# UNIT - II

## PROCESS MANAGEMENT

processes - Process Concept,
Process Scheduling,
Operations on Processes,
Inter Process Communications;

Threads - Overview,
· MultiCore Programming,
Multithreading Models;

· Windows 7 - Thread and SMP Management

Process Synchronization
- Critical Section Problem
· - Mutex Locks;
- Semaphores
- Monitors

CPU Scheduling and deadlocks.

# UNIT II

## PROCESS MANAGEMENT

### Processes:-

Multiple programs are loaded into memory and are executed concurrently This required firmer control and more compartmentalization of the various programs. The program that is currently executed is said to be a process. It is a unit of work in a modern time sharing system.

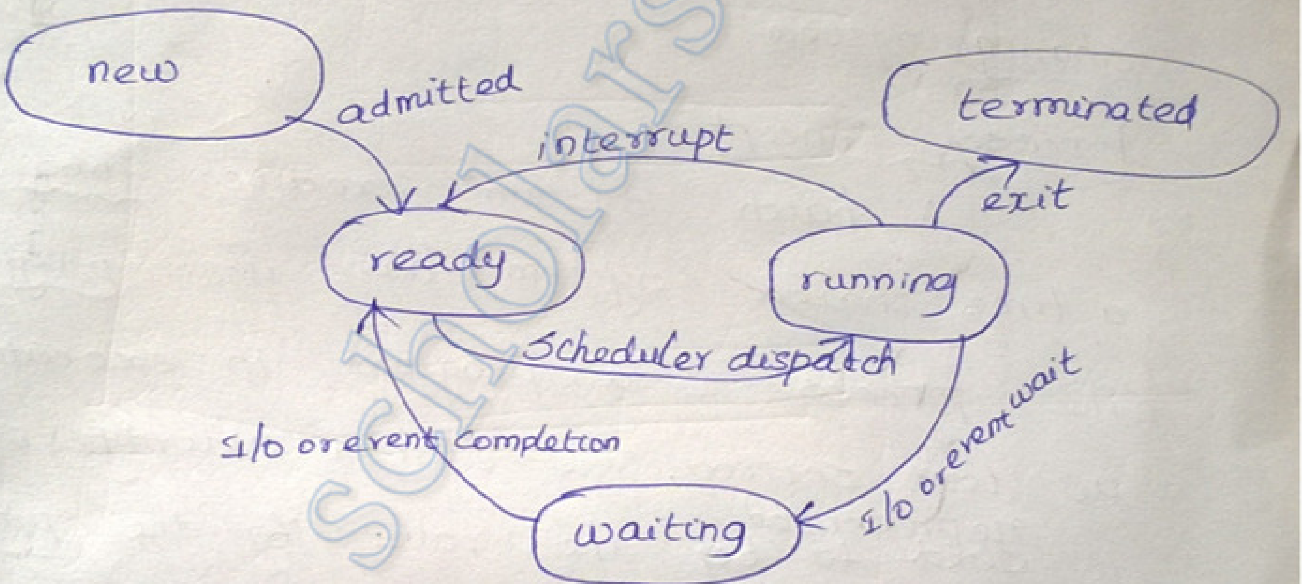
### Process Concept :-

A batch system executes jobs, whereas a time shared system has user programs or tasks. The process is a program in execution known as text section. The current activity of the process is ~~said to~~ represented be the value of the program counter and the contents of the processors registers. Stack represents the temporary data and the Data section contains global variables. Program is an passive entity whereas a process is a active entity.

## Process State:-

The state of a process is defined ~~in~~ by the current activity of the process. Each process may be in any one of the stat

- * New - The process is just created
- * Running - Instructions are executed
- * Waiting - The process is waiting for some event to occur.
- * Ready - The process is waiting to be assigned
- * Terminate - The process has finished execution



## Process Control Block:-

Each process is represented by the operating system by a process control block (PCB) or task Control block. It contains many pieces of informations like

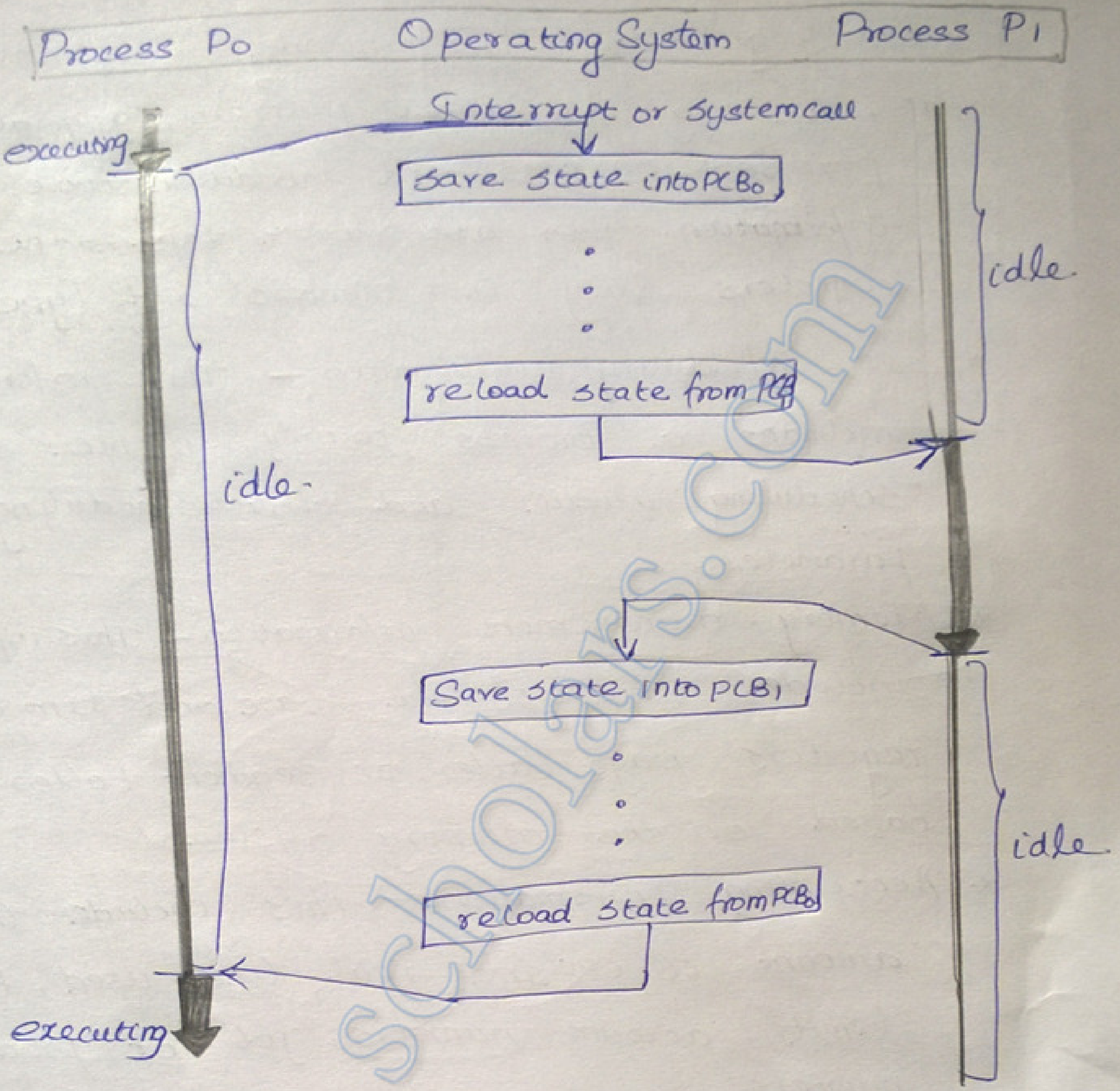
- * Process State:- The state may be new, ready, running, waiting, halted and so on.

- * Program Counter - This indicates the address of the next instruction to be executed.
- * CPU Registers - This includes accumulators, index registers, stack pointers, general purpose registers and condition code information. ~~They are various type~~. The registers vary in numbers and type.
- * CPU - scheduling Information - This information includes a process priority. pointers to scheduling queues and other scheduling parameters
- * Memory - management information - This information includes value of the base and limit registers, page tables, or segment tables based on the memory system.
- * Accounting Information :- This includes the amount of CPU and real time used, time limits, account numbers, job or process numbers.
- * I/O state Information - This includes the list of I/O devices allocated to this process, a list of open files and so on.

| | |
|---|---|
| Pointer | Process state |
| Process number | |
| Program Counter | |
| registers | |
| memory limits | |
| list of open files | |
| ... | |

Figure - Process Control block (PCB)

# CPU switch from Process to Process.



## Process Scheduling:

The objective of multiprogramming is to have some process running at all times, so as to maximize CPU utilization. The objective of the time sharing is to switch the CPU among processes so that the users can interact with each program while running. An uniprocessor has only one running process and if more process exists, the rest will wait until the CPU is free.

### a) Scheduling Queues

As the process enters the system they are put into a job queue. This queue has all the processes of the system. The process that are ready and waiting to be executed are kept on the ready queue. The ready queue pointer points to the first and final PCB in the list. The pointer of the PCB points to the next PCB in the ready queue. The list of process waiting for a particular I/O device is called a device queue. The device queue of every device is shown in the figure.

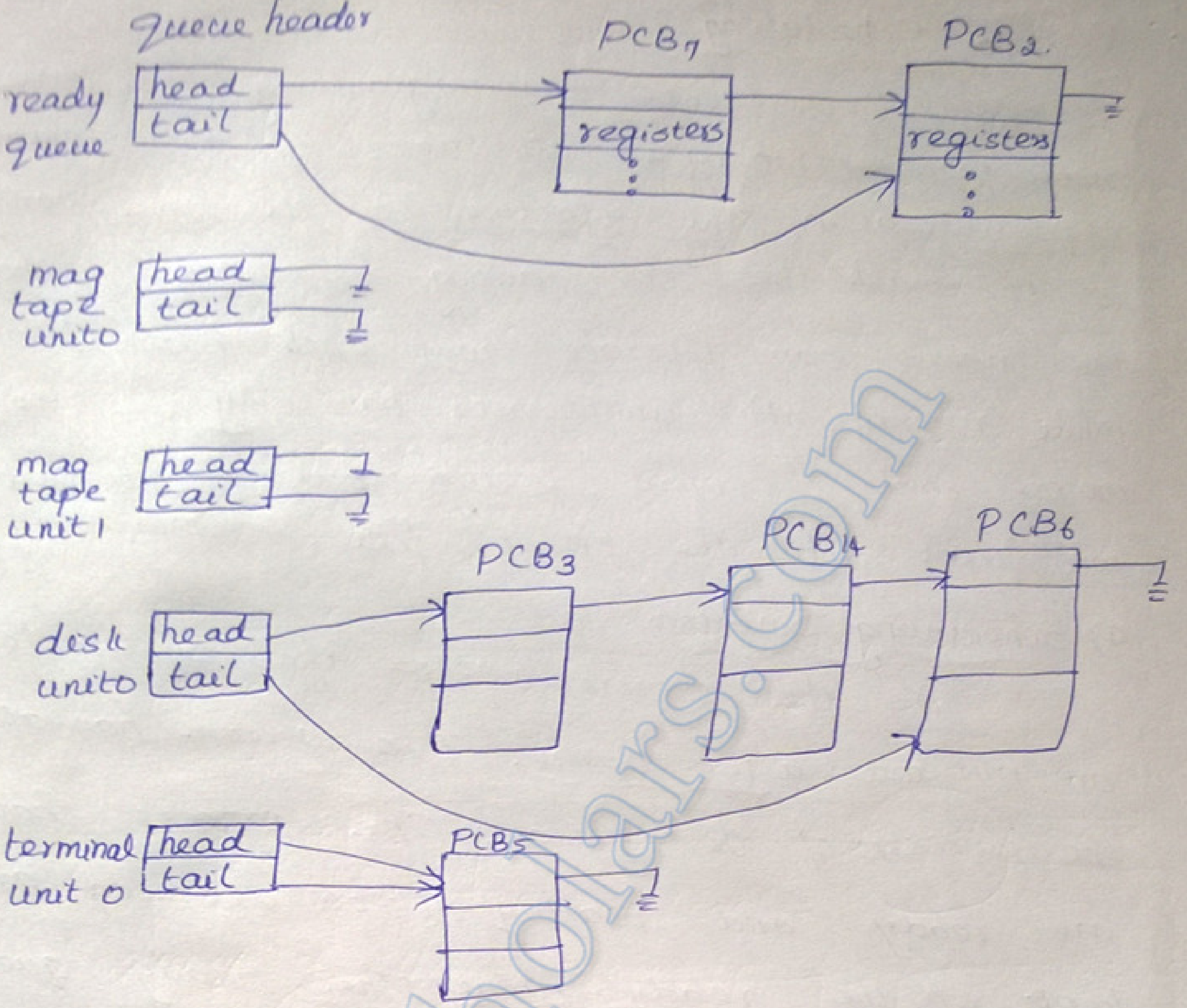


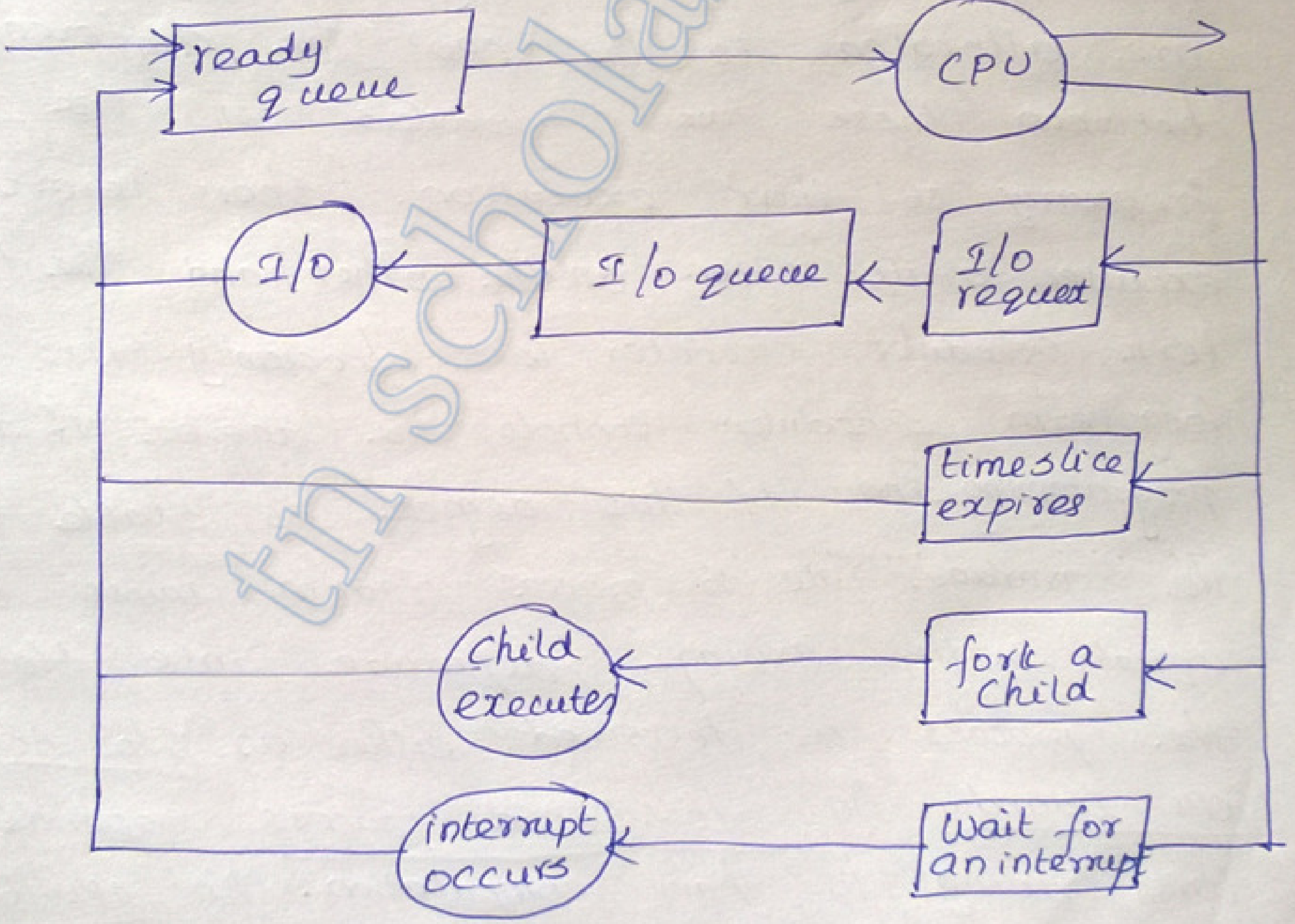
Fig: The ready queue and various I/o device queues.

Queuing diagram is used to represent the process scheduling queue is represented using a rectangular box. Circles represent the resources and the arrows represent the flow of process in the system. Two types of queue are there 1) ready queue

2) set of device queue

Any process is initially put in the ready queue. It waits until it is selected for execution or dispatched. Once it is assigned to the CPUO any one of the following event could occur

- * It can issue an I/o requst, and be placed in the I/o queue.
- * It can create a new sub process and wait for its termination
- * It can be removed from the CPU, and can be put back in the ready queue.



Queing diagram of a process scheduling

## b) Schedulers

A process migrates between various scheduling queues and the operating system selects these from the queues. This selection process is carried out by the schedulers

Many processes are spooled to a mass storage devices (disk) so that it can be executed later. Such processes are selected and loaded into the memory for execution by a long-term scheduler or job scheduler. The short term scheduler or the CPU scheduler selects the processes that are ready to execute and are allocated to the CPU. The difference between these two schedulers is the frequency of their execution. Short Term scheduler executes quite often an the other hand the long term scheduler executes less frequently. This long term scheduler controls the degree of multi programming and if thes degree is stable then the average rate of process creation must be equal to the average departure rate. Most of the process is described either as I/O bound or CPU bound. An I/O-bound process spends more time doing I/O than computation. An CPU-bound process ~~generates I/O~~ spends more time on computation than I/O bound process.

Medium term scheduler removes processes from memory and reduces the degree of multiprogramming. Swapping is a process where the process is reintroduced into memory and its execution can be continued from the point of left off. Medium is performed by the medium term scheduler and improves the process mix.

c) Context Switch

Context switching is a process of switching the CPU to another process by saving the state of the old process and later loading the saved state to a new process. The context of the process is represented in the PCP which includes values of the CPU registers process state and memory management informat. Context switches generates overhead and are highly dependent on hardware support. Context switch includes changing the pointers to the current register set.

## OPERATIONS ON PROCESSES :-

The processes in the system can be executed concurrently and must be created and deleted dynamically. Thus the operating system must provide a mechanism for process creation and termination.

### a) Process Creation :-

A process can create several new processes. The creating process is called a parent process, and the new process created is called the chidren of the process. This process can be continued forming a tree process. Every process will require resources to accomplish its task. Every subprocess will be able to obtain resources from the operating system or the parent may partition its resources among the children. When a process is created the parent process passes various physical and logical resources and initialization data to the child process. For example if the process has to display the status of a file 'F₁' on the screen of a terminal, it gets information from the parent process like 1) name of the file 'F₁'. 2) data to execute this file 3) and also the o/p device where it has to display.

when a process creates a new process, two possibilities exist interms of execution:

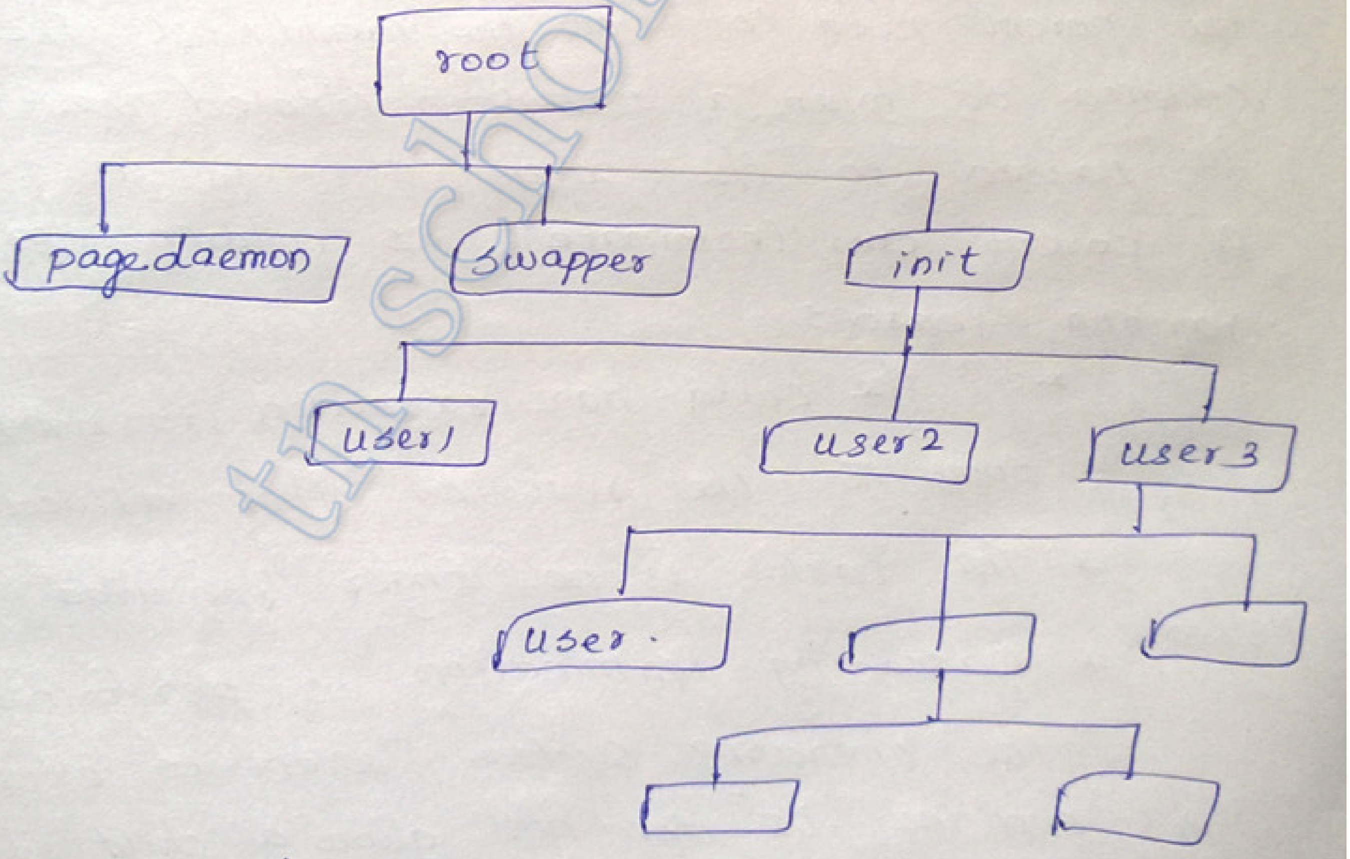
1) The parent continues to execute concurrently with its children

2) The parent waits until some or all of its children have terminated.

There are also two possibilities in terms of the address space of the new process:

1) The child process is a duplicate of the parent process

2) The child process has a program loaded into it.



Atree of processes on a typical UNIX System

b) Process Termination

A process terminates when it finishes executing its final statement and requests the operating system to delete this process using a system call exit. During this time the data is ~~de~~ returned to its parents process via a system call wait. All the ~~process~~ resources of the process are deallocated by the operating system.

Similarly a process can terminate another process via using an system call abort. This process can be performed only by the parent. The users can arbitrarily kill each others jobs. To trace this the parent need to know the identity of the children. So when a process is created the identity is passed to the parent.

A parent can terminate its children for various reasons

- * if the child has exceeded its usage of some of the resources than allocated.
- * The task is no longer required
- * Cascading termination is performed by the operating system. Parent is existing and the OS does not allow a child to continue if its parent terminates. If a parent process terminates, then all its children must also be terminated.

## Interprocess Communication:- (IPC)

This is performed to acheive a Cooperating processes to communicate with each other and synchronize their actions without sharing the same address space. example is a Chat program used on www.

a) Message passing system

This allows process to communicate with each other without the need to resort to shared data. This service operates ouside the kernel. Communication is performed by passing of messages. Thes performs two operations

a) send (message)
b) receive (message)

This message can be either fixed or variable size. Fixed size messages can be send by the system level implementation whereas variable size messages requires complex level implementation. several methods of logical implementing a link and the send/receive operations are

- * Direct or indirect communication
- * symmetric or asymmetric communication
- * Automatic or explicit buffering
- * send by copy or send by reference
- * Fixed - sized or variable - sized messages

## b) Naming

Processes can ~~ra~~ refer each other in two ways * direct Communication

* Indirect Communication

### i) Direct Communication

Each process has to explicitly name the recipient or sender of the communication.

* Send (P, message)

* Receive (Q, message)

A message is send to processP and the message is received from ProcessQ. A link is extasblished with exactly two processes and exactly one link exists between each pair of processes.

### ii) Indiret Communication

Using indirect Communication, the messages are send to and received from mailboxes or ports. Each mailbox has a unique identification In this scheme, a process can communicate with some other process via a number of different mailboxes. Two processes can comm only if they share a mailbox.

* Send (A, message)

* Receive (A, message)

A message is send ~~from~~ to mailbox A and receive a message from a mailbox A.

The communication link has the following properties

*A link is established between a pair of processes if both have a shared mailbox

* A link may be associated with more than two processes.

* A number of different links can exist between each pair of communication processes where each link corresponds to one mailbox.

A mail box can be owned by a process or operating system. If a mail box is owned by a process, then ower and the user has to be differentiated. when the process terminates the mailbox also terminates If the mailbox is owned by the operating system then it creates

* a new mailbox
* Send and receive message through the mailbox
* Delete a mailbox

c) Synchronization:

Message passing between processes by either blocking or nonblocking is said to be synchronous and asynchronous.

* Blocking send

The sending process is blocked until the messag is received by the receiving process or by the mailbox.

* Non Blocking send

The sending process sends the message and resumes operation.

* Blocking receive

The receiver blocks until a message is available.

* Nonblocking receive

The receiver retrieves either a valid message or a null.

d) Buffering

During the communication the processes reside in a temporary queue which is implemented in three ways

* Zero Capacity:- The queue has maximum length 0. This link cannot have any messages waiting into

* Bounded capacity: The queue has finite length 'n', where 'n' messages can be stored in it. If the queue is not full, the new message is placed in the queue and the sender can continue the execution without waiting.

* UnBounded Capacity: The queue has pottentially infinite length, and can hold any number of messages.

## Communication in Client-Server Systems

### Sockets:-

A socket is defined as an endpoint for Communication. A pair of socket is employed for the communicating processes. A socket consists of IP address concatenated with a port number. Sockets uses a client server architecture.

### REMOTE PROCEDURE CALLS:-

The RPC was designed as a procedure call mechanism between systems with network connections. The messages exchanged for RPC communication are well structured and are not just packets of data. The messages are addressed to an RPC daemon to a port on the remote system consisting of an identifier of the function and the parameters to be passed. The function is then executed and the output is send back to the requester in a seperate message.

A port is simply a number included at the start of a message packet. If an Remote process needs a service, it address it message to the proper port. eg if RPC attaches to a port 3027, then the RPC message is Send to port 3027

RPC allows a client to invoke a procedure on a remote host and the RPC system hides the necessary details allowing the communication to take place. This process is performed by the RPC by providing an stub on the client Side. Seperate stub exists for each remote procedure. when the client invokes a remote procedure, the RRC system calls the appropriate stub, passing it the parameters provided to the remote procedure.

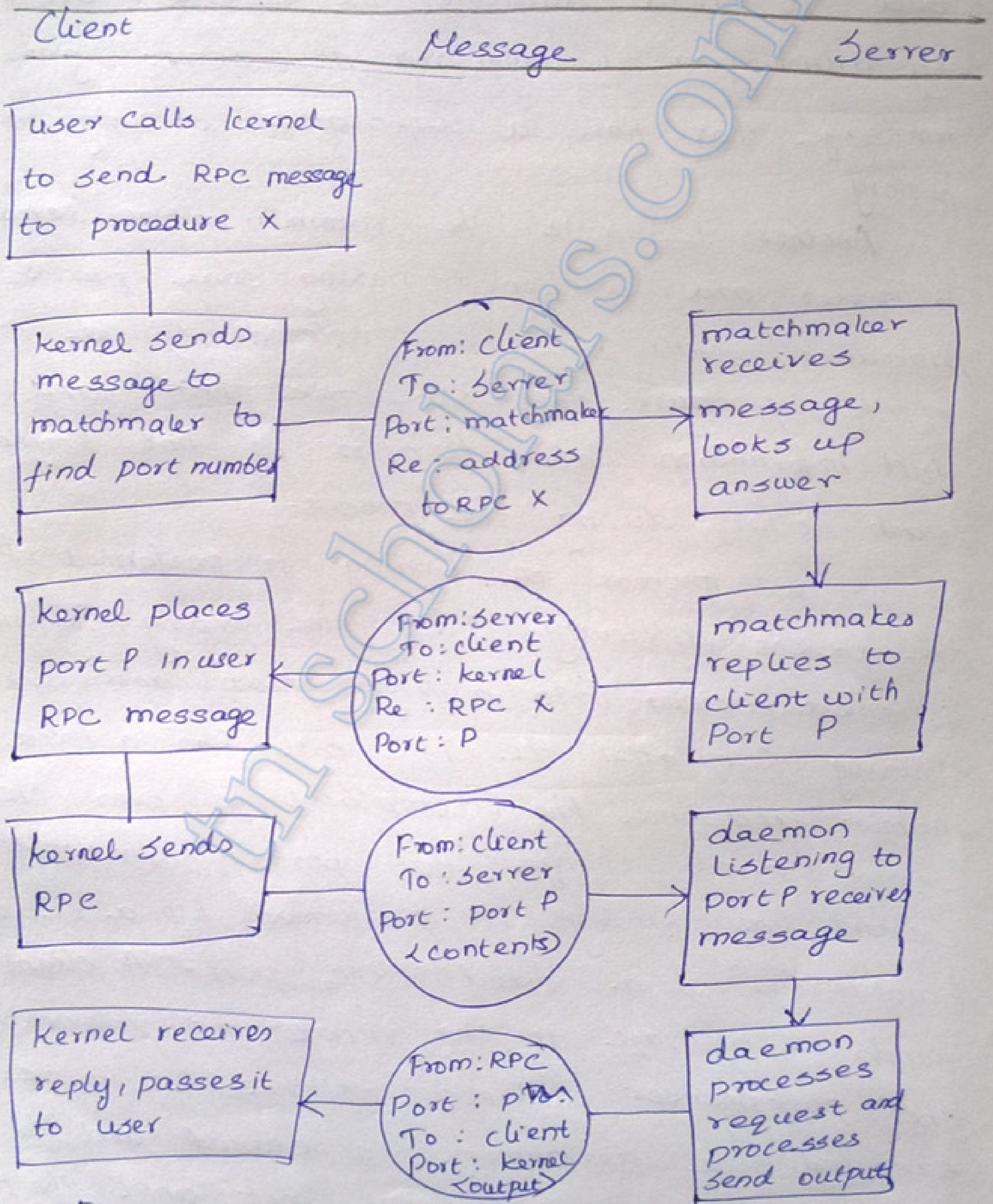
This stub locates the port on the server and marshalls the parameters. Parameters marshalling involves packaging the parameters into a form which may be transmitted over a network. The stub transmits the message to the server using message passing. Another stub on the server side receives this message and invokes the server procedure.

In order to deal with the difference in data representation, RPC systems defines an machine independent representation of data. One such representation is external data representation(XDR) On the client side the parameter marshalling Converts the data into XDR and then sends to server. Similarly on the other side the XDR dat a is unmarshalled and converted into machine dependent representation for the server.

One of the important issue is the semantic of a call. RPC can fail or can be duplicated or it can be executed more than once due to common network errors. The operating system has to identify the messages that was acted more than once. To avoid duplicates the messages are send by attaching a timestamp and the server ignores the message that have a timestamp already in the history.

Another issues is the communication between a server and a client. During some process the procedure calls name is replaced by the memory of the procedure call. ~~To solve this~~ and the full information about the other is not known to each other during the process.

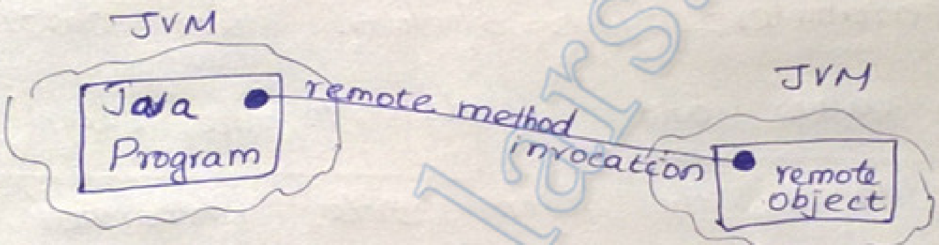
Two process of binding is performed to overcom the process difficulty. First the binding information is predetermined into a fixed port addresses. During compilation the RPC call has a fixed port number and once the compilation is over, the server cannot change the port number of the service. Second, the binding is performed by a redezvous (matchmaker) on a fixed RPC port. A client sends a message to the rendezrous requesting the port ~~number~~ address of the RPC to be executed. The port number is returned and the RPC calls may be sent to that port until the process terminates. The RPC scheme is useful in implementing a distributed file system. The messages are addressed to DFS port. Disk operations ~~like~~ may be read, write, rename, delete or status can be performed. The return messages are executed by the DFS daemon on behalf of the client.


Client
Message
Server
user calls kernel to send RPC message to procedure X
kernel sends message to matchmaker to find port number
From: client
To: server
Port: matchmaker
Re: address to RPC X
matchmaker receives message, looks up answer
kernel places port P in user RPC message
From: server
To: client
Port: kernel
Re: RPC X
Port: P
matchmaker replies to client with Port P
kernel sends RPC
From: client
To: server
Port: port P
<contents>
daemon Listening to port P receives message
kernel receives reply, passes it to user
From: RPC
Port: P
To: client
Port: kernel
<output>
daemon processes request and processes send output

Execution of a Remote Procedure Call (RPC)

## REMOTE METOHD INVOCATION (RMI)

RMI is a java feature similar to RPC. RMI allows a thread to invoke a method on a ~~mem~~ remote object. If the objects reside in a different java virtual machine (JVM), then these objects are considered to be remote. The remote objects can be in a different JVM on the same machine or in a remote host connected by a network.

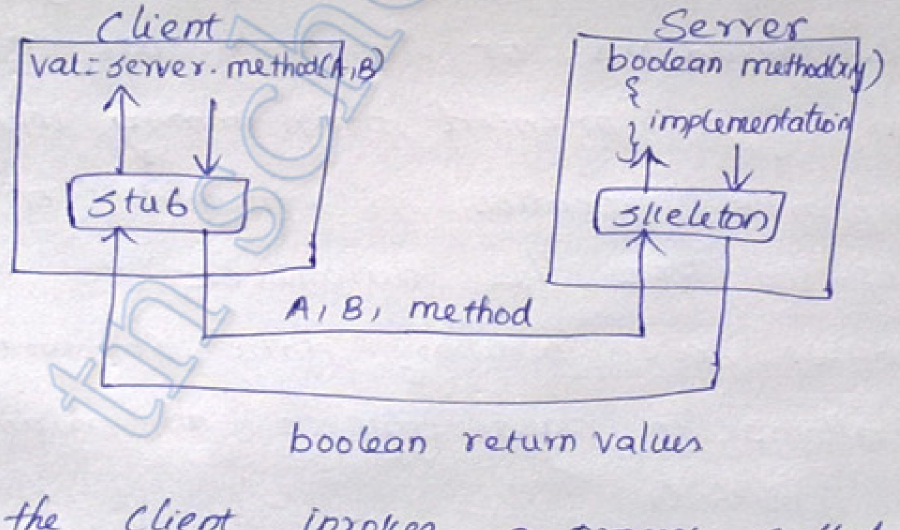


RPC supports procedural programming where only remote procedures or functions are called. RMI is an object oriented programming where where it supports invocation of methods on remote objects. In RPC the parameters to remote procedures are ordinary data structures whereas RMI allows to pass objects as parameters to remote methods.

To make the remote methods transparent to both the client and the server, RMI uses stubs and skeletons to do this.

A stub is a proxy for the remote object and it resides in the client. when a client invokes a remote method, the stub of the remote object is called. This stub creates parcel consisting of the name of the method to be invoked on the server and the marshalled parameters. This stub sends this parcel to the ~~ske~~ server where the skeleton for the remote object receives it.

The skeleton is responsible for the unmarsheling the parameters. The skeleton then marshals the return values and sends to the client side. The stub of the client side unmarshells and passes the return value to the client.



Here the client invokes a process called Method (A,B) The stub on the client side creates a parcel and marshalls the parameters and passes to the server side. The skeleton on the receiver side. accepts the parcel, unmarshals and executes the method using the parameters x and y.

The output value is returned to the client side by parcel. The stub unmarshalls and passes the return value to the client. Thus the level of stubs and skeletons are made transparent. The rules behind parameter passing are given below:-

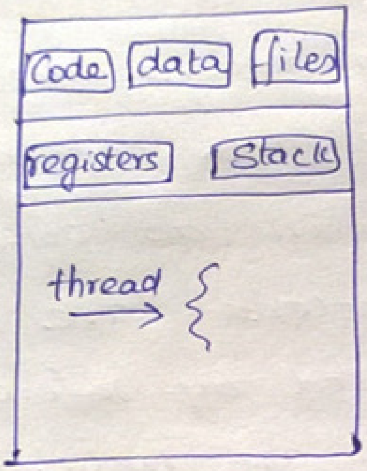
* if the marshalled parameters are local or non remote objects, they are passed by copy using a technique known as object serialization. If the parameters are remote objects, then they are passed by reference.

* If the local objects are to be passed as parameters to remote objects, they can be implemented using java.io.serializable. Object serialization allows the state of an object tobe written to a byte stream

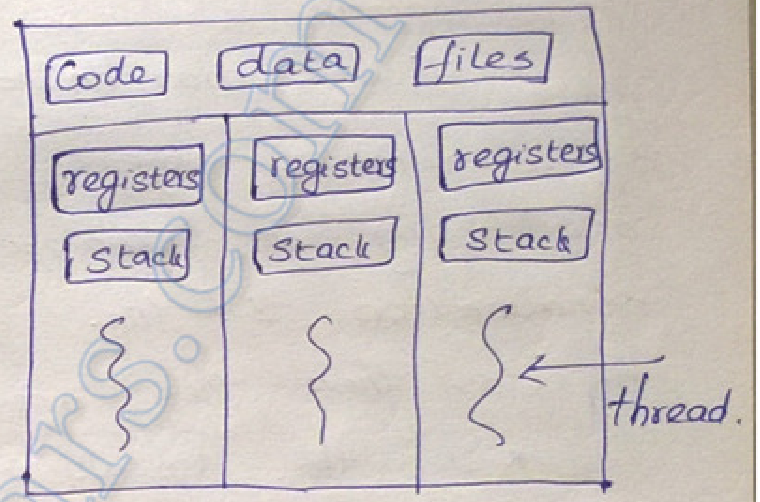
THREADS :-

Thread is a light weight process (LWP). It is a basic unit of CPU utilization. This includes a thread ID, program counter, a register set and a stack. It shares its code section, data section and other operating system resources such as open files and signals among the threads of the same process.

A heavyweight or traditional process has a single thread of control. If the process has multiple threads then it can perform more than one task at a time.



Single Thread



Multi Thread

Any single threaded process will be able to service only one client at a time. The waiting time for any request to be serviced is high. Multithreaded implements application as a seperate process with several threads of control. A web browser might display images using a thread, and at the same time another thread retreives data from the network.

Benefits of the multithread~~ing~~ed Programming are

1) Responsiveness

- This allows the program to continue running even if a part of it is blocked.

2) Resourse sharing

- Threads share the memory and the resources of the process to which they belong.

3) Economy

4) Utilization of multiprocessor architectures

- a single threaded process can run on one CPU where as multithreading runs on a multi-CPU machine thus increasing concurrency.

User and kernel Threads:-

a) User Threads:- It is supported above the kernel and are implemented by a thread library at the user level. This supports thread creation, scheduling, and management with no support from the kernel. User level threads are generally fast to create and manage.

## b) kernel Threads

This in directly supported by the operating system. This performs thread creation, scheduling and management in kernel space. kernel threads are generally slower to create and manage than are user threads.

## MULTITHREADING MODELS:-

Many systems supports both user and kernel threads termed as multithreading models.

### i) Many to One Model

This model maps many user-level threads to one kernel thread. This model manages the thread in user space but the entire process will be blocked if a thread makes a blocking system call. Since only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors. Solaris 2 uses a library called Green threads and uses this model. user level thread libraries are implemented on operating system that donot support kernel threads uses many to one model

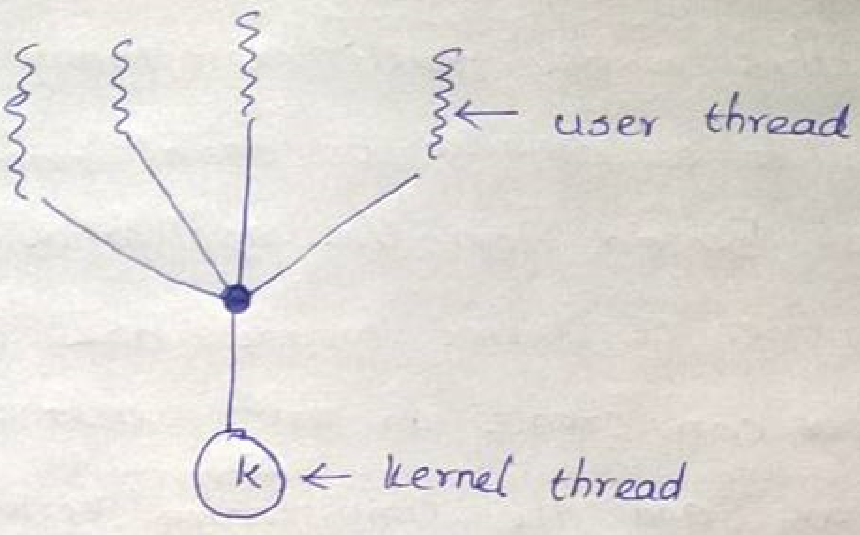


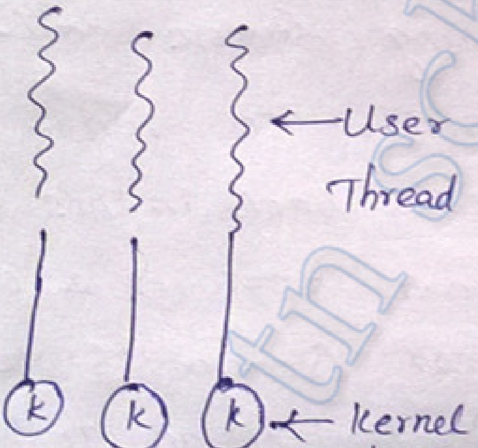
Figure - Many to one Model.

b) One to One Model

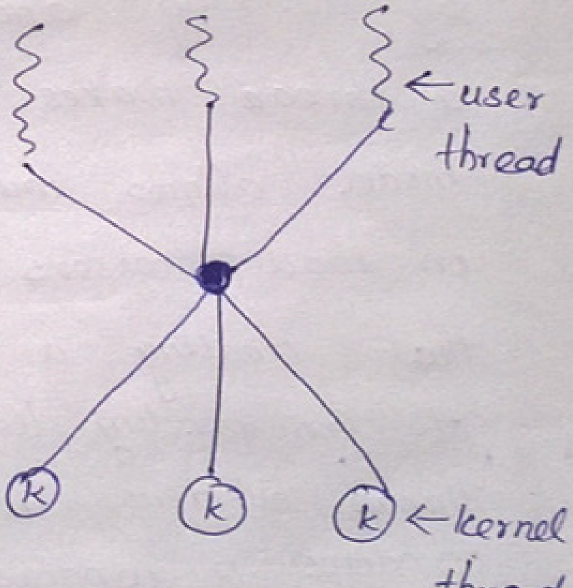
This model maps each user thread to a kernel thread. This prondes more concurrency than the many to one model by allowing another thread to run when a thread makes a blocking system call. This model allows multiple threads to run in paralle on multiprocessors. The draw back in this in that creating a user thread requires a creation of Corresponding kernel thread. Creation of the kernel Thread burdens the performance of an application. windows NT, windows 2000 and OS/2 implemet the one-to-one model.

## c) Many to Many Model

This model multiplexes many user-level threads to a smaller or equal number of kernel threads. Based on the application or machine the number of kernel threads are created. Developers can create as many user threads as necessary and the corresponding kernel threads can run in parallel on a multiprocessor. When a thread performs a blocking system call, the kernel can schedule another thread for execution. Solaris 2, IRIX, HP-UX and Tru64 UNIX supports this model.



One to One model



Many to Many model.

## PROCESS SYNCHRONIZATION:-

when a process co-operates among them this can affect or be affected by other processes. They may directly share alogical address space or share data among them. Concurrent access to shared data may result in data ~~inconsisten~~cy inconsistency.

### a) CRITICAL SECTION PROBLEM

Each process has a segment of code called critical section in to which the process may be changing common variables, updating a table, writing a file etc. When one process is executing in its critical section, ~~aer~~ no other process is allowed to execute in its critical section. The execution of critical section by the process is mutually exclusive in time. Each process requires a permission to enter its critical section. The section of code implementing this request is the entry section. The critical section is followed by an exit ~~byst~~ section. The remaining code is the remainder section.

The critical section problem satisfies the following three requirements:-

1. Mutual Exclusion:-

If a process $P_i$ is executing the critical section then no other process is allowed to execute in its critical section.

2. Progress :-

If no process is executing the critical section and the process that is not executing in their remainder section can enter their critical sections.

3. Bounded Waiting:-

there exists a bound on the number of times that a process is allowed to enter a critical sections after a process has requested to enter the critical section and before the request is granted.

```
do
  {
      Entry Section

          Critical Section

      Exit Section

          Remainder Section

  } while (1)
```

Structure of a typical Process $P_i$

A Solution is presented for the critical section problem for the Two process Solutions.

The processes are numbered $P_0$ and $P_1$. when ~~Representing~~ $P_i$, and $P_j$ ~~for~~ represents two different process where $j = 1-i$.

## Algorithm - 1

The first approach uses a common integer initialized as 0 or 1. If the variable turn==i then process $P_i$ is allowed to execute the critical section. when turn==0 then $P_0$ can use the critical section. Even if $P_1$ is ready and $P_0$ is in remainder section, $P_1$ cannot enter the critical Section if the variable turn ==0.

```
do
  {
   while (turn != i);
   Critical Section
   turn = j;
   remainder section
  } while (1);
```

Structure of the Process $P_i$

## Algorithm 2 :-

The problem in the above algorithm is it does not retain the sufficient information about the state of each process. The information it stores is about the process that entered the critical section. To overcome this problem the variable is replaced as

boolean flag[2];

The value of the array is initialized as false and if the value of flag(i) is true it indicates that Pi is ready to enter the critical section. Process Pi sets the flag[i] to be true, signaling that it is ready to enter its critical section. Pi checks to verify that process Pj is not also ready to enter its critical section. If Pj is ready, then Pi would wait until Pj had indicated that it no longer needed to be in the critical section. At this point Pi would enter the critical section.
On exciting the critical section, Pi would set flag[i] to be false allowing the other process enter the critical section.

Structure of Process Pi

```
do
  {
    flag[i] = true;
    while (flag[j]);

    Critical section

    flag[i] = false;

    remainder section
  } while (1);
```

Algorithm 3:

By combining the algorithm 1 and 2, a correct solution to the critical section where all the requirements are met. The processes share two variables

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

To enter the critical region the Process Pi set the flag[i] to be true and sets turn to the value j, therby allowing others also to enter the critical section. If two process requests at the same time, then both turn will be set to i & j both at the same time. When the first one writes the turn value the second process overwrites the first This resultant value of the turn decides which process to enter its critical Section.

This method *preserves the mutual exclusion
*Process requirement is satisfied
*Bounded waiting requirement is met.

## Structure of the process Pi

```
do {
    flag[i] = true;
    turn = j;
    while (flag[j] && turn == j);

    Critical Section

    flag[i] = false

    remainder section
} while (1);
```

# SEMAPHORES

To solve the critical section problem a synchronization tool called semaphore is used. Semaphore 'S' is an integer variable, accessed through two standard atomic operations: wait and signal. These operations are originally termed as P (for wait) and V (for signal). When one process modifies the semaphore value, no other process can simultaneously modify the same semaphore value.

Semaphore can deal with n-process critical-section problem. The n processes share a semaphore - mutex (standing for mutual exclusion) initialized to 1. Semaphores can be used to solve various synchronization problems.

Consider two concurrently running processes

P1 with statement S1
and
P2 with statement S2

Suppose if S2 has to be executed after S1 has completed, then this can be implemented by letting P1 and P2 share a common semaphore synch, initialized to 0 and by inserting the statements in process P1 and

```
S1;
Signal (synch);
do
  {
    wait (mutex);
    Critical section
    signal (mutex);
    remainder section
  } while (1);
```

the statements

```
wait (synch);
S2;
```

in process P2. Because synch is initialized to0, P2 will execute $S_2$ only after $P_1$ has invoked signal (synch), which is after $S_1$.

b) Deadlocks and Starvation:-

when semaphore is implemented, with some waiting queue result in a situation where two or more processes are waiting indefinitely due to one of the waiting processes. Such state is reached said to be deadlocked.

Starvation is a situation where processes wait indefinitely within the ~~same~~ semephore.

b)binary Semaphore:-

A binary semaphore is a semaphore with an integer value that can range only between 0 and 1.

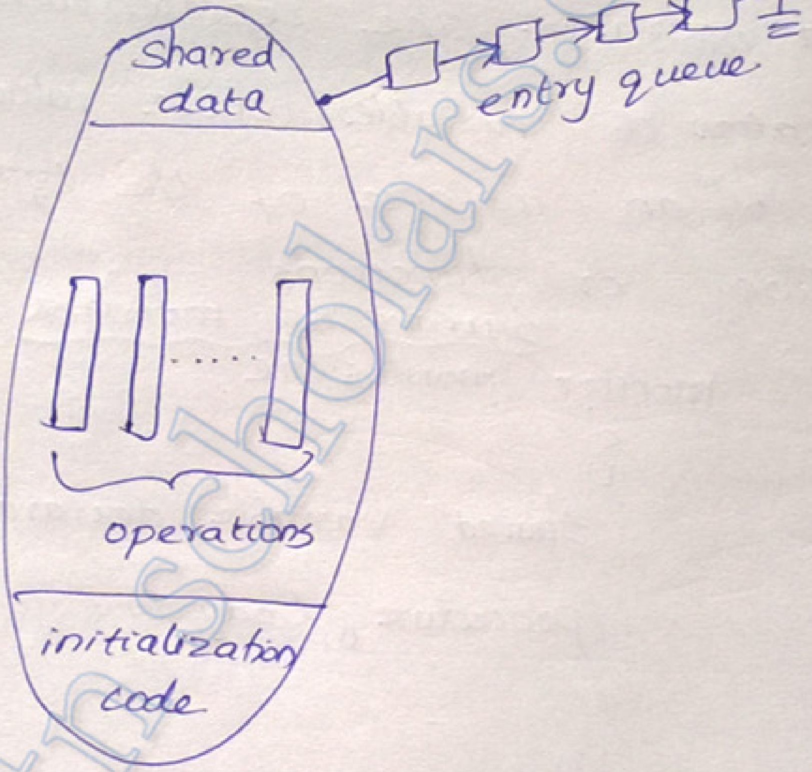
## MONITORS

Monitor is defined by a set of programmer-defined operators. The representation consists of declaration of variables, whose values defines the state of an instance of the type, bodies of procedures or functions.

### SYNTAX OF MONITOR

```
monitor monitorname
    {
      Shared variable declarations
      procedure body P1 ( . . . ) {
        - - - -
        - - - -
      }
      Procedure body P2( . . . ) {
        - - - -
        - - - -
      }
      Procedure body Pn ( . . . ) {
        - - - - -.
      }
      {
        initialization code
      }
    }
```

The representation of monitor type cannot be used directly by the various processes. The procedure defined within a monitor can access only those variables declared locally within the monitors and its formal parameters. Similarly the local variables of a monitor can be accessed by only the local procedures. Only one process at a time canbe active with in the monitor



schematic view of a monitor

In order to develop a powerful ~~cor~~ monitor construct few additional synchronization mechanisms are also defined. These mechanisms are provided by the conditional construct

Condition x, y ;

The operations that can be invoked on a condition variable are wait and signal.

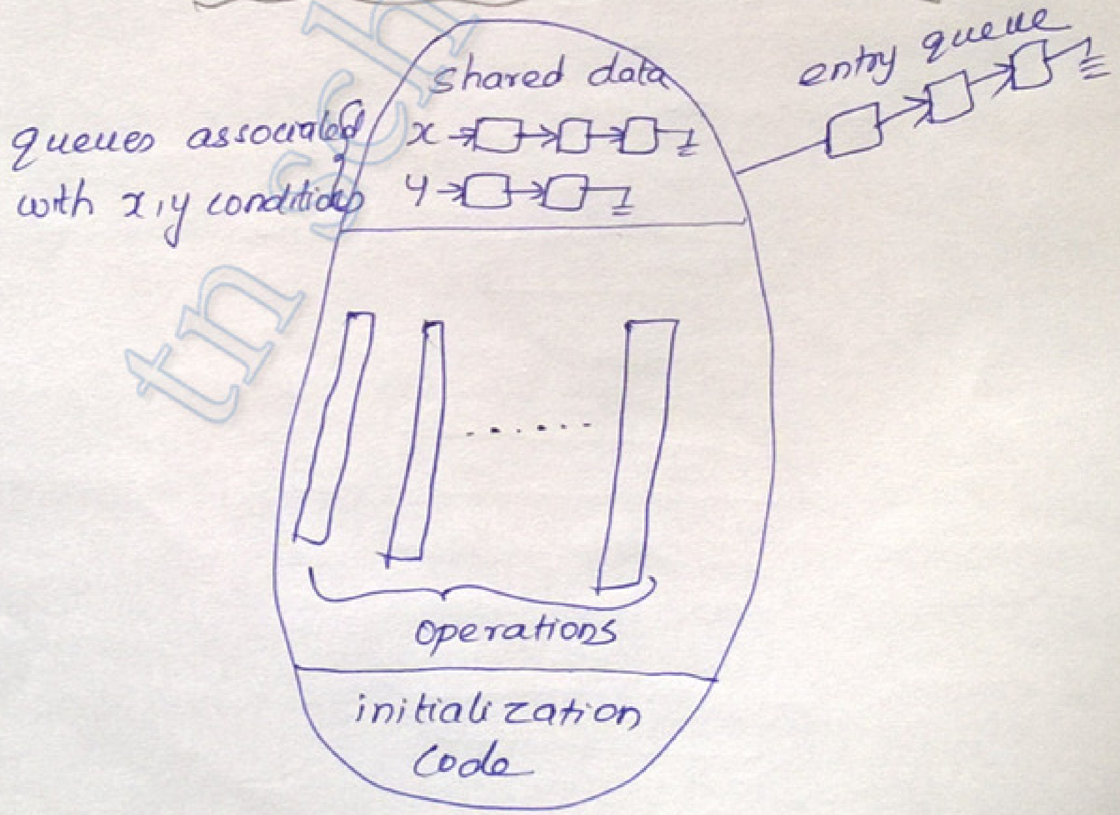
x. wait()

means the process invoking this operation is suspended until another process invokes

x. signal().

This x. signal() operation resumes exactly one suspended process. If no ~~operation~~ process is suspended then this ~~bas~~ operation has no effect.

monitor with condition variables



If x.signal() is invoked by a process P, and if there exists a suspended process Q associated with conditionx. If the suspended process Q is allowed to resume its execution then P has to wait. otherwise both P & Q will be active simultaneously within the monitor

Two possibilities exists in this case

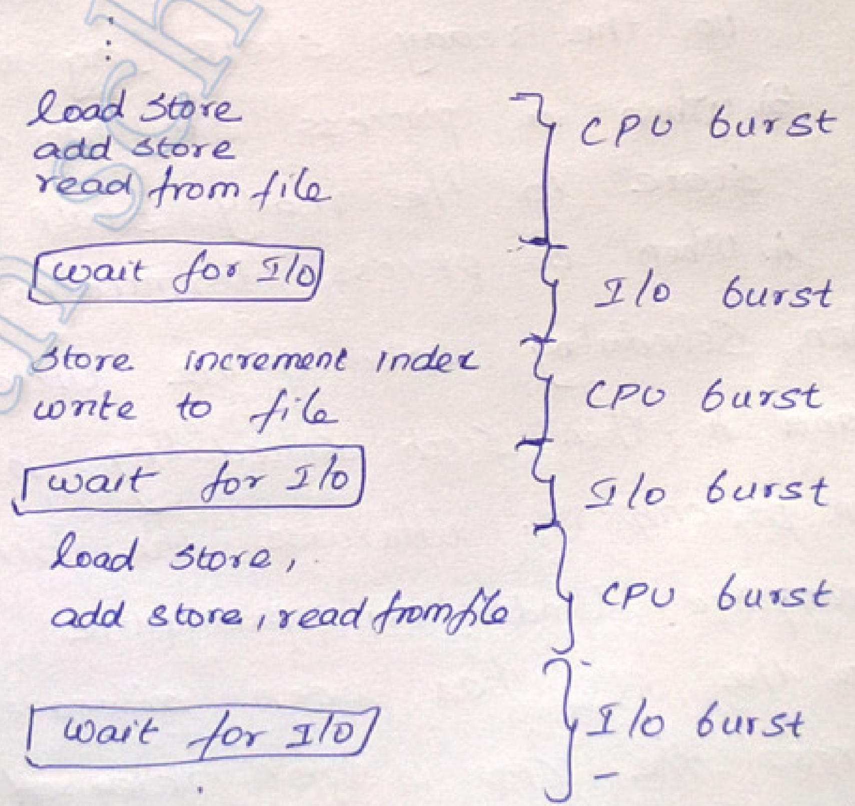
1. P either waits until Q leaves the monitor or waits for another condition
2. Q either waits until P leaves the monitor or waits for another condition

## CPU SCHEDULING

The objective of multiprogramming is to have some process running at all times, inorder to maximize CPU utilization. Scheduling is a fundamental operating-system function.

a) CPU - I/O Burst cycle

Process execution consists of a cycle of CPU execution and I/O wait. Processes alternate between these two states. Every process execution begins with a CPU burst. This in followed by an I/O burst, then another CPU burst then another I/O burst, and so on. The last CPU burst will end with a system terminate request to terminate execution.



Alternating sequence of CPU and I/O bursts

b) CPU Scheduler

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is done by CPU scheduler. The scheduler selects one of the ~~ready~~ process in memory that is ready for execution and allocates the CPU to this process.

c) Preemptive Scheduling

CPU ~~Preemptive~~ scheduling takes place under the following criteria

1) when a process switches from readystate to the waiting state (eg I/O request)
2) when a process switches from running state to the ready state (eg when interrupt occurs)
3) when a process switches from the waiting state to the ready state (eg Completion of I/O)
4) when a process terminates.

when scheduling takes place under the circumstances 1 and 4, then such scheduling is said to be non preemptive, otherwise the scheduling is preemptive. Under non preemptive scheduling once the CPU has been allocated with a process the CPU is not released until it terminates or switched to waiting state

d) dispatcher

dispatcher is a module that gives the control of the CPU to the selected process by the short term scheduler or CPU scheduler. This Involves

- * Switching context
- * Switching to user mode
- * jumping to the proper location in the user program to restart the program.

SCHEDULING CRITERIA

1) CPU Utilization :-

CPU utilization may range from 0 to 100%. In real System it ranges from 40% to 90%.

2) Through Put :-

It is the number of processes completed per time unit. for long process it is 1 process per hour and for short process it is 10 processes per second

3) Turn around time :-

The time ~~taken~~ interval between the time of Submission of a process to the time of Completion is called turn around time.

4. Waiting Time

It is the sum of period spent waiting in the ready queue

5) Response time

The time interval ~~fo~~ between the submission of a request until the first response is called response time.

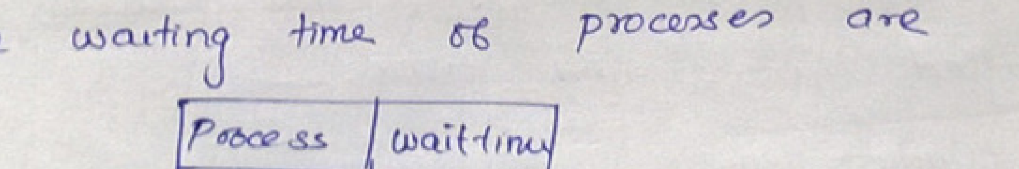
SCHEDULING ALGORITHMS:-

A) First Come First Served Scheduling

The process that request the CPU first is allocated with the CPU first. This is ~~paroot~~ performed using FIFO queue. As the process enters the ready queue, it is linked to the tail of the queue. whe the CPU is free the process in the head of the queue is allocated with the CPU.

| Process | Burst Time (millisec) | |
|---|---|---|
| P1 | 24 | order of arrival of processes P1, P2, P3 |
| P2 | 3 | |
| P3 | 3 | |

The results of Grantt Chart:-



The waiting time of processes are

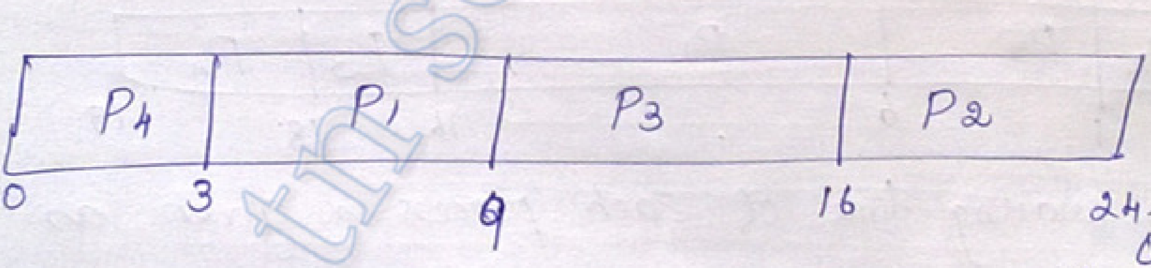
| Process | wait time |
|---|---|
| P1 | 0 |
| P2 | 24 |
| P3 | 27 |

The average waiting time is $= \frac{(0+24+27)}{3}$ =17 millisec

b) Shortest job first Scheduling

When the CPU is available, then it is assigned to the process that has the smallest burst time. The scheduling is performed based on the length of the CPU burt. If two has the same time then FCFS is used to break the tie.

| Process | Burst Time |
|---|---|
| $P_1$ | 6 |
| $P_2$ | 8 |
| $P_3$ | 7 |
| $P_4$ | 3 |



The grant chartt is derived based on the length of the burst time.

Average waiting time is given as

| Process | Wait time |
|---|---|
| $P_1$ | 3 |
| $P_2$ | 16 |
| $P_3$ | 9 |
| $P_4$ | 0 |

∴ average waiting time $= \frac{3+16+9+0}{4}$

= 10.25 millsec

## 3) Priority Scheduling

A priority is associated with each process, an the CPU is allocated to the process with highest priority.

| Process | Burst Time | Priority |
|---|---|---|
| $P_1$ | 10 | 3 |
| $P_2$ | 1 | 1 |
| $P_3$ | 2 | 4 |
| $P_4$ | 1 | 5 |
| $P_5$ | 5 | 2 |

based on the priority the grantt chart shows

| $P_2$ | $P_5$ | $P_1$ | $P_3$ | $P_4$ | |
|---|---|---|---|---|---|
| 0 | 1 | 6 | 16 | 18 | 19 |

The waiting time of each process is given as

| Process | Wait time |
|---|---|
| $P_1$ | 6 |
| $P_2$ | 0 |
| $P_3$ | 16 |
| $P_4$ | 18 |
| $P_5$ | 1 |

∴ average waiting time =

$$= \frac{6+0+16+18+1}{5}$$

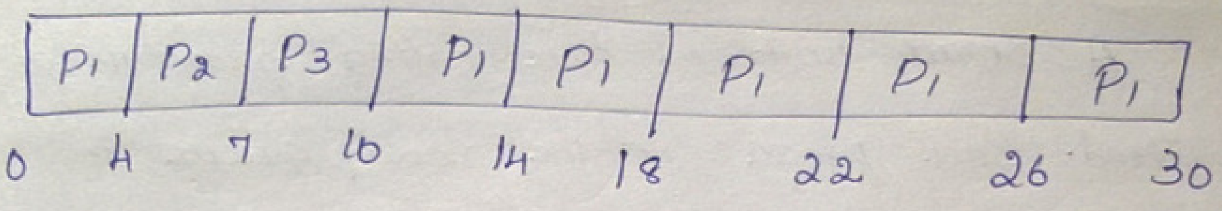
= 8·2 milliseconds

## 4) Round Robin Scheduling:-

A small unit of time called quantum is selected. Each process in the ready queue is allocated with the CPU for this time quantum. The ready queue is selected as a circular queue. The process will release the CPU when it performs I/O or it finishes executing before the time quantum is expired. Next process is selected ~~from~~ by the scheduler from the ready queue and is allocated with the CPU.

| Process | Burst Time |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

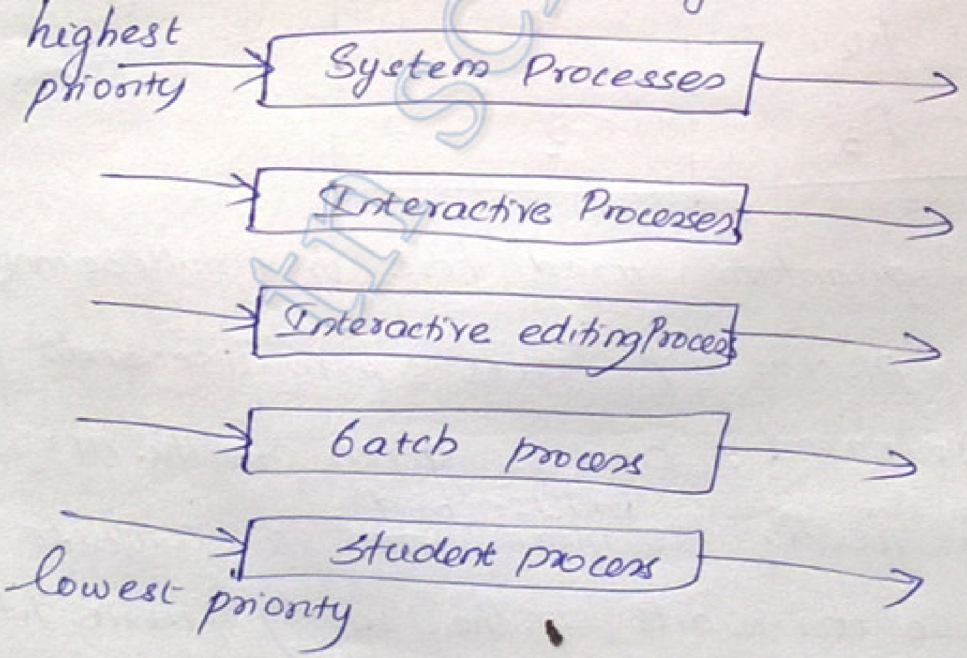
The time quantum used here is 4 milliseconds. First the process $P_1$ gets 4 milliseconds then CPU gets released and is given to the next process. Process $P_1$ needs 20 millisecond more ~~time~~ to complete the process, so it waits in the ready queue. Mean while CPU is allocated to $P_2$ and then to $P_3$. Then once again to the Process $P_1$. The schedule is shown as



The average waiting time is $\frac{17}{3} = 5.6$ milli seconds

5) Multilevel Queue Scheduling

This scheduling partitions the ready queue into several queues as shown in figure. The processes are permanently assigned to one queue based on memory size, priority or type. Each queue has its own scheduling algorithm. seperate queues can be used for foreground and background processes. The foreground queue might be scheduled by an RR algorithm while background queue can use FCFS algorithm.



Multilevel queue Scheduling

6) Multilevel Feedback Queue Scheduling

This is similar to multilevel queue scheduling, exept that the processes are allowed to more between queues. If the process uses too much CPU time then it is moved to low priority queue. If a process is waiting for a long time in a low priority queue then it is moved to high priority queue.

DEADLOCKS

If a process is requesting for a resource and if the resource is not available, then the process enters into wait state. This process cannot change to any other state because the requested resource is held by another waiting process. This situation is said to be deadlocks.

Dead lock Characterization

I) Necessary Conditions - a deadlock may occur if the following condition hold simultaneously in the system

a) Mutual exclusion

resources must be allocated to processes at any time in an exclusive manner and not on a shared basis for a deadlock to be possible. If another process requests that resource the requesting process must be delayed until the resource has been released.

2) Hold and wait:-

A process must be holding at least one resource and waiting to acqure additional resources that are currently being held by other processes.

3. No premption:

Resources cannot be preempted, ie the resource can be released only voluntarly by the process holding it, after that process has completed its task.

4) Circular wait

A set of waiting process ($P_1$, $P_2$ ... $P_n$) must exist such that $P_0$ is waiting for a resource held by $P_1$, $P_1$ is waiting for a resource held by $P_2$ ... $P_{n-1}$ is waiting for a resource held by $P_n$ and $P_n$ waiting for a resource held by $P_0$ is said to be circular wait

## Resource - Allocation Graph:-

Deadlocks can be described more precisely in terms of a directed graph called a system resource - allocation graph. Graph consists of a set of vertices V and a set of Edges E. The set of vertices V is partitioned into two different types of nodes. $P = \{P_1, P_2, \cdots P_n\}$ is a set consisting of all the active processes in the system, and $R = \{R_1, R_2 \cdot\cdot R_m\}$ is a set consisting of all resources types in the system.

A directed edge from $P_i$ to resource type $R_j$ is denoted by $P_i \rightarrow R_j$ this refers to the process $P_i$ requesting an instance of resource type $R_j$ and is currently waiting for that resource. A directed edge from resource type $R_j$ to $P_i$ is denoted as $R_j \rightarrow P_i$ , it refers to the instance of resource is allocated to process $P_i$. A directed edge $P_i \rightarrow R_j$ is called request edge and the directed edge $R_j \rightarrow P_i$ is called an assigment edge.

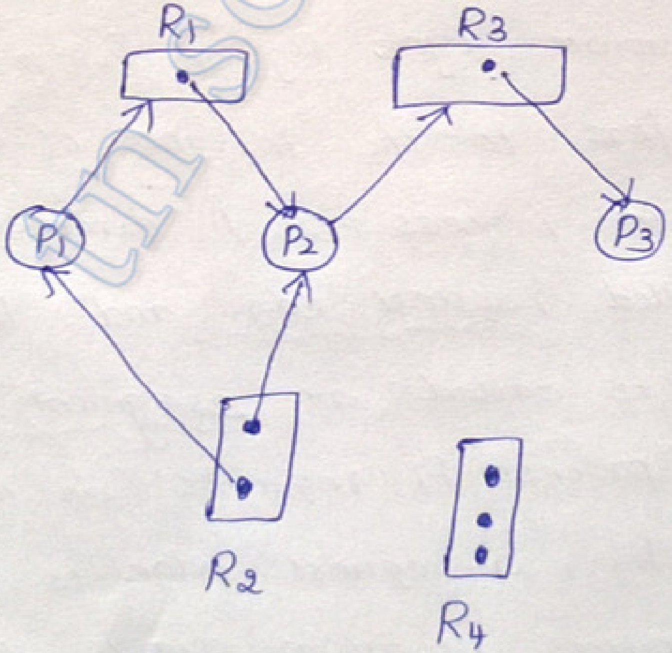
When a process $P_i$ requests an instance of resource type $R_j$ , a request edge is inserted in to the resource - allocation graph. when this request can be fulfilled then the request edge is transformed to an assignment edge. when the process no longer needs the access to the resource then it releases the resource and the assignment edge is deleted.

The sets P, R and E:

* $P = \{P_1, P_2, P_3\}$
* $R = \{R_1, R_2, R_3, R_4\}$
* $E = \{P_1 \to R_1, P_2 \to R_3, R_1 \to P_2, R_2 \to P_2, R_2 \to P_1, R_3 \to P_3\}$

Resource instances:-

* one instance of resource type $R_1$
* Two instance of resource type $R_2$
* One instance of resource ~~of~~ type $R_3$
* Three instances of resource type $R_4$



Resource allocation graph

* Process States:-

- Process $P_1$ is holding an instance of resource type $R_2$, and is waiting for an instance of resource type $R_1$.

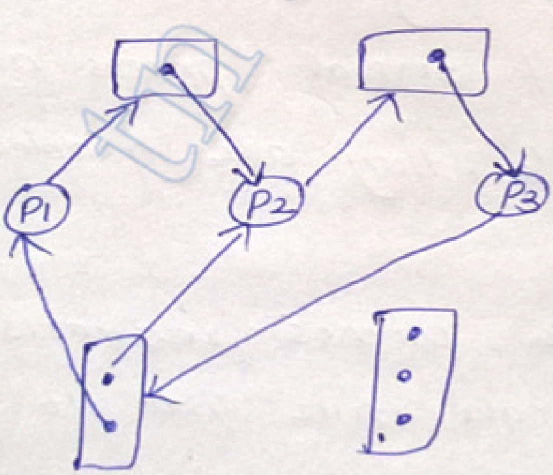
* Process $P_2$ is holding an instance of $R_1$ and $R_2$ and is waiting for an instance of resource type $R_3$.

* Process $P_3$ is holding an instance of $R_3$

If there is no cycles, then no process in the system is deadlocked. If the graph contains a cycle, then deadlock exists. If Request edge $P_3 \rightarrow R_2$ is added to the graph then there exists deadlocks with minimal cycles exist in the system

$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow R_1$

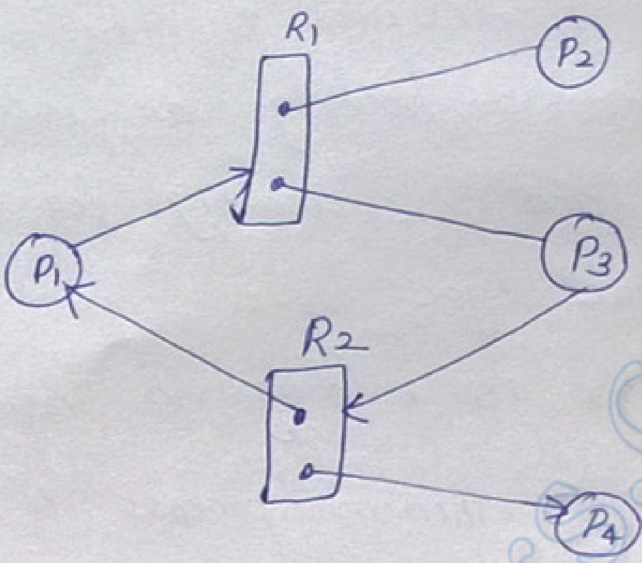
$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$



resource allocation graph with deadlock.

Similarly a resource allocation graph can be formed ~~but no~~ with cycles but no dead lock

$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$



Resource allocation graph with deadlocks

## Deadlock Prevention

The occurance of the deadloc can be prevented by

a) Mutual Exclusion

This condition must hold for nonsharable resources. A printer cannot be simultaneously shared by several processes. Sharable resources do not require mutually exclusive access eg Read only files. If several process attempts to read from the file, they are granted permission

b) Hold and Wait

whenever a process requests a resource then it has to make sure that it should not hold any resources.

One protocol that can be used is it requires each process to request and gets allocated with all its resources before it begins execution. To implement this system calls are used to request resources for a process to precede all other System Calls

An alternate protocol allows a process to request resources only when the process has no resources with it. The two drawbacks in this protocol is resource utilization may be low and Starvation is also possible

c) No Preemption:-

The third necessary condition is that there should be no preemption of resources that is already allocated. To ensure this if a process is holding some resources and requests another which cannot be allocated immediately, then all the resources that is currently been used are released implicitly. The preempted resources are added to the list of resources for which the process is waiting. The process will be restarted only when it can regain its old resources as well as the new ones that it is requesting

Alternatively if a process requests some resources, first it has to check whether they are available, if so allocate them. If not available, then it has to check whether this resources is available with a process that is waiting for another resource, if so preempt them and allocate them to the requesting process. If the resource is not avalable then it has to wait.

d) Circular Wait

One way to ensure that this condition does not hold is by imposing a total ordering of all resource types and requires that each process requests resources in an increasing order of enumeration.

Let R = {R1, R2, - - Rm) be set of resources and each resources are allocated with a unique integer so that it can be Compared among two resources. If the Set of resources R includes tape drives, disk drives, and printers and if F may be

F (tape drive) = 1

F (disk drive) = 5

F (printer) = 12

·Each process can request resources only in an increasing order of enumeration. If a process requests an resources then it has to check F(Rj) > F(Ri). A process that wants to use tape drive and printer then it has to request tape drive and then printer.

Alternatively whenever a process requests an resource type Rj, and it has released Ri such that F(Ri) ≥ F(Rj).

## DEADLOCK AVOIDANCE :-

deadlock avoidance is to require additional information about how resources are to be requested. Each process has to declare the number of ~~requ~~ resources needed. This priori information helps to construct an algorithm so that the process will never enter into deadlock This is said to be deadlock avoidance approach.

### a) Safe State

A System is in Safe State if the system can allocate resources to each process in some order and avoid deadlock.

c) BANKERS ALGORITHM:

When a new process enters the system it must declare the maximum number of instances of each resource times that it may need. This number should not exceed the total number of resources in the system. when a user requests set of resources, the system must determine whelt the allocation of these resources will leave the syste in safe state. If it will, the resources are allocate otherwise the process must wait until some other process release enough resources.

following data structure in needed

* Available → vector of length m indicating the number of available resources

* Max - maximum demand of each process.

* Allocation - no of resources currently allocated

* Need : remaining resource need of each process

c) BANKERS ALGORITHM:

When a new process enters the system it must declare the maximum number of instances of each resource times that it may need. This number should not exceed the total number of resources in the system. when a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in safe state. If it will, the resources are allocated otherwise the process must wait until some other process release enough resources.

following data structure in needed

* Available → vector of length m indicating the number of available resources

* Max - maximum demand of each process.

* Allocation - no of resources currently allocated

* Need : remaining resource need of each process