2/1/2018
Class 1

Computer.

A computer is a general-purpose machine capable of executing programs.

A program is a sequence of instructions.

→ Instruction Cycle (Not changed)
- Fetch, decode, execute
- only universal thing in Computer Science

Calculator

Not a computer
- Not Fetch, decode, execute
- perform operations but does not execute instruction.

→ A single program of computer can simulate a calculator.

SmartPhone

It is upto you, whether you see it a general purpose device or not.

CS
↓
Computer Science

is the study of computer, computing processes.

CE
↓
Computer Engineering.

There is no sharp boundary between these two.

Engineering → Application of real world problems.

IT = ICT
↓
Communication,
↳ Paytm
↳ Online Banking

Is it important for computer to be electronic device?
Need not to be an electronic device.

Computer = Hardware + Software
(Non-physical,
non-tangible components)
→ programs,
→ Source code + files,
+ device drivers, user manual.

User-manual, is meant to be read by the end user.
Documentation → Read by developers if developing diff. parts of ~~computer~~ modules)

Software is a collective noun,
(Softwares) ✗

Software
- Application software
  - ↓ Provide some specific purpose (utility to user)
  - Ex- Ms Word.
- System software
  - to create and use application software
  - Ex- OS, compiler, Assembler, Linker

Runtime Environment (part of system software)
does not provide any specific functionality

Operating System.
An operating system is a piece of system software that manages all resources of the computer, hardware or software, and provides a platform for easy and efficient execution of programs

Software resource - files, drivers

Why we use?
OS provides the easiest or reasonable way for interacting with hardware

→ How it is invented?
How it comes?

Charles Babbage → Analytical Engine
First design of computer,
(Never completed)

Ada Lovelace
She developed Bernoulli
Read about Analytical Engine,

First working computer developed in mid-1940s
Konrad Zuse → K4. (computer) [fetch, decode...]
No OS ENIAC
but there were some control programs.
OS evolved from control programs.

University of Manchester,
Late-1950s.
developed ATLAS

Class-2
2/1/2018

Computer Hardware

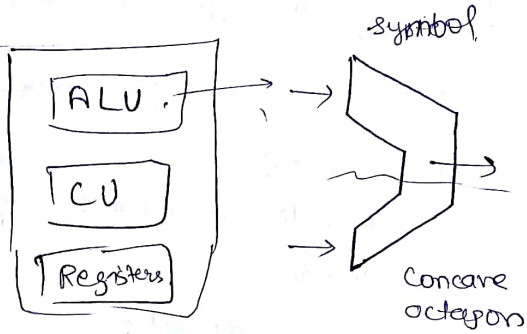
Major part → CPU

↳ Fetch, decode, execute is done by CPU
- To perform tasks

② Memory
store data, info & programs,

③ I/O device :- interact with devices,

CPU →

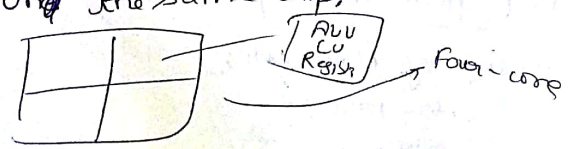


CU → two ways.
hardwired. → flip flop, register
microprogrammed → control programs,

CPU in Laptop → Processor.
Microprocessor used as CPU computers.
CPU become a single chip,
Faster, less energy, less space

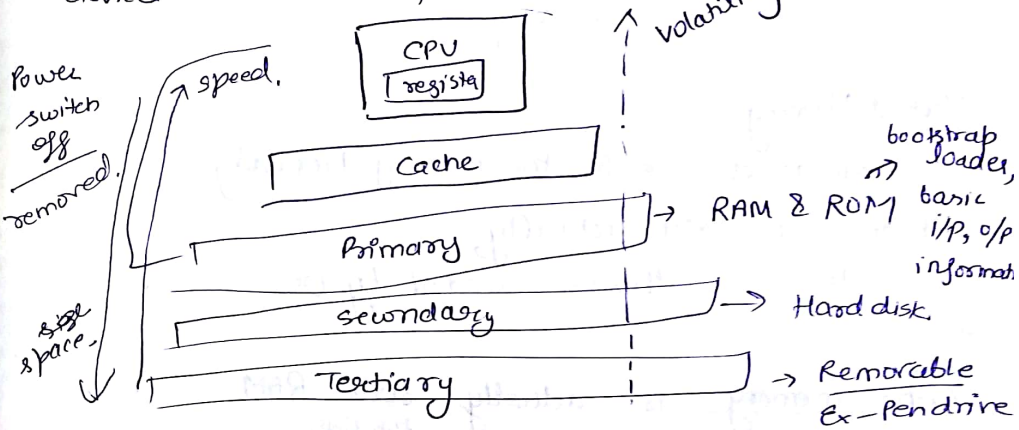
A computer that uses microprocessors as CPU is called micro-computers.

Multi-core → A processor with multiple sets of (ALUs, CUs) on the same chip,



→ Multiple core provides parallel-computing processing.
→ Ensure concurrency

Memory hierarchy.
↳ arrangement of different types of memory device in the computer.



ROM
0th address
↓
start bootstrap loader

Random access can also be perform in ROM

RAM
Programs currently executing and data associated with it

RAMdom access
↓
Any location of RAM can be accessed at any time.

When computer swiched off,
OS resides in Secondary storage (Hard disk)

When On, it execute bootstrap loader in ROM first address.
OS load into RAM.
till that time computer is On, major part of OS resides into RAM.

Cost per byte →

Register > Cache . . . . . > Tertiary

what is there at level i is a subset of level i+1.

Most persistent level ↓

tertiary

Virtual Memory

You cannot see in the memory hierarchy

It does not exist actually,

It is an illusion created by OS.

Cache memory is actually static RAM.
flip flops.

dynamic RAM capacitors used.

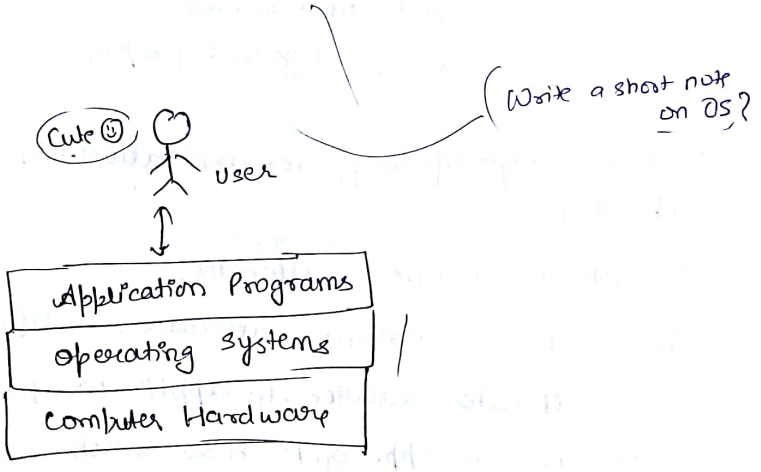
(more capacity, more bits per unit area)

What Os do in your computer? or your phones?

class-3
4/1/2018

↳ Os manages all the hardware
↳ os facilitates all the execution of application Programs
↳ Os act as an intermediate b/w user and compute hardware
↳ Os has to be design to convenient to use and efficient.

Block diagram



(Write a short note on OS?)

→ Does user directly interact with OS?
Yes, but it is quite limited, (like shut down, -etc)

OS ensures that all the hardware and software are used properly.

OS Design Goals :-

what should be our goals ?

1. Convenientee (ease of use). → personal computers.
2. Efficiency. (manage the resources in best possible way) → Super computers.
   Multitasking is one way of efficiency
3. Energy conservation (For mobile specifically) → Phones
4. Autonomous (It should have minimal user interface → Embedded Systems. (Ask in microprocessor class) ☺).
   Minimal user interface

Convenience and autonomous are used for different purposes.

To achieve high efficiency, resource allocation is important.

OS acts as resource allocator.

Q) 1) OS acts as resource allocator. Justify.

OS allocates resource to application programs.

Ex- which app. gets how much resource, for how much time is decided by OS.

Ex - Browser, Ms word, Paint allocate resources

Q) 2) OS acts as a control Program.

OS evolved from control programs, send signals to h/w & s/w. It controls all h/w & s/w, when which h/w & s/w should work.

First OS :- ATLAS.

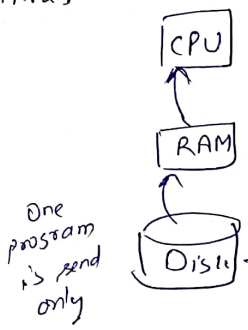
Our OS Name → KAAN

Types of Operating System:-

(1) Single process OS. [Eg. – MS DOS] (1983)

↳ Process is a program execution.

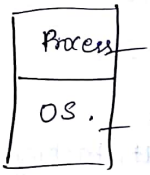
↳ Only one process is loaded into RAM at one time. (resource utilization is not proper)



One program is read only

Disk may contain any no. of programs.

- They are inefficient.



1 process at a time

till we donot switch off comp.

(2) Batch Processing OS :- (Eg- ATLAS)

Earliest OS developed (1957).

Inputs given using punch cards, primitive input devices.

Many single process clubbed together to form a batch. and computer executes the program one after other in a batch and output provided in a batch using primitive printer.

→ Program in a batch were of same type (all arithmetic)

→ At one time, one program was executed.

+ High inefficient and difficult to use

(3) Multiprogramming OS :-

With the advancement of technology, micro computer were made.)

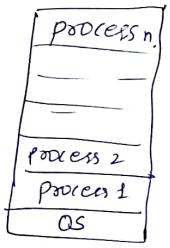
Load a few processes in memory simultaneously

→ Initially one process is executed, when that process halts, another process is executed in mean time.

→ CPU not sitting idle

→ Computer became efficient

→ CPU always had one program to execute



Ex - THE

Technique Hang en Enaur → in 1960

first multiprogramming OS.

(4) Multi-tasking OS.

Multi-programming and multi tasking are different.

Multi-tasking is enhancement of multiprogramming.

→ Logical execution of multiprogramming OS.

→ Multiple processes load in memory

→ CPU executes little bit of one process then little bit of another proces. - - -

and keep executing processes in some order.

switching b/w different processes. (in microseconds) is very fast, which make illusion that every program runs concurrently, simultaneously

To have multitasking OS, we need to have hands on computer,

where user can directly interact with computer (like our computer with keyboard/mouse)

(In early 1960s., computers are not touched, there were punch cards) so these computers can not have multitasking.

Kernel

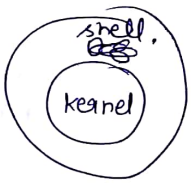
is a part of OS, that interacts with the hardware directly. Not whole OS is interact with h/w. Most imp. activity of OS perform by kernel.

→ Kernel is saved in RAM,
→ From disk till the time comp is shut down
→ stays in RAM, till comp. is switched on
→ Most important part of OS.
→ kernel is small; so that it occupies less space in RAM, so that RAM contains more programs.
→ If kernel is large, system become less secure (have more keys due to large size)
→ Most imp. activities are implemented within the kernel and less imp. activities outside kernel. (here kernel can be small)
→ When a kernel is very small so that it is minimal in size and provide base min facilities are called Micro kernal.

Very small in size, but only 1 microkernal in any system.

Shell = Command Line Interpreter,
(MS DOS had it)

It is that part of computer which directly interacts with user. User directly provides commands to shell, then shell gets command executed using other programs of OS.



Books:-

Operating System (Concepts/Principles)
By Silberschetz, Galvin, Gagnt.

8/1/2018
Class-4

Kernel

* Kernel resides in RAM for entire duration.
* Other utilies are implemented as seperate ~~not~~ modules outside the kernel.

Your program cannot interact with hardware; kernel can. How?

Dual mode operation of computer system.

(1) User mode
(everything outside, kernel works, shell of OS works in user mode).

(2) ~~Kernel~~ kernel / supervisor / system / priveledged mode.

Anything that runs in user mode, does not have the permission to interact with hardware.

* Kernel works in kernel mode.
* Any part of code executing in kernel mode can interact with the hardware.

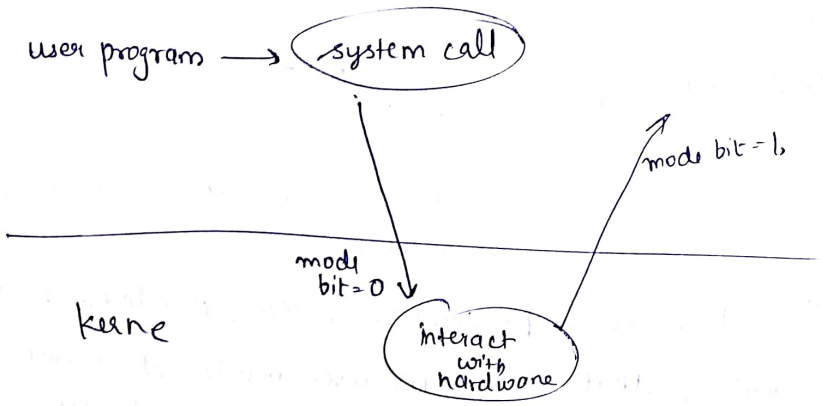
→ How will computer know in which mode it is working?

mode bit ⇒ stores bit that indicates in which mode we are working.
Values 0 or 1.
0 ⇒ kernel mode
1 ⇒ user mode

System call → is a request, always invokes from user mode to execute some part of code in kernel mode.

→ Requesting kernel to do some work which program is not permitted to do by itself.

→ If kernel has to do something, you need to switch to the kernel mode.



→ Some instructions in microprocess can be executed only in kernel mode

Privilege instruction if to be executed, you have to enter kernel mode

→ Why can't you execute whole user program in kernel mode?

→ Massive Security Problem.

↳ hardware can be damaged either genuine mistake or malicious,

tasks to kernel have to be invoked by system calls,

system calls → execution of prog.
→ allocation of mem. in RAM
→ in disk/sec storage

System calls are implements as C functions

System calls
- C functions
- fork, getpid, getppid, wait, exec
  related to execution of program.

(PROCESS CREATION later.)

What is dual mode operation?

2 or 3 marks
What is system call.

Duties of an operating system

① Process Management.
- Process is any program that is being executed. Only then process.
- includes every activity related to process
- starts with creation of process.
  - creation of user process
  - creation of system process, (generated by OS itself)

deleting of process also a duty of OS.

suspension of process.
Resumption of process } also a duty of OS.

* Interprocess communication duty of OS.
* synchronise of process.

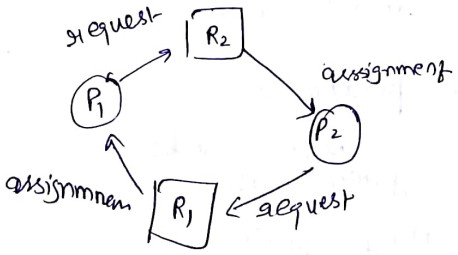
One program may have wait to other program.

OS ~~help~~.

Duty of OS help in process synchronization.

* In OS strange things happen.

Multiple process try to jump on one resource



P_1 needs R_2

P_2 needs R_1

→ deadlock condition.

It is again duty of OS to handle such situation.

All these comprises process Management

till midsem → we are discussing only Process Management)

3, or 4 chapters

* Concept of process
  - ↳ Threads
  - ↳ Scheduling Process
  - ↳ Process synchronisation
  - ↳ Deadlock

(2) Memory Management

RAM only.

OS & one & more processes in RAM, which computer on,

* Program & related data
* free/spare

It is duty of OS to keep track of free spaces in RAM.

If OS doesn't know where space, it cannot load programs.

↳ OS keeps track of which part of memory contains which program.

Memory management → 2 chapters.
After midsem.

#(3) Storage Management

Secondary memory.
or
hard disk.

↳ It is the job of OS to manage hard disk.

↳ logically stored files

file system → logical concept.

It is the duty of OS to keep track of ~~dogi~~ file system.

files → blocks → sectors → cylinders → plates/disk } rotate at high speed

schedule access to diff. parts of disks.

Storage Management → file system mgt.
→ mass storage mgt.

Later study.

(4) Caching

(5) Input/Output management.

It is the duty of OS to ensure that the user can use input/output devices properly.

Operating System Services :-

two parts

provide a large number of services.

(1) helpful to the users

(a) execution of our program

(b) an user interface

- CUI — character or command (also known as the shell)
- GUI — Graphical (one of the biggest success in computers)

GUI was popularised in 1980s in USA and Europe.

By 1990s to India.

(c) I/O operation.

(d) OS helps in file system manipulation.

(e) helps in interprocess communication

(f) Error detection (bad. alloc)

(2) helpful to the system

do not help the user directly but help the system to work properly.

(a) Resource allocation.

(b) facilitates protection and security.

(Tries to ensure diff. resource of computer are being user properly.

Operating System structures. —

- How the whole source code of OS is organised in OS.

Windows code is billions of lines.

OS are large programs.

- How to manage different modules.
- How different modules are arranged.

(1) Monolithic (one stone).

(Entire source code is jumbled into one module)

Any part of OS can interact with any part of OS or hardware,

(Everything at same level.

Ex- MSDOS, Unix, Linux

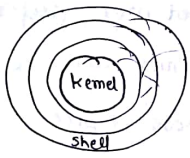
(a) Firstly it is insecure

(b) If we would have to change some code, it would be difficult.

(2) Layered architecture

E.g. THE

Any program at i^{th} layer can access $i-1^{th}$ layer



system programs.

Only kernel interacts with hardware,

* Improves software engineering aspect of OS
* Improves security aspect of OS.

(3) Micro-kernel based architecture:-

layered microkernel

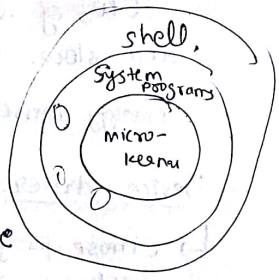
(a) Here kernel becomes minimal.

Hence it is highly secure

Outside — shell

System programs implemented as module

Eg- Mach, MINIX. (Education purpose OS)



Real Time Operating System. (RTOS)

9/1/2018
Class-5, 6

↳ special purpose OS
for specialized applications. Ex- Robots, moon rovers,

→ rigid time constraints
(if not follow whole system fail).
are generally on execution of program.

→ two types
↳ Hard RTOS
↳ Soft RTOS

Hard → It is absolutely necessary to follow time constraints.

Soft → it is expected that time constraint will be met.
If not met, performance will degrade.
Ex- Linux can be considered as Soft RTOS.

Q) SHORT note on these topics :

Operating Systems for smartphones –

When CPU for smartphones are designed, they are slow.

CPUs of smart phones are intentionally kept simple and slow. So that - heat does not generate.
Energy conservation.

Device drivers. (Software)

↳ those programs that allow us to interact with different I/O devices,

Q. Is ~~the~~ device driver is a part of OS?

Controversial → Installation from outside not always needed

→ once you install, it become part of OS.

→ Read in detail in chapter I/O management.

Case studies :- On OS

① CP/M

* Control program or monitor,
* used in late 70s, nearly 80s.
* made for 8 bit µpr.
* Gary kindlal,
* ~~sold it so much~~
* write about architecture, Read it

② MS DOS

Disk OS

Read about Architecture,

③ UNIX

④ Linux

⑤ Windows

Turing award is most prestigious award in computer science

Crypto, theoretical CS, AI.

ACM

Association of Computer Management.

gives award for Lifetime achievement award,

3 people got this award

→ KL Thompson

→ FP Brooks Jr,

wrote a book

Mythical Man Months.

Unit of effort. ← 1 man month = working on a particular job for one month, is a unit like km, m

1 man year. = Can be two man working in some project for 6 months,

→ Barbara Liskov,

(Professor in MIT)

* (Open Ended Project)

develop a very simpler minimal OS.

PC → easier

Smartphones → difficult

Unit-2

Process and Process Scheduling.

A process is a program that is in execution,

On OS,

process ≈ job

used synonymously,

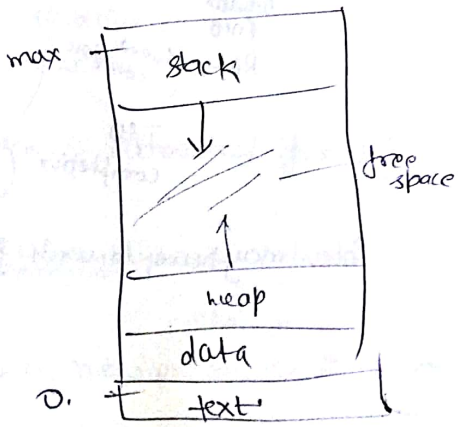
process has several components.

- Text Section (program) in ML,
- current activity represented by values of the registers,
- program stack,
  - (subroutines & function calls, whenever there is fn call, return address is saved,
  - used to evaluate multioperator expression,
- data section
  stored the global variable in this,
- heap
  to store dynamically allocated variables.

Process is a bigger entity than a program,

memory allocated to a program

Logical address. base of a process,



→ Size of stack always varies.

→ stack grows downward, (here)

→ heap grows upward

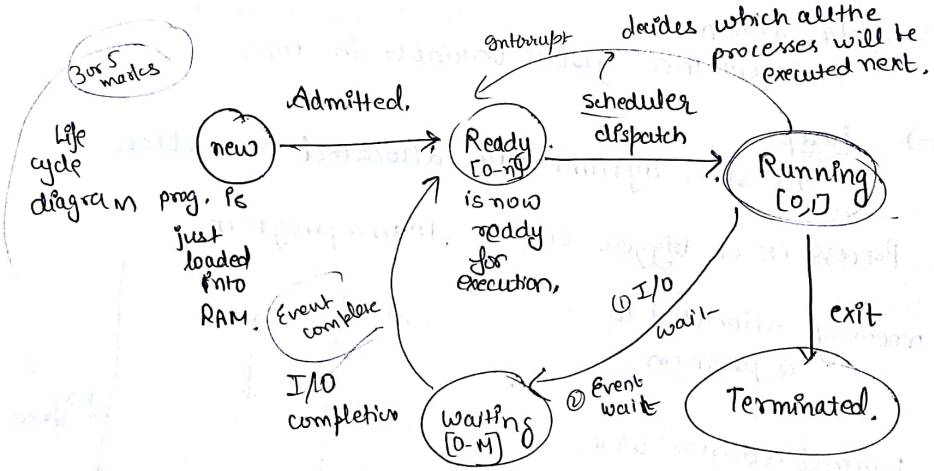
They should not collide;
If collide, unpredictable things occur.
↳ there are different techniques to check this,

A program is a passive entity.
A process is an active entity

Process state

→ tells what the process is actually doing right now.

→ state of a process changes quite a few time during its lifetime



↳ Process may have to wait for an event to occur

Interrupts

- the current instruction is always complete, in case of interrupt.
- every process starts from new stage

OS uses

Process Control Block (PCB) ⓒ.
- It is data structure
- stores all the imp facts.
- for each process OS has a PCB
- whenever process is in new state, its PCB is created.
- In RAM, there is PCB
- PCB is created by OS, OS has its own area & some free area.

what all stored in PCB

- process state
- values of the register for a process is stored
- CPU scheduling information
  - OS assigns priority
- accounting information,
  - is there any deadline
  - is there any timeline
- I/O status
  - which I/O device is allocated for this process.
- process id
  - OS allocates unique ~~proces~~ pid to all the processes
  - whole number

& Everytime program is executed, a new process id is generated.

Same program can create multiple process and can create different ids for the same program.

Process id of kernel is zero (in Linux) of type

Page - 6

(Diagram)

11/1/2018
Class 7

Process Scheduling.

OS is resource allocated.

Most important resource : CPU.

Duty : To use that resource to max,

→ done by OS
↳ selects one process, to be next executed,

Multiprogramming
↳ objective
to use maximum use of CPU.

Objective of multitasking :-
Switch CPU b/w resources.

→ To implement process scheduling, we have to use data structure,
scheduling queues. → itself is a data structure.
stored PCB as elements.
(PCBs of processes).

Types of scheduling queues.
↳ Job queue
(there is only one job queue in the system)
from birth to death,
↳ ready queue
(also one ready queue)
store PCBs of all such process which are in ready state.
(loaded into RAM)

All these queues are implemented as linked lists.

Q) short note on scheduling queues.

You should draw a diagram as were in the notes.

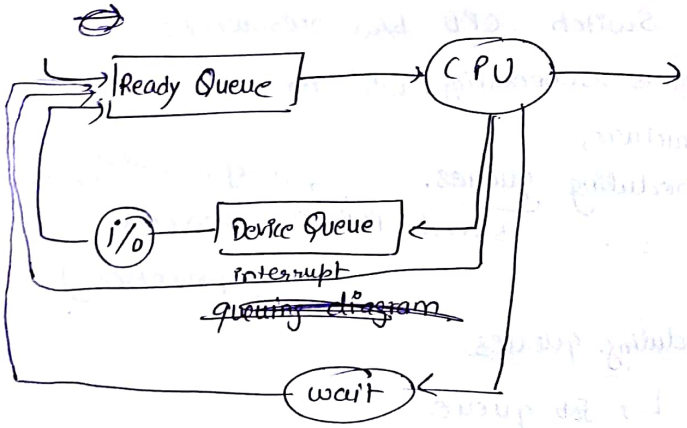
→ Device Queue

two or more process may be use same I/O devices.

most I/O devices are not shareable, so there is device queue for them.

Q) How many device queue?

Each device will have its own device queue.



Queuing diagram

→ Which will be given prority of I/O?

As in queue,

FIFO.

Priority Queue → Then queue will be manipulated)

→ Who will take this decision?

Scheduler.

- is module of OS,
- decides which process to be executed next.

3 types

① long-term scheduler ≡ job scheduler.
(decides which of the process to be loaded into memory at what time.

→ controls the degree of multiprogramming
no. of processes in RAM.

Why long-term ?

Long term scheduler invoke infrequently.
(only after a long time)
(it use quite less)

Two types of Process → CPU-bound (more calculations)
use CPU for maximum time for life time.

→ I/O-bound (Print, MS-Word, Music Player)

- Long term scheduler loads a proper mix of CPU-bound and I/O-bound into the memory
ratio is maintained
duty of long-term ~~pro~~ scheduler.

② Short-term scheduler
↳ it will invoke quite frequently.
↳ very fast.

→ decides which of the process currently in the memory will be executed next (given control of CPU).

→ they are softwares

→ also known as CPU scheduler

activity known as CPU scheduling.

↳ has to be very fast.

→ Most of the OS ex - Ubantu, Mac, Windows used only short term scheduler

As in computers, support to be used by one user

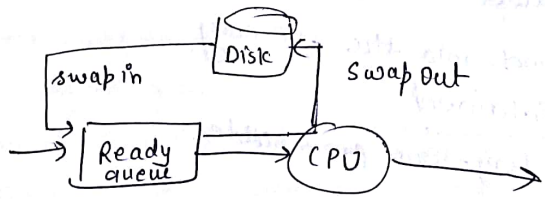
use will be automatically do long-term scheduler

(3) Medium term scheduler.

↳ is invoke less frequently than short term and more frequently than long term

↳ also known as swapper,

↳ if long term realizes that program is using a lot of memory then medium term will take such program from RAM to disk.



∴ That process is assigned

if X is executed half, swapped out, then resumed from where it left once (swapped in.)

Modern day OS also don't use Medium term.

MTS → improves CPU & I/O bound processes ratio.

Context switching
when,

① If one process is terminated.
② if current process is sent to wait state

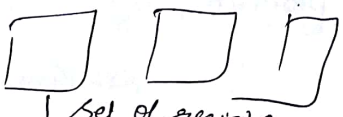
what to be done next?

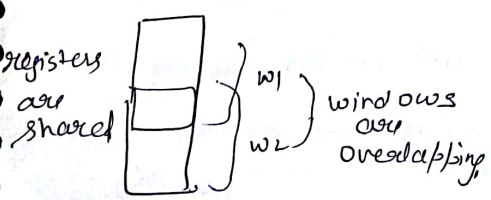
① When we have to switch, state of current process should be stored to PCB & state of next should be loaded / reloaded, ~~pure~~ ~~overhead~~.

→ context switching is pure overhead:
↳ happens very frequently
we should minimize the time of context switching.

Reduced Instruction Set Computer (RISC)

Used concept of overlapped register windows.
(Read about) it
(from Morris Mano)
2-3 page.





CWP

Current Window Pointer.

→ Overlapped will be necessary in case of interprocess Communication

↳ assign each process one register window. Only one window can be accessed at a time.

→ There are certain registers common to 2 window & some are for a specific windows.

→ There is a register called CWP who value changes when window is to be changed.

15/1/18
class 8

Process Creation.

Process
- user processes (doing something) (for user directly)
- system processes.
  - are parts of the operating system.

→ Connecting program to a process is process creation.

→ Who can create a process?
- One process can create another process.
  - parent of those other processes → children processes

→ Kernel is the first process to be loaded into system.

→ Kernel is the ancestor of all other processes.

→ Kernel creates more processes using shell, shell create more processes and so on.

→ If we draw a tree, root node is represented by kernel.

→ Once a process is created, it requires resource in h/w, s/w, CPU, memory.
- file database

who will give these resources?

→ A parent process can give resources to children

→ A process can directly request resource from OS.

It depends on OS to determine:

(1) Both child & parent will execute concurrently and will compete with each other for control of CPU.

(2) If the parent process is waiting for children to return value, then child process executed first.

The child process can be a clone of parent process or it may have a new program loaded.
It has same code and same data set.

Which system call is used to create a new process?

Ans- fork. → what does it return?

```
int main()
{
  int value;
  value = fork();
  if (value)
    printf("Parent");
  else
    printf("child");
  return 0;
}
```

If we want to parent process wait, wait();

It is used to create two processes.

→ fork will return to zero to the child process

→ return process ID of child it is creating to parent process.

This program is used in 2 processes, for parent process, non-zero value returned,

Output :- Parent or (Child
Child. Parent-wait)

Q How will process know its own id?

```
int main()
{
    printf("%d", getpid());
    printf("%d", getppid());
    return 0;
}
```

child process may not be done of parent.

```
int main()
{
    int value;
    value = fork();
    if (!value)
        exec("c:/calculator.exe");
    return 0;
}
```

Once a process has been executed, till the last instruction, it goes to terminated state.

Process termination

Deletion of PCB is needed from memory.
All resources allocated have to returned.
All files needed, needed to closed.

Sometimes, a parent process may also terminate a child process.

9) may have to kill,

- parent process may require to kill its own child process kill()

Reasons ?

-> it is taking too many time,

-> Parent asked child to do some calculation, that calculation not required any more than parent. kill child.

→ A process whose parent process is terminated is called orphan process.

→ Some OS do not allow orphan process to execute. or some allow ?

→ Some OS require parent process to be alive for orphan to execute otherwise tree of process degenerate into forest)

→ If parent process is terminating, it has to kill its children process and can lead to grand children, being terminated. This is cascading termination of processes.

Zombie process

A process which is terminated but its PCB is still in the system. because that process is supposed to return a value to its parent process.

Interprocess Communication

↳ Two process may require to communicate with each other.

→ Information Sharing,

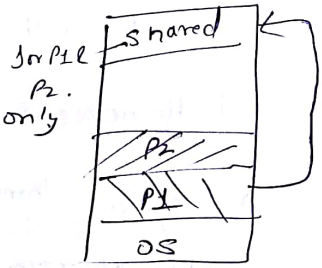
→ Communication speed up

→ same software tools as multiple processes (↑ modularity)

2 ways → ① shared memory

(multiple processes loaded into RAM)

P1 can write in shared memory which can read by P2.



② Message passing.

(No shared memory)

→ Processes can send messages to each other.

→ Process can send data using process format.

2 types of system calls (pseudo calls, abstract system call that has to be implemented)

① send (P, message);

P → an integer (id of receipt process)

message → message to be send

② receive (id, message);

receive message, send to a particular process

id → ID of process which has send the process

Threads

Earlier,
one process has sequence of instruction,
executed one after other,

→ Now,
process contains one or more thread.

→ Each thread is a sequence of instructions.

↓

→ Multiple threads can be executed - parallel and scheduled independly, this program paradigm, called multi-threaded programming,

→ A threads is also known as light weight process.

→ Different threads of same process shared the same ~~resources~~, memory, files, resources.

→ A thread has its own state

→ thread can be synchronized

→ thread create more threads.

→ process have its own state of memory, resources so, it is different from thread

→ Thread can wait for other threads

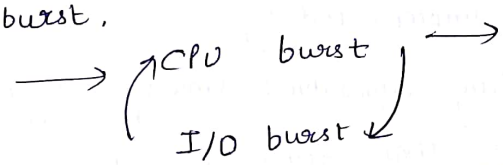
→ Context switching become more fast.

what note

16/1/2018
Class-9-10

When a process starts execu, it goes to the CPU & performs calculations for some time.

* When process starts perform CPU burst.
* After that process, does some I/O for some time I/O burst.



Execution of process is a sequence of alternate CPU burst & I/O burst.

CPU Burst → within a short duration several CPU processes are being executed.

When will ~~will~~ CPU scheduler invoke?

Under four particular conditions,

→ C1 : when a process goes from running → waiting.
CPU scheduler invoke

→ C2 : running ⟶ (low priority) ready,
~~so~~ that high priority execute.
then, CPU scheduler invoke,

→ C3 : waiting → ready
If current process does CPU burst, then I/O burst - CPU control is relinquished, so low priority is executed

→ C4 : running → terminated
running process is terminated.

In case of C1 and C4, CPU scheduler compulsory

C2 and C3 are optional

Interrupt ⇒ C2

→ On what basis, these CPU scheduler will work? that logic is implemented by :-

CPU scheduling algorithms. (must be very fast) (simple as possible)

↳ takes logic that select one of the many ready processes.

Non-preemptive
Eg- Window 3.1x
(C1 & C4).

Preemptive
Ex- Linux, Unix, Window 95 and higher
(C1- C4),

Non-preemptive

Once a process gets CPU, is allowed to execute as long as it wants.
i.e. till termination or need I/O

No other process of OS can evict it out of the processor (CPU)

→ also known as cooperative scheduling

→ it is simple to implement

→ they are not efficient.

Preemptive

A running process can be evicted out of CPU → priority based
or
another logic based

→ requires extra hardware

In case of C2 and C3, CPU scheduler may or may not be invoked.

MS-DOS
↳ single process OS
so there is no concept of CPU scheduling

Dispatcher.
↳ Model in OS called dispatcher
↳ part of software
↳ will give the control of the CPU to the chosen process (chosen / selected) by CPU scheduler.

Steps used by dispatcher
① switch context.
② switching to user mode
(change the value of mode bit)
③ jump to the proper location in the user program. (instruction that has to be executed next.)

→ Time taken by the dispatcher to do all the functions (3 steps) is called dispatch Latency.

termination or suspension of process to start or resuming another process

→ Hardware dependant
→ pure overhead

→ value is constant, must be minimum

→ no useful work can be done in it.

There are some parameters, according to which, we compare different cpu schedulers called

Scheduling criteria

① CPU utilization (should be highest)

For what % of time, CPU is busy (80-90%)

② throughput (as high as possible)

no of processes completed / unit time

③ Turnaround time (should be less)

time taken from its submission to its termination,

↳ time spend in CPU + I/O + ready queue + devices.

① and ② is calculated for the entire system.

③ can be calculated for individual processes.

Average turnaround time for all the processes

④ Waiting time (minimize)

time spend by the process in the ready queue (waiting only for CPU) only (not for I/O.)

can be calculated for individual process

if for whole system, take average

⑤ Response time (minimum)

time spent from the submission of the process to the arrival of first response.

↳ can be for individual

↳ take average for system.

min. variance

variance of response time of all processes should be minimum.

Gantt Chart

↳ type of notation used when study CPU scheduling also.

↳ used in management.

* CPU schedulings numerical (come) in exam.

CPU scheduling Algorithms.

① First-Come First Service (FCFS). Algorithm.

↳ Non preemptive algorithm.

↳ fair

↳ simple to implement.

↳ performance will not be good.

~~throughput &~~ waiting time & turnaround time will be high.

Convoy effect occur

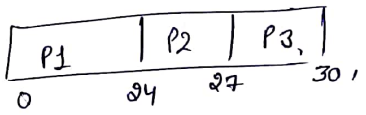
↳ stuck up in ready queue for a long time.

Q7)

Process	Arrived time (ms)	CPU burst time (ms)
P1	0	24
P2	1	3
P3	2	3

Ready queue : ~~P1~~ ~~P2~~, P3.

Gantt chart :



draw gantt chart in exam

Calculate different parameters:-

(1) throughput

$3/30 = 0.1$ process/ms.

(2) turn around time

P1 = 24-0 ms.

P2 = 27 - 1 = 26 ms

P3 = 30 - 2 = 28 ms.

Average = $\frac{24+26+28}{3} = 26$ ms.

Show calculations

(3) waiting time

	when it get control	−	when it arrive	
P1 :	0	−	0	= 0
P2 :	24	−	1	= 23 ms
P3 :	27	−	2	= 25 ms

Average = $\frac{23+25}{3} = \frac{48}{3} = 16$ ms.

CPU utilization ✗

if we calculate, it will become 100%. as we did not consider dispatcher,

Response time ✗

it is difficult to tell when the first response is coming?

assumption = 3 cpu.

Class-11
18/1/18
Not Attended

CPU scheduling Algorithms.

(2) Shortest Job First (SJF) Algorithm.

Process with smallest ~~burst~~ next CPU burst is selected for execution. (in ready queue).

Two or more process with same next CPU burst, FCFS breaks Tie.

⇒ Shortest-next CPU burst scheduling

Optimal ~~bursts~~ performance: avg. waiting/turnaround/response time will be minimum.

→ Best possible CPU scheduling also.

How do you find next CPU burst?

theoretical pratical

" cannot be implemented.

Impossible to predict next CPU burst without executing the program.

We have 24 ms, 3 ms type in yesterday, tues but that wasn't affecting the logic of scheduling.

→ we can try to predict next CPU burst time.

Concept of Exponential avg.

Predict next CPU burst time of particular process!

τ_{n+1} = trying to predict time of (n+1) CPU bursts.

'n' CPU burst have already occured.

Predicted time → $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$ ← Predicted value

t_n ↙ actual time taken in n^{th} CPU burst.

$0 \le \alpha \le 1$

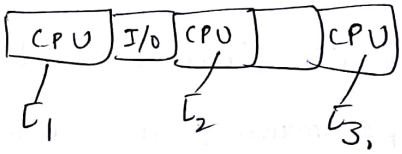
↓ constant

base case

For recursion

τ_0 = constant (use any number) or system avg

↙ Predict this value by average of τ of all system processes.



If $\alpha \to 0$, $\tau_{n+1} \approx \tau_n$,

Predicted time will be constant.

if $\alpha \to 1$, $\tau_{n+1} \approx t_n$

↗ Last CPU burst time

Recommended, $\alpha = 0.5$

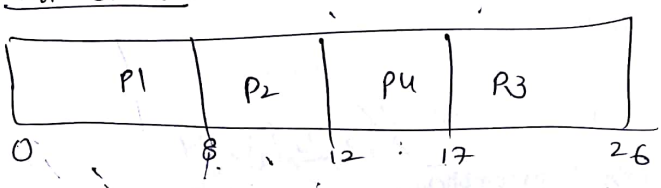
Q→ ∴ 4 processes.

	Arrival time (ms)	CPU burst time (ms)
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Non - preemptive algorithm is used,

Calculate throughput, avg. turn around time, avg. waiting time.

Gantt Chart



ready queue : ~~P1~~, P2, P3, ~~P4~~

continues till 8 ms.

throughput : $\frac{4}{26} = 0.15$ proc/ms.

turn around

P1 = 8 − 0 = 8

P2 = 12 − 1 = 11

P3 = 26 − 2 = 24

P4 = 17 − 3 = 14

} ms.

Avg = $\frac{8 + 11 + 24 + 14}{4} = 14.25$

waiting time

P1 = 0 − 0 = 0

P2 = 8 − 1 = 7

P3 = 17 − 2 = 15

P4 = 12 − 3 = 9

} ms.

avg. waiting time $= \frac{31}{4} = 7.75$ ms.

→ Preemptive versions shorted FCA is available.

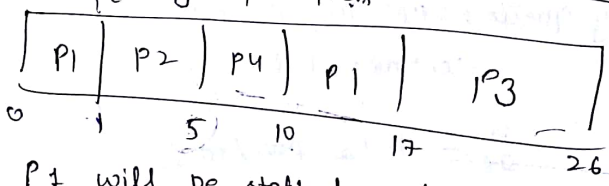
→ If next CPU burst time of new process is less than current processes, then new process can preempt other processes.

⇒ also called shortest-remaining time-first algorithm.

Q: Recalculate:

Ready queue: ~~P1~~, ~~P2~~, P1, P3, P4

↙ only 1 preemption



P1 will be stopped and P2 starts as 7 > 4 (8-1)

Throughput ∝ same

≈ 0.15 processes/ms.

Restarting time least for P4

short process gaining a lot and long process losing.

Turnaround

P1 = 17 - 0 = 17

P2 = 5 - 1 = 4

P3 = 26 - 2 = 24

P4 = 10 - 3 = 7

Avg = $\frac{52}{4} = 13$

⇒ Turnaround time reducing.

If not mentioned assume DL = 0

Large process, waiting process increases
Short process, waiting process decreases

Waiting process:

P1 : (0-0) + (10-1) = 9
P2 : (1-1) = 0
P3 : 17 - 2 = 15 ms
P4 : 5 - 3 = 2 ms

Avg = 6.5

	Throughput	Avg. turnaround time	Waiting time
FSFS.	0.15	15.25	8.75
SJF (Non preemptive)	0.15	14.25	7.75
SJF (Preemptive)	0.15	13.00	6.50

DL = 1ms (considering dispatch latency, CPU utilisation reduces)

NP	0.13	16.00	9.5
P	0.13	16.00	9.50

These values become worse in general

Preemptive → more context switching
more latency → less CPU utilization.

If DL = 0 ms.

If using non-preemptive algo, & giving all process together,

Response time ≈ waiting time

Else if using preemptive algo,

Response time may be diff waiting time.

Class 12
22/11/18

(3) Priority Scheduling

Each process is assigned a priority and it should be whole number.

↳ whether system or user process

→ higher priority process will be executed first.

→ In many OS, priority can be 0–7
and in some OS, 0–1023,

→ In some OS, 0 is highest priority and in some 7 is highest priority.

All depends on OS developer.

Factors on which priority depends,

1.) Internal / technical factors.

2.) External / organizational factors.
→ It depends on whose process it is, whether student, faculty, admin.
→ no depends on ~~bo~~computers
→ non-technical factors,

Internal factors → Ex: Time limit, memory requirement, no. of files it requires, no. of I/O devices it requires.

Priority is assigned when PCB is created and it is stored in PCB.

Ready queue → priority queue
homes

→ priority queue can be both preemptive and non-preemptive

→ SJF is a special case of priority scheduling.

SJF priority $\propto \frac{1}{\text{next CPU burst}}$.

- Starvation
  Indefinite wait for processor.

Method to solve

① Aging
→ Gradually increase the priority of process waiting for a long time.
→ Set limit to increase in priority (say upto 255).

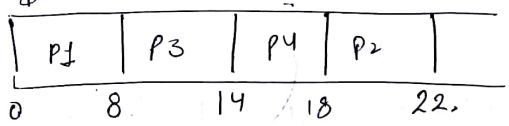
② Priority Inversion
~~priority of high~~
Assign the priority of high to low.

Process	Arrival Time (ms)	CPU burst time (ms)	Priority
P1	0	8	7
P2	2	4	5
P3	4	6	0 (max)
P4	6	4	1

① Non-Preemptive priority scheduling.

Ready Queue ~~P1, P2, P3, P4~~.

Gantt Chart



a) Throughput = $\frac{4}{22}$ = 0.18 proc/ms.

b) Turnaround time

P1 = 8 − 0
= 8 ms.

P2 = 22 − 2 = 20 ms

P3 = 14 − 4 = 10 ms

P4 = 18 − 6 = 12 ms.

Avg = $\frac{8+20+10+12}{4}$ = $\frac{50}{4}$ = 12.5 m/s.

c) Waiting Time

P1 = 0 − 0 ms.

P2 = 18 − 2 = 16 ms.

P3 = 8 − 4 = 4 ms.

P4 = 14 − 6 = 8 ms.

Avg = $\frac{28}{4}$ = 7 ms.

(2) Preemptive Priority Scheduling

Ready Queue : ~~P1, P2, P1, P3, P2, P4~~

Gantt Chart :

P1	P2	P3	P4	P2	P1
0	2	4	10	14	16	22

(a) Throughput

$= \frac{4}{22} = 0.18$ proc/ms

(b) Turnaround time.

P1 : 22 − 0 = 22 ms,

P2 : 16 − 2 = 14 ms

P3 : 10 − 4 = 6 ms,

P4 : 14 − 6 = 8 ms,

Avg = $\frac{22+14+6+8}{4}$ = 12.5 m/s,

(c) Waiting Time

P1 : (0−0) + (16−2) = 14 ms.

P2 : (2−2) + (14−4) = 10 ms.

P3 : (4−4) = 0 ms

P4 : (10−6) = 4 ms

Avg = $\frac{14+10+4}{4} = \frac{28}{4}$ = 7 ms,

Response Time

(4) Round Robin (RR) Scheduling.

23/11/18
class 13/4

OS developer, define some time quantum

Time quantum ≈ time slice
→ 15 ms. (Let's say)

→ Processor will be allowed to execute a process at max one time quantum.

→ Ready queue is implemented as a circular queue.

→ It is always preemptive.

If Time Quanto is very large, RR will be same as FCFS.

If time quanta is very small there is so much preemption.

Choose time quanta such 80% of CPU burst is less than time quanta.

→ Once it is decided, it is hard coded in OS, if we have to change, we make new version and change programming.

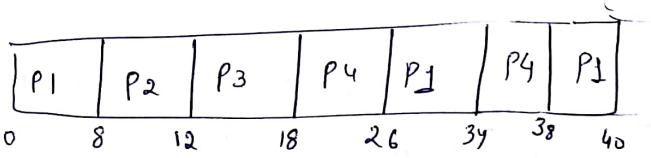
Ex→

Process	Arrival time (ms)	CPU burst time (ms)
P1	0	18
P2	2	4
P3	4	6
P4	6	12

TQ = 8 ms.

Ready Queue : ~~P1~~, ~~P2~~, ~~P3~~, ~~P4~~, ~~P1~~, ~~P4~~, P1

Gantt Chart :



Dispatch Latency = 1
CPU utilisation = 85.11%

1) Turnaround Time

P1 :- 40 − 0 = 40 ms

P2 : 12 − 2 = 10 ms

P3 : 18 − 4 = 14 ms

P4 : 38 − 6 = 32 ms.

Avg = $\frac{40+10+14+32}{4} = \frac{50+46}{4} = \frac{96}{4} = 24$ ms.

(b) Throughput

: $\frac{4}{40} = 0.1$ proc/ms,

(c) Waiting Time

P1 : (0−0) + (26−8) + (38−34) = 4+18 = 22 ms

P2 : (8−2) = 6 ms.

P3 : (12−4) = 8 ms

P4 : (18−6) + (34−26) = 12+8 = 20 ms.

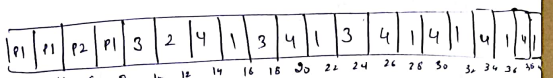
Avg :- $\frac{22+14+20}{4} = \frac{56}{4} = 14$ ms.

Q) Now

TQ = 2ms.

Ready Queue : ~~P4~~, ~~P1~~, P2, P1, P3, P2, P4,

Gantt chart :



→ Let us assume, that the process goes from ~~ready~~ running to ready part first.

a) Turnaround time

P1 : 40 − 0 = 40 ms.

P2 : 12 − 2 = 10 ms.

P3 : 24 − 4 = 20 ms.

P4 : 38 − 6 = 32 ms.

Avg = $\frac{50+52}{4} = \frac{102}{4} = 25.5$ ms.

b) Waiting Time

P1 :- (6-4) + (14-8) + (20-16) + (26-22) + (30-28) + (34-32) + (38-36)

= 2 + 6 + 4 + 4 + 6 = 22 ms

P2 : (4-2) + (10-6) = 6 ms,

P3 :- (8-4) + (16-10) + (22-18) = 4+4+6 = 14 ms,

P4 :- (12-6) + (18-14) + (24-20) + (28-26) + (32-30) + (36-34)

= 6 + 4 + 4 + 6 = 20 ms,

Avg = $\frac{22+20+20}{4} = \frac{62}{4} = 15.5$ ms,

CPU utilisation = 68.97%.

Dispatch latency = 1.

(5) Multilevel Queue Scheduling

→ Ready queue is partitioned into multiple queues.

Each queue will be there for each type of process.

For ex- seperate background process, user process, text processing.

Each of these queue, will be having its own scheduling algorithms.

two types of scheduling

① ↳ inter queue

② ↳ intra queue,

① we can used,

ⓐ Preemptive priority,

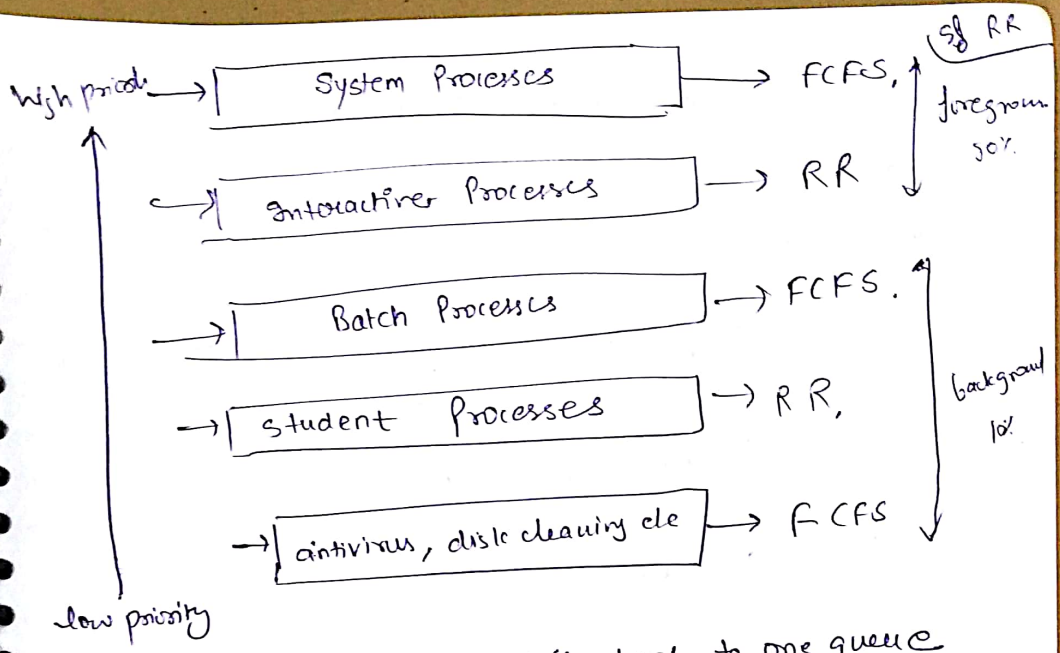
each queue will be having its priority.

ⓑ RR (80% time for foreground process (Ex- MS word,
and 20% time for background process)
Ex- Antivirus

② we can use,

it is upto OS developer,

book says it is FCFS.



* One process is assigned/located to one queue permanently,
* It ensures low waiting time for high priority processes.
* other processes too have satisfactory performance.

Problem

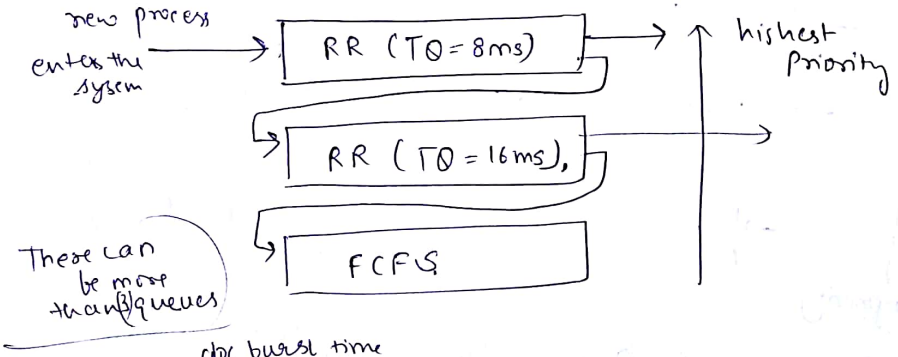
→ It is bit complex, and CPU scheduler becomes slow.

Q) Ex-5 on Pg-12

29/1/18
class-15

⑥ Multilevel feedback queue scheduling.

→ It is decomposed into multiple queues.
→ Each queue has its own priority.



Eg: If a process (cpu burst time) has less than 8 ms, then it terminated,
else
it go to next priority

→ Each queue is using its own algorithm, called intra queue scheduling.

→ interqueue scheduling also,
↳ If two processes, which queue to give,
use preemptive priority,

In ⑤ algo.
, process is assigned to queue for eternity,
process here can move in queue,

→ A process waiting too long in a low priority queue may be allowed to move higher priority queue.

One numerical given in notes, do it manually.

Generally numerical didn't come from algo ⑤ and ⑥ but → sometime bad luck ☹

Parallel Computing :-

→ If you can execute more than one instruction at a particular time,

→ Anything more than ~~to~~ one (1) can be (1.5)

→ Multicore processor

A single processor has more than one core, which can have ALU, AC, PC, registers, cache support.

→ Parallel computer
→ Distributed computer

, have multiple microprocessor on the same motherboard

Parallel computer = tightly coupled computer.

Distributed computer = loosely coupled computer

→ there is only one OS.
Ex- in parallel computing.

→ VLIW
→ Cluster Computing
→ Grid computer.
→ Cloud Computing
→ GPU

Read from Wikipedia

Multiple Processor Scheduling :-

→ We have multiple CPU.
→ We have to schedule them together.

Multiple processor system

homogeneous
→ All processors are same
→ symmetry in connection
→ RAM, peripheral devices are same

heterogeneous
(otherwise)

CPU scheduling

Asymmetric
→ there is one processor
→ one ready queue

Symmetric
→ Each processor is self scheduling
→ Each processor has its own ready queue

n processors → n queues

Symmetric ⇒ Two interesting issues

① Processor affinity

affinity of a process
or
necessity of a process to be executed,

→ we migrate a process, only when it is strongly needed, or it is worth.

Processor affinity

hard → we must execute processor on same processor.
Ex → If process has opened lot of files

~~soft~~

soft → If possible, we execute on same processor
Ex → processes complete

Load Balancing

→ All the processors should be equally loading.
→ for optimal use.

① push migration
if diff. in no. of process
then one processor can push processes,

② pull migration
highly loaded can pull processes.

But there is some threshold, limit to every push or pull.

③ Combination of both
load-balancing,
balance b/w ~~migration~~ & processor affinity.

Also consider affinity.
If a process wants to continue execution on a processor, then that process should not pull/push.

Chapter-3, 4, 5 done

New chapter ☺

30/1/2018
Class-16, 17

Process Synchronization

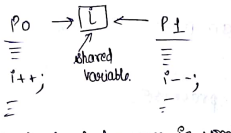
Cooperating Process.

→ If a process it can affect the execution of another process, is called cooperating process.

How?
by. → Shared variable.
→ Shared file
→ message passing
→ shared device

Why necessary?
→ to do some complex computing.

Let,



any high level language is compiled, into machine language, we use RTL, here.

P1

R1 ← i
R1 ← i+1
i ← R1

R2 ← i
R2 ← R2-1
i ← R2,

Let,

Initially: i = 10.

P0 : R1 = 10

P1 : R2 = 10

P0 : R1 = 11

P1 : R2 = 9

P0 : i = 11

P1 : i = 9

but if the order is different, we execute P1 before P0 then, at the end, result will be different.

This problem is known as a race condition,

↗ we have to do avoid this because it will unpredictable result.

We can avoid race condition by Process Synchronization

P0 and P1 should consult each other before changing any value

When P0 executed, P1 should not interrupt or vice-versa

↳ To ensure, that only one process can manipulate data at one time

Shared variable ← both can read,

→ Process synchronization is the task of a programmer

→ OS provides some mechanism to ensure process synchronization,

Critical Section Problem.

is a way to model cooperating process,

We have, $P_1, P_2, P_3 \ldots P_n$. (processes)

- all of them are using some shared variable,
- whole code is not accessing

but only part of code is accessing shared variable called critical section,

Critical section problem is to ensure that only one its process is executing its critical section.

Program.

```
do {
    Remainder section
    Entry section.
    Critical Section
    Exit section
    Remainder section
} while(TRUE);
```

a process, request or seeks permission, to enter its own critical section

only enter when other n-1 processes agrees otherwise wait force

> section that says everyone that its own critical section is complete, and other process can not execute its critical section.

1/2/2018
Class-18

Three properties

> necessary condition for a solution of critical section problems —

(Short question can come - ?

Please use terms as mentioned.

(1) Mutual Exclusion

→ It is a locking mechanism.

→ At one time, one of the process can access (Both read & write)

shared variables

(2) Progress.

$k \leq n$

Which process to come in critical section next will decided by the all processes which are in entry section ~~and~~ or in the exit section.

* Decision can not be ~~delay~~ postponed, indefinitely.
* which process gets to progress next.

(3) Bounded Waiting

There should be upper limit of the no. of times Other processes are allowed to enter in its critical section after one process request to enter into its critical section, ~~Otherwise it will lead to~~ before that request is granted. Otherwise, it will lead to starvation.

Peterson's Solution

proposed a solution for critical section problem. (only for 2 processes), → P_0 and P_1

On this solution,

there are two shared variable,

```
int turn;
Boolean flag[2];
```

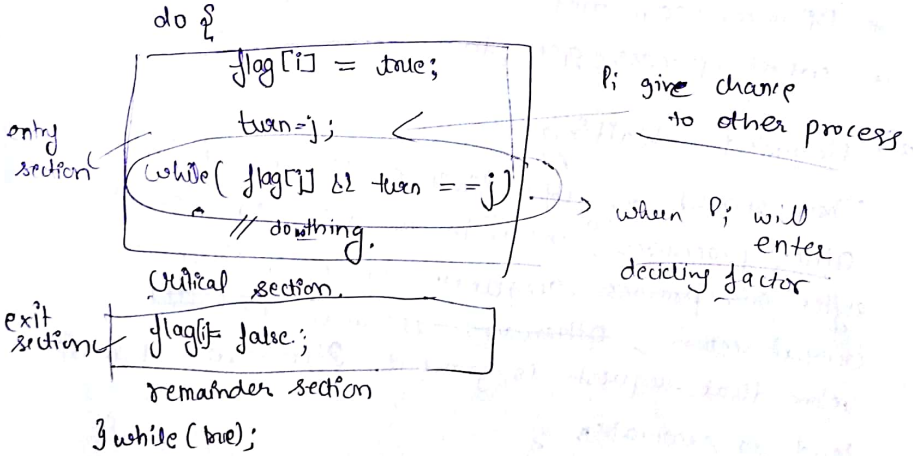
* flag[0] = True → P0 wants to enter into its critical section

if P0 exit;
flag[0] = false,

Same with P1,

flag[1] = ~~True~~ False.

Code for process i, Pi —



Q Prove that peterson's solution satisfy the 3 properties?

bounding waiting, (threshold value)

In this case it is ~~0~~ one (1)

5/2/2018
Class-19
(Not Attended)

Semaphores.

→ Low level construct which can be used to solve Critical section problem

→ better than Peterson's solution.

→ A semaphore is an integer variable, but it is not an ordinary integer.

→ It can be accessed by 2 atomic operations.

Atomic → "indivisible"
either the whole operation occurs or it does not occur at all,

What are these 2 atomic operations?
↳ wait()
↳ signal()

→ Semaphores can be used to ensure process synchronizatio among multiple processes.

→ can solve multiple process critical section problem.

→ only 1 process can access a semaphore at a time.

Assuming S is any semaphore :-

```
wait(S)
{ while (S<=0)
      ; // busy waiting
}
signal(S) { S++; }
```

"Busy doing nothing" ;")

Semaphores are of 2 types :-

(1) Binary Semaphore
→ can have only 2 values (0 or 1)
→ also called mutex / mutex lock (can be used to lock a process)
→ always initialized to 1

Let S be a binary semaphore :-

```
do {
    wait(S);
    CS
    signal(S);
    RS;
} while(true);
```

(2) Counting Semaphore

When the process are accessing multiple clone copies of the same resource,

→ initialized to (N) (no. of copies of that shared resc

→ control access to resources that have multiple instances.

Semaphores can create 2 problems -

→ Starvation

→ Deadlocks.

```
P0                  P1
wait(S)             wait(Q)
wait(Q)             wait(S);
 =                   =
 =                   =
signal(S)           signal(Q);
signal(Q)           signal(S).
```

Deadlock Sample

Producer Consumer Problem (Bounded-buffer problem)

⇒ Producer processes are producing some abstract items.

→ There are some consumer processes consuming those abstract items.

Semaphores

denotes the no. of buffers which are currently empty

```
mutex = 1 ; // access the buffer
empty = N ;
full = 0 ;
```

Producers

```
do {  // produce an item                    } RS
      wait (empty);                         } entry section
      wait (mutex);
   // add item to the buffer  → CS ;
      signal (mutex);                       } Exit section
      signal (full);
```

(shared resources i.e. buffer is being accessed)

Consumers

```
do {  wait (full);
      wait (mutex);
   // remove an item from buffers
      signal (mutex);
      signal (empty);
   // consume the item
} while (true);
```

class 20
6/2/2018

Readers - Writers Problem

Shared data -

int readcount = 0;

Semaphores -

mutex = 1; } binary semaphore

wsl = 1;

Writers

```
do{
wait(wsl);
//write
signal(wsl);
}while(true);
```

(critical section)

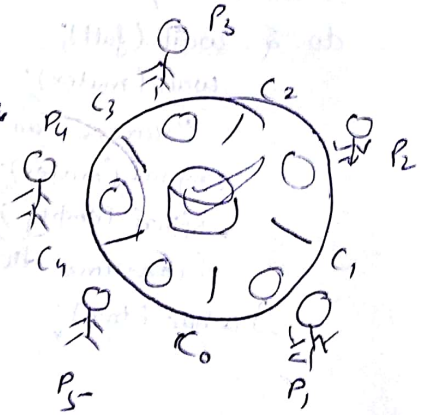
Readers

```
do{
wait(mutex);
readcount++;
if(readcount==1)
wait(wsl);
signal(mutex);
//read
wait(mutex);
readcount--;
if(readcount==0)
signal(wsl);
signal(mutex);
}while(true);
```

(Entry section, critical section, exit section)

Next Critical Section

Dining Philosophers Problem.



We have only 5 chopsticks

You have to write processes for philosopher, and there should be synchronization,

5 (chopsticks aren't identical)

* In case of buffer location was irrelevant.
* Here order matters:

5 semaphores needed.

Semaphores –

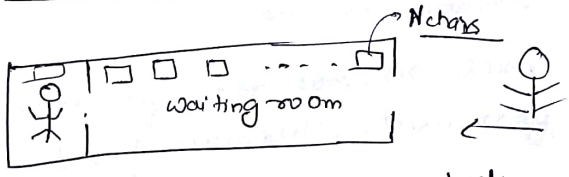
```
chopsticks[5];

P;
do {
    wait (chopstick[i]);
    wait (chopstick[(i+1)%5]);
    // Eat.
    signal (chopstick[i]);
    signal (chopstick[(i+1)%5]);
    // Think
} while (true);
```

what will be happen if all the philosophers get hungry at the same time?

Deadlock
No will able to eat, all will die (☹)

2. Sleeping Barber Problem.



Case 1 :- No empty chair, goes back.

Case 2 :- Some vacant chairs, goes & sit.

Case 3 :- All vacant, goes into barber's room

barber is busy with a customer
↓
when barber gets free, catches one of the customer and cut his hair.

barber is free and is sleeping
↓
customer wakes up the barber.

Barber

wait_seat[n];

for barber

Use counting sem

Class-2
8/2/2018
(Not Attended)

Deadlocks.

Multiple process trying to access same H/w or S/w.

Request
Permission granted } = writing state
Use resource
release that resource.

request → use → release.

- Processes are waiting indefinitely
- OS handles deadlock → Job of OS.

Resources: H/w or S/w.

Resource type:

1. Resources which have some functionality and can be used in exchange of each other → instances of that particular resource type.

eg - B, C, D, E → general purpose resource type.

- if necessary not complete, then will not happen but if complete, then may or may not occur.

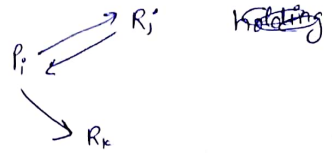
if sufficient complete, then will happen, but if not complete, then may or may not occur.

/* (Read monitor on your own) */

Necessary conditions for deadlocks;

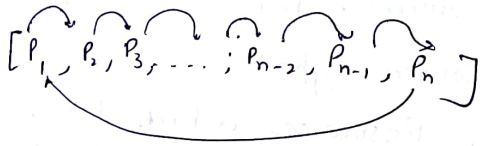
1.) Mutual exclusion: There must be one or more non sharable resource in system. Ex- printer, register.

2.) Hold & wait : Now Process P_i is holding R_j



3.) No preemption : There are some resources which can be preempted like HL register, suppose one process is using HL & some other process needs it, so it can snatch,
At the same time, there are certain resources like printer, speakers which can't be preempted,

4.) Circular wait :



a process waits for a resource held by the next process.

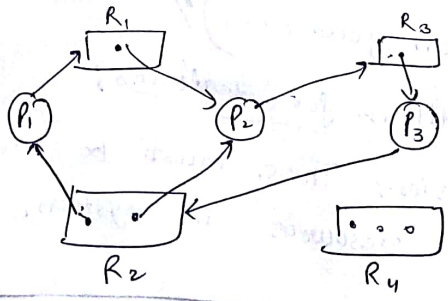
If there is hold & wait, only then we can have Circular wait.

If we can ensure any one of the four is false, then no deadlock.

Resource allocation Graph.

Circular nodes to represent the processes (P1)

Rectangles to represent the resources



Dots represent the instances of the resource

Request Edge : when a process requests for a resource.

Assignment Edge : when a proces is assigned a resource.

If there is a cycle, there can be deadlock. (necessary condition).

there are 2 cycles: $P_1 R_3 P_3 R_2 P_2$ & $P_1 R_1 P_2 R_3 P_3 R_2 P_1$

if each resource has only one instance, then occurence of cycle is sufficient condition for deadlock.

If in a graph, some resources have one instance & some have multiple instances, then if in a cycle if each resource has one instance then sufficient condition for deadlock.

Methods for deadlock handling :-

→ Deadlock prevention :- (prevention is better than cure).
To ensure atleast 1 of the 4 condition can't occur.
cycle detection $O(n^2)$
$O(E)$.

→ Deadlock avoidance :- Each process tells the OS its maximum requirement of resources as soon as it enters. Now it becomes the job of OS to schedule processes such that no deadlock occurs.

→ Deadlock detection & recovery from deadlock.

- Only one of the above three is required.
- If deadlock occurs, some processes hang.
- Deadlocks are very rare, once in a year.
- Modern OS don't handle deadlocks.
- The user has to kill such processes on their own (lol 😉).

Deadlock Prevention

Class-22
12/2/18
(Not attended)

(1) Mutual Exclusion

(2) Hold & Wait.

(1) → Protocol: Request & get all the resources at the beginning (no hold & wait).

Hold & wait condition made false, low resource utilization and starvation of process,

(2) Protocol :- should release its current resources before requesting for more resources,

Releasing the current resource & then asking for more resources, Not result into low resource utilisation

→ If a process require several resources at same time, then they (processes) may end up starving.

(3) No preemption

Some resources like files, registers, memory location can be preempted,

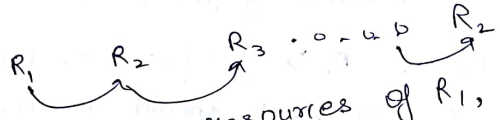
if we can make all resources preemptable,

(4) Circular wait.

Circular wait implies hold & wait. If we can ensure that there is no circular wait, then no deadlocks (hold & wait can be still there),

Arrange the resources types as $R_1, R_2, R_3 \ldots R_n$ (fixed order).

Protocol: a process can request resources is an increasing order only when process has obtained needed instances of R_1, it can ask for R_2 & so on. & it has got resources for R_2, it cannot ask for R_1 again. No circular wait happen.



Requires multiple resources of R_1, it should asks the same resources at same time.

(2) Protocol: If a process requests for R_i, then it must release.

R_j for $\forall j \geq i$

Eg. $i = 3$

R_1, R_2, R_3 (R_1)

process requesting for R_3 resource. it must release all R_j's where $j \geq 3$, release (page wale) resources if acquired by mistake.

Disadvantage: ↳ low resource utilization
↳ low throughput

- all instances of a resource must be acquired at the same time.

- Difficult to implement.

Deadlock avoidance

requires matrix addition & subtraction.

Problem : matrices can be very large & their size not static, but varies.

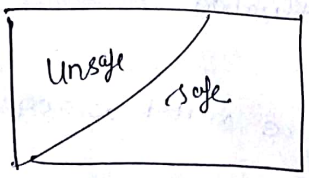
Safe state : State of the system which denotes the maximum no. of resources that can be allocated w/o creating resources.

Safe sequence : Sequence of processes

$< P_1, P_2, P_3 \ldots P_n >$ if for each process P_i, request for resources which should be less than or equal to ~~some~~ sum of resources of P_j ($j < i$) & the free resources.

Sequence $< P_1, P_2 \ldots P_n >$ is safe if for each P_i, ~~request for resources which should be less than or equal to su~~ the resources that P_i may still request can be satisfied by the free resources plus all resources held by P_j with $j < i$

A process must execute with its own resources, plus free resources & resources of processes on left.



if safe, then no deadlock
if unsafe, there may be a deadlock

Unsafe state is a necessary condition for deadlock

Banker's Algorithm ✱ Must (High probability)

class-23, 24

13/2/18

(Numericals)

Whenever a process, asked for more resources, then OS calculate if it is possible to allocate, then it will granted resource.

→ whether allocating resource leave in a safe state or not.

We have,

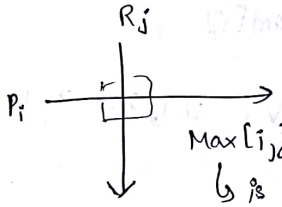
n processes

m resource types

Data structures —

→ Available [m] // gives no. of free instances of each resource

→ Max [n x m]



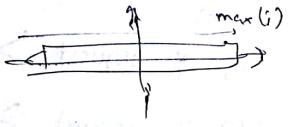
↳ is the max no. of instances of resources R_j required by process P_i.

→ Allocation [n x m]

how many instances of resource R_j is allocated to process P_i.

→ Need [n x m] Max − Allocation

denotes how many instances of R_j can be further requested by P_i



→ Max [i] → denotes.

Algorithm has two parts

1.) (i) Safety
(ii) Resource Request

(i) Safety Algorithm

checks whether a system is in a safe state or not.

Steps

1.) Let Work[m] and Finish[n] be vectors.

Work → integer
Finish → boolean.

Work = Available
Finish[i] = False, $\forall i = 0, 1, 2, \ldots n-1$

2.) Find a process P_i such that (Finish[i] == False && Need[i] <= Work)

If no such P_i can be found then goto 4

3. Finish[i] = true
Work = Work + Allocation[i]
Goto 2

4.) If finish[i] == true. $\forall i = 0, 1, 2, \ldots n-1$, then state is safe
else unsafe.

Time complexity : $O(mn^2)$.

(ii) Resource Request Algorithm

P_i is requesting for resources and Request[m] gives the request

Steps

1.) If request > Need[i], then error.

2.) If request > Available, then wait.

3.) Pretend to allocate the resources.

Available = Available − Request

Allocation[i] = Allocation[i] + Request

Need[i] = Need[i] − Request

If the resultant state is safe, then allocate resources else undo changes. Allocation, need, and available.

Q→ A system has 4 processeses, namely P_1 to P_4 and three resource types. A system in a safe state, with available = [1 1 1], Max = $\begin{bmatrix} 0 & 1 & 1 \\ 6 & 2 & 4 \\ 4 & 3 & 5 \\ 1 & 2 & 1 \end{bmatrix}$

and Allocation = $\begin{bmatrix} 0 & 0 & 1 \\ 4 & 2 & 3 \\ 3 & 3 & 3 \\ 0 & 2 & 1 \end{bmatrix}$

→ Which of the following request should be granted?

a) Request = [1 0 1] from P3

b) Request = [1 0 1] from P2

Soln RR algo —

Need = $\begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 1 \\ 1 & 0 & 2 \\ 1 & 0 & 0 \end{bmatrix}$

step 1 ×

step 2 ×

Now go to step 3

Available = [0 1 0]

0?)

Allocation = $\begin{bmatrix} 0 & 0 & 1 \\ 4 & 2 & 3 \\ 4 & 3 & 4 \\ 0 & 2 & 1 \end{bmatrix}$ Need = $\begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$

It is resultant state.

Now we have to check, whether it is safe state or not.

We check it by Safety Algorithm.

Initially =

Select	P1	Work	Finish
		0 1 0	F F F F
Select	P3	0 1 1	T F F F
Select	P2	4 4 5	T F T F
Select	P4	8 6 8	T T T F
		8 8 9	T T T T

this is a safe sequence (need not be unique)

Yes, request can be granted.

Next week se presentation

Your favourite topic (for 5 minutes) (No ppt)

Syllabus will be completed on Thursday

from monday (1 to 10 roll number)

(If somebody is absent) (He/She will get bengali rasgulla)

Class-25
15/2/18

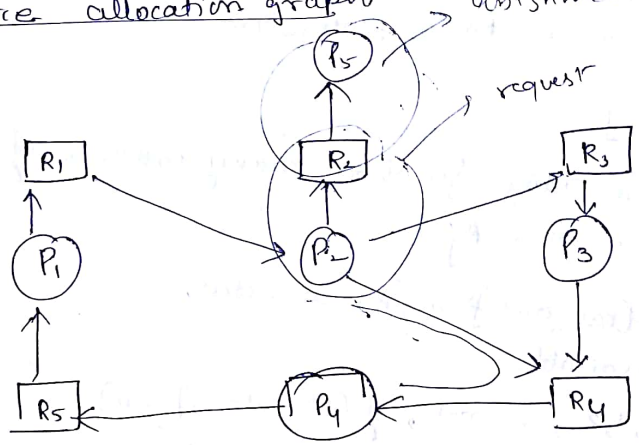
Deadlock Detection

Handling deadlocks

Let us assume, that each resource type has only one instance,

then we can use graphical methods for detection.

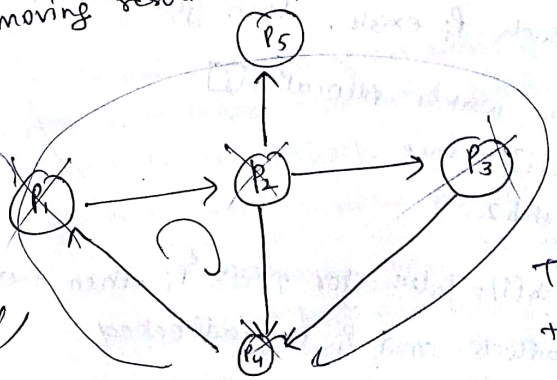
Resource allocation graph



If there is a cycle in a graph, then there will be a deadlock,

Wait-for graph

removing resources nodes



all are involved in deadlock. (deadlocked)

there are two cycles.

There are two deadlock

Cycle detection algorithm

↳ Time complexity $O(n^2)$

↳ Use this providing all resource type has single instance

(To detect deadlock)

~~be applied on~~ in which resource type has multiple instance

we use

Modified Version of Banker's Algorithm

Request [n x m]

↳ which process asking for how many instance of a particular resource R_j)

1.) Let work[m] and finish[n] be vectors.

work = available

For i = 0, 1, 2 . . . . n-1, if (Allocation[i] != 0)

then finish[i] = false

else

finish[i] = true,

2.) Find a P_i such that (finish[i] == false && Request[i] <= work)

If no such P_i exists, then goto step 4.

3.) Work = Work + Allocation[i]

finish[i] = true

Goto step 2

4.) If finish[i] = false for some P_i then there is a deadlock and P_i is deadlocked

Time Complexity : $O(m \times n^2)$

(In this topic, no numerical ydf)

Q.) How often to use this algo?,

→ at fixed time interval,

→ when you are suspicious,
(when you feel/ think there may be deadlock.

↳ Ex→ CPU utilization falls, beyond/below, a threshold.

Recovery from deadlocks.

→ (1) Process termination, or kill processes,
(terminate all the deadlocked process)
but you are decreasing overall system performance
(not efficient)

→ (2) Kill one process,
then check again that deadlock remove or not,

Kill processes one by one,
(bit efficient),

→ (3), Resource Preemption.
take away ~~all~~ resources from waiting process and give it to other deadlocked process
→ we have to use rollback
→ we have to select a victim process.
↳ a process may be starved if we have not used good logic or not good selection.
• Same process is getting selected again and again

3 types → Deadlock avoidance
Deadlock prevention
Deadlock detection

completed (till midsem)

Coding → Process synchronization
↳ Peterson algorithm or changes in them
↳ pseudo code of classical (3) problem in process synchronization
↳ code for monitors.

→ Theoretical question
(in depth) (not simple)

→ Read Threads (question can come), (short note)
4.1, 4.2 (6th edition)

→ numericals (main source,
↳ CPU scheduling
↳ banker
↳ use of banker in detection

1.1, 1.4 – 1.9, 1.11
2.1 – 2.5, 2.7, 2.10
3.1, 3.2, 3.3, 3.4
4.1, 4.2
5.1, 5.2, 5.3, 5.4
6.1 – 6.7
7.1 – 7.7