United International University (UIU)
Department of Computer Science and Engineering
CSE 4509: Operating Systems, Midterm Exam, Summer 2025
Total Marks: 30, Duration: **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. Marks are indicated on the right.

1. A process with Process ID (PID) 2025 is executed with the following code. Any child process created by it will assume the next available PID values sequentially (*e.g.*, 2026, 2027, ...).

foo.cpp

```
int main()
{
    if (fork() > 0) {
        cout << "This is parent" << endl;
        int i = 3;
        while (i > 0 || fork() > 0) {
            cout << i << endl;
            i--;
            if ( i < 0 ) break;
        }
        while ( wait(NULL) > 0 );
        if (i < 0 ) cout << "This is parent again" << endl;
        else cout << "This is a child" << endl;
    }
    else {
        cout << "This is another child" << endl;
        char *args[2];
        args[0] = strdup("./bar");
        args[1] = NULL;
        execvp(args[0], args);
        if( fork() == 0 ) cout << "This is grandchild" << endl;
    }
}
```

bar.cpp

```
int main() {
    cout << "This is another program" << endl;
}
```

a. Draw the process tree for the code in **`foo.c`** [4]

b. Find a possible output sequence if **`foo.c`** is executed. [4]

c. How does the shell (your terminal in linux) execute user commands using **`fork()`**, **`wait()`** and **`exec()`** system calls? [2]

2. Consider a system with **two** queues in an MLFQ scheduler. Each queue has a **time slice (quantum) of 5ms**, and **boosting occurs every 13ms**. No I/O operations are involved, and the round-robin scheduling is maintained in each queue. **When a job is moved to a lower-priority queue, it is added to the queue's tail. While selecting jobs to run, the one which is at the head of queue is taken first.** The processes arrive as follows:

- Jobs A, B, and C **arrive at time 0ms** with a **total running time of 10ms each**.
- Job D **arrives at time 14ms** with a **total running time of 5ms**.

The MLFQ scheduling rules are as follows:

- Rule 1: If Priority(A) > Priority(B), A runs (B doesn't).
- Rule 2: If Priority(A) = Priority(B), A and B run in round-robin fashion using the time slice (quantum length) of the given queue.
- Rule 3: When a job enters the system, it is placed at the highest priority (the topmost queue).
- Rule 4: Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (i.e., it moves down one queue).
- Rule 5: After some time period S, move all the jobs in the system to the topmost queue.

a. Draw the Gantt chart and simulate how the processes will run. [3]
b. What is the average turnaround time? [1]

A new OS scheduling policy estimates each process P_i's expected CPU burst using its average past burst time c_i. For new processes, default values of c_i are used. For every process in the *ready queue*, the scheduler computes:

$$f_i = \frac{1}{ac_i + b}$$

where a and b are positive constants. At each context switch (voluntary or involuntary), the scheduler selects the process with the highest f_i and runs it either until the time quantum expires or the process yields the CPU.

Answer **yes/no** to the questions below, with a **one-sentence justification**.

c. Does this policy prioritize I/O-bound processes over CPU-bound processes? [2]
d. Does this policy schedule all processes for equal number of times? [2]
e. Can this policy cause starvation of some processes? [2]

3. Every day, a UIU shuttle picks up waiting students from a shuttle stop periodically. The shuttle has a **capacity of K.** The shuttle arrives at the pickup point, allows up to K waiting students (fewer if less than K are waiting) to board, and then departs. Students have to wait for the shuttle to arrive and then board it. Students who arrive at the bus stop after the bus has arrived should not be allowed to board, and should wait for the next time the bus arrives. The shuttle and students are represented by threads in a program. **The student thread should call the function `board()` at the time of boarding, and the shuttle should invoke `depart()` when it has boarded the desired number of students and is ready to depart.** The threads share the following variables, none of which are implicitly updated by functions like **`board()`** or **`depart()`**. Below is given synchronized code for the student thread. You should not modify this in any way.

```
mutex m // lock variable
// three conditional variables
cv_shuttle_arrived, cv_student_boarded, cv_queue_open
waiting_count = 0 // integer
shuttle_arrived = false // boolean

def student_thread():
    lock(&m)

    while shuttle_arrived: // (late) not allowed on this bus
        // wait until the shuttle leaves to join the queue
        wait(&cv_queue_open, &m);

    waiting_count++

    while not shuttle_arrived:
        // wait until the shuttle arrives to board
        wait(&cv_shuttle_arrived, &m)

    board() // board the shuttle
    signal(&cv_student_boarded)

    unlock(&m)
```

a. Write down the corresponding synchronized code for the `uiu_shuttle_thread` that achieves the correct behavior specified above. [5]
The shuttle should board the correct number of students, based on its capacity and the number of those waiting. The shuttle should correctly board these students by calling the `wait/signal/broadcast` functions appropriately. The shuttle code should also update the waiting count as required. Once boarding completes, the shuttle thread should call **`depart()`**. You can use any extra local variables in the code of the shuttle thread, like integers, loop indices and so on. However, you must not use any other extra synchronization primitives (lock or conditional variables).

For the implementation of locking mechanism needed for your threads in the previous problem, consider the following code.

```
bool lock1 = false;
void acquire(bool *lock) {
   while (*lock);
   *lock = true;
}
```

b. What problem(s) may arise with the above implementation of lock? Propose a solution to tackle the problem(s). [3]

c. Why is it necessary for threads in a process to have separate stacks? [2]

— End of Question Paper —