**21L-7512 Abdullah Dar BSCS-4G**

**National University of Computer and Emerging Sciences**

**Operating System**

**Assignment No # 3**

**Question # 4:**

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]; }**

**};**

Assuming the functions push and pop can execute concurrently, synchronize the code using semaphores. Also, replace the busy waiting with proper waiting.

**Solution:**

**class Stack{**

**private:**

**int \*a;**

**int max;**

**int top;**

**sem\_t full;**

**sem\_t empty;**

**sem\_t sem\_push;**

**sem\_t sem\_pop;**

**public:**

**Stack(int m)**

**{**

**a = new int[m];**

**max = m;**

**top = 0;**

**sem\_init(&full,0,0);**

**sem\_init(&empty,0,m);**

**sem\_init(&sem\_push,0,1);**

**sem\_init(&sem\_pop,0,1);**

**}**

**void push(int x)**

**{**

**sem\_wait(&sem\_push);**

**sem\_wait(&empty);**

**a[top] = x;**

**++top;**

**sem\_post(&full);**

**sem\_post(&sem\_push);**

**}**

**int pop()**

**{**

**sem\_wait(&sem\_pop);**

**sem\_wait(&full);**

**--top;**

**int temp = a[top];**

**sem\_post(&empty);**

**sem\_post(&sem\_pop);**

**return temp;**

**}**

**};**

**----------------------------------------------------------------------**

**Question # 5:**

**Playing Cricket**

You need at least one bowler and one batsman to play cricket. There are two threads, one for bowlers and one for batsmen. There is room for only one practice session. A practice session is possible if there is a bowler and a batsman available. If a batsman arrives, and a bowler is already available, he will engage a bowler for a practice session and vice versa. If however, a new batsman arrives and a bowler is not available, the batsman will hold the court and wait for the bowler to arrive and vice versa. No two bowlers can coexist the same time the same way as no two batsmen. Synchronize your solution by using semaphore.

**Solution:**

| **// Declare any Shared Variable or Semaphores** |
| --- |
| int batsmanFlag = 0;  Int bowlerFlag = 0;  sem\_t batsman\_sem;  sem\_t bowler\_sem; |

| **Batsman** | **Bowler** |
| --- | --- |
| Batsman()  {  sem\_wait(&batsman\_sem);  batsmanFlag = 1;  while(bowlerFlag == 0); // do nothing  **Critical Section : PLAY();**  batsmanFlag = 0;  sem\_post(&batsman\_sem);  } | Bowler()  {  sem\_wait(&bowler\_sem);  bowlerFlag = 1;  while(batsmanFlag == 0); // do nothing  **Critical Section : PLAY();**  bowlerFlag = 0;  sem\_post(&bowler\_sem);  } |

**// OR** Not using while loop and replacing it with semaphores.

| **// Declare any Shared Variable or Semaphores** |
| --- |
| sem\_t batsmanFlag; // initillize to zero  sem\_t bowlerFlag; // initillize to zero  sem\_t batsman\_sem;  sem\_t bowler\_sem; |

| **Batsman** | **Bowler** |
| --- | --- |
| Batsman()  {  sem\_wait(&batsman\_sem);  sem\_post(&batsmanFlag);  sem\_wait(&bowlerFlag);  **Critical Section : PLAY();**  sem\_wait(&batsmanFlag);  sem\_post(&batsman\_sem);  } | Bowler()  {  sem\_wait(&bowler\_sem);  sem\_post(&bowlerFlag);  sem\_wait(&batsmanFlag);  **Critical Section : PLAY();**  sem\_wait(&bowlerFlag);  sem\_post(&bowler\_sem);  } |

**----------------------------------------------------------------------**