OS Assignment II

1.

A. **Provide two programming examples of multithreading giving improved performance over a single-threaded solution.**

The process of executing multiple threads simultaneously is known as multithreading.

Some examples where multi threading improves performance include:

Matrix multiplication — Individual rows and columns of the matrices can be multiplied in separate threads, reducing the wait time of the processor for addition.

UI updates — We can render some UI elements such as a view or images on a background thread so that the main thread does not block the view, causing performance issues and noticeable lags.

B. **Provide two programming examples of multithreading that would not improve performance over a single-threaded solution.**

If we are running a trivial program (constant time complexity) in a separate thread, the overhead of creating the threads exceeds the tasks performed by them, thus decreasing performance when compared to a single threaded alternative.

Some examples include:

Trivial operations on a list of numbers — Multi threading won't speed up the operations since the time taken by the operations is constant, and other elements of the list may or may not wait for the previous to finish.

Allocating memory to a set of data variables — Allocating memory is a very fast task, and the overhead of creating multiple threads to process separate blocks of variables exceeds the performance gained by multi threading.

2.

A. **What resources are used when a thread is created? How do they differ from those used when a process is created?**

When a thread is created, the threads does not require any new resources to execute the thread shares the resources like memory of the process to which they belong to. The benefit of code sharing is that it allows an application to have several different threads of activity all within the same address space.

Whereas a new process creation is very heavyweight because it always requires new address space to be created and even if they share the memory then the inter process communication is expensive when compared to the communication between the threads.

B. **Consider a multiprocessor system and a multithreaded program written using the many-to-many threading model. Let the number of user-level threads in the program be more than the number of processors in the system. Discuss the performance implications of the following scenarios:**

a. **The number of kernel threads allocated to the program is less than the number of processors.**

The scheduler can only schedule user level processes to the kernel threads, and since some of the processes are not mapped to the kernel threads, they will be idle.

b. **The number of kernel threads allocated to the program is greater than the number of processors but less than the number of user-level threads.**

All of the processes will be working simultaneously assuming there are enough user threads. If a kernel thread is blocked, it may be swapped out for one that isn't blocked.

3.

A. **Five jobs are waiting to be run. Their expected run times are 9, 6, 3, 5, and X. In what order should they be run to minimize average response time (of course your answer will depend on X)**

Shortest job first is a way to minimize the response time:

case $(0 < X \leq 3)$ — X, 3, 5, 6, 9
case $(3 < X \leq 5)$ — 3, X, 5, 6, 9
case $(5 < X \leq 6)$ — 3, 5, X, 6, 9
case $(6 < X \leq 9)$ — 3, 5, 6, X, 9
case $(X > 9)$ — 3, 5, 6, 9, X

B. **The exponential average formula with $\alpha = 1/2$ is being used to predict run times. The previous four runs, from oldest to most recent, are 40, 20, 40, and 15 ms. What is the prediction of the next time? (assume the initial guess is 40 ms)**

If we take all four runs in consideration (first 40); the prediction is =

$\frac{1}{2}(15 + \frac{1}{2}(40 + \frac{1}{2}(20 + \frac{1}{2}(40)))) = 22.5$

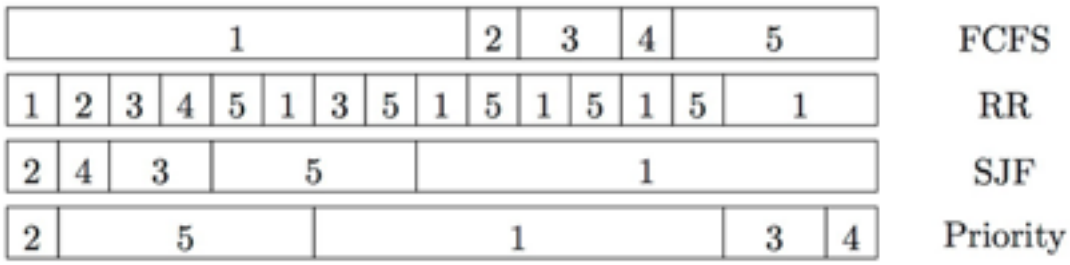
If we take the last two (last 40) into consideration; the prediction is =

$\frac{1}{2}(15 + \frac{1}{2}(40)) = 17.2$

4. Consider the following set of processes, with length of CPU burst given in ms.

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

The processes are assumed to have arrived in order P_1, P_2, P_3, P_4, and P_5 all at time 0. Which of the following algorithms: FCFS, SJF, nonpreemptive priority, and RR (quantum = 1) results in minimum average waiting time.



Turnaround time

	FCFS	RR	SJF	Priority
P_1	10	19	19	16
P_2	11	2	1	1
P_3	13	7	4	18
P_4	14	4	2	19
P_5	19	14	9	6

Waiting time(turnaround time minus burst time)

	FCFS	RR	SJF	Priority
P_1	0	9	9	6
P_2	10	1	0	0
P_3	11	5	2	16
P_4	13	3	1	18
P_5	14	9	4	1

Average waiting time in FCFS = 9.6ms
Average waiting time in SJF = 3.2ms
Average waiting time in Priority = 8.2ms
Average waiting time in RR = 5.4ms

Shortest Job First (SJF) results in minimum average waiting time.

5. Suppose that the following processes arrive for execution at the times indicated. Each process will run the listed amount of time. In answering the questions, use nonpreemptive scheduling and base all decisions on the information you have at the time the decision must be made.

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

a. What is the average turnaround time for these processes with the FCFS scheduling algorithm?

P_1	P_2	P_3

Average turnaround time = ((8 - 0) + (12 - 0.4) + (13 - 1.0)) / 3 = 10.53

b. What is the average turnaround time for these processes with the SJF scheduling algorithm?

P_1	P_3	P_2

Average turnaround time = ((8 - 0) + (13 - 0.4) + (9 - 1.0)) / 3 = 9.53