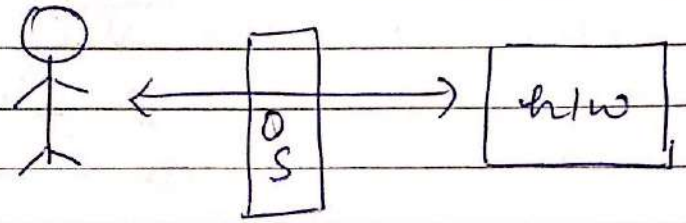
System software is a type of computer program that is designed to run a computer's hardware and application programs.

Operating Software :-

→ Operating is a system software.

→ It acts as an intermediatory between hardware and user.



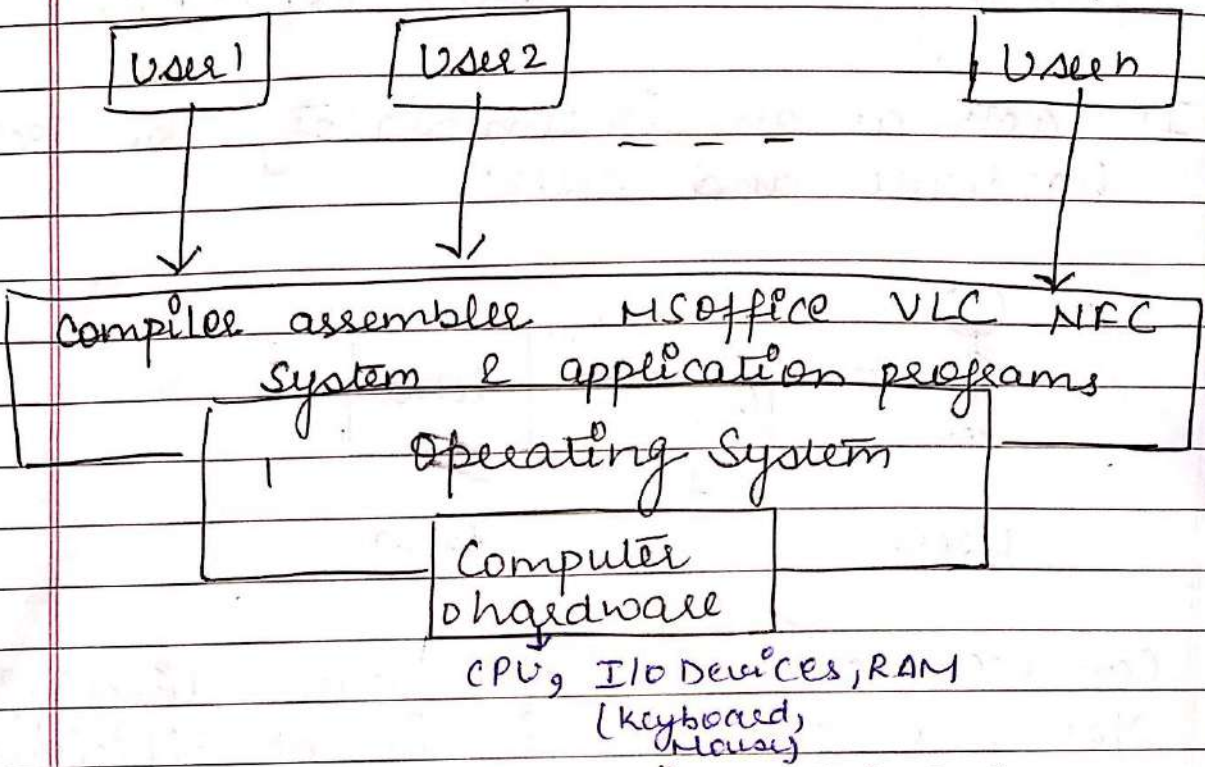
Can we access the hardware directly? ~~from~~
Yes, we can but it is not efficient and convinent.
For ex → We can switch on the ac without switch button. A mechanical engineer know how to open it but not everyone has the complete knowledge about it.

→ Resource Manager - manage system resources in an unbiased fashion both hardware & software.
all resources it will be difficult if user try to access and resources are too many then it will be difficult to manage

→ provides a platform on which other application progeams are installed.

Date / /
Page

Abstract view of OS



Goals and Functions Of OS

→ Primary Goal
Convinent / user friendly

→ Secondary Goal
efficient

Functions of OS
→ Process Management
→ Memory Management
→ I/O device Management
→ File Management
→ Network Mgt
→ Security & Protection

Evolution of OS

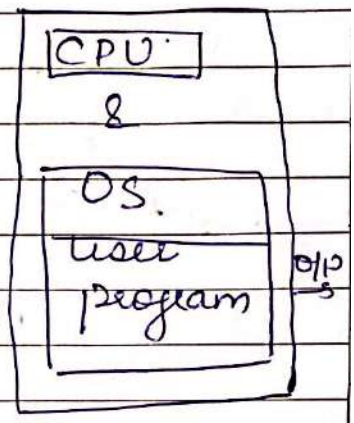
Date / /
Page

In starting mainframe computers

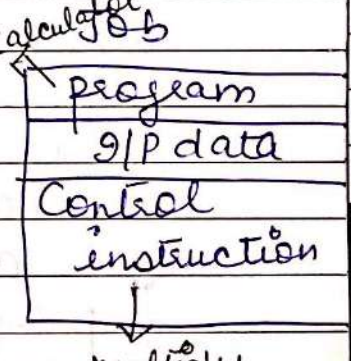
→ common I/P & O/P devices were card readers and tape drives

→ user prepare a job which consist of the program, input data, control instructions

→ I/P job is given in the form of punch cards and results also appear in form of punch card after processing.



→ So OS was very simple always present in memory major task is to transfer the control from one job to another.

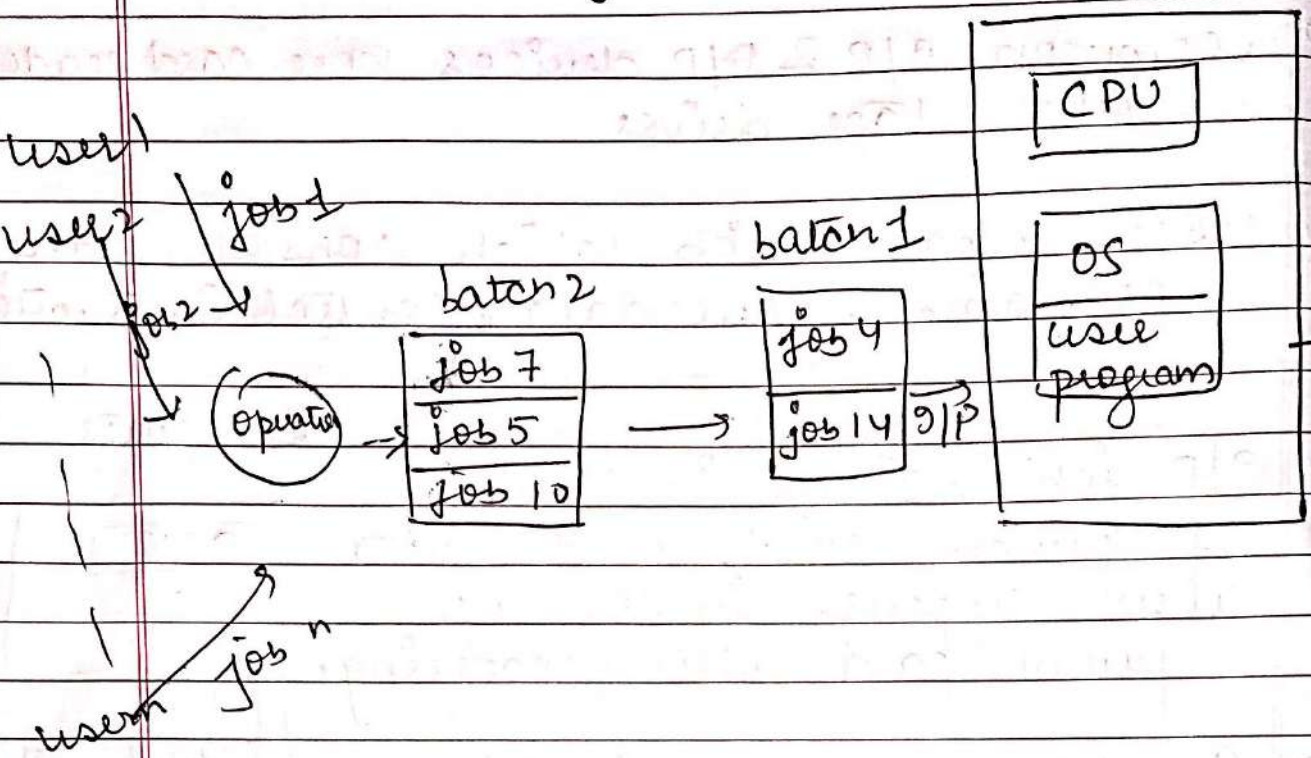


Scope of improvement

→ (Memory is very limited)

→ speed mismatch b/w I/P devices & processor.
(I/P device (1 sec.) & CPU (1 millisecond) most of the time CPU is idle.
(system was slow)

Batch processing



Operator इन jobs को batch में convert किस base पर कर रहा है।

depends on jobs nature

For ex→ FORTRAN (If program is written in FORTRAN then machine has to load the compiler of FORTRAN and all data and after that if the program is written in another language then you have to load the compiler of another language)

(Same requirements or needs jobs को साथ रखते हैं।)

→ jobs with similar needs are batched together and executed through the processor as a group.

→ Operator sort jobs as a deck of punch cards into batch with similar needs.

→ FORTRAN batch / COBOL batch

Advantages

→ in a batch job execute one after another saving time from activities like loading compiles.

→ during a batch execution no human intervention is needed.

Disadvantages ↓

→ memory limitation

→ interaction of I/P & O/P devices directly with CPU.

CPU is doing own work and at that time you are performing input/output operations.

I/P operations

Spooling
Simultaneous peripheral operations online

→ It is a acronym.
→ It is a technology where input/output operations are done simultaneously.
→ I/P/O/P devices are not digital in nature and CPU ~~are~~ is digital in nature means the speed of CPU is 2.3GHz is normal. If we assume 1GHz then it means 1,000,000,000 Hz
Hz means frequency means 1 sec.
main 1 sec baar clock chalata hai. CPU ki speed enormous hai
→ 1 sec. main 5-6 alphabets type kar sakte hai so ek mismatch hai.

X— X— X— X— X— X— X— X— X— X—

→ I/P & O/P devices are relatively slow compare to CPU (digital)

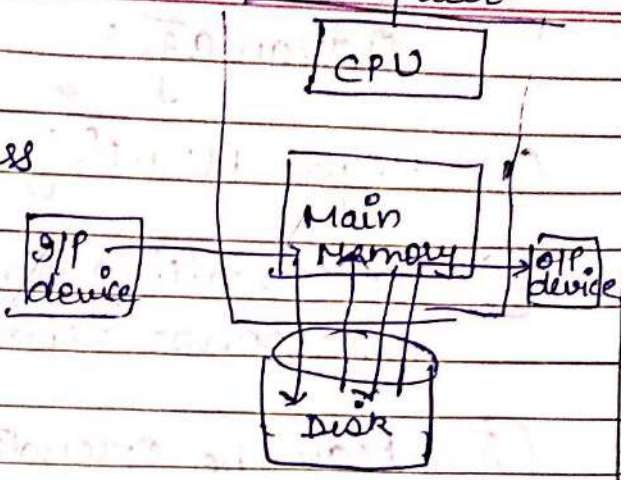
→ In spooling, data is stored first onto the disk, and then CPU interact with disk (digital) via. main memory
→ keyboard, mouse, printer etc.
→ Spooling is capable of overlapping I/O operations for one job with CPU operations of other jobs.

Advantage	Disadvantage
→ no interaction of I/P & O/P device with CPU	→ In starting, spooling was uniprogramming.
→ CPU utilization is more as CPU is busy most of the time	

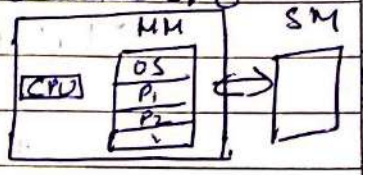
Date / /
Page

uniprogramming

(suppose apne koi process liya. CPU is executing that process but this process needs some I/O operations then CPU wait for this I/O operation. It is not intelligent that it will execute another process in between.)



Multiprogramming OS
(when multiple process reside in main memory)



→ Maximize CPU utilization

→ Multiprogramming means more than one process in main memory are ready to execute. (It doesn't mean we can execute more than one program on CPU).

→ Process generally require CPU time & I/O time. So if a running process perform I/O or some other event which donot require CPU then instead of sitting idle. CPU make a context switch and picks some other process and this idea will continue.

→ CPU never idle unless there is not process ready to execute or at time of context switch.

Advantages	Disadvantages
(1) High CPU utilization.	(1) Difficult scheduling.
(2) less waiting time, response time etc.	(2) Main memory mgt is required.
(3) May be extended to multiple users.	(3) Memory fragmentation
(4) Now-a days useful when load is more.	(4) Paging (Non-contiguonous memory allocation).

Multitasking OS / Time Sharing / Fair share / Multiprogramming with Round-Robin

→ Multitasking is multiprogramming with time sharing.

→ Only one CPU but switches between processes so quickly that gives an illusion that all executing at same time.

→ The task in multitasking may refer to multiple threads of same program.

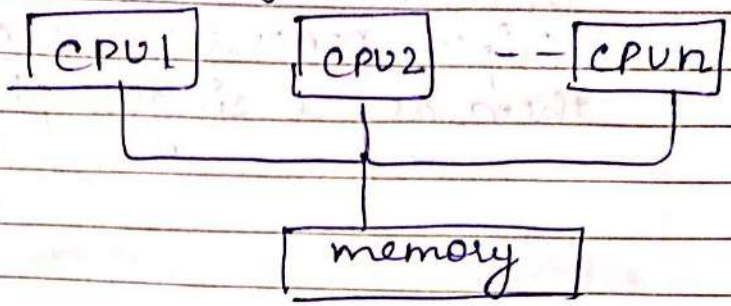
→ Main idea is better response time and executing process together.

Date / /
Page

RANKA

Multiprocessing OS

→ Two or more CPU with a single computer, in close communication sharing the system bus, memory & other I/O devices.



- Different process may run of different CPU, ~~each CPU has equal rights~~. true parallel execution
- Symmetric → one OS control all CPU, each CPU has equal rights
- Assymetric → Master slave architecture, system task take on one processor & application on other or one CPU will handle all h/w interrupt or I/O devices, they are easy to design but less efficient

Advantage	Disadvantage
→ Increased throughput	- more complex
- Increased reliability	- overhead of coupling reduce throughput
→ cost saving	- large main memory
→ battery efficient	
- true parallel processing	

Date / /
Page
RANKA

Multicore refers to an architecture in which a single physical processor incorporates the core logic of more than one processor. A single integrated circuit is used to package or hold these processor. Multicore architecture places multiple processor cores and bundles them as a single physical processor.

Date / /
Page

RANKA

Process Concept

Processes

Ex: .c

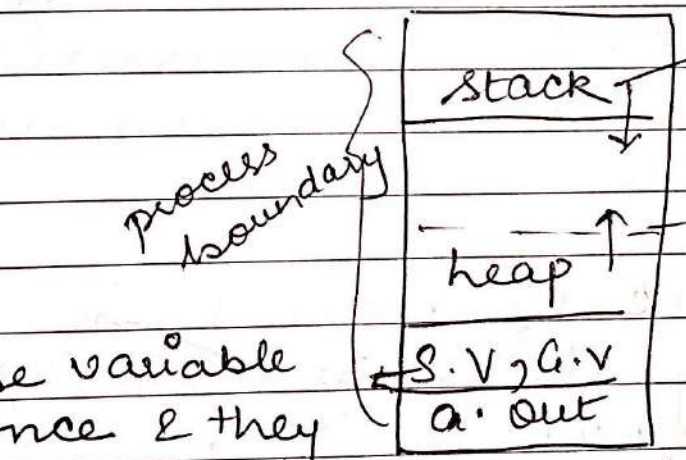
↓

compiler

↓

a.out	executable code

This executable code is stored in hard disk whenever the CPU wants to execute it. The executable code is brought up to the main memory whenever the executable code is brought up to the main memory. One kind of data structure is formed in main memory.



These variable created once & they remains forever as it is (lifetime of process)

Process in memory

→ This process is given to CPU to execute.
→ and CPU will execute it.
→

→ Program by itself is not a process
→ Process → A program in execution is called process.

→ Process Needs → CPU + I/O devices + Memory + Files

→ Program is a passive entity (such as a file containing a list of instructions stored on disk) whereas process is an active entity.

– The program becomes process when an executable file is loaded into memory.

– Two common techniques for loading executable files are double-clicking an icon representing the executable file and entering the name of executable file on the command line.

→ There are lot of processes in main memory. How to identify these processes? Through PCB.

↓

It will contain the info which process is running and which process is waiting.

→ All info. about process is stored in PCB.

long Term Scheduler - Job scheduler → determines which programs are admitted to the system for processing.

Date / /
Page

RANKA

Process State

- As a process executes, it changes state.
- The state of a process is defined by the current activity of that process.
- Each process may be in one of the following states -

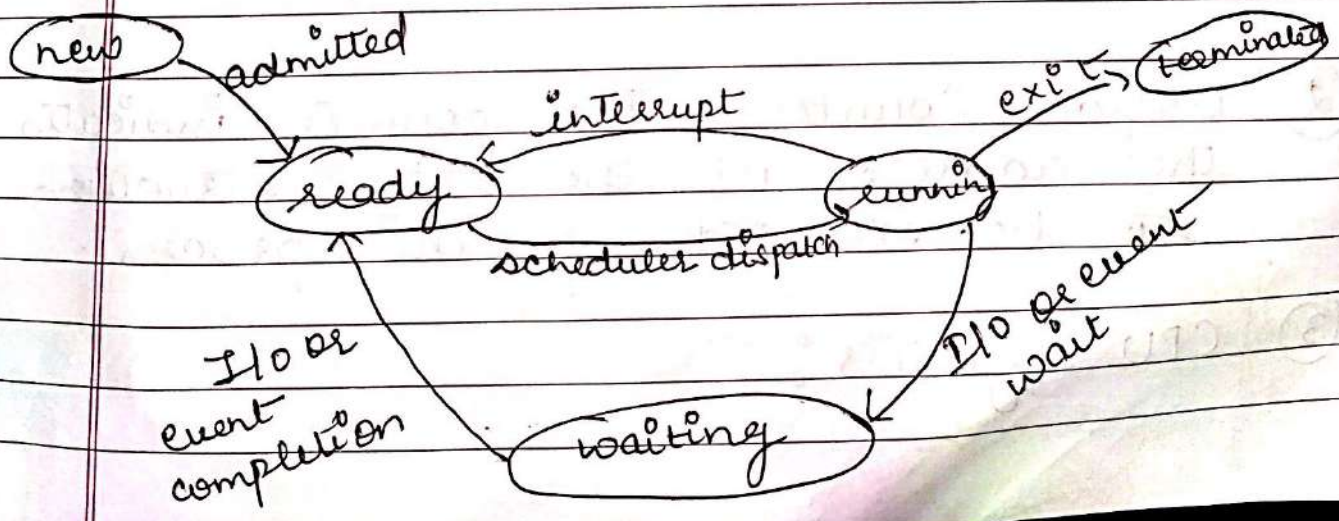
(1) New — The process is being created.

(2) Running — Instructions are being executed.

(3) Waiting : — The process is waiting for some event to occur (such as an I/O completion or reception of a signal).

(4) Ready : — The process is waiting to be assigned to a processor. (on the ready list, waiting for the CPU)

(5) Terminated : The process has finished execution.



Interrupt - means issue

Date / /
Page

Only one process can be running on any processor at any time.

Process Control Block

PCB or task control block serves as repository for any informations that vary from process to process.

Information associated with each process.

process state
process number
program counter
registers
memory limits
list of open files

(1) Process state - The state may be new, ready, running, waiting, halted and so on.

(2) Program Counter - The counter indicates the address of the next instruction to be executed for this process.

(3) CPU registers:

(4) CPU registers & scheduling information :- priorities, scheduling queue pointers.

(5) Memory Management information:
memory allocated to the process.
This information includes such information as the value of the base and limit registers, the page tables or the segment tables, depending on the memory system used by the operating system.

(6) Accounting information :-
This information may include the amount of CPU and real time used, clock time, time limits, job or process numbers.

(7) I/O status information :-
This information includes the list of I/O devices allocated to a process, a list of open file and so on.

Paging & Segmentation
(It is a memory mgt scheme by which a computer stores or retrieves data from secondary storage for use in main memory)

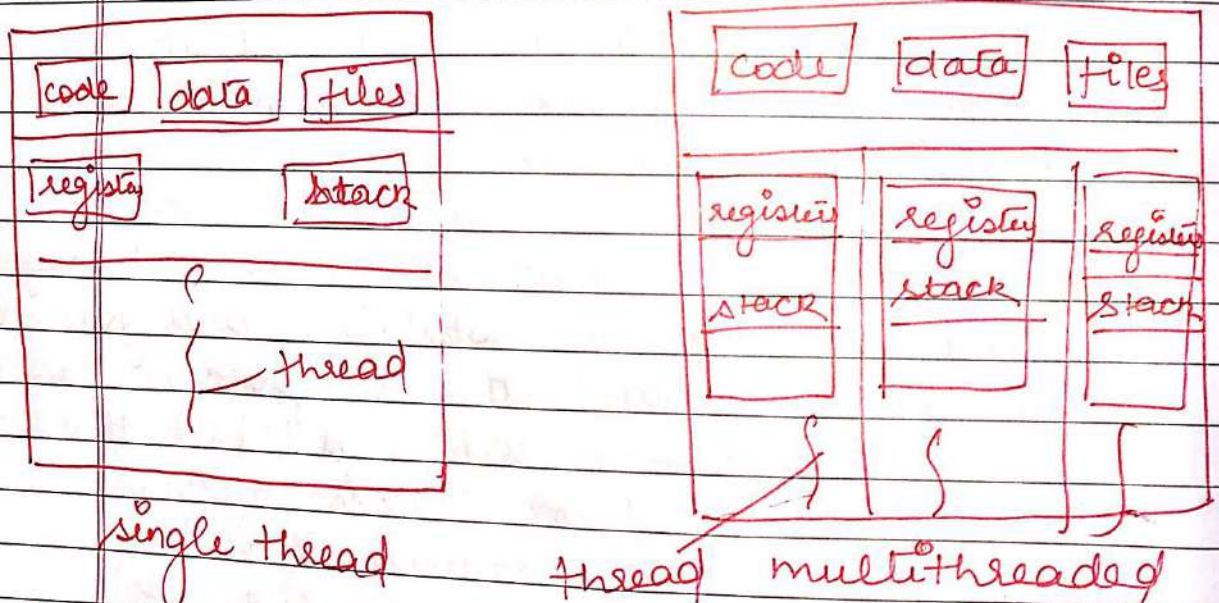
Date / /
Page

Threads

→ A thread is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers.

→ Traditional (heavyweight) processes have a single thread of control — There is one program counter, and one sequence of instructions that can be carried out at any given time.

→ Multithreaded applications have multiple threads within a single process, each having their own program counter, stack, registers and certain data structure.



single thread multithreaded

Q→ Threads of a process share
a) global variables but not heap
b) heap but not global variables
c) neither global variables nor heap
d) both heap and global variables

Date / /
Page

Threads

A process can be multithreaded.
A process may have multiple unit of execution.

Ex→ Microsoft soft word ~~is~~ entirely can be treated as a process.
⇒ functionality of MS-word is accept input, format the text, spellchecking green line for semantic & grammatical mistakes and store the data
5 different functions are the part of the processes.

It is not necessary that all 5 functionalities of the program are done by one single sequential code.

That give us notation of multithreaded program where each one of the thread is doing one of its functionalities when you invoke MS-word then 5 different execution units in action.
A process is a program that performs a single threaded execution.
* when a process is running a word-processor program, a single thread of instructions is being executed.
⇒ This single thread of control allows the process to perform only one task at one time.
⇒ The user cannot simultaneously type

Process is a program in execution and thread is also program in execution.

Date
Page
RANKA

in characters and run the spell checker within the same process.

⇒ Many modern OS have extended the process concept to allow a process to have multiple threads of execution and thus to perform more than one task at a time.

⇒ On a system that supports threads the PCB is expanded to include information for each thread.

⇒ Difference b/w thread & process

Similarities

① like process, thread can create children.

② And like process, if one thread is blocked, another thread can run. (in some scenarios)

③ like processes, threads share CPU and only one thread active (running) at a time.

Differences → (MS word & MS powerpoint)
they are different processes and they share less information or may be not shared information.
MS-word 5 threads are executing in processes they share lot of information

for eg → the thread accepts the input

the thread that formats the input,
" " draw green line for the input
" " red " " " "
" " actually saves the input

Note that input is used in all 5 lines
It means threads share lot of information among themselves and collectively try to do a single job while processes can be two different entities and perform different jobs. (functional level)

architectural level -

i) Threads are not independent of one another whereas process are independent of each other.
② Threads are designed to assist one other.
③ as processes may originate from different users they may or may not assist one another.

Benefits of Thread

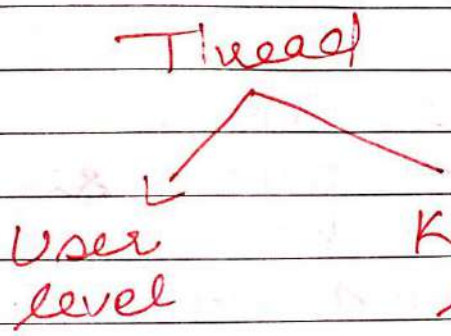
A process with multiple threads make a great server because threads can share common data, they do not need to use interprocess communication. Because of the very nature, threads can take advantage of multiprocessors.

Date / /
Page
RANKA

(2) Threads only need a stack and storage for registers; threads are easy to create.

(3) Threads use very little resources of an OS in which they are working; threads do not need new address space, global data, program code or OS resources.

(4) Context switching are fast when working with threads. The reason is that we only have to save or restore PC, SP and registers.



Benefits of Thread

(1) It takes far less - time to create a new thread in an existing process than to create a new process.

(2) It takes less time to Teminate a a thread than a process.

(3) It takes less time to switch b/w two threads within the same process than to switch b/w processes.

(4) Threads enhance efficiency in communication b/w different executing programs. In most OS, communication b/w independent process require the intervention of the kernel to provide protection and the mechanisms needed for communication. Because threads within the same process share memory and files, they can communicate with each other without invoking the kernel.

Kernel is a compulter program that is the core of a computer's OS. The kernel facilitates interactions b/w hardware & software components. One most systems, it is one of the first progrars loaded on start-up.

Date / /
Page
RANKA

X — X — X — X — X — X — X — X — X —

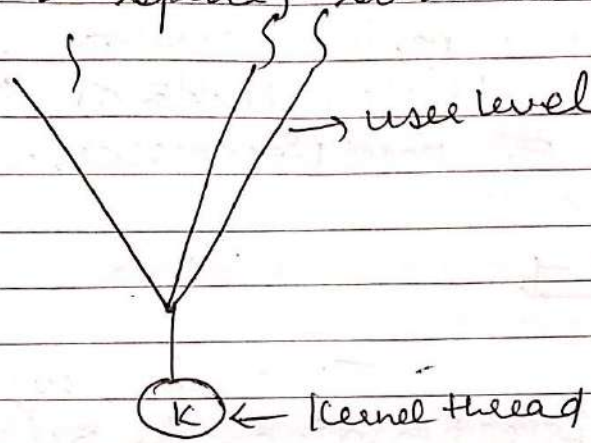
	User-level thread	Kernel-level thread
i)	Implemented by user or programmer.	1) Implemented by OS
ii)	OS doesn't know about it (OS sees user level thread as only process)	2) OS recognises the kernel level thread.
iii)	If one user level thread gets blocked then entire process get blocked	3) If one kernel level thread gets blocked others will continue to execute.
iv)	Dependent threads	4) Independent threads.
v)	Implementation is easy	5) Implementation is complicated
6	less time Context	6 more time Context
7	No Hardware support req. for the user level threads eg- Javathread, posix thread	7 Required Hardware support. ex- window Solaris thread

Date / /
Page

RANKA

Many-to-one Model

The many-to-one model maps many user-level threads to one kernel thread. Thread management is done by the thread library in user space, so it is efficient.



Advantage - It incurs low switching overhead, as the kernel is not involved when switching b/w thread.

Disadvantage

(1) If one user level thread issues a blocking system call, the kernel blocks the whole parent process.

(2) As the kernel level thread can be accessed by only one user-level thread at a time, multiple user level threads cannot run in parallel on multiple CPU's, thereby resulting in ~~less~~ no concurrency.

Date / /
Page

RANKA

One-to-One Model

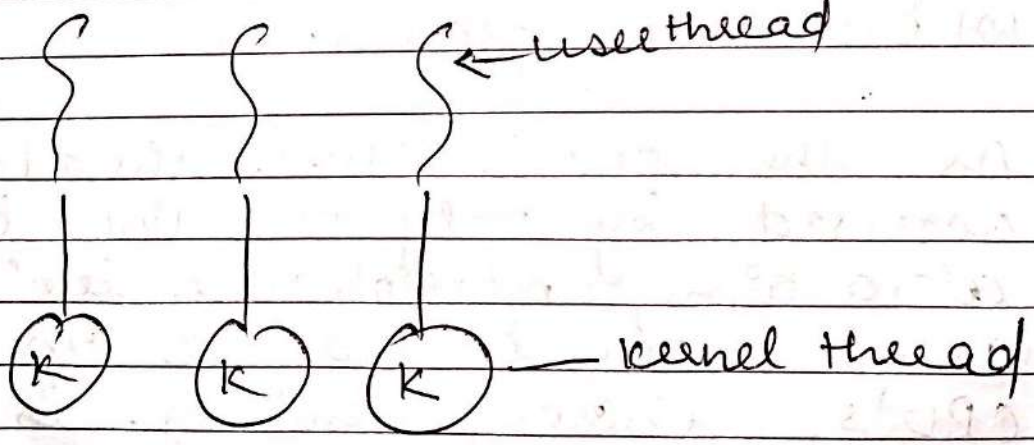
It maps each user thread to a kernel thread. It provides more concurrency than the many-one model by allowing another thread to run when a thread makes a blocking system call. It allows multiple threads to run in parallel on multiprocessors.

Drawback :-

Creating a user thread requires creating the corresponding kernel thread.

Advantage

Multiple threads can run in parallel on multiple CPU in a multiprocessor enviornment, thereby achieving greater concurrency.



Date / /
Page

RANKA

Many-to many

It multiplexes many user-level threads to a smaller or equal number of kernel threads. The no. of kernel threads may be specific to either a particular application or a particular m/c.

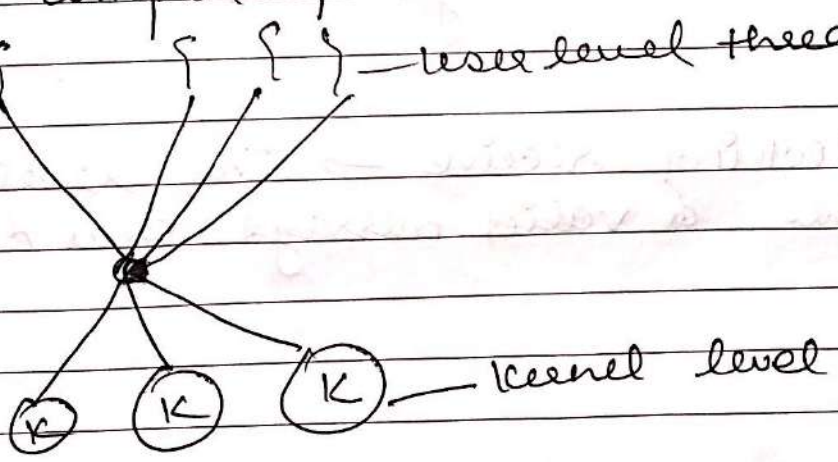
Advantages

i) Many user level threads can be made to run in parallel on different CPU by mapping each user-level thread to a different kernel-level thread.

ii) Blocking of one-user level thread does not result in the blocking of other user are mapped into different kernel level threads.

Disadvantage

i) Implementing this model involves a very complex procedure.



Date / /
Page

RANKA

Process scheduling

→ act of determining which process is in ready state and should be moved to the running state to maintain to keep CPU busy all the time.

→ Process scheduling is to used to timeshare a processor among multiple processes that are ready to run, enabling the concurrent execution of several processes.

→ It is the central mechanisms that enables multiprogramming and efficiency.

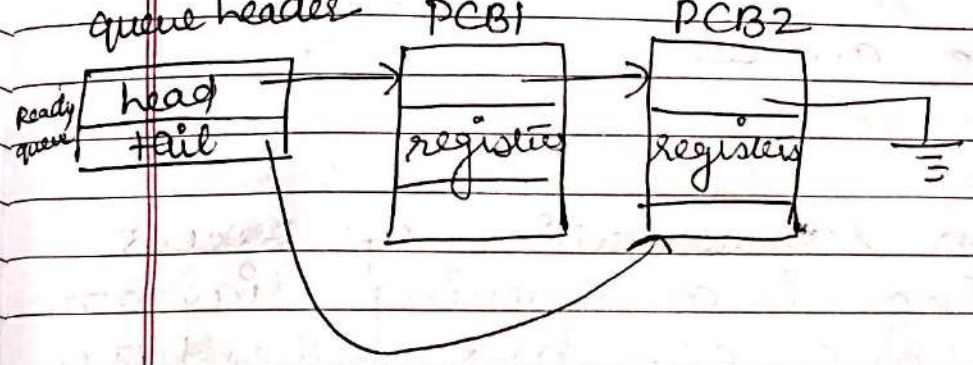
Scheduling Queues

→ As processes enter the system, they are put into job queue, which consists of all processes in the system.

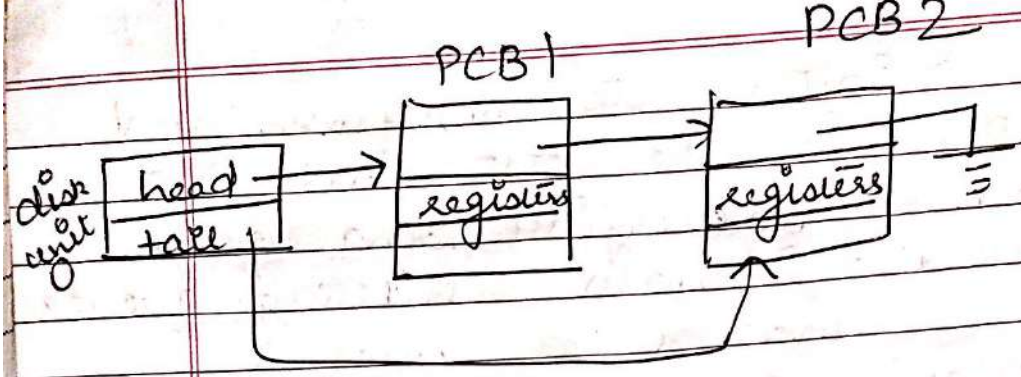
→ The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue.

→ This queue is generally stored as a linked list. A ready-queue header

consists pointers to the first and final PCB's in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue.



- The system also includes other queues. When a process is allocated the CPU, it executes for a while and eventually quits, is interrupted, or waits for the occurrence of a particular event, such as the completion of an I/O request. Suppose the process makes an I/O request to a shared devices, such as a disk. Since there are many processes in the system, the disk may be busy with the I/O request for some other process. The process therefore may have to wait for the disk. The list of processes waiting for a particular I/O device is called a device queue. Each device has its own device queue.



device queue

A common representation of process scheduling is a queuing diagram,

- Each rectangular box represents a queue. Two types of queues are present: the ready queue and a set of device queue.

— The circle represent the resources that serves the queues, and the arrows indicate the flow of processes in the system.

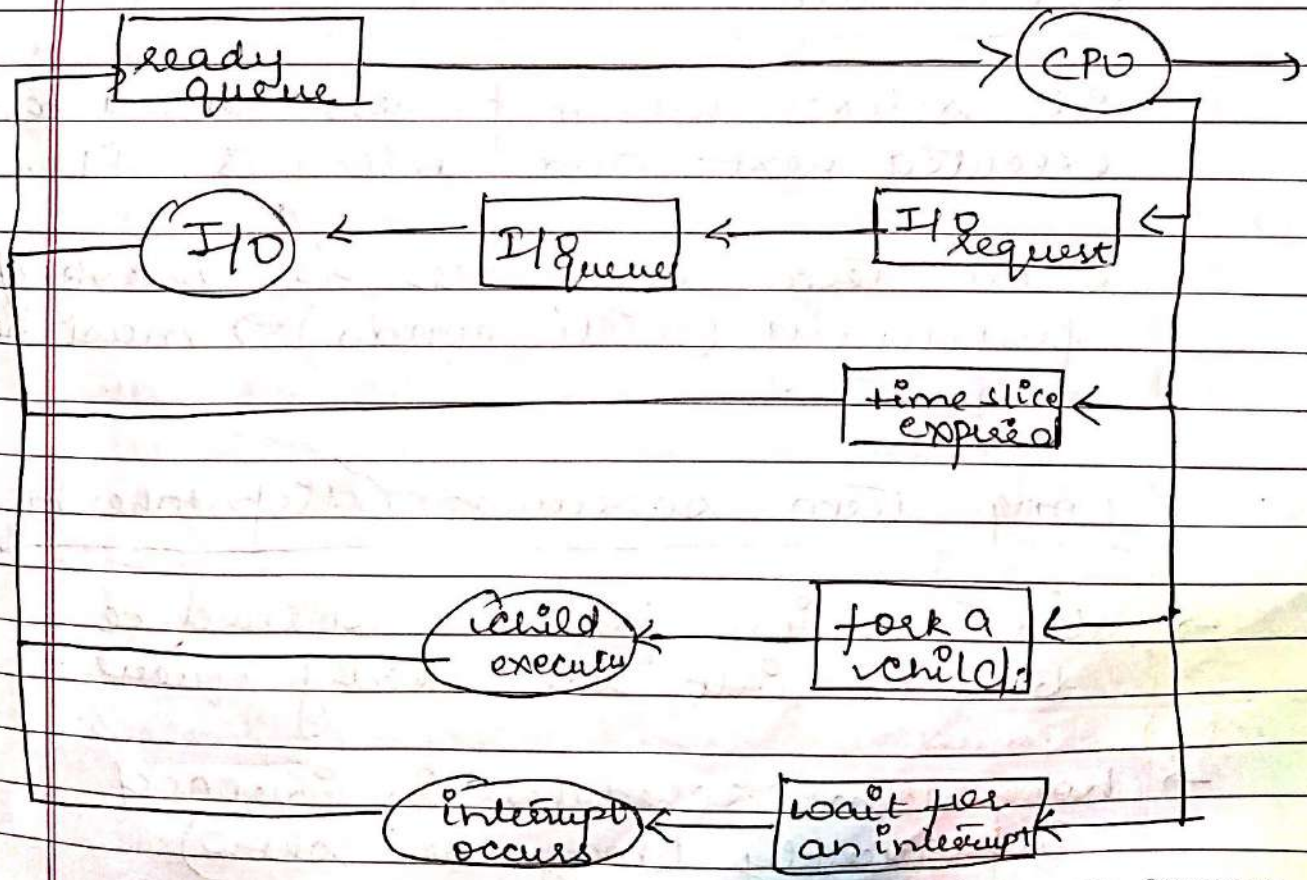
— A new process is initially put in the ready queue. It waits there until it is selected for execution or is dispatched. Once the process is allocated the CPU and is executing one of several events can occur—

- The process could issue an I/O request and then be placed in an I/O queue.
- The process could create a new subprocess

and wait for the subprocess termination.

- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

- In the first two cases, the process eventually switches from the waiting state to the ready state and then put back into the ready queue. A process continues this cycle until it terminates, at which time it is removed from all queues and its PCB and resources deallocated.



Queuing diagram representation of process scheduling

Date / /
Page
RANKA

Schedulers

- A process migrates among the various scheduling queues throughout its lifetime. The operating system must select processes from these queues in some fashion. The selection process is carried out by the appropriate scheduler.

There are ~~two~~ three types of schedulers

i) Short term scheduler (CPU scheduler)
ii) Long term scheduler (Job scheduler)
iii) Medium term scheduler.

Short-term scheduler :-

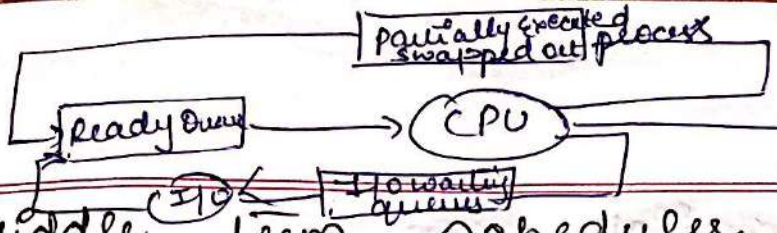
It selects which process should be executed next and allocates CPU.

Short term scheduler is invoked frequently (milliseconds) ⇒ must be fast

Long term scheduler (dispatcher) :-

→ selects which processes should be brought into the ready queue.

→ Long term scheduler is invoked infrequently (may be slow)



Date / /
Page

RANKA

Middle term scheduler

→ It is a process swapping scheduler
→ speed is b/w both short and long term scheduler.

→ It reduces the degree of multiprogramming
→ Medium term scheduler used to remove process from memory, later, the process can be reintroduced into memory, and its execution can be continued where it left off.

→ It is important that long-term scheduler makes a careful selection.

→ In general most processes can be described as either I/O bound or CPU bound.
→ An I/O bound process is one that spends more of its time doing I/O than it spends doing computation.
→ An CPU bound process, in contrast, generate I/O requests infrequently, using more of its time doing computations.

→ It is important that the long-term scheduler select a good process mix of I/O bound and CPU bound processes.

Date / /
Page

RANKA

Context switch

Transferring the control of CPU from one process to another demands for saving the context of currently running processes and loading the context of another ready process. This mechanism of saving and restoring the context is known as context switch.

→ Portion of PCB include the process state memory management information and CPU scheduling information together constitute the context of a process.

→ Context switch occur due to a number of reasons some of which are as follows—

i) The current process terminates and exits from the system.

ii) The time slice of the current process expires.

iii) The process has to wait for I/O or some other resource.

iv) Some higher priority process enters the system.

Date / /
Page
RANKA

Context switch is performed in two steps -

① Save context : In this step, the kernel saves the context of the currently executing process in the PCB of the process so that it may restore this context later when its processing is done and the execution of the suspended process can be resumed.

② Restore context : In this step, the kernel loads the saved context of a different process that is process is executed next. Note that if the process to be executed is newly created and the CPU has not been allocated to it, there will be no saved context. In this case, the kernel loads the context of the new process. However, if the process to be executed was in waiting state due to temporarily unavailability of I/O or for some other reason, there will be saved context that can be restored.

One of the major drawbacks of using context switching is that it involves a huge cost to the system in terms of a real time & CPU cycles, because the system does not perform any productive work during switching.

Date / /
Page

RANKA

Operating System } Process state
P_1 : running
P_2 : ready

Interrupt or Systemcall

Save context PCB1

Restore context PCB2 } P_1 : ready
P_2 : ready

} P1 : ready
P2 : running

Interrupt or Systemcall

Save context PCB2

Restore context PCB1 } P1 : ready
P2 : ready

} P1 : running
P2 : ready

System call is a way to request services from kernel.

Date / /
Page

Operations on Processes

→ A process may create several processes, via create-process system, during the course of execution.

→ The creating process is called parent process and the new processes are called the children of that process.

→ Each of these new processes may in turn other processes, forming a tree of processes.

→ For creating a new process we use CreateProcess() in windows and fork() in Unix.

When a process creates a new Process, two possibilities exist in terms of execution –

(1) The parent continues to execute concurrently with its children

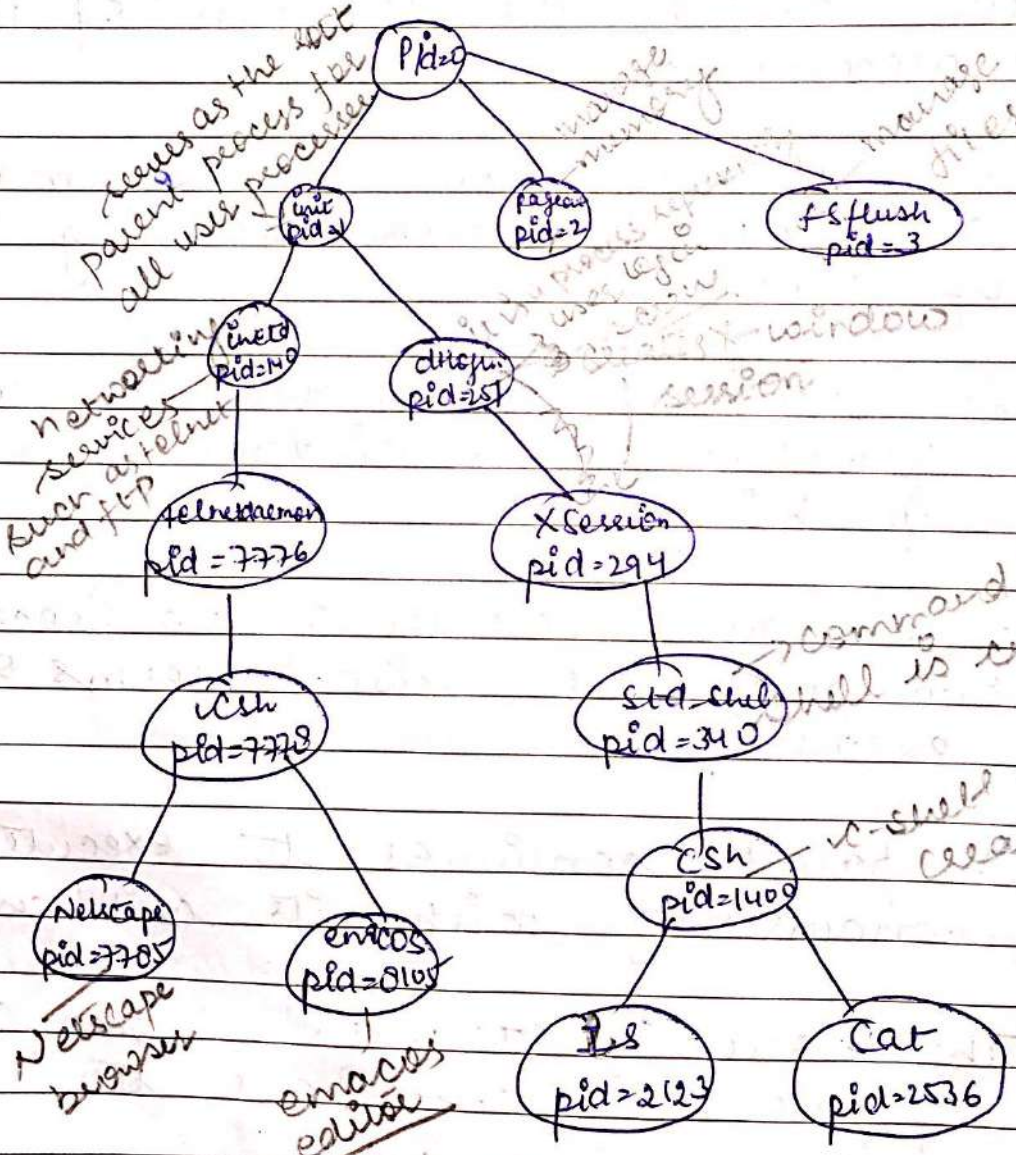
(2) The parent waits until some or all of its children have terminated.

Resource sharing options
- Parent and children share all resources
- Children share subset of parent's resources
- Parent and child share no resources.

There are also two possibilities in term of the address space of the new process-

(1) The child process is a duplicate of the parent process (it has the same program and data as the parent).

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



process tree for Solaris

Date / /
Page
RANKA

Process Termination

A process terminates when it has finished executing its last statement.

It tells the OS by issuing some form of exit() system call.

At that point, the process may return a status value (typically an integer) to its parent process (via wait() system call).

All the resources of the process - including physical and virtual memory, open files and I/O buffers - are deallocated by the OS.

A parent may terminate the execution of its children for a variety of reasons -

(1) The child has exceeded its usage of some of the resources that it has been allocated.

(2) The task assigned to the child is no longer required.

(3) The parent is exiting and the OS does not allow a child to continue if its parent terminates.

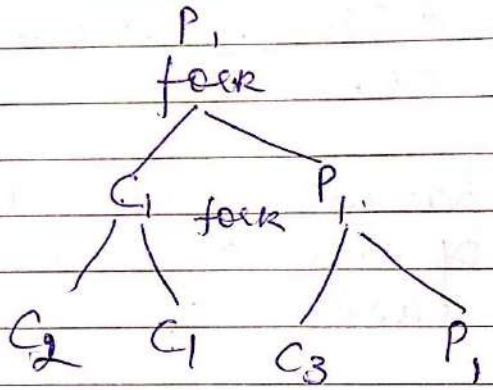
Some VMS, do not allow child to exist if its parent has terminated.

If a process terminates (either normally or abnormally) then all its children must also be terminated. This phenomenon reffered to as cascading termination, is normally initiated by the OS.

Q-

fork() < 0 → child process
1 → parent process
-1 → error

```
main()
{
fork();
fork();
}
```



three children processes are created.

wait() system call;
The parent process may wait for termination of a child by using the wait() system call. The call returns status information and the pid of the terminated process

pid = wait(&status);

Date / /
Page

RANKA

Interprocess communication

Processes executing concurrently in the OS may be either independent processes or cooperating processes.

→ Independent Processes are those processes that cannot affect or be affected by the other processes executing in the system. A process that donot share data to other processes.

→ An cooperating processes are those processes that can affect or be affected by the other processes executing in the system. Clearly any process that shares data with other process

→

There are several reasons for providing environment that allows process cooperation:-

i) Information sharing ii) Computation speedup
iii) Modularity iv) Convenience.

Cooperating processes require an Interprocess communication (IPC) mechanism that will allow them to exchange data and information

There are two fundamental model for IPC
i) Shared memory
ii) Message passing

i) Shared memory

In a shared memory model, a region of memory that is shared by cooperating processes

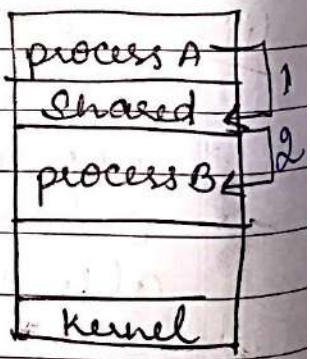
→ Processes can then exchange information by reading and writing data to the shared region.

→ Shared memory region resides in the address space of the process creating the shared memory segment.

→ Other processes that wish to communicate used this shared memory segment must attach it to their address space.

→ Process must also ensure that they do not write to the ~~th~~ same location simultaneously.

Producer - Consumer Example



Producer: Produces information that will be consumed by Consumer.

Consumer: Consumes information produced by the producer.

Two version of Producer buffer

Unbounded

bounded

Date / /
Page

Unbounded buffer — places no practical limit on the size of the buffer.

Bounded buffer — assumes a fixed size buffer

One sol^n to the producer - consumer problem uses shared memory.

- To allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer. This buffer will reside in a region of memory that is shared by the producer and consumer processes.
- A producer can produce one item while the consumer is consuming another item.
- Producer and Consumer must be synchronized so the consumer does not try to consume an item that has not yet been produced.
- The shared buffer is implement as a circular array with two logical pointers in and out.
- 'in' points to the next free position in the buffer.
- 'out' points to the first full position in the buffer.

→ The buffer is empty when in == out;
the buffer is full when ((in+1) % buffsize) == out

```
int n=4, item, in, out;
in=0
out=0
```

// producer process	Consumer process
while (true)	while (true)
{	{
while ((in+1)%n == out)	while (in == out
{	{
// Do nothing	// Do nothing
}	}
buffer[in] = nextProduction;	nextConsumed =
in = (in+1)%n	buffer[out];
	out = (out+1)%n

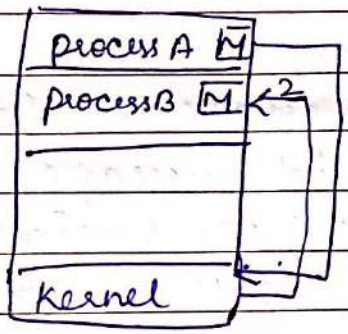
(2) Message Passing Model

In the message passing model, communication takes place by means of messages exchanged between the cooperating processes.

→ This model is particularly useful in a distributed envt, where the communicating processes may reside on different computers connected by a network. For ex., a chat program used on the world wide web could be designed so that chat participants communicate with one another by exchanging messages.

Date / /
Page

→ A message-passing facility provides atleast two operations: send(message) and receive(message).



→ It can also be implemented more easily than shared memory model.

→ It is typically using system calls and is more time consuming.

→ There is a communication link must exist b/w them.

There are three methods when one process send message ~~send~~ to another process –

i) Naming ii) Synchronization iii) Buffering

i) Naming → Processes that want to communicate must have a way to refer to each other

Direct Commⁿ, Indirect Commⁿ

Direct commⁿ → Each process that wants to communicate must explicitly name the recipent or sender of the communication. In this scheme, send() and receive() primitive are defined as–

i) Send (P, message) – Send a message to Process P.
ii) Receive (Q, message) – Receive a message from Process Q.

Symmetric communication → Both sender and receiver process must name the other to communicate.

Assymetric communication
Only sender process names the receiver, the receiver is not required to name the sender.

Send (P, message) → Send a message to Process P.
Receive (id, message) → Receive a message from any processes; the variable id set to name of the process with which comm^n has taken place.

Indirect Communication
On this messages are sent to and received from mailboxes or ports.
Send (A, message) – Send a message to mailbox A.
Receive (A, message) – Receive a message from mailbox A.

Properties of Communication link
Indirect communication

→ A link is established b/w a pair of process only if both have a shared mailbox.
→ A link may be associated with more than two processes.

Date / /
Page

Message - Passing Model

Synchronization :-

Communication b/w processes takes place through calls to send() and receive() primitives.

There are different design issue to implement these calls -

i) Blocking or Synchronous
ii) Non-blocking or Asynchronous

Blocking send - Sending process is blocked until the message is received by the receiving process or by the mailbox.

Non-blocking send - The sender process sends the message and resumes operation.

Blocking receiver → The receiver blocks until a message and resumes operation.

Non-blocking receive → The receiver receives either a valid message or a null.

Date / /
Page

RANKA

Buffering

Message exchanged via any type of communication resides in a temporary queue. These queues can be implemented in the following three ways —

i) Zero capacity → Queue has max. length of zero. Link cannot have message waiting in it. Sender must wait until recipent receives the message.

ii) Bounded capacity — Queue has finite length of n.

iii) Unbounded capacity → Queue has length is infinite.

Zero capacity sometimes call as a message system with no buffering.