Spring 2022

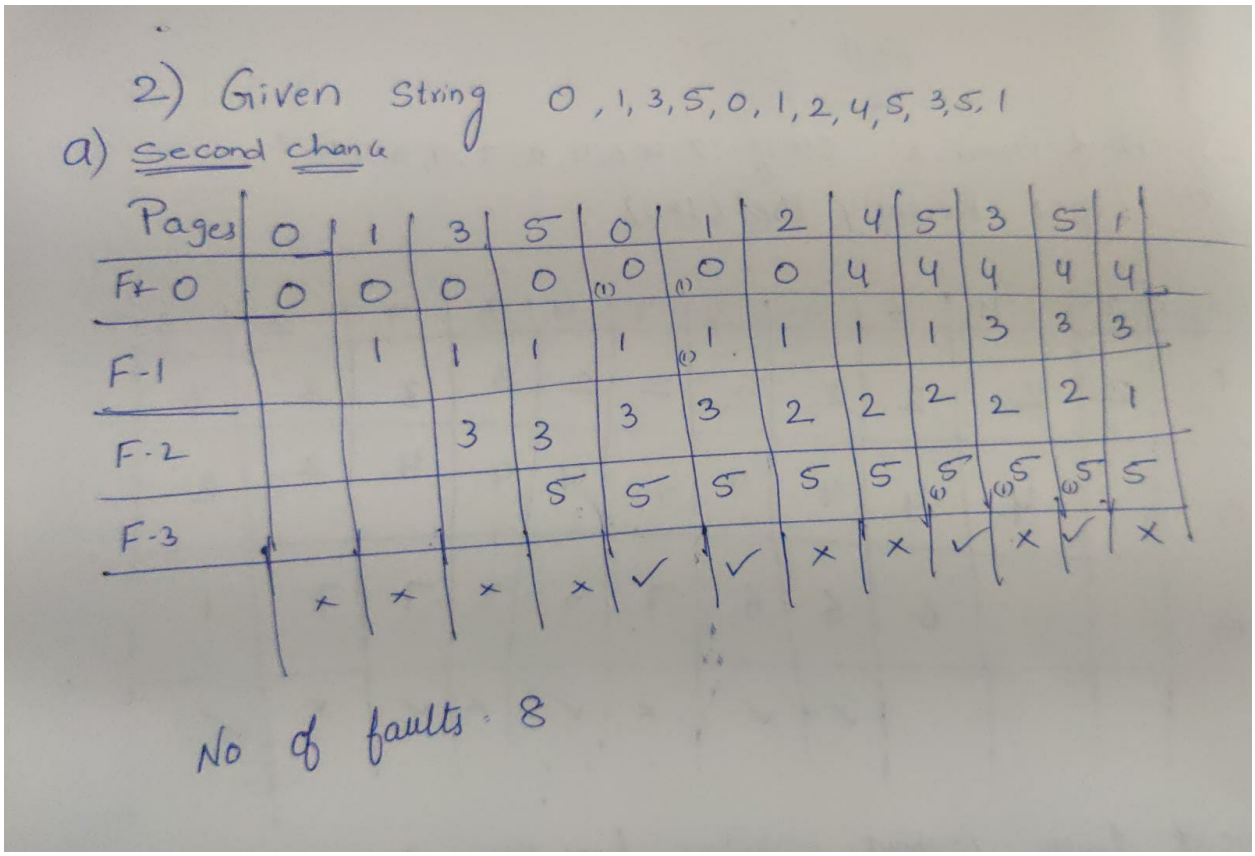
1. a) Three Standard goals for Mutual Exclusion problem for 2 process:

    1. Mutual Exclusion is Guaranteed
    2. Deadlock cannot occur
    3. Indefinite postponement cannot occur

    b) Indefinite postponement is occuring which is violating the goal.
    If Process p1 enters and runs the test_and_set and sets the lock to true and by the time it sets the lock to false p2 gets preempted and it keeps on happening every time lock is setting to false. So there is indefinite postponement.

    c) Mutual Exclusion is guaranteed.
    If Process p1 enters and runs the test_and_set and sets the lock to true. SO process p2 keeps waiting. Once p1 finishes execution it sets the lock value to false. Then p2 gets in after changing the lock value to true and finishes its execution. So at a time only one process is able to get into the critical section. So mutual exclusion is guaranteed.

2. a)



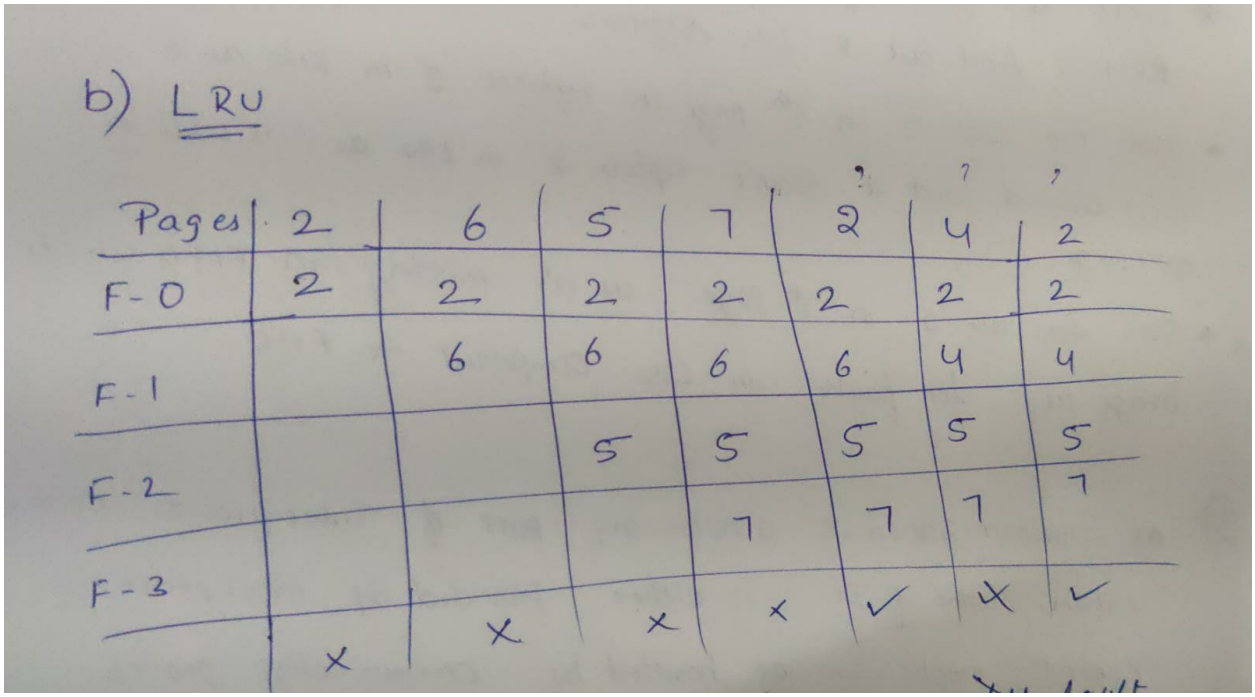
2) Given String 0, 1, 3, 5, 0, 1, 2, 4, 5, 3, 5, 1

a) Second chance

Pages	0	1	3	5	0	1	2	4	5	3	5	1
F-0	0	0	0	0	⓪ 0	⓪ 0	0	4	4	4	4	4
F-1		1	1	1	1	⓪ 1	1	1	1	3	3	3
F-2			3	3	3	3	2	2	2	2	2	1
F-3				5	5	5	5	5	⓪ 5	⓪ 5	⓪ 5	5
	x	x	x	x	✓	✓	x	x	✓	x	✓	x

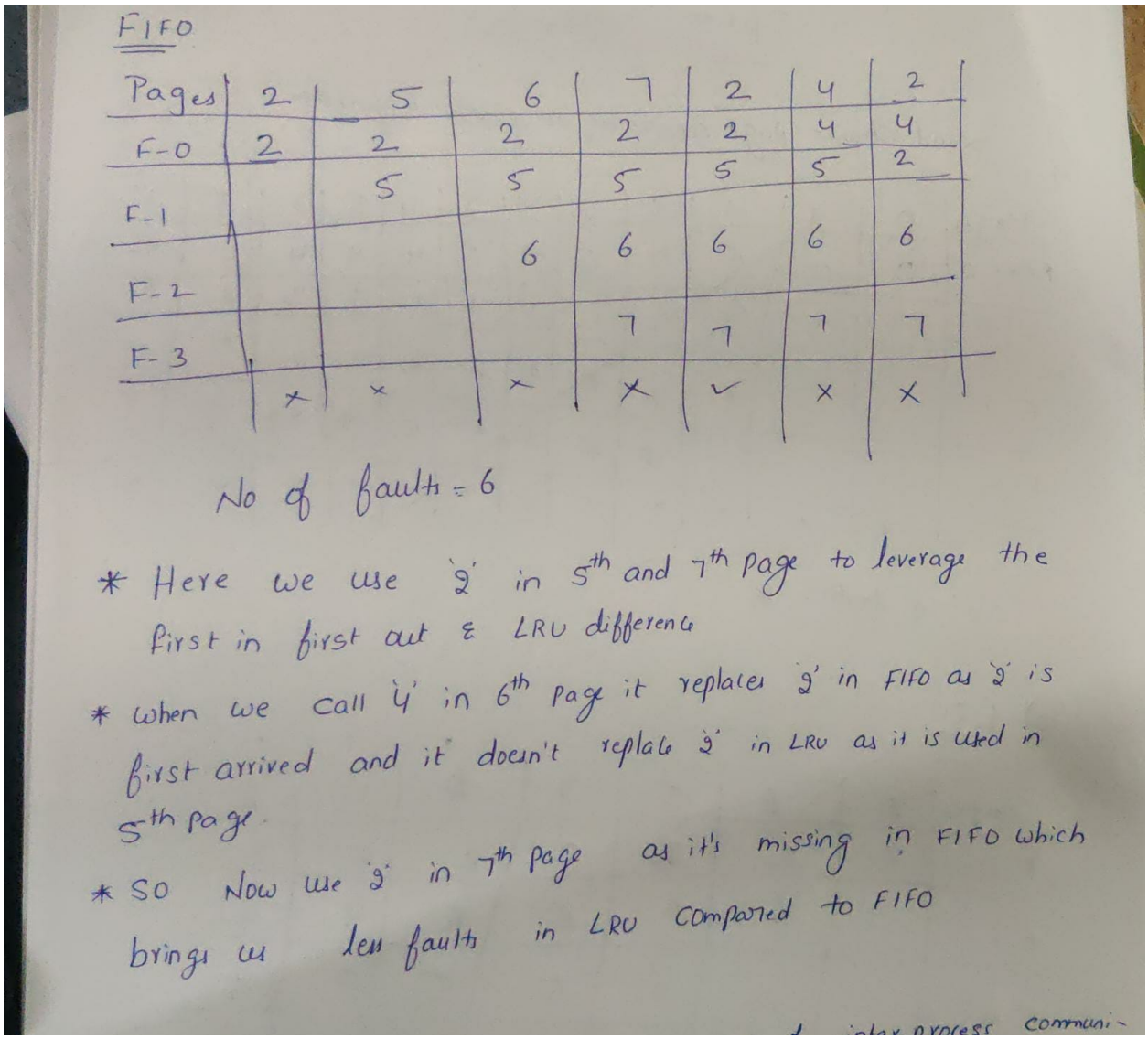
No of faults : 8

b)



b) LRU

Pages	2	6	5	7	2	4	2
F-0	2	2	2	2	2	2	2
F-1		6	6	6	6	4	4
F-2			5	5	5	5	5
F-3				7	7	7	7
	x	x	x	x	✓	x	✓



FIFO

Pages	2	5	6	7	2	4	2
F-0	2	2	2	2	2	4	4
F-1		5	5	5	5	5	2
F-2			6	6	6	6	6
F-3				7	7	7	7
	×	×	×	×	✓	×	×

No of faults = 6

* Here we use '2' in 5th and 7th page to leverage the first in first out & LRU difference
* when we call '4' in 6th page it replaces '2' in FIFO as '2' is first arrived and it doesn't replace '2' in LRU as it is used in 5th page.
* So Now use '2' in 7th page as it's missing in FIFO which brings us less faults in LRU compared to FIFO

inter process communi-

3. **A. Semaphore and Mutex were the two fundamental mechanisms that ensures the process synchronization.**

B. Test and Set is a hardware based solution

C. Short term schedulers are those who have short term effect on the perfomance. It is responsible for getting the process from ready state and scheduling it in running state
Long Term scheduler are those who have long term effect on the performance. It is responsible for getting the process from new state and scheduling it into ready state.
D. Context Switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point as earlier. This is a feature of a multitasking operating system and allows a single CPU to be shared by multiple processes.

Fall 2021

1. a) **<u>Race condition:</u>** Race condition occurs when multiple threads tries to access the same section. In the context of the reader/writer problem, race condition occurs when a writer writes on the file while a reader is reading the same file.

   b) Yes, this is the correct solution for the reader/writer problem.

   c) If lines 15 and 16 are swapped, it will **result in an incorrect solution.** In that case, mutex is released before readcount is compared with 1. Assume the first thread comes and increments readcount, then releases the mutex. Now, before thread 1 can compare it with 1, thread 2 comes and increments readcount (as mutex is free). Now when both of them compare readcount with 1, it will result in false, so wrt will never be held. So writers will be able to write along with readers, leading to race condition.

   d) If lines 18 and 21 are omitted, it will **result in no significant change**. Even if some reader thread modifies readcount in line 19 when some other thread is modifying it in line 14, the increment and decrement will cancel each other out, and won't affect the outcome.

2.) a) First Fit: Choose the first big enough block to accomodate the process.

212k, 417k, 112k, 300k, 150k
P1 P2 P3 P4 P5

Process	Memory Partition		
212k P1	500k P1 (500-212) 288k	300k	600k
417k P2	288k P1	300k	600k P2 (600-417)
112k P3	288k P1, P3 (288-112) 176k	300k	183k P2
300k P4	176k P1, P3	300k P4 (300k-300k) 0k	183k P2
150k	176k P1, P3, P5 176k-150k (26k)	0k P4 (0k)	183k P2 (183k)
final allocation	500k P1, P3, P5	300k P4	600k P2

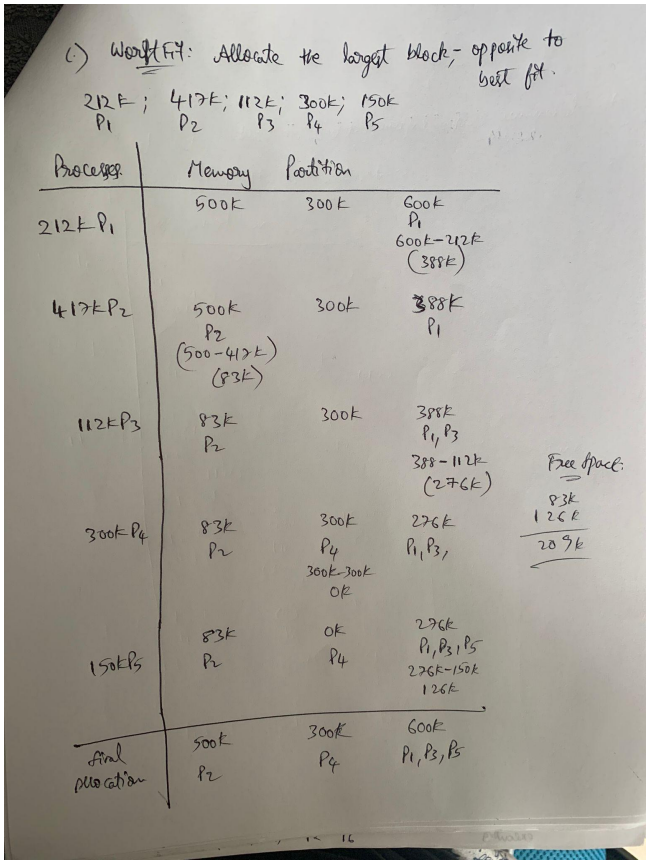
* free space:-
26k + 183k
209k

b) Best Fit:- Allocate the smallest block that is big enough

212k, 417k, 112k, 300k, 150k
P1 P2 P3 P4 P5

Process	Memory Partition		
212k P1	500k	300k 300-212 (88k)	600k
417k P2	500k P2 (500k-417k) 83k	88k P1	600k
112k P3	83k P2	88k P1	600k P3 (600-112k) 488k
300k P4	83k P2	88k P1	488k P4 488k-300k (188k)
150k P5	83k P2	88k P1	188k P3, P4, P5 (188k-150k) (38k)
final allocation	500k P2	300k P1	600k P3, P4, P5

Free space
183
+ 88
+ 38
209k



d) The most efficient algorithm is best fit, because it makes the most efficient use of the available memory and tries to reduce the number of holes.

e) **Internal fragmentation** occurs when the memory is distributed into fixed-sized blocks. If the memory allocated to the process is slightly larger than the memory demanded, then the difference between the allocated and demanded memory is internal fragmentation

f) **External fragmentation** occurs when the total unused memory space is enough to answer all the allocation requests. Here the memory is non-contiguous. Therefore, the

memory has empty blocks scattered all over. These memory blocks are insufficient to be allocated by other programs.

g) **Dis. Adv.**
Worst Fit: Slow process, it traverses all the partitions in the memory and then selects the largest partition among all the partitions.
Best Fit: Slow process, checking the whole memory for each job makes the working of the OS very slow.

3 a)

Mutual Exclusion: Only one process can be inside the critical section at anytime. If any process require one, should wait until free.

Deadlock: It is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function

Bounded Waiting: It means each processes must have a limited waiting time. It should not wait endlessly to access the critical section.

b) NO Mutual exclusion

c) Deadlock will occur when both process are caught inside the while loop infinitely waiting. Here such situation will never arrive. Thus deadlock is not possible here

d) *Doubt

Spring 2021

1. a) The three services that an Operating System provides:

1)Protection and security:
Protection involves ensuring that all access to system resources is controlled.
To make a system secure, the user needs to authenticate himself or herself to the system
2)File manipulation:
To create, delete, update the files and directories file manipulation services are used.
3)GUI:
A window system with a pointing device to direct I/O, choose from menus and make selections, and a keyboard to enter text

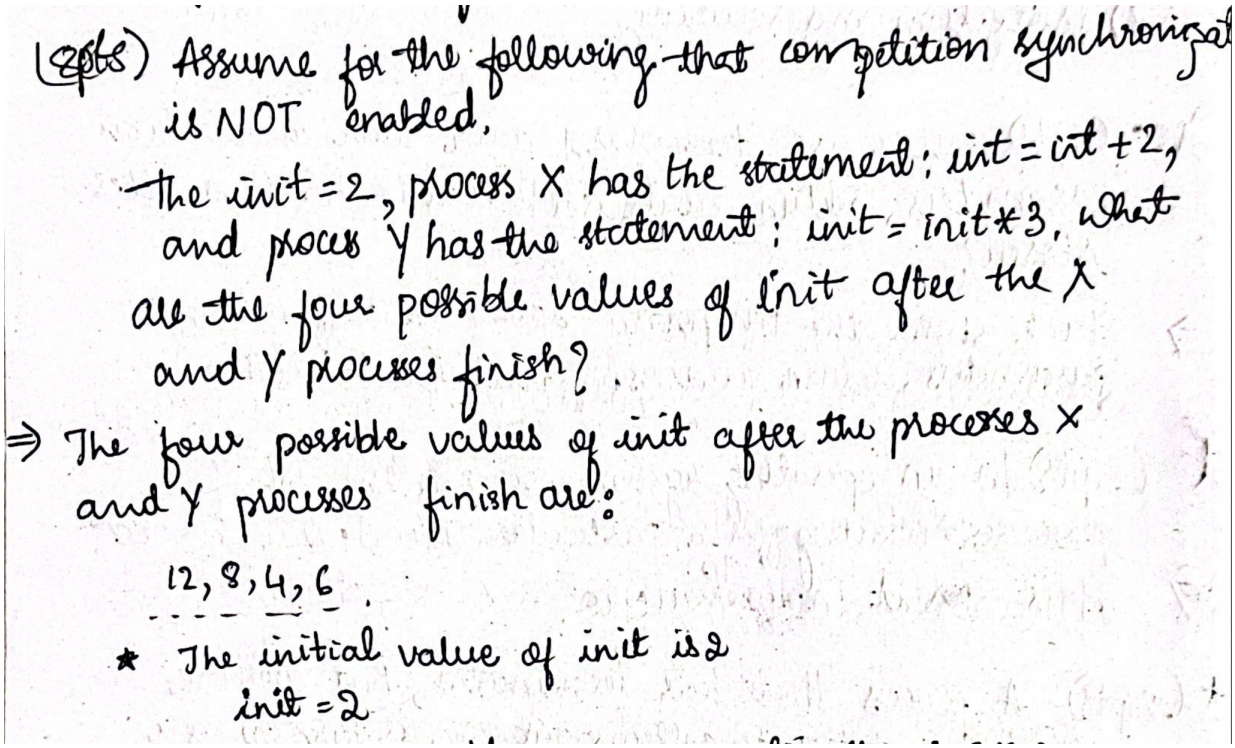
b) **Stack** stores the temporary data such as function parameters, return addresses, and local variables.

c) list of processes waiting for a particular I/O device stored in - **I/O queue**

c) A process that has terminated, but whose parent has not yet called wait(), is known as a **Orphan process**.

d) Race condition occurs when multiple threads tries to access the same section at a time. The solution mechanism to solve the race condition is:
Use Thread Synchronization: A given part of the program can only execute one thread at a time. Synchronization is the cooperative act of two or more threads that ensures that each thread reaches a known point of operation in relationship to other threads before continuing.

e)


(Q6) Assume for the following that competition synchronisat is NOT enabled.
The init = 2, process X has the statement: init = init + 2, and process Y has the statement: init = init * 3. What are the four possible values of init after the X and Y processes finish?
⇒ The four possible values of init after the processes X and Y processes finish are:
12, 8, 4, 6.
* The initial value of init is 2
init = 2

There are 4 possible ways of executing the processes.

process X : init = init + 2

process Y : init = init * 2

Case 1: Process X executes completely, the process Y executes next

process X executes: init = init + 2

init = 2 + 2 = 4

init = 4

Process Y executes: init = init * 3

init = 4 * 3

init = 12

The final value of init is 12.

Case 2: Process Y executes completely, then process X executes next.

process Y executes: init = init * 3

init = 2 * 3 = 6

init = 6

process X executes: init = init + 2

init = 6 + 2 = 8

init = 8

The final value of init is 8.

Case 3: Process X executes partially reads the original value of init, then X continues its execution and executes completely.

Process X reads the original value of init as 2.
init = 2 is the value read by X.

Process Y executes: init = init * 3
init = 2 * 3 = 6
init = 6

Process X executes: init = init + 2, but process X cannot see the latest value of init as updated by Y, since it already read the old value of init, it uses the old value. This scenario is called a lost update, this occured due to context switches in the operating system and there is no synchronization enabled.

init = 2 + 2 = 4
init = 4

The final value of init is 4

Case 4: Process Y executes partially reads the original value of init, then process X executes completely, then Y continues its execution and executes completely.

process Y reads the original value of init as 2: init = 2 is the value read by Y.

process X executes: init = init + 2

init = 2 + 2 = 4

init = 4

Process Y executes: init = init * 3, but process Y cannot see the latest value of init as updated by X, since it already read the old value of init, it uses the old value.

init = 2 * 3 = 6

init = 6

The final value of init is 6

2.

(4 pts) give two (2) solutions for breaking deadlock among processes.

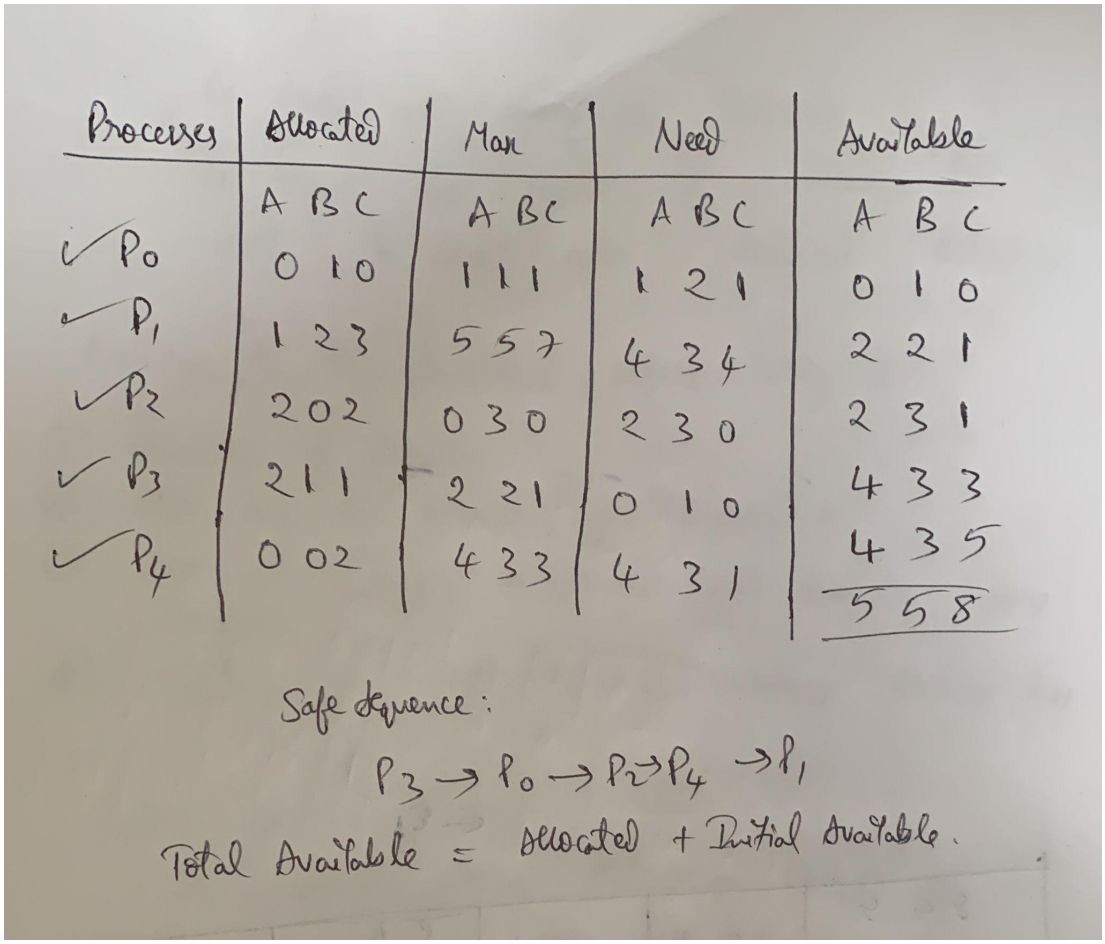
⇒ There are 2 options for breaking deadlock:

1. to abort one or more processes to break circular wait.
2. to preempt some resources from one or more of the deadlocked processes.

in first approach → In this we terminate 1 or more deadlocked processes for deadlock recovery. we can terminate either 1 deadlocked process or all deadlocked processes until we recover from deadlock. if we remove all deadlocked processes then system is free from deadlock but if we remove 1 deadlocked process at a time we need to have extra overhead for checking either system is in deadlock or not.

in second approach ⇒ this involves recovery by checkpoint and rollback. Checkpointing is process of saving state of process so that process can restart from checkpoint later. restarting a process is called rollback.

-> A state is safe if the system can allocate all resources requested by all processes without entering a deadlock state.



Processes	Allocated	Max	Need	Available
	A B C	A B C	A B C	A B C
P_0	0 1 0	1 1 1	1 2 1	0 1 0
P_1	1 2 3	5 5 7	4 3 4	2 2 1
P_2	2 0 2	0 3 0	2 3 0	2 3 1
P_3	2 1 1	2 2 1	0 1 0	4 3 3
P_4	0 0 2	4 3 3	4 3 1	4 3 5
				5 5 8

Safe sequence:

$P_3 \rightarrow P_0 \rightarrow P_2 \rightarrow P_4 \rightarrow P_1$

Total Available = Allocated + Initial Available.

-> Let the given request for P3 is (1, 1, 0):
This leads system to unsafe state as the need should be less than available.

Or

-> Let the request from P3 arrives for B:
This also leads system to unsafe state

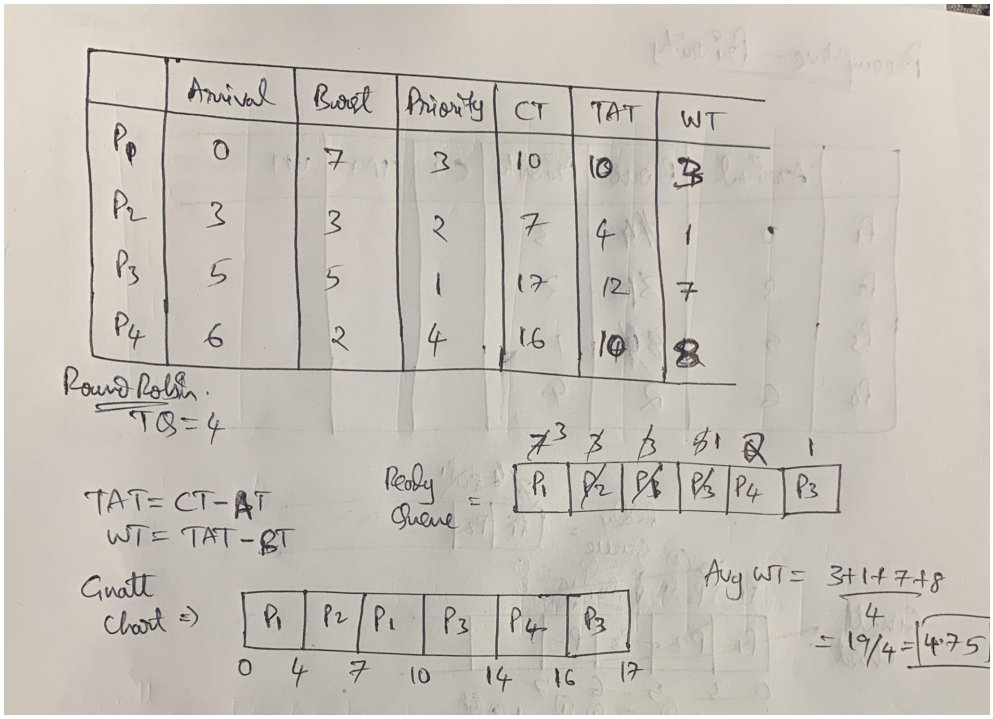
3.

-> When time quantum is too large Robin Robin scheduling algorithm behaves like FCFS

Convoy effect might occur

-> SJF performs best in general

Its mode is non pre-emptive and criteria is Burst time. It gives maximum throughput and minimum avg waiting time and Turn Around time.



	Arrival	Burst	Priority	CT	TAT	WT
P_1	0	7	3	10	10	3
P_2	3	3	2	7	4	1
P_3	5	5	1	17	12	7
P_4	6	2	4	16	10	8

Round Robin.
TQ = 4

TAT = CT - AT
WT = TAT - BT

Ready Queue = P_1 P_2 P_3 P_4 P_3

Gantt Chart => P_1 | P_2 | P_1 | P_3 | P_4 | P_3

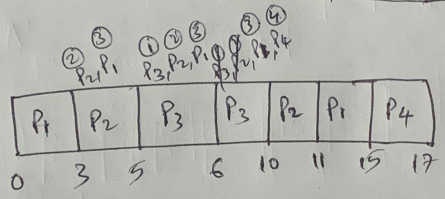
0 4 7 10 14 16 17

Avg WT = (3+1+7+8)/4 = 19/4 = 4.75

Preemptive - Priority:

	Arrival	Burst	Priority	CT	TAT	WT
P_1	0	~~10~~	3	15	15	8
P_2	3	~~8~~	2	11	8	5
P_3	5	~~5~~	1	10	5	0
P_4	6	2	4	17	11	9

Gnatt Chart :-



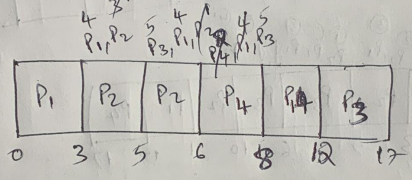
Avg TAT = $\frac{39}{4}$ = 9.75 Sec

Avg WT = $\frac{23}{4}$ = 5.5 Sec

Preemptive – SJF

	Arrival	Burst	Priority	CT	TAT	WT
P_1	0	~~4~~	3	12	12	5
P_2	3	3	2	6	3	0
P_3	5	5	1	17	12	7
P_4	6	2	4	8	2	0

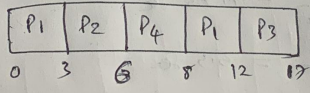
Gantt Chart :-



Avg TAT = $\frac{29}{4}$ = 7.25.

Avg WT = $\frac{12}{4}$ = 3.

Cs



Fall 2020

1 a) The mentioned action will sometimes increase the number of page faults with the FIFO page replacement. This is because increasing the page frames will not always decrease the page fault and this is known to be Belady's Anomaly. Belady's Anomaly results when increasing the page frames leads to an increase in page faults.

b)

6 bits for offset

So, page size = 2^6 = 64Bytes

Number of pages in a virtual memory = 2^{32-6} = 2^26

Virtual memory access = number of pages * page size

Virtual memory access = 2^26 * 64Bytes = 4GB,/??

c)

Firstly, here page size

= 16 KB

= (2^4 * 2^{10}) bytes

= 2^{14} bytes

Thus,

- Lowest 14 bits of virtual address denote the page-offset.
- Remaining highest (32 - 14 =) 18 bits of virtual address denote the page-number.

Now, TLB may contain the following forms :

Direct-Mapped :

Since total number of entries is 512 = 2^9, so in this case :

- Lowest 9/18 bits of page-number, will serve as TLB-Index.
- Remaining 9/18 bits of page-number, will serve as TLB-Tag.

Fully-Associative :

In this case, there is no indexing, and any page <=> frame mapping can be put at any place.

Thus, number of bits in TLB-Tag is 18.

k-Way-Associative :

Here, the number of bits in TLB-Tag will be in between the above two values.

For example, if k = 2, then

- Number of sets, to which indexing happens = 512 / 2 = 256 = 2^8
- Thus,
  - Lowest 8/18 bits of page-number, will serve as TLB-Index.
  - Remaining 10/18 bits of page-number, will serve as TLB-Tag.

For example, if k = 4, then

- Number of sets, to which indexing happens = 512 / 4 = 128 = 2^7
- Thus,
  - Lowest 7/18 bits of page-number, will serve as TLB-Index.
  - Remaining 11/18 bits of page-number, will serve as TLB-Tag.

Thus, the minimum size of TLB Tag is

9 bits

d) Effective Access Time is given by:

EAT = hit ratio *(TLB Access time + Memoery access time) + miss ratio*(TLB access time + (L+1) * memory access time)

L= number of page table level

Given:

Number of levels of page table = 4

TLB access time = 80 ns

Main memory access time = 160 ns

TLB Hit ratio = 92% = 0.92

TLB Miss ratio

= 1 – TLB Hit ratio

= 1 – 0.92

= 0.08

Effective Access Time-

= 0.92 x { 80 ns + 160 ns } + 0.08 x { 80 ns + (4+1) x 160 ns }

= 0.92 x { 240 ns } + 0.08 x { 80 + 800 ns}

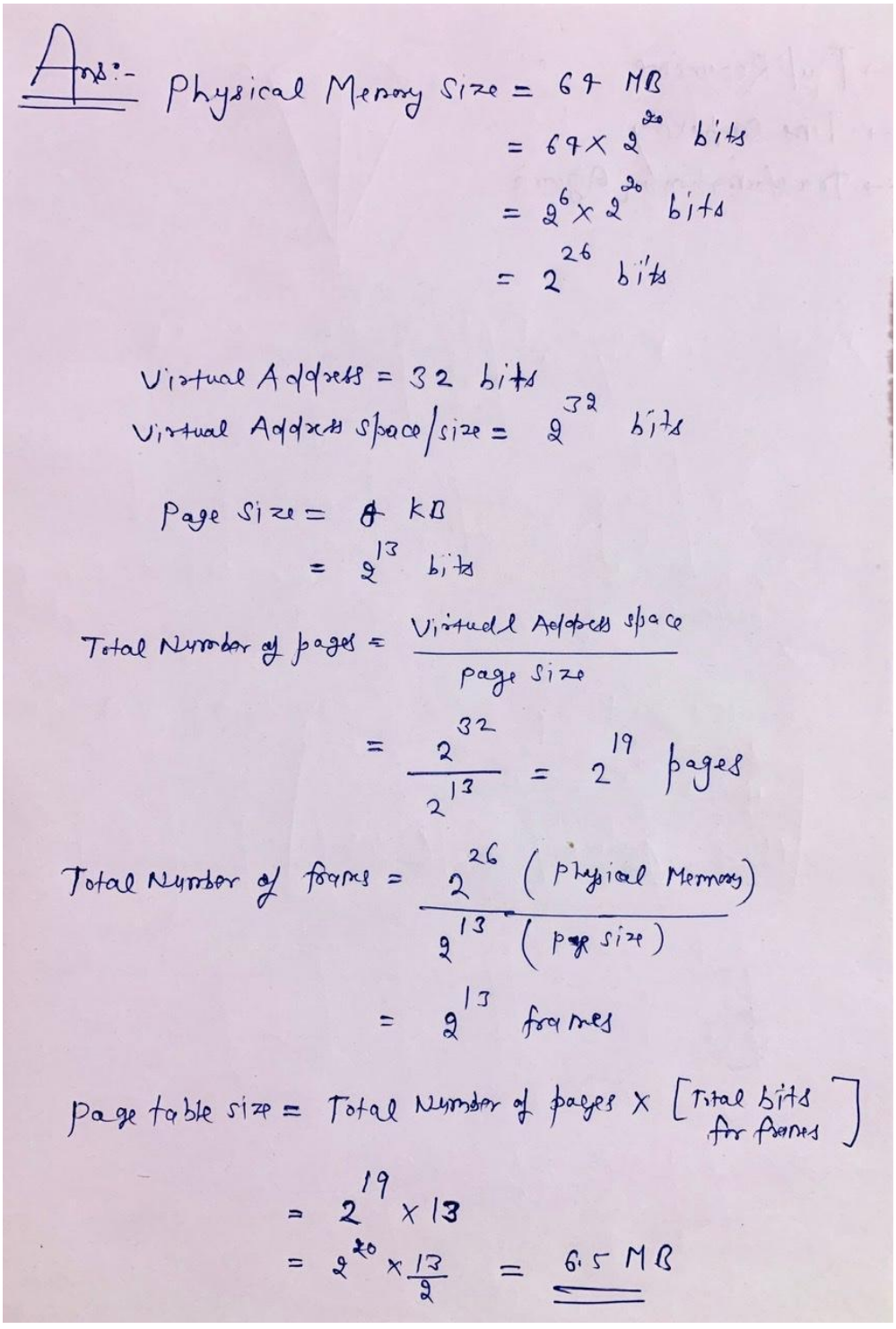
= 0.92 x 240 ns + 0.08 x 880 ns

= 220.8 + 70 40

= 291.2 ns

Effective access time = 291.2

e)



2 a)

Deadlock can arise if following four condition hold simultaneously

1. **Mutual exclusion**
2. **Hold and wait**
3. **No preemption**
4. **Circular wait**

Mutual exclusion

- Only one process at a time can use a resource
- If process p1 is using a resource R at a particular instant of time,then some other process P2 can't uses the same resource at that same instant of time

Hold and wait

- There must exist a process which holding atleast one resources and waits for additional resource held by some other processes.
- A process P1 can hold two resources R1 and R3 and request and wait for resource R3 held by other process P2.

No preemption

- A resource can only be released by the process holding it, after the process has completed its task
- If P1 process is using resource R, then P2 process can't use resource R until the P1 is completed

Circular Wait

- There exists a set {p0,p1,p2,p3.....pn} of waiting processes such that p0 is waiting for a resource held by p1, p1 is waiting for resource held by p2..,Pn-1 is waiting for a resource that held by pn, and pn is waiting for a resource that is held by p0.

b) This statement is a True.

Release all the resources before making a new request is a valid deadlock prevention scheme .it avoids hold and wait and starvation problem.

Hold and wait is situation where a process is holding a resource and waiting for the other resource.

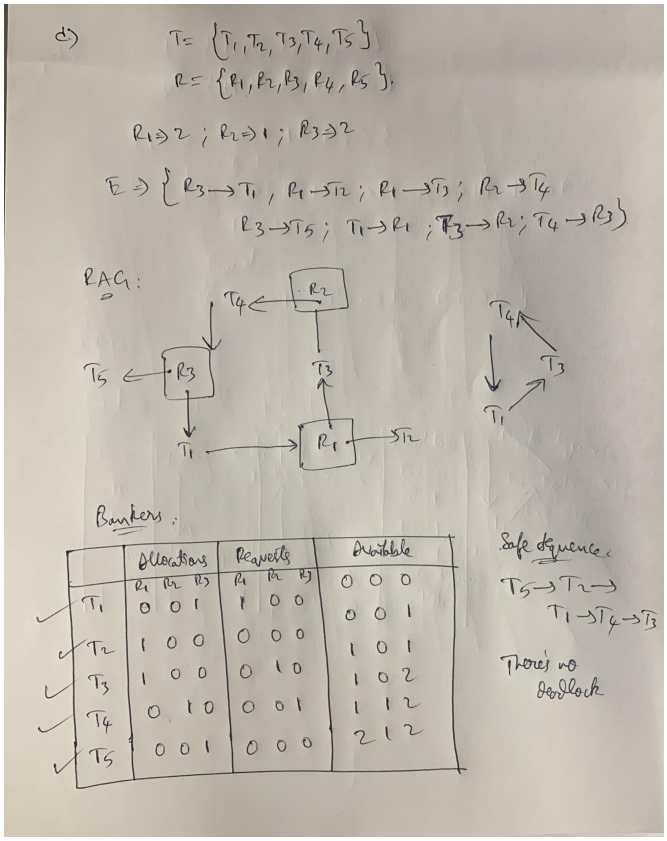
Releasing the resources before making a new request helps the other process to access the resources .if not the process has to wait untill the process release the granted resources .

Suppose p1 is assigned with resource r1 and once it completed with the process release the resources and request for resource r2 .the r1 resource can be immediately allocated to the process p2(requested for r1).

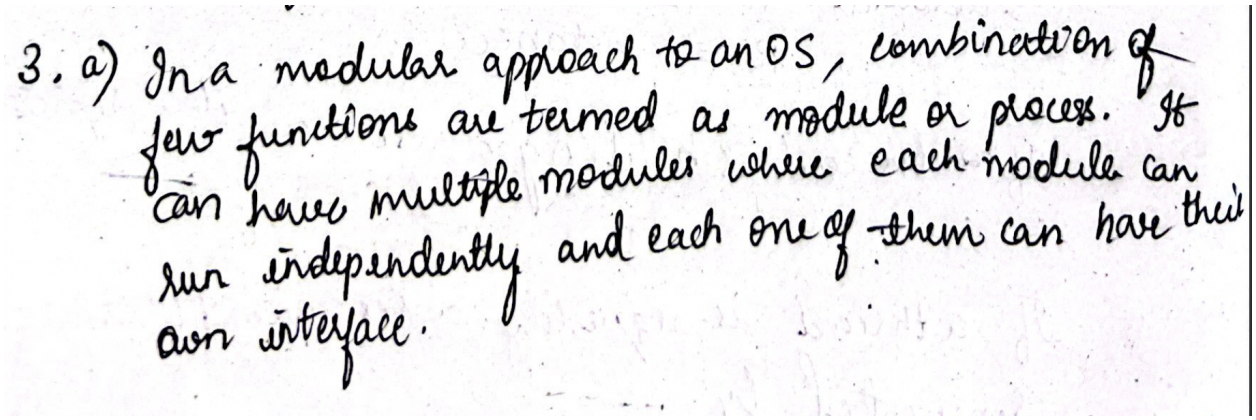
c)

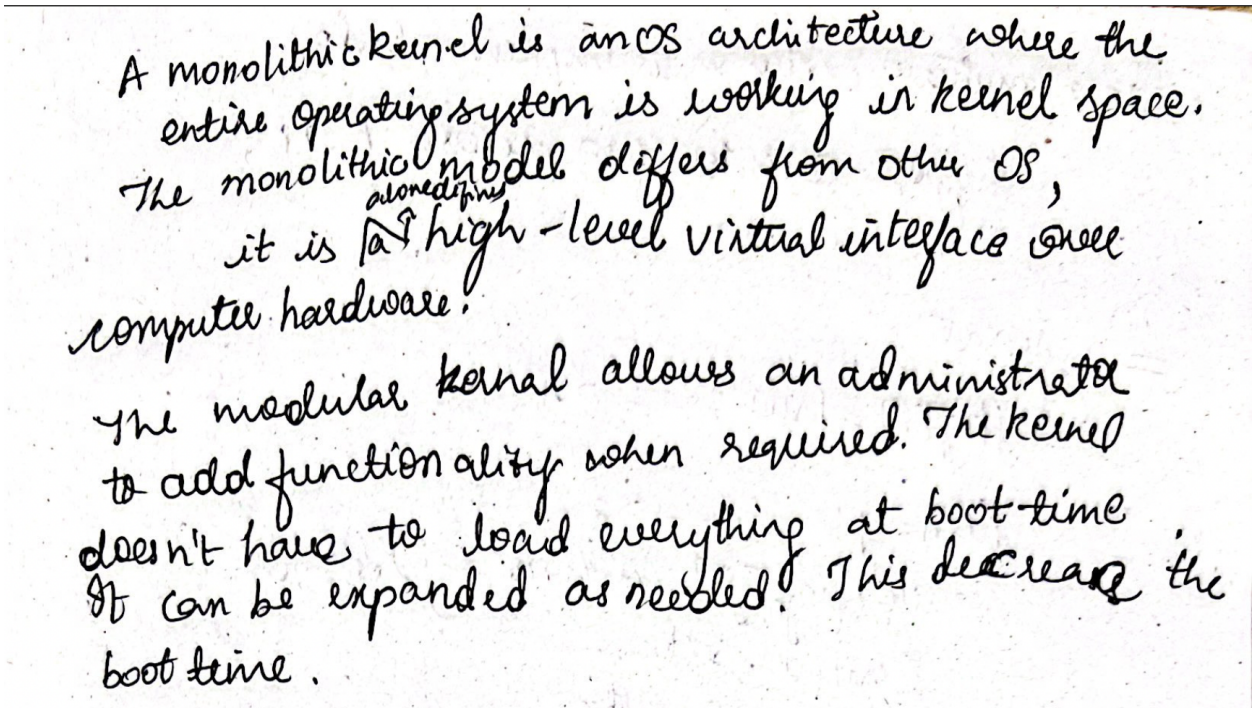
The given statement is True. Because in RAG where there is a single instance in each resource, a cycle indicates that the resource of each process is held by some other process in the cycle.

d)



3. a)





b) The statement is false . never requesting a resource after releasing a resource is NOT a valid deadlock prevention scheme .

The operating system requests a resource after releasing the resource held by the one, if required in the deadlock prevention system . This is not a condition that a process cannot request a resource.

c)

3c)

```
repeat
flag[i] = true;
turn = J;
while (flag(J) && turn == J) do; // J is another process.
    // critical section
    flag[i] = false;
    // non-critical section
until false;
```

Mutual Exclusion: It is a property which defines that "no two processes can exist in the critical section at any given point of time".

How mutual exclusion is implemented?

Suppose consider two processes P_i, P_j

P_i

repeat

flag[i] = true ; ⟶ A

turn = j; ⟶ B

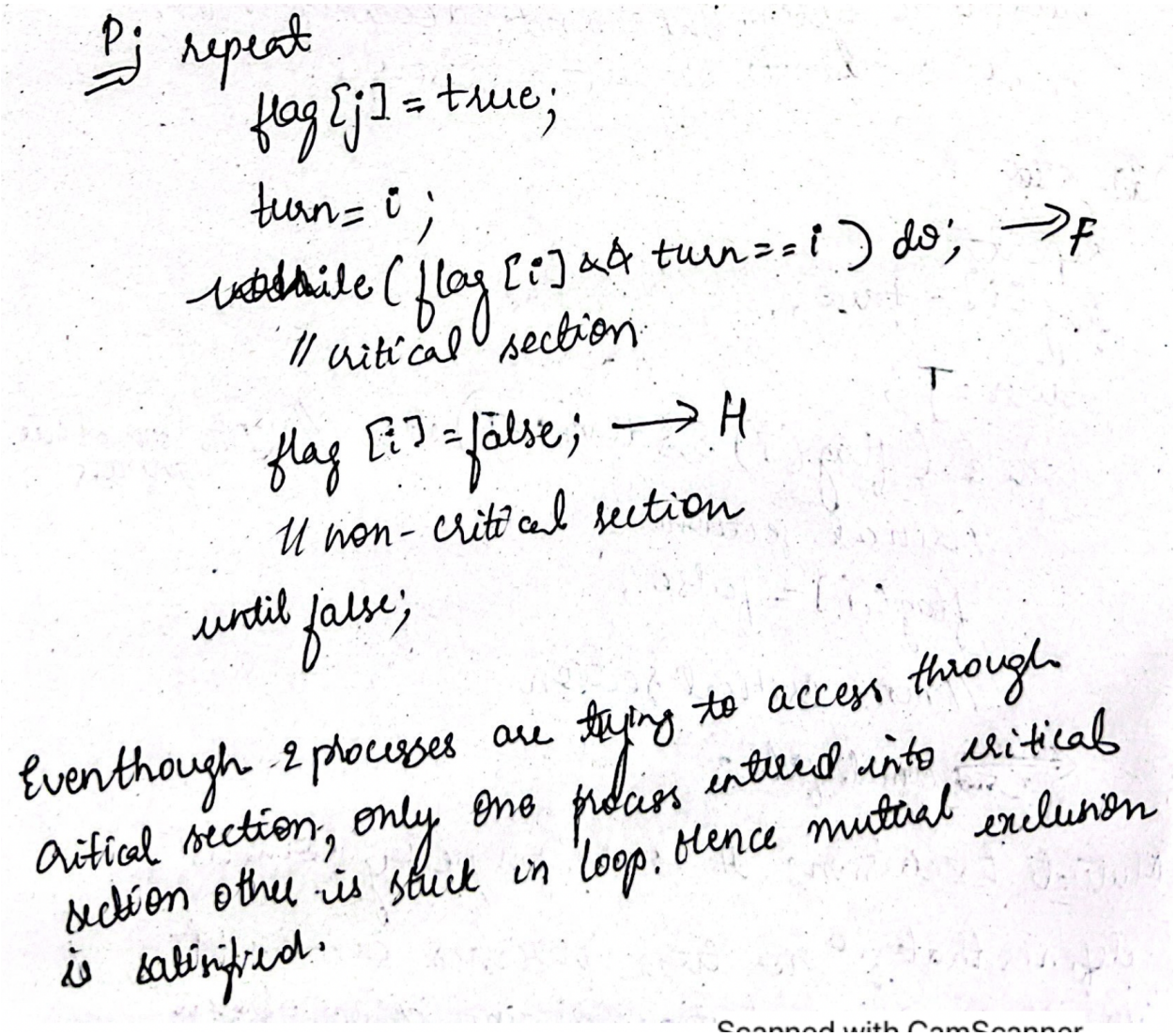
while (flag[j] && turn == j) do; ⟶ C

// critical section

flag[i] = false ; ⟶ G

// non-critical section

until false;



d) Mutual Exclusion is guaranteed.
If Process p1 enters and runs the test_and_set and sets the lock to true. SO process p2 keeps waiting. Once p1 finishes execution it sets the lock value to false. Then p2 gets in after changing the lock value to true and finishes its execution. So at a time only one process is

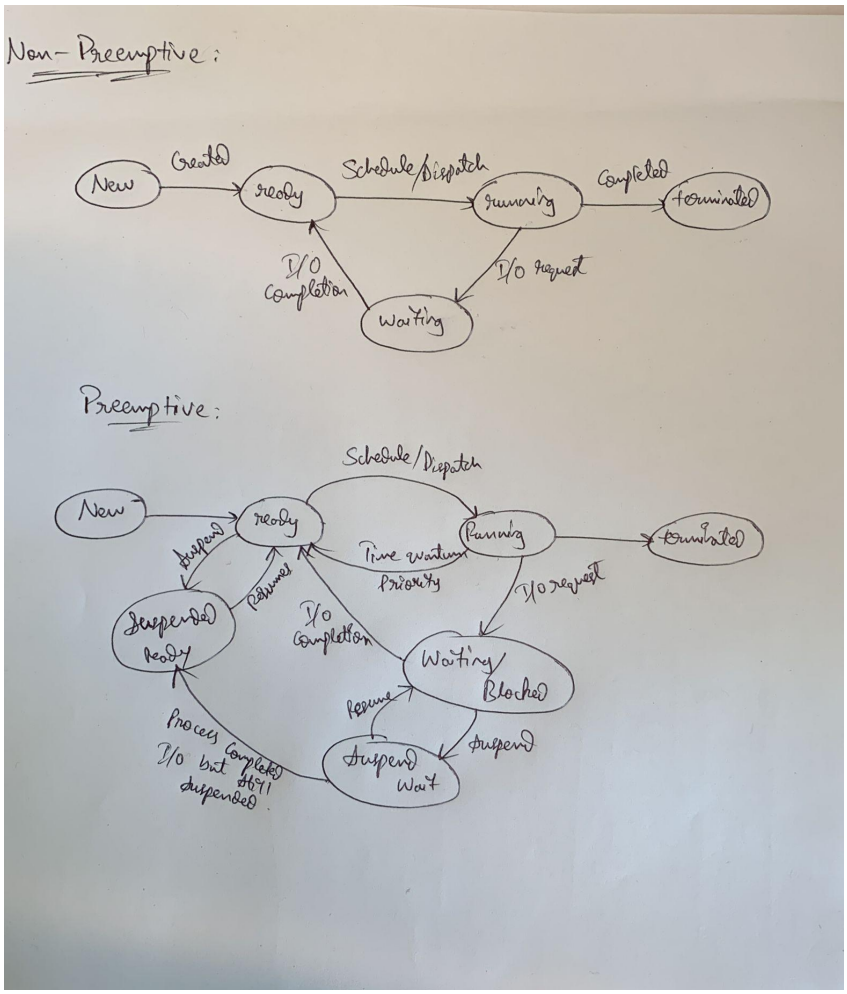
-> **Busy waiting** means that a process is waiting for a condition to be satisfied in a tight loop without terminating it.

-> **Indefinite postponement** is occuring which is violating the goal.

If Process p1 enters and runs the test_and_set and sets the lock to true and then gets pre-empted. If process p2 starts it will be waiting for indefinite time to get into the critical section.

Spring 2020

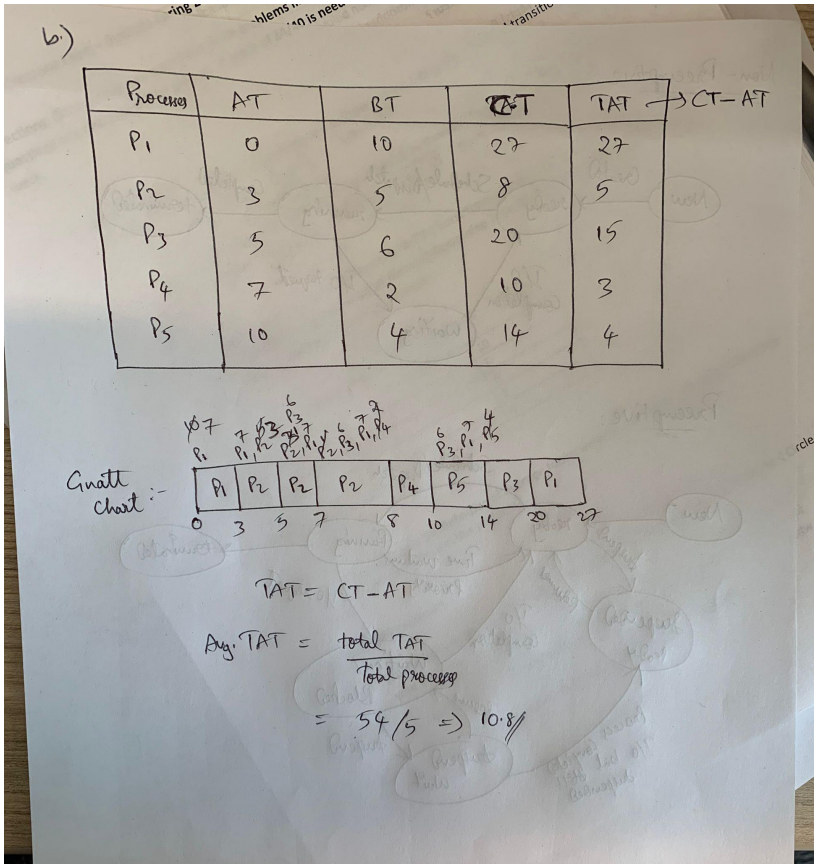
1 a)



a) Possible states of process

1. New state - The process is being created.
2. Ready state - The process is waiting to be assigned to the processor.
3. Running state - The process is being executed.
4. Waiting state - The process is waiting for some event to occur.
5. Terminate state - The process has finished execution.
6. Suspend Ready state - The process remains in the state until the main memory becomes available.
7. Suspend Wait state - When waiting state is not able to occupy more states, then some states are suspended in Suspend Wait state.

b) A thread has its own stack and register stack.
A thread shares resources with other threads of the same process : code, data, file pointers.



b)

Process	AT	BT	CT	TAT → CT − AT
P_1	0	10	27	27
P_2	3	5	8	5
P_3	5	6	20	15
P_4	7	2	10	3
P_5	10	4	14	4

Gnatt chart :-

TAT = CT − AT

Avg. TAT = $\frac{\text{total TAT}}{\text{Total process}}$

= 54/5 ⇒ 10.8

2 a) Starvation

b)

Deadlock can arise if following four condition hold simultaneously

1. **Mutual exclusion**
2. **Hold and wait**
3. **No preemption**
4. **Circular wait**

Mutual exclusion

- Only one process at a time can use a resource
- If process p1 is using a resource R at a particular instant of time,then some other process P2 can't uses the same resource at that same instant of time

Hold and wait

- There must exist a process which holding atleast one resources and waits for additional resource held by some other processes.
- A process P1 can hold two resources R1 and R3 and request and wait for resorce R3 held by other process P2.

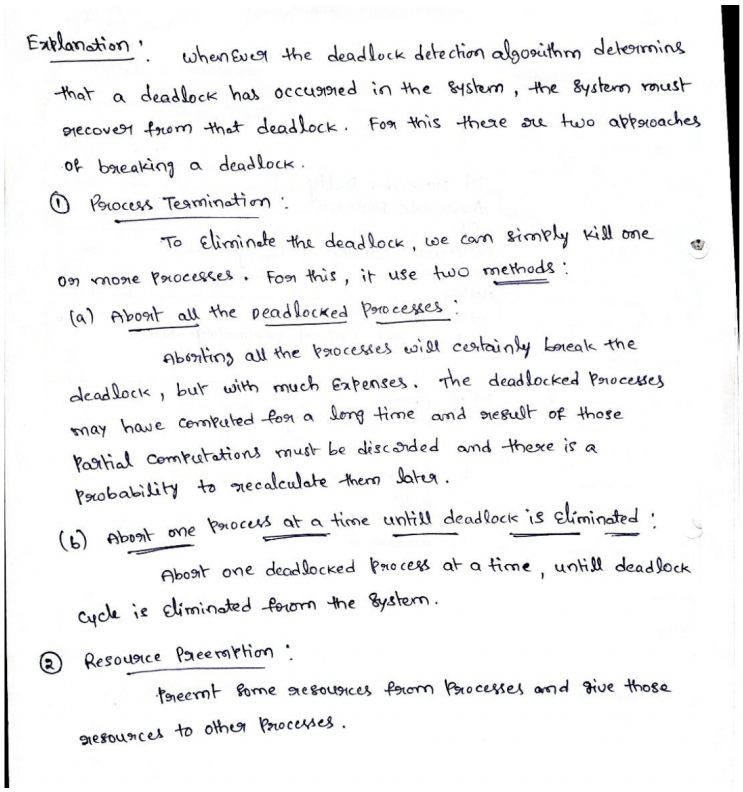
No preemption

- A resource can only be released by the process holding it, after the process has completed its task
- If P1 process is using resource R, then P2 process can't use resource R until the P1 is completed

Circular Wait

- There exists a set {p0,p1,p2,p3.....pn} of waiting processes such that p0 is waiting for a resource held by p1, p1 is waiting for resource held by p2..,Pn-1 is waiting for a resource that held by pn, and pn is waiting for a resource that is held by p0.

C.



Explanation: Whenever the deadlock detection algorithm determins that a deadlock has occurred in the system, the system must recover from that deadlock. For this there are two approaches of breaking a deadlock.

① Process Termination:

To eliminate the deadlock, we can simply kill one or more processes. For this, it use two methods:

(a) Abort all the deadlocked processes:

Aborting all the processes will certainly break the deadlock, but with much expenses. The deadlocked processes may have computed for a long time and result of those partial computations must be discarded and there is a probability to recalculate them later.

(b) Abort one process at a time untill deadlock is eliminated:

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

② Resource Preemption:

Preempt some resources from processes and give those resources to other processes.

D. Doubt

A function wait(), when implemented using LIFO or Last In First Out, can cause starvation. In LIFO, the last entered event in the queue is executed first.

For example -

There's a stack where a few processes entered in the wait() function in order - A -> B -> C.

STACK - Top [C, B, A] Bottom

Now, C popped from the stack and goes into execution state.

STACK - Top [B, A] Bottom

Meanwhile, some more processes entered in the order - D -> E -> F -> G -> H

STACK - Top [H, G, F, E, D, B, A] Bottom

Now, H popped from the stack and goes into execution state.

STACK - Top [G, F, E, D, B, A] Bottom

Meanwhile, some more processes were entered.

The thing to notice here is the processes entered initially are at the bottom of the stack such as A and B. However, the newer processes will be executed quickly. But, if more processes continuously keep on entering the stack, the processes at the bottom will go into starvation. They will never get a chance to go to the execution state.

E.

This is why LIFO can cause starvation.

3.

First Fit

In case of first fit we allocate the first free partition that is large enough to store the process.

Advantage

It is the Fastest algorithm because it searches as early as possible.

Disadvantage

In the end the unused memories left after allocation become waste. Thus it request for larger memory cannot be fulfilled.

Best Fit

In case of best fit we allocate the smallest free partition which meets the requirement of the process. This algorithm first searches the entire list and find the smallest portion that is enough to store the process .

Advantage

Memory utilization is optimized than first fit because it searches the smallest free partition first available.

Disadvantage

It is slower then the others because it need to travel the whole list to find the best position.

please give a upvote if u feel helpful.

b.

Thrashing is a condition or a situation when the system is spending a major portion of its time servicing the page faults, but the actual processing done is very negligible.

- If a single process is too large for memory, there is nothing the OS can do. That process will simply thrash.
- If the problem arises because of the sum of several processes:
    - Figure out how much memory each process needs.
    - Change scheduling priorities to run processes in groups that fit comfortably in memory: must shed load.

Spring 2016

1)Out of Syllabus

2)

The following is the solution:

Given both code for consumer and procedure .

So considering each choice one by one we have

$\mathbf{A(n),\ B = 0\ , C = 0}$

Here we have there semaphore.

$\mathbf{A(n)}$ = It must be empty purpose ,So from the local.

in procedure we can not perform wait (c) also in the same consumer we can not perform wait (B) and wait(c) because to perform the value must be 1.

So this choice is false because of error.

2-> A=n, B= 0, (=1

Procedure code => It can execute line I then line 2 wait (A) and wait (c) then it can also Signal (C) and (B) easily

Consumer code After procedure code over the signal in C result in the Execution of the lode as well so this and Soln in free from free. fee from deadlock and error free.

Final answer

3-> A=0, B=n C=1

Producer code: It will Not Execute because A=0 so Signal must be important for A which is Only done by consumer.

so this choice is also wrong because. first produce Code Need to be Execute then consumer if there in Nothing to Consume then it does Not Consume

So there is a error in this code.

A=0,B=0, C=1

Produce code: Again due to A=0 the produce code won't run until consumer code do. So it is also wrong. because of error.

A=n, B= 0, C= 2.

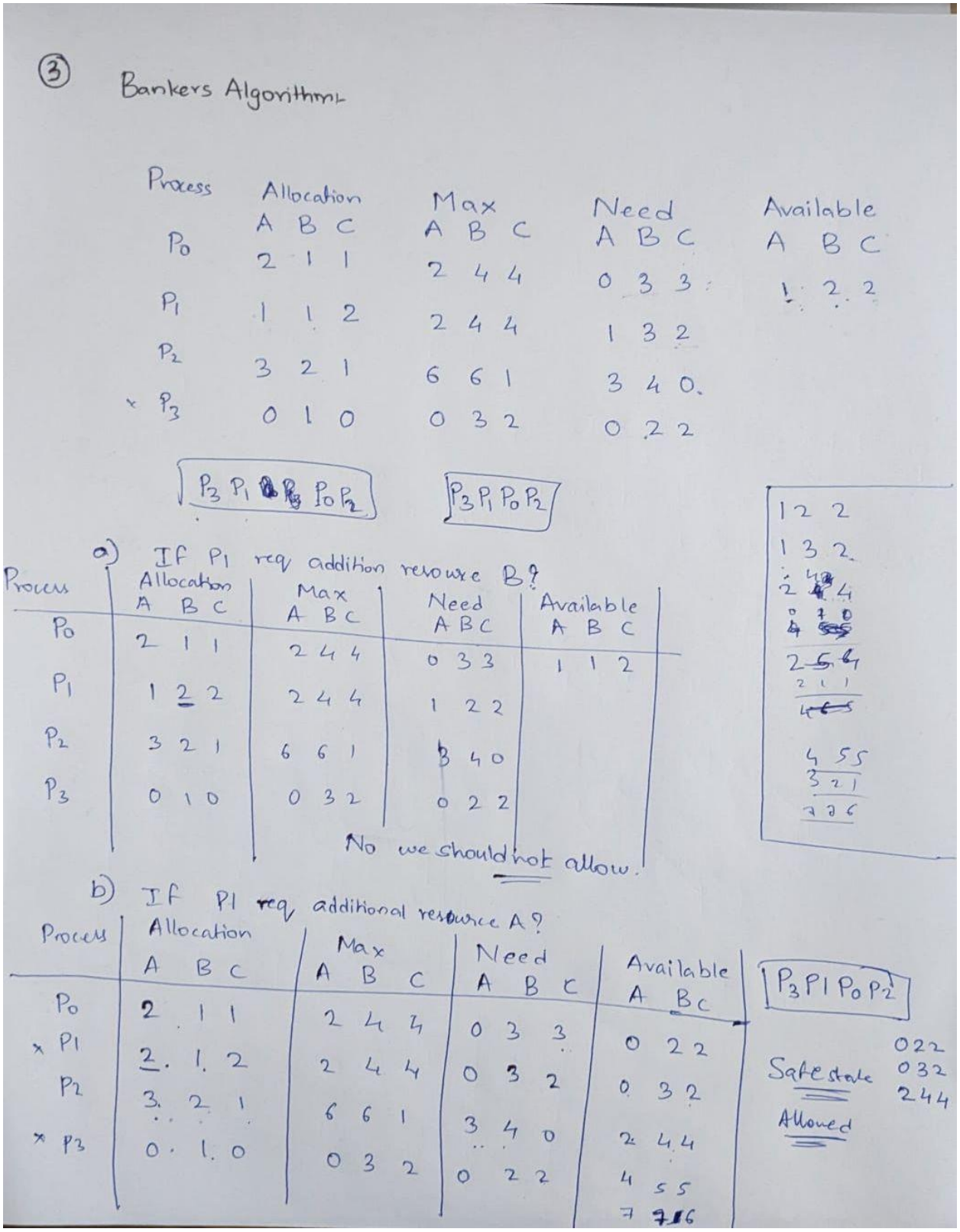
Producer code : Wait (A) wait(() Signal (() signal 18 Consumer code: 7 wait (B), wait (0) Signal (0) Signal (A)

The code is error free.

Now check for dead lock.

Produce Code: wait(A),wait(c), Signal(B),Signal (c)

3)Bankers Algorithm



③ Bankers Algorithm

Process	Allocation A B C	Max A B C	Need A B C	Available A B C
P_0	2 1 1	2 4 4	0 3 3	1 2 2
P_1	1 1 2	2 4 4	1 3 2	
P_2	3 2 1	6 6 1	3 4 0	
× P_3	0 1 0	0 3 2	0 2 2	

$P_3 P_1 P_0 P_2$ $P_3 P_1 P_0 P_2$

1 2 2
1 3 2
2 4 4
4 5 5
3 2 1
7 7 6

a) If P_1 req addition resource B?

Process	Allocation A B C	Max A B C	Need A B C	Available A B C
P_0	2 1 1	2 4 4	0 3 3	1 1 2
P_1	1 2 2	2 4 4	1 2 2	
P_2	3 2 1	6 6 1	3 4 0	
P_3	0 1 0	0 3 2	0 2 2	

No we should not allow.

b) If P1 req additional resource A?

Process	Allocation A B C	Max A B C	Need A B C	Available A B C
P_0	2 1 1	2 4 4	0 3 3	0 2 2
× P_1	2 1 2	2 4 4	0 3 2	0 3 2
P_2	3 2 1	6 6 1	3 4 0	2 4 4
× P_3	0 1 0	0 3 2	0 2 2	4 5 5
				7 7 6

$P_3 P_1 P_0 P_2$

Safe state
Allowed

022
032
244

Winter 2016

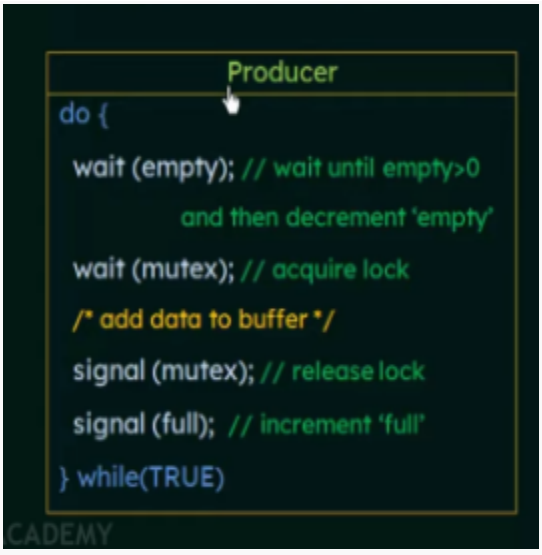
1) Out of Syllabus'
2)

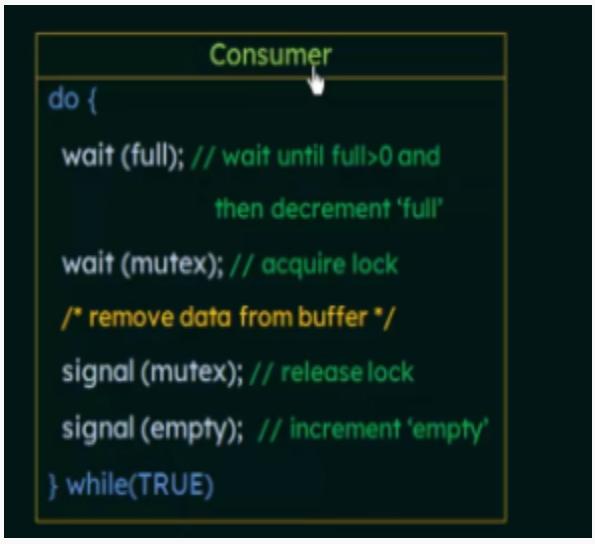
- The producer tries to insert data into an empty slot of the buffer.
- The consumer tries to remove data from a filled slot in the buffer.
- The Producer must not insert data when the buffer is full.
- The Consumer must not remove data when the buffer is empty.
- The Producer and Consumer should not insert and remove data simultaneously.

Solution to the Bounded Buffer Problem using Semaphores:

We will make use of three semaphores:

1. m (mutex), a binary semaphore which is used to acquire and release the lock.
2. empty, a counting semaphore whose initial value is the number of slots in the buffer, since, initially all slots are empty.
3. full, a counting semaphore whose initial value is 0.

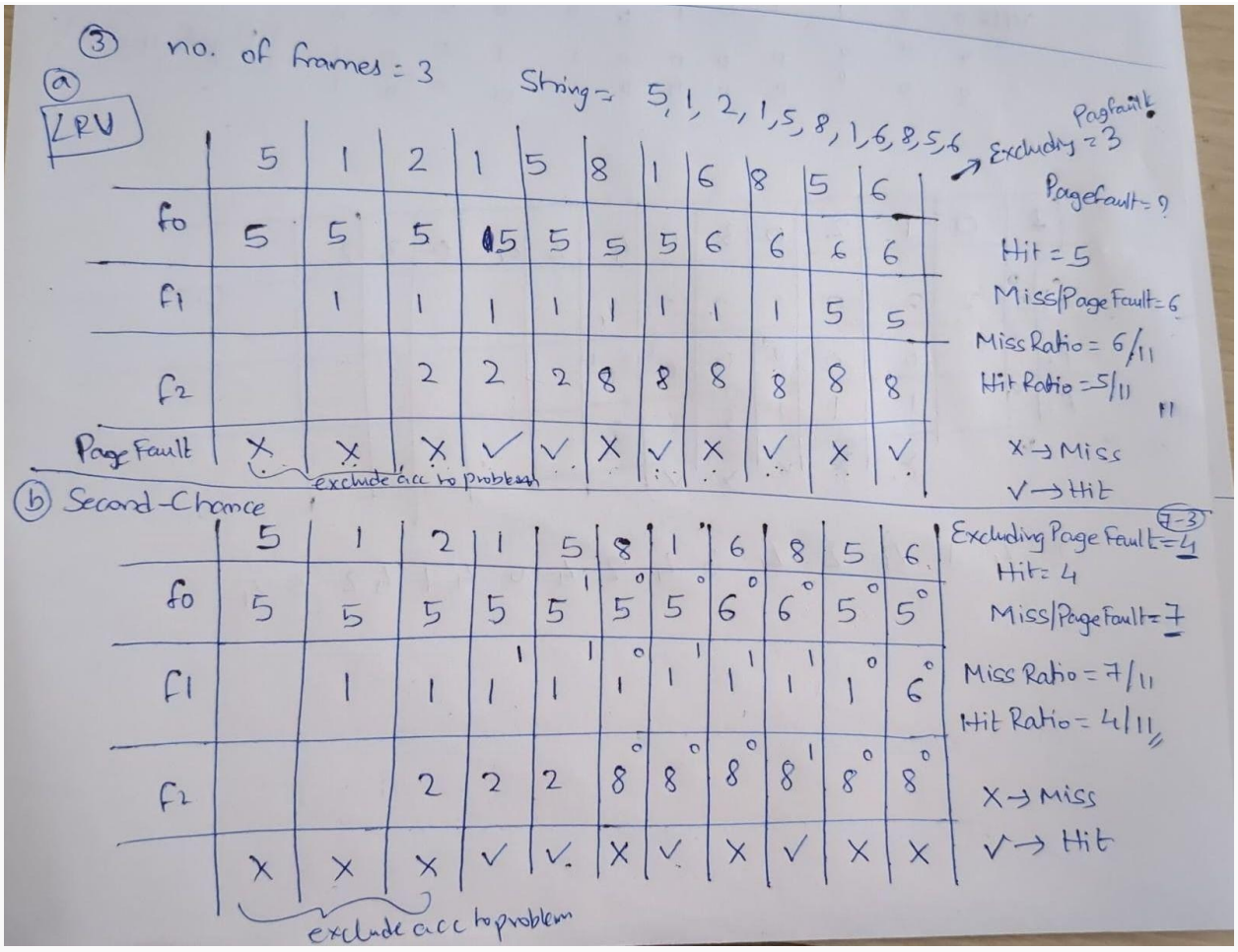




Producer Code Consumer Code

ooo

3) In the question they asked to exclude the first demand paging initial page faults so for LRU page faults is 3 and Second Chance page faults is 4.



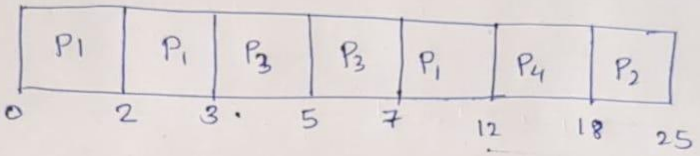
Fall 2016

1) Out of Syllabus
2)

② a) Preemptive Shortest Job First (SRTF):-

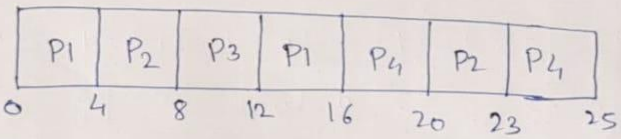
Process	(AT) Arrival Time	(BT) Expected CPU time	CT (Chart)	TAT (CT−AT)	WT (TAT−BT)	Resp Time
P1	0	8	12	12	4	0
P2	2	7	25	23	16	16
P3	3	4	7	4	0	2
P4	5	6	18	13	7	7
				52/4	27/4	

Gantt Chart:

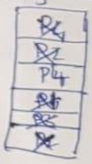


b) Round Robin with time quantum = 4

Gantt Chart:



Ready Queue



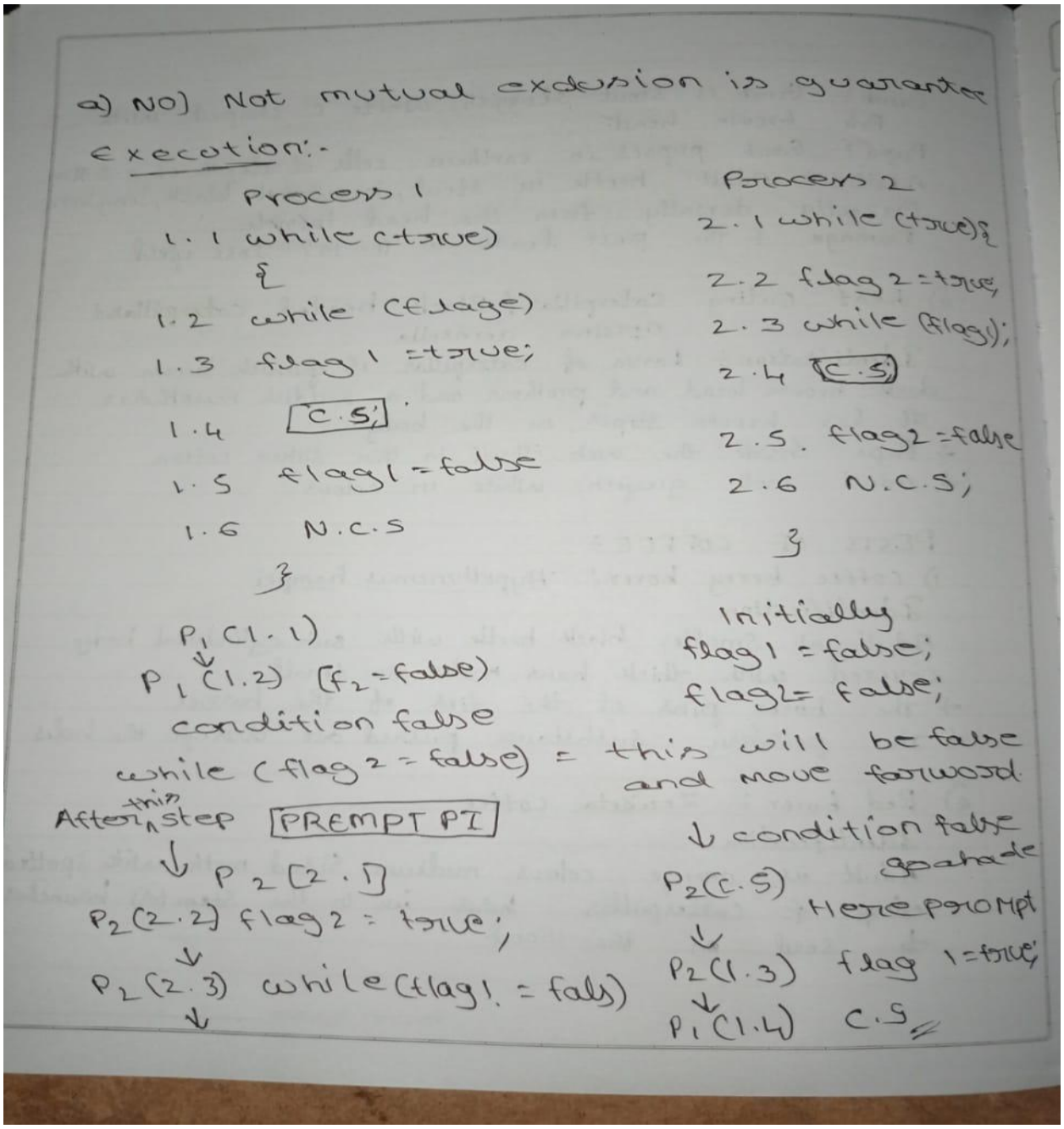
	CT	TAT	WT	Resp. Time
P1	16	16	8	0
P2	23	21	14	2
P3	12	9	5	5
P4	25	20	14	11

TAT = CT − AT

WT = TAT − BT

$$\text{Avg TAT} = \frac{\sum CT - AT}{\text{Total}} = \frac{16+21+9+20}{4} = \frac{66}{4} = 16.5 \text{ sec}$$

3) Doubt



a) NO) Not mutual exclusion is guarantee

Execution:-

Process 1

1.1 while (true)
{
1.2 while (flage)
1.3 flag1 = true;
1.4 C.S;
1.5 flag1 = false
1.6 N.C.S
}

Process 2

2.1 while (true){
2.2 flag2 = true;
2.3 while (flag1);
2.4 C.S;
2.5 flag2 = false
2.6 N.C.S;
}

P_1(1.1)
↓
P_1(1.2) [f_2 = false)
condition false
while (flag2 = false) = this will be false and move forword
After this step PREMPT PT
↓ condition false
↓ P_2[2.1]
P_2(2.2) flag2 = true;
↓
P_2(2.3) while(flag1 = fals)
↓

Initially
flag1 = false;
flag2 = false;

P_2(C.S) ↓ Here prompt
↓
P_2(1.3) flag 1 = true;
↓
P_1(1.4) C.S

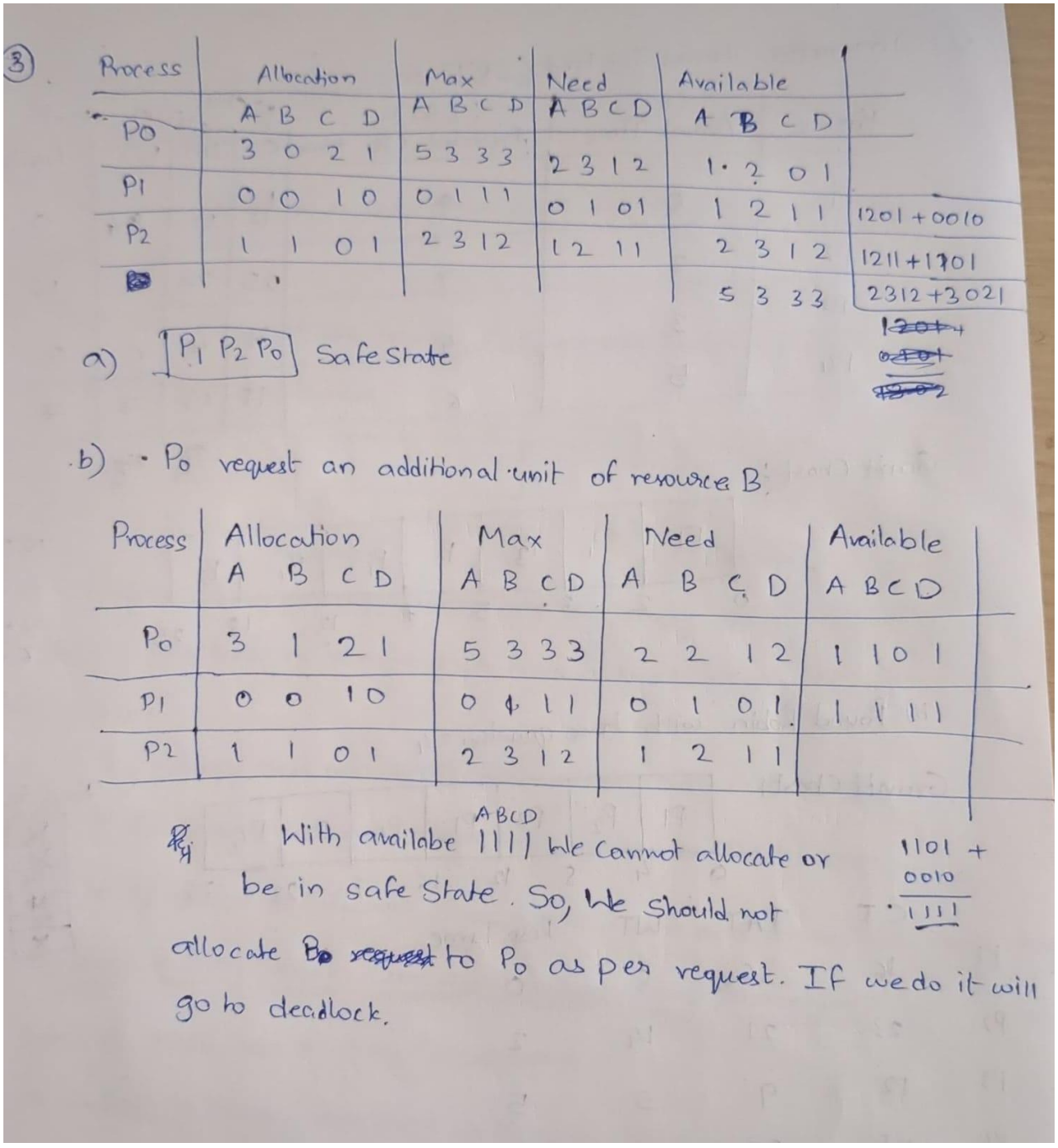
b) NO, deadlock cannot occur is the correct the above code the common variable flag1 and flag2 are being vidated in their respective process only.

means flag 1 is undated twice but only in process 1 similarly for flag2 also. which means if any one of process is ongoing it can't affect and deadlock other process and gets deadlock itself too.

c) Yes, indefinite postpone cannot occur in this code.

Winter 2017

1) Out of syllabus
2) Doubt Same Question as Fall 2016 Question 3
3) Bankers Algorithm:--



3)

Process	Allocation	Max	Need	Available
	A B C D	A B C D	A B C D	A B C D
P0	3 0 2 1	5 3 3 3	2 3 1 2	1 2 0 1
P1	0 0 1 0	0 1 1 1	0 1 0 1	1 2 1 1
P2	1 1 0 1	2 3 1 2	1 2 1 1	2 3 1 2
				5 3 3 3

1201 + 0010
1211 + 1101
2312 + 3021

a) $P_1\ P_2\ P_0$ Safe State

b) · P_0 request an additional unit of resource B

Process	Allocation	Max	Need	Available
	A B C D	A B C D	A B C D	A B C D
P0	3 1 2 1	5 3 3 3	2 2 1 2	1 1 0 1
P1	0 0 1 0	0 1 1 1	0 1 0 1	1 1 1 1
P2	1 1 0 1	2 3 1 2	1 2 1 1	

With available ABCD 1111 We cannot allocate or be in safe State. So, We should not allocate B to P_0 as per request. If we do it will go to deadlock.

1101 + 0010 = 1111

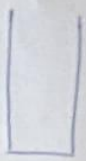
Spring 2017

1)Out of Syllabus

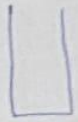
2)

② input string 2,6,5,7. to fill 4 frames.

a) Show that
LRU pagefault < FIFO Page Fault aba



Logical Memory 10 pages 0–9



physical Memory 4 frames (0–3)

LRU

	2	6	5	7	2	4	2
f1	2	2	2	2	2	2	2
f2		6	6	6	6	4	4
f3			5	5	5	5	5
f4				7	7	7	7
	×	×	×	×	✓	×	✓

Page Fault = 5
Hits = 2
X – Miss
✓ – Hit

FIFO

	2	6	5	7	2	4 (Random)	2
f1	2	2	2	2	2	4	4
f2		6	6	6	6	6	2
f3			5	5	5	5	5
f4				7	7	7	7
	×	×	×	×	✓	×	×

Page Fault = 6
Hit = 1
X – Miss
✓ – Hit

In this reference string of 2657242 the pagefaults of LRU < pagefaultof FIFO.

b) Show that Page Fault FIFO < LRU

LRU

	2	6	5	7	2	3 (Random)	6
f1	2	2	2	2	2	2	2
f2		6	6	6	6	3	3
f3			5	5	5	5	6
f4				7	7	7	7
	×	×	×	×	✓	×	×

PageFault = 6 Hit = 1

FIFO

	2	6	5	7	2	3	6
f1	2	2	2	2	2	3	3
f2		6	6	6	6	6	6
f3			5	5	5	5	5
f4				7	7	7	7
	×	×	×	×	✓	×	✓

Page Fault = 5 Hit = 2

In this Reference String of 2657236 the pagefault = 5, for FIFO but LRU pagefault = 6. So, FIFO < LRU pagefault.

3)

③ In this code:-

```
writer()                    Reader()
{                           {
  wait(wt);                   if(read count == 0) wait(wt);
// Do the writing            read count ++;
  Signal(wt);                 // Do the Reading
}                             read count --;
                              if(read count == 0) Signal(wt);
                            }
```

Sol Let consider this sequence where it leads to read and write access a a time.

* In the problem any number of reader can read a file but only one process can write at a time.

Let us assume

* P_0 process come to read a file

  First process to read a file So read count will be 0

  read count == 0 True So, wait(wrt) // initally 1 → 0. now wrt=0.

  increament readcount;

  Do Reading.

* Now P_1 process came to read a file

  readcount == 0 False as readcount value is 1.

  readcount ++; Do reading for P_1

* Now P_2 process came to read a file

  readcount == 0 False as readcount = 2.

  readcount ++; but suddenly an interrupt (prempt) occured and P_2 got prempted and now in I/O.

* Now P_1, P_0 are in reading and P_2 is in I/O. P_1 complete reading perform readcount -- ; So readcount=1

  P_0 Completed reading perform readCount--, So readcount=0.

  if readcount == 0 Signal(wrt) // wrt 0 → 1 wrt=1

If wrt=1 the P_0 want to write So

wait(wrt) => wrt 1→0 wrt=0

P_0 is writing.

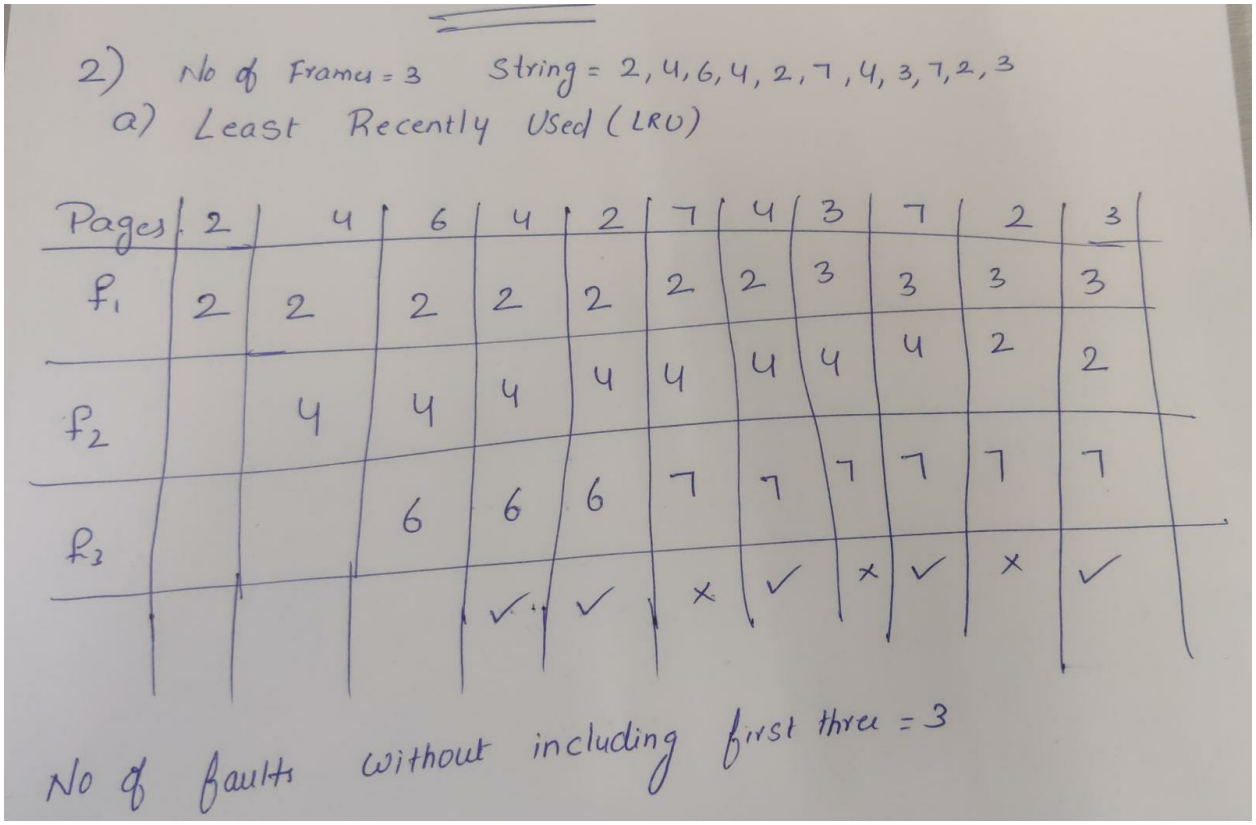
Now suddenly P_2 finishes its I/O and perform (write on memory readcount=1) and starts reading.

Now both P_0 and P_2 are performing write and read respectively,

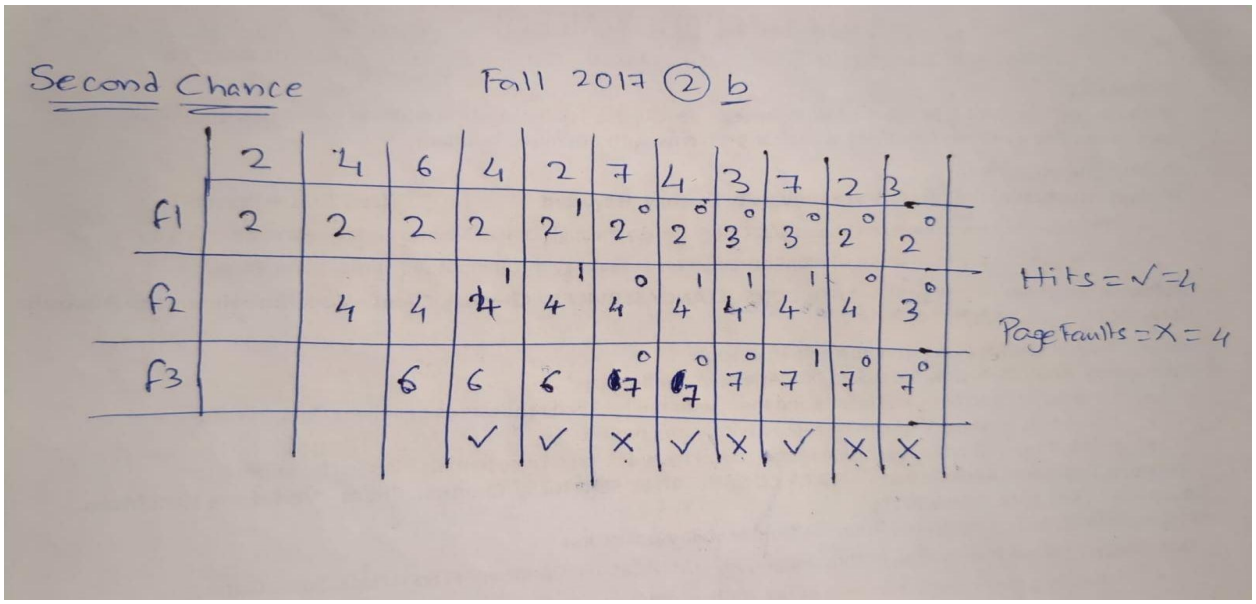
So both this way both reader and writer have access at the same time.

Fall 2017

1. Out of syllabus
2. a)



b)



3) Doubt We should write more of execution sequence

a)We should bring priority here.

If the outer process is more efficient, then the process which is running should wait until the outer process goes to running state and left

This is indefinite postponement. So indefinite postponement occurs here which is the one goal which wont satisfy the 2- process mutual exclusion problem.

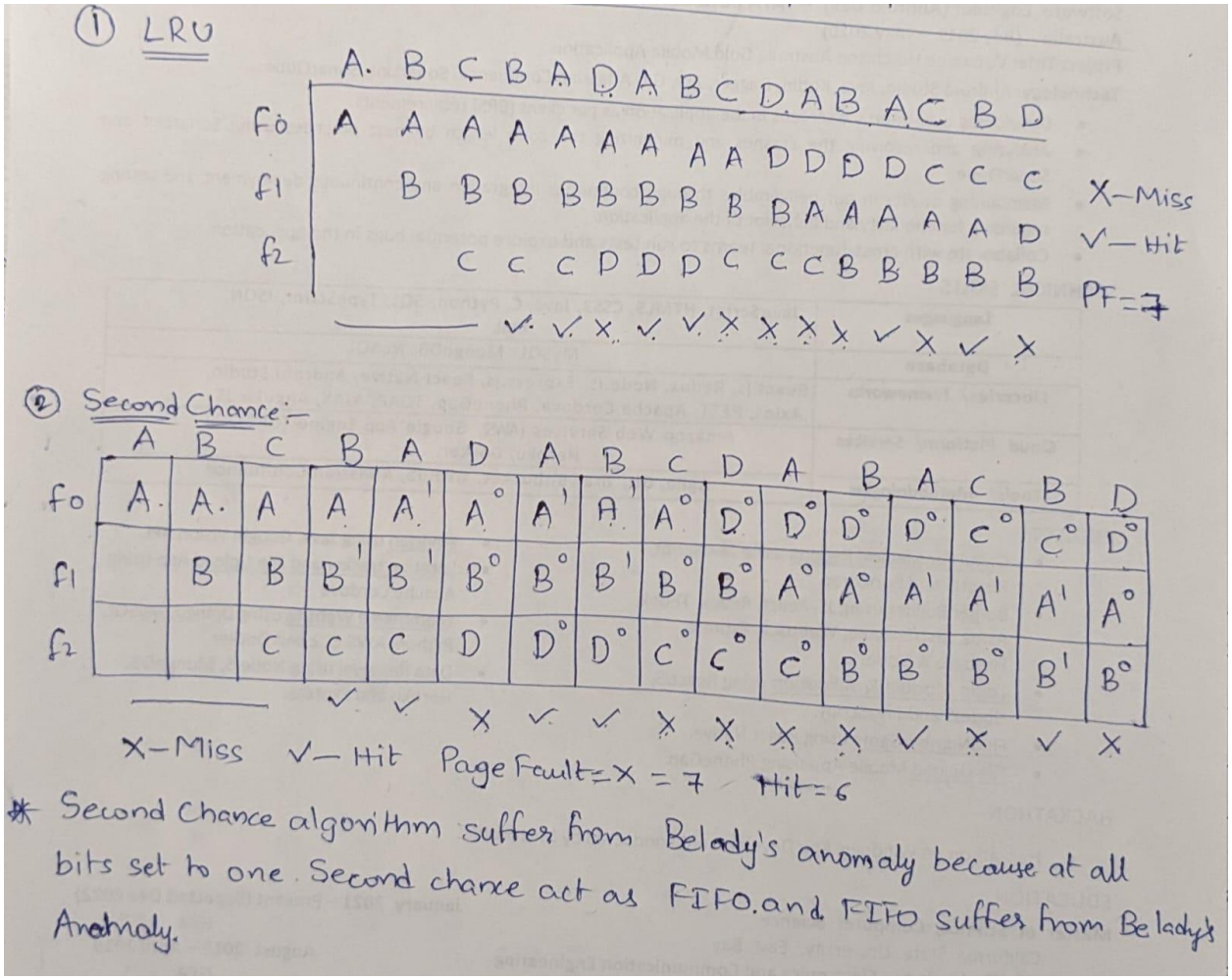
b) This Code gaurantee mutual Exclusion because one process is locked if it is running

Lock was set to false until one process completes its process

Another process will enter only if lock become true indirectly one process shoud leave it.

Spring 2018

1)

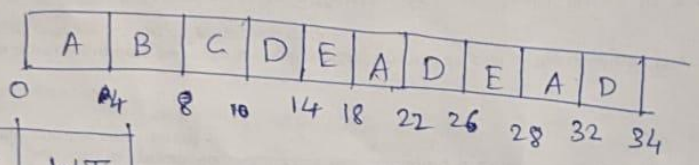


2)

Job	Arrival time	Time (msec)	Priority
A	0	12	1 (Gold)
B	2	4	3 (Bronze)
C	5	2	1 (Gold)
D	8	10	3 (Bronze)
E	10	6	2 (Silver)

a) Round Robin (RR) [quantum = 4]

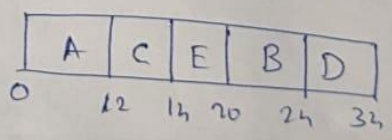
Gantt Chart



Jobs	CT	TAT	WT
A	32	32	20
B	8	6	2
C	10	5	3
D	34	26	16
E	28	18	12

b) Preemptive Priority :-

Gantt Chart



jobs	CT	TAT	WT
A	12	12	0
B	24	22	18
C	14	9	7
D	34	26	16
E	20	10	4

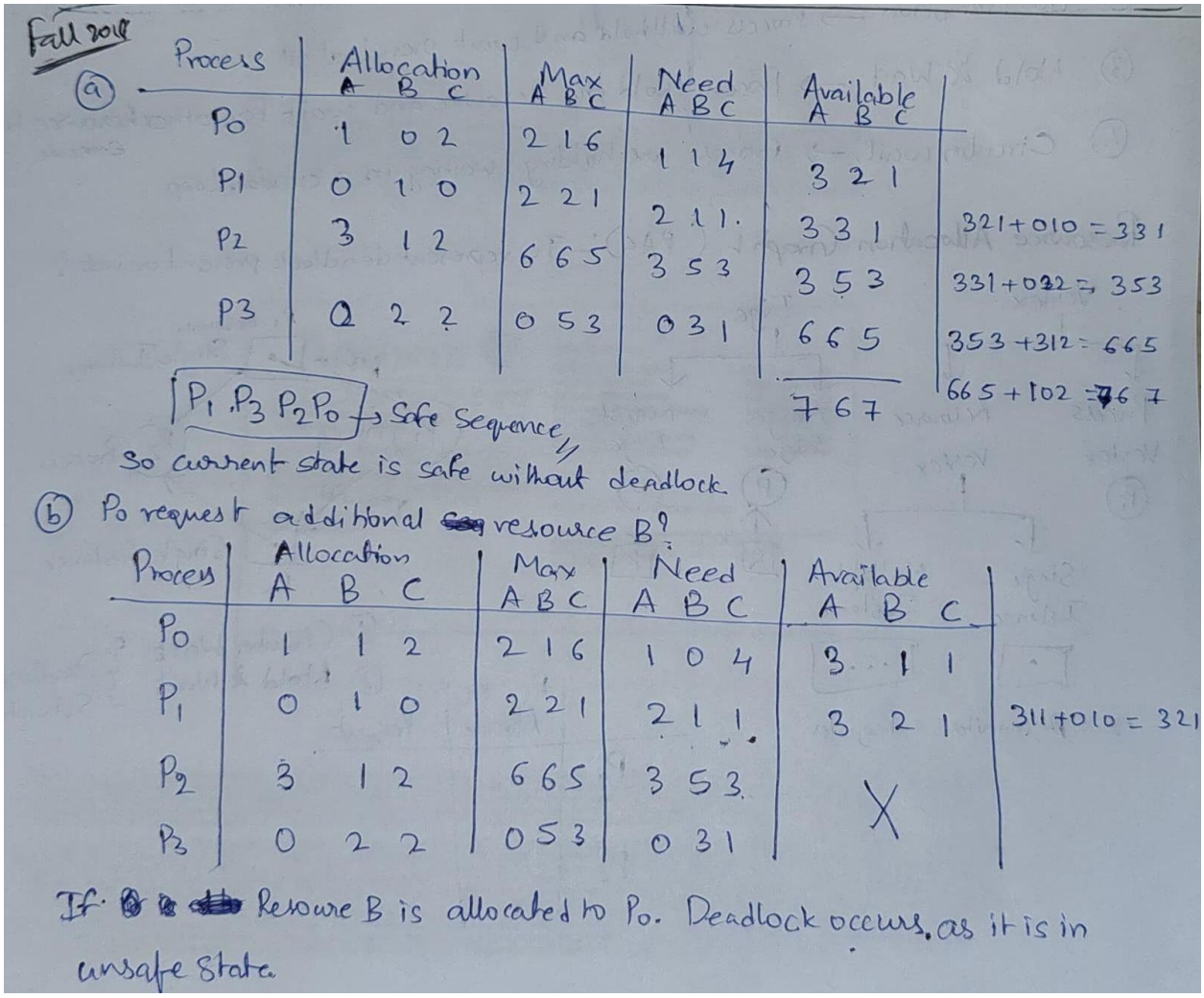
Average wait time = $\frac{0+18+7+16+4}{5}$

$= \frac{45}{5} = 9$

3) Out of syllabus

Fall 2018

1) Bankers Algorithm



Fall 2018

(a)

Process	Allocation A B C	Max A B C	Need A B C	Available A B C	
P_0	1 0 2	2 1 6	1 1 4	3 2 1	
P_1	0 1 0	2 2 1	2 1 1	3 3 1	321 + 010 = 331
P_2	3 1 2	6 6 5	3 5 3	3 5 3	331 + 022 = 353
P_3	0 2 2	0 5 3	0 3 1	6 6 5	353 + 312 = 665
				7 6 7	665 + 102 = 767

$P_1 P_3 P_2 P_0$ → Safe Sequence

So current state is safe without deadlock.

(b) P_0 request additional resource B?

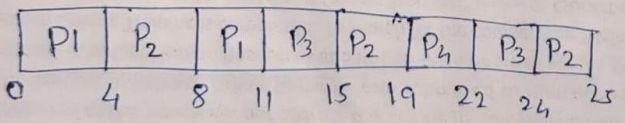
Process	Allocation A B C	Max A B C	Need A B C	Available A B C	
P_0	1 1 2	2 1 6	1 0 4	3 1 1	
P_1	0 1 0	2 2 1	2 1 1	3 2 1	311 + 010 = 321
P_2	3 1 2	6 6 5	3 5 3	X	
P_3	0 2 2	0 5 3	0 3 1		

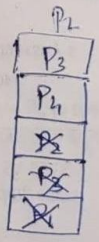
If Resource B is allocated to P_0. Deadlock occurs, as it is in unsafe state.

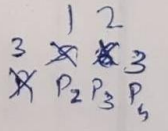
2)

Process	Arrival Time	Burst Time	CT	TAT	WT	RT
P1	0	7	11	11	4	0
P2	3	9	25	22	13	1
P3	5	6	24	19	13	6
P4	9	3	22	13	10	10

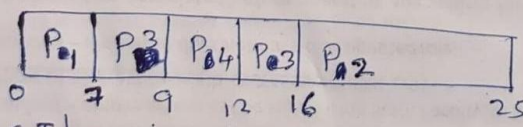
a) Round Robin – Time quantum = 4







b) SRTF:–



Process	AT	BT	CT	TAT	WT	RT
P1	0	7	7	7	0	0
P2	3	9	25	22	13	13
P3	5	6	16	11	5	2
P4	9	3	12	3	0	0

Avg. Turnaround Time = $\frac{7+22+11+3}{4}$

$= 43/4 = 10.75$

3) Doubt

Still needs explanation for this

Fall 2018 :-

a) ME is not possible

b) DL doesnot occur

c) IP is possible