ROLL NUMBER:- 18115069

SUBSECT :- OPERATING SYSTEM

DATE :- 23/11/2020

SUBJECT CODE:- CS2051(CS)

NAME:- SANJEEB NATH

1.) a) How the problem - - - - - - - - - - - -

b) The abilities - - - - - - - - - - - - - -

Ans) (a) single-processor systems :-

Most systems use a single processor. The variety of single processor systems may be surprising, however, since these systems range from PDAs through mainframes. On a single-processor system, there is one main CPU capable of executing a general-purpose instruction set, including instructions from user processes.

multiple-processor systems :- Multiple processor systems have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices.

multiprocessor systems have three main advantages :-

(I) Increased throughput:- By increasing the number of processors, we expect to get more work done in less time. The speed-up ratio with N processors is not N, however; rather, it is less than N. When multiple processors cooperate on a task, a certain amount of overhead is incurred in keeping all parts working correctly. This overhead, plus contention for shared resources, lowers the expected gain from additional processors.

(II) Economy of scale:- Multiprocessor systems can cost less than equivalent multiple single-processor systems, because they can share peripherals, mass storage, and power supplies. If several programs operate on the same set of data, it is cheaper to store those data on one disk and to have all processors share them than to have many computers with local disks and many copies of the data.

(III) Increased reliability:- If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down.

Example:- If we have ten processors and one fails, then each of the remaining nine processors can pick up a share of work of the failed processor. Thus, the entire system runs only 10 percent slower.

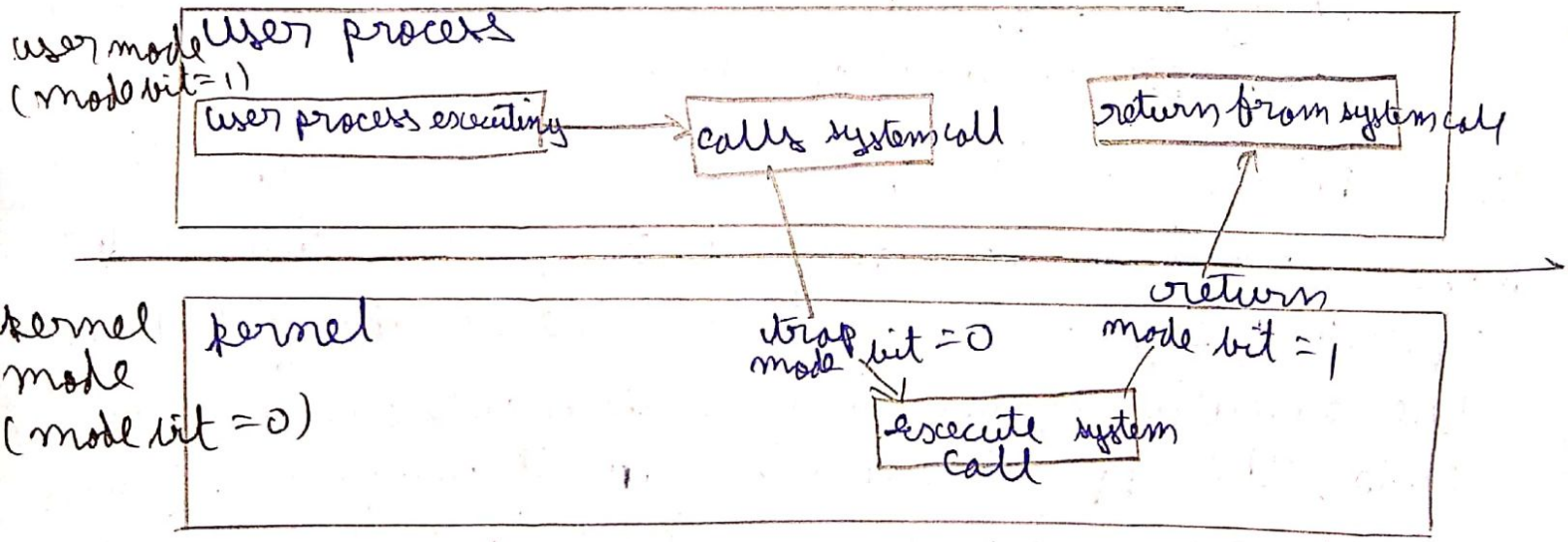
(b.) Increased reliability of a computer system is crucial in many applications. The ability, to continue providing service proportional to level of surviving hardware is called graceful degradation. Some systems go beyond graceful degradation and are called fault tolerant, because they can suffer a failure of any single component and still continue operation.

Q.2) Discuss in brief - - - - - - - - - - operations

Ans.2) In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating-system code and user-defined code. The approach taken by most computer systems is to provide hardware support that allows us to differentiate among various modes of execution.

At the very least, we need two separate modes of operation; user mode and kernel mode. A bit, called the mode bit, is

added to the hardware of the computer to indicate the current mode: Kernel(0) or user(1). With the mode bit, we are able to distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user. application, when the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system (via a system call), it must transition from user to kernel mode to fulfill the request.



The dual mode of operation provides us with the means for protecting the operating system from errant users and errant users from one another. We accomplish this protection by designating some of the machine instructions that may cause harm as privileged instructions. The hardware allows privileged instructions to be executed only in kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware ~~traps~~ does not execute the instruction but rather treats it as illegal and traps it to the operating system.

The instruction to switch to kernel mode, ~~the hardware~~ is an example of privileged instruction. Some other examples include I/O control, timer management, and interrupt management.

3) Consider the below statement:-

"Boundless . . . . . "

Ans 3) Organizing Threads:- A good solution is to limit the number of runnable threads to the number of hardware threads, and possibly limit it to the number of outer-level caches, if cache contention is a problem. Because target platforms vary in the number of hardware threads, avoid hard coding your program to a fixed number of threads. Set your program's degree of threading adapt to the hardware.

Runnable threads, not blocked threads, cause time-slicing overhead. When a thread blocks on an external event, such as a mouse click or disk I/O request, the operating system takes it off the round-robin schedule, so the thread no longer incurs time-slicing overhead. A program may have more software threads than hardware threads, and still run efficiently if most of those software threads are blocked.

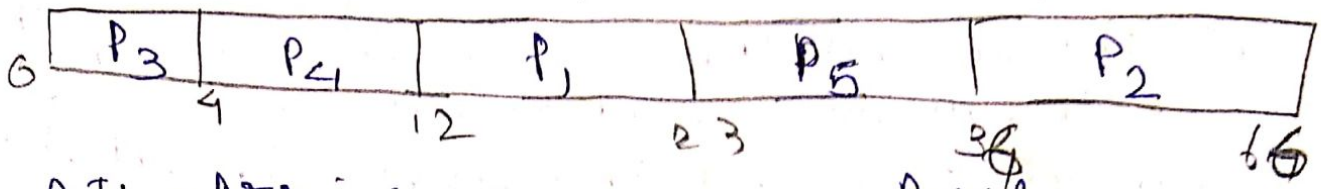
A helpful organizing principle is to separate compute threads from I/O threads. Compute threads should be the threads that are runnable most of the time, and ideally never block on external events. The number of compute threads should match the processor resources. The I/O threads are threads that wait on external events most of the time, and thus do not contribute to having too many threads.

4) Consider - - - - - - - - - waiting time

Ans 4) Given :-

Process	Burst Time
P_1	11
P_2	30
P_3	4
P_4	8
P_5	13

(10) SJF :- The GANT chart would be :-



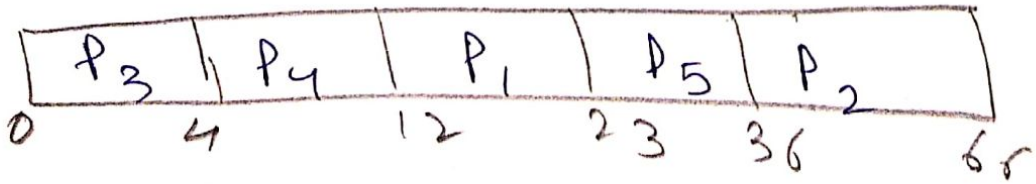
Let AT :- Arrival time, BT :- Burst time
FT :- Finish time, WT :- Waiting time.

	1	2	3	4	5
AT	0	0	0	0	0
BT	11	30	4	8	13
FT	23	66	4	12	36
WT	12	36	0	4	23

Average waiting time :- $\frac{12+36+0+4+23}{5}$

$= 15$ units

Preemptive SJF :- In this type of scheduling preemption occurs if processes arrive at different times, but here it will be same as SJF.



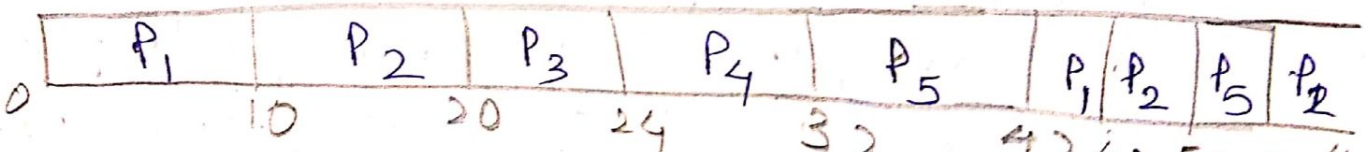
	1	2	3	4	5
AT	0	0	0	0	0
BT	11	30	4	8	13
FT	23	66	4	12	36
WT	12	36	0	4	23

Average wait time = $\frac{12+36+0+4+23}{5}$

= ~~15 units~~ 15 ms.

Round Robin :- In this scheduling process will be preempted if it expires the time quantum.

The GRANT chart will be :-



	1	2	3	4	5
AT	0	0	0	0	0
BT	11	30	4	8	13
FT	43	66	24	32	56
WT	32	36	20	24	43

Average wait time :- $\frac{32+36+20+24+43}{5}$ = 31ms

So as we can see :-
Average wait time of :-
(i) SJF = 15 ms
(ii) Premtive SSF = 15 ms
(iii) Round Robin = 31 ms

Q.5) Consider the code :- . . . . . .

Ans 5) [redacted]; In this [redacted]

To prove, that the solution is correct, we must examine the three conditions :-

(1) Mutual exclusion :- If one process is executing their critical section when the other wishes to do so, the second process will become blocked by the flag of the first process. If both processes attempt to enter at the same time, the last process to execute "~~turn = i~~ turn = j" will be blocked

(2) Progress :- Each process can only be blocked at the while if the other process wants to use the critical section (~~flag~~ b[j] == true), AND it is the other process's turn to use the critical section (~~turn~~ turn == j). If both of

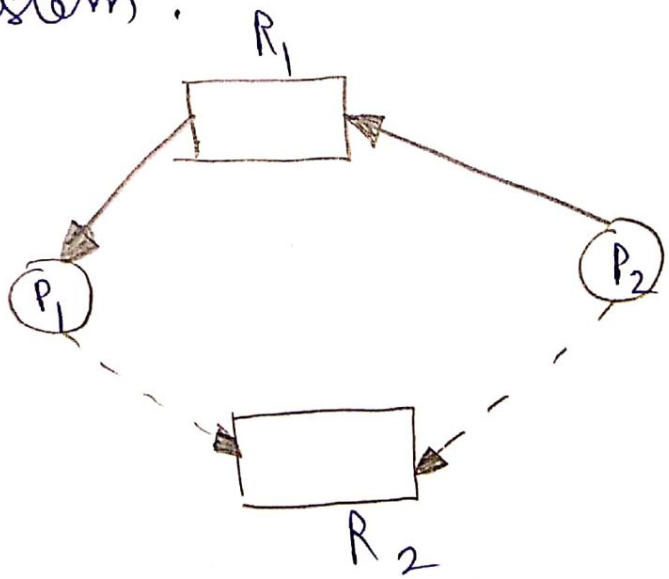
these conditions are true, then the other process(j) will be allowed to enter the critical section and upon exiting the critical section, will set flag[j] to false, releasing process i. The shared variable turn assures that only one process at a time can be blocked, and the flag variable allows one process to release the other when exiting their critical section.

(3) Bounded waiting :- As each process enters their entry section, they set the ~~turn~~ go variable to be the other process turn. Since no process ever sets it back to their own turn, this ensures that each process will have to let the other process go first at most one time before it becomes their turn again

Q. 6) Explain — — — — —

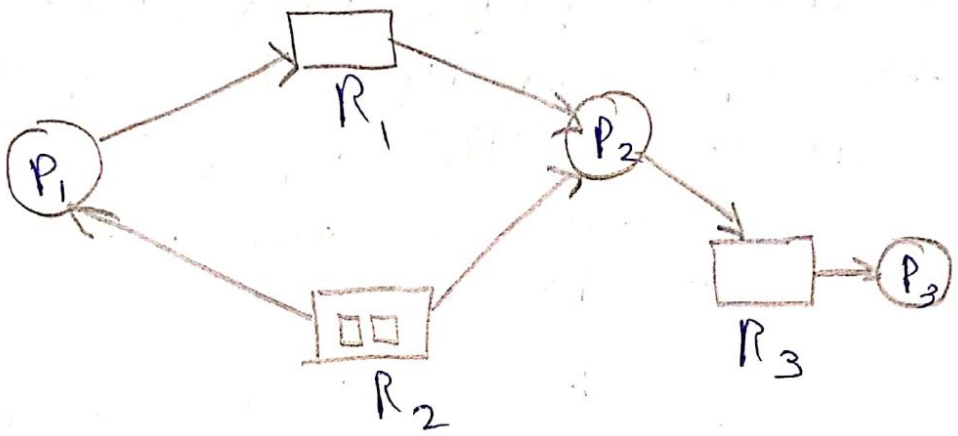
Ans 6) (a) Resource - Allocation Graph algorithm:-

- It is a graphical approach which can be used to describe deadlocks.
- Claim edge $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j; represented by a dashed line.
- Claim edge converts to request edge when a process requests a resource.
- When a resource is released by a process, assignment edge reconverts to a claim edge.
- Resources must be claimed a priori in the system.



The graph represents :-

- P_1 holds R_1 and needs a max of R_2.
- In order to avoid deadlock P_1 must run before P_2.



(ii) No, the system is not in deadlock because as soon as P_3 gets completed R_3 is free thus P_2 gets R_3 and gets completed which in turn releases R_1. Now P_1 gets R_1 and it also gets completed.

(iii) If P_3 requests a unit resource of R_2 then, Yes the system will be in a deadlock situation where P_1 is waiting for P_2 to get completed and

release R_1, but P_2 is waiting for P_3 to complete so it can get R_3, but P_3 is waiting either P_1 or P_2 to obtain access to resource R_2.

Ans 7) (i) First In First Out:-

Initially all page frames will be empty, so 4 page faults will occur, then the sequence is there will be a page fault when E arrives and again when A arrives at the last.

A	B	C	D	(4 faults)
B	C	D	E	(5^{th} fault)
C	D	E	A	(6^{th} fault)

Page B will not B available in the memory and there will be 6 page faults.

Ans → B won't be present 6 page faults.

(ii) Number of Page faults initially = 4
1 more fault occurs when E arrives and A is taken out.
So page fault before 25th page = 5

A	B	C	D

4 faults

E	B	C	D

5th fault

A arrives

E	B	C	D
?	?	?	?

6th fault

As when A arrives it is the last page we can't say which page will be removes and 6th fault will occur.

Ans:- 6 Page faults, can not be determined

(iii) After the initial 4 faults, when E arrives one more fault occurs and when A arrives again.

Freq of B = 9
Freq of C = 5
Freq of D = 6
Freq of E = 3

Based on this, E will be replaced by A and 6th fault will occur

Ans:- Page Faults = 6
E won't be resident in memory

(iv) After first 4 faults, A will be replaced by E, and when A is encountered again, least recently used page 'E' is removed and 6th fault occurs

Ans:- Page Faults = 6
E won't be resident in memory

Roll Number :- 18115069

Ans 8) (c) Given :-

Page size = 4KB

Page table entry = 4B

Outer page table size = 4KB

Paging = single level.

Ans **Virtual ~~do~~ Address space = 4KB**

(d) Given :-

Page size = 4KB, Page table entry = 4B, Outer page table size = 4KB, Paging = two level

Number of entries in outer page table = $\frac{4KB}{4B} = 2^{10}$

Therefore 2nd level page has 2^{10} pages. So virtual address space = $4KB * 2^{10} \times 4KB = 2^{22}$ Bytes

Ans 2^{22} Bytes

Roll Number:- 18115069

(C) ~~Given~~ Given:- Page size = 4KB, Page Table entry = 4B, outer page table size = 4KB, Paging = two level.

(C) Given :- Page size = 4KB, PTE = 4B, outer Page Table size = 256B, Paging = 3 level.

Number of entries in outer page table = $\frac{256}{4} = 64 = 2^6$

Number of pages in 2nd level = 2^6

Number of entries in each page = $\frac{4KB}{4B} = 2^{10}$

Number of pages in 3rd level = 2^{10}

So, virtual address space = $256B * 2^6 * 2^{10} * 4KB$

$= 2^{28} B$

Ans 2^{28} B

Roll Number :- 18115069

Ans 9) Given :-

Number of surfaces of the disk - 32 surfaces
No. of tracks per surface - 256
No. of sectors per track = 512
No. of Bytes per sector = 1024

(a) Capacity of Disk = (32 surface) * (2^8 track/surface) * (2^9 sec/track) * (2^{10} bytes/sector)

$= 2^{32}$ bytes $= 4 \times 2^{30}$ bytes

Ans = 4 Gigabytes (GB)

(b) No. of bits required to access a sector is $\lceil \log_2 N \rceil$

Here, N is number of sectors

No. of sectors = (32 surface) * (2^8 tracks/surface) * (2^9 sectors/track)

$N = 2^{22}$ sectors.

No. of bits required = $\lceil \log_2 2^{22} \rceil = 22$

Ans = 22

Roll Number:- 18115069

(C) Formatted overhead = 64 bytes/sector

Formatting overhead = Total no. of sectors * overhead per sectors

$= 2^{22} \times 64$ bytes $= 2^{28}$ or 256 MB

Formatted disk space = total space − total overhead

= 4 GB − 256 MB

Ans = 3.744 G

Ans 10) size of single disk block = 1024 bytes

$= 2^{10}$

No. of addresses per block = 256 = 2^8

Maximum file size = 10 direct + 1 single indirect + 1 double indirect + 1 Triple indirect

$= 10 + 2^8 + 2^8 \times 2^8 + 2^8 \times 2^8 \times 2^8$

$\approx 2^{24}$ blocks

Since, each block is of size 2^{10}, maximum file size = $2^{24} \times 2^{10} = 2^{34}$ bytes (approx)

Ans 2^{34} bytes