Page No.
Date: / /

Unit : 2

Processes and Threads.

Processes and Programs :-

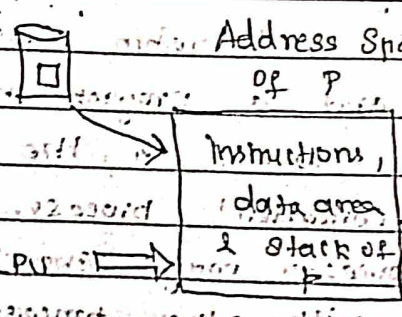
A program is a passive entity that does not perform any actions by itself; it has to be executed to the actions it calls for are to take place.

A process is an execution of a program. It actually performs the actions specified in a program. An Operating System shares the CPU among processes.

What is a Process?

For example: Consider a program P contains declarations of a file info and a variable item and statements that read values from info. Use them to perform some calculations and print a result before coming to a halt.

To realize the execution of P, the OS allocates memory to accomodate P's address space, allocates a printer to print its results, set up an arrangement through which P can access file info. and schedules P for execution. The CPU is shown as a lightly shaded box because it is not always executing instructions of P — the OS shares the CPU between the execution of P and execution of other programs.




```
Program P
file info;
int item;
Open (info, "read");
while not
    end-of-file(info)
    read (info, item);
```

Page No.
Date:

Process :-

An execution of a program using resources allocated to it.

When a user initiates execution of a program, the OS creates a new process and assigns a unique id to it. It allocates some resources to the process, sufficient memory to accomodate the address space of the program, and some devices such as keyboard and a monitor.

Accordingly, a process comprises six components (id, Code, data, stack, resources, CPU state)

id is the unique id assigned by the OS

Code is the data used in the execution of the program including data from files

Stack contains parameters of functions and procedures

Resources is the set of resources allocated by the OS.

CPU state is composed of content of the PSW

*) Relationship between Processes and Programs

Two kinds of relationship that can exist between processes and programs. A one to one relationship exists when a single execution of a sequential program is in program progress.

A many to one relationship exists when between many processes and a program in two cases

Prorosses that coexist in the system at some time are called concurrent process.

Concurrent processes may share their code, data and resources with other processes

Child Process :-

The kernel initiates an execution of a program by creating a process for it. We call it as primary process for the program execution. The primary process may make system calls to create other processes - these processes become its child processes, and primary process become their parent. A child process may itself can create other processes and so on.

The parent - child relationship between these processes can be represented in the form of a process tree, which has a primary process as its root.

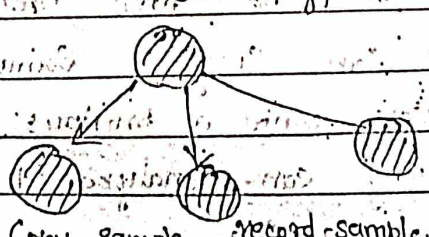
Benefits of child process :-

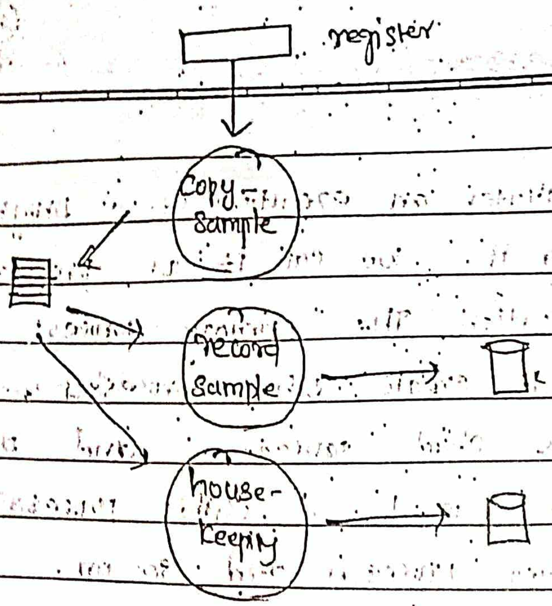
Computation Speedup :- It may reduce the duration (i.e) running time of the application

Priority for Critical functions :- A child process that performs a critical function may be assigned a high priority

Guarding a parent process's against errors :- The kernel aborts a child process if an error arises during its operation.

eg:-





The real-time data logging application receives data samples from a satellite at the rate of 500 samples per second and stores them in a file.

The primary process of the application which we will call the data-logger process, has to perform the following three functions.

1) Copy the sample from the special register into memory
2) Copy the sample from memory into a file.
3) Perform some analysis of a sample and record its results into another file used for future processing.

It creates three child processes named copy-sample, record-sample, and housekeeping leading to the process tree.

Operation of the three process can overlap : Copy-Sample can copy a sample into buffer-area, record-sample can write a previous sample into the file, while housekeeping can analyze it and writes its results into the other file.

1) Creating a child process and assign priority to it
2) Terminating a child process
3) Determining the status of a child process.
4) Sharing, Communication, and Synchronization b...

Page No.
Date: / /

Implementing Processes :-

The kernel is activated when an event, which is a situation that requires the kernel's attention, leads to either a hardware interrupt or a system call.

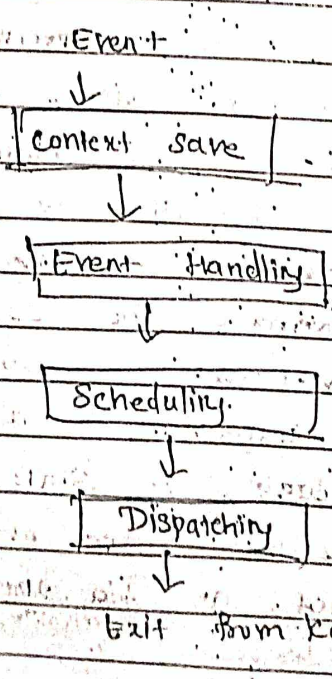
The kernel now performs four fundamental functions to control operation of processes,

1) Context Save :- Saving CPU state and information concerning resources of the process whose operation is interrupted

2) Event Handling :- Analyzing the condition that led to an interrupt (or) the request by a process that led to a system call, and taking appropriate action.

3) Scheduling :- Selecting the process to be executed next on the CPU

4) Dispatching :- Resume the operation of the process from where it left the execution to access other resources.



Page No.
Date: / /

Process State and State Transitions :-

Process State :-

The indicator that describes the nature of the current activity of a process.

The kernel uses process states to simplify its own functioning, so the number of proces states and their names vary across OS's.

Most operating system uses the four fundamental states.

The fundamental process states are Running, Blocked, Ready, Terminated.

A conventional computer system contains only one CPU and so at most one process can be in the running state. There can be any number of process in the blocked, ready and terminated states.

Running :- A CPU is currently executing instructions in the process code.

Blocked :- The process has to wait until resource request made by it is granted or it wishes to wait until a specific event occurs.

Ready :- The process wishes to use the CPU to continue its operation.

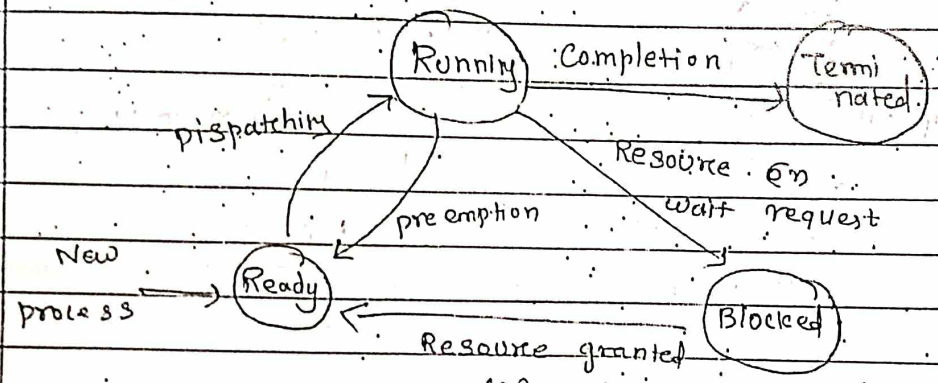
Terminated :- The operation of the process (i.e) the execution of the program is completed normally the OS has aborted it.

Process state Transitions :-

A state transition for a process Pi is a change in its state. A state Transition is caused by the occurrence of some event such as the start (or) end of an I/O operation.

when the event occurs, the kernel changes the state of an affected process.

When a process P_i in the running state makes an I/O request, its state has to be changed to blocked until its I/O operation completes. At the end of the I/O operation P_i's state is changed from blocked to ready because it now wishes to use the CPU.



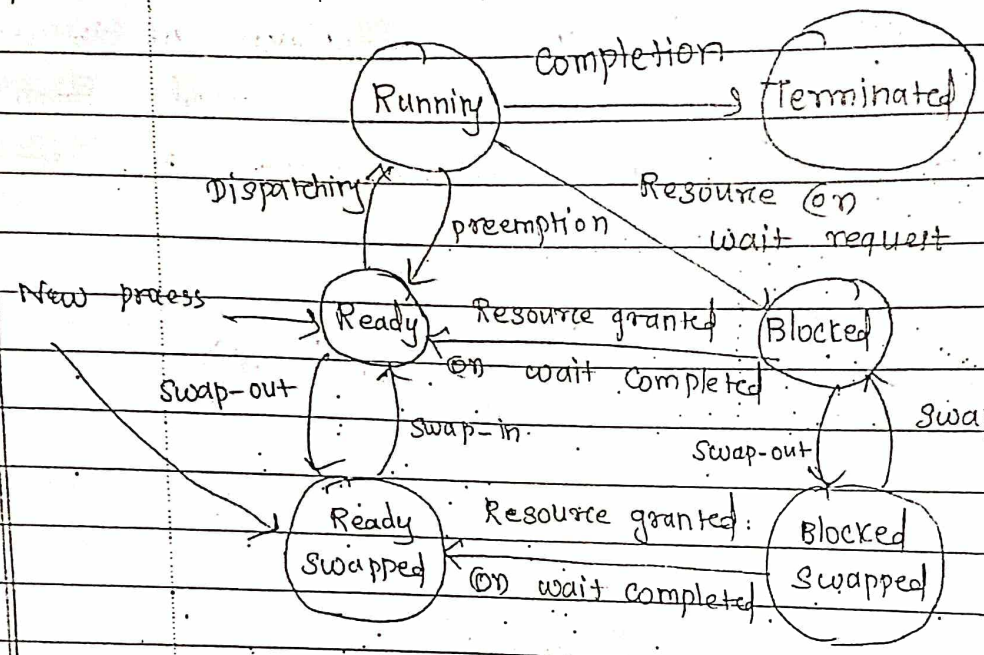
Process State Transitions in a Time-Sharing System

Time	Event	Remark	P_1	P_2
0		P_1 is scheduled	running	ready
10	P_1 is preempted	P_2 is scheduled	ready	running
20	P_2 is preempted	P_1 is scheduled	running	ready
25	P_1 starts I/O	P_2 is "	blocked	running
35	P_2 is preempted		blocked	ready
		P_2 is scheduled	blocked	running
45	P_2 starts I/O		blocked	blocked

Suspended Process :-

Consider a process that was in the ready or blocked state when it got swapped out of memory. The process needs to be swapped back into memory

before it can resume its activity. Hence it is no longer in the ready or blocked state, the kernel must define a new state for it. We call such a process a suspended process.



*) Process Context and the Process Control Block :-

The process context consists of the following :

1) Address space of the process :-

The code, data, and stack components of the process.

2) Memory allocation information :

Information concerning memory areas allocated to a process.

3) Status of file processing activities :-

Information about files being used, such as current positions in the files.

4) Process interaction information :-

Information necessary to control interaction of the process with other processes, ids of parent and child processes.

5) Resource information :- Information concerning resources allocated to the process

6) Miscellaneous information :- Miscellaneous information needed for operation of a process.

Process Control Blocks (PCB) :-

The process control block (PCB) of a process contains three kinds of information concerning the process. - identification information such as the process id, id of its parent process and id of the user who created it

PCB field :-

1) Process id :- The Unique id assigned to the process at its creation

2) Parent, child ids :- These ids are used for process Synchronization

3) Priority :- The priority is typically a numeric value. A process is assigned a priority at its creation. The Kernel may change the priority dynamically depending on the nature of the process

4) Process State :- The current state of the process

5) PSW :- This is a Snapshot (i.e) an image of the PSW

6) GPRs :- Contents of the general purpose registers when the process last got blocked (or) was preempted.

7) Event Information :- For a process in the blocked state, this field contains information concerning the event for which the process is waiting

8) Signal information :- Information concerning location of signal handler.

9) PCB pointer :- This field is used to form a list of PCB's for scheduling purposes.

*) Event Handling :-

The following events occur during the operation of an OS.

1) Process creation event :- A new process is created.
2) Process termination event :- A process completes its operation
3) Timer event : The timer interrupt occurs
4) Resource request event :- Process makes a resource request
5) Resource release event :- A process releases a ~~Ere~~ resource
6) I/o initiation request event :-
7) I/O Completion event :-
8) Message Send event :-
9) Message receive event :-
10) Signal Send event :-
11) Signal receive event :-
12) A program interrupt :-
13) A hardware malfunction event :-

Event Control Block (ECB) :-

When an event occurs, the kernel must find the process whose state is affected by it.

For eg :- When an I/o Completion interrupt occurs, the kernel must identify the process awaiting its completion. It can achieve by Searching the event information field of the PCBs of all processes. This Search is expensive, so OS use various schemes to speed it up.

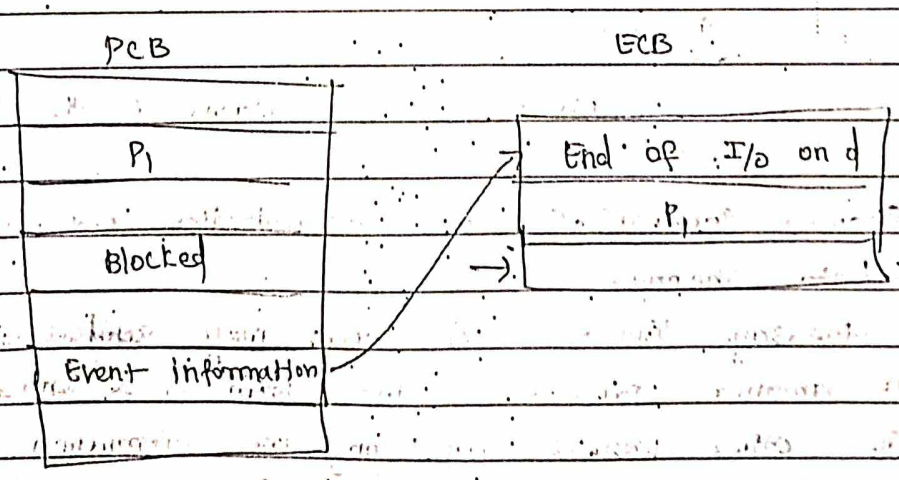
ECB contains three fields. The event description

field describes an event, and the process id field contains the id of the process awaiting the event.

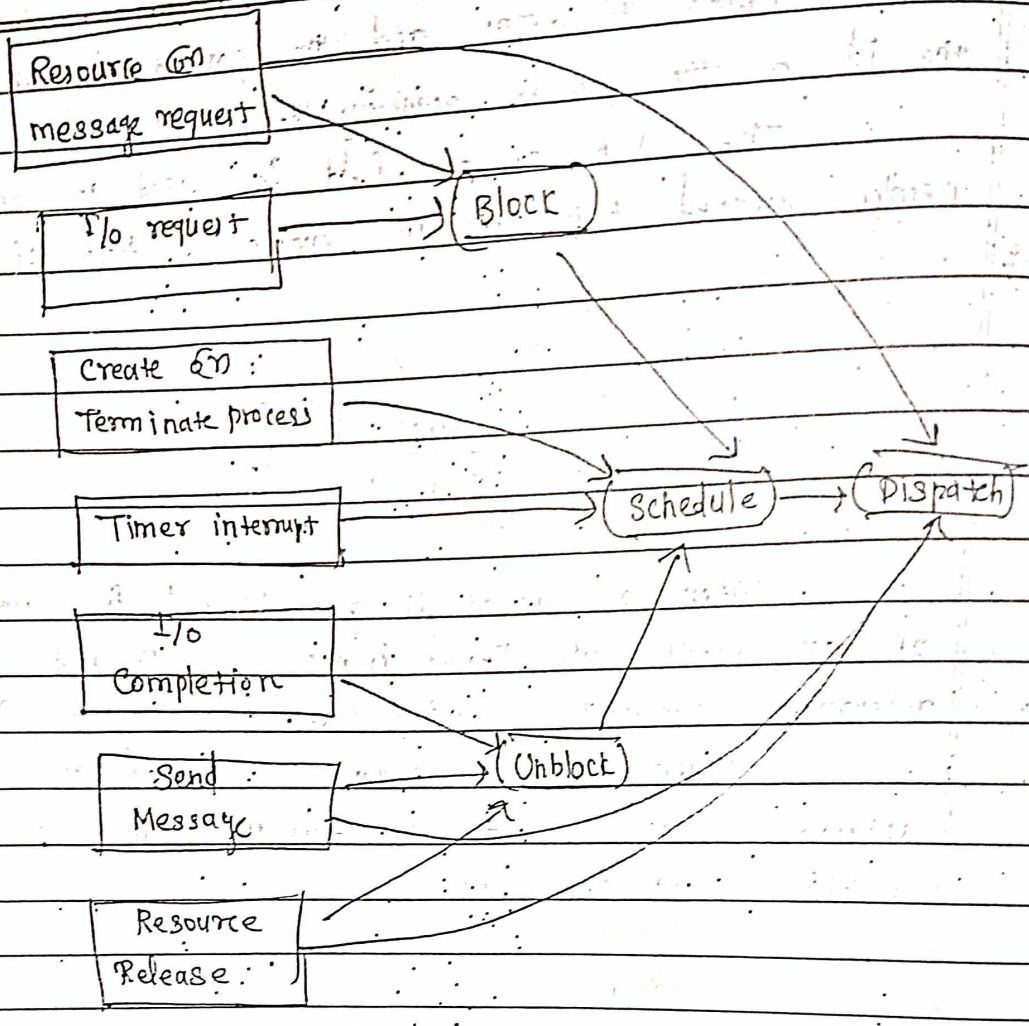
The ECB pointer field is used to enter the newly created ECB into an appropriate list of ECB's

Event description
Process id
ECB pointer

When a process P_i gets blocked for occurrence of an event, the kernel forms an ECB and puts relevant information concerning event and process P_i into it. When an event occurs, the kernel scans the appropriate list of ECBs to find an ECB with a matching event description.



PCB - ECB relationship



Event handling actions of the kernel.

*) Sharing, Communication and Synchronization between processes :-

~~1) Data Sharing :-~~

1) Message Passing :- A process may send some information to another process in the form of a message. The other process can copy the information into its own data structures and use it. So the information exchang becomes a part of the convention (or) protocol between processes.

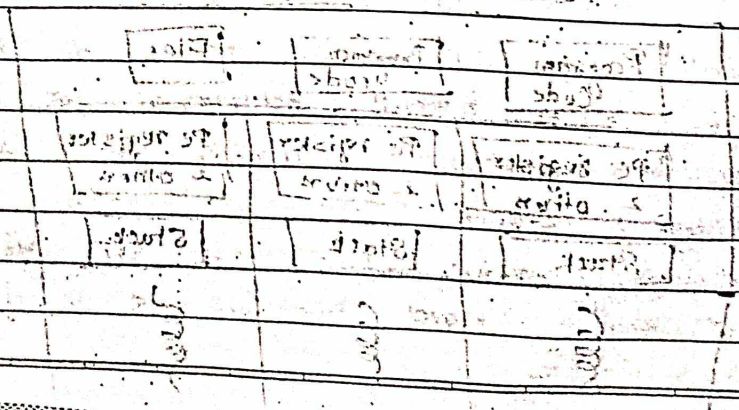
2) Synchronization :- The logic of a program may require that an action a_i should be performed only after some action a_j has been performed. Synchronization between

processes is required if these actions are performed in different processes - the processes that wishes to perform action a_i is made to wait until another process performs action a_j.

3) Signals :- A signal is used to convey an exceptional situation to a process so that it may handle the situation through appropriate actions. The code that a process wishes to execute on receiving a signal is called signal handler. Thus when a signal is sent to a process, the kernel interrupts operation of the process and executes a signal handler.

4) Data Sharing :- For eg :- If two processes concurrently execute the same statement $a = a + 1$, when a is a shared variable, the result may depend on the way the kernel interleaves their execution.

To avoid this problem, only one process should access shared data at any time, so data access in one process may have to be delayed if another process is accessing the data. This is called mutual exclusion.



Page No.
Date:

*) Threads :-

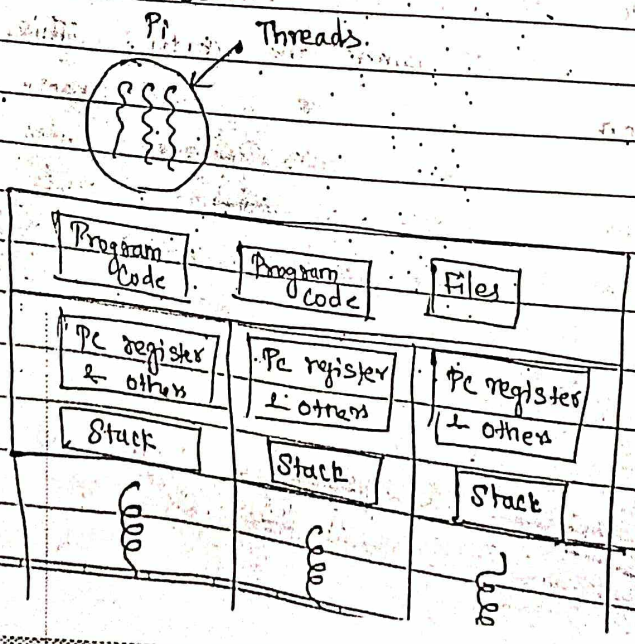
An execution of a program that uses the resources of a process.

A process creates a thread through a system call. The thread does not have resources of its own. It accesses the resources of the process through it.

Process P_i has three threads which are represented by wavy lines inside the circle representing process P_i has a context and PCB.

Each thread of P_i is an execution of a program, so that it has its own stack and a thread control block (TCB.), which stores the following information.

1) Thread scheduling information - threadid, priority and state
2) CPU state (i.e). contents of the PSW and GPRs
3) Pointer to PCB of parent process
4) TCB pointer which is used to make lists of TCB for Scheduling.



The process can be split down into so many threads. For eg :- in a browser, many tabs can be viewed as threads. MS words uses many threads - formatting text from one thread, processing input from another thread, etc.

Advantages of Threads over Processes :-

1) Lower overhead of creation and switching
2) More efficient communication
3) Simplification of design

Kernel-Level, User-level, Hybrid-level thread :-

There three models of threads differ in the role of the process and the kernel in the creation and management of threads.

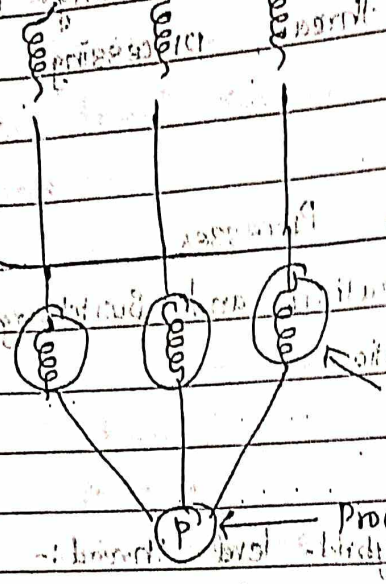
Kernel-Level thread :-

A kernel-level thread is implemented by the kernel. Hence creation and termination of kernel-level of threads and checking of their status is performed through system calls.

When a process makes a create thread system call, the kernel creates a thread assigns an id to it, and allocates a thread control block (TCB). The TCB contains a pointer to the PCB of the parent process of the thread.

When an event occurs, the kernel saves the CPU state of the interrupted thread in its TCB. After event handling, the scheduler considers TCBs of all threads and select one ready thread, then PCB pointer in its TCB to check whether the selected thread belongs to a different process than the

interrupted thread



The kernel thread recognizes the operating system. There is a thread control block and process control block in the system for each thread and process in the kernel-level thread. The kernel-level thread is implemented by the OS.

The kernel knows about all the thread and manages them. The kernel-level thread offers a system call to create and manage the thread from user-space. The implementation of kernel thread is more difficult than the user thread.

Advantages:-

1) The kernel-level thread is fully aware of all threads.

2) The scheduler may decide to spend more CPU time in the process of threads.

Disadvantages:-

1) The implementation of kernel level thread is difficult than user level.

2) It is slower than user-level thread.

Page No.
Date: / /

User - level thread :-

User - level thread are implemented by a thread library, which is linked to the code of a process. The library sets up the thread implementation without involving the kernel and itself interleaves operation of thread in the process.

Thus, the kernel is not aware of presence of user - level threads in a process.

A process invokes the library function create - thread to create a new thread. The library function creates a TCB for the new thread and starts considering the new thread for "Scheduling".

The library function invokes "Scheduling" and switches to another thread of the process.

If the thread library cannot find a ready thread in a process, it makes a block me system call. The kernel now blocks the process. It will be unblocked when some event activates one of its threads and will resume the execution which will perform scheduling and switch to execution of the newly activated thread.

Advantages :-

User - level threads are easier and faster to create than kernel - level

User - level can run on any os

Disadvantages :-

The entire process is blocked if one user - level thread performs blocking operation

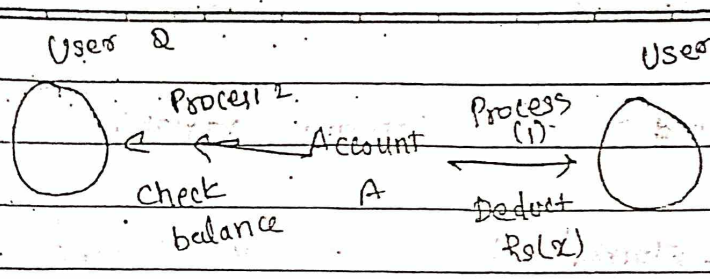
*) Process Synchronization :-

Process Synchronization is the way by which processes that share the same memory space are managed in an OS. It helps maintain the consistency of data by using variables (or hardware) so that only one process can make changes to the shared memory at a time. There are various solutions for the same such as semaphores, synchronization hardware, etc.

An operating system is a software that manages all applications on a device and basically helps in smooth functioning of our computer. Because of this reason, the OS has to perform many tasks, and sometimes simultaneously. This is not usually a problem unless these simultaneously occurring processes use a common resource.

For ex:- Consider a bank that stores the account balance of each customer in the same database. Now suppose you initially have x rupees in your account. Now you take out some amount of money from your bank account and at the same time, someone tries to look at the amount of money stored in your account.

As you are taking out some money from your account, after the transaction, the total balance left will be lower than x. But the transaction takes time, and hence the person reads x as your account balance which leads to inconsistent of data.



In the above image, process P_1 and P_2 happens at the same time, User 2 will get the wrong account balance as Y because of process 1 being transacted.

Inconsistency of a data occur when various processes share a common resources in a system.

Process Synchronization occurs in the co-operative process.

*) Race condition :-

When more than one process is either running the same code (or) modifying the same memory (or) any shared data, there is a risk that the result (or) value of the shared data may be incorrect because all processes try to access and modify this shared resource. Thus, all the processes race to say that my result is correct. This condition is called race condition.

To avoid Race condition :-

By treating the critical section as a section that can be accessed by only a single process at a time. This kind of section is called atomic section.

Page No.
Date:

Example :-

(shared = 5) common variable

P1	P2
int x = shared	int y = shared ;
x++ 6	y-- ; 4 ;
sleep(1) ;	sleep (1)
shared = x ;	shared = y ;

output

1) shared = 4
2) shared = 6

There are many outputs are possible for above processes so it is called Race condition.

*) Critical Section :-

A part of code that can only be accessed by a single process at any moment is known as critical section.

This means that when a lot of program want to access and change a single shared data

A process that is executing a critical section is said to be in a "critical section". We also use the terms "enter a critical section" and "exit a critical section" for situation where a process starts and completes an execution of a critical section.

Eg:- Use of critical section in airline reservation system.

```
if nextseatno < capacity
then
    allotedno = nextseatno ;
    nexseatno = nextseatno + 1 ;
```

else

display "sorry, no seats available";

Process : 1

if nextseatno < capacity

then

allotedno = nextseatno;

nextseatno = nextseatno + 1;

else

display "sorry no seats available";

Process Pj

Essential Properties of a critical section

1) Mutual exclusion :-

If a process is running in the critical section, no other process should be allowed to run in that section at that time.

2) Progress :-

When no processes is executing in critical section for a data item one of the processes wishing to enter a critical section for data item will be granted entry.

3) Bounded wait :-

Bounded wait property ensures that this does not happen by limiting the number of time other processes can gain entry to a critical section ahead of a requesting process P_i.

Page No.
Date:

*) Process Synchronization approaches :-

1) Hardware Support for process Synchronization :-

Process Synchronization involves executing some sequence of instructions in a mutually exclusive manner.

On a Uniprocessor System, this System can be achieved by disabling interrupts while process executing such a sequence of instructions. So that it will not be preempted and also delays the process of interrupts.

It is not applicable for multiprocessor systems. For these reason the os implements critical section

2) Use a Lock Variable :-

The lock variable has only two possible variat values :- open and closed. When a process wants to execute a critical section, it tests the value of the lock variable. If the lock is open, it closes the lock and executes the critical section and open the lock after executing.

To avoid race condition the lock variable is used.

```
do
{
    acquire lock
    CS
    release lock
}
```

```
entry_test: if lock = closed
            then goto entry_test;
            lock = closed
{
    critical section
}
lock = open;
```

3) Test and Set instruction :-

1) while (test & set (&lock));
{
    cs
}
lock = false;

2) boolean test_and_set (boolean * target)
{
    boolean r = * target;
    * Target = True;
    return = r;
}