Operating System

GATE CSE NOTES

NAME: Rakesh Nama. STD.: _____ SEC.: _____ ROLL NO.: _____ SUB.: O.S

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.		• Introduction of Operating system.	1-6	
2.		• Process Management :		
		(a) Process concept. (1)		
		(b) Threads. (1)		
	40%	(c) Interprocess communication. (2)		
		(d) Concurrency. (C.S) (2)		
		(e) Synchronization. (2)		
		(f) Deadlock. (3)		
		(g) CPU scheduling. (*) (1)		
3.		• Memory Management :		
	40%	(a) Memory mangement & virtual memory. (4)		
4.		• File System :		
		(a) File and Device management. (5)		

(www.gatenotes.in)

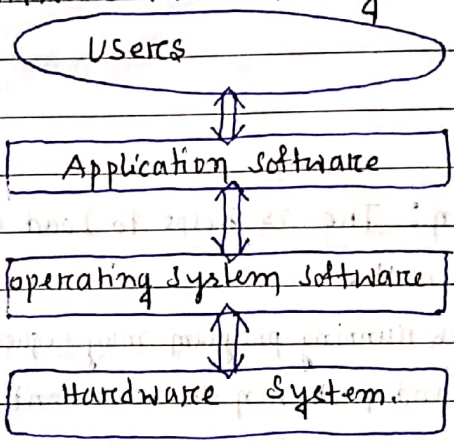
Saathi

Date 06/04/2019

INTRODUCTION OF OS

- What is Operating System?

→ An Operating System acts as an Intermediary between the user of a computer and the computer hardware. An operating System in a software that manage the computer hardware.



(Basic Diagram)

- Hardware: It provides the basic computing recources for the System. It consists of CPU, Memory and I/O Devices.

- Application programs: Define the ways in which these recources are used to solve user's computing problems. e.g Word processors, Spreadsheets, Compilers, and web browsers, Accounting software.

- Components of Operating System:
  - process management.
  - Main memory management.
  - File management.
  - I/O System management.
  - Secondary-storage Management.
  - protection System.
  - Networking.
  - Command-Interpete system.

- Function of operating System:
  1. program execution.
  2. I/O operation.

Date __/__/__

3. File system manipulation.
4. Communication.
5. Error Detection.
6. Resource allocation.
7. Accounting.
8. protection system.

- program execution: The OS helps to load a program into main memory and run it.
- I/O operation: Each running program may request for I/O operation and for efficiency and protection the user can't control I/O devices directly. Thus, the OS must provide some means to do I/O operation.
- File system manipulation: file are the most important part which needed by programs to read and write the files and files may also be created and deleted by name or by the programs. The OS is responsible for the file management.
- Communication: OS performs the communication among various types of processes in the form of shared memory. In the multitasking environment, the processes needed to communicate with each other and to exchange their information. These processes are created under a hierarchical structure where the main process is known as parent process and the subprocess are known as child processes.
- Error detection: It is neeessary that OS must be aware of possible errors and should take the appropriate action to ensure correct and consistent computing.
- Resource Allocation: Allocation of resources to the various processes is managed by O.S.
- Accounting: OS may also be used to keep track of the various computer resources and how much and which users are using these resources.
- protection: If a computer system has multiple users and allows

the concurrent execution of multiple processes, then the various processes must be protected from one another's activities.

- System call:

```
main()
{
  pf("....");
  sf("....");
}
```

Internally calls write() system calls in order to communicate with the monitor.
It is the request mode by the user program to the OS in order to get any kind of service.

- Types of operating system:

The operating system can perform a single operation and also multiple operations at a time. So there are many types of operating systems those are organized by using their working techniques.

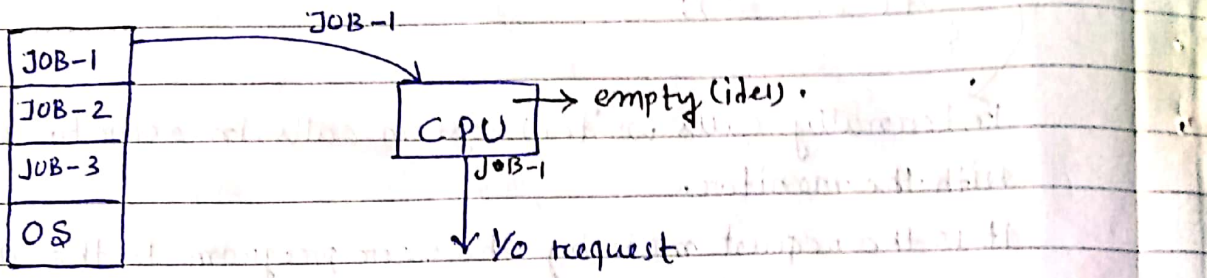
1. Serial processing: The serial processing operating system are those which performs all the instructions into a sequence manner or the instructions those are given by the user will be executed by using the FIFO manner means first in first out. Mainly the punch cards are used for this. In this all the jobs are firstly prepared and stored on the card and after that card will be entered in the system and after that all the instructions will be executed one by one. But the main problem is that a user doesn't interact with the system while he is working on the system, means the user can't be able to enter the data for execution.

2. Batch processing (O.S): The Batch processing is same as the serial processing Technique. But in the Batch processing similare types of job are firstly prepared and they are stored in the card and that card will be submit to the system for the processing. ~~The main~~

JOB = (CPU time + Yo time)
execution time

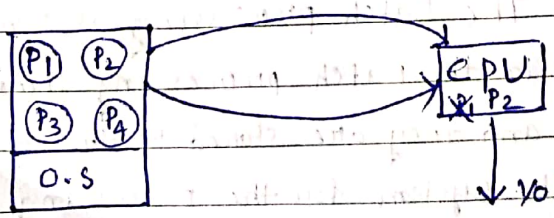
Date ___/___/_____

The main problem is that the jobs those are prepared for execution must be the same type and if a job requires for any type of input then this will not be possible of the user. The batch contains the jobs and all those jobs will be executed without the user intervention.



- If the job is fully completed, then only another job will be scheduled.
- Increased CPU idelness.
- The Throughput of the system will decreased.
  Throughput (the number of jobs completed per unit time is called throughput of the system).

3. Multiprogramming Operating System: Execute Multiple programs on the system at a time and in the Multiprogramming the cpu will never get idels, because with the help of Multiprogramming we can execute many programs on the System and when we are working with the program then we can also submit the second or another program for running and the CPU will then execute the second program after the completion of the first program. And in this, we can also specify our input means a user can also interact with the system.



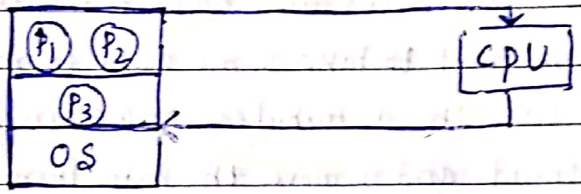
- Increased CPU Utilization.
- Increased throughput.

Saathi

Date ___ / ___ / _____

4. Multitasking operating System: Multiple applications ~~app~~ operates at the same time.

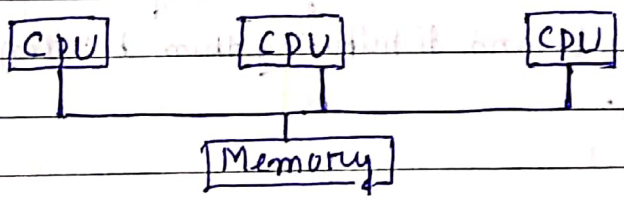
- Multitasking is an extension to Multiprogramming.
- The jobs will be executed in the time sharing mode.



(We see illusion)

Memory (Multitasking)

5. Multiprocessing operating System: In the Multi processing there are two or more cpu in a single operating system, if one cpu will fail, then other cpu is used to providing backup to the first cpu. With the help of ~~put~~ multiprocessing, we can execute many jobs at a time. All the operations are divided into the number of cpu's. If first cpu completed his work before the second cpu, then the work of second cpu will be ~~b~~ divided into the first and second cpu.



(Multiprocessing)

6. Realtime Operating System: In this, Response time is already fixed. Means time to display the results after processing has fixed by the processor or CPU. Real time system is used at those places in which we requires higher and timely response.

- Hard Real-time System: In the hard real-time system, time is fixed and we can't change any moments of the time of process the data as we enters the data.
- Soft Real-time system: In the soft real-time system, some moments can be changed. Means after giving the command to the CPU, CPU performs the operation after a microsecond.

Date __/__/____

7. Distributed Operating System: Distributed Means data is Stored and processed on Multiple locations. When a data is stored on to the multiple computers, those are placed in different locations. Distributioned means in the network, Network collections of computers are connected with each Other. Then If you we want to take some data from other computer, then we uses the distributed processing system. And we can also insert and remove the data from out location to another location. In this Data is shared between many users. And we can also Access all the input and output devices are also accessed by Multiple users.

8. parallel Operating System: These are used to interface multiple networked computers to complete task in parallel. parallel OS are able to use Software to manage all of the different resources of the computers running in parallel, such as memory, caches, storage space, and processing power. A parallel OS works by dividing sets of calculations into smaller parts and distributing them between the machines on a network.

Saathi

Date / /

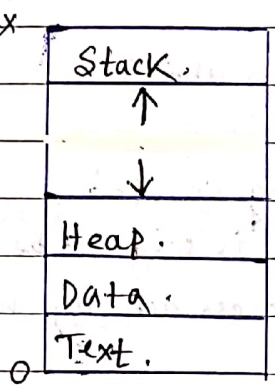
PROCESSES - (Process management)

RBR

- **Process:** A process can be defined in any of the following ways:
  - A process is a program in execution.
  - It is a asynchronous activity.
  - It is the entity to which processors are assigned.
  - It is dispatchable unit.
  - It is the unit of work in a system.

A process is more than the program code. When we execute a program, it becomes a process which performs all the task mentioned in the program.

When a program is loaded into memory and it becomes a process, it can be divided into four sections - stack, heap, text and data. The following image shows a simplified layout of a process inside main memory -



(process in memory)

- **Stack:** The process stack contains the temporary data such as function parameters, return address, and local variables.
- **Heap:** This is dynamically allocated memory to process during its run time.
- **Data:** This section contains the global and static variable.
- **Text:** This includes the current activity represented by the value of program counter and the contents of the processo's register. (Code section)

Saathi

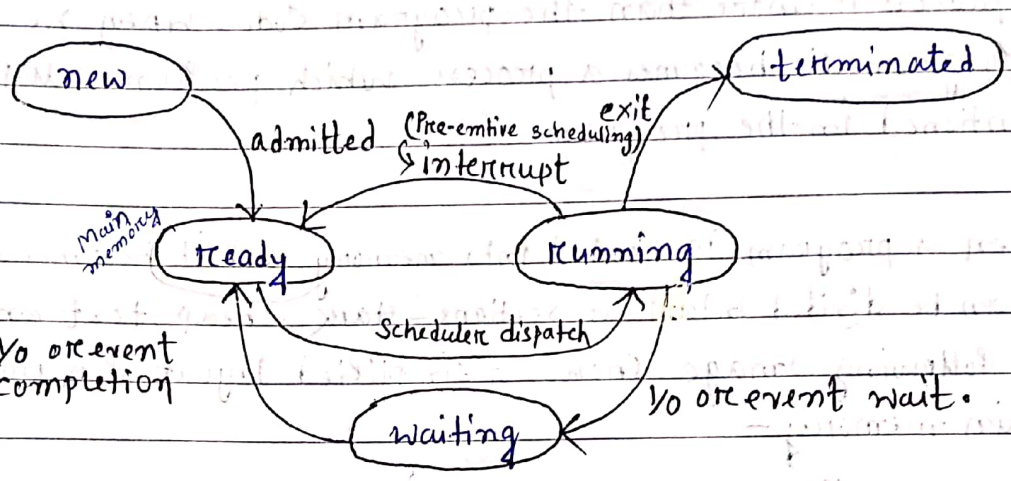
Date __/__/__

RBR

Process State Diagram:

When a process executes, it passes through different states. This stages may differ in different OS, and the name of these states are arbitrary.

A process can have one of the following five states at a time.



(Diagram of process state)

1. Start: The process is being created.
2. Running state: A process is said to be running if it has the CPU, that is, process actually using the CPU at that particulate instant.
3. Waiting (or Blocked) state: A process is said to be blocked if it is waiting for some event to happen such that as an I/O completion before it can proceed. Note that, a process unable to run until some external event happens.

4. Ready: A process is said to be ready if it use a CPU if one where available. A ready state process is runnable but temporarily stopped running to let another process run.
5. Terminated State: The process has finished execution.

Saathi

Date ___ / ___ / ____

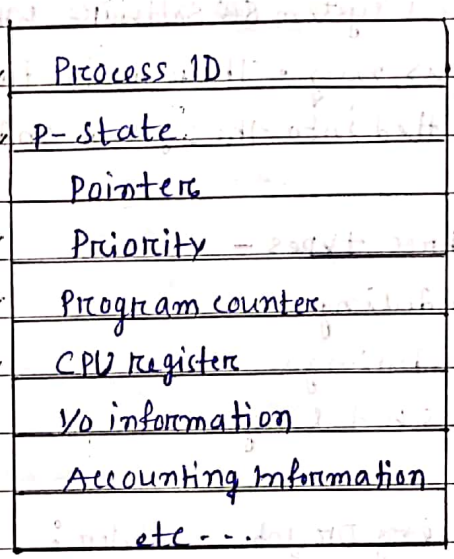
RBR

Process Control Block (PCB):

A process in an OS is represented by a data structure known as process control block (PCB) or process descriptor.

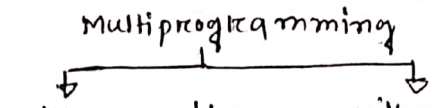
A PCB keeps all the information needed to keep track of a process.

The architecture of PCB is completely dependent on operating system and may control different information in different OS.



(PCB Diagram)

1. process State: The current state of process i.e, whether it is ready, running, waiting, or whatever.
2. process privileges: This is required to allow/disallow access to system resources.
3. process ID: Unique identification for each of the process in OS.
4. pointer: A pointer to parent process.
5. program Counter: PC is a pointer to the address of the next instruction to be executed for this process.
6. CPU registers: various CPU registers where process need to be stored for execution for running state.
7. Accounting information: This includes the amount of CPU used for process execution, time limits, execution ID etc.
8. I/O status information: This includes a list of I/O devices allocated to the process.
9. CPU-scheduling information: process priority and other scheduling informatio



Date __/__/____

Saathi

which is required to schedule the process.

10. Memory management information: This includes the information of page table, memory limits, segment table depending on memory used by the ~~oper~~ OS.

RBR

Schedulers:

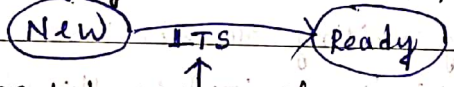
Schedulers are special system ~~software~~ software which handle process scheduling in various way. Their main task is to select the jobs to be submitted into the system and to decide which process to run.

Schedulers are of three types -

1. long Term scheduling.
2. short Term scheduling.
3. Medium Term scheduling.

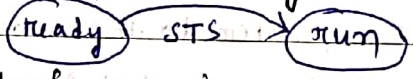
1. Long Term Scheduler or job scheduler:

LTS is responsible for creating new processes and bringing them into System. LTS select processes from job pool (where processes are kept for later execution) and load them into memory for execution. LTS control degree of Multiprogramming.



2. Short term Scheduler or CPU scheduler:

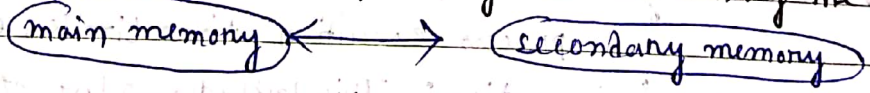
STS ~~is~~ is responsible for scheduling one of the process from ready sate to running State:



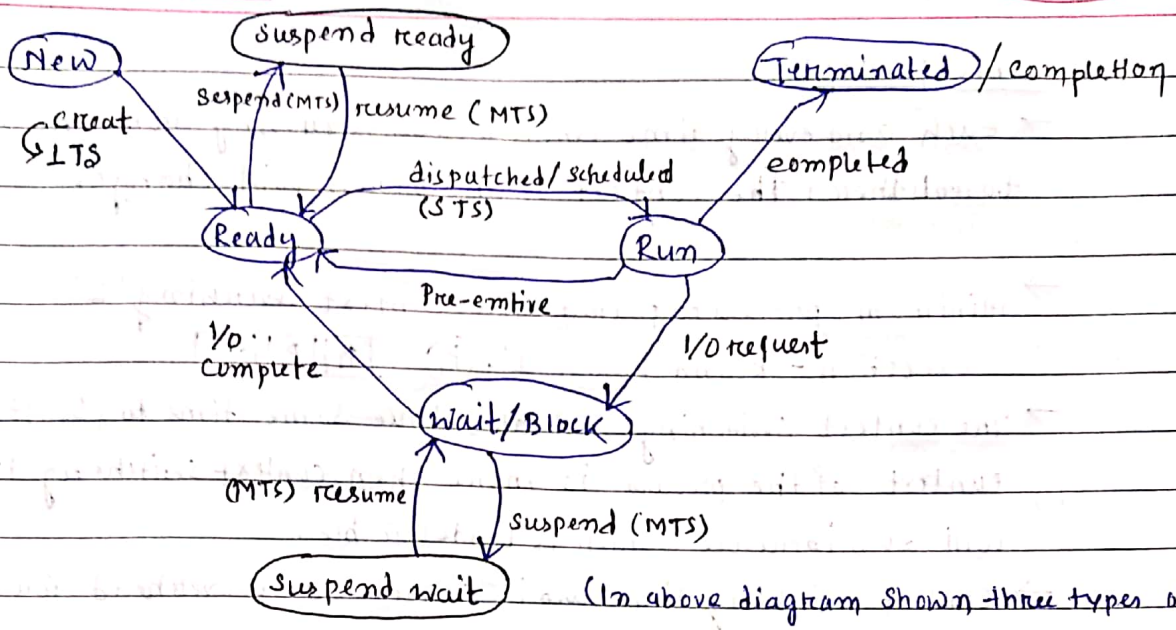
STS selects from main memory among the process that are ready to execute and allocate the CPU to one of them.

3. Medium Term scheduler:

MTS is responsible of suspending and resuming the processes.



The medium term scheduler are also called a SWAPPER.



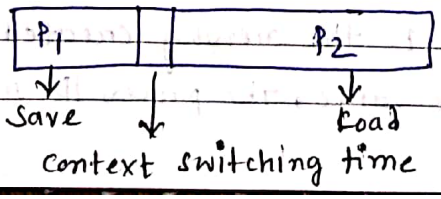
(In above diagram shown three types of schedulers)

Aspects :	LTS	MTS	STS
called as	It is Job scheduler	It is a process swapping	It is a cpu scheduler
Speed	Spee is less than STS	Speed is in between both LTS and STS.	Speed is fastest among two Schedulers.
Multiprogramming	It Control the Degree of Multiprogramming	It reduce the Degree of multiprogramming.	It provides less control over Degree of multiprogramming
Time-sharing System.	It is almost absent or minimal in time sharing System.	It is part of Time sharing System.	It is also ~~nominal~~ minimal. in time Sharing System.
processes	It Selects process from pool and loads them into memory for execution.	It can reintroduce the process into memory and execution can be continued.	It selects those process which are ready to execute.

PBR

Context Switching :

Saving the context of 1 process and loading the context of another process is called Context Switching.



Saathi

Date __/__/____

Some points:

→ Each and every time when process is moving from one state to another, the context of the process will change.

→ Minimum process required for context switching '2'.
Exception - Round Robin '1'. (P1) [P1 | P1 | ..]

→ The context switching will also take some time so, so if the context of the process is more then context switching time will also increase which is undesirable.

→ Context switching time also is considered as overhead for the system.

• Dispatcher:

Dispatcher is responsible for saving the context of 1 process and loading the context of another process. The context switching is done by dispatcher.

RBR • The fork():

→ System call fork() is used to creat processes. It takes no arguments and returns a process ID.

→ The purpouse of fork() is to create a new process, which becomes the child process of the caller.

→ After a new child processes is created, both processes will executed the next instruction following the fork() system call.

→ Therefore, we have to distinguish the parent from the child.

This can be done by testing the returned value of fork():

→ • If fork() return a negative value, the creation of a child process was unsuccessful.

→ • If fork() return a zero to the newly created child process.

→ • If fork() return a positive value, the process ID of the child process.

to the parent. Normally the process ID is an Integer. Moreover, a process can use function ~~getpdf~~ getpid() to retrieve the process ID assigned to this process.

• Conclusions –

→ If the program contains n fork() calls it will creat $2^n - 1$ child processes.

→ When the child process is created by using fork system call, both parent and children will have.

→ Relative address = Same for both parent and child process.

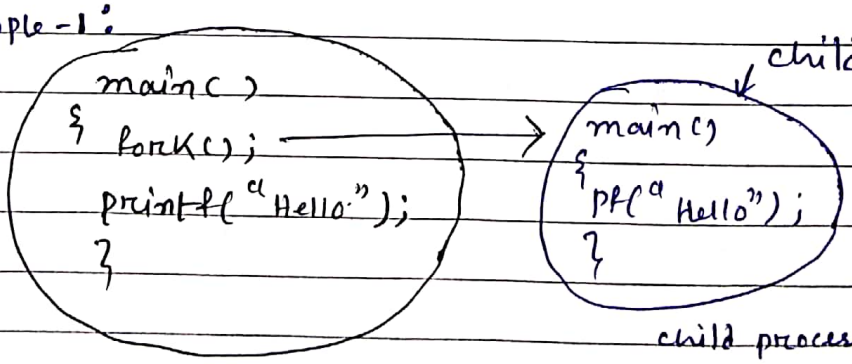
→ Absolute address = Different for both parent and child proces.

• fork() system call Implementation –

```
main
{
 int pid;
 pid = fork();
 if (pid<0)
  { pf("child process creation failed");
 else if (pid == 0)
  { pf("child process");
  }

 else
  { pf("parent process");
  }
}
```

• Example-1:



output: Hello
Hello

2 process / 2 time Hello. / 1 child ($2^n - 1 = 2^1 - 1 = 1$).

Saathi

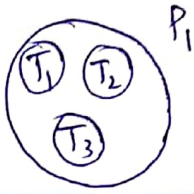
Date ___ / ___ / _____

RBR

Q. Question on process states —

Consider a system with 'N' CPU processors and 'M' processes, then,

	min	max
ready	0	M
running	0	N
block or wait	0	M



Date __/__/____

THREADS

Thread:

A Thread is a single sequence stream within a process. Because threads have some of the properties of processes, they are sometimes called lightweight processes.

→ A thread can be any several States (Running, Blocked, Ready, or Terminated)

→ Each thread has its own stack.

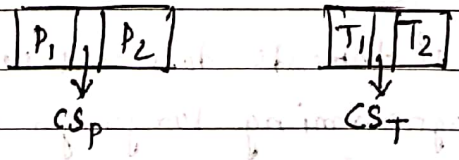
→ A thread has or consists of a program counter (PC), a register sets, and a stack space.

→ Threads are not independent of one other like processes a result threads shares with other threads their code section, data section, OS resources also known as task, such as open files and signals.

Advantages of Thread:

1. Responsiveness:
   → If the process is divided into multiple threads, then 1 thread is completed, then immediately output will be responded.

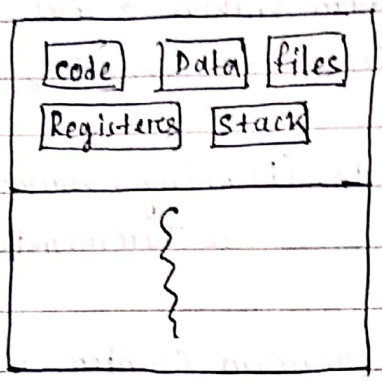
2. Faster context switch:
   → The context switching time between the threads is very very less as compared to context switching time between the process. Because the threads will have less context compared to process.



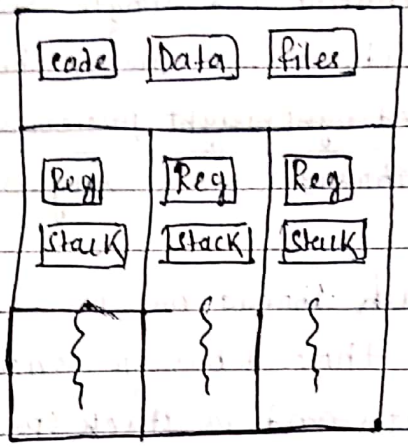
$$CS_T << CS_P$$

Date __/__/____

3. Resource Sharing:



single threaded process



Multi threaded process

→ The resources like code, data, and files will be shared among all the threads with in the process but every thread will have its own stack and registers.

4. Effective Utiligation of Multiprocessor Systems:

→ If the process is divided into multiple threads, then different threads of the process can be scheduled onto a different CPU, so that the process execution will be faster.

5. Enhanced throughput of the System:

→ If the process is divided into multiple threads, if we consider 1 thread as 1 job then number of job completed per unit time will increased and throughput of the system will be enhanced.

6. Economical:

→ The implementation of threads does not requires any cost. There are various programming languages API's which support implementation of thread e.g JAVA's API, thread API

Saathi

Date / /

• Types of Threads:

Threads are categoriyed into 2 types,—

→ User level threads.
→ Kernel level threads.

• Difference between them—

User level thread	Kernel level thread
→ These are created by user or programmers.	→ These are created by operating System (OS).
→ OS can't recognige the user level threads.	→ OS recogniges the Kernel level threads.
→ If one user level thread performing Block system call, then entire process will be blocked.	→ If one Kernel level thread performing Blocking System call, then ~~oth~~ another thread will continue the execution.
→ Dependent threads.	→ Independent threads.
→ Designing user level threads is easy.	→ Designing Kernel level threads is complicated.
→ Less context.	→ More Context.
→ NO H/W support required.	→ Scheduling of Kernel level threads requires H/W support.

*. Thread scheduling also known as GANG scheduling.

• There are three common ways of establishing relationship between user threads and kernel threads—

→ Multi ... are ~~two~~ three types—
- Many-to-many model.
- one-to-one model.
- Many-to-one model.

Saathi

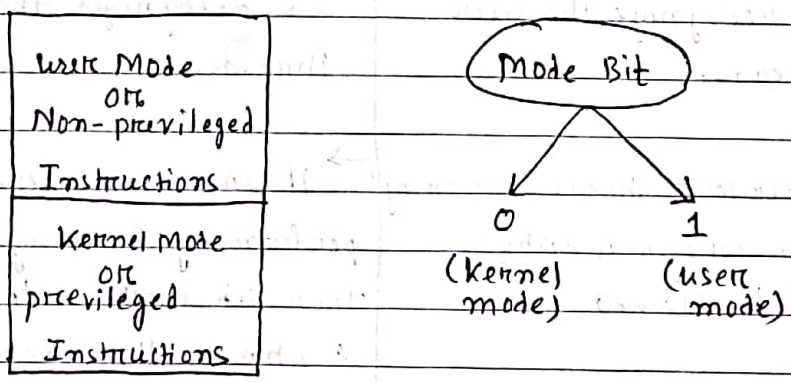
Date ___ / ___ / _____

1. Many-to-many model -
→ Allows many user level threads to be mapped to many kernel threads.

2. One-to-one model -
→ Each user level threads maps to kernel thread.

3. Many-to-one model -
→ Many user level threads mapped to single kernel thread.

Double Mode of Operation :



→ The o.s executes the instructions in 2 different modes -
1) User mode.
2) Kernel mode.

→ The dual mode of operation is required to protection and security to the o.s and also to the user programs from errant users (corrupted users).

→ In which particular mode the current instruction is executing will be decided by mode bit. → 0 / 1

→ At boot time the system always starts in the kernel mode.

→ Depending upon the type of instruction the o.s will decide in which particular mode instruction has to be executed.

Saathi

Date ___ / ___ / _____

→ Generally privileged instructions will be executed in the Kernel mode, and Non-privileged instructions will be executed in the User mode.

- privileged Instructions: (examples)
  1) Context Switching.
  2) Disabling Interrupts.
  3) set the time of clock.
  4) Changing the memory map.
  5) I/O operation (Reading/file data from disc)
- Non-privileged instructions: (examples)
  1) Reading time of the clock.
  2) Sending the final print to the printer.
  3) Reading the status of the CPU.

DONE

Date ___ / ___ / _____

Date ___/___/___

Inter-Process Communication

→ process executing concurrently in the operating system may be either independent or cooperating processes.

→ A process is independent; if it can't affect or be affected by other processes executing in the system.

→ A process ~~is cooperating process, if~~ that shares data with other processes is a cooperating process.

→ Advantages of process cooperation are –
  → Information sharing.
  → Computation speed up.
  → modularity and convenience to work on ~~may~~ many task at ~~the~~ same time.

→ Cooperating processes require an inter process communication (IPC) mechanism that will allow that to exchange data and information.

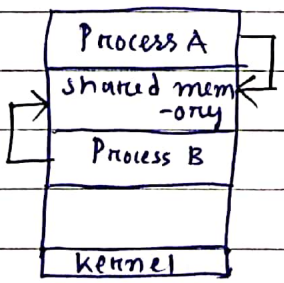
→ There are 2 fundamental models of IPC:
  i) Shared Memory.
  ii) Message passing.

1. Shared memory:

→ In this method 2 or more processes shared a single chunk of memory to communicate.

→ used for exchanging larg amount of data.

→ faster than message passing.



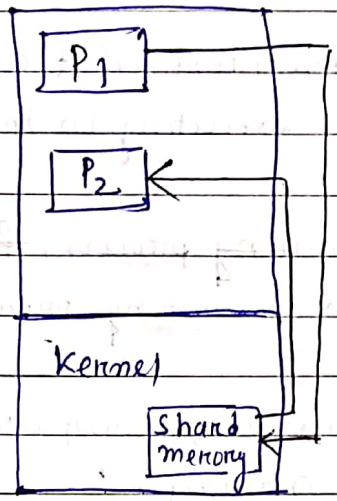
(Shared memory)

Date ___ / ___ / ____

2. Message passing:

→ shared memory created in the kernel.

→ System call such as send and receive used for communication.



→ used for exchanging small amount of data.

→ slower than shared memory.

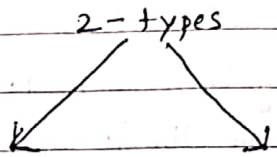
(Message passing)

Saathi

Date ___/___/_____

PROCESS SYNCHRONIZATION

*. The processes with respect to synchronization are of 2-types



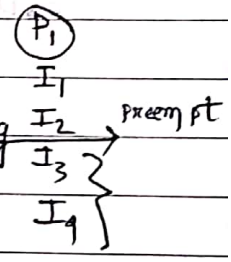
Co-operating process — Independant process.

The execution of 1 process affects or affected by other process then these processes are said to be co-operative processes, Other wise they are said to be independant process.

→ Affection occurs due to shared variable, (common shared Resource / Data.

- Some Import points:

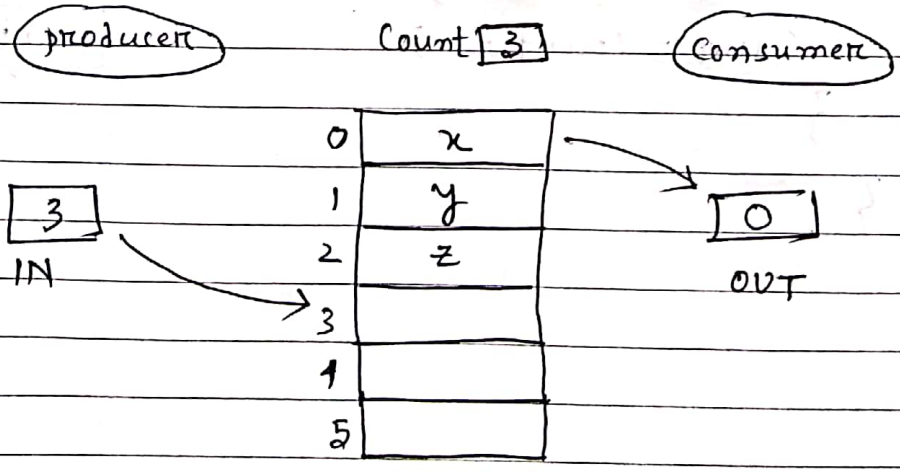
→ The pre-emption is just a temporary stop, the will come back and continue the execution.



→ If there is any possibility of solution becoming wrong by taking the pre-emption.

→ If any solution has the be deadlock, then the progress is not satisfied.

- producer-consumer problem:



Date

IN → is a variable used by producer to Identify the next empty slot in the Buffer.

OUT → is a variable used by Consumer to Identify the slot from where it has to consume the item.

Count → is a variable used by consumer and producer to identify the no of items present in the Buffer at any point of time.

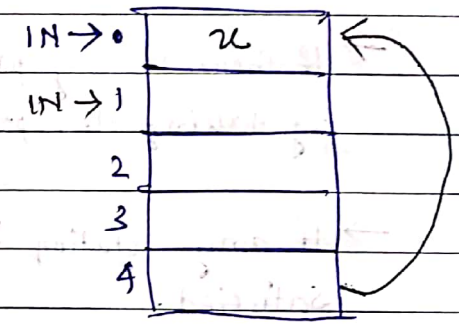
Conditions :-

1) → When the buffer is full, the producer is not allowed to produce the item.

2) → When the buffer is empty consumer is not allowed to consume the item.

• Producer Code →

```
int count = 0;
void producer (void)
{
  int temp;
  while (TRUE)
  { produce_item (Itemp);
    while (count == N);
    Buffer [IN] = Temp;
    IN = (IN+1) MOD N;
    Count = count + 1;
  }
}
```



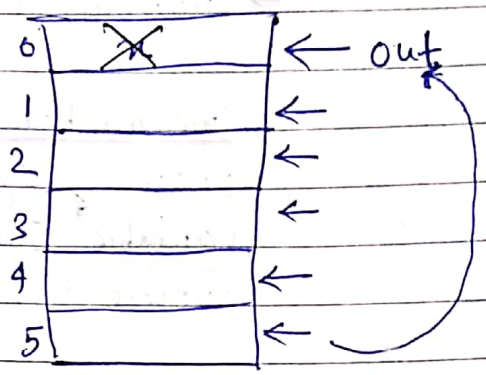
n = 5

count = 1

Date ___/___/_____

• Consumer Code →

```
void consumer(void)
{
   int itemc;
   while(TRUE)
   {
      while (count==0);
      Item c = Buffer[OUT];
      OUT=(OUT+1) MODN;
      count = count -1;
   }
}
```



n = 6

Count = 1 / 0

• producer Consumer problem Analysis:

⊛ producer

count = count + 1;

⇓

I. LOAD R_p, m[COUNT];
II. INCR R_p
III. STORE m[COUNT], R_p

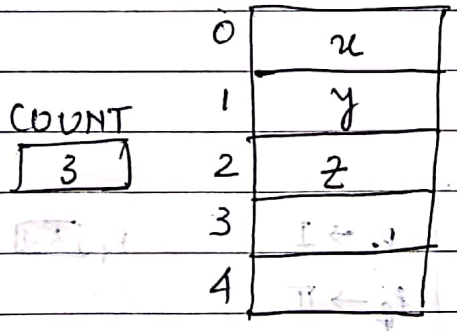
⊛ Consumer

count = count - 1;

⇓

I. LOAD R_c, m[COUNT]
II. DECR R_c
III. STORE m[COUNT], R_c

Analysis – **Race around condition**



{different count value for Producer and consumer}

Now,

P → I R_p 3
P → II R_p 4
preempt
C → I R_c 3
C → II R_c 2
✓ C → III 2 ✓
✓ P → III 4 ✓

Saathi

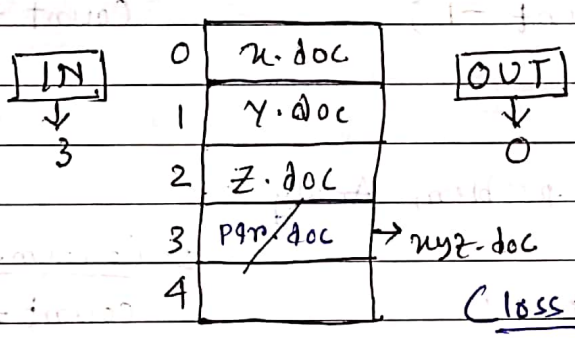
Date ___ / ___ / _____

- producer Consumer problem Conclusion –

→ problems of Non-Synchronigation.

Inconsistency :- The producer and consumer are not properly Synchroniged while ~~sharing~~ shearing the common variable COUNT. Hence it is leading inconsistency. No proper Synchronigation between producer and consumer

- pointer - Spooler Domain problem –



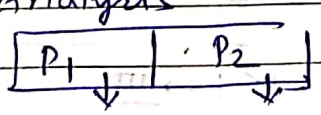
(IN) → is a variable used by all processes to identify the next empty place in the spooler directory.

(OUT) → is a variable used ~~by~~ only by printer to identify the slot from where it has to print the document.

Enter file –

1) load R_i, m[IN]
2) STORE S_0[R_i], "file name".
3) INCR R_i.
4) STORE m[IN], R_i.

Analysis –



$P_1 \rightarrow$ I R_1 [84]
$P_2 \rightarrow$ II
$P_1 \rightarrow$ III → preempt
$P_2 \rightarrow$ I R_1 [3]
$P_2 \rightarrow$ II

Date

- problem of Non-Synchronigation:

→ Inconsistency: The producer and consumer are not properly Synchroniged while sheaning the common variable COUNT. Hence it is leading to inconsistency. No proper synchronigation between producer and consumer.

→ Data Loss of Data: The process are not properly synchroniged while sharing the common variable [IN], hence it is leading to lose of data.

→ Deadlock: If the process are not properly synchroniged the there is also a possibility of deadlock.

- Definations:

1) Critical Section (C.S): The portion of program text where sheared variables are placed.

example – In producer/consumer problem:

count = count + 1 → C.S

2) Non-Critical section: The portion of program text where the Independent code of the process will be placed.

3) Race Condition: The final value of any variable depends on the execution sequence of process. This condition is called as Race condition.

- The Critical Section ~~Problem~~ – Structure –

```
do
{
    entry section
        critical section.
    exit section
        reminder Section / NCS.
}
while (TRUE);
```

Saathi

Date __/__/____

- Synchronization conditions :-

1) MUTUAL EXCLUSION: No two process may be simultaneously present inside the critical section at any point of time. Only 1 process is allowed into critical section at any point of time.

2) progress : No process running outside the critical section should block the other interested process from entering into c-s when the c-s if free.

3) Bounded Waiting : No process should have to wait forever from entering into critical section. There should be a bound on getting chance to enter into critical section. If Bounded waiting is not satisfied, then it is possible for starvation.

- Solution Types :

1) Software Type :-
   a. Lock variables.
   b. strict Alternation (or) Decker's Algorithm.
   c. peterson's Algorithm.

2) Hardware type :-
   a. TSL Instruction set (Test & set lock)

3) O.S Type :-
   a. Counting Semaphore.
   b. Binary semaphore.

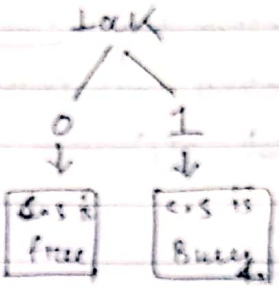
4) programming language compiler support type :-
   a. Monitors.

Date

Lock variables : → M·E → X, Progress → ✓, B·W → X

Entry section -

I. load R_i, m[lock]
II. CMP R_i, #0
III. JNZ tostep①
IV. STORE m[lock] #1
V. CS
VI. STORE m[lock], #0



let, lock ⇒ 0 X 1

R·Q | P_1 | P_2

$P_1 \to$ I R 0
$P_1 \to$ II
$P_1 \to$ III → preempt.

$P_2 \to$ I R 0
$P_2 \to$ II
$P_2 \to$ III
$P_2 \to$ IV, V

$P_1 \to$ IV,
$P_1 \to$ V

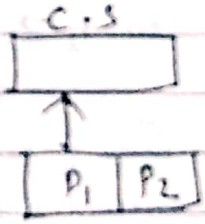
C·S: P_2, P_1 → M·E-X

✓ (Mutual exclusion not satisfy) - ✗

Entry section.

I. LOAD R_i, m[lock]
II. CMP R_i, #0
III. JNZ tostep①
IV. STORE m[lock] #1.
V. CS } critical section
VI. STORE m[lock] #0. } remainder section

} entry section



Lock = 0 / 1

$P_2 \to$ I R 0
$P_2 \to$ II, III
$P_2 \to$ IV → preempt

$P_1 \to$ I R 1
$P_1 \to$ II, III

✓ process one satisfy ✓
✓ Bounded Waiting - ✗
(If process one finish then B·W satisfy other wise not satisfy)

(Here no. of process infinite)

Date ___/___/_____

- Strict Alternation (or) Decker's Algorithm :- M.E → ✓, P → X, B.W → ✓

→ process takes turn to enter into critical solution.

→ solution will work for 2 - process.

(P0) (P1)

turn

0 → P0 (entry)

1 → P1 (entry)

$P_0 \rightarrow 0$	$P_1 \rightarrow 1$
while (TRUE)	while (TRUE)
{	{
Non C.S ;	Non C.S ;
while (turn != 0) ;	while (turn != 1) ;
C.S	C.S
turn ⇒ 1 ;	turn ⇒ 0 ;
}	}

Analysis (for M.E, P, B.W) —

turn ⇒ 0

C.S

P0 → (M.E) ✓

for finite process → (B.W) ✓

turn → Ø, 1, 0

P0, P1, (P1)

here P0 not interested to go C.S but for turn=0 P1 can't go C.S. (P0 stored P1) → (Progress) ✗

- 2nd defination of progress :-

Which process will go next into C.S is decided by only those process, who want to go into CS. I this decision the process in the remainder section and the process which is not interested to go into the C.S should not take participation.

Saathi

Date ___/___/_____

Peterson's Algorithm :- (ME → ✓, P → ✓, B.W → ✓)

(2 process solution)

```
#define N 2
#define TRUE 1
#define FALSE 0
int turn;
int interested[N];
void enter_Region(int process)
{
  1. int other;
  2. other = [1-process];
  3. Interested[process] = TRUE;
  4. turn = process;
  5. while (turn == process && interested[other] == TRUE);
       CS;
}

void leave_Region(int process)
{
  interested[process] = false;
}
```

Initially –

- Interested[0] = false.
- Interested[1] = false.

P0 → other = 1 ; P1 → other = 0

Analysis :-

(next page)

Saathi

Date ___ / ___ / ____

P_0	P_1
1.	1. →
2. other → 1-0 = 1	2. other = 0
3. Interested[0] = True.	3. Interested[1] = True.
4. turn = 0	4. turn = 1
5. → CS [P_0]	5. Loop CS [P_0]

So, M.E = ✓.
progress = ✓
Bounded waiting = ✓

* peterson practise Question :-

Code –

```
void enter_Region()
{
1. int other ;
2. other = 1- process ;
3. Interested[process] = True;
4. turn = process ;
5. while (turn == process && Interested[other] == True);
6. c.s ;
}
```

P_0	P_1 ✓
1. →	1. →
2. → other = 1	2. other = 0
3. → Interested[0] = True	3. Interested[1] = True.
4. → turn = 0;	4. turn = 1;
	5. → C.S [P_1]

Q. If both the process P_0 and P_1 are trying to enter in cs at same time, then which process will go into cs first.

a. The process which executes statement 2 first.
b. " " " " " 3 " .
c. " " " " " 4 " .
d. We cannot say.

→ Ans → (c).

Saathi

Date ___ / ___ / _____

- TSL Introduction: M.E → ✓, P → ✓, B.W → X

→ TSL ~~flag~~ Register flag :- Copies the current value of flag into register and store the value of 1 into the flag in a single atomic cycle, without any preemption.

→ Disabling the interrupt is at H/W level that is why it is called H/W approach.

→ Entry section –

flag

0 — CS is free

1 — CS is busy

*1) TSL R_i, m[flag]
2) cmp R_i, #0
3) JNZ to step (1).
4) C.S
5) STORE m[flag], #0

Analysis –

flag = 0 / 1

RQ = [P_0 | P_2]

P_1 → I R_1 [0]

P_1 → II

P_1 → III

P_1 → IV → pre-empt CS [P_1] M.E → ✓

P_2 → I R_2 [1]

P_2 → II, III

[P_1 | P_2] flag = 0 / 1

CS [P_1]

P_1 → I R_1 [0] → preemptive

P_2 → I R_2 [1] progress → ✓

P_2 → II, III

Where no. of process infinite so, B.W → X.
(Bound, waiting)

Date

* Some Important points :- (TSL)

→ In Lock variables mutual exclusion is not satisfied because loading and storing were not done in same step.

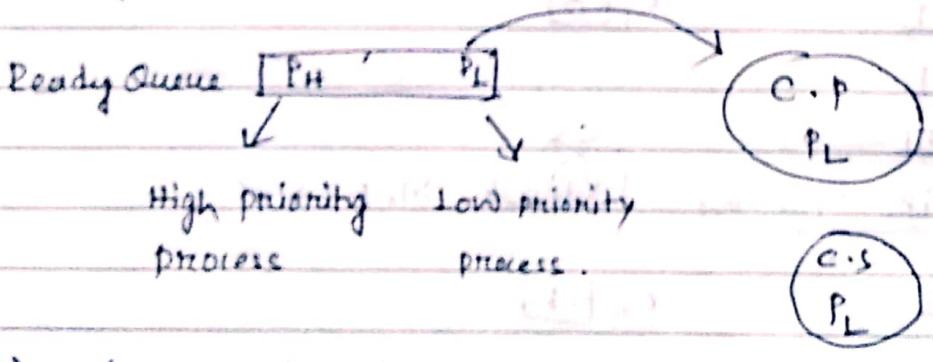
→ progress is satisfied in TSL.

→ Bounded waiting is not satisfied Because no. of processes is not countable.

** Summary :-

Solution	Mutual exclusion	progress	Bounded waiting
Lock variable	X	✓	X
Strict Alternation	✓	X	✓
Peterson's Solution	✓	✓	✓
TSL	✓	✓	X

• Priority Inversion problem —



→ P_L is currently executing CS code. Now suddenly P_H coms, then we are going to pre-empt P_L. ~~P_H is even~~ though P_H is scheduled by CPU but it can not enter in C.S because it is locked by P_L.

↓

Problem is called Live-Lock

↓

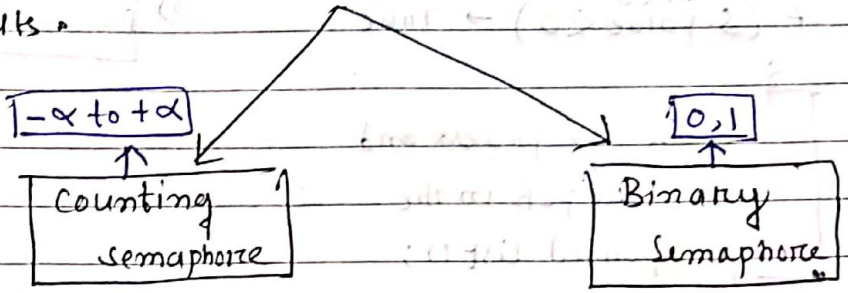
Solution → Priority Inversion / Inheritance.

Date ___ / ___ / _____

→ This allows that P_1 can continue its execution in c.s and then Leave.

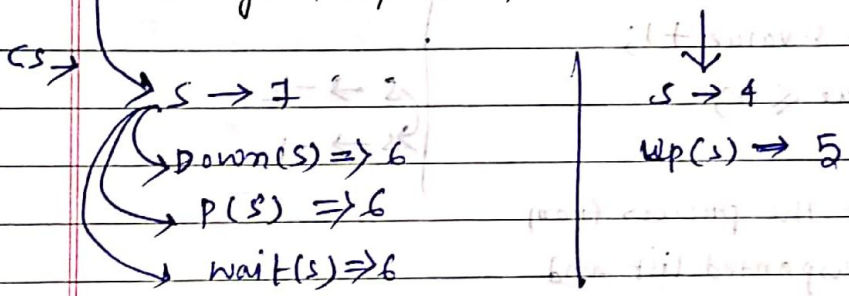
*. Semaphore :

→ is an Integer variable which is used by various process in a mutual exclusive manner to achive Synchronigation. The ~~proper~~ Improper usage of semaphore will be given Improper results.



Two different operations performed on semaphore variable are –

1) Down Operation on P() or wait().
2) signal Operation on V() or release() or UP().



bs ⇒ 1
down(bs) ⇒ 0

bs ⇒ 0
up(bs) ⇒ 1

Date ___ / ___ / _____

Counting Semaphore –

→ **Down operation –**

```
Down operation (semaphore S)
{
  S.value = S.value - 1;
  If (S.value < 0) → True
  {
    Block the process and
    Place the PCB in the
    suspended list ();
  }
}
```

X

S → 2
1) Down (S) → 1
2) Down (S) → 0
3) Down (S) → -1

→ **Up operation –** (always successful)

```
Up operation (semaphore S)
{
  S.value = S.value + 1;
  If (S.value ≤ 0)
  {
    Select the process from
    suspended list and
    wakeup ();
  }
}
```

S → 2
S → 3

S → -2
S → -1

→ After performing the down operation, if the process is getting suspended, then it is called unsuccessfull down operation.

→ After performing the down operation, if the process is not getting suspended, then it is called successfull down operation.

Date ___/___/_____

→ If it is successfull down Operation, then only the process will continue the execution. The down operation on the counting semaphore is successfull only if the semaphore value is ≥ 1

→ There is no unsuccessfull up operation, the up Operation is always successfull because it does not suspend any process.

→ If $S = +6$ it means we can perform 6 successfull down Operation.

→ If $S = -6$ it means that there are already 6 suspended processes.

*Question -

Consider a system where a counting semaphore value is initialiged to +17.

The various semaphore operations like-

23P, 18V, 16P, 14V, 1P are performed.

what is the final value of semaphore.

$\Rightarrow 17 - 23 + 18 - 16 + 14 - 1$

$\Rightarrow 9 - 6 + 2 + 13$

$\Rightarrow (7)$

Binary semaphore: 0 ↔ 1

→ Down Operation

```
Down (Semaphore S)
{
  if (S.value == 1)
    S.value = 0;
  else
  {
    Block the process and
    place the PCB in the
    Suspended list();
  }
}
```

→ Up Operation

```
up(semaphore S)
{
  if (suspended list() is empty)
    S.value = 1;
  else
  {
    Select a process from
    suspended list and WeakUp()
  }
}
```

→ After performing down Operation if the process is getting suspended then it is called unsuccessfull down Operation.

→ After performing down Operation, if the process not getting ~~never~~ suspended then it is called successfull down Operation.

→ The down Operation on the Binary semaphore is successfull only if the semaphore value is 1.

→ If it is successfull down operation, then only the process will continue the execution.

→ There is no unsuccessfull up operation. The up Operation will be always successfull. The process performing the up Operation will definately continue the execution.

Question : (I) (II) (III)

Each process $P_i = 1$ to 9 executes the following code -

```
repeat
    P(mutex);
      c.s
    V(mutex);
forever
```

The process P_{10} execute the following code.

```
repeat
    V(mutex);
      c.s
    V(mutex);
forever
```

⇒ The initial value of Binary semaphore mutex = 1.

⇒ What is the maximum no. of process may be present inside the cs at any point of time ?

mutex ⇒ 1

mutex → 1 0 1 0 1 0 1 0

P_1	P_{10}	P_2	P_3	P_4	$P_5 \cdots P_9$

c.s ↓

→ Ans : (10) process.

P(mutex); then find ?

→

mutex = 1 0 1 0

P_1	P_{10}	P_2

c.s ↓ — 3(P).

→ Ans : (3) process.

p(mutex); then find ?

mutex = 1 0

P_1

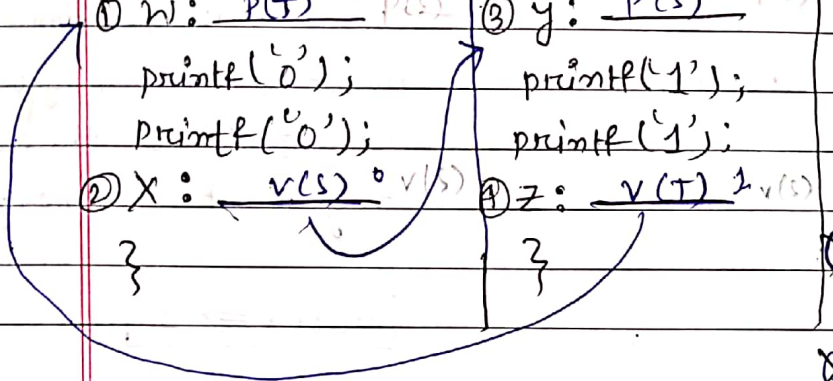
c.s

→ Ans : (1) process.

- GATE-2003 Question :-

Consider the two concurrent process 'P' and 'Q' executing their ~~ex~~ respective codes.

process 'P' code	process 'Q' code
while (TRUE)	while (TRUE)
{	{
① W : P(T)	③ y : P(S)
printf('0');	printf('1');
printf('0');	printf('1');
② X : V(S)	④ Z : V(T)
}	}



what should be the semaphore operations on W, X, Y, Z respectively. And what should be the initial value of Binary semaphore 'S' and 'T' inorder to get the output as 0011 0011 00 ...

(a) W = P(T), X = V(T), Y = P(S), Z = V(S), S = T = 1 ;

(b) W = P(T), X = V(T), Y = P(S), Z = V(S), S = 1, T = 0 ;

(c) W = P(T), X = V(S), Y = P(S), Z = V(T), S = 1, T = 1 ;

(d) W = P(T), X = V(S), Y = P(S), Z = V(T), S = 0, T = 1 ;

→ PQPQ ...

- Question :- process code same.

Which of the following will ensure that the output string never contains a substring of the form 01^n0 or 10^n1 where n is odd.

(a) W = P(S), X = V(S), Y = P(T), Z = V(T), S = T = 1 ;

(b) W = P(S), X = V(T), Y = P(T), Z = V(S), S = T = 1 ;

(c) W = P(S), X = V(S), Y = P(S), Z = V(S), S = 1 ;

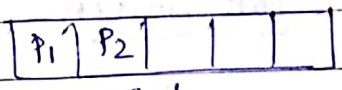
(d) W = P(S), X = V(T), Y = P(S), Z = V(T), S = T = 1 ;

S →

→ 0011 0011 ...

- Producer & consumer problem solve with the help of semaphore.

semaphore S = 1
semaphore E = n here n = is of size of buffer
Semaphore F = 0

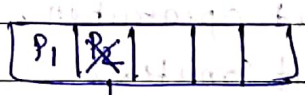


C.S

```
* void producer()
  {
     While(T)
     {
        produce()
        wait/down(E)
        wait(s)/down(s)
        append()
        signal(s)/up(s)
        signal(F)/up(F)
     }
  }
```

Let S = 1 0 X 0 1
E = ~~5~~ ~~4~~ 3
F = 0 1 2

```
* void consumer()
  {
     While(T)
     {
        wait(F)/down(F)
        wait(S)
        take()
        signal(s)/up(s)
        signal(E)
        use()
     }
  }
```



S = 1 0 1
E = ~~3~~ 3 4
F = 0 1 1

- READ-WRITE PROBLEM:
  (Solve by Simaphorce)
  (there ~~using~~ used 3 semaphore).

for writer

I. wait(wrt) (1)	wrt = 1
II. write operation	
III. signal(wrt)/up(wrt).	

for read

I. wait (mutex) (1)	I. mutex = 1
II. readcount ++	II. readcount = 0
III. If (readcount == 1) wait(wrt)	III. wrt = 0 (no writer can enter in critical section).
IV. signal (mutex)	
V. read operation.	
VI. wait (mutex)	
VII. readcount --	
VIII. If (readcount == 0) signal (wrt)	
IX. signal (mutex).	

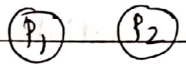
*. At a time more than one read operation done in critical section.

*. At a time only one write operation done in C.S.

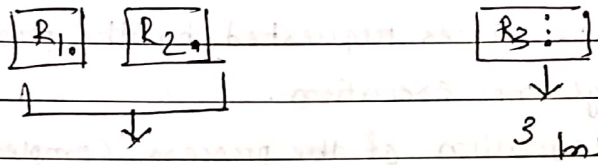
DEADLOCK

- Defination:- If two or more processes are waiting for some event to occure, which never happens, the processes are said to be involved in deadlock.
- Resource Allocation Graph (RAG):

→ Basics:- process will be represented with circle.



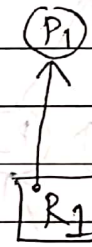
Resources will be represented with rectangle.



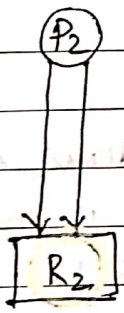
*. Requesting edge

process P_1 is requesting 1 Instance of R_1

*. Allocation edge.



one Instance of 'R_1' is allocated to 'P_1'.



process P_2 is requesting for 2 instences of 'R_2'.

Two Instance of 'R_2' is allocated to 'P_2'.

→ Resource request and Resource allocation will be ~~requested in the~~ represented in the resource allocation graph (RAG).

$(R.A.G) = G(V, E)$

V → processes & Resources. (vertex)

E → Requesting and Allocation edge.