2nd Phase Spring End Semester Exam-2020

Operating System → Question & Solution

Section -: A

1.

a) Let's assume you are designing a pre-emptive scheduling algorithm where the primary concern is the **fairness** among the process execution time. Suggest which algorithm would you choose to maintain fairness and why?

Ans:-

Ans 1 (a) :- If fairness is the requirement for a preemptive scheduling algo, then we can use the Round Robin algorithm.

In Round Robin, each job will get the CPU after a fixed number of CPU Time [(n-1) time quantums]. Hence there is no starvation and every one has the fair chance to get the CPU.

b) Apply your suggested algorithm to the following problem:
Let there be 4 processes, all are arriving at time zero, with total execution time of 10, 20, 30 and 40 units respectively. Each process spends the first 10% of execution time doing I/O, the next 70% of time doing computation, and the last 20% of time doing I/O. Assume that all I/O operations can be overlapped as much as possible. For what percentage does the CPU remain idle? What would be the average turnaround time and average response time?

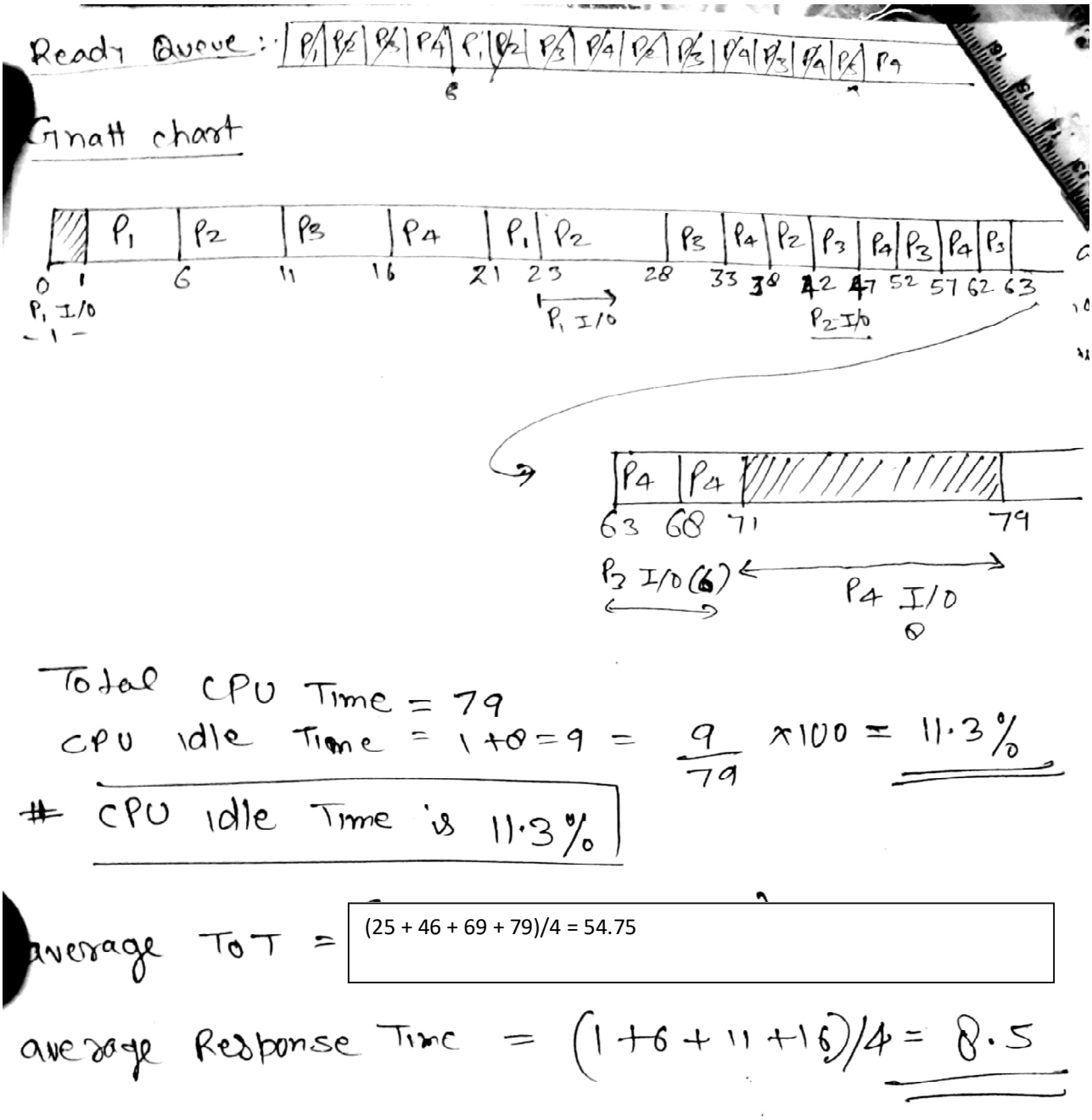
Ans:-

Ans 1 b)

Arrival Time → 0

Pid	Total Execution Time (ET)	I/O (10%) of ET	BT (70% of ET)	I/O (20% of ET)	TOT	WT	RT
(0) P_1	10	1	7	2	25	15	1
(0) P_2	20	2	14	4	46	26	6
(0) P_3	30	3	21	6	69	39	11
(0) P_4	40	4	28	8	79	39	16

Here we are solving it using Round Robin algo.
The Time Quantum for RR is not given and students are suppose to assume a TQ value of their choice.
the answer will depend upon the choice of TQ value, hence may be differ among students.
Here I assume the TQ value for our RR is = 5 CPU units



2.

a) Write down WAIT( ) and SIGNAL( ) operations on counting semaphore.
Ans:-

```
struct Semaphore {
int value;
Queue q;
}
```

First	Second

WAIT(Semaphore s)
{
s.value = s.value - 1;
if (s.value < 0) {
s.q.insert(P_i);
block(P_i);
}
}
	SIGNAL(Semaphore s)
	{
	s.value = s.value + 1;
	if (s.value <= 0) {
	P_i = s.q.remove();
	wakeup(P_i);
	}
	}

b) Consider a ticket counter where a single employee is sitting to provide tickets to many travelers in one by one basis. At any time more than one traveler can come to the waiting hall. One traveler can book only one ticket. There is a box similar to buffer having unlimited number of slots where the travelers have to put his travel request from. When the box is accessed by the employee, he takes the request form one at a time, and books the ticket based on the detail given on the request form and put the ticket in the same slot so that the corresponding traveler can collect his ticket and release the slot. Once the slot gets empty that can be used by another traveler for putting his travel request form. The slots of the box will be accessed by the travelers and employee in a sequential manner (i.e. 0, 1, 2, 3 ...). Write separate functions for traveler and employee (using semaphore and WAIT ( )/SIGNAL ( ) operation) with ensuring no race condition even if multiple travelers and employee access the same box as buffer.

Ans:-

Global:
int buf[]; //assume that buffer is filled with NULL
int in=0, out=0;
Semaphore count1=0, mutex=1, sem[]={0};
Local:
int myticket_no; //for each process

Traveler	**Employee**

//***Ticket request***
p(mutex)
myticket_no=in
--- Keep the travel request form in ***buf[in]*** ---
in++
v(mutex)
 	
v(count1)
p(sem[myticket_no])
 	
//***Ticket collect***
p(mutex)
--- Collect ticket from ***buf[myticket_no]*** no slot ---
buf[myticket_no]=NULL
v(mutex)
	While(1){
	p(count1)
	//***Ticket approve***
	p(mutex)
	--- book ticket for ***buf[out]*** ---
	v(sem[out])
	out++
	v(mutex)
	}

NOTE :

- While giving marks please consider that shared variables must be under proper mutex.
- There are three basic parts, i.e. Ticket request, Ticket approve, and Ticket collect. For each part, if it is properly written, 2 marks each. Variable declaration 1 mark. Overall correct 3 mark, if not then 0.5 marks.
- You need to check that the traveler should not collect other's ticket.
- Traveler and employee code must be properly synchronized

Section -: B

3.

a) What are the possible recovery strategies once any deadlock is detected in a system? Explain briefly.

Ans:-

i) Process Termination:

Abort all the Deadlocked Processes:

Aborting all the processes will certainly break the deadlock, but with a great expenses.

The deadlocked processes may have computed for a long time and the result of those partial computations must be discarded and there is a probability to recalculate them later.

Abort one process at a time until deadlock is eliminated:

Abort one deadlocked process at a time, until deadlock cycle is eliminated from the system.

Due to this method, there may be considerable overhead, because after aborting each process, deadlock detection algorithm has to be run to check whether any processes are still deadlocked.

ii) Resource Preemption:

To eliminate deadlocks using resource preemption, some resources will be preempted from processes and allocated those resources to other processes. This method will raise three issues:

Selecting a victim: determine which resources and which processes are to be preempted and also the order to minimize the cost.

Rollback: determine what should be done with the process from which resources are preempted. One simple idea is total rollback. That means abort the process and restart it.

Starvation: In a system, it may happen that same process is always picked as a victim. As a result, that process will never complete its designated task. This situation is called Starvation and must be avoided. One solution is that a process must be picked as a victim only a finite number of times.

b) kZJXCBjkXBV
Consider the following snapshot of a system with 5 processes (P_0, P_1, P_2, P_3, P_4) and 3 resource types (R_0, R_1, R_2,):

Process	Max Need			Allocation			Available		
	R_0	R_1	R_2	R_0	R_1	R_2	R_0	R_1	R_2
P_0	0	4	1	0	1	1	1	T	2
P_1	1	5	U	1	0	0			
P_2	2	3	5	1	1	5			
P_3	0	5	5	0	2	3			
P_4	2	U	5	0	0	1			

Answer the following questions using the banker's algorithm:

(where U = (N MOD 4) + 2; T= N MOD 6 + 1; assume N = Your Roll Number)

i. Find the safe sequence if the system in a safe state?

ii. If a request from process P1 arrives for (0, 4, 0), can the request be granted immediately?

Ans:-

U = (N MOD 4) + 2; T= N MOD 6 + 1; assume N = Your Roll Number

U = 2, 3, 4, 5

T = 1, 2, 3, 4, 5, 6

Let consider one combination U = 2, T = 1.

Process	Max Need			Allocation			Available		
	R_0	R_1	R_2	R_0	R_1	R_2	R_0	R_1	R_2
P_0	0	4	1	0	1	1	1	**1**	2
P_1	1	5	**2**	1	0	0			
P_2	2	3	5	1	1	5			
P_3	0	5	5	0	2	3			
P_4	2	**2**	5	0	0	1			

Process	Max Need			Allocation			Need			Available		
	R_0	R_1	R_2	R_0	R_1	R_2	R_0	R_1	R_2	R_0	R_1	R_2
P_0	0	4	1	0	1	1	0	3	0	1	1	2
P_1	1	5	2	1	0	0	0	5	2			
P_2	2	3	5	1	1	5	1	2	0			
P_3	0	5	5	0	2	3	0	3	2			
P_4	2	2	5	0	0	1	2	2	4			

i. Not a safe sequence

ii. Not granted

Let consider one combination U = 2, T = 6.

Process	Max Need			Allocation			Available		
	R_0	R_1	R_2	R_0	R_1	R_2	R_0	R_1	R_2
P_0	0	4	1	0	1	1	1	**6**	2
P_1	1	5	**2**	1	0	0			
P_2	2	3	5	1	1	5			
P_3	0	5	5	0	2	3			
P_4	2	**2**	5	0	0	1			

Process	Max Need			Allocation			Need			Available		
	R_0	R_1	R_2	R_0	R_1	R_2	R_0	R_1	R_2	R_0	R_1	R_2
P_0	0	4	1	0	1	1	0	3	0	1	6	2
P_1	1	5	2	1	0	0	0	5	2			
P_2	2	3	5	1	1	5	1	2	0			
P_3	0	5	5	0	2	3	0	3	2			
P_4	2	2	5	0	0	1	2	2	4			

i) Safe sequence: <P_0, P_1, P_2, P_3, P_4>

Process	Max Need			Allocation			Need			Available		
	R_0	R_1	R_2	R_0	R_1	R_2	R_0	R_1	R_2	R_0	R_1	R_2
P_0	0	4	1	0	1	1	0	3	0	1	2	2
P_1	1	5	2	1	4	0	0	1	2			
P_2	2	3	5	1	1	5	1	2	0			
P_3	0	5	5	0	2	3	0	3	2			
P_4	2	2	5	0	0	1	2	2	4			

ii) Safe Sequence <P_1, P_2, P_3, P_4, P_0>

Check for all possible values of U & T.

4.

a) Consider two machines A and B of different architectures, running two different operating systems OS-A and OS-B. An application binary that has been compiled for machine A may have to be recompiled to execute correctly on machine B. Justify it.
Ans:-

High level languages need not to be modified if we run them in two different architecture as they are architecture independent. However, machine language programme (compiled binary code) differs from one architecture to another. Therefore, an application binary that has been compiled for machine A must have to be recompiled to execute correctly on machine B.

Application --> Compiler (OS-A) --> Binary code for machine A

Application --> Compiler (OS-B) --> Binary code for machine B

Now, Binary code for machine A differs from Binary code for machine B. So, Binary code for machine A can not run on machine B and vice versa.

b) Consider a simple system where the RAM can hold F (F=(Your Roll No. MOD 3) + 2) number of physical frames. The size of physical frames and logical pages is 16 bytes. The virtual addresses of the process are 6 bits in size. The program generates the following 20 virtual address references as it runs on the CPU: 0, 1, 20, 2, 20, 21, 32, 31, 0, 60, 0, 0, 16, 1, 17, 18, 32, 31, 0, 61. (Note: the 6-bit addresses are shown in decimal here.) Assume that the physical frames in RAM are initially empty.

i. Translate the virtual addresses above to logical page numbers referenced by the process. That is, write down the reference string of 20 page numbers corresponding to the virtual address accesses above. Assume pages are numbered starting from 0, 1, …

ii. Calculate the number of page faults generated by the accesses above, assuming a FIFO page replacement algorithm.

iii. Repeat (ii) above for the LRU page replacement algorithm.

Ans:-

For 6 bit virtual addresses, and 4 bit page offsets (page size 16 bytes), the most significant 2 bits of a virtual address will represent the page number. Divide 16 in the virtual address string to generate the logical page nos.

VA:-Virtual Address, LPN:-Logical page Number

VA :- 0, 1, 20, 2, 20, 21, 32, 31, 0, 60, 0, 0, 16, 1, 17, 18, 32, 31, 0, 61

LPN:-0, 0, 1, 0, 1, 1, 2, 1, 0, 3, 0, 0, 1, 0, 1, 1, 2, 1, 0, 3

So the reference string is 0, 0, 1, 0, 1, 1, 2, 1, 0, 3 (repeated again).

iv. Calculate the number of page faults generated by the accesses above, assuming a FIFO page replacement algorithm.

Ans:

Assuming Number of page frame = 3

Page faults with FIFO = 8. Page faults on 0,1,2,3 (replaced 0), 0 (replaced 1), 1 (replaced 2), 2 (replaced 3), 3.

v. Repeat (ii) above for the LRU page replacement algorithm.

Ans:

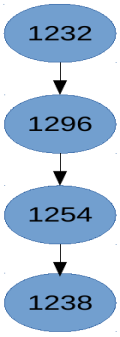
Assuming Number of page frame = 3

Page faults with LRU = 6. Page faults on 0, 1, 2, 3 (replaced 2), 2 (replaced 3), 3.

Number of page faults will differ for different number of page frame allocation (based on Roll No). Please calculate accordingly.

5.

a) Consider the following process tree where nodes are represented with process ID. These processes are created in the Unix platform. This figure shows the parent child relationship among the process.



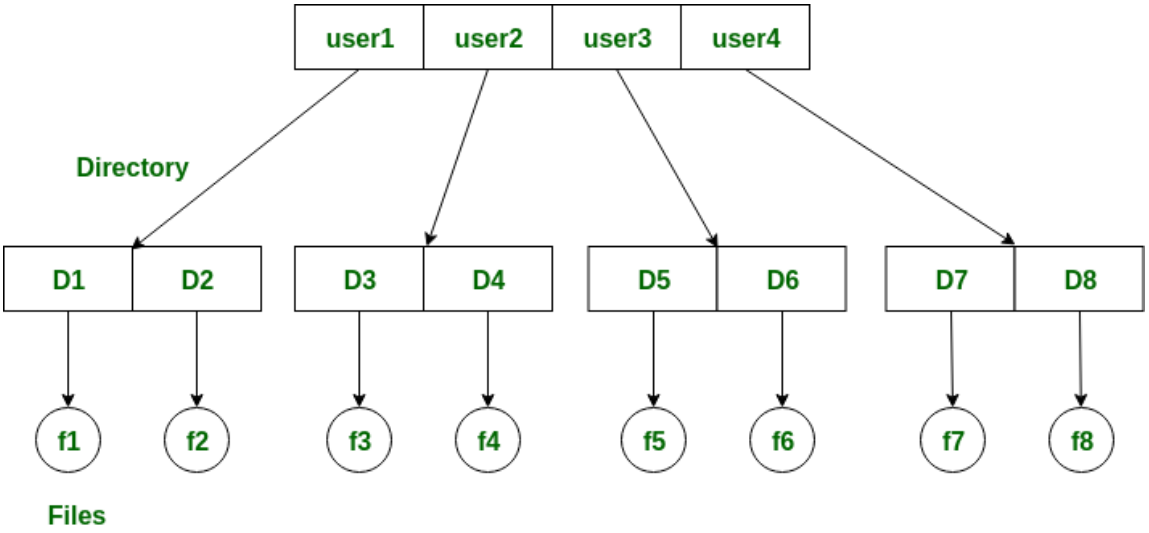
Let, after creation of all these four process, while the processes 1232 and 1296 continue with their assigned job, process 1254 terminates. At this scenario, what is the state of process 1254 after terminates? After termination of process 1254, if process 1238 gets terminated, then which process is responsible for receiving the exit status of process 1238?
Ans:-

process 1254 will enter in Zombie state and init process is responsible for receiving the exit status of process 1238.

b) Describe briefly two level directory structure. What is the role of an I-node? Describe the structure of I-node in UNIX. What entries undergo changes when a file is opened to read / write / copied / renamed.
Ans:-

As we have seen, a single level directory often leads to confusion of files names among different users. the solution to this problem is to create a separate directory for each user.

In the two-level directory structure, each user has there own user files directory (UFD). The UFDs has similar structures, but each lists only the files of a single user. system's master file directory (MFD) is searches whenever a new user id=s logged in. The MFD is indexed by username or account number, and each entry points to the UFD for that user.



Advantages:

We can give full path like /User-name/directory-name/.

Different users can have same directory as well as file name.

Searching of files become more easy due to path name and user-grouping.

Disadvantages:

A user is not allowed to share files with other users.

Still it not very scalable, two files of the same type cannot be grouped together in the same user.

The inode (index node) is a data structure in a Unix-style file system that describes a file-system object such as a file or a directory. Each inode stores the attributes and disk block locations of the object's data. File-system object attributes may include metadata (times of last change, access, modification), as well as owner and permission data.

I-node is a data structure that keeps track of all the information about a file. You keep your information in a file and the OS stores the information about a file in an I-node.

Data structures that contain information about files in UNIX file systems that are created when a file system is created. Each file has an I-node and is identified by an I-node number (I-number) in the file system where it resides. I-nodes provide important information on files such as user and group ownership, access mode (read, write, execute permissions) and type.

The I-node contains the following pieces of information:

• Mode/permission (protection)

• Owner ID

• Group ID

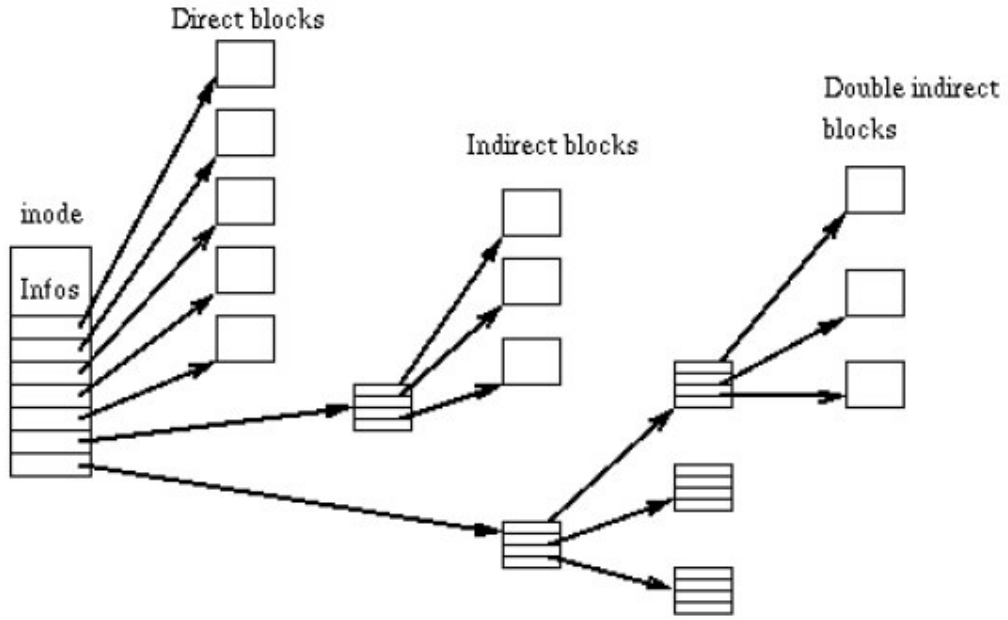
• Size of file

• Number of hard links to the file

• Time last accessed

• Time last modified

• Time I-node last modified



When a file is created inside a directory then the file-name and Inode number are assigned to file. These two entries are associated with every file in a directory. The user might think that the directory contains the complete file and all the extra information related to it but this might not be the case always. So we see that a directory associates a file name with its Inode number.

When a user tries to access the file or any information related to the file then he/she uses the file name to do so but internally the file-name is first mapped with its Inode number stored in a table. Then through that Inode number the corresponding Inode is accessed. There is a table (Inode table) where this mapping of Inode numbers with the respective Inodes is provided.

Following entries will change while:

Reading and writing-----> Offset

Copied---------->Index structure

Rename---------->name in file table