Homework 2:

4.8)

The components of program state that are shared across threads are Heap Memory and Global Variables. Each thread has its own set of CPU registers and Private Stack (local variables, function calls, and return addresses).

4.9)

A multithreaded solution using only user-level threads does not perform better on a multiprocessor system because the OS only schedules the entire process on one CPU. To utilize multiple processors more efficiently, the system must support kernel-level threads. User-level threads are scheduled and managed entirely in user space without the kernel's involvement and the operating system is unaware of the existence of these threads.

4.7)

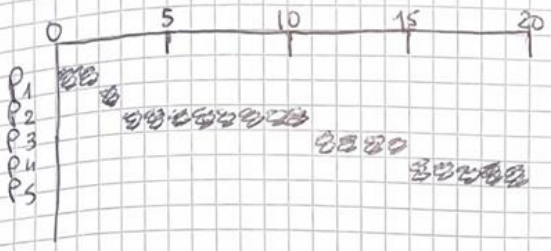
One circumstance can be when a thread makes a block system call. In a single threaded process if this happens then the entire process is stalled. Vice versa, in a multithreaded process with kernel threads, the OS can schedule another thread from the same process to execute. Or also when a thread is waiting for I/O, other threads could execute. But generally multithreaded can be better because of responsiveness, resource sharing, economy, and scalability. Responsiveness because let's say a user wants to click on a button in a single threaded application, he would not be able to do so until the current operation is not completed. In contrast, if the time-consuming operation is performed in a separate thread, the application remains responsive. Resource sharing because thread share the memory and the resources of the process to which they belong by default. Scalability because many benefits of multithreading can be even greater in a multiprocessor architecture, where threads may be running in parallel on different processing cores.

6.16)

6.16 a)

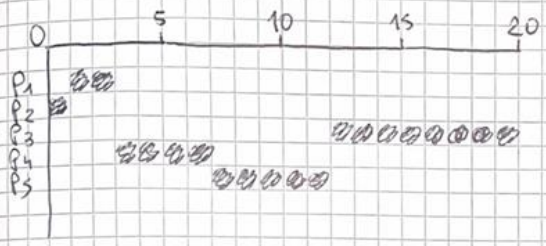
FCFS:

P_1	P_2	P_3	P_4	P_5
0-2	2-3	3-11	11-15	15-20



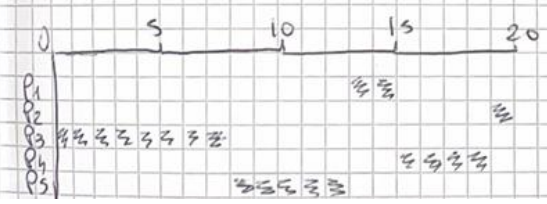
SJF:

P_2	P_1	P_4	P_5	P_3
0-1	1-3	3-7	7-12	12-20



- NON-PREEMPTIVE PRIORITY SCHEDULING

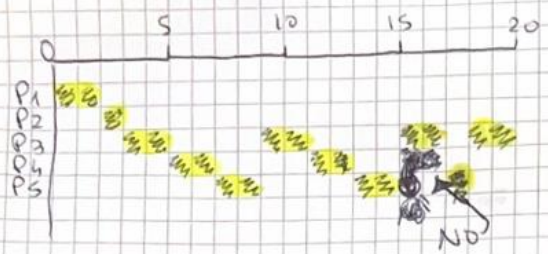
larger priority number equals higher priority

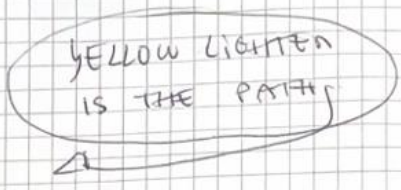


P_3	P_5	P_1	P_4	P_2
0-8	8-13	13-15	15-19	19-20

ROUND ROBIN

quantum = 2





P1 AND P2 FINISH AT FIRST EXECUTION, THEN:

P_1	2
P_2	1
P_3	8
P_4	4
P_5	5

$P_5(5)$ | $P_4(4)$ | $P_3(8)$

$P_3(6)$ | $P_5(5)$ | $P_4(4)$

$P_4(2)$ | $P_3(6)$ | $P_5(5)$

$P_5(3)$ | $P_4(2)$ | $P_3(6)$

$P_3(4)$ | $P_5(3)$ | $P_4(2)$

$P_3(4)$ | $P_5(3)$

$P_5(1)$ | $P_3(4)$

$P_3(2)$ | $P_5(1)$

$P_3(2)$

P_1	P_2	P_3	P_4	P_5	P_3	P_4	P_5	P_3	P_5	P_3
0-2	2-3	3-5	5-7	7-9	9-11	11-13	13-15	15-17	17-18	18-20

(b)

TURNAROUND TIME: COMPLETION TIME − ARRIVAL TIME (IS 0 FOR ALL PROCESSES)

FCFS :

P_1	P_2	P_3	P_4	P_5
2	3	11	15	20

SJF :

P_2	P_1	P_4	P_5	P_3
1	3	7	12	20

NPPS (PRIORITY) :

P_3	P_5	P_1	P_4	P_2
8	13	15	19	20

RR :

P_1	P_2	P_4	P_5	P_3
2	3	13	18	20

(c)

waiting time = turnaround time − burst time

FCFS

P_1	P_2	P_3	P_4	P_5
0	2	3	11	15

SJF

P_2	P_1	P_4	P_5	P_3
0	1	3	7	12

NPPS (PRIORITY)

P_3	P_5	P_1	P_4	P_2
0	8	13	15	19

RR

P_1	P_2	P_4	P_5	P_3
0	2	9	13	12

(d)

Minimum average waiting time:

FCFS = (0+2+3+11+15)/5 = 6.2 ms

SJF = (0+1+3+7+12)/5 = 4.6 ms

NPPS (PRIORITY) = (0+8+13+15+19)/5 = 11.0 ms

RR = (0+2+9+13+12)/5 = 7.2 ms

SJF has the minimum average waiting time

6.19)

SJF and Priority scheduling can result in starvation because in SJF, processes with long burst time may never get scheduled if shorter jobs keep arriving, pushing them back. In Priority scheduling, lower-priority processes may be indefinitely delayed if higher-priority processes keep arriving, preventing them from execution.