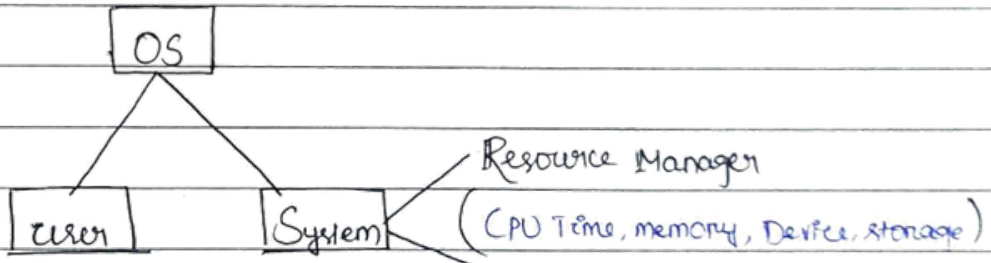
classmate
Date 12·10·22
Page

:- Operating System :- Class no:- 2

- Introduction to Operating System:-
- What is System?
- Every System has a set of independent components and each component having their own functions. Components are linked according to a plan for achieving Goal. Exchange of information is happened among all Components.

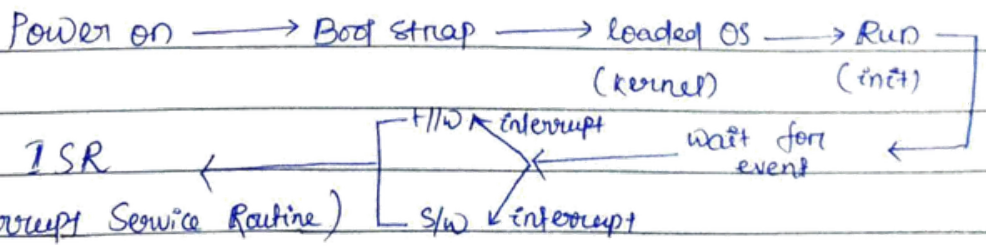
- What is Operating System?

Operating System is a System software that acts as intermediary between the user and computer hardware. The purpose of OS is to Provide a convient environment in which user can execute programs in a convient and efficient manner. It is a resource manager responsible for allocating and de allocating System resources and it controlls program which controls the operation of computer hardware and manages execution of user program to prevent Error from occuring and improper uses of Resources.



- acts as a interface between user and hardware
  - PC - Ease user
  - mainframe - CPU Utilization Required
  - Handheld (mobile phone)

- How you switch on your System?



classmate
Date 13.10.22
Page

Class no: 3

Computer System Architecture :→

- Single processor
- Multi processor
  - Symmetric
  - Asymmetric
  - Multi Core System.
- Cluster System :-

Operating System structure :→

- Batch System.
- Multi programming. - Capale to run one or more program (job)
  - job pool - set of job which are going to be executed.
  - job queue - present in main memory
  - job scheduler.
- Multi tasking — job comes from storage to memory.

Multi Programming :-

A multi program Operating System can execute a no. of program concurrently. The operating system fetches a group of program from the job pool. Which contains all the program (job) to be executed and places them in main memory. This process is called as job scheduling. Then it chooses a program from the ready queue (Queue, the main memory where job ready to be executed are stored) This process is called as CPU Scheduling and give it to CPU for execution when an executing program IO Operation OS fetches another program from ready and hand it to CPU for execution.

imp **Multi tasking / Time sharing System:→**

It is a logical extension of multiprogramming. Here user can interact with the program. The CPU executes multiple jobs by switching among them. This switching occurs so frequently that the user feel as if the Operating System is running only on his/her program. Some Time sharing System allows many user to use the System simultaneously. So the issues for Operating System are Job scheduling, CPU scheduling, memory management, storage management.

Class no:-4

Dt:- 18.10.22

- Real Time System.
- Operating System Operation

gmp ~~Dual~~ Dual-mode operations. Program in execution → Process.

Operating System Components:→

1. Process management:-
   - i) Creation and killing of process.
   - ii) Suspending a process.
   - iii) Provides mechanism inter process communication.
   - iv) Provides for process Synchronization
   - v) provide mechanism for deadlock handling.
2. Main Memory Management:-

```
int main()                    A.c -> program file.
{
Print ("Hello");
fork(); ——→ Child process // Clone of parent // Creates a new process.
Printf ("Bye");    * All are managed by OS.
return 0;
}
```

i) Keep track on which memory part being used and by whom
ii) Decides which processes are to be loaded into memory space becomes available.
iii) Allocating and deallocating memory space as needed.

3. Storage Management:-

i) free Space Management.
ii) Storage allocation
iii) Disk Scheduling.

4. File Management:-

i) Creating and deleting files.
ii) Creating and deleting directories.
iii) Supporting primitives for manipulating files and directories.
iv) Mapping files onto Secondary Storage.
v) Backing up files on to Secondary Storage.
vi) Backing up files on stable (non-volatile) storage media.

Class no:-05

Dt:- 19.10.2022

- Operating System Service.

- Provides services to programs as well as users.

1. User Interface (UI)
   - OS have UI.
   - Different interface can.
   i) CLI (command line Interface).
      - text comands
      - method for entering the next comands.

ii) Batch interface:-

- Comands are entered to feles and those files are executed.

iii) GUI (Graphical User Interface):-

- Window System.
- direct I/O.
- choose from menus and make Selection.
- Keyboard to enter text.

2. Program Execution:-

- System must be able to load a program in to memory.
- run the program

3. I/O Operation:-

- muser can't control I/O directly.
- specific function may be desired.

4. File System Manipulation:-

- read & write files
- create & delete "
- allow or deny access to file.

5. Communications:-

- inter process communication is done 2 ways.

6. ——→ memory share & Error detection.

- hardware generated Error.
- OS should take appropriate action to ensure correct & consistent computing

7. Resource allocation:-

- manages different resources.
- resource may have special allocation code.

8. Accounting:-

- we want to keep a track of
  - How many times is consumed.
  - Which kind of resources.

9. Protection :→

Class no:- 06

Dt:- 20.10.2022

System Calls :→

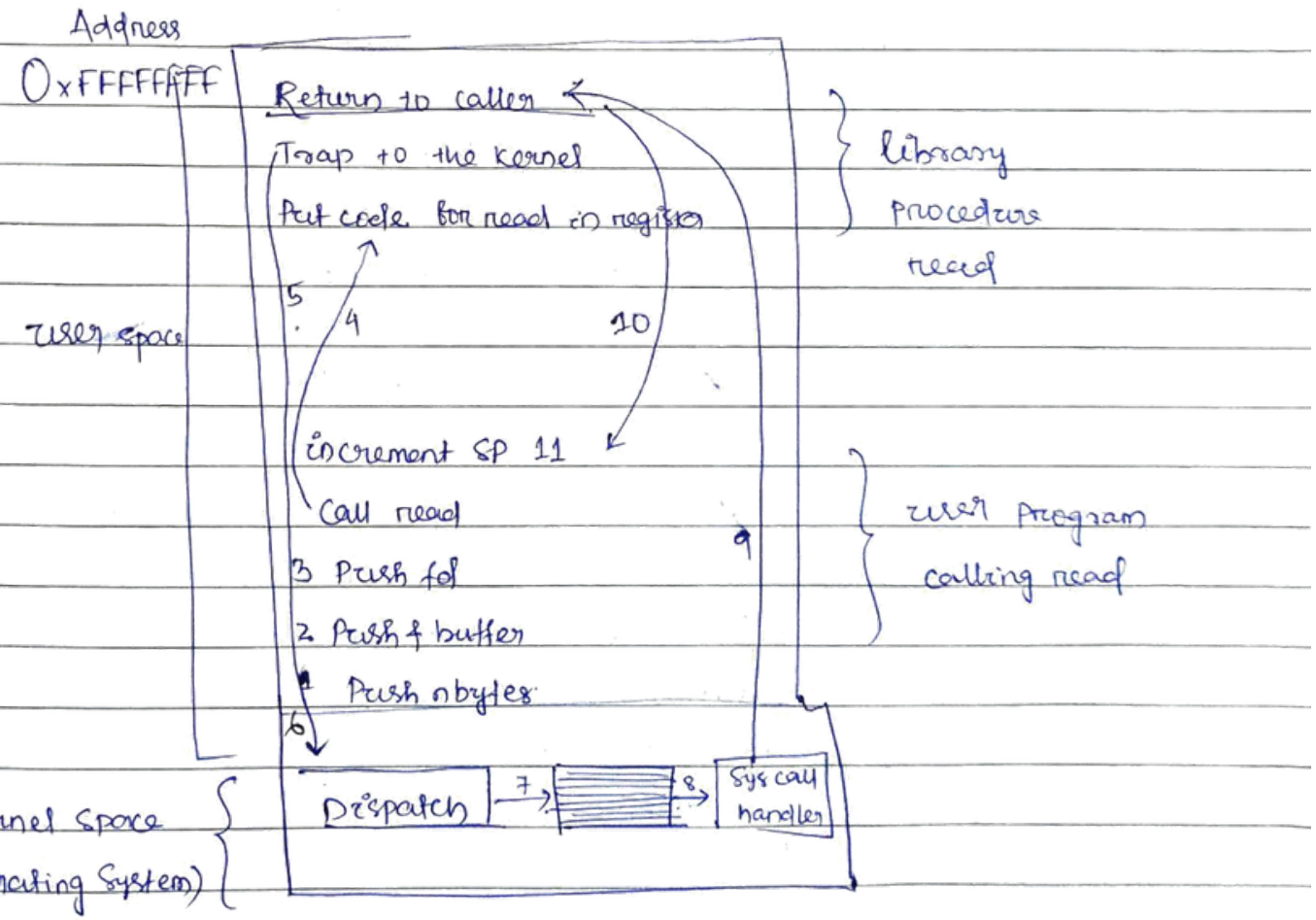
- These provide an interface to the OS Service.
- These are available as routines in C and C++.
- The programmers design programs according to API.
  (API = Application Program Interface).
- This is because bcz the following benefits:
  - Program portability.
  - Actual System calls are more detailed (and difficult) to work with than the API available to.

- The API defines a set of function that are available to the programmer.
- The API includes the parameters passed to functions and the return values.
- The function that make up an API invoke the actual System calls on behalf of the programmer.
- Three General methods are used to pass parameters to the OS:
  1. Via register
  2. using a table in memory & the address is passed as a parameter in a Register.
  3. The use of stack is possible where parameters are pushed on to a stack and popped off.
- Three of the most common APIs available to application programmers are the Win32 API for Windows Systems, The POSIX API for POSIX-based System (which include virtually all version of UNIX, Linux, and mac OS(X), and the Java API for designing

Programs that run on Java Virtual Machine.

- The run time support system (a set of functions built into libraries included with a compiler) for most programming languages provides a system call interface that serves as the link to the system calls made available by the Operating System. The system-call interface intercepts function calls in the API and invokes the necessary system call ~~in~~ within the Operating System. Typically, a number is associated with each system call, and the system-call interface maintains a table indexed according to these numbers. The system call interface then invokes the intended system call in the Operating System Kernel and returns the status of the system call and any return values.

C program might look like this: count = read (fd, buffer, nbytes)



Class no: 07
Dt:- 21.10.22

Operating - System Design Approaches :⇒

1) Kernel Approach :⇒

- The first OSs were written a single program.
- This approach to building the OS is called the kernel or monolithic kernel approach.
- Kernel usually refers to that part of the OS that implements basic functionality and always present in memory.

2> Layered Approach :⇒

- The Operating System is broken up into a number of layers (or levels), each built on top lower layers.
- The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface.
- A typical operating system layer consists of data structures and a set of routines that can be invoked by higher-level layers and can invoke operations on lower-level layers.
- The main advantage of the layered approach is modularity.
- The layers are selected such that each uses functions (or operations) and services of only lower-level layers. This approach simplifies debugging and system verification.
- A layer doesn't need to know how these operations are implemented; it needs to know only what these operations do. Hence, each layer hides the existence of certain data structures operations, and hardware from higher-level layers.
- The major difficulty with the layered approach involves the careful definition of the layers, because a layer can use only those layer below it.
- Another .

←: PROCESS MANAGEMENT SYSTEM :→

- A process is a program in execution. It includes current activities value of different registers, stack, etc.

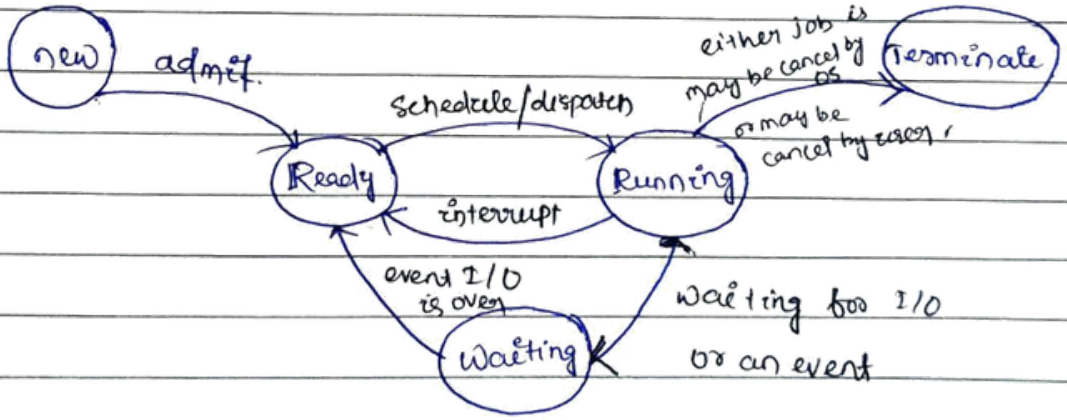
Difference b/w a Program & Process :→

Process	Program
- Program in execution is called process	- Set of instruction to carry out a task is called program
- It is a dynamic entity	- It is a static entity
- Resides Generally in main memory	- Resides in secondary storage.
- Time span is limited.	- Time span is unlimited.

State of Process :→

- Processes are in following states.

(i) new
(ii) Ready
(iii) Running
(iv) Waiting
(v) Terminated.



Class no:- 08
Dt:- 26.10.22

1. New :→ (State) :→

- When a process is created, it is said to be in new state and it is waiting to be admitted to the readyqueue.

2. Ready State :-

- A process is said to be in ready state if it is loaded to main memory, resources it needs are granted to it except the CPU time.
- The Ready processes are kept in readyqueue. Readyqueue may contain more than one ready process.

3. Running State :→

A process is said to be in running state. If instruction of it are being executed.

4. Waiting State :→

A process is said to be waiting state. If it is waiting for an event to occur like completion of I/O or message from other process etc.

5. Terminate State :→

A Process is said to be in terminate state if it is terminated or completed it's execution or it is killed.

Transition :→

1. new ⟶ Ready

- The state or process is changed from new to ready when it is admited to the Readyqueue.

2. Ready ⟶ Running.

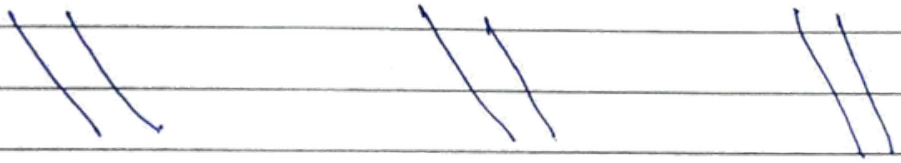
- The state or process is changed from ready to running when it is dispatched and CPU start executing it's instruction.

3. Running → Ready

- The state or process is changed from Running to Ready when it is pre-empted and the CPU is given to some other process.
- This may happen because of the arrival of a higher priority process time slice assign to it is over.

4. Running → Waiting.

- The state or process is changed from Running to waiting if the process wait for completion of event.
- The major reasons are.- Process request for I/O operations
- Process requests for memory or some other resources.
- Process wishes to wait for some interval of time.
- Process wishes to wait for some action by other process
- process wait for a message from other process.



5. Waiting → Ready.

- The state or process is changed from waiting to ready when the event for which the process waiting is over. like completion of IO.

6. Running → Terminated:-

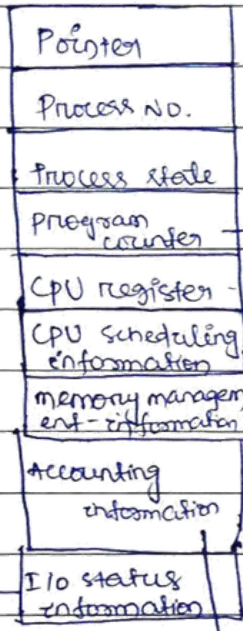
- The state or process is change from Running to terminate when the execution of the program is completed.
- The major reasons are.

1. Safe termination:-
2. It is terminated by the parent process.
3. If exceeds resource utilizations.
4. abnormal cond' during execution
5. incorrect interaction between other process.

Class no: 09
Dt:- 27.10.22

Process Control Block :→ (PCB)

- PCB stands for Process Control Block.
- The information regarding a process is kept in a data structure called as Process control Block. i.e each process is represented by this process control Block.
- It is also called as Task-control Block.
- PCB contains Generally the following instruction.



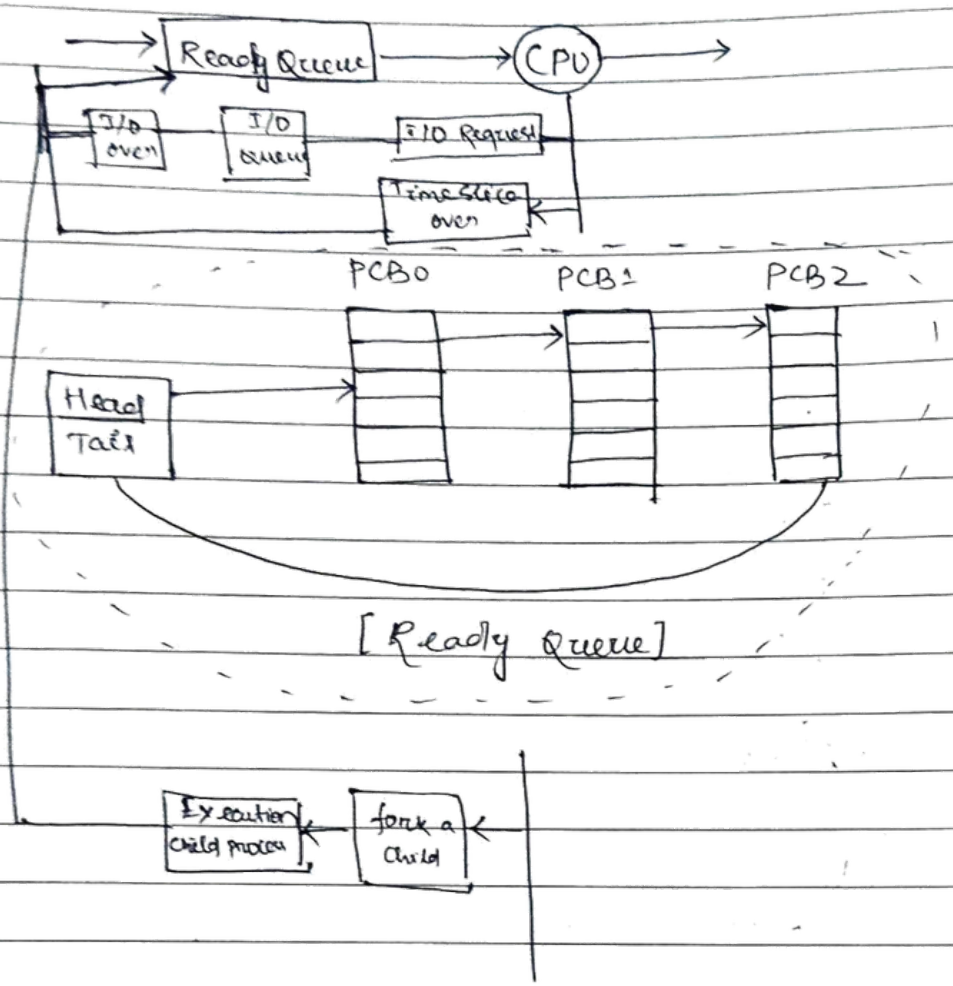
- PCB keep the data about the current state of the process like ready, running, waiting etc.
- It keep the address of the next instruction which is going to be executed
- PCB keeps data about CPU Scheduling like time slice, Priority, etc.
- PCB keeps memory management information regarding Process like address of base register, value of limit Register, address of pages, address of segments etc.
- PCB keeps accounting information like amount of CPU time used
- PCB keeps data like devices collected to the process, opened files.

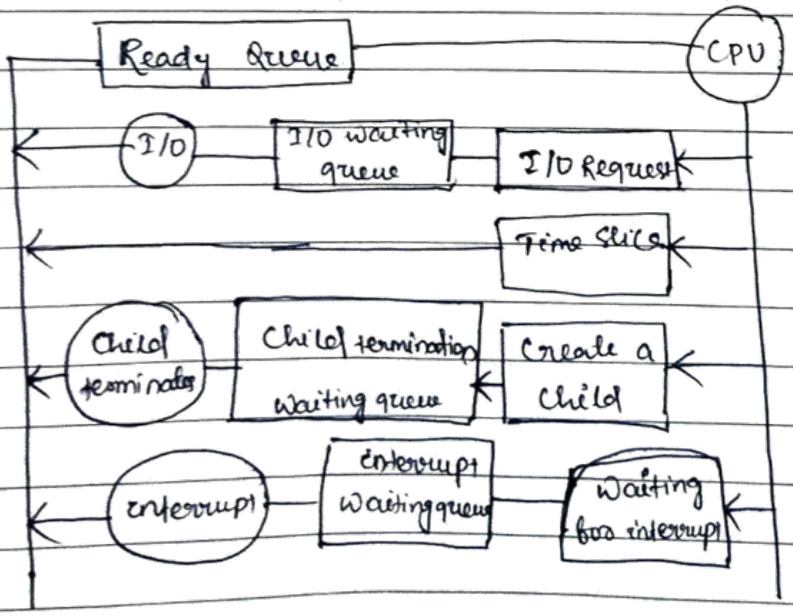
Scheduling Queue :→

Device Queue:-

Each Device has it's own device queue.

Queuing Diagram:-





Class no:- 10

Dt:- 28.10.22

Content Switch:→

- Switching the CPU to another process requires performing a state save of the current process and state restore of another process. This task is called as content switch.
- When a content switch occurs, the kernel saves the content of old process with the PCB and load the save content of the new process to run. some time is spend for this purpose. This time depends on the system.

Q What is CPU and I/O Burst?

Ans:- CPU burst is time interval when a process uses CPU only.

I/O burst:-

I/O burst is a time interval when a process uses I/O device only.

What is CPU bound job?

The job thats spend more time on computation are called as CPU bound job -eg:- Scientific program.

Q What is I/O bound job?

- The job that spends more time on input output operation like business application.
- CPU burst is longer in CPU bound.

Q what is CPU Scheduling:→

- ~~cpu~~ Sch The problem of determining when processor should be assigned and to which process is called as CPU scheduling or process scheduling.
- When Scheduling decission takes place:-
  1. when a process switches from running to waiting state
  2. when a process switches from running to ready state.
  3. when a process switches from waiting to ready state.
  4. when a process terminates.

- The Scheduling algorithm can be up to 2 types.
1. Non-Pre emptive Scheduling
2. Pre-emptive Scheduling.

Class no:- 11

Dt:- 01.11.2022

Non-Pre-emptive Scheduling :→

- Once a CPU is assign to a process. It can't be taken away before it CPU burst time is over.
- first come first serve algorithm.

Pre-emptive Scheduling :→

- In this type of scheduling CPU can be taken away from a process before the completion of it's CPU burst time.
- This can happen because of scheduling policies like round rubbing or arrival of higher priority process.

Scheduler:-

(i) long term Scheduler
(ii) short term Scheduler
(iii) - Medium term Scheduler

Long term Scheduler :→

- This is a part of OS. that determines which job are to be brought in to memory (main memory). This controls the degree of Multiprogramming. It selects a good mixture of CPU bound and I/O bound job.
- Degree of multiprogramming means no. of jobs currently in memory.

Short-term Scheduler :→

- This is a part of OS that decides which job / process will assign to CPU. It is also called as CPU Scheduler.

Medium-term Scheduler :→

- This Scheduler is used especially with time sharing System as an intermediate scheduling level
- A swapping Scheme is implemented to remove partially ~~swapped~~ run program from memory and reinstate them later to continue where they left off i.e. to perform Swapping Out & swapping in operation.

Dispatcher :→

- Dispatcher is a part of OS that gives the control of CPU to the process selected by short-term Scheduler.
- This function involves

i) Switching the Context
ii) Switching to User mode.
iii) Jumping to the proper location of the program to restart it.

CPU Scheduling Criterias :→

i) Turn around time
- The interval from the time of submission of the process on completion of the program is called turn around time
Turn around time = Completion time - Arrival time.

ii) Waiting time :-
It is the sum of periods spent in the readyqueue.
Waiting time = Turn around time - CPU time.

iii) Response time :→
Time from the submission of request until the first response.

(iv) CPU Utilization :→
It is the percentage of time, cpu is busy. Normally it is from 40% - 90%.

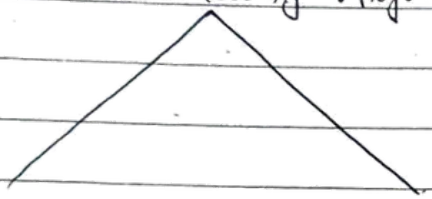
(v) Throughput :→

- no. of process that are completed in unit time.
- Aim is maximize CPU utilization and throughput and minimize turn around time, waiting time & response time.

Class no! - 12

Dt:- 02.11.2022

Scheduling Algorithm



Non-Pre-emptive

- FCFC
- SJF
- priority.

(Without pre-emption)

Pre-emptive.

- ROUND ROBIN
- Priority (with pre-emption)
- SRTF

First come First Serve CPU Scheduling Algorithm :→

- In this scheme, the process that request the CPU first with allocate the CPU First. Here ready queue is implemented as FIFO queue.
- It is one kind of non-pre-emptive.
- It is rarely used as a master scheme

Advantages :→

- Simple in logic
- simple to implement.

Disadvantages :

- Average waiting time is quite long.
- produce convoy effect i.e if a process with longer CPU time is in front of queue and shorter process are in the rear end of the queue, then shorter process has to wait for the longer period of time.

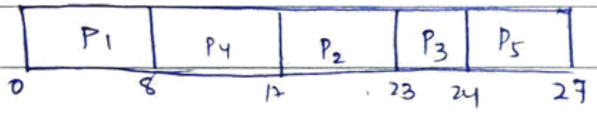
Q.1

Process	CPU Burst	Arrival Time	Turning around time	WT	RT
P_1	8	0	8 - 0 = 8	8 - 8 = 0	0
P_2	6	2	23 - 2 = 21	21 - 6 = 15	15
P_3	1	2	24 - 2 = 22	22 - 1 = 21	21
P_4	9	1	17 - 1 = 16	16 - 9 = 7	7
P_5	3	3	27 - 3 = 24	24 - 3 = 21	21

Avg = 18.2

WT same as Response Time.

Avg wi = 12.8 = Avg RT



Note:-

- If Arrival time not given then it assume to zero(0).

2. Shortest Job First (SJF) CPU Scheduling Algorithm:-

- This is a non-pre-emptive algorithm
- When CPU is available it is assign the process that has smallest next CPU burst.
- This algorithm is especially appropriate for batch job. This algorithm gives minimum average waiting time for a given set of process es. It favours shortest job for the expensive of longer job.
- It requires ~~besides~~ precise knowledge of how long the job will run.
- It is not useful in time-sharing system.
- The length of next CPU Burst is estimated using the following formula.

$$T_{n+1} = \alpha T_n + (1-\alpha) T_n$$

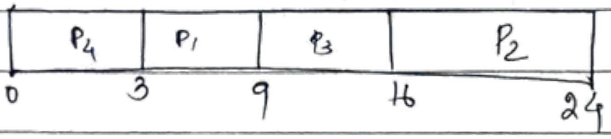
where $\alpha = \alpha$ $(0 \leq \alpha \leq 1)$

Class no:- 13

DP:- 03.11.2022

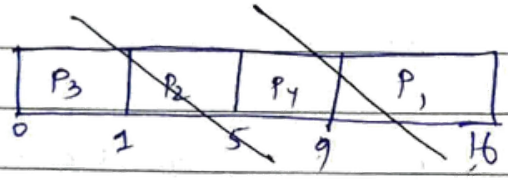
Ex-1

Process	CPU Burst	Turn around time	WT	Response Time
P_1	6	9 - 0 = 9	3	3
P_2	8	24 - 0 = 24	16	16
P_3	7	16 - 0 = 16	9	9
P_4	3	3 - 0 = 3	0	0
		Avg = 13	Avg: 7	Ave = 7

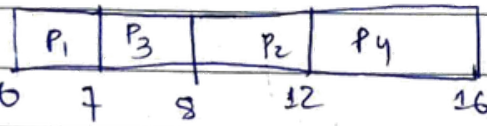


Ex-2

Process	AT	CPU Burst	TAT	WT	TAT	WT	RT
P_1	0	7	~~16~~ - 0 = ~~16~~	9	7	0	0
P_2	2	4	5 - 2 = 3	-1	10	6	6
P_3	4	1	1 - 4 = -3	-4	4	3	3
P_4	5	4	9 - 5 = 4	0	11	7	7
					Avg: 8	4	4



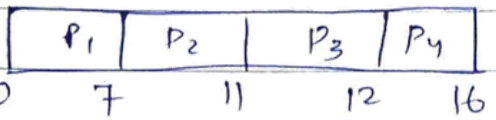
SJF



Solve the same problem with FCFS.

Process	AT	CPU Burst	TAT	WT	RT
P_1	0	7	7	0	0
P_2	2	4	9	5	5
P_3	4	1	8	7	7
P_4	5	4	11	7	7
			Avg:- 8.75	4.75	4.75

FCFS



Class no:- 14

Dt:- 04.11.2022

SJF

Shortest remaining time first :→

- It is the pre-emptive version of SJF
- In this scheme if a new process arrives with CPU burst time less than the remaining time of the current ~~exhibiting~~ executing process. Then pre-empt the currently executing process allocate the CPU to the new process.

Advantage :-

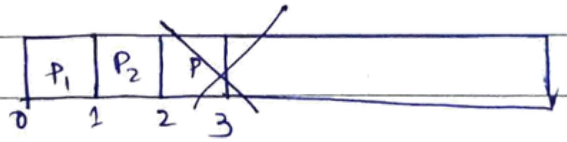
- minimize average waiting time.

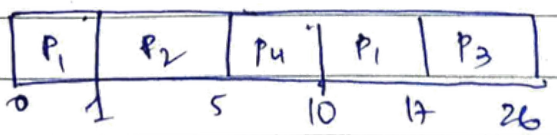
Disadvantage.

- Difficult to determine the next CPU Burst.
- Starvation may occur for the longer jobs.

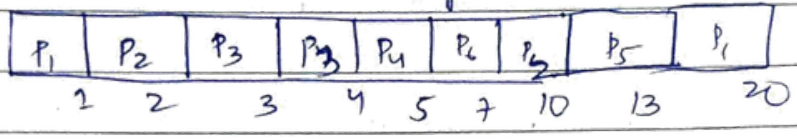
Ex:-

Process	AT	CPU Burst	TAT	WT	RT
P_1	0	8	17	9	0
P_2	1	4	4	1	0
P_3	2	9	24	15	15
P_4	3	5	~~7~~	2	2
		Avg	13	6.75	4.25





Ex	Process	AT	CPU Burst	TAT	WT	AT
	P_1	0	8	20	12	
	P_2	1	4	9	5	
	P_3	2	2	2	0	
	P_4	3	1	2	1	
	P_5	4	3	9	6	
	P_6	5	2	2	0	
				Avg.	4	



Class no:-15

Dt:- 09.11.2022

Priority Scheduling:→

- In this type of scheduling, A priority is associated with each process.
- Priorities can be assigned internally or externally.
- The CPU is allocated to the process with highest priority.
- FCFS is used to break the tie if any.
- It has 2 version
  1. non-Pre-emptive priority scheduling
  2. Pre-emptive priority scheduling.

Non-Pre-emptive priority Scheduling:→

In this scheme when CPU is available, It is assigned to a process with highest Priority.

Pre-emptive-Priority scheduling:→

- In this schem, when the process arrives in the ready queue,
- It's Priority is compared with the priority with the currently executing process. If the priority of the current Process is less than the priority of the currently executing process is pre-empted and cpu is assign to the newly arrived process.

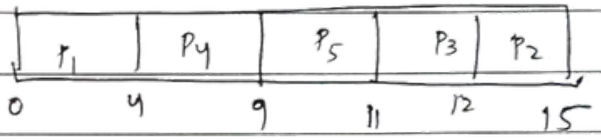
Problem in Priority Scheduling:⇒

- Starvation or indefinite blocking may occur to the lower priority process.
- Solution to starvation :- (Aging) ⇒

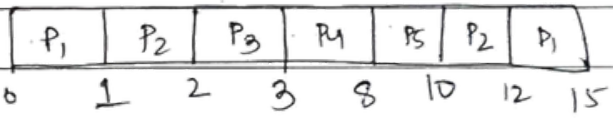
It is a process in which as time programmable priority of the process increases.

- e.g:- of Non-pre-emptive Priority Scheduling.

Process	AT	CPU Burst	Priority	TAT	WT	RT
P_1	0	4 3	2	4	0	0
P_2	1	3 2	3	14	11	11
P_3	2	1	4	10	9	9
P_4	3	5	3	6	1	1
P_5	4	2	5	7	5	5
				Avg 8.2	5.2	3.2



e.g:- of pre-emptive Priority Scheduling:⇒



Process	TAT	WT	RT
P_1	15	11	
P_2	11	8	
P_3	1	0	
P_4	5	0	
P_5	6	4	
	Avg 7.6	Avg 4.6	

Round ROBIN Algorithm :→

Class no:- 16

Dt: 10.11.2022

→ It is one type pre-emptive algorithm

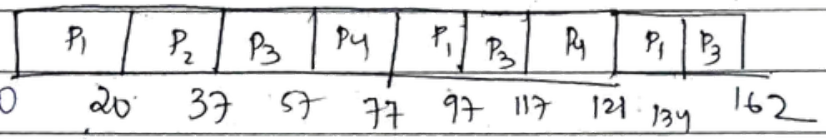
→ each process gets a small unit of CPU Time called as Time slice or Time quantum.

→ After the time slice is over the process is pre-empted and added to the end of the ready queue.

- CPU Scheduler goes around to the Ready queue allocating CPU to each Process for a interval of 1 time quantum/slice.
- Ready queue is implemented as a circular queue.
- If a process has a CPU burst less than 1 time quantum it releases the CPU voluntarely after it's completion.

Ex:-

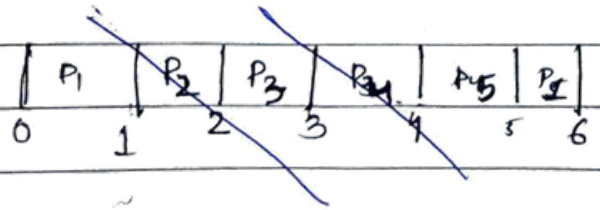
	Process	CBT	TAT	WT	RT
	P_1	~~53~~ 33	134	81	0
	P_2	~~17~~ 0	37	20	~~20~~
TS=20	P_3	~~68~~ 48	162	94	~~19~~
	P_4	~~24~~ 4	121	97	~~2~~
			113.5		

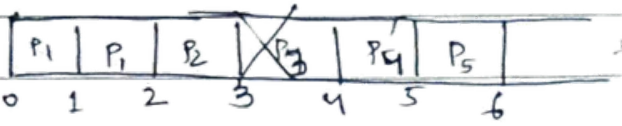


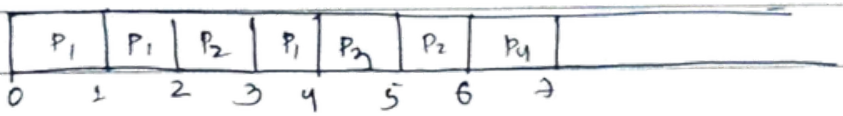
Ex:-2

Process	AT	CBT
P_1	0	~~3~~ 2
P_2	2	~~3~~ 1
P_3	3	5
P_4	4	2
P_5	5	3

TS=1







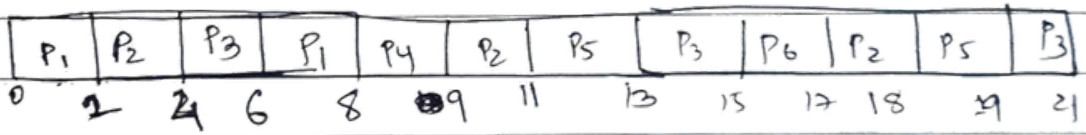
Class no:- 17

Date:- 11-11-2022

Process	AT	CBT
P_1	0	~~4~~ 2
P_2	1	~~5~~ 3
P_3	2	~~6~~ 4
P_4	~~3~~ 4	1
P_5	~~4~~ 6	3
P_6	~~6~~ 7	2

TS=02 X

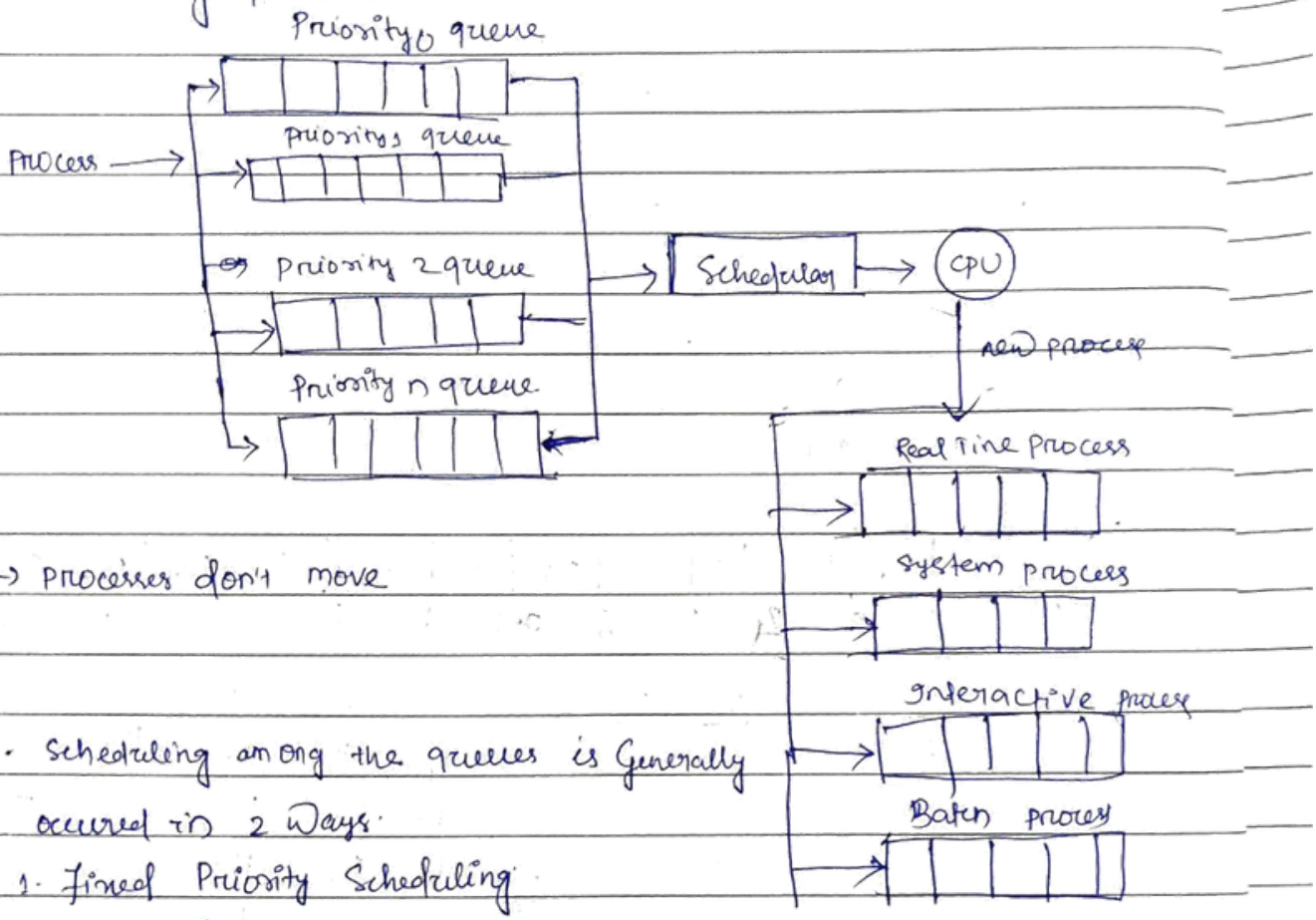
Gantt chart



0	1	2	3	4	5	6	7	
P_1	P_2	P_2 P_3 P_1	P_3 P_1	~~P_3~~ P_1 P_4 P_2	P_1 P_4 P_2	~~P_1~~ P_4 P_2 P_5	~~P_4~~ P_2 P_5 P_6	P_2 P_5 P_6

Multi-level Queue Scheduling :→

- Ready is partition into no. of separate queues on the basis of types of Process.
- Each queue may have it's own scheduling policy.
- The processes are permanently assign to one queue based on some properties of the process.
- Scheduling policy must be there for queue as well as among queues.



→ Processes don't move

- Scheduling among the queues is Generally occured in 2 Ways.

1. Fixed Priority Scheduling
2. Time Slice.

1. Fixed Priority Scheduling :→

Each queue has absolute Priority over lower priority.

2. Time Slice :→

- Each queue gets certain amount of CPU time which it can Schedule among the processes in that queue.

classmate
Date
Page

Class no:-18

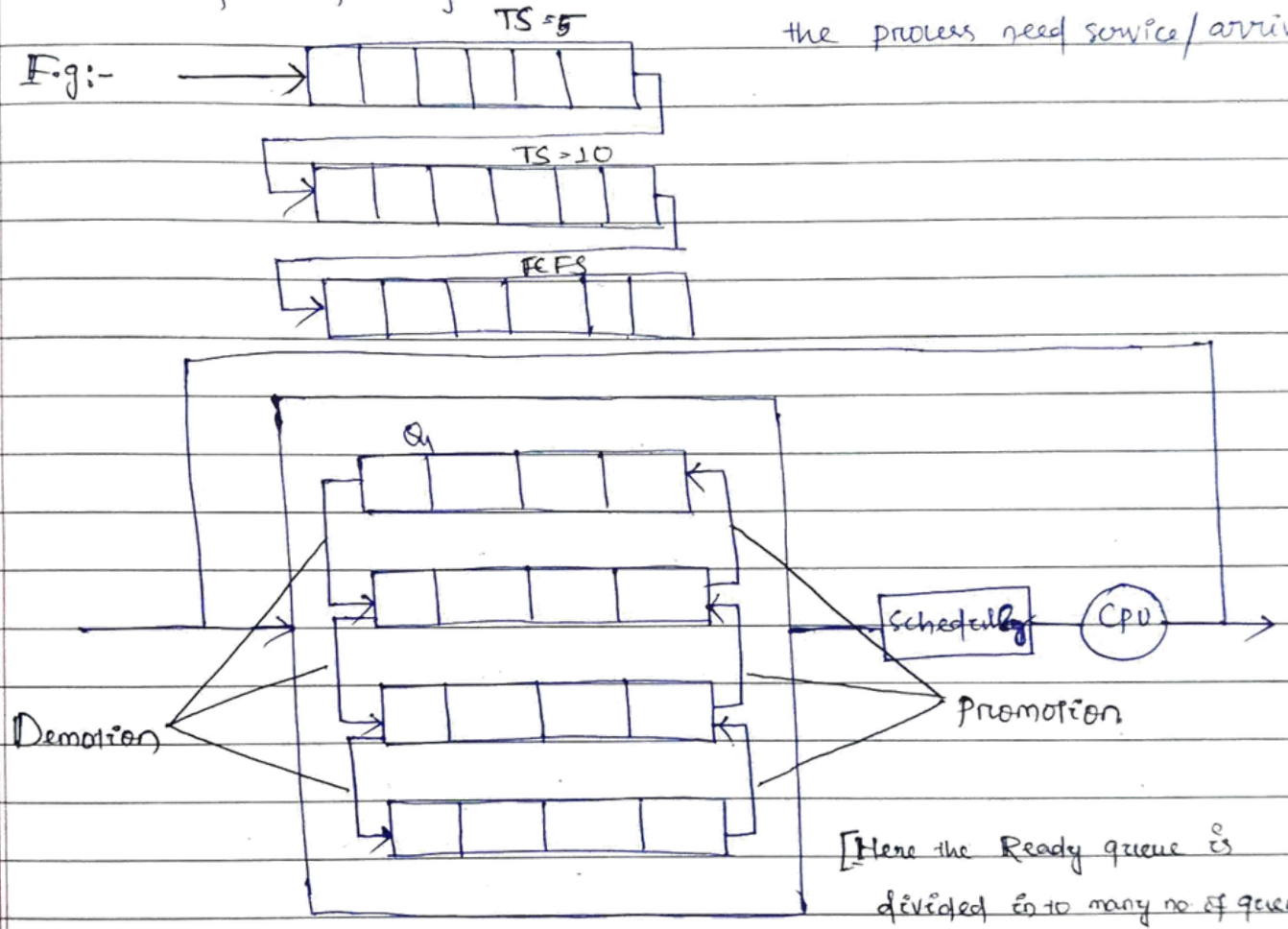
Multi-level-Feedback Queue Scheduling:→

DT:-16.11.2022

- In this scheme ready queue is partitioned in to no of separate queues
- a process can move between various queues.
- The Scheduler is defined by the following parameters

1. The no. of queues
2. Scheduling algorithm for each queue.
3. Method used to determine when to upgrade a process.
4. Method used to determine when to demote a process
5. Method used to determine which queue a process will enter when the process need service/arrives.

F.g:-



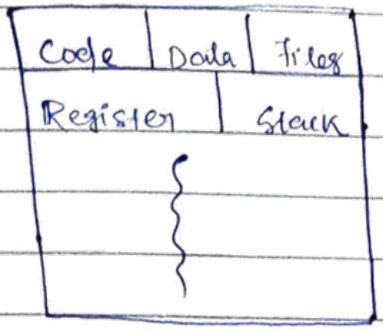
[Here the Ready queue is divided in to many no of queues]

[Diagram of multi-level-Feedback Queue Scheduling]

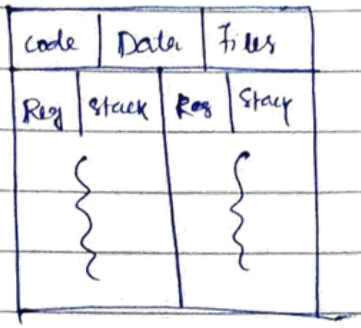
Threading:-

- Thread is a light-way process. It is a basic unit of CPU utilization
- It is a path of execution with in a process
- It consists of program counter, a stack & set of register etc.
- It share with peer threads code section, data section, OS resources like files etc.

- A process can contain one thread or can contains many threads.
- If the process contains more than one thread is called multithreaded program.



[e.g: Single threading]



[e.g:- Multithreading]

Class no:- 19

Date:- 17.11.2022

Benifits of multiple threading:→

1. Responsiveness:→

- It allows a program to continue running even if parth of it is blocked or performing leigthy operation. So it increases Responsiveness.

2. Resource sharing:→

By default thread sharing common code & common data & other resources which allows multiple task to be performed simultaneously with a single adofress space.

3. Economic:→

Creating & managing thread is much faster than performing the same for the processes.

4. Better utilization of multiprocessor Architecture:→

- Execution of multi-threaded application may be split among available processor. So it is better utilized on the multiprocessor architecture.

* Differentiate between process and thread:-

Process	Threads
- A process is an executing instance of a application	- A thread is a path of execution with in process.
- A process may contain one or more thread.	-

- Processes are heavy as they uses many resources	- Threads are light way as they uses less resources
- Different processes Generally don't share address space.	- Thread with is same process share same address space.
- process are generally independent	- Threads are generally inter dependent.
- processes are used for heavy task	- Threads are used for light task

Types of Threads :→

- There are 2 types of threads

1. User-level thread :→

User-level threads are supported above the Kernel with-out Kernel support and implemented by thread library rather than via System call of user-level.

e.g!- Multithreading in Java.

- The library provides supports for thread creation, thread Scheduling & thread management with-out the suppor of kernel.
- These threads are fast to create and manage.

adv. ✓ If kernel is single threaded then any user-level thread performing a blocking system call will cause the entire process to block.

- e.g!- posix p-threads, Win-32 threads

2. Kernel thread :→

→ Kernel threads are supported directly by the operating System.

→ The Kernel performs thread creation, scheduling & management in kernel address space.

→ Kernel threads are slower to create and manage user threads

→ If a thread perform a blocking System call, the kernel can schedule another thread for executing.

→ In a multiprocessor environment the kernel can Schedule thread on different processor.

→ example of OS that supports multithreading, Windows 2000, Windows 10, Solaris etc.

Class no:- 20
Dt:- 18.11.2022

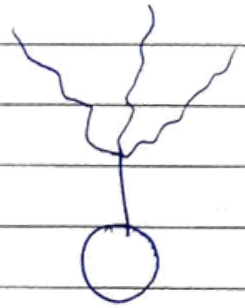
Multi Threading Model :→

- mapping of user-level thread to kernel level thread is known as Threading Model / Multi threading Model.

1- Many to One
2- One to One
3 - Many to many.

1. Many to one :→

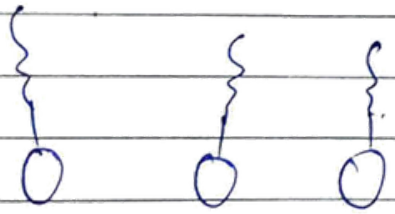
- In this model many user-level threads are mapped to one Kernel thread
- Thread management is handled by the thread library in the case of space.
- If a blocking System call is made, the entire process is blocked.
- This model doesn't allow individual process to split across multiple CPUs.



2. One to One :→

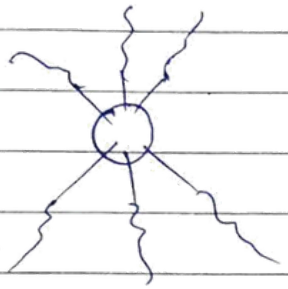
- This model creates ~~one~~ Separate Kernel thread to handle each user thread.
- ~~Block~~ More implementation of this model places a limit on how many threads can be created.

Some time is spent in managing one to one mapping



Many to many:-

- This model multiplexes any no. of user threads to an equal or smaller no. of kernel thread.
- It combines the best feature of one to one & many to one.
- users have no restriction on the no. of threads created.
- Blocking System call don't block the entire process
- process can split across multiple processor.
- One popular variation of this model is two level model.



Thread library :⇒

- It provides the programmer with API to create an manage threads.
- Implementation of thread library, there are 2 ways.

1- provide library entirely on user space with no kernel support all code, data structures exist in user space.

2. It is implemented as kernel level. The library is supported by the OS cod, data structure etc are all in kernel space. invoking a function in API results in a system call to kernel.

- The three main library are POSIX P-thread, Windows, Java.

1. P-thread :-

The thread extension of POSIX may provided as either user level or kernel level library.

2. Window :-

- Kernel level library.

3. Java :-

- It allows threads to be created & managed directly by Java Program.

→ Java API's generally implemented using the library provided by core System.

Thread creation in 2 ways :-

1. Asynchronous
2. Synchronous

Asynchronous :→ after creation of child process

When a parent thread creates a child process, the parent thread can execute simultaneously as well as the child process

Synchronous :→

After the creation child process. The parent will wait till the completion of all child process, then it resumes or starts But the child process can run parallelly.

Class no:-21

Dt:- 23.11.22

Thread Issues :→

- Issues with fork and exec
- fork has 2 version in multithreading programming.
- In the first version of fork all the threads of the parent process are duplicated in the child & in the 2nd version only the thread that invoke the fork is duplicated in the child process.

• Issues with signal handling

- Signal is used to notify a process that a particular event has occurred
- all signal has the following pattern
  1. Signal is generated by occurrance of a particular event
  2. Generated Signal is deliver to a process
  3. Once delivered, the signal must be handled.
- Signal may be recieved either Synchronously or Asynchronously.
- In Synchronously signals are deliver to the same process.
  e.g:- Signals generated for illegal memory Access or division by zero.
- In Asynchronously case. Here Signals are generated by an event external to the process.

e.g.: fetching some pressing some key.

When a multithreaded process receive a signal which thread to what thread जाएगा.

There are 4 major options

1. Deliver the signal to the thread to which the signal applies.
2. Deliver the signal to every thread in the process.
3. Deliver the signal to certain threads of the process not to all.
4. Assign a specific thread to recieve all signal the best choice may depends on the type of signal.

Thread Cancellation issues :→

- The thread that can be cancelled are called as target thread.
- cancellation of thread may occurred in 2 ways.

1. Asynchronous
2. Differed cancellation

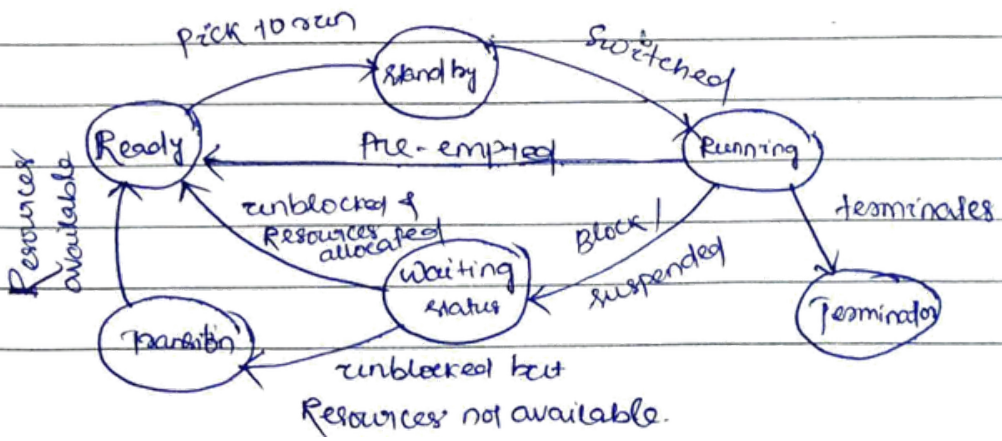
Asynchronous :→

One thread immediately terminates the target.

Differed cancellation :→

The target thread periodically check wheather it should terminate itself or not

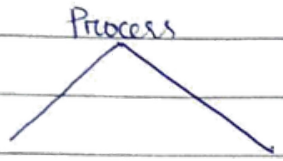
Thread States in Windows :→



Class no:-22

DT:-24.11.22

Module 3 :- Inter Process Communication :-



→ Don't affect or affected by other process

- can affect or can be affected by other process.

Why we need co-operating System?

1. Information sharing.
2. computational speedup
3. Modularity
4. convenience.

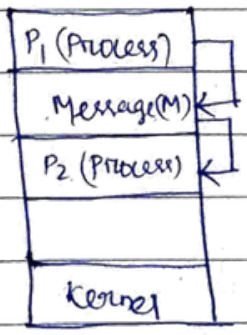
1. information sharing:-

- co-operating process can be used to share information between various processes.
- It could involve having access to the same files.
- A technique is necessary so that the processes may access the file concurrently.

IPC Model

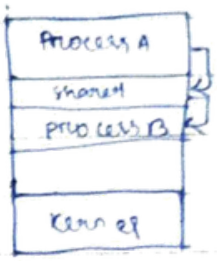
* Shared memory ✓

Process those thing to be shared in memory location, which is accessable by given processes



* Message Passing :→

- Message can be passed through kernel or between the protected process.
- e.g:- Producer - consumer problem.



[shared memory]

Class no:-23
Dt:-25.11.22

⇐: Process Synchronisation :→

Process Synchronisation is a mechanism that controls the execution of process running concurrently to ensure that consistent results are produced.

~~Race condition~~ Critical Section :→

Critical section is a section of program where a process access and update shared data during it's execution.

Race Condition :→

It is a situation where the final out put produced depends on the execution order of the instruction of the co-operating System.

Critical Section Problem :→

- It is a design a protocol that Processes can use to Synchronise their activity so as to co-operatively to share data.

- Process having 4 Critical Section.

```
Process
  {
  Entry Section
  Critical Section
  Exit
  Remainder
  }
```

Entry Section :→

- Each process must request permission to enter in to critical Section.
- The Section of code implementing this request is called as entry Section.

Critical Section :→

- Critical section is a section of program where a process access and update shared data during it's execution.

Exit Section :→

The critical Section is followed by exit section in which some activities are taken place so that other processes can enter their critical Section.

Reminder Section:→

Remaining part of the code are called as reminder Section.

Requirement for Critical Section Problem Solution:→

- Any Solution to Critical Section problem should meet the following criterias.

1. Mutual Exclusion.
2. Progress
3. Bounded Wait.

Mutual Exclusion:-

if process P_i is executing in it's Critical Section, then no other process can be executing in their Critical Section.

progress:-

If no process is executing in critical section and some process wish to enter the critical section. those processes that are not executing in their reminder section can participate in deciding which will enter it's critical section next & this selection can't be postponed indefinitely.

Bounded Wait:→

There exist a bound or limit on the no. of times, that other processes are allowed to enter their critical section after process has made a request to enter it's critical section and before that request is granted.

gmg

Peterson's Solution:→

- It is a Software based solution to critical Section Problem.
- It can solve critical Section problem for 2 process Let say P_i & P_j
- It requires to share data items,
  1. int turn
  2. boolean flag[2]

- The variable turn indicates whose turn it is to enter critical section.
- The flag array is used to indicate if the process is ready to enter the critical section or not.

```
do {
    flag[i] = true;
    turn = j;
    while ( Flag[j] && turn == j );
      Critical section
      flag[i] = false;
      Remainder section.
      while(1);
    }
```

for process P_i

Class no:- 24

Dt:- 29.11.2022

Semaphore

→ It is an integer variable(s)

→ apart from operations are

i- initialization

ii- wait / P } both are atomic → when one process is updating value of s no others are allowed to update it.

iii- Signal / V }

→ wait operations are like this

→ wait(s)

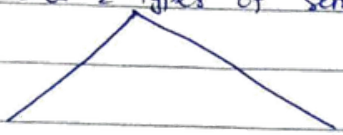
```
{
while( s<=0);
  s--;
}
```

→ signal operations are like this

→ signal(s)

```
{
 s++;
}
```

- There are 2 types of Semaphore.



Binary semaphore can take 2 values 0, 1.
- It is also called as mutex (mutual ~~exploisation~~ exclusive)
- It is used to deal with critical section problem for multiple Processes.
- do
{
wait (mutex)
CS
signal (mutex)
RS
}
While (1);
- The mutex value is initialize to 1.

E.g:-2

We have 2 instruction S1 & S2 of Process P_1 & P_2, we want that S_2 should run only after execution of S1 use semaphore to solve it.

P_1	P_2
S_1	S_2

P_1	P_2
signal (mutex)	wait (mutex)
S_1	S_2

Drawback of Semaphore :→

- It uses busy waiting which wasted a CPU cycle.
- Busy waiting means, a process is waiting for a condition to be satisfied in a tight loop without releasing the processor.
- ~~To overcome this drawback~~
- This Busy waiting is also called spin lock
- To overcome the drawback of busy waiting, the wait operation can be modified such that, when a process find that semaphore value is not positive rather than engaging in busy waiting it block itself & scheduler can select another job for execution.
- Signal operation can be modified such that when a process execute signal, the waiting process is restarted by wake up operation.

Counting Semaphore :-

- The value range over an unrestricted domain.
- It is used to control access to a given resources consisting of a finite number of instances.
- The Semaphore is initialized to the no. of resources available.
- Each process that wishes to use resources perform wait operation.
- When it realeases the resource it perform signal operation.

Class no:- 25

Dt:- 30.11.2022

Classical problems of Synchronisation :→

- Bounded Bubber
- Dining philosopher
- Reader - Writer.
- Sleeping - Barber.

Pi Pj

Producer Consumer

Semaphore empty, mutex, full

↓ ↓ ↓

n 1 0

```
do                                  do
{                                   {
Produce item nextp                  wait (full);
- - - - - - - - - -                 wait (mutex);
wait (empty);                       - - - - - - - - -
wait (mutex);                       remove an item from buffer to nextp
- - - - - - - - - - -               - - - - - - - - -
add nextp to buffer.                Signal (mutex);
- - - - - - - - -                   Signal (empty);
Signal (mutex);                     consume nextp;
Signal (full);                      - - - - - - - - - - - - - -
} while (1);                        } while (1)
```

This is the process for producer. This is the process for consumer.

Reader - Writer :->

Semaphore mutex, wrt ; readcount

↓ ↓ ↓

1 1 0

Writer

```
do
{
wait (wrt);
- - - - - - - -
Write the item
- - - - - - -
Signal (wrt.);
- - - - - -
} while(1);
```

Read

```
do
{ wait(mutex);
readcount ++;
if (readcount == 1)
wait (wrt);
Signal ( mutex);
```

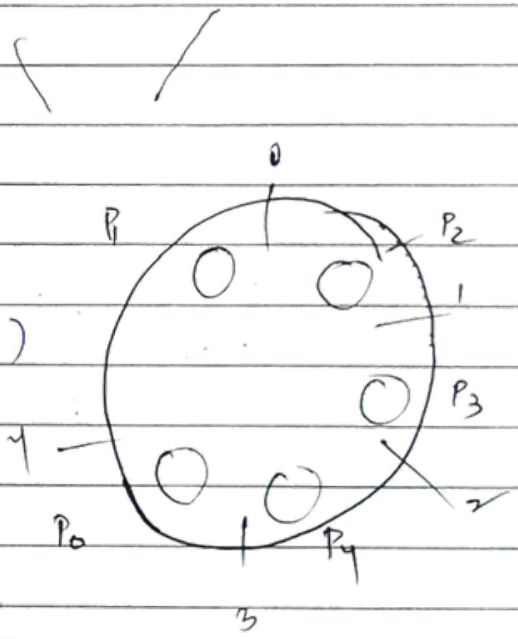
```
do reading
wait (mutex);
read count --;
if (readcount == 0)
 signal (write);
 signal (mutex
 =
} while(1);
```

Class no: 26

Dt:- 01.12.2022

Dining Philosophere :⇒

```
do
 {
  wait ( chopstic [i] )
  wait ( chopstic[i] )
  wait ( chopstic [(i+1) % 5] )
  - - - - - -
  eat
  - - - - - -
  signal ( chopstic[i] )
  signal ( chopstic [i+1]: MAX]
  - - - -
  Think
} while(1)
```



Synchronization Hardware :→

- A uni-processor System interrupts can be disabled while a process is in critical section to achieve mutual exclusion.
- Modern machine provides special atomic hardware instruction.
- One such operation is test & set

~~Tested~~

```
Test and Set ( boolean * target)
 {
   boolean rv = *target;
     *target = true;
     return(rv);
 }
```

- The structure of process that uses in Test and Set.

```
do
{
 While ( Test and set (& lock));
       // CS
     lock = false;
     // R S
} while(1);
```

Using swap we can access Synchronisation part.

```
void swap ( boolean *a , boolean * b)
 {
   boolean  * t = *a;
         * a = *b;
         * b = t;
 }
```

The structure of process that uses in swap

```
do
 {
   key = true;
```

```
while( key == true)
swap ( &lock, &key);
[CS]
lock = false;
} while(1).
```

Class no:- 27.

Dt:- 02.12.2022

Deadlock :→

- A set of processes are in deadlock, if each member of the set is waiting for the measure or event that can be released by another process in the set.

Preemptive :→

- A Pre-emptive resource is that resource which can be taken away from the process with-out any ill effect.
- e.g:- memory.

Non-Preemptive :→

- A non-Preemptive resource is one that can't be taken away from the process with-out causing any ill effect.
- e.g:- CD Drive.

Neccesary & Subficient deadlock to happen :→

- The following conditions must hold simultaneously for deadlock to happen.

1. mutual exclusion :→

   Atleast one resource must be held in non-sharable mode.

2. hold & wait :→

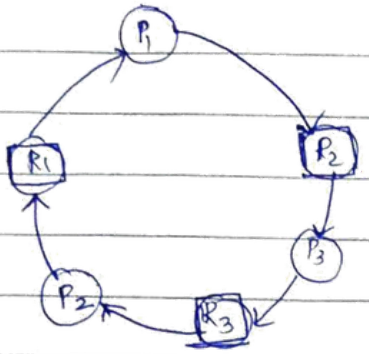
   Every process must hold a resource and requesting for other resources.

3. Non-preemptive :→

   - resources already allocated to a process can't be pre-empted.

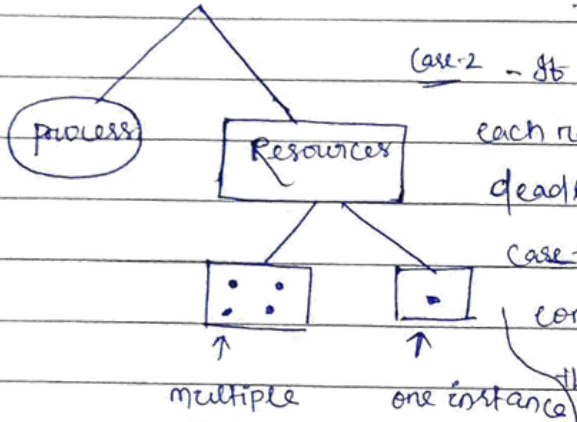
4. Circular wait :→

   The processes in the set formed a circular list where each process is waiting for a resource held by next process in the list.



- Resource allocation Graph :→ (RAG)

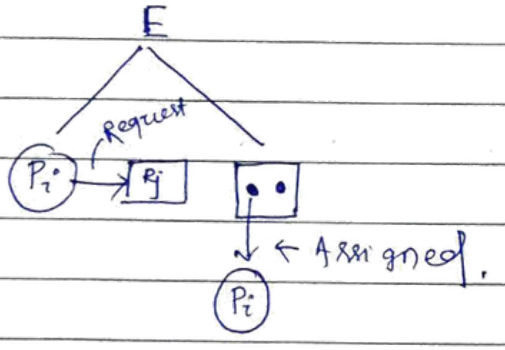
G (V, E)

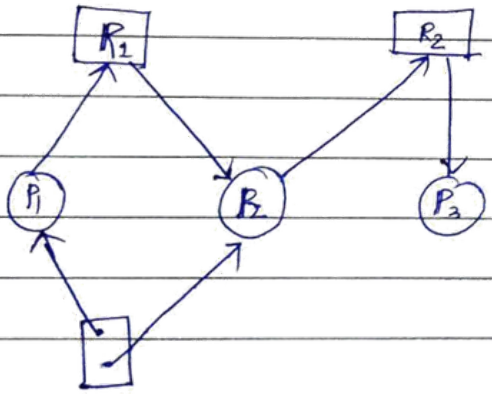


Case-1 - If the Graph contain cycle then the system is not in deadlock

Case-2 - If the Graph contain cycle & each resource has only one instance then there deadlock exists.

Case-3 If resource category contains multiple instances, then the presence of cycle indicates possibility of deadlock, but doesn't guarantee it.





- Deadlock handling :→

→ Ostrich approach
→ Deadlock prevention
→ Deadlock avoidance
→ Deadlock detection & Recovery

Class no :- 28
Dt :- 06.12.2022

1. Ostric approach :→
- Ignores the deadlock

2. Deadlock prevention :-
- In prevention approach, prevent atleast one of the four neccessary condition for deadlock from happening.
- ~~It is can be done~~.

3. Deadlock avoidance :-
- avoid deadlock by carefull resource scheduling

4. Deadlock detection :-
- detect the deadlock & take neccessary action to recover from the deadlock.

Deadlock Prevention :→
- Prevention of ~~user~~ mutual exclusion :-
- This condition is difficult to eliminate because some resources like Printer, tape Drive, are non-shcarable.
- Prevention of hold & wait :-
Approach 1 :- In this approach processes collect all the resources they need before execution. It may results in low resource utilization.
- It is also difficult to know the required resources in advance.
- This approach may also creates starvation.
Approach 2 :- Processes must give up all the resources they have and then make request for the resources they need.

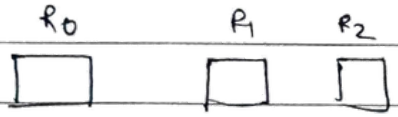
- Prevention of no preemption condition -

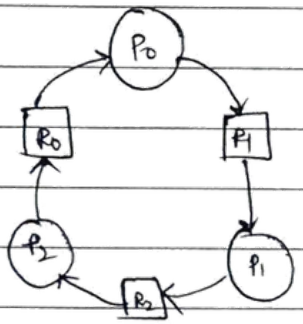
Approach-1 :- If the process that is holding some resource, request another resource that can't be allocated immediately then all the resources currently held by the process are released.

- Preempted resources are added to the list of resource for which the process is waiting
- Process will restart only when it regain the old resource & the other required resources.

Approach-2 :- The required resources are taken back from the waiting process holding it or then and given to the requesting process otherwise the requesting process should wait.

Prevention of circular wait condition :→





Class no: - 29

Dt:- 05.12.2022

→ This approach imposes a total ordering of all resource types and it requires that each process request a resource in increasing order of enumeration - i.e. If a process P_i requests for ~~par~~ Resource R_j if $P_i < R_j$ ∀ R_i the process is currently holding.

Dis-advantage:→

- Adding a new resource can set the Ordering & may require modification of numbering.
- Resource numbering may affect efficiency.
- A process may have to request a resource well before it need just because of the requirement that it must request the resource in ascending order.

Deadlock avoidance:→

- A decission is made dynamically whether the current resource allocation will if request is granted potentially lead to deadlock.
- OS will never allocate resource if it will lead to deadlock in future
- Processes must tell OS in advance. how many resource They may require.
- A resource allocation state is defined by the no. of available & allocated resource and maximum requirement of all process in a System.

Safe state:→

A state is safe if the System can allocate resources to each process in some order and still avoid deadlock

- A system is in safe state. If there exist a safe sequence.
- A sequence of processes. let say $\langle P_1, P_2 \cdots P_n \rangle$ is a safe sequence for the current allocation state if for each P_i, the resource request P_i can still make can be satisfied by a currently available resources plus the resources held by all process P_j such that $j < i$
- e.g:- $R = 12$, we have 3 processes P_0, P_1, P_2 at time t_0

The resource max need - 10 4 9

Process	Max Need Resource	currently Allocated
P_0	10	5
P_1	4	2
P_2	9	2

$\langle P_1, P_0, P_2 \rangle$

- This state is safe. because we have safe sequence. (P_1, P_0, P_2)

Class no:- 29

Date:- 09.12.2022

Example:-

MAX R = 12

unsafe state example.

Process	Maxneed	Allocation
P_0	10	5
P_1	4	2
P_2	9	(3)

(No Safe Sequence)

At t_1

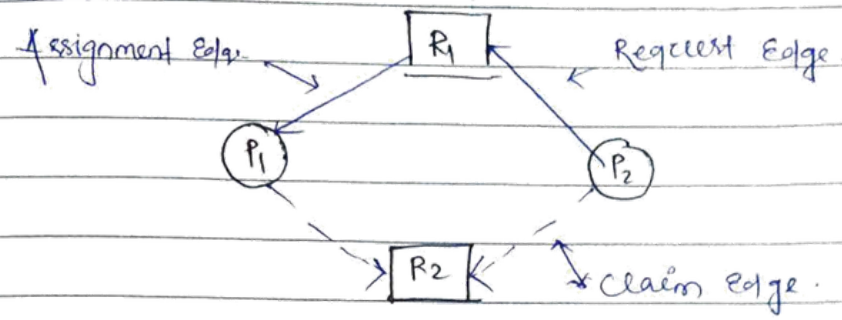
There is no state exist for safe state.

So This state is not safe.

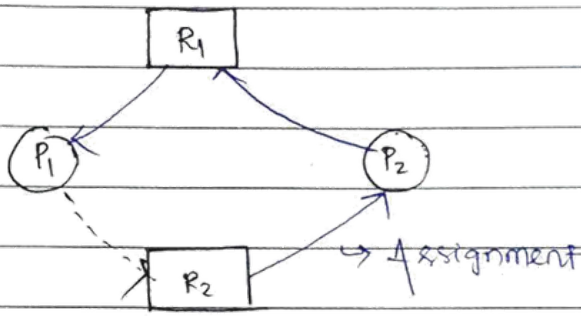
Deadlock avoidance method :→

- Deadlock avoidance using resource allocation graph.
- This algorithm is used only when one instance of each resource type is there.
- In addition to request edge and assignment edge, claim edge is introduced.
- Claim Edge P_i - - - -> R_j indicates that process P_i may request for R_j in future.
- claim edge is converted to request edge, when a ~~resor~~ process makes a request to that resource.
- When a resource is released by the process, then the assignment edge is converted to claim edge.
- Before a process start executing all of it's claim edge must appear in the resource allocation graph.
- Suppose that process P_i makes a request for ~~request~~ resource R_j, the request can be granted only if converting the request as P_i - - -> R_j and assignment as R_j - - -> P_i does not resulting in a cycle in the resource allocation graph.

Example:→



In the above example if Process P_2 makes a request for R_2, then the request will not be granted by the OS. Because it may result in a cycle in the Graph.



Class no :- 30

Dt:- 13.12.2022

Banker's Algorithm for Deadlock avoidance Multiple instance of Resource:→

- The RAG-Based algorithm can not be applied when there are multiple instances of resources
- Banker's Algorithm is designed by Dijkstra to deal with deadlock in system with multiple instances of resources types.
- The banker's algorithm has 2 parts.

i) The first part is Safety Test algorithm that checks whether current state is safe or not.

ii) The second part is resource request handling algorithm that verifies whether the requested resources, when allocated to the process, makes the state safe or not. If unsafe the request is denied otherwise granted.

In this way, banker's algorithm avoids the deadlock.

5 marks

Data structures used in Banker's Algorithm:-

- several data structures must be maintained to implement the banker's algorithm. These data structures encode the state of the resource allocation system.

Let there are:

n number of resource type in system

p numbers of process in system.

1. Total resource in a system:-

It stores the total number of resources in a system. Let us denote it as Total_Res[i] = j

It means there are j instances of Resources type R_i in the system. Thus this is a vector length n.

2. Maximum demand of a Process:-

Whenever a process enters the system, it declares it's maximum demand of resources, which is stored in the data structure Let us denote it as:

Max[i,j] = K ;

It means process P_i has a total demand of K instances of resource type R_j. Thus, this data structure is a p×n matrix, where p is the number of process, and n is the number of resource type.

3. Current allocation of instances of each type of resource:-

- It indicates the current allocation status of all resource type to various processes in the system. Let us denote it as,

Allocation[i,j] = K;

- It means process P_i is allocated K instances of resource type R_j. Thus this is also a p x n matrix.

4. No. of available resources :-

- This data structure stores the current available instances of each resource type. Let us denote it as,

Available[i] = j;

- It means, j instances of resource tye R_j are available, ~~instance~~

Thus this is a vector of length n, that is there are n number of resources type.

5. Current need of a Process :-

- This data structure indicates the current remaining resource need of each process, let us denote it as.

$$Need[i][j] = K$$

- It means Process P_i may require K more instance of resource type R_j so that it is can complete it's execution.

- Thus this is a P x n matrix. This data structure is, in fact, the difference between the maximum demand of a process and the available resource that is

$$Need[i,j] = Max[i,j] - Allocation[i,j]$$

6. Request for a Process : →

Request is the vector store the resource request for process P_i. Let us denote it as.

$$Request[i,j] = K$$

- It means, Process P_i has requested K instances of resource type R_j.

Safety Test Algorithm :-

Let work and finish be two vectors of length n and p, respectively. Safe string is an array to store the process IDs.

steps :-

1. Initialize Work = Available and finish[i] = false for i = 1 to p
2. Find an index i such that both
   i. Finish[i] = False
   ii. Need[i] ≤ Work
   if no such i exists, go to step 4.
3. Work = Work + Allocation[i]
   Finish[i] = true
   Save the process number in SafeString[]
   Go to step 2
4. if (Marked[i] == true) for all Process, then the system is in Safe state
   Print Safe String
   Otherwise, the System is not in Safe state, and is in deadlock.

Class no: 31
Dt: 14.12.2022

Q. following the current state of System.

Process	Allocation			MAX			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

No. of Process = 5
No. of Res Type = 3
No. of A = 10
B = 5
C = 7

Q1 Check whether the current state is safe or not. If safe show a safe sequence.

Q2 In this state P_1 requests (1,0,2) whether it will be granted or not.

Need (MAX − Allocation)

	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Work = 3 3 2

P_0 need can not be satisfied ✗

Q2 Consider the following snapshot of a system.

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

i) Need matrix
ii) Safe state ?
iii)

Need

	A	B	C	D
P_0	0	0	0	0
P_1	0	7	5	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	6	4	2

Work : 1 5 2 0

(ii) State safe or unsafe :-

Finish = F, F, F, F, F

Work = Available = 1 5 2 0

P_0 need can be satisfied.

So Finish = T, F, F, F, F

Work = 1 5 2 0

+ 0 0 1 2 = 1, 5, 3, 2

1 5 3 2

Safe seq $\langle P_0 \rangle$

P_1 need can't be satisfied.

P_2 need can be satisfied

So Finish = T, F, T, F, F

Work = 1 5 3 2

+ 1 3 5 4

2 8 8 6 Safe seq $\langle P_0, P_2 \rangle$

P_3 can be satisfied

So, Finish = T, F, T, T, F

Work = 2 8 8 6

0 6 3 2

2 14 11 8 Safe $\langle P_0, P_2, P_3 \rangle$

P_4 can be satisfied

So Finish = T, F, T, T, T

Work = 2 14 11 8

0 0 1 4

2 14 12 12 Safe $\langle P_0, P_2, P_3, P_4 \rangle$

P_1 can be satisfied

So Finish = T, T, T, T, T

Work = 2 14 12 12

1 0 0 0

3 14 12 12

Safe seq $\langle P_0, P_2, P_3, P_4, P_1 \rangle$

0 4 2 0 = The request of P_1

1 0 0 0 It is a valid request

1 4 2 0 because is less than equal to max

Request of P_1 ↑ Allocation of P_1 ≤ max P_1

Need

0 3 3 0

$1\ 4\ 2\ 0 \leq 1\ 7\ 5\ 0$

current request can be satisfied by current available

- Pretens that the request is granted

so new allocation = 1 4 2 0

New need will be.

	A	B	C	D
P_0	0	0	0	0
P_1	0	3	3	0
P_2	1	0	6	2
P_3	0	0	2	0
P_4	6	6	4	2

1 5 2 0

− 0 4 2 0

1 1 0 0

Finish = F, F, F, F, F

work = available = ~~1 5 2 0~~ 1 1 0 0

P_0 need can be satisfied

so Finish T, F, F, F, F

work = 1 1 0 0

1 0 0 1 2

~~1 5 3 2~~

1 1 1 2 Safe seq $\langle P_0 \rangle$

P_1 need can't be satisfied.

P_2 need can be satisfied

So finish T, F, T, F, F

work = 1 1 1 2

1 1 3 5 4

2 4 6 6 so Safe $\langle P_0, P_2 \rangle$

P_3 can be satisfied

So Finish T, F, T, T, F

Work = 2 4 6 6

+ 0 6 3 2 Safe seq $\langle P_0, P_2, P_3 \rangle$

2 10 9 8

P_4 can be satisfied

So finish T, F, T, T, T

work = 2 10 9 8

+ 0 0 1 4

2 10 10 12 Safe seq $\langle P_0, P_2, P_3, P_4 \rangle$

P_1 req can be satisfied

So Finish T, T, T, T, T

work = 2 10 10 12

+ 1 4 2 0

3 14 12 12

Safe seq $\langle P_0, P_2, P_3, P_4, P_1 \rangle$

Deadlock Detection & Recovery :→ Class no:- 32

Date:- 15.12.2022

- allow the system to enter deadlock
- detect the deadlock
- Recover from the deadlock.

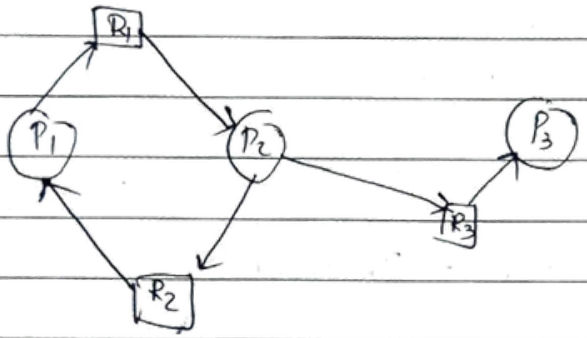
Deadlock detection :→

- only one instance of resource type → weight for graph Algo
- Multiple instances → Var. Banker's Algorithm

Wait for Graph :→

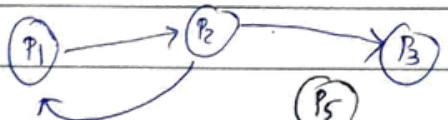
- This is applicable in a system where single instance of each resource type is available
- Weight for Graph is a variation of resource allocation Graph.
- A wait for Graph can be constructed from resource allocation graph by eliminating the resource node and collapsing the associated ages.
- And are from process from $P_i \rightarrow P_j$ indicates that P_i is waiting for a resource that process P_j is currently holding
- If the wait for graph contains cycle then it indicates deadlock
- This algo must maintain WFG & periodically search it for cycle.

e.g:-1

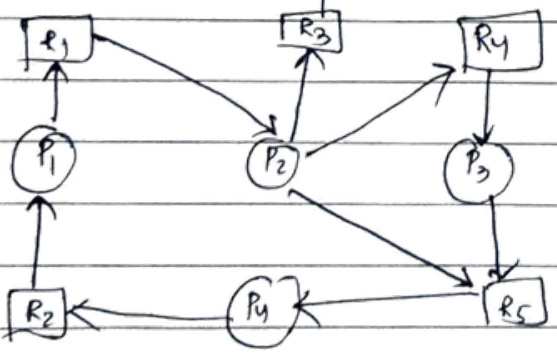


- Remove all resource nodes in between them
- If any cyclic diagram is find out then it indicates the deadlock.

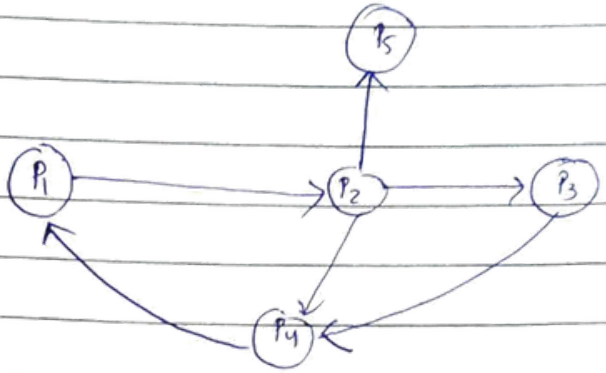
Ans:-



e.g:- 2



Ans :-



Deadlock detection using Banker's Algorithm :-

Class no:- 33
Date:- 16.12.2022

- Data structure to be used are
- available Available [i] = k
- allocation Allocation [i][j] = k
- Request Request [i][j] = k
- finish.

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

Finish = F F F F F

Work = 0 0 0

P_0 request can be satisfied.

Work = 0 0 0 So finish = T, F, F, F, F

+ 0 1 0

0 1 0

P_1 request can't be satisfied

P_2 request can be satisfied

So finish = T, F, T, F, F

Work = 0 1 0
3 0 3
3 1 3

P_3 Request can be satisfied

so finish = T, F, T, T F

Work = 3 1 3
2 1 1
5 2 4

P_4 Request can be satisfied

so finish = T, F, T, T, T

Work = 5 2 4
0 0 2
5 2 6

~~Safe S~~

P_1 Request can be satisfied

so Finish = T, T, T, T, T

Work = 5 2 6
2 0 2
7 2 8

Safe seq $\langle P_0, P_2, P_3, P_4, P_1 \rangle$

if request arises in instance 'c' as 1

so Request will be

	A	B	C	Available A B C
P_0	0	0	0	0 0 0
P_1	2	0	2	
P_2	0	0	0 → 1	
P_3	1	0	0	
P_4	0	0	2	

↑ Available me kuchh change nhi hoga.