United International University

Department of Computer Science and Engineering

CSE 4509/CSI 309: Operating System Concepts/Operating Systems

Midterm Examination: Fall 2024

Total Marks: 30 Time: 1 hour 30 minutes

Any examinee found adopting unfair means will be expelled from the trimester / program as per UIU disciplinary rules.

Answer all the questions. Numbers to the right of the questions denote their marks.

1. (a) How does virtualization of CPU give us the illusion of running multiple applications simultaneously? [2]

   (b) How does the shell (your terminal in linux) execute user commands using fork(), wait() and exec() system calls? [3]

2. Consider the following codes: Given the **initial process id is 1501**, subsequent process IDs will increment by one with each new process created.

f1.c

```
1.  int main()
2.  {
3.      int c1 = fork();
4.      if(c1 == 0 || fork())
5.      {
6.          printf("Linux\n");
7.          char *args[2];
8.          args[0] = strdup("./f3");
9.          args[1] = NULL;
10.         execvp(args[0], args);
11.     }
12.     else if(c1 > 0)
13.     {
14.         waitpid(c1, NULL, 0);
15.         printf("Windows\n");
16.     }else
17.     {
18.         printf("Android\n");
19.     }
20. }
```

f3

```
1.  int main()
2.  {
3.      printf("Best of luck\n");
4.  }
```

(a) Draw the Process tree for the code in f1.c [3]

(b) Find the possible output if f1.c is executed. [2]

3. Consider the following rules of the MLFQ scheduling algorithm and answer the questions below:

Rule 1: If Priority(A) > Priority(B), A runs (B doesn't). **Rule 2:** If Priority(A) = Priority(B), A & B run in RR. **Rule 3:** When a job enters the system, it is placed at the highest priority (the topmost queue). **Rule 4a:** If a job uses up an entire time slice while running, its priority is reduced (i.e., it moves down one queue). **Rule 4b:** If a job gives up the CPU before the time slice is up, it stays at the same priority level.

(a) Find out two problems (or limitations) of the above algorithm. Use gantt charts to show the examples of each problem. [4]

(b) Propose modifications of the above rule(s) that will solve the problems you have mentioned in 3(a). [2]

(c) Consider the following data and draw a gantt chart using the modified version of the algorithm mentioned above and **find the average turnaround time.** Additionally, there are **3 Queues: Q1, Q2 and Q3 and the priority order is: Q3 > Q2 > Q1. Time slice = 3 ms.**

Priority boosting occurs after every 24 ms
[4]

Process	**P**	**Q**	**R**	**S**	**T**	**U**
Arrival (nth ms)	0	9	15	15	15	72
Duration (ms)	12	15	6	18	15	3

4. (a) Let's examine a program having two threads:

Thread 1	**Thread 2**

`pending = 1;`
`while(pending){`
`  printf("Hello\n");`
`}`
	`pending = 0;`

How could we rewrite the code such that Thread 2 would only run after "Hello" has been printed at least twice? [2]

(b) Consider the following implementation of a spin lock and show how a race condition may occur. [3]

```
void acquire(bool *lock)
{
    while(*lock);
    *lock = true;
}
```

```
void release(bool *lock)
{
    *lock = false;
}
```

5. Consider the following synchronization problem. A group of children are picking chocolates from a box that can hold up to N chocolates. A child that wants to eat a chocolate picks one from the box to eat, unless the box is empty. If a child finds the box to be empty, she wakes up the mother, and waits until the mother refills the box with N chocolates. Unsynchronized code snippets for the child and mother threads are as shown below: [5]

Child	**Mother**
`while(true) {` `  getChocolateFromBox();`	`while(true) {` `  refillChocolateBox(N);`

`eat(); }`	`}`

Now we need to add suitable synchronization such that a child invokes getChocolateFromBox() only if the box is non-empty, and the mother invokes refillChocolateBox(N) only if the box is fully empty. The synchronization part in the mother thread is done for you. **Understand the code and you need to complete the child threads' part.** Solve this question using only locks and condition variables, and no other synchronization primitive. The following variables have been declared for use in your solution.

int count = 0;

mutex m; // you may invoke lock and unlock

condvar fullBox, emptyBox; //you may perform wait and signal //or signal_broadcast

Synchronization Code for Mother Thread

```
while(true) {
    lock(m);
    if(count > 0) {
        wait(emptyBox, m);
    }
    refillChocolateBox(N);
    count += N;
    signal_broadcast(fullBox);
    unlock(m);
}
```