United International University
Department of Computer Science and Engineering

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

Midterm Examination: Fall 2023

Total Marks: 30 Time: 1 hour 45 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) **Find** the all possible output of the following code. [5]

```
#include <bits/stdc++.h>
using namespace std;

int O = 0, C = 1, G = 0;
void P(int *sem){
    while(*sem<=0);
    *sem = *sem - 1;
}
void V(int *sem){
    *sem = *sem + 1;
}
void claims(){
    while(1){
        P(&O);
        printf("Os is Fun\n");
        V(&G);
    }
}
void observations(){
    P(&G);
    printf("Os is disaster\n");
    V(&O);
    V(&C);
}
void greetings(){
    do{
        P(&C);
        printf("Welcome to OS world!\n");
        V(&O);
    }while(1);


}
int main()
{
    thread greetingsThread(greetings);
    thread claimsThread(claims);
    thread observationsThread(observations);

    greetingsThread.join();
    claimsThread.join();
    observationsThread.join();

    return 0;
}
```

(b) For the following code, **find** out where the race condition arises. Then **modify** the code to resolve those potential race conditions using the mutex lock approach. [3 + 2]

```
#include<bits/stdc++.h>
#include <mutex>
#include <semaphore.h>
using namespace std;

const int capacity = 100;
int products = 0, share = 50, amounts = 100, price;
void shopkeeper()
{
    int item;
    while (TRUE) {
        item = produce_item( ); // produce an item
        if (products == capacity) sleep( );
        price += item * 2;
        amounts -= price;
        share = price-products;
        products = products + 1;
        if (products == 1) wakeup(customer);
    }
}
void customer()
{
    int item;
    while (TRUE) {
        if (products <= 0) sleep( );
        item = buy_item(); // buy an item from shopkeeper
        products = products-1;
        amounts += 10;
        item = 0;
        if (products == capacity-1) wakeup(shopkeeper);

    }
}

int main() {
    srand(static_cast<unsigned int>(time(nullptr)));

    thread shopThread(shopkeeper);
    thread customerThread(customer);


    shopThread.join();
    customerThread.join();

    return 0;
}
```

[You do not need to write the full code. Write only the modifications.]

2. Suppose a multiprocessor system has 3 processors labeled P_1, P_2, and P_3 respectively, and 3 scheduling queues labeled Q_1, Q_2, and Q_3 respectively. For any processor P_n, the respective queue is Q_n. Any processes can be assigned to P_1 and P_2 but only real-time processes can be assigned to Q_3. The details of the processes can be found in 1a and queues in 1b.

   (a) **Draw** the Gantt charts for all three processors. [5]

   (b) **Calculate** turnaround time, response time, and wait time for all three processes [3]

   (c) If all three processors start at timestep 0, **calculate** CPU utilization for all three processors. [2]

   [**Note: the scheduler always prioritizes assigning processes to an idle processor over an already busy one.**]

Process ID	Process type	Arrival Time	Burst Time
1007	Batch	0	7
1013	Batch	2	4
312	Real Time	4	3
597	Interactive	5	8
1523	Interactive	7	3
5971	Real Time	7	4

(a) Details of the Processes

Queue	Algorithm	Remarks
Q_1	Shortest Job First	Preemptive version
Q_2	Round Robin	Time Quantum=4
Q_3	First Come First Serve	-

(b) Details of the Queues

Table 1: Necessary information for question 2

3. (a) Consider the following code:

```
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
        pid_t id1 = fork();
        pid_t id2 = fork();

        if (id1 > 0 && id2 > 0) {
                printf("Parent Terminated\n");
        }
        else {
                pid_t id3=fork();
                if(id3>0){
                        printf("Child Terminated\n");
                }else{
                        printf("Grand child Terminated\n");
                        exit(0);
                }
        }
        printf("Bye\n");
        return 0;
}
```

i. **Draw** a process tree for the above code. [2]

ii. **Write** the possible output for the above code. [3]

(b) i. Suppose that a new process has been created. It has the Burst time (in ms) as follows:

CPU burst	I/O burst	CPU burst
4	3	8

Table 2: burst time for the process

Now, **write** the sequence of states as process executes from creation to termination[Consider that no other process is in the system].

ii. What is a time-sharing operating system, and how does it enable multiple users to efficiently share a single computer system's resources? [1+2]

(c) Imagine you are tasked with designing an operating system for a highly modular and extensible autonomous vehicle. The goal is to create a system that allows for easy updates to individual components like sensors, control algorithms, and communication protocols. Which structure/architecture will you prefer to design this operating system? Give proper reasoning for your answer. [2]