BAHRIA UNIVERSITY (KARACHI CAMPUS)

OPEN ENDED LAB II – **Fall22**

(System Programing (LAB) **CSC-454)**

Class: BSE [4]-5 (B) (Morning)

Course Instructor: **Engr Rizwan Fazal / Engr Rehan Baig**

Time Allowed: **1.5 Hour**

Max Marks: **6**

Student's Name: **Usama**

Reg. No: **69991**

Instructions:

1. Submit your answers within file against each question with screenshot of both code and solution output.
2. File must be submitted in .pdf.

[CLO#05, 6 marks]

SCENARIO:

You are working as a system engineer in a Microsoft vendor company that creates Apps for Microsoft store.
Your Project manager assigned you a task to design an application for code editor for Microsoft store. For that you need to analyze the basics of NotePad/WordPad applications that comes built-in with Microsoft windows. You need to create a process and analyze the following for notepad and WordPad.

Q1: Run a loop or Use Recursion which enable program to print 5 times following for both Notepad and WordPad **(versionId, ThreadId, processId)**, meanwhile use exit thread function that-should be interrupt when counter reaches on 4rth iteration. (**4 Marks**)

Solution:

```
#include <iostream>
#include <thread>
#include <unistd.h>
using namespace std;

void print_info(int counter) {
    if (counter == 4) {
        pthread_exit(NULL);
    }
    string version_id = (counter % 2 == 0) ? "Notepad" : "WordPad";
    cout << "versionId = "<<version_id << ", ThreadId = " << this_thread::get_id()
<< ", ProcessId = " << getpid() << endl;
}

int main() {
    cout << "\n\nName: Usama \nEnrollment: 02-131202-042\n" << endl;
    for (int i = 0; i < 5; i++) {
        cout << "\nIteration " << i+1 << ": " << endl;
        thread t(print_info, i);
        t.join();
    }
    return 0;
}
```

Description:

In this code, main function creates a new thread t and assigns it the print_info function with counter variable as argument.
In the print_info function, it checks whether the counter is equal to 4 or not. If it's equal to 4 then it exit the thread by calling pthread_exit(NULL) function.
Then it prints the versionId and threadId and processId. The versionId is determined by counter modulus 2, if it's even it will print "Notepad" otherwise "WordPad". The threadId and processId are obtained by calling the this_thread::get_id() and getpid() functions respectively.
After each iteration, the thread t is joined to the main thread. When the counter reaches 4, the thread exits without printing anything and the loop ends.

OUTPUT:

```
Name: Usama
Enrollment: 02-131202-042


Iteration 1:
versionId = Notepad, ThreadId = 140031364749056, ProcessId = 13027

Iteration 2:
versionId = WordPad, ThreadId = 140031364749056, ProcessId = 13027

Iteration 3:
versionId = Notepad, ThreadId = 140031364749056, ProcessId = 13027

Iteration 4:
versionId = WordPad, ThreadId = 140031364749056, ProcessId = 13027
```

Q2: Write a code for any two synchronization objects from following. (**2 Marks**)

1. Events
2. Semaphores
3. Mutexes

Using Event Synchronisation:

```
#include <iostream>
#include <thread>
#include <condition_variable>
using namespace std;

condition_variable cv;
mutex m;
bool event_set = false;

void worker() {
    unique_lock<std::mutex> lock(m);
    cv.wait(lock, [](){return event_set;});
    cout << "Event set, continuing execution" << endl;
}

int main() {
    cout << "Name: Usama\nEnrollment: 02-131202-042\n" << endl;
    cout << "Synchronization using Events" << endl << endl;
    thread t(worker);
    cout << "Waiting for event to be set..." << endl;
    unique_lock<std::mutex> lock(m);
    event_set = true;
    cv.notify_one();
    t.join();
    std::cout << "Event has been set, worker thread has completed." << endl;
    return 0;
}
```

Description of Code:

In this code, main function creates a new thread t and assigns it the worker function. Then, it prints "Waiting for event to be set..." to indicate that the main thread is waiting for the event to be set.

In the worker function, a unique_lock is created and the wait function is called on the condition_variable object cv with the lock and a lambda function as arguments. This lambda function returns event_set, which is initially set to false. This causes the worker thread to wait until the event is set.

```
Name: Usama
Enrollment: 02-131202-042

Synchronization using Events

Waiting for event to be set...
Event set, continuing execution
Event has been set, worker thread has completed.
```

Using Semaphore Synchronization:

```
#include <iostream>
#include <thread>
#include <semaphore>

using namespace std;

semaphore sem(3);

void worker() {
    sem.wait();
    cout << "Acquired semaphore, continuing execution." << endl;
    sem.post();
}

int main() {
    cout <<"Name: Usama\nEnrollment: 02-131202-042"<< endl << endl;
    cout << "\n\t\tSynchronization using Semaphore\n" << endl;
    thread t[5];
```

```
    for (int i = 0; i < 5; ++i) {
        t[i] = thread(worker);
    }
    for (int i = 0; i < 5; ++i) {
        t[i].join();
    }
    return 0;
}
```

Description of Code:

In this code, main function creates an array of 5 threads t and assigns them the worker function. Then, main thread is creating threads for semaphore synchronization.
In the worker function, semaphore's wait function is called which decrement the semaphore's count by 1. If the count is less than or equal to 0 then the thread will wait until a post() operation increments the count. Once the count is greater than 0, the wait function will decrement the count and continue execution. After that semaphore's post function is called which increments the semaphore's count by 1 and wake up one thread that is waiting on this semaphore.
In this way, the semaphore's count will be 3, and only 3 threads will execute the worker function at a time and will print "Acquired semaphore, continuing execution.".
After that all the threads joined the main thread, the program exits.

OUTPUT:

```
Name: Usama
Enrollment: 02-131202-042


        Synchronization using Semaphore

Acquired semaphore, continuing execution.
Acquired semaphore, continuing execution.
Acquired semaphore, continuing execution.
Acquired semaphore, continuing execution.
Acquired semaphore, continuing execution.
```