte hoy; save prev, reload new)

same priority hole, otai continue hobe

* Priority scheduling Problem

low priority process
high pp'er behind'e thaake, } Starvation
and doesn't get CPU for a
long time.

Problem: indefinite blocking or Starvation

Bound waiting

Aging - increase priority in a fixed interval / নির্দিষ্ট time পর পর

A solution to the problem of indefinite blockage of low-priority process is aging.

* Aging is a technique of gradually increasing the priority of processes that wait in the system for a long period of time.

* If At. same for all process,

$$np = p$$

because, we're not going to get a shorter process with lower B.T. (SJF)

because, we're going to pick highest priority / and no other process arrive at that time,
∴ run till completion. (Priority)

GitforBash.org

Lab-04

ctrl+l → clean

labmid → 17th Feb
Shell command, script

Regex — lab 01

$grep

[^^]

c
d — d/Project
f
f

cd /mnt
cd d
cd Project

nano text.txt

touch text.txt

e, m, t

Don2uxin

a*
aaaa

(7)

(1) abc* a*

(1½) \d

(2) \.

[Ab^]

(3) can|man|fan

(4) [^bog^]

(5) [A-Z]

^[A-Z]

(6) z{2,6}

(7) aa+

(8) \d+ files? found\?

(9) \d\.\s+abc

Lecture

Round Robin (RR)

11-02-20

preemptive because processes are assigned CPU only for a fixed slice of time at most

Starvation-free as all processes get fair share of CPU.

Disadv: Overhead of context switching

Process	B.t.	A.t.	Q.t.
P_0	~~0~~ 1	12	3
P_1	~~9~~ ~~6~~ 3	3	
P_2	~~8~~ 4	8	
P_3	13−3=10, 7	0	
P_4	~~10~~ 7	6	
P_5	2	5	

RR is a CPU scheduling algo where each process is assigned a fixed slot in a cyclic way.

RR is the algo used in Timesharing OS

P_3	P_1	P_3	P_5	P_4	P_1	P_2	P_3	P_0	P_4	P_1	P_2	P_3	P_4	P_3	P_4
0–3	3–6	6–9	9–11	11–14	14–17	17–20	20–23	23–24	24–27	27–30	30–31	31–34	34–37	37–38	38–39

Queue: ~~P3 P1 P3 P2 P4 P1 P2 P3 P0 P4 P1 P2 P5 P4 P2 P4~~

Quantime theke beshi korte parbona, but kom korte parbo.

It can never be non-p.

- OR3K of time e, insert in queue
- not finished, shift in queue

RR is always preemptive

- Content Switch = 10
- R.t. $P_1 = 3-3 = 0$
  $P_2 = 17-8 = 9s$
- w.t. $P_0 = 23 - At = 11$ (At = 12)
  $P_1 = 27-(3)-(3) - At = 18$
  $P_2 = 30 - 3 - 8 = 19s$

TQ ↑, CS ↓

- t.t. ⇒ w.t. + B.t / or / CT − At
  $= 18 + 9$ $= 30 - 3$
  $= 27s$ $= 27s$

(if TQ very large, then starvation)

TQ ↓, CS ↑ starvation

$P_2 = 31 - 8 = 23s$

w.t. = Ct − B.t. → At = t.t − Bt

t.t. = Ct − At

Qt 3 hole, RR na hoye FCFS hoye jabeh.

FCFS → np

SJF → p / np

Priority → p / np

RR → p

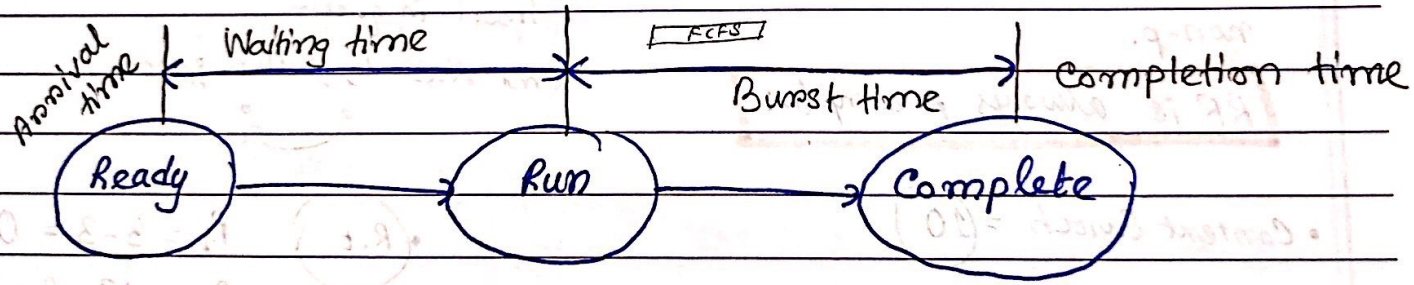
Priority RR

high p process

Qt

4

30



T.t. = C.t. − A.t [T = C − A]

W.t. = T.t − B.t. [W = T − B]

= Ct − At − Bt [W = C − A − B]

tuberculosis

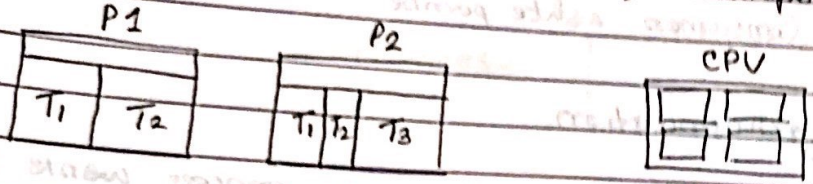
Lecture

16th Feb, 2020

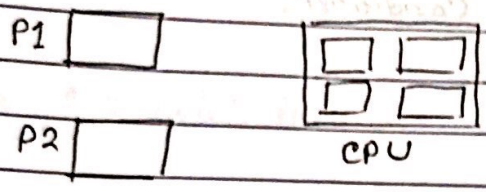
Process Synchronization

different data (process / thread), so independent.

Thread



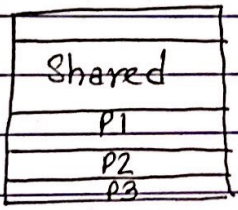
Process

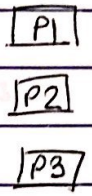


multiprocessing environment

* diff only in **management**, otherwise, not much of a diff b/n thread and process.

IPC → dependency

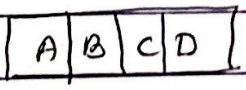


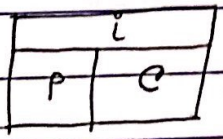


main memory

* Prob: Multiple access at the same time, shows data inconsistency

(i is a shared value)





Producer Consumer

i = 3

P

buffersize = 5

bs == i ?

"sleep"

"produce"

i++

context-switch hoye gese

C

i = 0

" sleep

// consume

i--

i=2

* Soln to Critical Section Problem *

(imp slide)

3 criteria →

1. Mutually exclusive (one at a time)
   mandatory criteria → kokhonoi ekshate duita kaaj use korte parbena
2. Progress → interested
   ... (request accepted only if space is available in order)
3. Bounded Waiting optional criteria

Criteria 1, 2 holei, the soln is an adequate one

Peterson's Solution (Two-process Soln)

```
do {
    flag[0] = True;
    turn = 1;
while (flag[1] && turn == 1);

    // Critical Section

    flag[0] = false;
    // Remainder Section;
} while (TRUE)
```

entry code: 3 lines

Statement ⇒ 2ns

P. checking: 5 lines

exit section: 1 line

∴ 9 lines
⇒ 18 ns

P.T.O.

Shared region'e, more than one process cannot be executed together otherwise data inconsistency, and no Context Switch allowed; an jodi Context Switch hoye thaake, tahole Consumer ashte pombe.

Fix ⇒ Synchronization

Shared variable (A) when more than one process wants to access, the situation is called 'Race Condition'.

Critical Section
↳ Shared region

entry section → (ensures keu ache kina)

P1
P5 \\ Critical Section
P10
P11

exit section

Context Switch holeo, I've to restrict the area. (telebooth) analogy

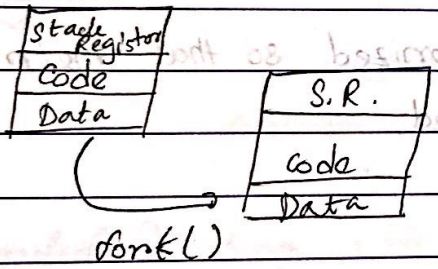
P1 checking

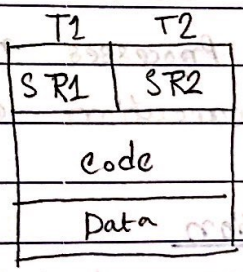
```
do {
    flag[1] = True;
    turn = 0;
while (flag[0] && turn == 0);

    // C. Section

    flag[1] = false;

    // Remainder Section
} while (TRUE)
```

entry

exit section

	Process	Threads
1.	System calls involved in Process	1. There is no system call involved.
2.	OS treats different processes differently	2. All user level threads treated as single task for OS
3.	Different processes have diff copies of data, files, code	3. Threads share the same copy of code and data
4.	Context-Switching is slower	4. C.S. is faster (less overhead)
* 5.	Blocking a process will not block another	5. Blocking a thread will block an entire process.
6.	Independent	6. ~~Independent~~ Interdependent





Processes executing concurrently in the OS might be either independent or cooperating processes.

Process Synchronization

On the basis of synchronization, processes are categorized as two types.

Process

- **Independent Process**
  Execution of one process does not affect the execution of other processes.
- **Cooperative Process**
  Execution of one process affects the execution of other processes.
  because: Share memory, resources, variable, code

* Process Synchronization problem arises in the cases of Cooperative Process also because resources are shared in Cooperative Process

When two or more processes cooperate with each other, their order of execution must be preserved, otherwise, there can be conflicts in their execution and inappropriate outputs can be produced.

Cooperative Processes need to be synchronized so that their order of execution can be guaranteed.

* Race Condition

A race condition typically occurs when two or more threads try to read, write and possibly make the decisions based on the memory that they are accessing concurrently.

#

Critical Section

[part of a program which tries to access shared resources]

The regions of a program that try to access shared resources and may cause race conditions are called Critical Section.

To avoid race condition among the processes, we need to assure that only one process at a time can execute within the critical section.

The critical section cannot be executed by more than one process at the same time.

In order to solve the Cooperative Process, our main task is to solve the Critical Section problem.

Requirements of Synchronization Mechanisms (M.P.B.)

(any soln to Critical Section must satisfy 3 criterias)

Primary

1. **Mutual Exclusion** (if one process is executing inside critical section then other process must not enter in the c.section.)

   For M.E. to prevail in the system, the process must be a uniprocessor

2. **Progress** (if one process doesn't need to execute into c.section, then it should not stop other processes to get into the critical section)

3. **Bounded Waiting** (We should be able to predict the Waiting Time for every process to get into the critical section. The process must not be endlessly waiting for getting into the c. section.)

P.S. is a classical software-based solution to the critical section problem.

Peterson's Solution

In P.S., we have two shared variables.
(i: turn variable
· interested variable

```
do {
    flag[i] = TRUE;
    turn = j;
    while ( flag[j] && turn == j );   // loop

        [ critical section ]

    flag[i] = FALSE;

        [ remainder section ]

} while (TRUE);
```

- boolean flag[i]: initialized to FALSE; initially no one is interested in entering the critical section.
- int turn: The process whose turn is to enter the critical section

Peterson's Soln preserves all three conditions:

* Mutual Exclusion is assured as only one process can access the critical section at any time.
* Progress is assured, as a process outside the c.section does not block other process from entering the critical section.
* Bounded Waiting is preserved as every process gets a fair chance.

Disadv:
- It involves busy waiting
- It is limited to two processes

Lecture

18-02-2020

Peterson Solution

```
do {
    flag[0] = True;
    turn = 1;
while (flag[1] && turn == 1);        (P0)
      F      &    T
    // critical soln;
flag[0] = false;
    // remainder;
} while (TRUE);
```

```
do {
    flag[1] = True;
    turn = 0;
while (flag[0] && turn == 0);        (P1)
       T     &    T
    // critical section
flag[1] = False;
    // remainder
} while (TRUE);
```

P_0	P_1
T	F

Criteria

- mutually exclusive
- progress
- bounded waiting

flag

P_0	P_1
F	F

flag → default: False

turn → default: can be 0 or 1

Scheduling (maths & theory)

Q. (2ns) for every line to execute

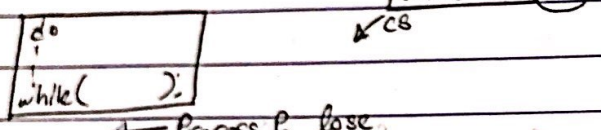
CS → 4 lines

Context Switch → 6ns নয়বার

Peterson Solle, duita process.

Ans:

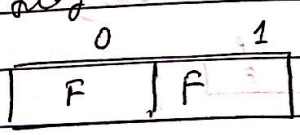
P_0 P_1

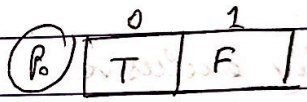


← Process P_0 lose the processor, so Context CS

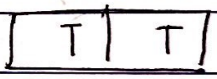
CS 4 lines

flag





turn = 1;

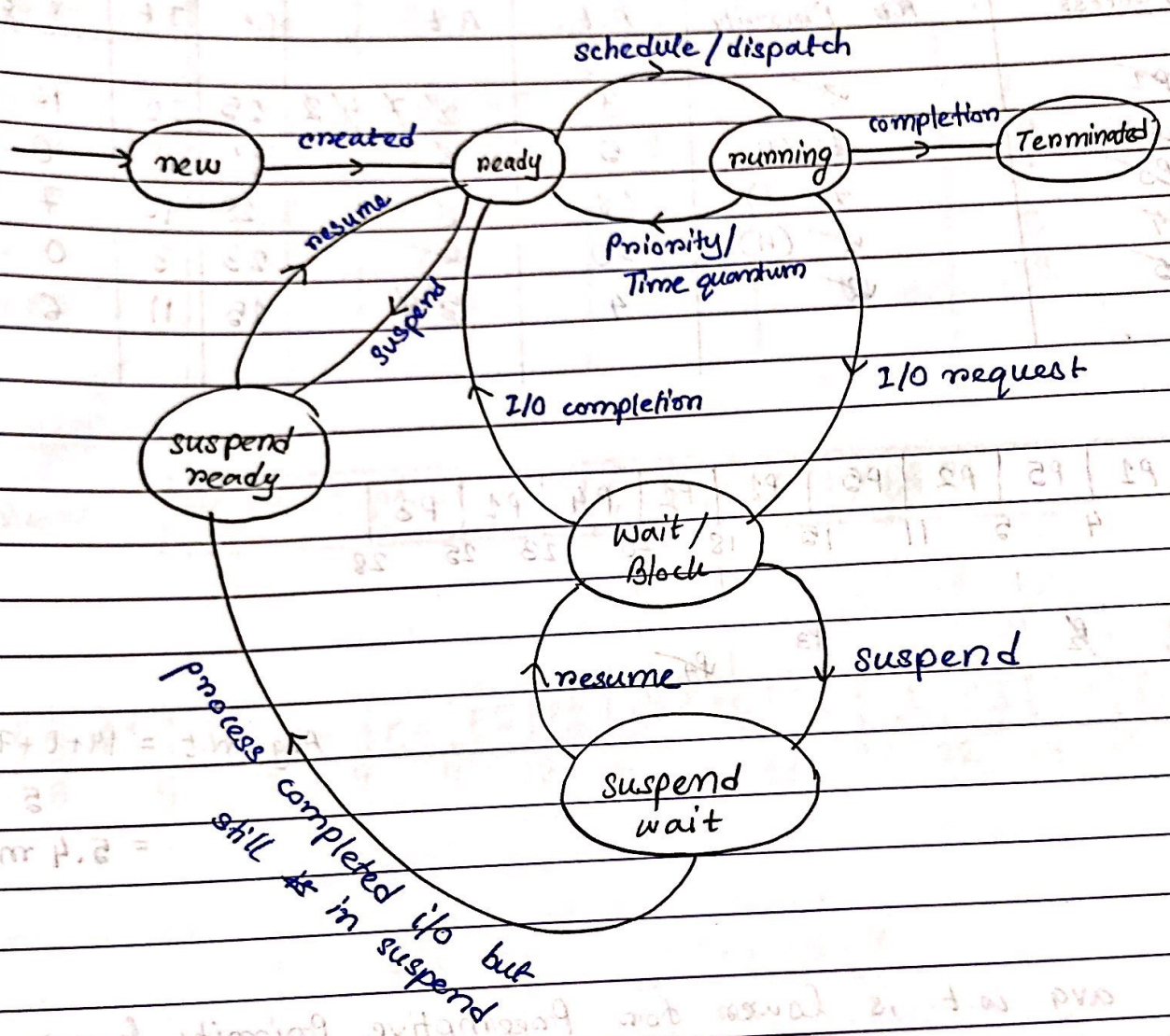


turn = 0;

[Midterm Fall 2019, Spring 2019]

Ans 2:

a) Draw process state diagram for process scheduling.



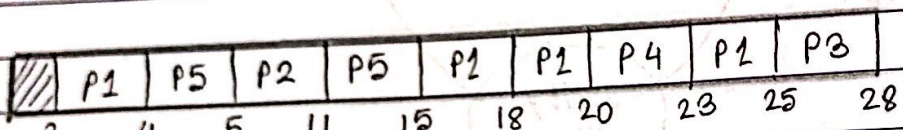
(a)

$T = C - A$

$W = T - B$

Ans 3 (a) Preemptive Priority

Process	Priority	A.t.	B.t.	Ct.	T.t	W.t.
P1	3	3	8 7 4 2	25	22	14
P2	1 (H)	5	6	11	6	0
P3	4 (L)	18	3	28	10	7
P4	1 (H)	20	3	23	3	0
P5	2	4	5 4	15	11	6



P1 P2 P3 P4 P5

Avg. W.t. $= \frac{14+0+7+0+6}{5}$

$= 5.4$ ms

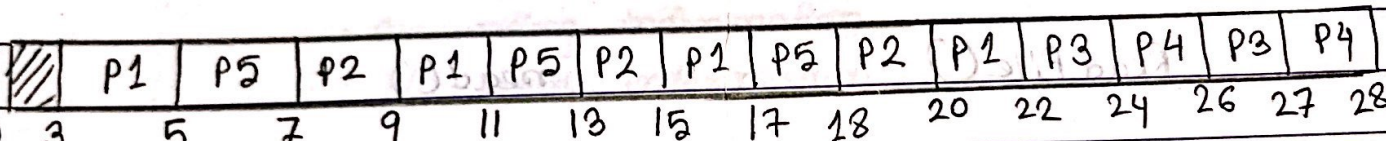
Since avg. w.t. is lower for Preemptive Priority, hence, this algorithm is the best.

Round Robin

Time Quantum = 2 ms

Process	A.t.	B.t.	C.t.	T.t.	W.t.
→ P1	3	8 6 4 2	22	19	11
→ P2	5	8 4 2	20	15	9
→ P3	18	3 1	27	9	6
→ P4	20	3 1	28	8	5
→ P5	4	5 3 1	18	14	9

Queue: P1 P5 P2 P1 P5 P2 P1 P5 P2 P1 P3 P4 P3 P4



$$\text{Avg. W.t} = \frac{11+9+6+5+9}{5}$$

$$= 8 \text{ ms}$$

Ans 1.c) Define System Call.
Mention two system calls of an OS.

System call: It is the programmatic way in which a computer program requests a service from the kernel of the OS it is executed on.

Windows	Linux
CreateProcess()	fork() **
ExitProcess()	exit()
Sleep()	sleep()
ReadFile()	read()
WriteFile()	write()
CreateFile()	open()
SetTimer()	alarm()
CreatePipe()	pipe()

* PCB is a data structure that contains information of the process related to it.
* PCB also defines the current state of the OS.

1b) What is PCB?
Mention its attributes needed for efficient Process Management.
What are the differences between STS and LTS?

PCB - Process Control Block : It is a data structure in the OS kernel containing the information needed to manage the scheduling of a particular process.

(info associated with each process) (PCB changes with the state of the process)

- attributes :
  - Process state
  - Program counter
  - CPU registers
  - CPU Scheduling information
  - Memory-management information
  - Accounting information
  - I/O status information

Location of PCB
The PCB is kept in a memory area that is protected from the normal user access.
This is done because it contains important process information.
Some of the OS place the PCB at the beginning of the kernel stack for the process as it is a safe location.

The OS maintains a PCB during the lifetime of a process; PCB is deleted when process terminated.

• Ready queue is maintained in primary (main) memory.

• To maximize CPU utilization, there must be proper scheduling of the processes.

1. c)

Difference betⁿ LTS and STS

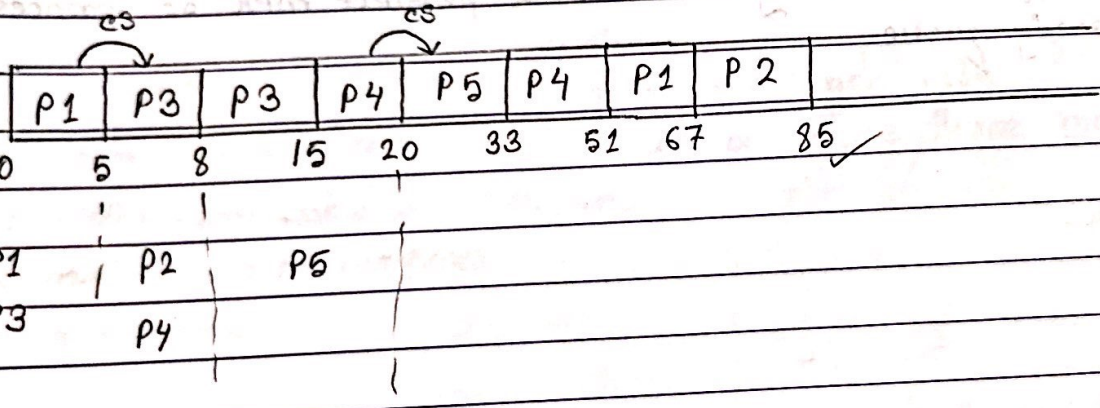
	Long-term Scheduler (Job Scheduler)	Short-term Scheduler (CPU Scheduler)
frequency 1.	• executes less frequently invoked infrequently	• executes more frequently invoked very frequently
selects 2.	• selects processes the process from the Job Pool / Job Queue and then load them in the ready queue for execution.	• selects the process from the Ready Queue and allocate CPU to it for execution.
multiprogramming 3.	* • controls the degree of multiprogramming	• less control over degree of multiprogramming.
present 4.	• present in Batch Systems, but may or may not be in Time Sharing System.	• present in Batch System, and is minimally present in the Time Sharing System.
	LTS selects which processes should be brought into the ready queue.	STS selects which process should be executed next and allocates CPU.
		problem: if STS makes mistake while selecting job, starvation may arise

1. c)

Preemptive Priority

Process	A.t.	Priority	B.t.	C.t.	T.t. = C-A	W.t = T-B
→ ~~P1~~	0	~~4~~	~~21~~ 16	67	67	46
→ P2	8	5 (L)	18	85	77	59
→ ~~P3~~	~~5~~	~~1~~ (H)	~~10~~ 7	15	10	0
→ ~~P4~~	8	~~3~~	~~28~~ 18	51	43	20
→ ~~P5~~	20	~~2~~	13	33	13	0

Σ BT = 85 ✓



P1	P2	P5
P3	P4	

Avg. waiting time = $\frac{46+59+0+20+0}{5} = 25$ s

No. of Context Switch = 2

✓

Class :

MTS - Midterm Scheduler

MTS involves swapping out a process from main memory. The process can be swapped in later from the point it stopped ~~working~~ executing.
It is also called suspending and resuming the process and it is done by MTS

* This is helpful in reducing the degree of multiprogramming.
Swapping is also useful to improve the mix of I/O bound and CPU bound processes in the memory.
Swapping is necessary to have a perfect mix of processes in the ready queue

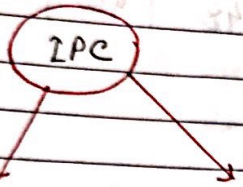
IPC (Inter-process Communication)

Q. Define IPC? Discuss two models of IPC.

Examples of IPC systems:
- Windows XP (MP)
- Posix (SM)

Cooperating processes need IPC.

IPC provides a mechanism to allow processes to communicate and to synchronize their actions (Process to process communication)

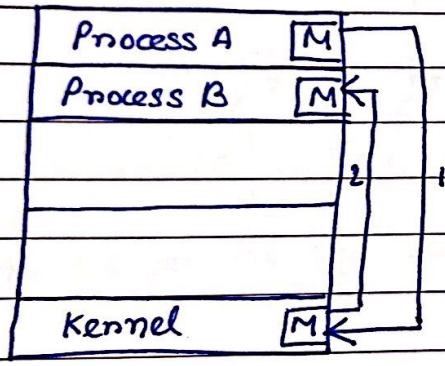


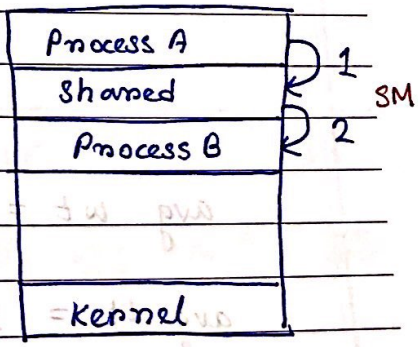
Two primary models of IPC

* Info large hole, use Shared Memory

Message Passing (MP)
- good for smaller data
- good for processes running on diff. machines in a network

Shared Memory (SM) (speed beshi)
- faster
- good for large data
- must be in the same machine





- In shared-memory model, a region of memory which is shared by cooperating processes gets established. Processes can then be able to exchange info by reading and writing all the data to the shared region.
- In the message passing model, communication takes place by wa of messages exchanged among the cooperating processes.

Q. identify one problem for each model.

Avg wt, tt, resp t.

TQ = 50

• Round Robin

3.b)

Process	A.t	B.t	C	T	W	R
→ P1	0	120 70 20	120	0	0	0
→ P2	135	102 52 2	287	152	50	0
→ P3	200	65 15	302	102	37	35
→ P4	300	148 98 48	450	150	2	2

P1 P2 P1 P2 P3 P2 P3 P4

Queue

P1	P1	P1	idle	P2	P2	P3	P2	P3	P4	P4	P4

0 50 100 120 135 185 235 285 287 302 352 402 450

P3 P2 P4 P3

avg wt = 22.25 s

avg tt = 131 s

avg rt = 9,25 s

• Round Robin

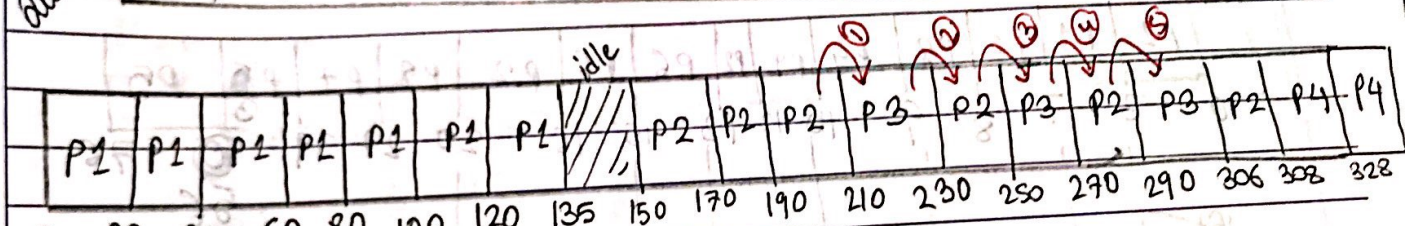
2.a)

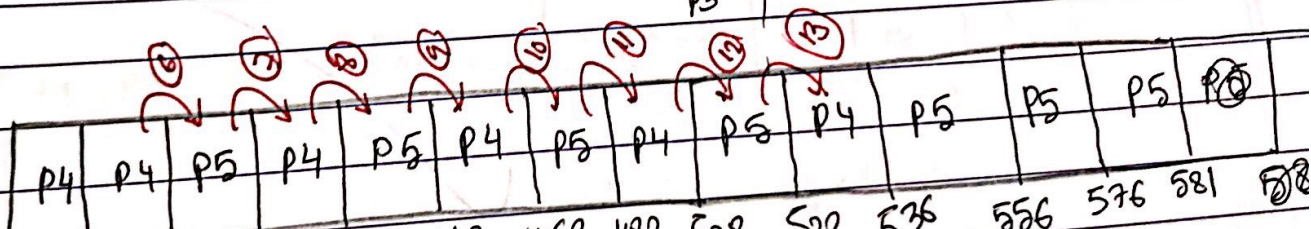
TQ = 20 time units

Process	A.t.	B.t	C	T = C-A	W = T-B
P1					
→ P2	0	135	135	135	0
→ P3	150	158	308	158	56
→ P4	200	106	306	106	50
P5	300	236	536	236	88
	380	201	581	201	76

i) GANTT chart

Queue: P1 P2 P3 P2 P3 P2 P4 P5 P4 P5 P4 P5 P4 P5 P4 P5





(ii) no. of C.S. = 13

Quantum time depends on the Burst Time.

Preemptive SRTF ⇒ SRTF

Process	At	Bt	C	T = C − A	W = T − B
P1	0	8 7 1	8	8	0
P2	3	8 5 4	20	17	11
P3	7	(1)	56	49	32
P4	9	3 2	12	3	0
P5	10	(20)	76	66	46
P6	13	6 1	16	3	0
P7	15	(12)	39	24	12
P8	20	(7)	27	7	0

P1	P2	P1	P2	P4	P4	P2	P6	P6	P2	P8	P7	P5	P5
0	3	7	8	9	10	12	13	15	16	20	27	39	56 (3) 76

P1 P2 P3 P4 P5 P6 P7 P8

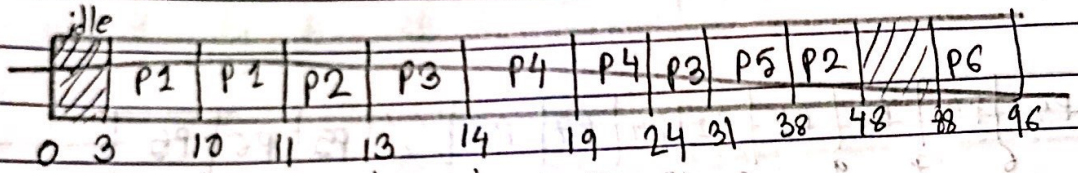
Quantum time depends on the Burst Time.

Preemptive SJF

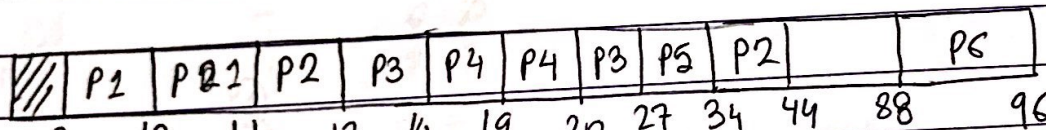
6.

Process	At.	Bt	Bt
P1	3	8 2	8 2
P2	10	12 10	12 10
P3	13	8 7	8 7
P4	14	6 5 1	6 2
P5	19	7	7
P6	88	8	8

$\sum Bt = 49$



P2
P1
P3
P4
P5



P2
P3
P4
P5

Preemptive SJF

Process	A.t	B.t
→ P0	6	7
→ P1	7	4 1
→ P2	3	1
→ P3	6	2
→ P4	4	3 1
P5	40	10 7
P6	43	3
→ P7	12	4
P8		ΣBt = 38

C	C	T	W
28	28	22	15 ✓
13	13	6	2
4	17	14	9
9	9	3	1
17	7	3	0
53	53	13	3 ✓
46	46	3	0
21	21	9	5

	P2	P4	P4	P3	P1	P1	P4	P7	P0	//	P5	P6	P5

0 3 4 6 7 9 12 13 17 21 28 40 43 46 53

P2 P0 P1
P4 P3

2 C.S.

	P2	P4	P4	P3	P2	P1	P2	P7	P0	//	P5	P6	P5

① ②

0 3 4 6 7 9 12 13 17 21 28 40 43 46 53

P2 P0 P2 P7
P4 P3

Non-preemptive Priority

(Highest priority)

Process	At.	Priority	B.t	C	I	W
→ P0	7	2 (H)	6	13	6	0
→ P1	0	3	8	27	27	24
→ P2	5	4	2	16	11	8
→ P3	0	5	7	7	7	0
→ P4	9	2 (L)	2	29	20	18
→ P5	15	4	8	24	9	1

P3	P0	P2	P5	P1	P4

0 7 16 24 27 29

13

P1
P5

Round Robin (always preemptive)

TQ = 3

Process	At	Bt
→ P0	5	6 3
→ P1	4	8 5 2
→ P2	3	7 4 1
→ P3	0	6 3
→ P4	9	2

7 context switches

P3 P2 P3 P1 P0 P2 P4 P3 P1 P0 P2 P1

Queue

P3	P2	P3	P1	P0	P2	P4	P3	P1	P0	P2	P1

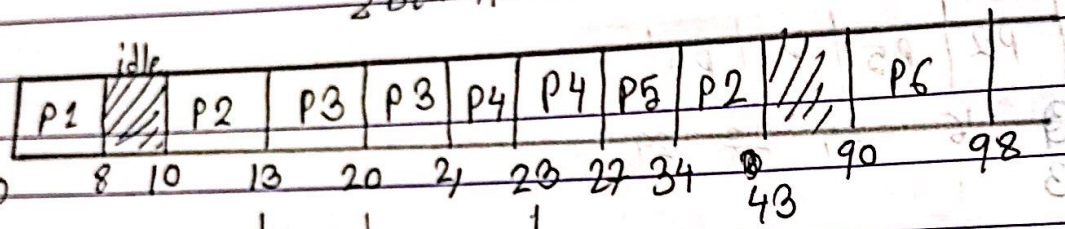
1 2 3 4 5 6 7

0 3 6 9 12 15 18 20 23 26 29 30 32

Preemptive SJF

Process	At	B.t
→ P1	0	8
→ P2	10	12 (9)
→ P3	13	(8) (1)
→ P4	20	8 (4)
→ P5	23	(7)
P6	90	8

$\sum Bt = 49$



49

P2 / P3	P4	P5

Lecture

March 01, 2020

2a (i)

$$\begin{bmatrix} 10 & -4 & 6 \\ 2 & 3 & -5 \\ 4 & -5 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 12 \\ 0 \\ 5 \end{bmatrix}$$

determinant, $|A| = 10(6\times3 - (-5\times-5)) + 4(6\times2)$

determinant, $|A| = 10\{(6\times3) - (-5\times-5)\} - (-4)\{(6\times2) - (-5\times4)\}$
$+ 6\{(-5\times2) - (3\times4)\}$

$\Rightarrow |A| = 10(-7) + 4(32) + 6(-22)$
$= -74$

$$|x_1| = \begin{bmatrix} 12 & -4 & 6 \\ 0 & 3 & -5 \\ 5 & -5 & 6 \end{bmatrix} = 12(18-25) + 4(0+25) + 6(0-15)$$
$= -74$

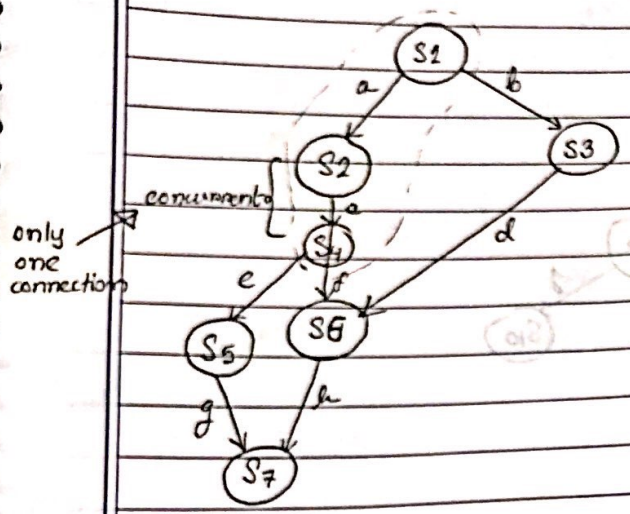
$$|x_2| = \begin{bmatrix} 10 & 12 & 6 \\ 2 & 0 & -5 \\ 4 & 5 & 6 \end{bmatrix} = 10(0+25) - 12(12+20) + 6(10-0)$$
$= -74$

$$|x_3| = \begin{bmatrix} 10 & -4 & 12 \\ 2 & 3 & 0 \\ 4 & -5 & 5 \end{bmatrix} = 10(15-0) + 4(10-0) + 12(-10-12)$$
$= -74$

Lecture

March 03, 2020

Semaphore implementation for multiple process



Semaphore

→ wait()
// CS
signal

(প্রত্যেকটা process'এর জন্য একটা variable থাকবে)

* wait() → P()
* signal() → V()

(signal S2, S3)

begin: S1 ; V(a) ; V(b) ; end ;

[signal na dile, busy waiting'e thaakte thakbe]

P(a); S2 ; S4 ; V(e) ; ~~V(d) ; end~~ V(f)

concurrent

P(a); S2 ; V(c) ; P(e) ; S4 ; V(e) ; V(f) ;

so ekbare likha jabe if concurrent

P(b); S3 ; V(d)

P(c); S5 ; V(g)

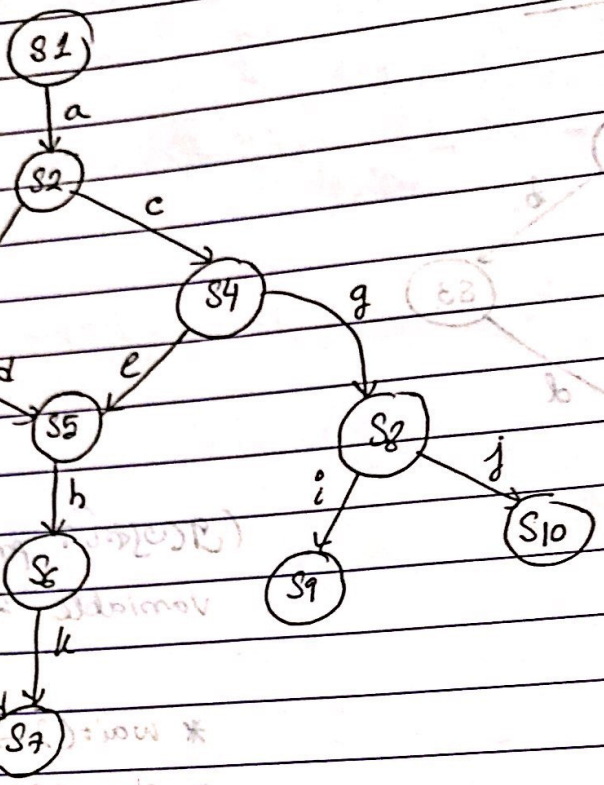
~~P(f); S6~~

P(f); P(d); S6; V(h)

P(g); P(h); S7

end

~~HHPS~~



begin: S1; V(a);

P(a); S2; V(b); V(c);

P(b); S3; V(d); V(f);

P(c); S4; V(e); V(g);

P(d); P(e); S5; V(h);

P(h); S6; V(k);

P(f); P(k); S7

P(g); S8; V(i); V(j);

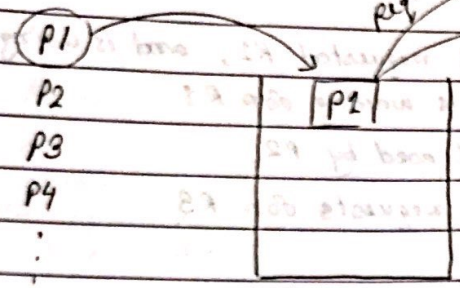
P(i); S9

P(j); S10

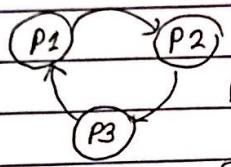
end

Deadlock

→ Process



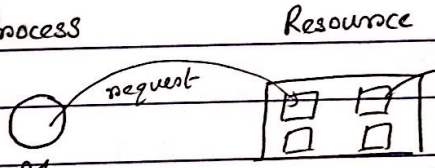
Deadlocks prevent sets of concurrent processes from completing their tasks.



deadlock

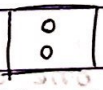
[deadlock হলে কোনো process'i execute হবেনা]

- Each process utilizes a resource as follows:
  1. request
  2. use
  3. release



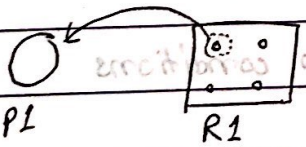
R1 → Processor

4 ta R1 type resource



R2 → Printer

2 instance i.e. 2 printers



Resource Allocation Graph

4 types of Resources

Eg. (instance)

R1 → 1

R3 → 1

R2 → 2

R4 → 3

P1 requested R1, and is using R2.

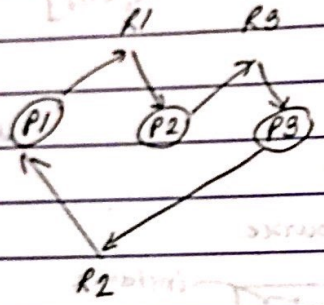
P1 is waiting for R1.

R1 used by P2.

P2 requests for R3.

no cycle, so no deadlock.

* to identify deadlock, see look for a cycle *



* first, identify a cycle.

* secondly, check if the cycle can be broken or not.

• break hole → no deadlock

Deadlock Characterization

1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait

} If one of them is NOT present in a system, no deadlock will arise

Deadlock can arise if four conditions hold * simultaneously *

Lecture

8/3/2020

Chapter 7: Deadlocks

Deadlock Characterization

Request → Resource (Req > Resource) → deadlock

Deadlock handling

- Prevention
- Avoidance *
- Detection and Termination

Avoidance

- Banker's Algorithm *

~~It is~~ This algorithm is used for deadlock avoidance.

Process	Allocation	Max	Need ⇒ (Max − Allocation)	Available
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	7 4 3	3 3 2
P_1	2 0 0	3 2 2	1 2 2	
P_2	3 0 2	9 0 2	6 0 0	
P_3	2 1 1	2 2 2	0 1 1	
P_4	0 0 2	4 3 3	4 3 1	

Banker's Algorithm →(also used for detection)

(প্রথমে process কে FALSE set করে নিবো)

• FALSE →(True, only if executed)

• P_0 → FALSE

Need ≤ Available

False

• P_1 → false

Need ≤ Available

Available = Available + Allocation

P1 → True

(next iteration)

New Available

A	B	C
~~3~~	~~3~~	~~2~~
5	3	2

• P2 → FALSE

• P3 → FALSE → True

• P4 → F → T

~~5 3 2~~

~~7 4 3~~

~~7 4 5~~

Available বারবার বাড়বে, not individual.

* Safe Sequence

P1, P3, P4, P0, P2

[এভাবে execute করলে, deadlock হবে না]

deadlock হওয়ার possibility'ও নাই, because processes executed in safe sequence manner.

• P_0 → f → T

7 5 5

• P_2 → f → T

* 10 5 7 *

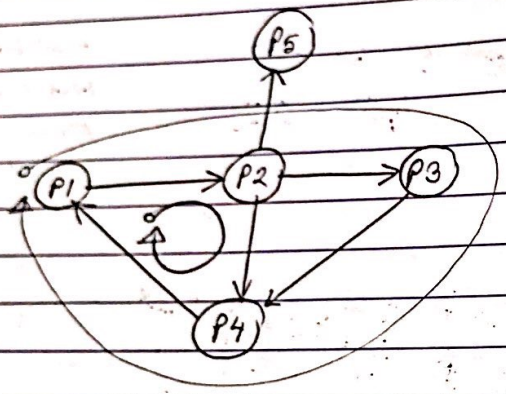
all resources relocated.

Resource - Allocation Graph and Wait-for Graph

* Single instance (only one instance for resource)
  - cycle → (deadlock)

Wait-for-graph

Process to Process



[cycle determines deadlock]

⊛ For single instance, must use wait-for-graph.

⊛ For multiple instance, wait-for-graph a lot of things taken into consideration.
cycle → possibility of a deadlock.

* Multiple instance

Lecture

10/03/2020

* ~~Max~~ ≤

Request $P_i (1,0,2)$

Need

Step 1 ⊛ Request ≤ need (if accept)

Step 2 ⊛ Request ≤ available (if accept)

- Available = Available − Req
- Need = Need − Request
- Allocation = Allocation + Request

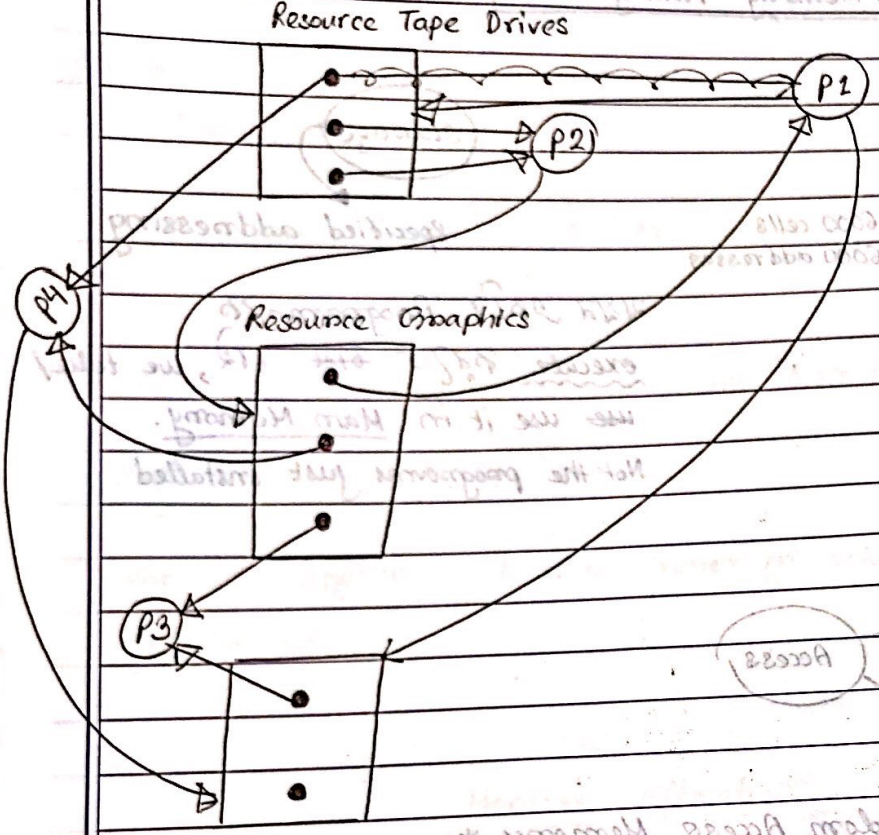
Available ⇒ ~~3 3 2~~ 2 3 0

Need ⇒ 0 2 0

Allocation ⇒ 3 0 2

Step 3 ⊛ If safe, then accept.

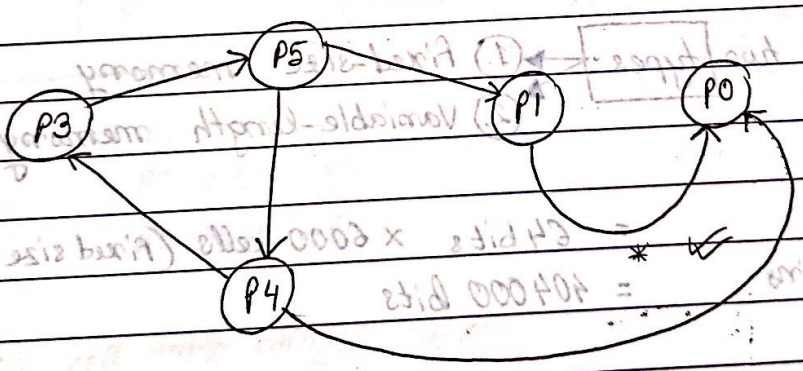
1.



- P3 chole jabeh, P2 ta resource pabe bole jabeh.

(ultimately deadlock hobena)

3.



~~deade~~ deadlock

4. C.S. ⇒ marking (ekta arekjon aslle same value teh problem ache)

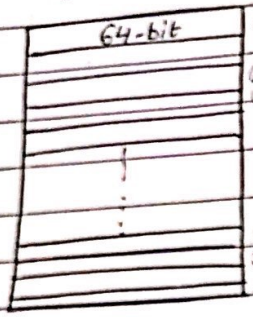
Scanned with CamScanner

16/03/2020

Lecture

Chapter 8: Memory Management

Memory



Storage

Specified addressing

যখন একটা Program-কে execute করতে চাই, we take/use it in Main Memory.

Not the programs just installed.

Main memory

(data storage)

Access

Random Access

Sequential Access

* Main Memory is Random Access Memory *

(RAM)

Main memory is of two types:

1. Fixed-size memory
2. Variable-length memory

Fixed-sized memory

Example

Total memory size = 64 bits × 6000 cells (Fixed size memory)

if each cell contains 64-bits = 404000 bits

P1 → 130 bits

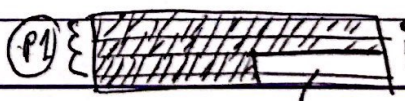
$\frac{130}{64} = 2.03 \approx 3$ cells

P2 → 50 bits

P3 → 35 bits

So,

[P1'er valid address 0-2]



fragment (unused)

Q. | 4GB | memory → 32 bit

4 → convert into bit

then divide by 32

$\frac{4 \times 10^9 \times 8 \text{ bit}}{32}$ = | addressing cells |

PC install korlei, <u>Kernel</u> is in the Main Memory initially

Main Memory সবসময় fixed size.

Variable lengthe onek beshi Addressing Schemes (for each character), hence not feasible.

Memory Allocation

- Contiguous Allocation
- Non-contiguous Allocation

---

inclusive

Contiguous Allocation

P1 → 30 cells

P2 → 500 cells

P3 → 150 cells

যতটুকু cell ~~লাগে~~ লাগবে, একই সাথে allocate করতে হবে

(range'ta eki shathe thaakte hobe)

store at the same time

Drawback: (bhenge dite parbona)

• implementing C. Allocation

- * base (kothae thaka pai'si)
- * limit (kotokhani jaiga lagbe)

} we can manage the total scheme using these variables

base → 25600

limit → 30 cells

So, P1 → ~~25600 – 25029~~ | 25600 — 25629 |

valid address range

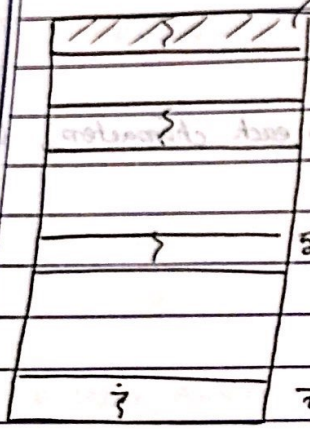
P2 → | 25630 — 26130 |

Valid, invalid } We check for these using base and limit

HW Address Protection with Base and Limit Registers.

base ≥	base + limit <

Contiguous Allocation



(P1 ⇒ 20 cells used
So, 80 cells reallocated)

Sequential Request

P1 → 20 cells
P2 → 100 cells
P3 → 30 cells

Fit Algorithm
→ optimize/utilize
best fit

requirement অনুযায়ী free-space
দেখিয়ে ache kina.

base	limit
300	100
450	30

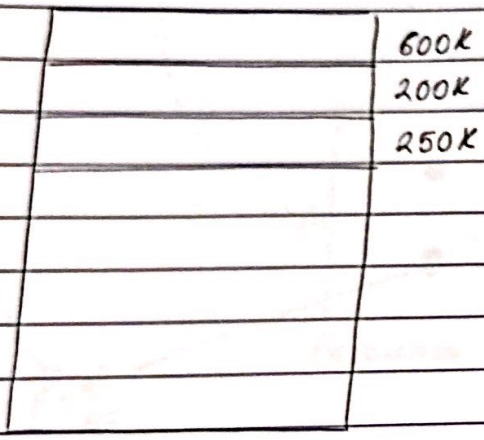
maintain a database → kon kon jaiga faka ache ⇒ base
kototuku jaiga ache ⇒ limit

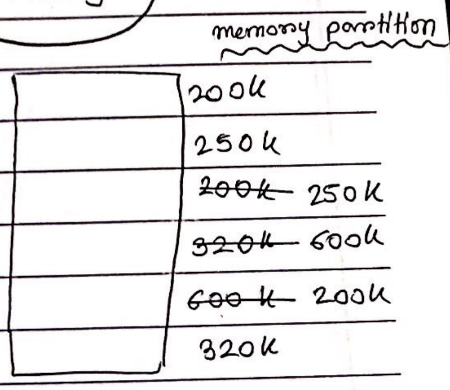
Dynamic Storage - Allocation Problem

1. First - fit
2. Best - fit
3. Worst - fit

} For Contiguous Allocation

K → kilo byte





First-fit → প্রথম first fittable, baki gula check korbena
place

(কম time complexity)

Best-fit → Check all, shobtheke কম jekhane noshto kortesi, ekdom best fittable. (shobche kom fragment jekhane create hoche)

most effective because fragment gula কম and beshi process allocate kora jae

* fragments gula Reallocatable *

Worst-fit → shobche boro jaigae boshe jabe ashar shathe shathe.

24/03/2020

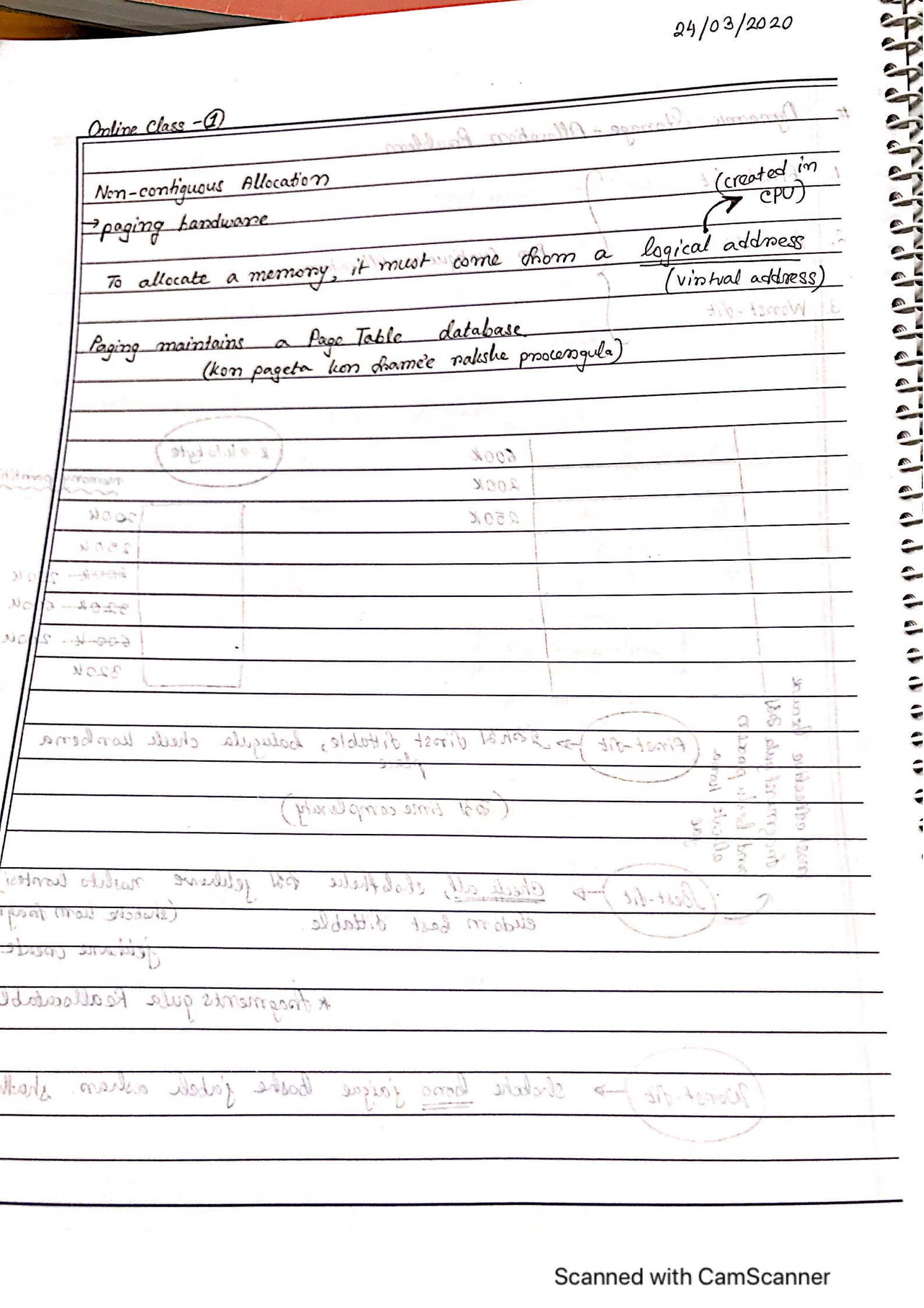
Online Class - ①

Non-contiguous Allocation

→ paging hardware

To allocate a memory, it must come from a logical address (created in CPU)
(virtual address)

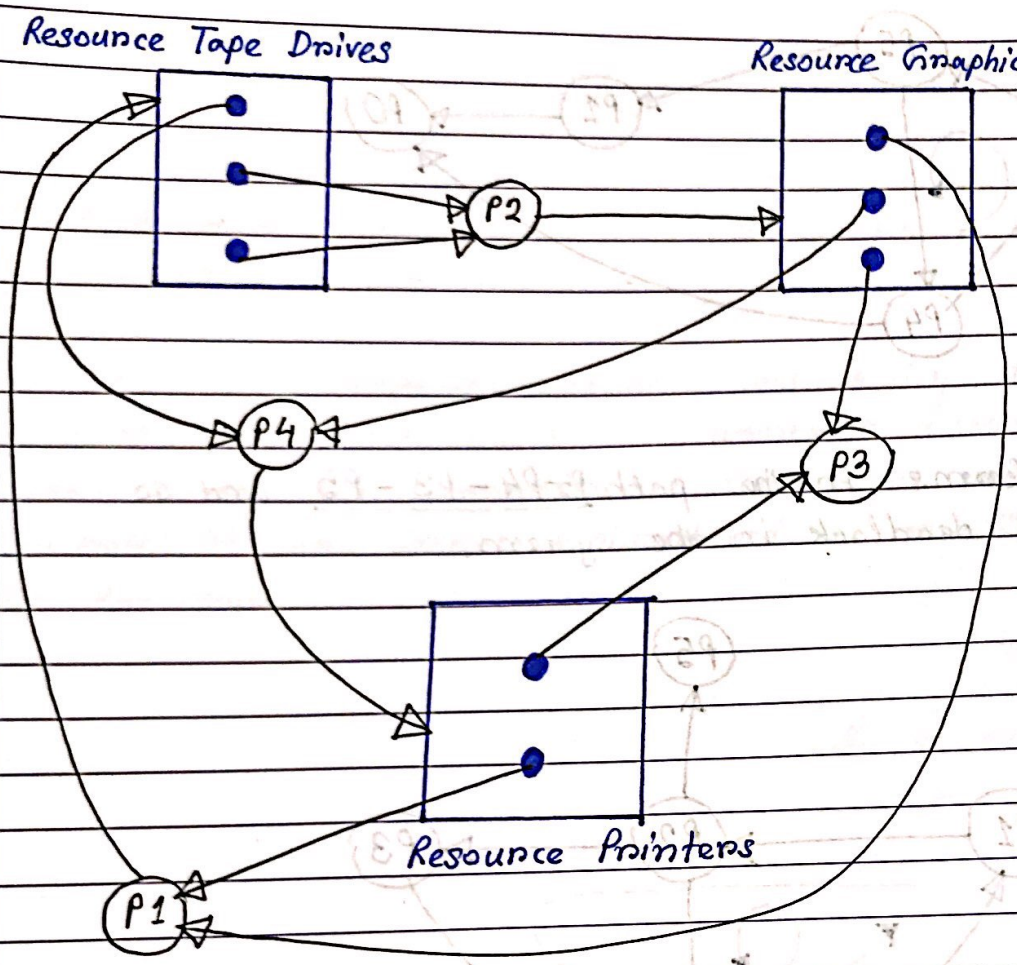
Paging maintains a Page Table database.
(kon pageta kon frame'e rakhle processgula)



CSE321 Assignment

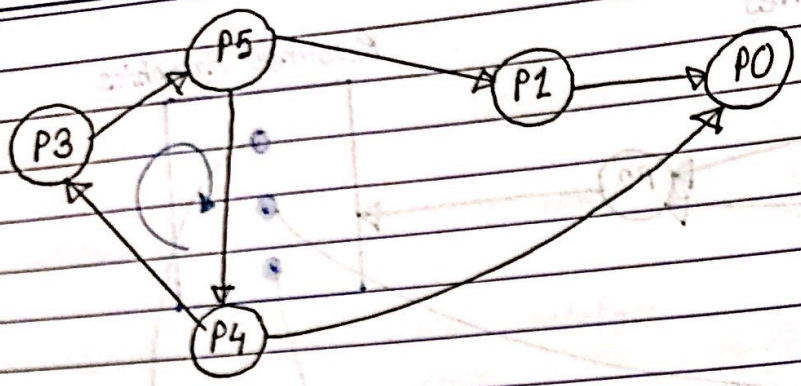
Name : Subhi Bhuiyan
ID : 17201116

1.



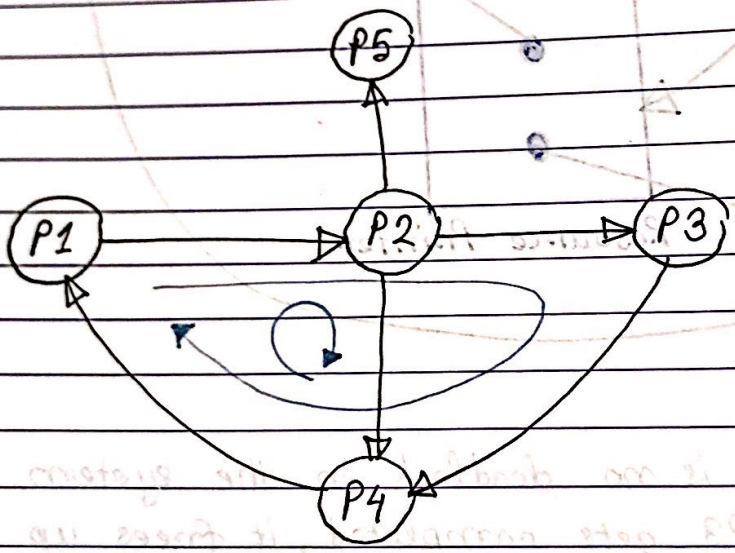
Eventually there is no deadlock in the system.
Because, when P3 gets completed, it frees up one resource instance from both Resource Graphics and Resource Printers. So, P4 will get hold of one instance from Resource Printer, and P2 will get hold of one instance of Resource Graphics. Eventually, all the processes waiting for an instance gets assigned to it, and hence there is no deadlock formed in the system.

③



A cycle forms in the path P5-P4-P3-P5 and so there is a deadlock in the system.

②



Cycle formation in path P1-P2-P3-P4-P1 and path P1-P2-P4-P1.

Since this is a single instance graph, formation of cycle indicates deadlock in the system.

(4.)

Critical Section ⇒ (Marking)

Critical Section cannot be executed by more than one process at the same time.

So while the teacher is marking individually, if another student (process) interrupts and wants to access, race condition may arise. Hence, marking is treated as an atomic instruction, and therefore, identified as the Critical Section in this case.