**Homework Assignment #2**

***Due Date: 2/28, 11:59 p.m., please submit a soft copy via Blackboard. Late submissions are accepted till 3/2, 11:59 p.m., with 10% penalty each day.***

***Please name your submission file starting with “LastName\_FirstName\_HW2”.***

**Q1.** What is a process in an operating system? What is a thread in an operating system?

The process is an instance of program execution which associated with resources. On the other hand, the thread is the minimal unit that the OS allocates, which means one process can have many threads, and each thread can fetch and execute instructions independently.

The different process has different resources and different executions, where different thread in the same process shares the same resources and different executions.

**Q2. Modeling multiprogramming:** (a) Assume the I/O fraction time of all processes is 20%, and assume processes are independent from each other, what’s the CPU utilization, if the number of processes, *n* = 1, 2, 4, and 8, respectively? (b) If the I/O fraction time is 50% for all processes, what’s the CPU utilization again, if the number of processes, *n* = 1, 2, 4, and 8, respectively?

1. Based on 1-p^n

When n = 1, result is 0.8

n = 2, result is 0.96

n = 4, result is 0.9984

n = 8, result is 0.99999774

1. If p = 0.5

n = 1, result is 0.5

n = 2, result is 0.75

n = 4, result is 0.975

n = 8, result is 0.99609375

**Q3.** Consider a system that has two CPUs and each CPU has two hardware threads (hyperthreading). Suppose three processes, P0, Pl, and P2, are started with run times of 5, 10 and 20 msec, respectively. How long will it take to complete the execution of these processes? **Please discuss all possibilities** depending on different processes scheduled to run on different CPUs/threads, and **what is the minimum execution time**? Assume that all three processes are 100% CPU bound, do not block during execution, and do not change CPUs once assigned.

Since there are only two CPUs, then:

CPU1 for P0 and CPU2 for P1 and P2, so it will take 5 and 10+20 msec, which is 30.

CPU1 for P0 and P1, CPU2 for P2, so it will take 5+10 and 20 msec, which is 20.

CPU1 for P0 and P2, CPU2 for P1, so it will take 5+20 and 10 msec, which is 25.

CPU1 for P1 and P2, CPU2 for P0, so it will take 10+20 and 5 msec, which is 30.

If P0, P1 and P2 are scheduled with same CPU, then it will take 5+10+20, which is 35.

So, the minimal time is 20 and maximal time is 35.

**Q4 (Problem 12).** In Fig. 2-8, a multithreaded Web server is shown. If the only way to read from a file is the normal blocking read system call, do you think user-level threads or kernel-level threads are being used for the Web server? Why?

It shall use kernel-level threads because it will block some threads but will not affect other threads. On the other hand, user-level threads will block the entire process, which violate the multithreading.

**Q5 (Problem 27).** In a system with threads, is there one stack per thread or one stack per process when user-level threads are used? What about when kernel-level threads are used? Please explain.

Each thread should have its own stack in order to save local variables, no mater it’s user-level threads or kernel-level threads.

**Q6** Please briefly discuss the advantages and disadvantages of implementing threads in user space and kernel space, respectively.

For user space:

Advantages:

Fast thread switching

Each process has its own scheduling algorithm

Disadvantages:

Block entire process when system call

Page faults will block entire process

No clock interrupts

Limited performance

For kernel space:

Advantages:

Performance gain

None-block compare to user-space

Disadvantages:

Higher cost due to its life cycle

**Q7 (Problem 24).** Does Peterson's solution to the mutual exclusion problem shown in Fig. 2-24 work when process scheduling is preemptive? How about when it is non-preemptive?

Yes it works fine when process scheduling is preemptive, but it could fail if working under non-preemptive because while turn is always zero but process 1 runs first, in that case, it will cause a dead lock.

**Q8 (Problem 43).** Measurements of a certain system have shown that the average process runs for a time T before blocking on I/O. A process switch requires a time S, which is effectively wasted (overhead). For round-robin scheduling with quantum Q, give a formula for the CPU efficiency, defined as the useful CPU time divided by the total CPU time including the overhead, for each of the following:

1. Q = ∞
2. Q > T
3. S< Q< T
4. Q = S< T
5. Q nearly 0

Based on Efficiency = T / (T+ST/Q)

For:

* + - * 1. S = T / T = 100%
        2. S = T / T = 100%
        3. Efficiency = Q / (Q+S)
        4. S = T / 2T = 50%
        5. Efficiency = 0% as Q goes to 0

**Q9 (Modified Problem 45).** Five batch jobs A through E, arrive at a computer center at almost the same time. They have estimated running times of 10, 4, 2, 6, and 8 minutes. Their (externally determined) priorities are 3, 5, 4, 2, and 1, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the average process turnaround time. Ignore process switching overhead.

1. Round robin.
2. Priority scheduling.
3. First-come, first-served (run in order 10, 4, 2, 6, 8).
4. Shortest job first.

For (a), assume that the system is multiprogrammed, and that each job gets its fair share of the CPU. For (b) through (d) assume that only one job at a time runs, until it finishes. All jobs are completely CPU bound.

Round Robin:

* + - 1. First round:

Took 10 minutes since 5 process and C took 10 minutes to finish.

After 10 minutes, ABCDE left 8 2 0 4 6 minutes

* + - 1. Second round:

Took 8 minutes since 4 process and B took 8 minutes to finish.

After 8 minutes, ABCDE left 6 0 0 2 4 minutes

* + - 1. Third round:

Took 6 minutes since 3 process and D need 6 minutes to finish.

After 6 minutes, ABCDE left 4 0 0 0 2 minutes

* + - 1. Fourth round:

Took 4 minutes since 2 process and E need 4 minutes to finish.

* + - 1. Fifth round:

Took 2 minutes to finish all process.

So, the average time = (10+18+24+28+30)/5 = 22 minutes

For part b,

Based on priority, B->C->A->D->E, average = (4+6+16+22+30)/5 = 15.6 minutes

For part c,

FCFS, average = (10+14+16+22+30)/5 = 18.4 minutes

For part d,

Shortest job first, average = (2+6+12+20+30)/5 = 14 minutes

**Q10 (Problem 55).** Consider the procedure *put\_forks* in Fig. 2-47. Suppose that the variable *state*[*i*] was set to *THINKING* *after* the two calls to *test*, rather than *before*. How would this change affect the solution?

This change will result in a dead lock due to the adjacent philosophers will stay hungry for ever.

THE END.