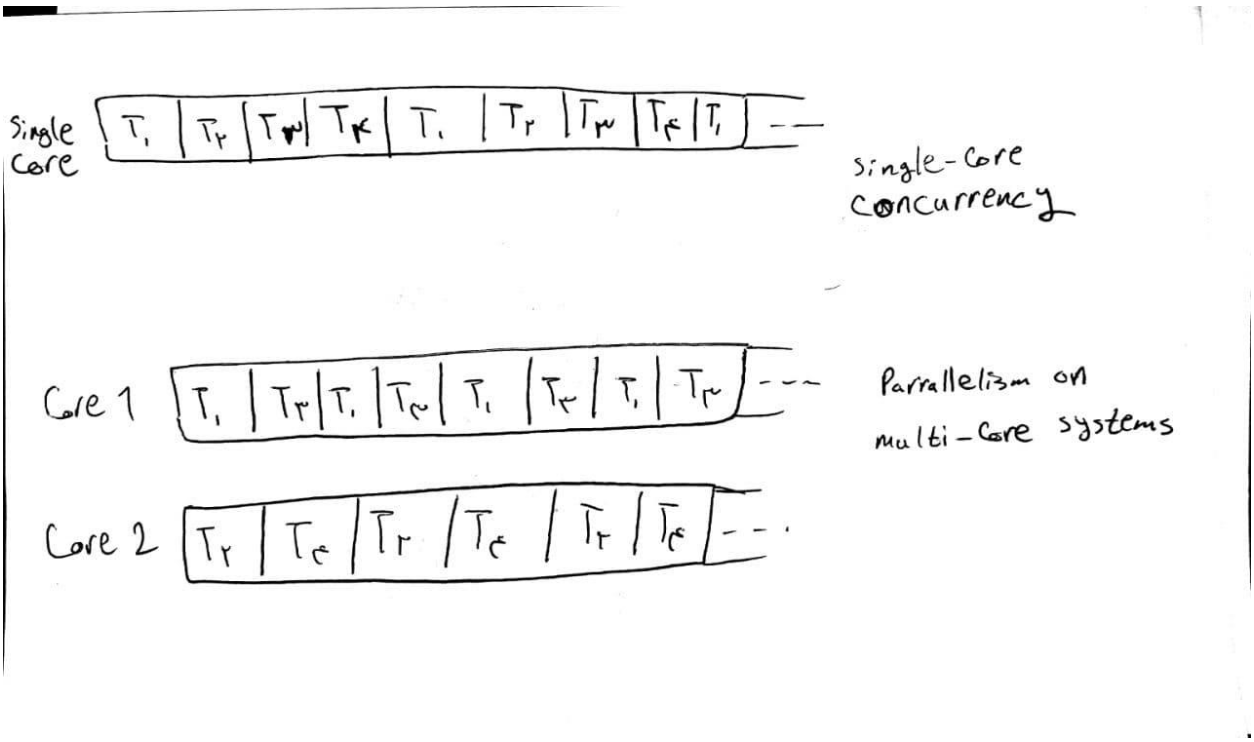
Sina Ziaee 97521387

Question 1)

- Thread creation is much cheaper than process creation and they are more light-weighted than processes
- Thread switching has much lower overhead on OS than process switching
- Unlike processes, if our process is blocked they can continue execution which is very important for user interfaces and real time operations
- Thread communication is much easier and cheaper especially if all of them belong to a specific process (as they share the same memory of the their parent process) (but they are also more vulnerable to other threads damages)

Question 2)



Concurrency means that an application is making progress on more than one task at the same time(concurrently). If we have a single core single processer CPU then not more than one task is not processed exactly at the same time but more than one task is being processed at a time inside a core , which means it does not completely finish one task before it begins the next one. (tasks are divided into separate parts by time and each part is processed in different times, in overall a user from an outside perspective thinks that all these tasks are processed at the same time while the time is actually shared and the actions are finished really fast.)

Concurrency means executing multiple tasks at the same time but not necessarily simultaneously.

Parallelism means spilting tasks up into smaller subtasks and which can be process in parallel for instance multiple CPUs at the exact same time.

Parallelism requires hardware with multiple processing units, essentially. In single-core CPU, you may get concurrency but NOT parallelism. Parallelism is a specific kind of concurrency where tasks are really executed simultaneously.

A system is said to be concurrent *if it can support two or more actions* in progress *at the same time. A system is said to be* parallel *if it can support two or more actions executing simultaneously.*

The key concept and difference between these definitions is the phrase *in progress.*
An application can be parallel — but not concurrent, which means that it processes multiple sub-tasks of a task in multi-core CPU at the same time.

IF WE HAVE SIMULTONUSLY PROCESSING THAN WE HAVE PARALLEL EXECUTION

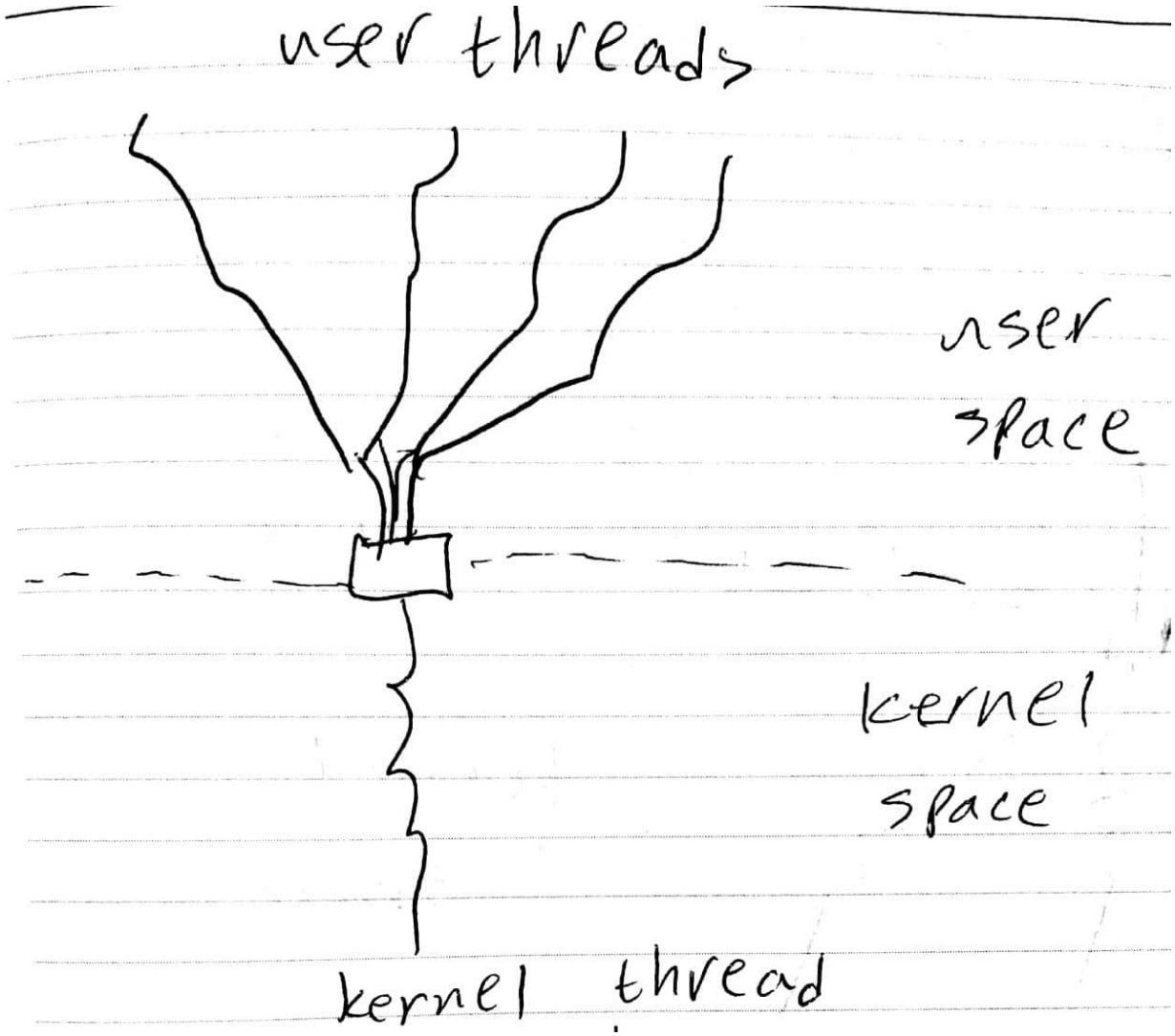
Question 3)

When we say we have parallel data processing , it means we divide our data into multiple parts and distribute it across different cores to be operated in the same way simultaneously.

When we have parallel task processing, it means we have different threads distributed along different cores and perform unique operations simultaneously.

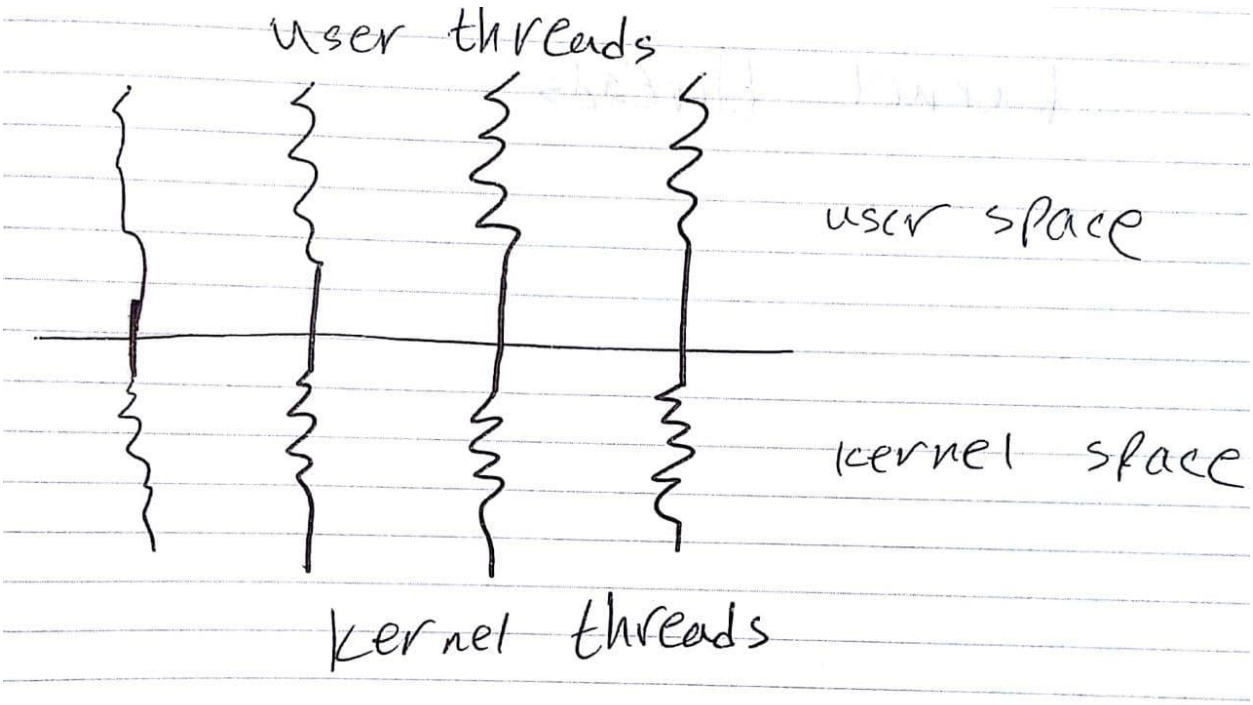
Question 4)

Many to one (not recommended)



Different user level threads are handled by one kernel thread, the implementation is much simpler but if one user thread is blocked then kernel thread is block which means all other user threads are also blocked , which is not a good thing.

One to One (implemented in most famous operating systems, which means is common)

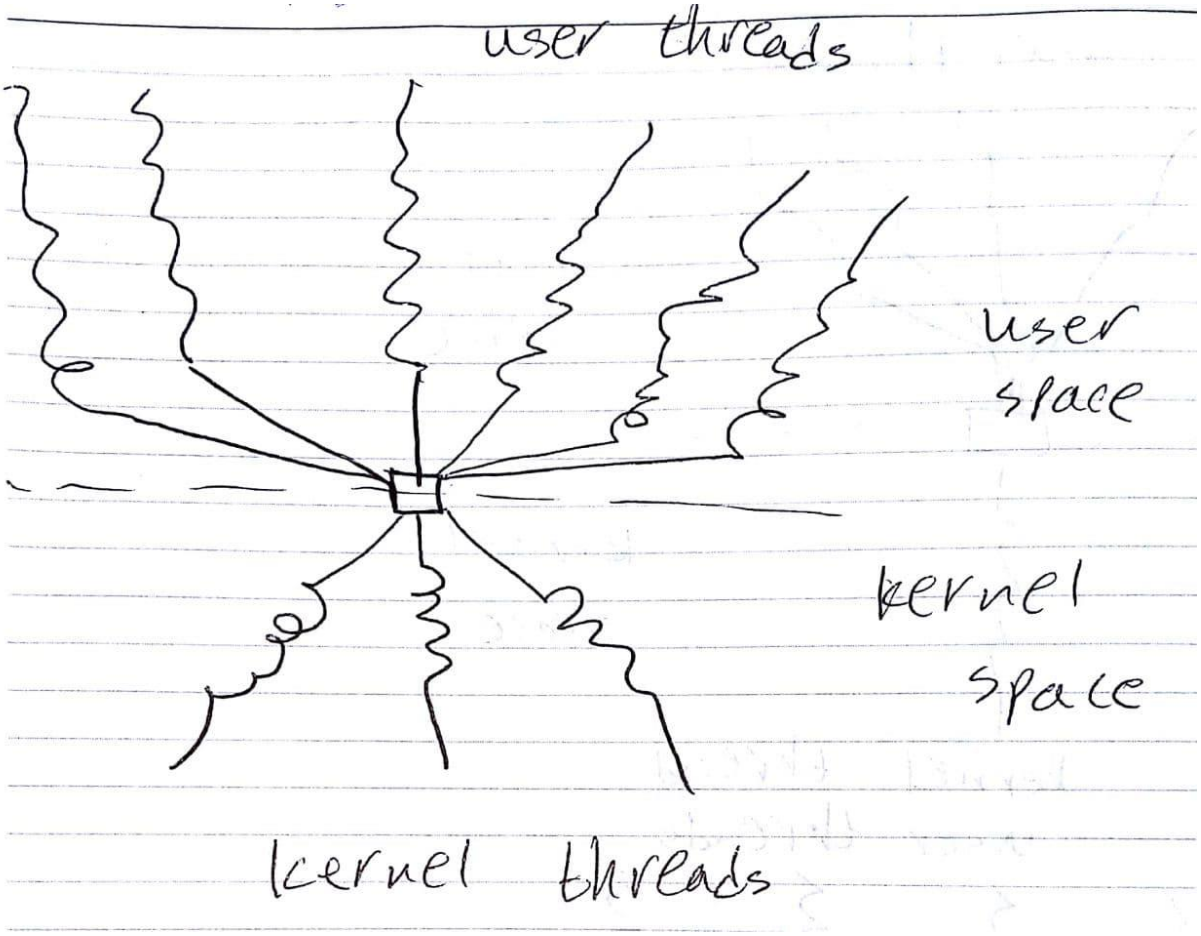


Each user level thread is handled by exactly one kernel level thread which is more complex and harder to implement rather then many to one, but also we do not have the problem of if one user thread is blocked then all the other threads are blocked which is a very good and optimal thing.

We have more concurrency compared to “many to one”.

Some times the number of threads is limited due to system overhead which means we may encounter some delays in user space as sometimes we can not provide required threads.

Many to Many (Not very common)



Many user level thread are mapped to many kernel level threads but does not mean each specific user thread is mapped to exactly one kernel level thread . it allows the operating system to create enough threads to handle user level threads.

Question 5)

As Amdahl's law declares :

Serial portion of an application has disproportionate effect on performance gained by adding additional cores

Which means if some of our tasks are process serially and are dependent of previous tasks then we can not process them simultaneously which means by using more cores we can not speed up the operations in these serial parts so if we want to speed up things we must have tasks are not dependent serially to each other so we can operate them simultaneously and on different cores.

So as the applications are comprised in both parallel and serial operations, the Amdahl's law is this:

$$\text{Speedup} \leq \frac{1}{S + \frac{(1-S)}{N}} = \frac{N}{S(N-1)+1}$$

$S \rightarrow$ serial portion of tasks

As N grows infinitly, SpeedUp grows $\frac{1}{S}$

$$N=2 \rightarrow \text{seedUp} \leq \frac{1}{S + \frac{(1-S)}{2}} = \frac{2}{S+1}$$

$$N=4 \rightarrow \text{speedUp} \leq \frac{1}{S + \frac{1-S}{4}} = \frac{4}{3S+1}$$

$$N=8 \rightarrow \text{speedUp} \leq \frac{1}{S + \frac{1-S}{8}} = \frac{8}{7S+1}$$

$$N=100 \rightarrow \text{SpeedUp} \leq \frac{1}{S + \frac{1-S}{100}} = \frac{100}{99S+1}$$

Question 6)

CPU scheduling is the process of determining which process will own the CPU for execution while another CPU is on hold. The main task of CPU scheduling is to make sure whenever the CPU remains idle, the OS at least selects one of the processes available in the ready queue for execution, So optimizing CPU scheduling is maximizing the number of tasks that can be processed by CPU and make the CPU as much as busy as possible (mostly around 80 % of the CPU).

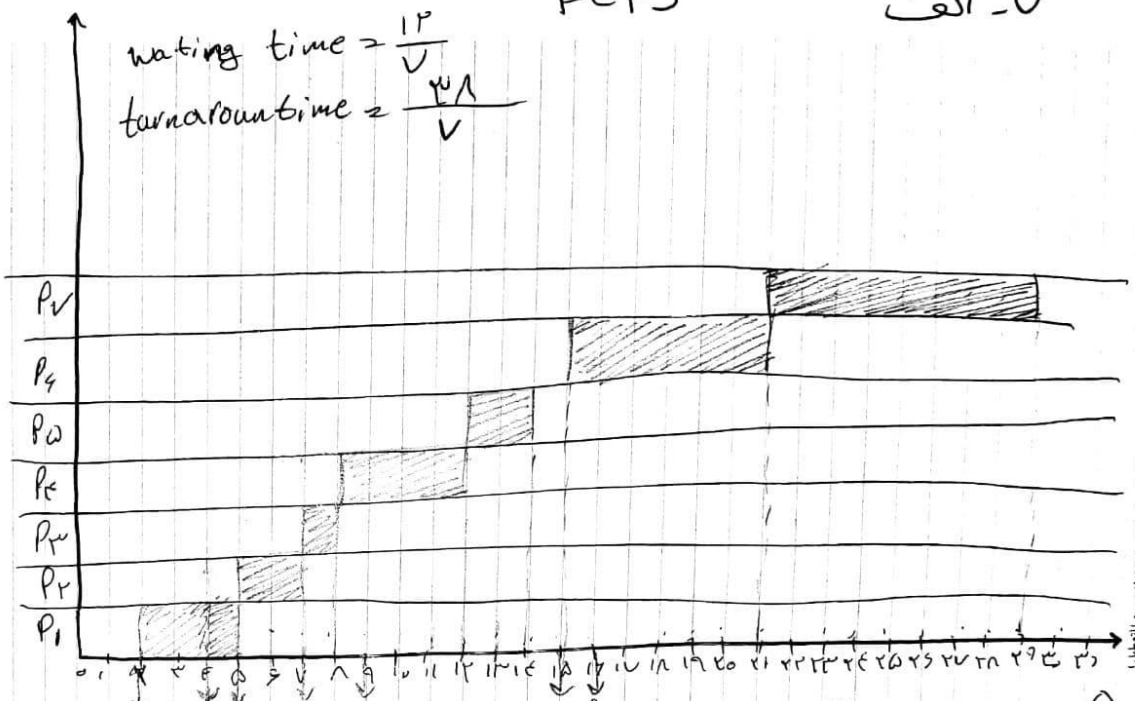
We have a number of useful criteria for optimizing CPU scheduling as listed below:

- CPU utilization: The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible. Theoretically, CPU utilization can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the load upon the system (Maximize this)
- Throughput: A measure of the work done by CPU is the number of processes being executed and completed per unit time. This is called throughput. The throughput may vary depending upon the length or duration of processes (Maximize this)
- Turnaround Time: For a particular process, an important criteria is how long it takes to execute that process. The time elapsed from the time of submission of a process to the time of completion is known as the turnaround time. Turn-around time is the sum of times spent waiting to get into memory, waiting in ready queue, executing in CPU, and waiting for I/O. (Minimize this)
- Waiting time: A scheduling algorithm does not affect the time required to complete the process once it starts execution. It only affects the waiting time of a process i.e. time spent by a process waiting in the ready queue. (Minimize this)
- Response time: In an interactive system, turn-around time is not the best criteria. A process may produce some output fairly early and continue computing new results while previous results are being output to the user. Thus another criteria is the time taken from submission of the process of request until the first response is produced. This measure is called response time. (Minimize this)

Question 7)

۵- الف

FCFS



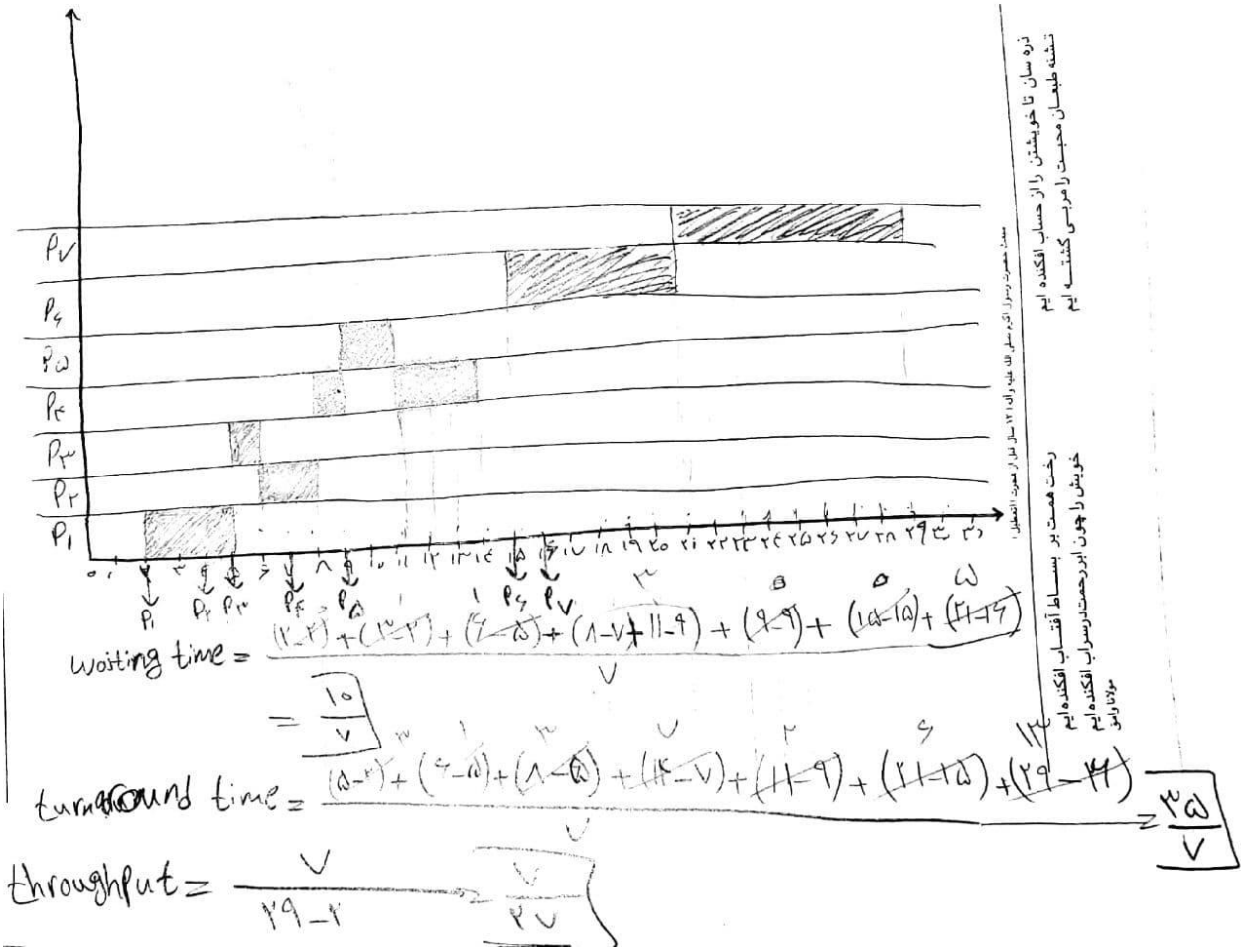
$$\text{waiting time} = \frac{(۴-۲)+(۵-۴)+(۷-۵)+(۱۱-۷)+(۱۳-۹)+(۱۵-۱۴)+(۲۱-۱۶)}{۷} = \frac{۱۲}{۷}$$

$$\text{turnaround time} = \frac{(۴-۲)+(۷-۳)+(۱۱-۵)+(۱۳-۷)+(۱۶-۹)+(۲۱-۱۵)+(۲۹-۱۶)}{۷} = \frac{۳۸}{۷}$$

$$\text{Throughput} = \frac{\text{Number\_of\_Processed\_done}}{\text{Max(Completiontime)} - \text{Min(Arrivaltime)}} = \frac{۷}{۲۹-۲} = \frac{۷}{۲۷}$$

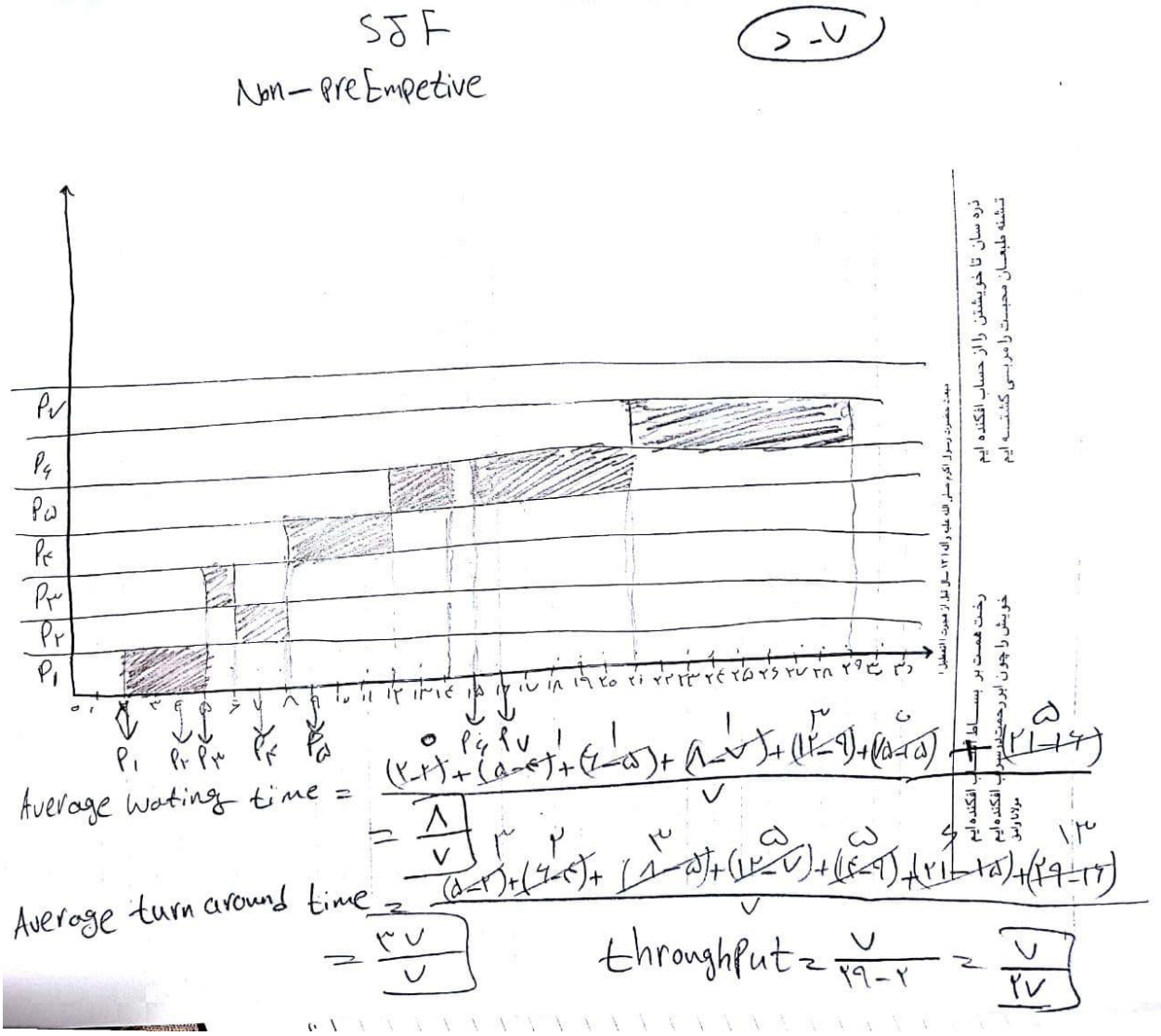
SRJF

ب-۷



7-c) According to our average waiting time and average turn around time in FCFS and SRJF :

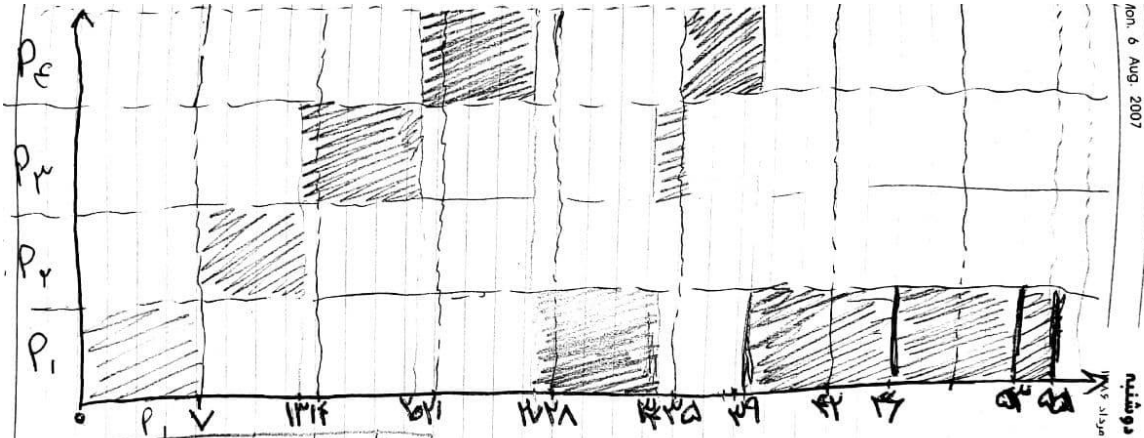
In this particular set of tasks , SRJF has better results compared to FCFS. Also as we saw SRJF was a preEmpetive and FCFS was a non-PreEmpetive algorithm.



7-e)

By comparing 7-b and 7-d results we can find out that in this particular example with this set of tasks, Shortest Remaining Job First which is a PreEmpetive algorithm is more optimal than Shortest Job First which is a Non-PreEmpetive algorithm.

Question 8)



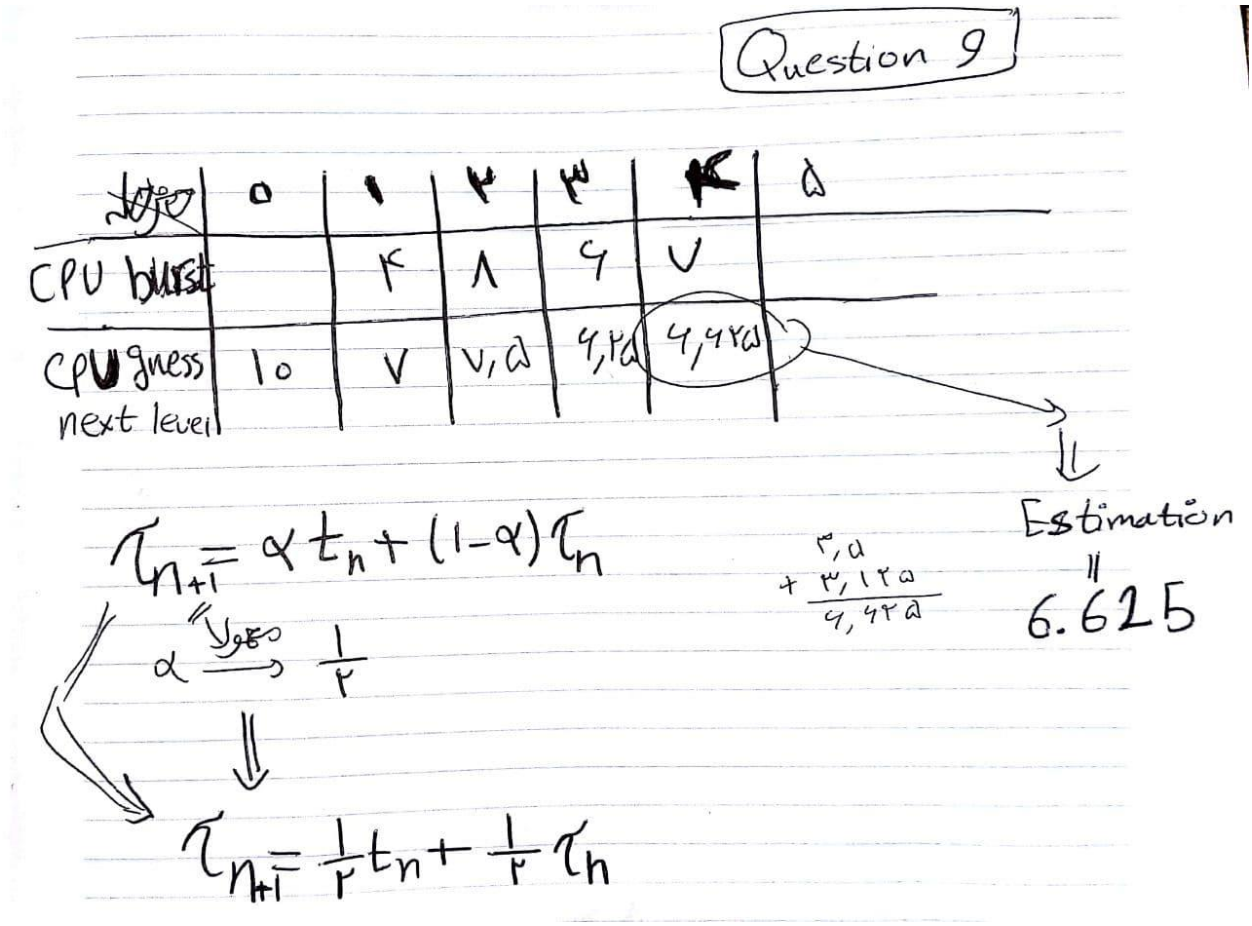
Process	P_1	P_2	P_3	P_4
End time	۵۵	۱۳	۳۵	۴۹
Turn around Time	۵۵	۱۳	۳۵	۴۹
Wating Time	۲۰+۵ ۲۵	۷	۱۳+۶ ۲۲	۲۰+۲۸ ۲۸

$$\text{Average wating time} = \frac{۲۵+۷+۲۲+۲۸}{۴} = ۲۱,۷۵$$

$$\text{Average TurnAround time} = \frac{۵۵+۱۳+۳۵+۴۹}{۴} = ۳۵,۵$$

$$\text{ThroughPut} = \frac{۴}{۵۵-۰} = \frac{۲}{۵۵}$$

Question 9)



Question 10)

a) 80% of CPU bursts should be shorter than time quantum
b) First come first server (FCFS or FIFO)
c) q must be large with respect to context switch, otherwise overhead is too high
d) No we can't

Question 11)

a) This algorithm is called priority scheduling which may face a problem called STARVATION that says low priority processes may never execute.
b) One of the solutions for starvation problem is AGING which says as the time of the progresses increases , increase the priority of these that are assigned CPU before, so in the near future they may get a CPU burst time.

Question 12)

Prioritization based on process type in Multi-level Queue scheduling can be done with four types of queues base on the type of process

1- Real time Processes
2- System Processes
3- Interactive Processes
4- Batch Processes

All these above types are showed by the priority from top to bottom respectively

Question 13)

When a thread has been running over one processor, the cache contents of that processor stores the memory accesses by that thread. We refer to this as a thread having affinity for a processor. Or in other words

PROCESSOR AFFINITY SPECIFIES WHICH PROCESSORS A GIVEN PROCESS OR THREAD SHOULD RUN ON.

We have two types of processor affinities :

Soft affinity: The processor attempts to keep a thread running on the same processor, but no guarantees.

Hard affinity: Allows a process to specify a set of processors it may run on.