Roll Number:____________________________

Thapar Institute of Engineering & Technology

Department of Computer Science and Engineering

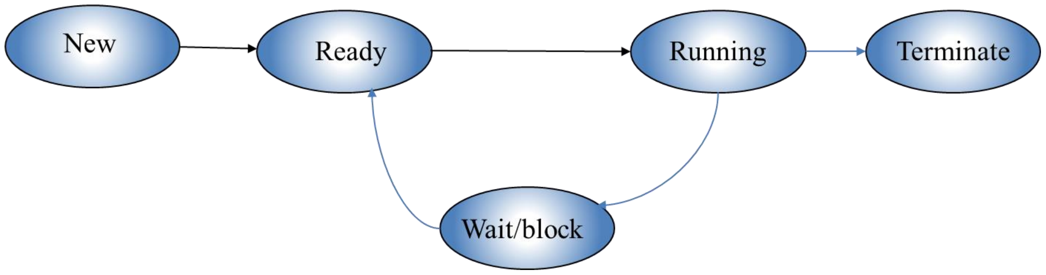
MID SEMESTER EXAMINATION

B. E. (Second/Third Year): Semester-I (2019)
(COE/ENC/ECE)
September 26, 2019
Time: 2 Hours, Max Marks: 25

Course Code: UCS303
Course Name: Operating Systems
Monday, 10:30 – 12:30 Hrs
Name of Faculty: Dr Vinay Arora, Dr Raman Goyal, Dr Tarunpreet Bhatia, Dr Deepshikha Tiwari, Dr Amritpal Singh, Dr Neenu Garg, Ms Swati Kumari

Attempt all questions and its part(s) in serial order.

Q.1 a) Make a suitable illustration related to the process state transition diagram for dealing with a specific scenario, where each process is in one of the five states. The operating system uses only non-preemptive scheduling algorithm. 02

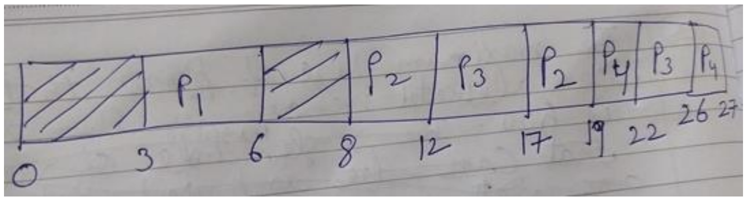


01 mark when all the states have been drawn
01 mark when **all the arrows/directions** are correct (one full mark will be deducted if you have drawn arrow from Running → Ready State)

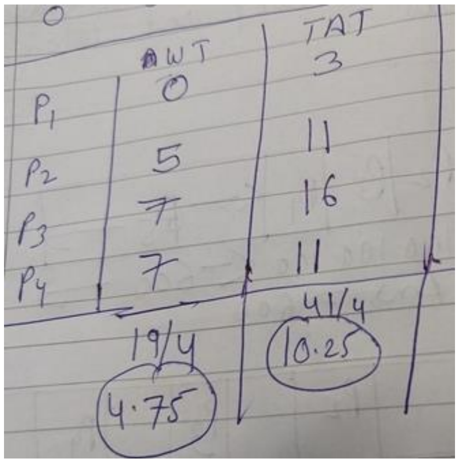
b) Considering a single input queue for the uniprocessor multitasking system where processor has been initiated/running right from 0th time; the operating system chooses a different quanta/time slice on the basis of nature of the process. Here, the processes P1, P2, P3 and P4 are of different flavor and the quanta received by the one process is different as from another; and it's 3,4,5, and 3 respectively for P1, P2, P3, and P4. Table given below shows the arrival and CPU burst time. CPU remains idle, when there is no process to execute. 03

Process	Arrival Time	CPU Burst Time
P1	3	3
P2	8	6
P3	10	9
P4	16	4

Drawing a suitable GANTT chart, calculate the average waiting time, and average turnaround time for the given system (Take all time values in milliseconds).



GANTT Chart correct up-to 17 milliseconds 0.5 mark
GANTT Chart Full Correct 1 mark



	WT	TAT
P1	0	3
P2	5	11
P3	7	16
P4	7	11
	19/4	41/4
	4.75	10.25

Computations for P1, P2, P3, P4 w.r.t WT 0.5 mark
Computations for P1, P2, P3, P4 w.r.t TAT 0.5 mark

Average of WT 0.5 marks
Average of TAT 0.5 marks

c) Consider a system (that follows pre-emption) consists of three processes namely, Process1, Process2, and Process3, where each process has N number of instances, and1st instance of all the processes arrives at the 0th millisecond. The arrival of the next instances is periodic, with time interval of 3 milliseconds for Process1, 7 milliseconds for Process2 and 20 milliseconds for Process3. Every instance of Process1, Process2, and Process3 needs CPU burst time of 1, 2, and 4 milliseconds, respectively. The priority of each process is inverse of its burst time. Processes get the CPU in the order of their priority (highest one gets the CPU first). With a suitable GANTT chart, explain the scenario and **Compute the following** (taking the system as uniprocessor): 03

i. Time where 1st instance of the Process3 will get pre-empted.
ii. Time where 1st instance of the Process3 will complete its execution.

Sol. 1: Periodic arrival of T_1: 0, 3, 6, 9, 12, 15, 18, 21, ….

Priority of T_1 = 1/3, service time of T_1 = 1.

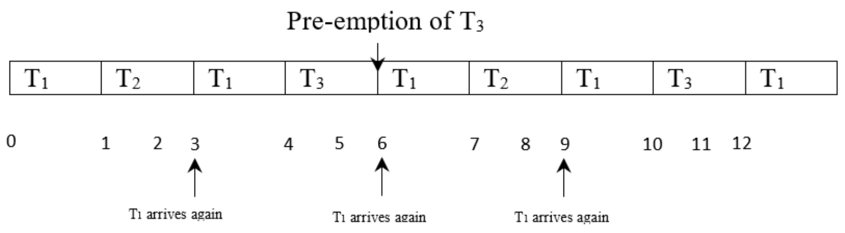
Periodic arrival times of T_2: 0, 7, 14, 21, ….

Priority of T_2 = 1/7, service time of T_2 = 2.

Periodic arrival times of T_3: 0, 20, 40, ….

Priority of T_3 = 1/20, service time of T_3 = 4.

T_1 has highest priority and T_3 has the lowest priority.



Periods of T1, T2 and T3 are 3ms, 7ms and 20ms
Since priority is inverse of period, T1 is the highest priority task, then T2 and finally T3

Every instance of T1 requires 1ms, that of T2 requires 2ms and that of T3 requires 4ms

Initially all T1, T2 and T3 are ready to get processor, T1 is preferred. Second instances of T1, T2, and T3 shall arrive at 3, 7, and 20 respectively. Third instance of T1, T2 and T3 shall arrive at 6, 14, and 49 respectively.

Time-Interval	**Tasks**
0-1	T1
1-2	T2
2-3	T2
3-4	T1 [Second Instance of T1 arrives]
4-5	T3
5-6	T3
6-7	T1 [Third Instance of T1 arrives]

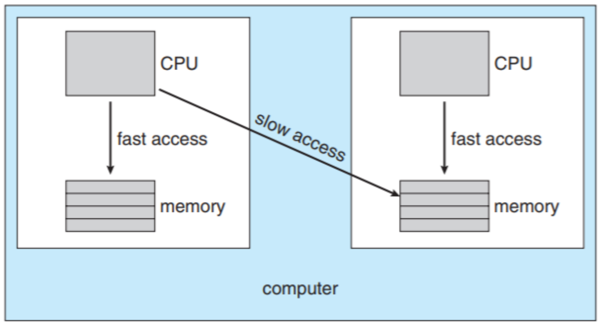
[**Therefore 1st instance of T3 is pre-empted at 6**] 01 mark GANTT Chart up-to this point 0.5 mark

7-8	T2 [Second instance of T2 arrives]
8-9	T2
9-10	T1 [Fourth Instance of T1 arrives]
10-11	T3
11-12	T3

[**First Instance of T3 completed at 12**] 01 mark GANTT Chart up-to this point 0.5 mark

Q.2 a) Give the full form of NUMA and draw a suitable diagram to depict the same. Why a program need to have the processor affinity? 02

NUMA – Non uniform Memory Access 0.5 mark



0.5 Mark

Most SMP systems try to avoid migration of processes from one processor to another and instead attempt to keep a process running on the same processor. This is known as processor affinity—that is, a process has an affinity for the processor on which it is currently running.

The data most recently accessed by the process populate the cache for the processor. Successive memory accesses by the process are often satisfied in cache memory. If the process migrates to another processor. The contents of cache memory must be invalidated for the first processor, and the cache for the second processor must be repopulated. Due to the high cost involved for invalidating and repopulating the caches. I mark

b) Consider a system having five processes namely, $Process_1$, $Process_2$, $Process_3$, $Process_4$, $Process_5$ that are waiting for processing unit to get executed/run. The execution times for the said processes are 9, 6, 3, 5, and X units respectively. What could be the various orders for executing the said processes (based on the value of X), so that the average waiting time is minimal for each order (consider non-preemptive scheme only). 02

All we need to do for minimizing response time is to run jobs in increasing order of burst time.

```
if X is <=3
than the order is X,j3,j4,j2,j1
if X is >3 and <=5
than the order is j3,X,j4,j2,j1
if X is >5 and and <9
than the order is j3,j4,j2,X,1
if X is >=9
than the order is j3,j4,j2,j1,X
```

0.5 mark for each sequence

Q.3 a) Consider an instance (at t_0) of a system having N processes that are requesting for and releasing the space of RAM. System is having a RAM of size 1200 KB, where, first chunk of 400 KB will be consumed permanently by the Kernel module; as it will be present all the time in the memory. Now, the remaining 800 KB will get managed using variable partitioning method (and without using compaction). At time instance t_1 the system has three processes P_1, P_2, P_3 that occupies 200 KB, 134 KB, and 90 KB of memory respectively. Take P4 as next 03 process that requests for the RAM space. [Take the given scenario independently for solving i and ii; and P_4 requests memory in integer values only.]

i. What could be the maximum size (in KB) for P_4 that can be allocated memory by the operating system? (Explain with proper depiction)

ii. What could be the minimum size (in KB) for P_4 that could be rejected? (Explain with proper depiction)

With n processes, we can create a minimum 1 hole.
Hole = 800 – (200+134+90) = 376
The maximum process can be allocated is 376 KB

Kernel	P1	P2	P3	Hole
400	200	134	90	376

0.5 mark is for final answer *i.e.* 376
01 mark for description (Theory or Diagram)

With n processes (except kernel), we can create maximum n+1 holes.

Kernel	hole	P1	hole	P2	hole	P3	hole
400		200		134		90	

Hole size = Total size available for holes/Available holes = 376/4 = 94 KB
Therefore, if a process comes with the size 95 KB, we can't allocate.

0.5 mark is for final answer *i.e.* 95
01 mark for description (Theory or Diagram)

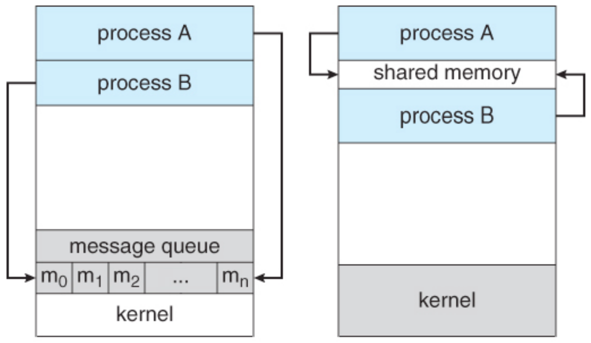
b) With a suitable illustration describe the various fundamental modes available for inter-process communication. 02

Working together with multiple processes, require an inter-process communication (IPC) method which will allow them to exchange data along with various information. There are two primary models of inter-process communication:

i. shared memory and
ii. message passing

In the shared-memory model, a region of memory which is shared by cooperating processes gets established. Processes can be then able to exchange information by reading and writing all the data to the shared region. In the message-passing form, communication takes place by way of messages exchanged among the cooperating processes.

The two communications models are contrasted in the figure below:



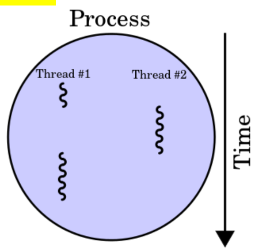
(a) Message Passing (b) Shared Memory

0.5 mark for diagram of message passing, 0.5 for description
0.5 mark for diagram of shared memory, 0.5 for description

Q.4 a) Using a suitable diagram, explain the concept of Multi-threading. Give a suitable representation for Many-to-one model for multi-threading. Give a scenario where Many-to-one model will be having serious disadvantage of using it. 02

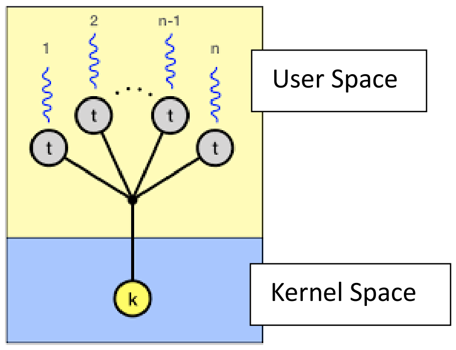
Multi-threading: It is a type of execution model that allows multiple threads to exist within the context of a process such that they execute independently but share their process resources. A thread maintains a list of information relevant to its execution including the priority schedule, exception handlers, a set of CPU registers, and stack state in the address space of its hosting process. Multithreading is also known as threading

Example/Sample Diagram:



0.5 mark for diagram

Many-to-One Model



I mark for diagram

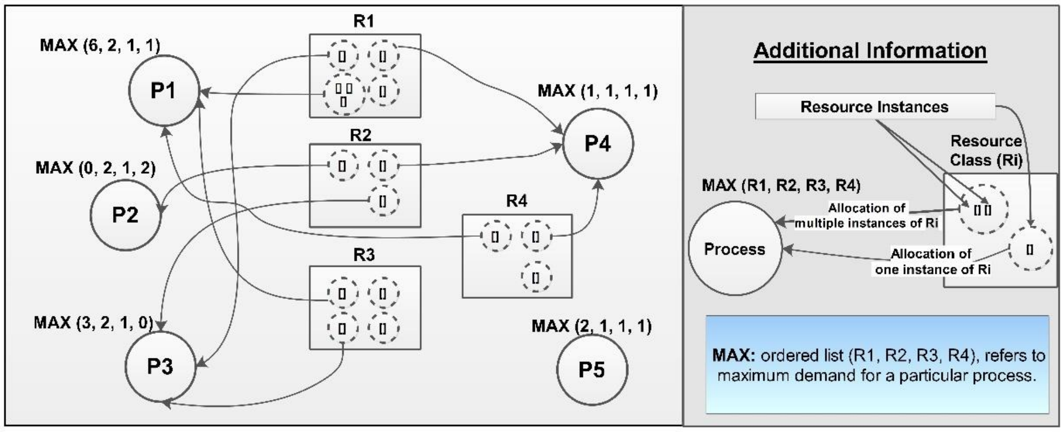
In this model, we have multiple user threads mapped to one kernel thread. In this model when a user thread makes a blocking system call entire process blocks. As we have only one kernel thread and only one user thread can access kernel at a time, so multiple threads are not able access multiprocessor at the same time. 0.5 mark for description

b) Comment on the statement "*While Multicore systems are multi-processor systems, not all multi-processor systems are multicore*". 01

The CPU (Central Processing Unit) or the processor is the brain of the computer. It handles all the functionalities of the other components. The execution unit of the CPU is called the core. It reads and executes instructions. The instructions can be a calculation, a data transferring instruction, branch instruction, etc. A CPU with a single core is called a uniprocessor. When a system has more than one core, it is called a multicore. A CPU with two cores is called a dual-core processor while a processor with four cores is called a quad-core processor. Moreover, high-performance computers can have six to eight cores. The main advantage of a multicore is that it is capable of executing multiple instructions simultaneously on separate cores.

A **multicore** is a single CPU or processor with two or more independent processing units called cores that are capable of reading and executing program instructions. Where, core can be taken virtually synonymous to a stand-alone processing unit. **Multiprocessor** is a system with two or more CPUs that allows simultaneous processing of programs. In this case a processing unit may or may not comprises core in it.

Q.5 Diagram given below depicts a snapshot of a system at a time instance t_0; where, the system is having five processes and four resource classes. Further, each resource class has some resource instances. Write Resource-Request algorithm; and compute the values for all the involved data structures at t_0. With a suitable explanation, comment on the feasibility for executing/processing following three resource requests initiated by three processes one after another: P1 requests for (1, 1, 0, 0); P2 requests for (1, 0, 1, 1); P5 requests for (1, 0 1, 0). 05



Resource Request Algorithm 01 Mark

Let ***Request***$_i$ be the request vector for process P_i. If ***Request***$_i$ $[j]$ $==$ k, then process P_i wants k instances of resource type R_j. When a request for resources is made by process P_i, the following actions are taken:

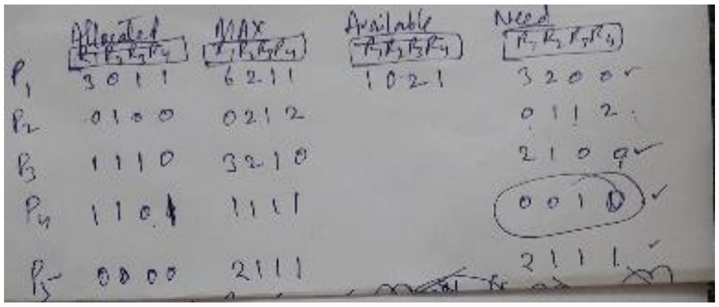
1. If ***Request***$_i$ $\leq$ ***Need***$_i$, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.
2. If ***Request***$_i$ $\leq$ ***Available***, go to step 3. Otherwise, P_i must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process P_i by modifying the state as follows:

$$\begin{aligned}
&\textbf{\textit{Available}} = \textbf{\textit{Available}} - \textbf{\textit{Request}}_i;\\
&\textbf{\textit{Allocation}}_i = \textbf{\textit{Allocation}}_i + \textbf{\textit{Request}}_i;\\
&\textbf{\textit{Need}}_i = \textbf{\textit{Need}}_i - \textbf{\textit{Request}}_i;
\end{aligned}$$

If the resulting resource-allocation state is safe, the transaction is completed, and process P_i is allocated its resources. However, if the new state is unsafe, then P_i must wait for ***Request***$_i$, and the old resource-allocation state is restored.

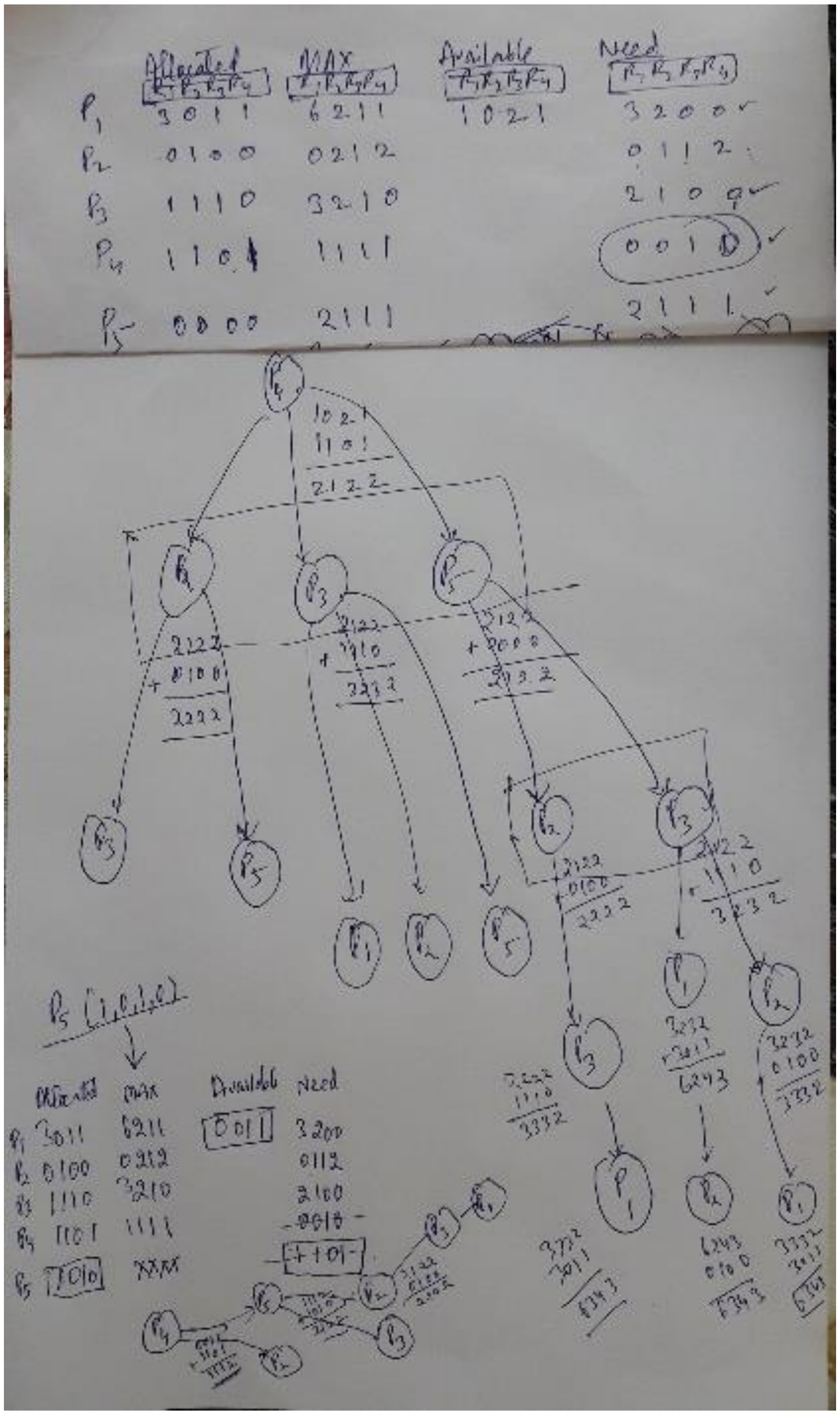
Data Structures

Allocation 0.5 mark, Available 0.5 mark, Need 0.5 mark



	Allocated	MAX	Available	Need
	$R_1 R_2 R_3 R_4$	$R_1 R_2 R_3 R_4$	$R_1 R_2 R_3 R_4$	$R_1 R_2 R_3 R_4$
P_1	3 0 1 1	6 2 1 1	1 0 2 1	3 2 0 0 ✓
P_2	0 1 0 0	0 2 1 2		0 1 1 2
P_3	1 1 1 0	3 2 1 0		2 1 0 0 ✓
P_4	1 1 0 1	1 1 1 1		0 0 1 0 ✓
P_5	0 0 0 0	2 1 1 1		2 1 1 1 ✓

Safe Sequence 0.5 mark



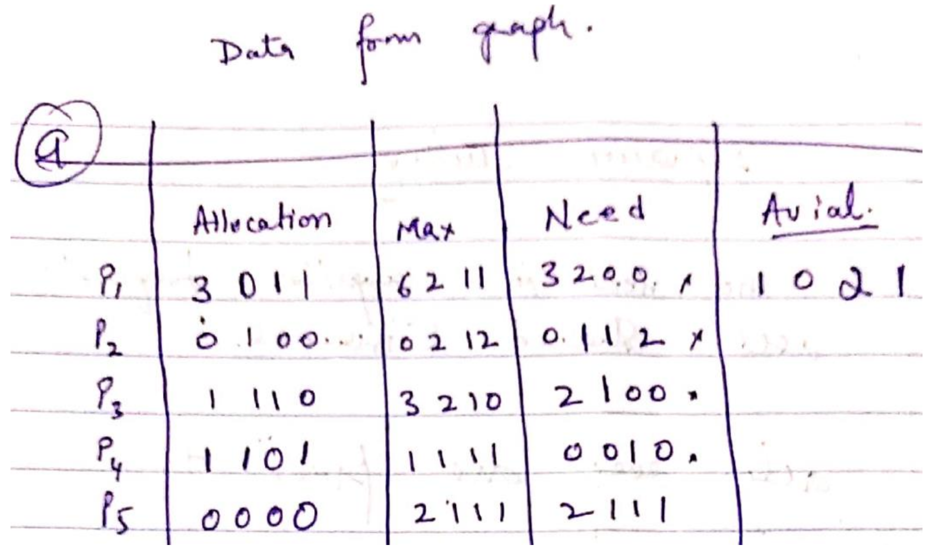
	Allocated $R_1R_2R_3R_4$	MAX $R_1R_2R_3R_4$	Available $R_1R_2R_3R_4$	Need $R_1R_2R_3R_4$
P_1	3 0 1 1	6 2 1 1	1 0 2 1	3 2 0 0 ✓
P_2	0 1 0 0	0 2 1 2		0 1 1 2
P_3	1 1 1 0	3 2 1 0		2 1 0 0 ✓
P_4	1 1 0 1	1 1 1 1		0 0 1 0 ✓
P_5	0 0 0 0	2 1 1 1		2 1 1 1 ✓

P_4
1 0 2 1
1 1 0 1
2 1 2 2
P_1
P_3
P_5
2 1 2 2
+ 0 1 0 0
2 2 2 2
2 1 2 2
+ 1 1 1 0
3 2 3 2
2 1 2 2
+ 0 0 0 0
2 1 2 2
P_3
P_5
P_1
P_2
P_5
P_2
P_3
2 1 2 2
+ 0 1 0 0
2 2 2 2
2 1 2 2
+ 1 1 1 0
3 2 3 2
P_3
P_1
P_2
2 2 2 2
1 1 1 0
3 3 3 2
3 2 3 2
+ 3 0 1 1
6 2 4 3
3 2 3 2
0 1 0 0
3 3 3 2
P_1
P_2
P_1
3 3 3 2
3 0 1 1
6 3 4 3
6 2 4 3
0 1 0 0
6 3 4 3
3 3 3 2
3 0 1 1
6 3 4 3
P_5 (1, 0, 1, 0)

	Allocated	max	Available	Need
P_1	3011	6211	0011	3200
P_2	0100	0212		0112
P_3	1110	3210		2100
P_4	1101	1111		0010
P_5	1010	XXM		1101

P_4
P_5
P_2
P_3
P_1
P_2
P_3

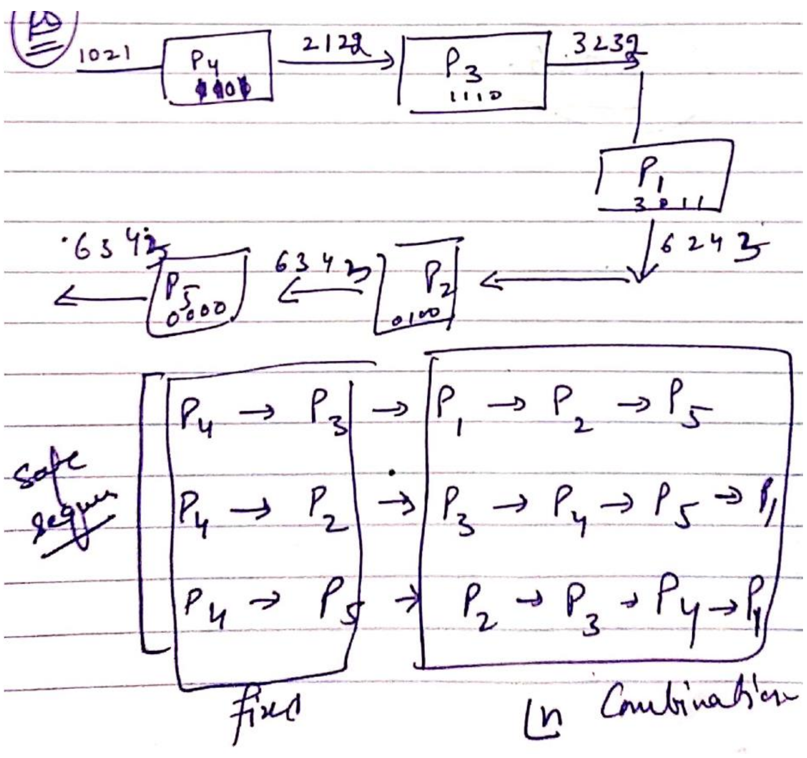
OR



Data from graph.

(a)

	Allocation	Max	Need	Avial.
P_1	3011	6211	3200	1021
P_2	0100	0212	0112	
P_3	1110	3210	2100	
P_4	1101	1111	0010	
P_5	0000	2111	2111	



(b)

$$\begin{tikzcd}
{} \arrow[r, "1021"] & \boxed{P_4\ 1101} \arrow[r, "2122"] & \boxed{P_3\ 1110} \arrow[r, "3232"] & \boxed{P_1\ 3011} \arrow[d, "6243"] \\
{} & \boxed{P_5\ 0000} \arrow[l, "6343"'] & \boxed{P_2\ 0100} \arrow[l, "6343"'] & {} \arrow[l]
\end{tikzcd}$$

safe sequence

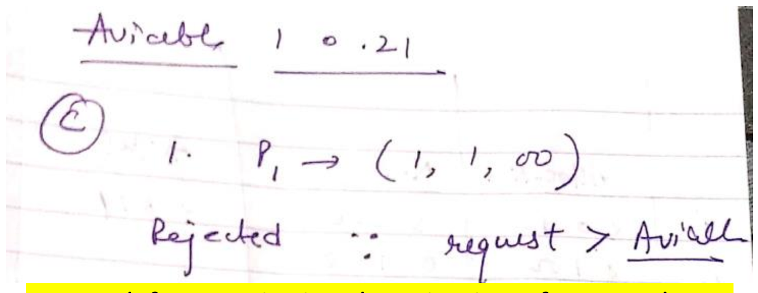
$P_4 \rightarrow P_3 \rightarrow P_1 \rightarrow P_2 \rightarrow P_5$

$P_4 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1$

$P_4 \rightarrow P_5 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_1$

Fixed

In Combination



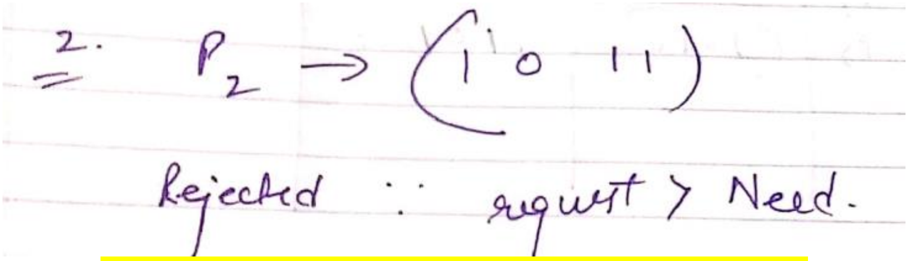
Aviable 1 0 21

(c)

1. $P_1 \rightarrow (1, 1, 00)$

Rejected ∵ request > Aviable

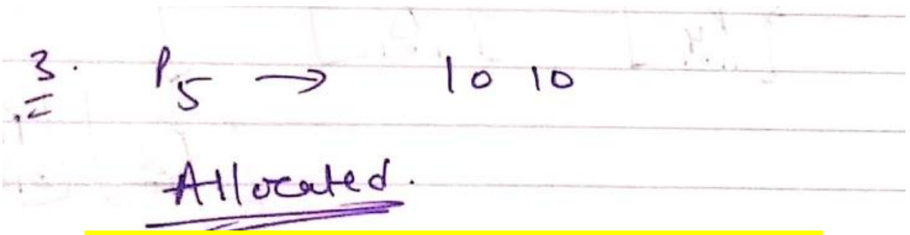
0.5 mark for mentioning the rejection of request by P1



2. $P_2 \rightarrow (1\ 0\ 1\ 1)$

Rejected $\because$ request > Need.

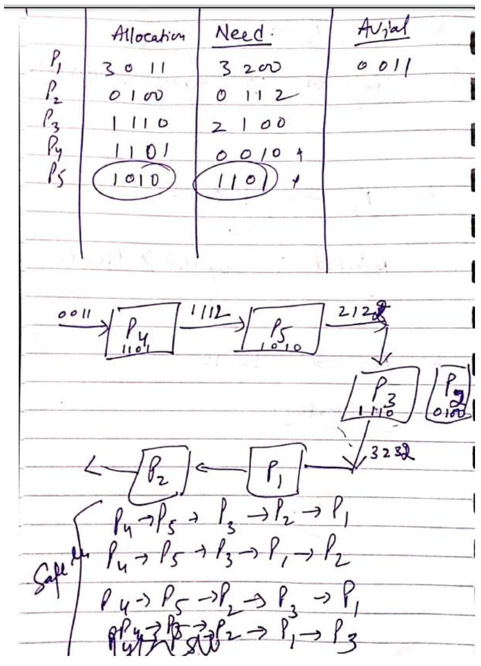
0.5 mark for mentioning the rejection of request by P2



3. $P_5 \rightarrow 1010$

Allocated.

0.5 mark for mentioning the acceptance of request by P5
0.5 mark for mentioning/computing the safe sequence



	Allocation	Need.	Avial
P_1	3 0 1 1	3 2 0 0	0 0 1 1
P_2	0 1 0 0	0 1 1 2	
P_3	1 1 1 0	2 1 0 0	
P_4	1 1 0 1	0 0 1 0	
P_5	1 0 1 0	1 1 0 1	

Safe seq.
$P_4 \rightarrow P_5 \rightarrow P_3 \rightarrow P_2 \rightarrow P_1$
$P_4 \rightarrow P_5 \rightarrow P_3 \rightarrow P_1 \rightarrow P_2$
$P_4 \rightarrow P_5 \rightarrow P_2 \rightarrow P_3 \rightarrow P_1$
$P_4 \rightarrow P_5 \rightarrow P_2 \rightarrow P_1 \rightarrow P_3$