National University of Computer and Emerging Sciences, Lahore Campus

	Course:	**Operating Systems**	**Course Code:**	**CS 2006**
	Program:	**BS(CS)**	**Semester:**	**Spring 2024**
	Section:	**F,G,H**	**Total Marks:**	**75**
	Assignment:	**3** **SOLUTION**	**Pages**	**(4)**

Question 2: [15 marks]

Consider the following code for a simple Stack:

```
class Stack {
private:
	int* a; // array for stack
	int max; // max size of array
	int top; // stack top
public:

	Stack(int m) {
		a = new int[m]; max = m; top = 0;
	}

	void push(int x) {
		while (top == max)
			; // if stack is full then wait
		a[top] = x;
		++top;
	}

	int pop() {
		while (top == 0)
			; // if stack is empty then wait
		int tmp = top;
		--top;
		return a[tmp];
	}
```

```
};
```

You can see from the code that a process blocks if it calls push() when the stack is full, or it calls pop() when the stack is empty, the same behavior should be present in the answer. Assuming that the functions push and pop can execute concurrently, synchronize the code using semaphores. Also eliminate the busy waiting.

```
class Stack {
private:
        int* a;  // array for stack
        int max;        // max size of array
        int top; // stack top
        sem push_s;
         sem pop_s;
        mutex m;
public:

        Stack(int m) {
                a = new int[m];max = m;         top = 0;
                 push_s=10; pop_s=0; m=false
        }

        void push(int x) {
                wait(push_s)
                wait(m)
                a[top] = x;
                ++top;
                signal(m)
                signal(pop_s)
        }

        int pop() {
                wait(pop_s)
                wait(m)
                int tmp = top;
                --top;
                signal(m)
                signal(push_s)
                return a[tmp];
        }
};
```

Question 3: [20 marks]

Picture a theater with separate sections for comedy and drama performances. When someone enters either section, others interested in the same genre may join, but those preferring the other genre must wait. A sign outside each section indicates its current status:

- Empty
- Comedy show in progress
- Drama show in progress

Develop the synchronization procedures for the following scenarios:

1. comedy_fan_wants_to_enter
2. drama_enthusiast_wants_to_enter
3. comedy_fan_leaves
4. drama_enthusiast_leaves

Make use of counters, semaphores, or synchronization techniques.

You can submit a doc file or upload a scanned handwritten solution in pdf format for Q2 and 3.

Note: The proposed solution is similar to the First Readers Writes structure that is presented in section 7.1.2. of the textbook.

semaphore cr_entry_mutex = 1; //a binary semaphore

semaphore comedy_fan_mutex = 1; //a binary semaphore to ensure mutual exclusion for the variable comedy_fan_count

semaphore drama_enthusiast_mutex = 1; //a binary semaphore to ensure mutual exclusion for the drama_enthusiast_count

int comedy_fan_count = 0; //keeps track of how many comedy fans are in the theater

```
int drama_enthusiast_count = 0; //keeps track of how many drama enthusiast are in the theater
```

Structure of a drama enthusiast process

```
while (true)

{

        /*structure of the procedure for “drama_enthusiast_wants_to_enter”*/

wait (drama_enthusiast_mutex);

drama_enthusiast_count++;

if (drama_enthusiast_count == 1)

        wait(cr_entry_mutex);

signal(drama_enthusiast_mutex);

.       .       .

/* Entry in Theater*/

.       .       .

        /* structure of the procedure for drama_enthusiast_leaves*/

        wait(drama_enthusiast_mutex);

        drama_enthusiast_count--;

        if (drama_enthusiast_count== 0)
```

```
                    signal (cr_entry_mutex);

                signal(drama_enthusiast_mutex);

}
```

Structure of a comedy fan_process

```
while (true)

{

                /* structure of the procedure for comedy_fan_wants_to_enter*/

wait (comedy_fan_mutex);

comedy_fan_count++;

if (comedy_fan_count == 1)

                wait(cr_entry_mutex);

signal(comedy_fan_mutex);

.               .               .

/* Entry in Theater*/

.               .               .

                /* structure of the procedure for comedy_fan_leaves*/

                wait(comedy_fan_mutex);

                comedy_fan_count--;
```

```
                    if (comedy_fan_count == 0)

                            signal (cr_entry_mutex);

                    signal(comedy_fan_mutex);

}
```