BAHRIA UNIVERSITY (KARACHI CAMPUS)

OPEN ENDED LAB II – **Fall22**

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

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

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

Time Allowed: **1.5 Hour**

Max Marks: **6**

Student's Name: **GHULAM MOHIUDDIN SHAIKH**

Reg. No: **_69995__**

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**)

CODE:

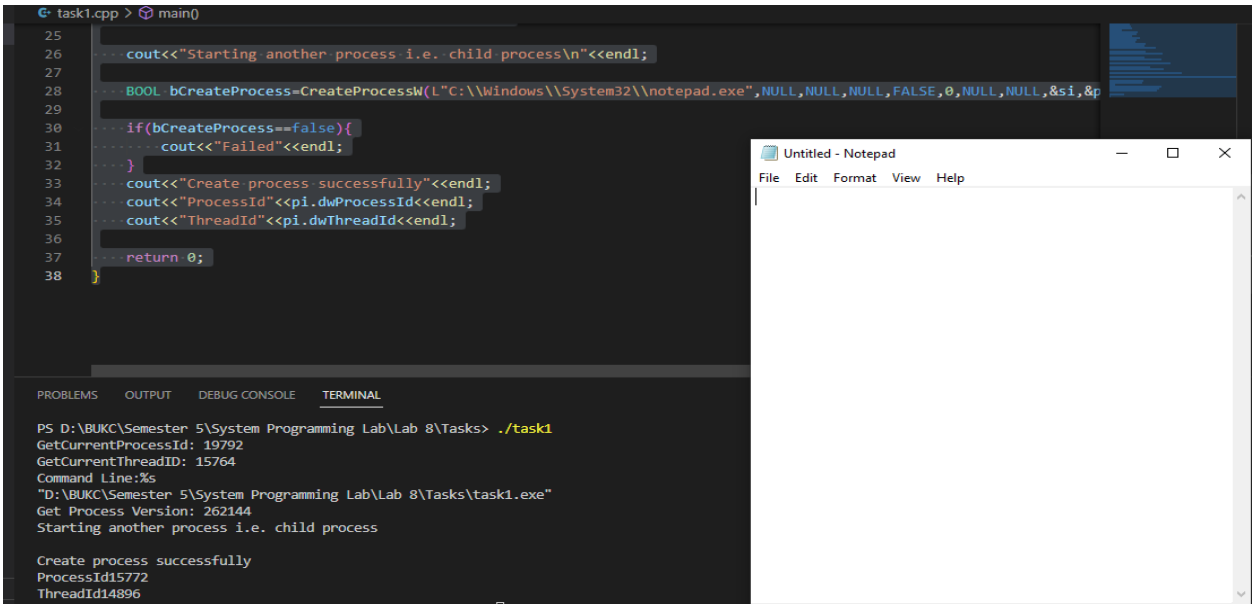
```
#include<iostream>
#include<Windows.h>
using namespace std;
int main(){
   HANDLE hprocess=NULL;
   HANDLE hthread=NULL;
   STARTUPINFOW si;
   PROCESS_INFORMATION pi;
   DWORD dwProcessId=0;
   DWORD dwthreadId=0;
```

```
    ZeroMemory (&si,sizeof(si));
    ZeroMemory(&pi,sizeof(pi));
    DWORD Ret=0,dwPID=0,dwTID=0,dwPver=0;

    dwPID=GetCurrentProcessId();
    cout<<"GetCurrentProcessId: "<<dwPID<<endl;

    dwTID=GetCurrentThreadId();
    cout<<"GetCurrentThreadID: "<<dwTID<<endl;

    cout<<"Command Line:%s\n"<<GetCommandLine()<<endl;

    dwPver=GetProcessVersion(dwPID);
    cout<<"Get Process Version: "<<dwPver<<endl;

    cout<<"Starting another process i.e. child process\n"<<endl;

    BOOL
bCreateProcess=CreateProcessW(L"C:\\Windows\\System32\\notepad.exe",NULL,NULL,NULL,FALSE,0,NULL,NUL
L,&si,&pi);

    if(bCreateProcess==false){
        cout<<"Failed"<<endl;
    }
    cout<<"Create process successfully"<<endl;
    cout<<"ProcessId"<<pi.dwProcessId<<endl;
    cout<<"ThreadId"<<pi.dwThreadId<<endl;

    return 0;}


#include<iostream>
#include<Windows.h>
using namespace std;
int main(){
    HANDLE hprocess=NULL;
    HANDLE hthread=NULL;
    STARTUPINFOW si;
    PROCESS_INFORMATION pi;
    DWORD dwProcessId=0;
    DWORD dwthreadId=0;
    ZeroMemory (&si,sizeof(si));
    ZeroMemory(&pi,sizeof(pi));
    DWORD Ret=0,dwPID=0,dwTID=0,dwPver=0;

    dwPID=GetCurrentProcessId();
    cout<<"GetCurrentProcessId: "<<dwPID<<endl;

    dwTID=GetCurrentThreadId();
    cout<<"GetCurrentThreadID: "<<dwTID<<endl;

    cout<<"Command Line:%s\n"<<GetCommandLine()<<endl;
```

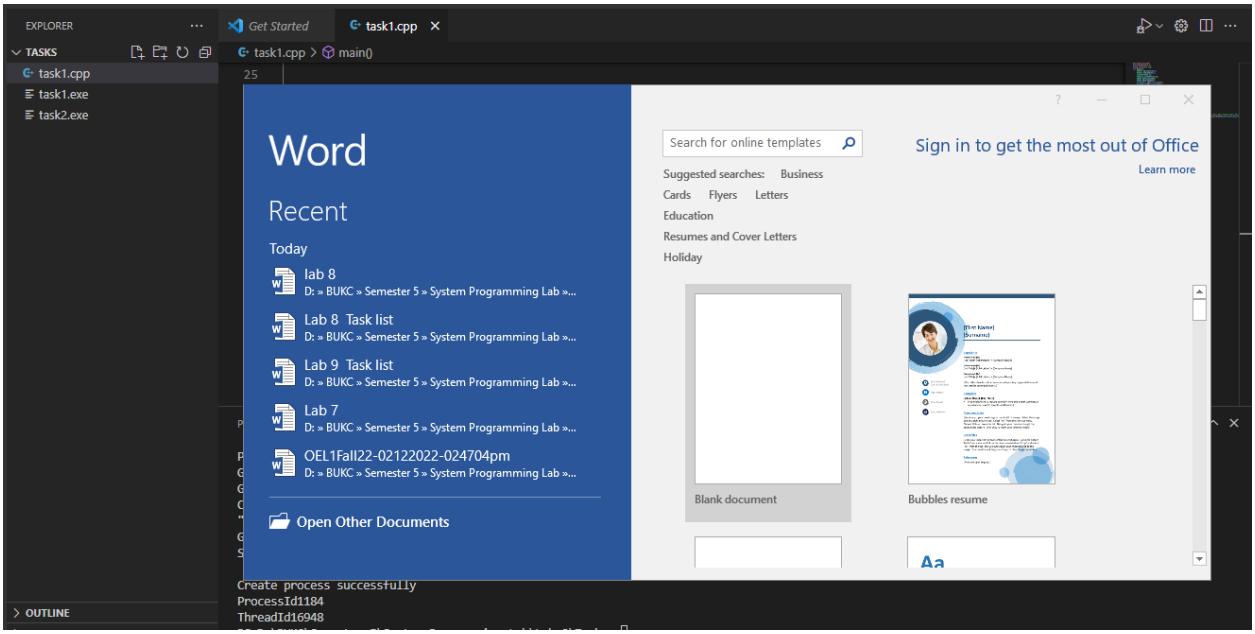
```
    dwPver=GetProcessVersion(dwPID);
    cout<<"Get Process Version: "<<dwPver<<endl;

    cout<<"Starting another process i.e. child process\n"<<endl;

    // BOOL
bCreateProcess=CreateProcessW(L"C:\\Windows\\System32\\notepad.exe",NULL,NULL,NULL,FALSE,0,NULL,NUL
L,&si,&pi);
    BOOL bCreateProcess=CreateProcessW(L"C:\\Program Files\\Microsoft
Office\\Office16\\WINWORD.exe",NULL,NULL,NULL,FALSE,0,NULL,NULL,&si,&pi);
    if(bCreateProcess==false){
        cout<<"Failed"<<endl;
    }
    cout<<"Create process successfully"<<endl;
    cout<<"ProcessId"<<pi.dwProcessId<<endl;
    cout<<"ThreadId"<<pi.dwThreadId<<endl;

    return 0;
}
```

OUTPUT:

```
GetCurrentProcessId: 19792
GetCurrentThreadID: 15764
Command Line:%s
"D:\BUKC\Semester 5\System Programming Lab\Lab 8\Tasks\task1.exe"
Get Process Version: 262144
Starting another process i.e. child process

Create process successfully
ProcessId15772
ThreadId14896
```



```
GetCurrentProcessId: 18588
GetCurrentThreadID: 788
Command Line:%s
"D:\BUKC\Semester 5\System Programming Lab\Lab 8\Tasks\task2.exe"
Get Process Version: 262144
Starting another process i.e. child process

Create process successfully
ProcessId1184
ThreadId16948
```



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

1. Events

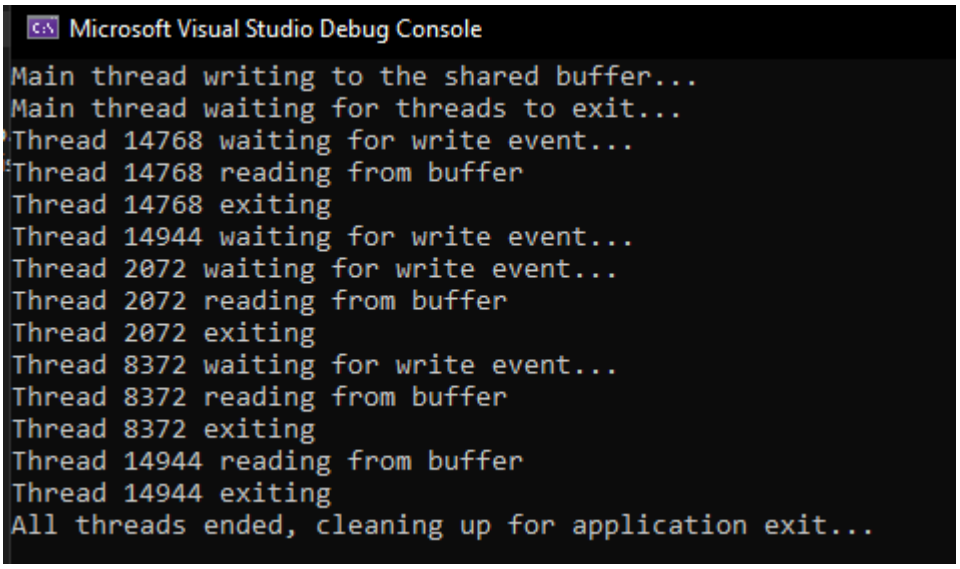
CODE:

```
#include <windows.h>
#include <stdio.h>

#define THREADCOUNT 4

HANDLE ghWriteEvent;
HANDLE ghThreads[THREADCOUNT];

DWORD WINAPI ThreadProc(LPVOID);

void CreateEventsAndThreads(void)
{
   int i;
```

```
    DWORD dwThreadID;

    ghWriteEvent = CreateEvent(
        NULL,               // default security attributes
        TRUE,               // manual-reset event
        FALSE,              // initial state is nonsignaled
        TEXT("WriteEvent")  // object name
        );

    if (ghWriteEvent == NULL)
    {
        printf("CreateEvent failed (%d)\n", GetLastError());
        return;
    }

    for(i = 0; i < THREADCOUNT; i++)
    {
        ghThreads[i] = CreateThread(
            NULL,              // default security
            0,                 // default stack size
            ThreadProc,        // name of the thread function
            NULL,              // no thread parameters
            0,                 // default startup flags
            &dwThreadID);

        if (ghThreads[i] == NULL)
        {
            printf("CreateThread failed (%d)\n", GetLastError());
            return;
        }
    }
}

void WriteToBuffer(VOID)
{
        printf("Main thread writing to the shared buffer...\n");
    if (! SetEvent(ghWriteEvent) )
    {
        printf("SetEvent failed (%d)\n", GetLastError());
        return;
    }
}

void CloseEvents()
{

    CloseHandle(ghWriteEvent);
}

int main( void )
{
    DWORD dwWaitResult;
    CreateEventsAndThreads();
```

```
    WriteToBuffer();

    printf("Main thread waiting for threads to exit...\n");
    dwWaitResult = WaitForMultipleObjects(
        THREADCOUNT,
        ghThreads,
        TRUE,
        INFINITE);

    switch (dwWaitResult)

        case WAIT_OBJECT_0:
            printf("All threads ended, cleaning up for application exit...\n");
            break;
        default:
            printf("WaitForMultipleObjects failed (%d)\n", GetLastError());
            return 1;
    }

    CloseEvents();

    return 0;
}

DWORD WINAPI ThreadProc(LPVOID lpParam)
{
    UNREFERENCED_PARAMETER(lpParam);

    DWORD dwWaitResult;

    printf("Thread %d waiting for write event...\n", GetCurrentThreadId());

    dwWaitResult = WaitForSingleObject(
        ghWriteEvent,
        INFINITE);

    switch (dwWaitResult)
    {
            case WAIT_OBJECT_0:
            printf("Thread %d reading from buffer\n",
                GetCurrentThreadId());
            break;
        default:
            printf("Wait error (%d)\n", GetLastError());
            return 0;
    }

    printf("Thread %d exiting\n", GetCurrentThreadId());
    return 1;
}
```

OUTPUT:



2. Semaphores

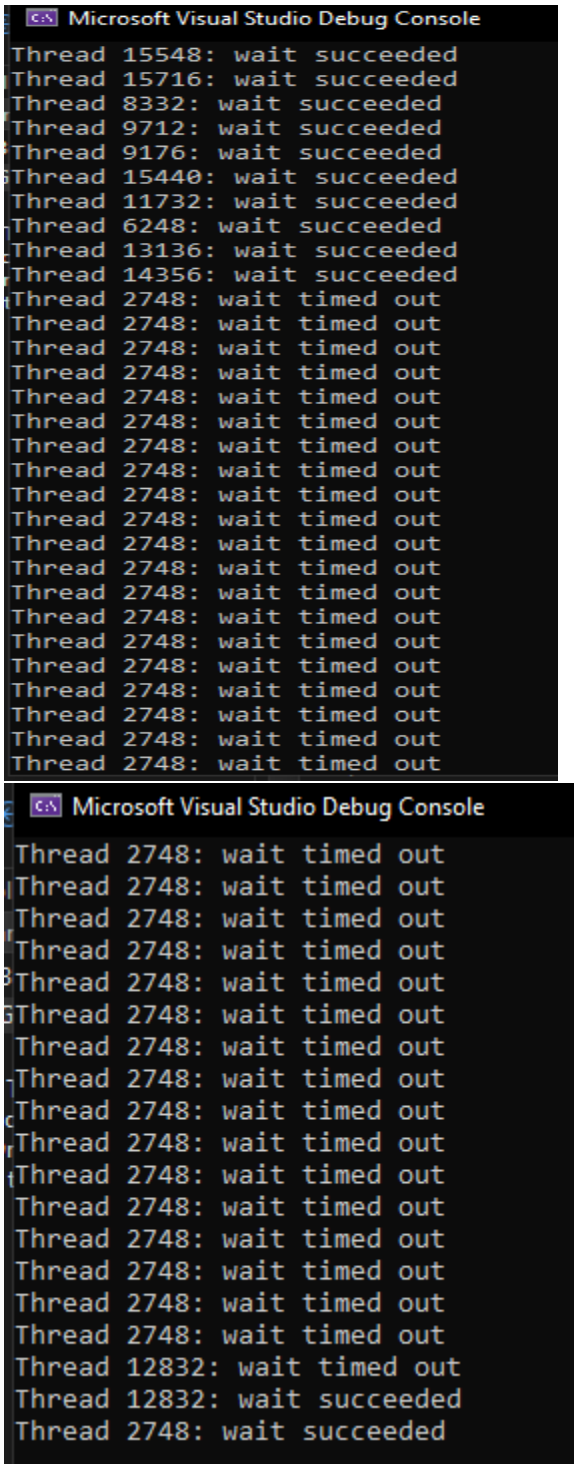
CODE:

```
#include <windows.h>
#include <stdio.h>

#define MAX_SEM_COUNT 10
#define THREADCOUNT 12

HANDLE ghSemaphore;

DWORD WINAPI ThreadProc(LPVOID);

int main(void)
{
    HANDLE aThread[THREADCOUNT];
    DWORD ThreadID;
    int i;

    // Create a semaphore with initial and max counts of MAX_SEM_COUNT

    ghSemaphore = CreateSemaphore(
        NULL,           // default security attributes
        MAX_SEM_COUNT,  // initial count
        MAX_SEM_COUNT,  // maximum count
        NULL);          // unnamed semaphore

    if (ghSemaphore == NULL)
    {
        printf("CreateSemaphore error: %d\n", GetLastError());
```

```
        return 1;
    }

    // Create worker threads

    for (i = 0; i < THREADCOUNT; i++)
    {
        aThread[i] = CreateThread(
            NULL,       // default security attributes
            0,          // default stack size
            (LPTHREAD_START_ROUTINE)ThreadProc,
            NULL,       // no thread function arguments
            0,          // default creation flags
            &ThreadID); // receive thread identifier

        if (aThread[i] == NULL)
        {
            printf("CreateThread error: %d\n", GetLastError());
            return 1;
        }
    }

    // Wait for all threads to terminate

    WaitForMultipleObjects(THREADCOUNT, aThread, TRUE, INFINITE);

    // Close thread and semaphore handles

    for (i = 0; i < THREADCOUNT; i++)
        CloseHandle(aThread[i]);

    CloseHandle(ghSemaphore);

    return 0;
}

DWORD WINAPI ThreadProc(LPVOID lpParam)
{

    // lpParam not used in this example
    UNREFERENCED_PARAMETER(lpParam);

    DWORD dwWaitResult;
    BOOL bContinue = TRUE;

    while (bContinue)
    {
        // Try to enter the semaphore gate.
```

```
    dwWaitResult = WaitForSingleObject(
        ghSemaphore,   // handle to semaphore
        0L);           // zero-second time-out interval

    switch (dwWaitResult)
    {
        // The semaphore object was signaled.
    case WAIT_OBJECT_0:
        // TODO: Perform task
        printf("Thread %d: wait succeeded\n", GetCurrentThreadId());
        bContinue = FALSE;

        // Simulate thread spending time on task
        Sleep(5);

        // Release the semaphore when task is finished

        if (!ReleaseSemaphore(
            ghSemaphore,  // handle to semaphore
            1,            // increase count by one
            NULL))       // not interested in previous count
        {
            printf("ReleaseSemaphore error: %d\n", GetLastError());
        }
        break;

        // The semaphore was nonsignaled, so a time-out occurred.
    case WAIT_TIMEOUT:
        printf("Thread %d: wait timed out\n", GetCurrentThreadId());
        break;
    }
}
return TRUE;
          }
```

OUTPUT:



3. Mutexes

CODE:

```
#include <windows.h>
#include <stdio.h>

#define THREADCOUNT 2
```

```
HANDLE ghMutex;

DWORD WINAPI WriteToDatabase(LPVOID);

int main(void)
{
    HANDLE aThread[THREADCOUNT];
    DWORD ThreadID;
    int i;

    // Create a mutex with no initial owner

    ghMutex = CreateMutex(
        NULL,              // default security attributes
        FALSE,             // initially not owned
        NULL);             // unnamed mutex

    if (ghMutex == NULL)
    {
        printf("CreateMutex error: %d\n", GetLastError());
        return 1;
    }

    // Create worker threads

    for (i = 0; i < THREADCOUNT; i++)
    {
        aThread[i] = CreateThread(
            NULL,       // default security attributes
            0,          // default stack size
            (LPTHREAD_START_ROUTINE)WriteToDatabase,
            NULL,       // no thread function arguments
            0,          // default creation flags
            &ThreadID); // receive thread identifier

        if (aThread[i] == NULL)
        {
            printf("CreateThread error: %d\n", GetLastError());
            return 1;
        }
    }

    // Wait for all threads to terminate

    WaitForMultipleObjects(THREADCOUNT, aThread, TRUE, INFINITE);

    // Close thread and mutex handles
```

```
    for (i = 0; i < THREADCOUNT; i++)
        CloseHandle(aThread[i]);

    CloseHandle(ghMutex);

    return 0;
}

DWORD WINAPI WriteToDatabase(LPVOID lpParam)
{
    // lpParam not used in this example
    UNREFERENCED_PARAMETER(lpParam);

    DWORD dwCount = 0, dwWaitResult;

    // Request ownership of mutex.

    while (dwCount < 20)
    {
        dwWaitResult = WaitForSingleObject(
            ghMutex,    // handle to mutex
            INFINITE);  // no time-out interval

        switch (dwWaitResult)
        {
            // The thread got ownership of the mutex
        case WAIT_OBJECT_0:
            __try {
                // TODO: Write to the database
                printf("Thread %d writing to database...\n",
                    GetCurrentThreadId());
                dwCount++;
            }

            __finally {
                // Release ownership of the mutex object
                if (!ReleaseMutex(ghMutex))
                {
                    // Handle error.
                }
            }
            break;

            // The thread got ownership of an abandoned mutex
            // The database is in an indeterminate state
        case WAIT_ABANDONED:
            return FALSE;
        }
    }
```

```
return TRUE;
        }
```

OUTPUT: