

Process

A process in UNIX is created by the fork system call. Every process except process 0 is created when another process executes the fork system call. Process 0 is a special process that is created when the system boots. After forking another process (Process 1) process 0 becomes the swapper process.

Process 1 known as "init" is the ancestor of every other process in the system.

`pid = fork();`

[Context Switching]

Process P₀
executing



OS

interrupt/
system call.

save state in
PCB₀.

reload state from PCB₁.

Process P₁

{
idle
period}

idle time
for P₀

@

Save
state into PCB₁

FEBRUARY						
Wk	M	T	W	Th	F	S
05						1
06	2	3	4	5	6	7
07	9	10	11	12	13	14
08	16	17	18	19	20	21
09	23	24	25	26	27	28

All our dreams can come true - if we have the courage to pursue them.
+load state from PCB₀

19

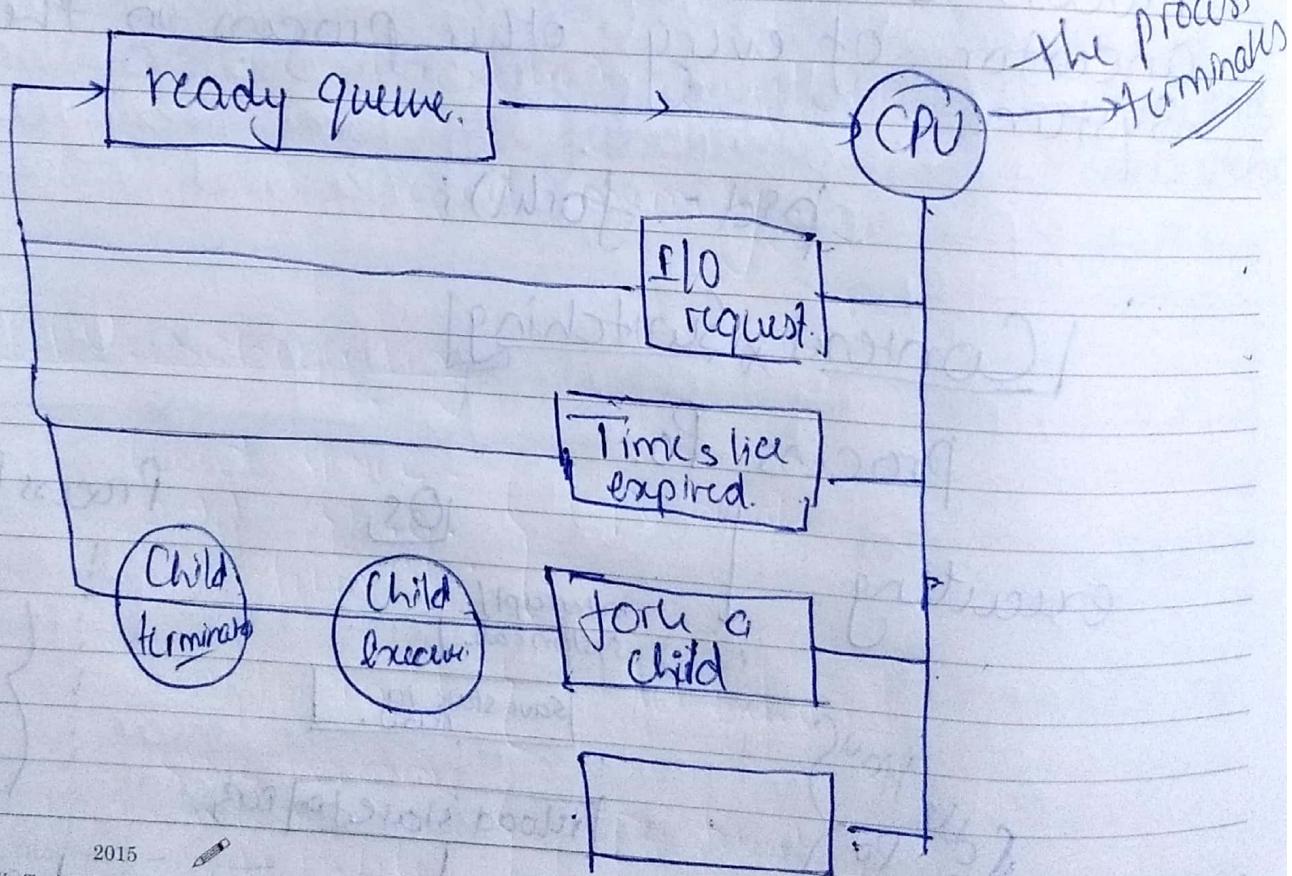
January
MONDAY
Day 019-346

2015
Week-04

- More context switching leads to higher idle time for processor
- As an operating system designer your task should be to minimize this idle time.

Process Scheduling

Objective of multiprogramming is to maximize CPU utilisation.



JANUARY 2015						
Wk	M	T	W	T	F	S
01			1	2	3	4
02	5	6	7	8	9	10 11
03	12	13	14	15	16	17 18
04	19	20	21	22	23	24 25
05	26	27	28	29	30	31

2015
Week-04

January
TUESDAY
Day 020-345

20

Scheduler

Short term Scheduler
(CPU scheduler).

long term scheduler
(Job scheduler).

Controls the degree of
multiprogramming

Process

Threads.

It is defined by the
resources it uses and
by the location at
which it executes.

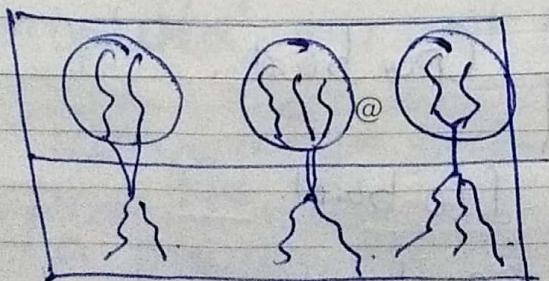
A thread is a basic
unit of CPU utilisation,
and consists of a program
counter, a register set and
a stack space.

It shares with its peer
threads (from the same
process) its code section,
data section & operating
system resources, like open
files, signals etc.

The context switching between threads
is very fast because no memory management
related work need to be done.

User

kernel.



FEBRUARY 2015						
Wk	M	T	W	T	F	S
05						1
06	2	3	4	5	6	7
07	9	10	11	12	13	14
08	16	17	18	19	20	21
09	23	24	25	26	27	28

Courage is resistance to fear; mastery of fear - not absence of fear.

21

January
WEDNESDAY
Day 021-344

2015
Week-04

1st Assignment

→ Term paper on kernel ~~and~~ threads
in different OS.

Inter, Process, Communication

- Information sharing
- Computation speed up
- Modularity
- Convenience



IPC facilities available.

(i) Shared memory.

CPU, Scheduling

load store ·
add store ·
read from file · } CPU burst.

wait for I/O } { I/O burst.

Store increment ·
index ·
write to file · } CPU burst.

wait for I/O } { I/O burst.

load store · } CPU burst.

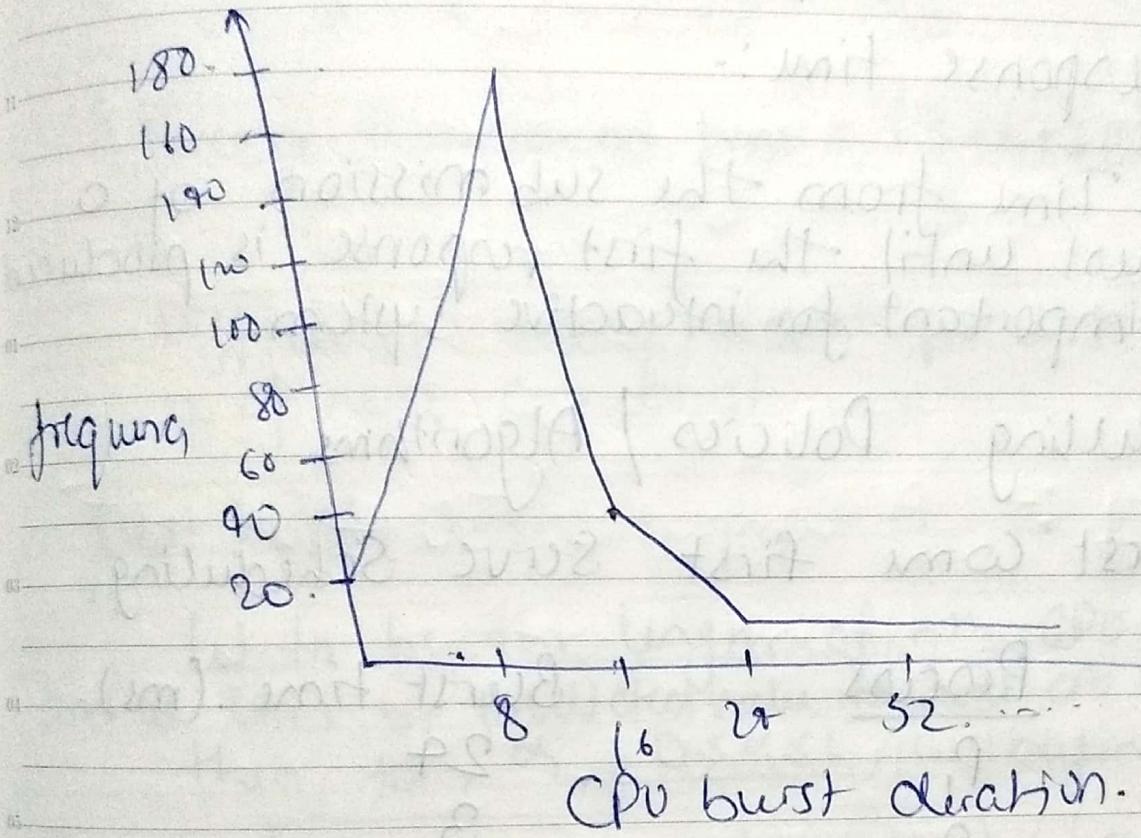
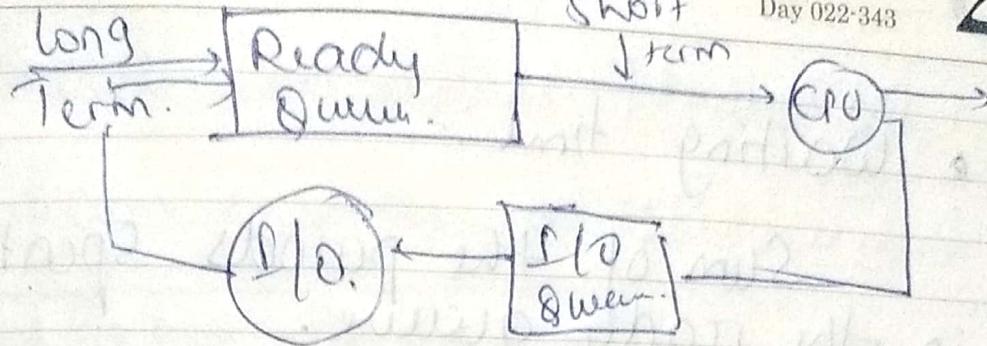
He who ~~is~~ courageous enough to take risks will accomplish nothing in life.

JANUARY 2015						
Wk	M	T	W	T	F	S
01			1	2	3	
02	5	6	7	8	9	10
03	12	13	14	15	16	17
04	19	20	21	22	23	24
05	26	27	28	29	30	31

2015
Week 04

January
THURSDAY
Day 022-343

22



Scheduling criteria.

- CPU utilization:- to keep the CPU busy as much as possible.
- Throughput: The no. of processes completed per unit time.
- Turnaround@ time- the interval from the time of submission of a process to the time of completion.

FEBRUARY 2015						
Wk	M	T	W	T	F	S
05						1
06	2	3	4	5	6	7
07	9	10	11	12	13	14
08	16	17	18	19	20	21
09	23	24	25	26	27	28

23

January
FRIDAY
Day 023-342

2015
Week-04

- 08 • Waiting time :-

09 Sum of the periods spent waiting
10 in the ready queue.

- 11 • Response time :-

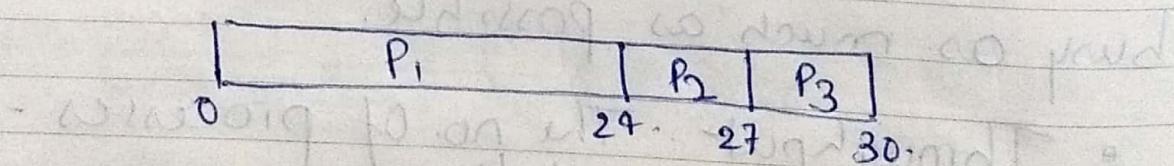
12 Time from the submission of a
request until the first response is produced.
01 is important for interactive system.

* Scheduling Policies / Algorithms.

- 03 1. first come first serve Scheduling,

<u>Process</u>	Burst time .(ms).
P ₁	24
P ₂	3
P ₃	3

07 Gantt chart.



Performance is represented by average turnaround time :

JANUARY 2015						
Wk	M	T	W	T	F	S
01		1	2	3	4	
02	5	6	7	8	9	10 11
03	12	13	14	15	16	17 18
04	19	20	21	22	23	24 25
05	26	27	28	29	30	31

$$= \frac{24 + 27 + 30}{3} = \frac{81}{3} = 27.$$

It is easy to be brave from a safe distance.

2. Shortest Job First Scheduling.

Same example as 1: (18) round

P ₂	P ₃	P ₁
0	3	6

8 = 2 + 6 30.

$$\text{Average turnaround time} = (3+6+30)/3 \\ = 13.$$

Difficulty: We need to predict the length of next CPU burst.

One Simple Way.

Let t_n be the length of n^{th} CPU burst,
and let τ_{n+1} be predicted value for next CPU burst.
then, for α , $0 \leq \alpha \leq 1$, we define

$$\boxed{\tau_{n+1} = \alpha t_n + (1-\alpha) \tau_n} \quad // \tau_n \text{ is the guess of the CPU burst at } n^{\text{th}} \text{ slot.}$$

if $\alpha=0$, then $\tau_{n+1}=\tau_n$ i.e. the recent history has no effect on prediction.

if $\alpha=1$, then $\tau_{n+1}=t_n$. i.e. only the most recent CPU burst information, history has no effect.

if $\alpha=0.5|0.7|0.3$ - for mix of both the history and current history

I'd rather give my life than be afraid to give it.

FEBRUARY 2015						
Wk	M	T	W	T	F	S
05						
06	2	3	4	5	6	7
07	9	10	11	12	13	14
08	16	17	18	19	20	21
	23	24	25	26	27	28

25

January
SUNDAY
Day 025-340

2015
Week-04

08 Actual CPU burst (H) :- 6ms, 4ms, 6ms, 4, 13
 $\Delta = 0.5$

09 Arrival (Z_i) :- 10ms, 8ms, 6ms, 6, 5,

$$Z_{i+1} = 10 \times 0.5 + 10 \times 10.5 \\ = 3 + 5 = 8.$$

$$Z_{i+2} = 8 \times 0.5 + 8 \times 0.5 \text{ ms wt. prava} \\ = 6$$

$$Z_{i+3} = 6 \times 0.5 + 6 \times 0.5 = 15$$

Priority :- A computer T094 at MIT in 1973 they found that a task of 1967 was still unexecuted.

Aging :- To gradually increase the priority of jobs that wait in a system for long time.

JANUARY 2015						
Wk	M	T	W	T	F	S
01		1	2	3	4	
02	5	6	7	8	9	10 11
03	12	13	14	15	16	17 18
04	19	20	21	22	23	24 25
05	26	27	28	29	30	31

Men are born to succeed, not fail.

January
MONDAY
Day 026-339

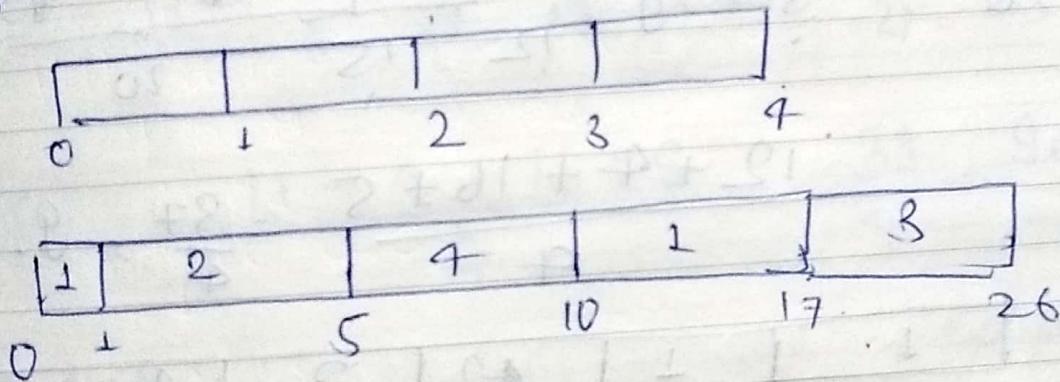
26

2015
Week-05

Premptive Algorithms.

SJF (Shortest Job first scheduling)

Job.	Arrival time	Burst time,
1	0	8
2	1	4
3	2	9
4	3	5



Average turn around time:

$$= \frac{(17-0) + (5-1) + (26-12) + (10-3)}{4}$$

$$= \frac{17+4+24+7}{4} = \frac{52}{4} = 13.$$

Non preemption.

$$\frac{(8-0) + 11 + 14 + 24}{4} = \frac{57}{4} = \underline{\underline{14.25}}$$

FEBRUARY 2015						
Wk	M	T	W	T	F	S
05						1
06	2	3	4	5	6	7
07	9	10	11	12	13	14
08	16	17	18	19	20	21
09	23	24	25	26	27	28

27

January
TUESDAY
Day 027-338

2015
Week-05

Job . Arrival Time.

BT.

Priority

08		0	8	3rd
09	1	5	4	2nd
10	3	10	9	3rd
	4	15	5	1st
				2nd.

Preemptive:-

12	1	2	3	4	5
11	X	0	5	9	12
01		.	.	12	15
02				20	26

$$\frac{12 + 4 + 16 + 5}{4} = \frac{37}{4} = 9.25$$

4	1	2	3	4	5
04	0	5	8	12	15
05		.	.	20	26
06					

$$\frac{8 + 7 + 16 + 5}{4} = \frac{36}{4} = 9.$$

ATT =

0	1	2	3	4	5
	8	10	12	19	20
					26

$$\frac{8 + 10 + 12 + 19 + 20}{5} = \frac{71}{5} = 14.2$$

JANUARY 2015						
Wk	M	T	W	T	F	S
01		1	2	3	4	
02	5	6	7	8	9	10 11
03	12	13	14	15	16	17 18
04	19	20	21	22	23	24 25
05	26	27	28	29	30	31

Do not let what you cannot do interfere with what you can do.

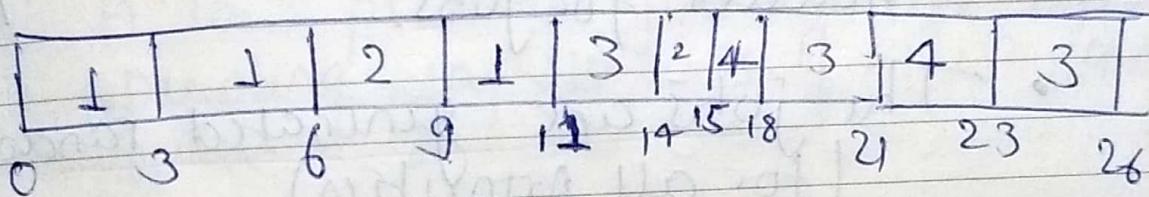
January
WEDNESDAY
Day 028-337

28

2015
Week-05

Round Robin (There should be a time quantum for every process).

<u>Job</u>	<u>Arrival time</u>	<u>Burst time</u>
1	0	8
2	5	4
3	10	9
4	15	5



$$ATF = \frac{11 + 4 + 16 + 8}{4} = \frac{39}{4} = 9.75$$

Priority

1st → System processes/Jobs.

2nd → Interactive Procs/Jobs.

3rd → Batch processing.

Ready Queue

@

FEBRUARY 2015						
Wk	M	T	W	T	F	S
05						1
06	2	3	4	5	6	7
07	9	10	11	12	13	14
08	16	17	18	19	20	21
09	23	24	25	26	27	28

One's best success comes after their greatest disappointments.

29

January
THURSDAY
Day 029-336

2015
Week-05

Assignment-2 :-

Determine the length of the "Ready Queue" used in CPU scheduling of your computer system.

- * Assume there are three(3) priority queues for jobs.
- The jobs are generated randomly (for all priorities).
- Define job sizes (say, $\frac{1}{CPU \text{ unit of time}}$, $\frac{2}{CPU \text{ unit of time}}$, $\frac{3}{CPU \text{ unit of time}}$, $\frac{4}{CPU \text{ unit of time}}$).

Input: No. of priority of jobs

Ready Queue size :- 20 / 30

Output: 1. Total number of jobs generated after simulation time of 300 time units.

2. Total jobs served by CPU
3. No. of Jobs remaining in ready queue @

JANUARY 2015						
Wk	M	T	W	T	F	S
01			1	2	3	4
02	5	6	7	8	9	10 11
03	12	13	14	15	16	17 18
04	19	20	21	22	23	24 25
05	26	27	28	29	30	31

There is only one success - to be able to spend your life in your own way.

January
FRIDAY
Day 030-335

30

2015
Week-05

Different modes:

Mode 1

Whenever a higher priority job arrives in ready queue, low priority job in CPU has to be preempted. And high priority job is to be allocated.

(b). A low priority job will age after spending say 10 units of time in ready queue.

Mode 2

(a). Here ready queue will act as first come first serve queue.

(b) The job transfer from the priority queues to ready queues will as per following probability.

1st priority is 0.5

2nd priority is 0.3

3rd priority is 0.2

@

FEBRUARY 2015						
Wk	M	T	W	T	F	S
						1
05						
06	2	3	4	5	6	7
07	9	10	11	12	13	14
08	16	17	18	19	20	21
09	23	24	25	26	27	28

We can do if you knew you would not fail?

31

January
SATURDAY
Day 031-334

2015
Week 1

Concurrency: Mutual exclusion and Synchronisation.

Management of processes and threads.

Multiprogramming: The management of multiple processes within a uni-processor system.

Multiprocessing: The management of multiple processes within a multi processor system.

Distributed processing: The management of multiple processes executing on multiple distributed computer system.

* PARAM is based on cluster computing

Concurrency is the fundamental aspect of all these abovementioned method.

Design issues:-

- Communication among processes
- Sharing of and competing for resources
- Synchronisation of the activities of multiple processes.

* Allocation of processor time to processes

JANUARY 2015						
Wk	M	T	W	T	F	S
01			1	2	3	4
02	5	6	7	8	9	10 11
03	12	13	14	15	16	17 18
04	19	20	21	22	23	24 25
05	26	27	28	29	30	31

01

February
SUNDAY
Day 032-333

2015
Week-05

Principle of Concurrency :-

In a single-processor multi-programming system, processes are interleaved in time to yield the appearance of simultaneous execution.

Eg.: If two processes use same global variable and both perform read & write on the variable 'a'.

To manage the allocation of resources in a system optimally.

PROCESS SYNCHRONIZATION

The critical section problem (CS Problem).

- Consider a system consisting of n processes $\{P_0, P_1, P_2, \dots, P_{n-1}\}$
- Each process has a segment of code, called a critical section, in which the process may be changing common variables, update a table, writing a file, and so on.
- When one process is executing in its critical section, no other process is to be allowed to execute in its critical section.

FEBRUARY 2015						
Wk	M	T	W	T	F	S
05					1	
06	2	3	4	5	6	7
07	9	10	11	12	13	14
08	16	17	18	19	20	21
09	23	24	25	26	27	28

Success usually comes to those who are too busy to be looking for it.

2015
Week 06

February
MONDAY
Day 033-332

02

General Structure of a typical process pi.
repeat.

Entry section

critical section

Exit section

remainder
section.

until false;

Two process solution ($P_i \& P_j$)

Algo 1.

Structure of process P_i
repeat.

// a common integer
variable called
"turn" initialized
to either i or j.

while turn $\neq i$ do no op

critical section.

turn = j;

It requires
strict alternation.

remainder section

until false;

Success is the sum of small efforts, repeated day in and day out.

MARCH 2015						
Wk	M	T	W	T	F	S
09	30	31				1
10	2	3	4	5	6	7
11	9	10	11	12	13	14
12	16	17	18	19	20	21
13	23	24	25	26	27	28

03

February
TUESDAY
Day 034-331

2015
Week 9

Issue: If process j has wide gap between two critical sections than process j then j will have to wait for j in critical section of arrive.

;

$\equiv j.cs_1$

$\equiv j.cs_2$

$\equiv j.cs_3$

j

$\equiv j.cs_j$

\equiv

$\equiv j.cs_{j_2}$

Algorithm.2:- var flag: array[0 ... 1] of boolean

Structure of process j

repeat

$flag[i] = true$

while $flag[j]$ done. op

Critical section

~~Exit section~~
 $flag[i] = false$

Remainder_@ section

until false;

FEBRUARY 2015						
Wk	M	T	W	T	F	S
05						1
06	2	3	4	5	6	7
07	9	10	11	12	13	14
08	16	17	18	19	20	21
09	23	24	25	26	27	28

Do a little more each day

2015

Week-06

February
WEDNESDAY
Day 035-330

04

Algorithm 3: Structure of process pi

repeat

flag[i] = true

turn = j

while (flag[j] and turn == j) do noop;

Critical section ;

flag[i] = false .

Remainder. Section

Until false;

MARCH 2015						
Wk	M	T	W	T	F	S
09	30	31				1
10	2	3	4	5	6	7
11	9	10	11	12	13	14
12	16	17	18	19	20	21
13	23	24	25	26	27	28

People rarely succeed unless they have fun in what they are doing.

05

February
THURSDAY
Day 036-329

2015
Week 08

08

09

10

11

12

01

02

03

04

05

06

07

FEBRUARY 2015						
Wk	M	T	W	T	F	S S
05					1	
06	2	3	4	5	6	7 8
07	9	10	11	12	13	14 15
08	16	17	18	19	20	21 22
09	23	24	25	26	27	28



@

Success is never wondering whether you're

2015
Week-06

February
FRIDAY
Day 037-328

06

Semaphore:

Usage: we can use semaphore to deal with the n-process critical section problem

"mutex" initialize to 1

Structure of process p_i is organized as

repeat

[wait(mutex);

critical section.

Px

[Signal(mutex);

remainder section.

until false;

Counting Semaphore implementation with binary semaphore

Let S be a counting semaphore. To implement it in terms of binary semaphores we need the following data structure:-

Var S1: binary semaphore.
S2: binary semaphore.
S3: binary semaphore.
C: integer;

MARCH 2015						
Wk	M	T	W	T	F	S
09	30	31				1
10	2	3	4	5	6	7
11	9	10	11	12	13	14
12	16	17	18	19	20	21
13	23	24	25	26	27	28
						29

It takes 20 years to make an overnight success.

07

February
SATURDAY
Day 038-327

2015
Week 13

Initially $S_1 = S_3 = 1$, $S_2 = 0$. and the value of integer c is set to the initial value of the counting semaphore S (No. of processes you want to allow simultaneously in CS).

Entry section \rightarrow Wait operation:-

wait(S_3);

wait(S_1);

$c = c - 1$;

if ($c < 0$)

then begin

signal(S_1);

wait(S_2);

end. end

else. signal(S_2)

signal(S_3).

Exit section \rightarrow Signal operation.

wait(S_1);

$c = c + 1$.

if $c \geq 0$ then signal(S_2).

Signal(S_1);

FEBRUARY 2015						
Wk	M	T	W	T	F	S
05					1	
06	2	3	4	5	6	7
07	8	10	11	12	13	14
08	15	17	18	19	20	21
09	23	24	25	26	27	28

@

Set your goals high, and...

2015
Week-06

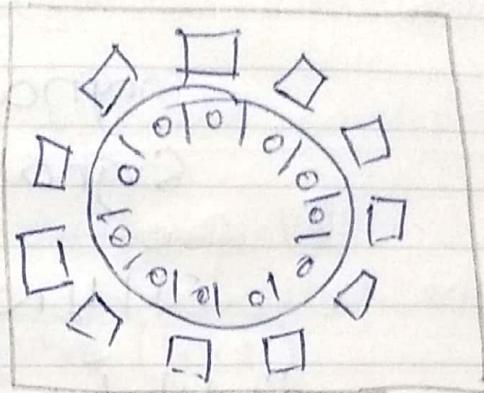
February
SUNDAY
Day 039-326

08

08 Classical Problem of Synchronization:

The Dining Philosophers problem.

Consider a no. of philosophers who spend their lives thinking and eating.



- When a philosopher thinks he/she does not interact with his colleagues.
- from time to time a philosopher gets hungry and tries to pick up the two chopsticks that are in his both sides of the plate.
- When one finishes eating, he puts down both of its chopsticks.

One Simple Solution: We represent each Chopstick by a Semaphore.

Var chopsticks: array [0...n-1] of semaphore.
all elements of chopsticks are initialised to 1.

Structure of philosopher :

repeat

wait (chopstick[i])

MARCH 2015						
Wk	M	T	W	T	F	S
09	30	31				1
10	2	3	4	5	6	7
11	9	10	11	12	13	14
12	16	17	18	19	20	21
13	23	24	25	26	27	28
						29

First say to yourself what you would be; and then do what you have to do.

09

February
MONDAY
Day 040-325

wait (Chopstick [i+1 mod n]).

EAT

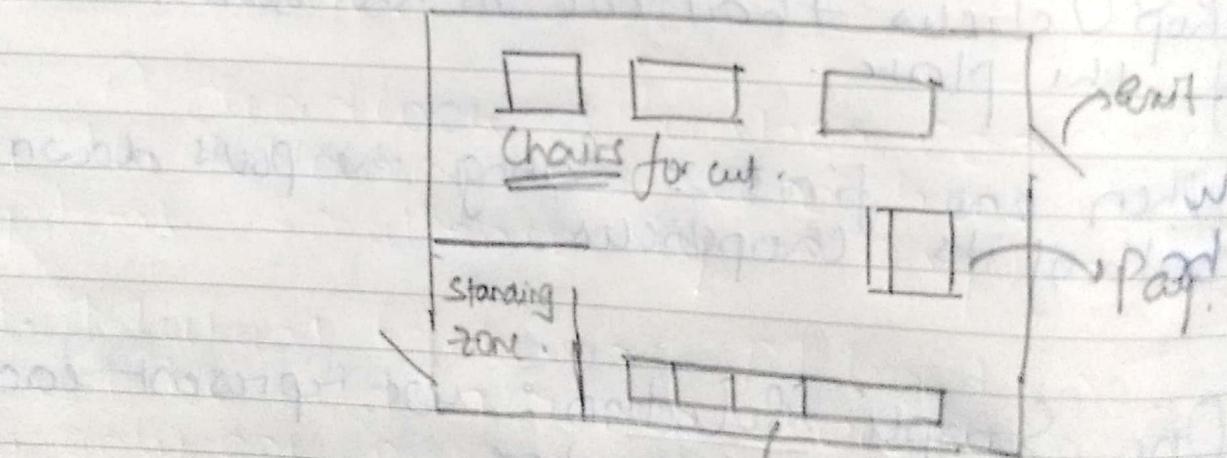
signal (Chopstick [i+1 mod n]).

Signal (Chopstick [i]).

THINK

Until false.

BABBER x SHOP x PROBLBM



- Shop has three chairs and three barbers.
- A waiting area that can accommodate few customers on a sofa.

FEBRUARY 2015						
Wk	M	T	W	T	F	S
05			1			
06	2	3	4	5	6	7
07	8	9	10	11	12	13
08	14	15	16	17	18	19
09	20	21	22	23	24	25
	26	27	28			

- Has a standing zone for additional customers.

By losing your goal, You have lost your way.

2015
Week-07

February
TUESDAY
Day 041-324

10

→ Total no. of customers in the shop can be 20.

Semaphore max-capacity = 20;

Semaphore sofa = 4;

Semaphore barber-chair = 3;

Semaphore coord = 3

void customer().

{
wait(max-capacity);
wait();

enter_shop();

wait(sofa);

sit-on-sofa();

wait(barber-chair);

get-up-from-sofa();

signal(sofa);

sit-in-barber-chair();

signal(cost-ready);

wait(finished);

leave-barber-chair();

signal(leave-b-chair);

pay();

signal(payment);

wait(receipt);

exit-shop();

signal(max-capacity);

void barber().

{
while(true){

wait(cost-ready);

wait(coord);

cut-hair();

signal(coord);

signal(finished);

wait(leave-b-chair);

signal(barber-chair)

void cashier().

{
while(true){

wait(payment);

wait(coord);

accept-pay();

signal(coord);

signal(coord);

FEBRUARY 2015						
Wk	M	T	W	TH	F	S
1						
2						
3						
4						
5						
6						
7						
8						
9						
10	1	2	3	4	5	6
11	7	8	9	10	11	12
12	13	14	15	16	17	18
13	19	20	21	22	23	24
14	25	26	27	28	29	

11

February
WEDNESDAY
Day 042-223

2015
Week-07

DEADLOCKS

Several processes may compete for a finite no. of resources.

System Model

A process may utilise a resource in the following sequence.

1. Request.

2. Use

3. Release.

A deadlock situation can arise if all the following four conditions hold simultaneously in a system:

1. Mutual Exclusion: At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource. Others have to wait.

2. Hold and Wait: A process must be there which is holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

FEBRUARY 2015						
Wk	M	T	W	T	F	S
05					1	
06	2	3	4	5	6	7
07	9	10	11	12	13	14
08	16	17	18	19	20	21
09	23	24	25	26	27	28

People with goals succeed because they know where they're going.

2015

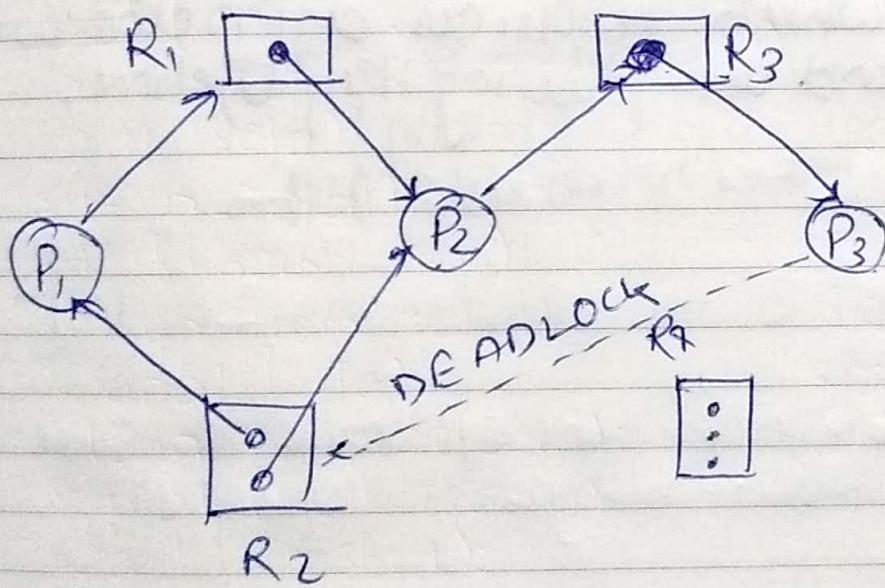
Week-07

February
THURSDAY
Day 043-322

12

3. No Preemption:- Resources cannot be preempted
4. Circular wait:- There must exist a set of waiting processes $\{P_0, P_1, \dots, P_n\}$ such that P_0 is waiting for a resource held by P_1 , P_1 is waiting for P_2 ... P_{n-1} for P_n and P_n for P_0 .

Resource Allocation Graph.



$$P = \{P_1, P_2, P_3\} \quad R = \{R_1, R_2, R_3, R_4\}$$

$$E = \{ P_1 \rightarrow R_1, P_2 \rightarrow R_3 \}$$

{

@

MARCH 2015						
Wk	M	T	W	T	F	S
09	30	31				1
10	2	3	4	5	6	7
11	9	10	11	12	13	14
12	16	17	18	19	20	21
13	23	24	25	26	27	28

You will never find time for anything. If you want time you must make it.

13

February
FRIDAY
Day 044-321

2015
Week-07

Methods of Handling Deadlocks

Deadlock Prevention by ensuring that at least one of the recovery conditions can hold

Deadlock Avoidance

Requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime.

FEBRUARY 2015						
Wk	M	T	W	T	F	S
05					1	
06	2	3	4	5	6	7
07	9	10	11	12	13	14
08	16	17	18	19	20	21
09	23	24	25	26	27	28



@

It is

2015
Week-07

February
SATURDAY
Day 045-320

14

Deadlock Prevention

08 Circular Wait is a cause of deadlock.

09 10 Order the resource types and each process requests in an increasing order.

11 Let $R = \{R_1, R_2, \dots, R_m\}$ be the set of resources types we assign to resource type α unique integer number.

$$f: R \rightarrow N$$

$$f(\text{printer}) = 1$$

$$f(\text{disk drive}) = 5$$

03 The

MAR

APR

MAY

@

MARCH 2015						
Wk	M	T	W	T	F	S
09	30	31				1
10	2	3	4	5	6	7
11	9	10	11	12	13	14
12	16	17	18	19	20	21
13	23	24	25	26	27	28

It takes time to build a castle.

15

February
SUNDAY
Day 046-319

2015
Week-07

08 Deadlock & Avoidance :-

09 Safe State :- A state is safe if the system
10 can allocate resource to each process
11 (upto its maximum) in some order and
12 still avoid deadlock. A system is safe
13 if there exists a safe sequence.

12 Ex :- We consider a system with 12
magnetic tapes drives and three processes
13 Max. Need. Current Need.
P₀ 10 5

P₁

4

2

P₂

9

2/3 → deadlock
→ safe state

At time t₀, whether the system is in safe state
or not. If safe, in what sequence?

FEBRUARY 2015						
Wk	M	T	W	T	F	S
05					1	
06	2	3	4	5	6	7
07	8	9	10	11	12	13
08	14	15	16	17	18	19
09	20	21	22	23	24	25
	26	27	28			

@

2015
版權所有

February
MONDAY
Day 047-318

16

2

MARCH							2015
Wk	M	T	W	T	F	S	S
09	30	31					1
10	2	3	4	5	6	7	8
11	9	10	11	12	13	14	15
12	16	17	18	19	20	21	22
13	23	24	25	26	27	28	29

Today is the tomorrow we worried about yesterday.

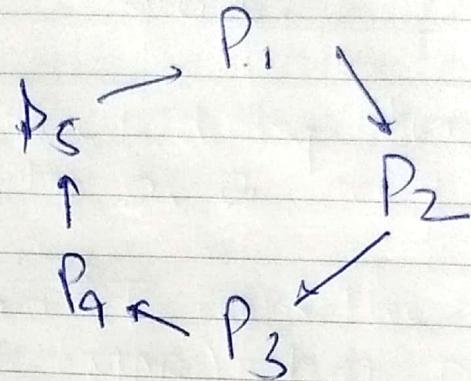
17

February
TUESDAY
Day 048-317

2015
Week 08

Recovery from Deadlock.

Process Termination



- Abort one process at a time

To choose a process:

- What is the priority of the process?
- How long the process has completed and how long is required?
- How many and what types of resources the process has used?
- How many more resource a process needs to complete its execution?
- Whether the process is interactive or batch?

FEBRUARY 2015						
Wk	M	T	W	T	F	S
05						1
06	2	3	4	5	6	7
07	8	9	10	11	12	13
08	14	15	16	17	18	19
09	20	21	22	23	24	25
	26	27	28			

@

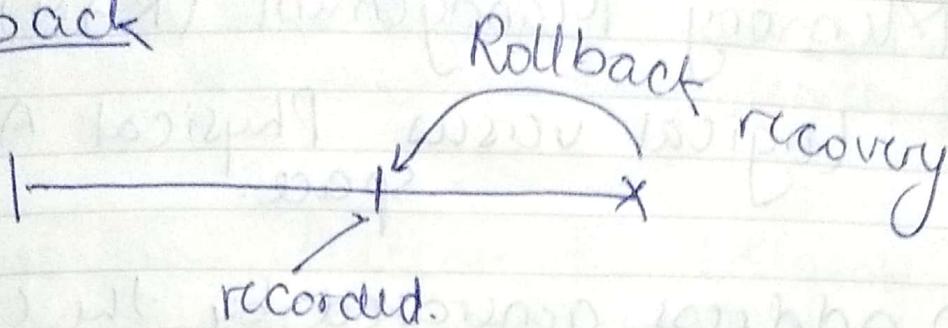
One thing you can't recycle is wasted time.

2015

Week 08

February
WEDNESDAY
Day 049-316

18

Rollback

Barbershop Problem (Revisit).

Semaphore max_capacity = 20;

Semaphore sofa = 4;

Semaphore barber-chair = 3, word = 3;

Semaphore mutex = 1, mutex2 = 1;

Semaphore cust_ready = 0, leave_b_chair = 0,

payment = 0, receipt = 0.

Semaphore finished [100] = {0};

int counts;

MARCH 2015						
Wk	M	T	W	T	F	S
09	30	31				1
10	2	3	4	5	6	7
11	9	10	11	12	13	14
12	16	17	18	19	20	21
13	23	24	25	26	27	28

You may delay, but time will not.

19

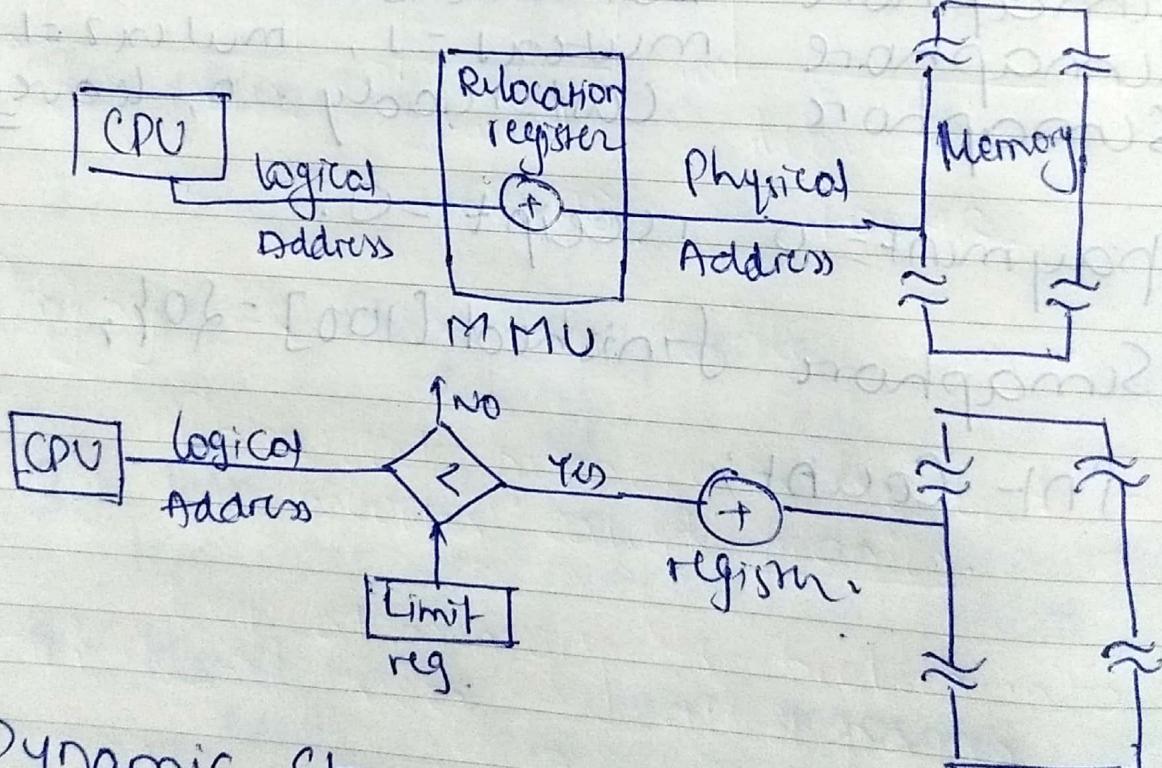
February
THURSDAY
Day 050-315

2015
Week 08

Memory Management (RAM).

Logical versus Physical Address space.

- An address generated by the CPU is commonly referred to as a "logical address".
- An address loaded into the "Memory Address Register" (MAR) of a memory is commonly referred to as a physical address.



Dynamic Storage Scheme

- first-fit: Allocate the first hole that is big enough and stop searching as soon as a free hole is found.

FEBRUARY 2015						
Wk	M	T	W	T	F	S
05				1		
06	2	3	4	5	6	7
07	8	9	10	11	12	13
08	14	15	16	17	18	19
09	20	21	22	23	24	25
	26	27	28			

We must use time as a tool, not a

2015
Week 08

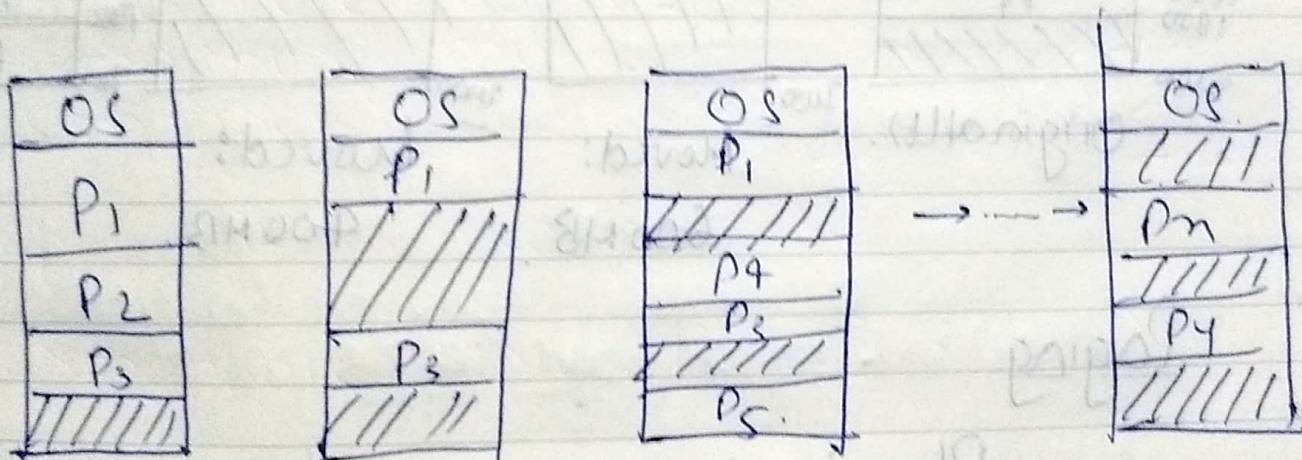
February
FRIDAY
Day 051-314

20

- Best-fit: Allocate the smallest hole that is big enough such search for the entire list.

- Worst-fit: Allocate the largest hole. Search for the entire list.

Best fit leaves out many small holes in the memory which remains unused over time leading to wastage of memory resource. So, Best-fit is not a good idea!



External fragmentation

Internal fragmentation

@

MARCH 2015						
Wk	M	T	W	T	F	S
09	30	31				1
10	2	3	4	5	6	7
11	9	10	11	12	13	14
12	16	17	18	19	20	21
13	23	24	25	26	27	28

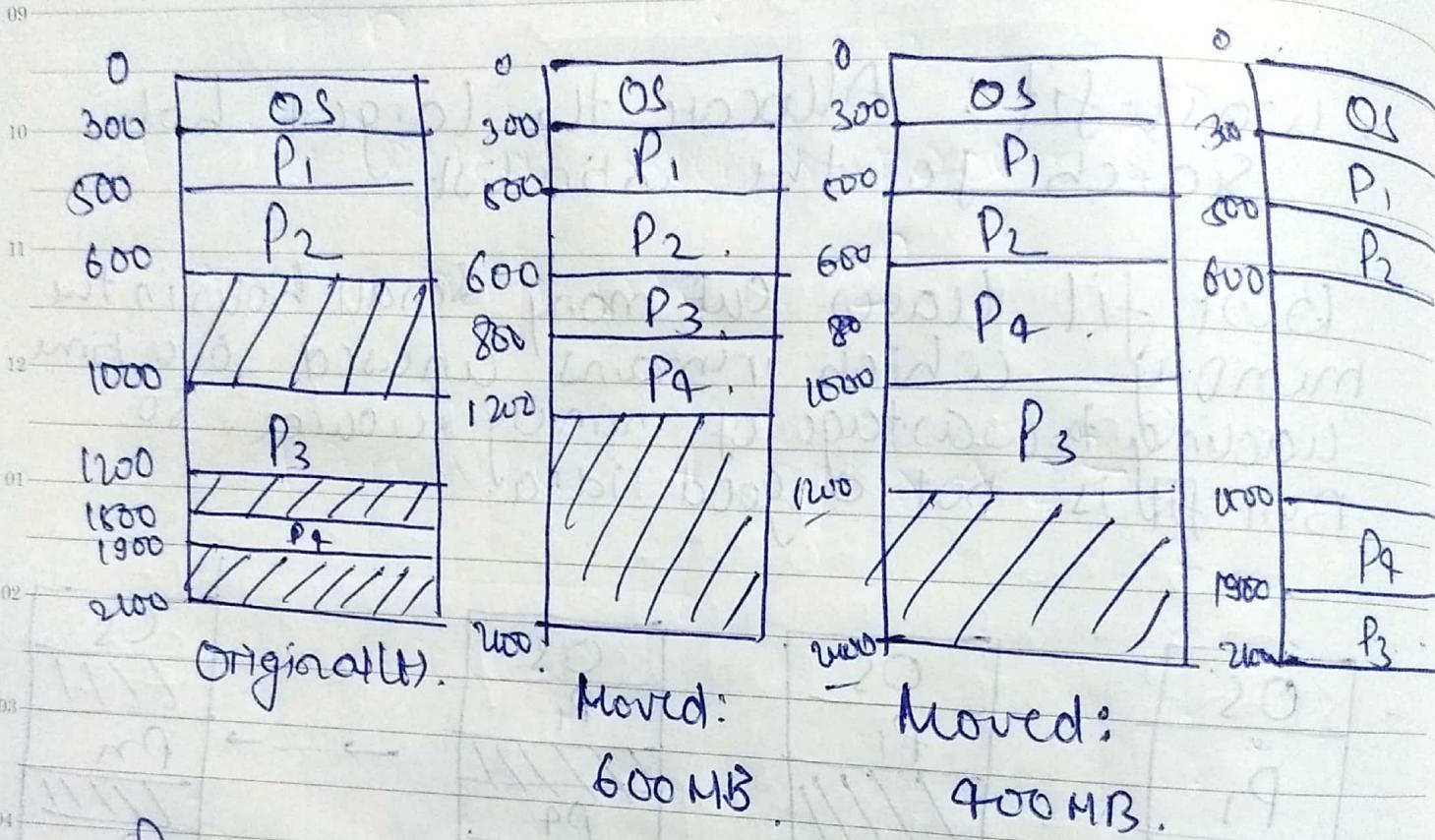
Nothing is worth more than this day.

21

February
SATURDAY
Day 052-313

2013
Week 10

Compaction: Compaction is possible only if relocation is dynamic (not static).



Paging

Physical memory is broken into fixed size blocks called frames.

Logical memory is broken into blocks of the same size called pages.

The secondary storage (hard disk) is also divided into fixed size blocks.

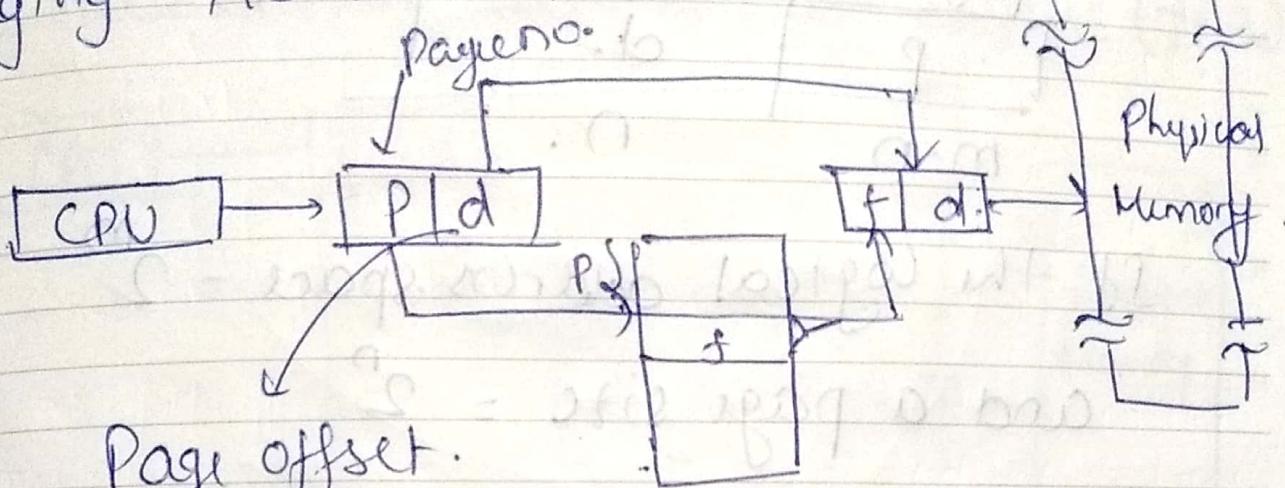
FEBRUARY 2015						
Wk	M	T	W	T	F	S
05						
06	2	3	4	5	6	7
07	9	10	11	12	13	14
08	16	17	18	19	20	21
09	23	24	25	26	27	28

2015
Week-08

February
SUNDAY
Day 053-312

22

Paging Hardware



Page offset.
(displacement
within the
page).

MARCH 2015						
Wk	M	T	W	T	F	S
09	30	31				1
10	2	3	4	5	6	7
11	9	10	11	12	13	14
12	16	17	18	19	20	21
13	23	24	25	26	27	28

14/02/22

23

February
MONDAY
Day 054-311

Page no. Page offset.

	P	d.
08		
09	m-n	n.

If the logical address space = 2^m and a page size = 2^n Page size = 4 bytes $\Rightarrow 2^2$ Structure of a page table

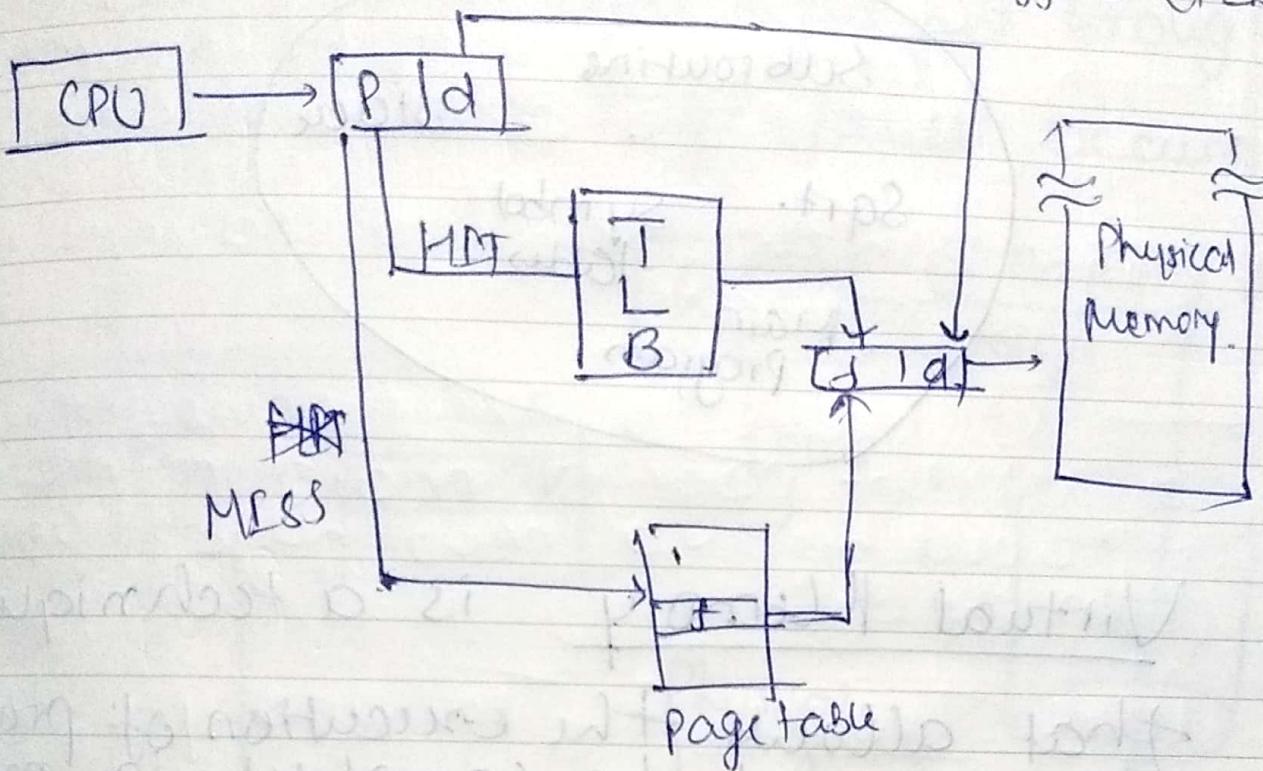
Each OS has its own method of storing page tables. It is usually stored in the memory.

- Most of the OS allocate a page table for each process.
- A pointer to the page table is stored in the process control block (PCB).

FEBRUARY 2015						
Wk	M	T	W	T	F	S
05						1
06	2	3	4	5	6	7
07	9	10	11	12	13	14
08	16	17	18	19	20	21
09	23	24	25	26	27	28

A small, special, fast, lookup hardware cache called associative register
If not us, who? If not now, when?

or translation look aside buffer (TLB).



Let the TLB hit ratio = 80%.

20 nscc is reqd. to search the associative register (TLB).

100 nscc is reqd to access memory.
Then 120 nscc is required when the page no. is in TLB.

However if it fails. (miss) then it will take 20+100+100 nscc (220 nscc).

To find the effective memory access time by probability

$$\text{Eff. mem.access time} = 0.8 \times 120 + 0.2 \times 220 = 190 \text{ nscc}$$

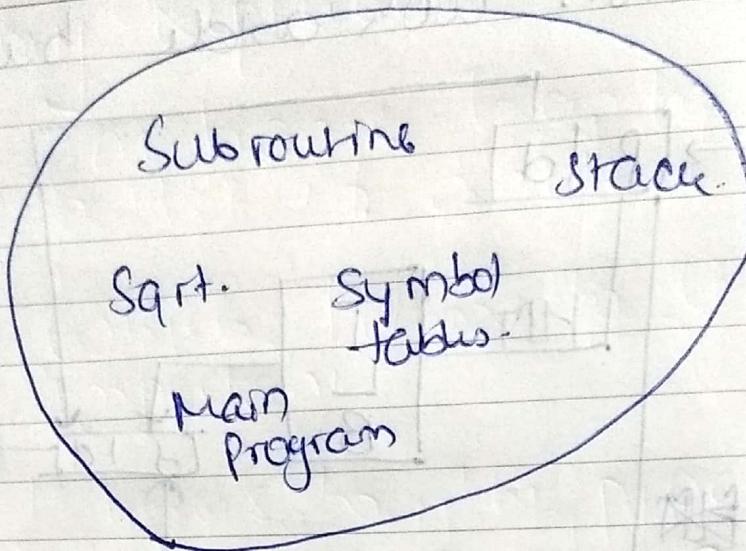
Don't let what you can't do interfere with what you can do.

MARCH 2015						
Wk	M	T	W	T	F	S
09	30	31				1
10	2	3	4	5	6	7
11	9	10	11	12	13	14
12	16	17	18	19		22
13	23	24	25	26	27	28

25

February
WEDNESDAY
Day 056-309

2015
Week-08



Virtual Memory is a technique that allows the execution of processes that may not be completely in memory.

- The programs can be larger than the physical memory.
- The physical memory is partially filled up with other processes.
- It abstracts main memory into an extremely large, uniform array of storage.
- Commonly implemented by demand paging.

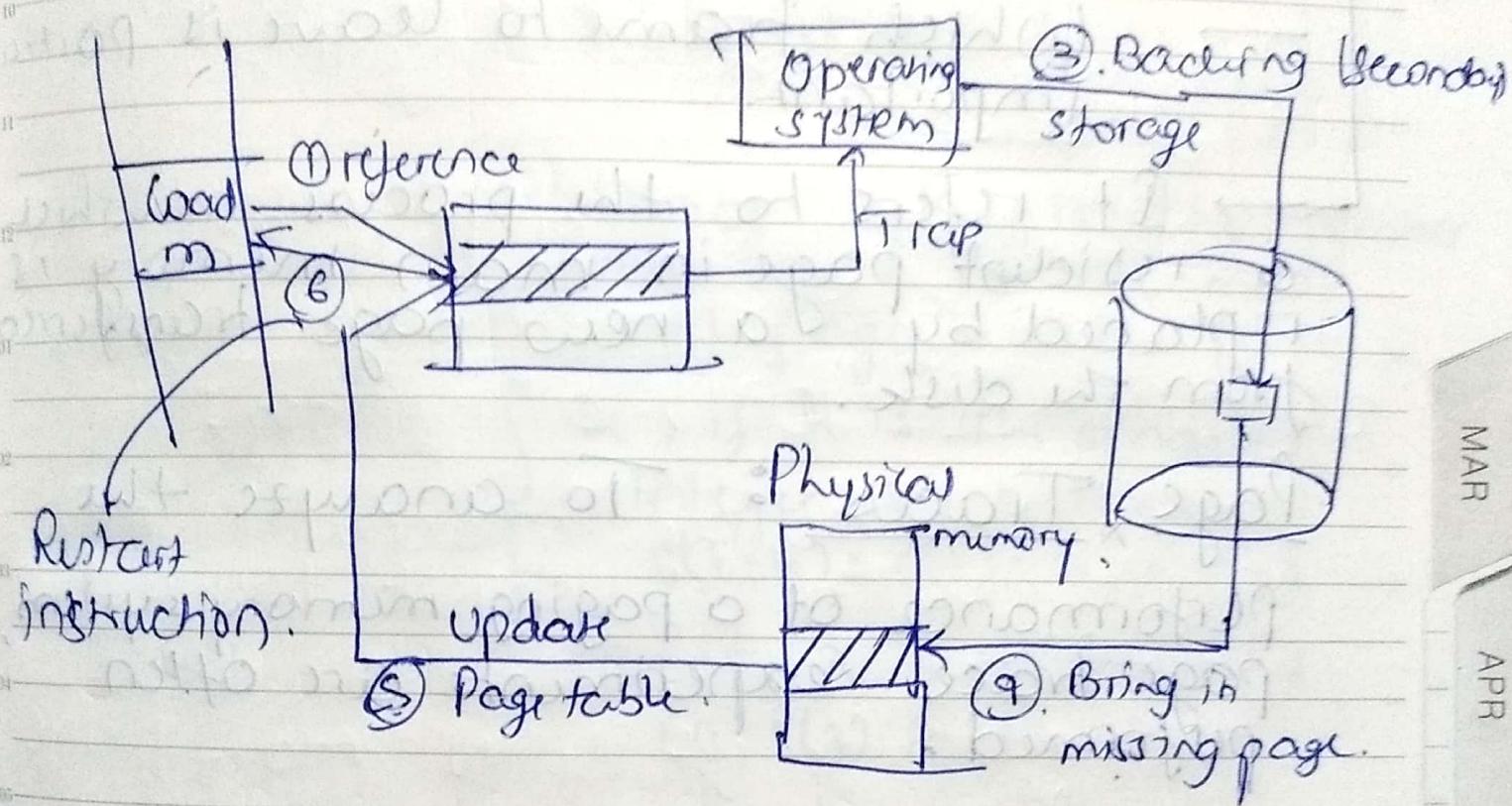
FEBRUARY 2015						
Wk	M	T	W	T	F	S
05				1		
06	2	3	4	5	6	7
07	9	10	11	12	13	14
08	16	17	18	19	20	21
09	23	24	25	26	27	28

2015
Week 09

February
THURSDAY
Day 057-308

26

What happens when the process tries to use a page that was not brought into memory? → Page fault occurs.



MARCH 2015						
Wk	M	T	W	T	F	S
09	30	31				1
10	2	3	4	5	6	7
11	9	10	11	12	13	14
12	16	17	18	19	20	21
13	23	24	25	26	27	28

If life were measured by accomplishments, most of us would die in infancy.

27

February
FRIDAY
Day 058-3072015
Week-15

Memory Replacement Policy

- One frame has to leave memory to let in another frame.
- Which frame to leave is particularly important.
- It refers to the process in which a resident page in main memory is replaced by a new page transferred from the disk.

Page Traces: To analyze the performance of a paging memory system, page trace experiments are often performed.

A page trace is a sequence of page frame numbers (PFNs) generated during the execution of a given program / process.

$$P(n) = r(1) \ r(2) \dots \ r(n), \ r(t) \text{ is the PFN request at time } t.$$

forward distance $f_t(x)$: for page x

is the number of time slots from time t to the first replicated reference of page x in the future.

High achievement always takes place in the framework of high expectation.

FEBRUARY 2015						
Wk	M	T	W	T	F	S
05					1	
06	2	3	4	5	6	7
07	9	10	11	12	13	14
08	16	17	18	19	20	21
09	23	24	25	26	27	28

2015
Week 09

February
SATURDAY
Day 059-306

28

$f_t(x) = \begin{cases} k, & \text{if } k \text{ is the smallest integer such that } r(t+k) = r(t) = x \text{ in } P(n), \\ \infty, & \text{if } x \text{ does not reappear in } P(n) \text{ beyond time } t. \end{cases}$

Backward Distance $b_f(x)$ is the number of time slots from time t to the most recent reference of page x in the past.

$b_f(x) = \begin{cases} k, & \text{if } k \text{ is the smallest integer such that } r(t-k) = r(t) = x \text{ in } P(n), \\ \infty, & \text{if } x \text{ never appeared in } P(n) \text{ in the past.} \end{cases}$

Let $R(t)$ be the resident set of all pages residing in main memory at time t . Let q_{tH} be the page to be replaced from $R(t)$ when a page fault occurs at time t .

MARCH 2015						
Wk	M	T	W	T	F	S
09	30	31				1
10	2	3	4	5	6	7
11	9	10	11	12	13	14
12	16	17	18	19	20	21
13	23	24	25	26	27	28

Page replacement policies.

1. Least recently used (LRU)

This policy replaces the page in $R(t)$ which has the longest backward distance.

$$g(M) = y \text{ iff } b_f(y) = \max_{x \in R(t)} \{ b_f(x) \}$$

2. Optimal (OPT) algorithm :

This policy replaces the page in $R(t)$ with the longest forward distance.

$$g_f(M) = y \text{ iff } f_f(y) = \max_{x \in R(t)} \{ f_f(x) \}$$

3. first in first Out (FIFO) :

This policy replaces the page in $R(t)$ which has been in memory for the longest time.

4. Least frequently used (LFU) :-

This policy replaces the page in $R(t)$ which has been least referenced in the past.

NOTES

8. Random Replacement: ~~No. of page faults~~

This is a trivial

Page trace Experiment:

Consider a paged virtual memory system with two-level hierarchy.

Main memory (M_1) and Disk memory (M_2)

Assume a page size of four words. The no. of page frames in M_1 is 3 labelled a, b & c.

The no. of pages in M_2 is 10, identified by 0, 1, 2, ..., 9. The i th page in M_2 consists of word addresses $4i$ to $4i+3$ for all $i = 0, 1, 2, \dots, 9$.

A certain program generates the following sequence of word addresses

Word trace: $\underbrace{0, 1, 2, 3}_2, \underbrace{4, 5, 6, 7}_3, \underbrace{8, 9, 10, 11, 12}_2, \underbrace{13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30}_7$

Page trace:

Word trace: $\underbrace{8, 9, 10}_2, \underbrace{11, 12, 13, 14, 15}_3, \underbrace{16, 17, 18, 19, 20, 21, 22}_7$

MAR

APR

MAY

Initially all PFs are empty

	PF.		
LRU	a b c <u>faults</u>		3/11
OPT	a b c <u>faults</u>		4/11
FFO	a b c <u>faults</u>		2/11A

01

March
SUNDAY
Day 060-305

201
Week
V

A page fault causes the following sequence to occur:-

1. Trap to the operating system
2. Save the user registers and process state
3. Determine that the interrupt was a page fault.
4. Check that the page reference was legal and determine the location of the page on the disk.
5. Issue a read from the disk to a free frame.
6. While waiting, allocate the CPU to some other user.
7. Interrupt from the disk (I/O) completed
8. Save the registers and process state for the other user (if step 6 executed)
9. Determine that the interrupt was from the disk.

MARCH 2015						
Wk	M	T	W	T	F	S
09	30	31				
10	2	3	4	5	6	7
11	8	9	10	11	12	13
12	14	15	16	17	18	19
13	20	21	22	23	24	25
	26	27	28	29		

10. Correct the page table and other tables to show that the desired page is now in memory.

If you can DREAM it, you can DO it.

2015
Week-10

March
MONDAY
Day 061-304

02

- 08 11. wait for the CPU to be allocated
09 to this process again.
- 10 12. Restore the user registers, process
11 state, and new page table, then
12 resume the interrupted instruction.

Performance of Demand Paging

Let p be the probability of a page fault ($0 \leq p \leq 1$) we expect p to be close to zero.

i.e. Only few page faults occur.

Effective access time = $(1-p) \times ma + p \times \text{page fault}$.

Where $ma \Rightarrow$ memory access time.

≈ 10 to 100 or more nano seconds.

We may consider an average page fault service time of 25ms and memory access time of 100ns .

$$\text{Effective Access time} = (1-p) \times 100 + p \times 25 \times 10^6 \text{ ns}$$

$$= 100 + 25,000,000 p \text{ ns}$$

We can see that the effective access time is directly proportional to the page fault rate.

All our dreams can come true - if we have the courage to pursue them

APRIL 2015						
Wk	M	T	W	T	F	S
15	6	7	8	9	10	11
16	13	14	15	16	17	18
17	20	21	22	23	24	25

03

March
TUESDAY
Day 062-303

2015
Week-10

If one access out of 1000 causes a page fault (i.e. $P = \frac{1}{1000}$) the effective access time is $25,000 \text{ ns} \approx 25 \mu\text{sec}$

The computer will be slowed down by a factor of 250 ($\because \frac{25000}{100}$) because of demand paging.

If we want less than 10% degradation we need $110 > 100 + 25 \times 10^6 P$.

$$10 > 25 \times 10^6 P$$

$$\frac{10}{25 \times 10^6}$$

$$4 \times 10^{-8} > P$$

∴ less than 1 memory access out of 25,000,000 to page fault is permitted. for a reasonable slowdown.

MARCH 2015						
Wk	M	T	W	T	F	S
09	30	31			1	
10	2	3	4	5	6	7
11	8	10	11	12	13	14
12	16	17	18	19	20	21
13	23	24	25	26	27	28
					29	

Go confidently in the direction of your dreams. Live the life you have imagined.