**Instructions:**

**BAHRIA UNIVERSITY (KARACHI CAMPUS**

)

OPEN ENDED LAB II

–

**Fall22**

**(**

System Programing (LAB)

**CS**

**C**

**-**

**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:**syed Muhammad saad bin shaukat** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Reg. No: **02-131202-074\_\_\_\_\_\_\_\_\_**



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**)

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

1. Events
2. Semaphores
3. Mutexes

**TASK 1:**

#include <string>

#include <iostream>

#include <thread>

#include <Windows.h>

using namespace std;

// The function we want to execute on the new thread.

void task() {

HANDLE hprcess = NULL;

HANDLE hthread = NULL;

STARTUPINFO si;

PROCESS\_INFORMATION pi;

DWORD dwProcessId = 0;

DWORD dwThreaId = 0;

ZeroMemory(&si, sizeof(si));

ZeroMemory(&si, sizeof(pi));

BOOL bCreateProcess = NULL;

BOOL nCreateProcess = NULL;

DWORD Ret = 0, dwPID = 0, dwTID = 0, dwPver = 0;

dwPID = GetCurrentProcessId();

cout << "Current PID" << dwPID << endl;

dwPID = GetCurrentThreadId();

cout << "Current TID" << dwPID << endl;

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

dwPver = GetProcessVersion(dwPID);

dwPID = GetCurrentThreadId();

cout << "Current PV" << dwTID << endl;

cout << "Starting child process\n" << endl;

bCreateProcess = CreateProcessW(

L"C:\\Program Files\\Windows NT\\Accessories\\wordpad.exe",

NULL,

NULL,

NULL,

FALSE,

0,

NULL,

NULL,

&si,

&pi

);

if (bCreateProcess == FALSE)

{

cout << "Failed";

}

else {

cout << "Process Creation Successful!";

cout << "PID" << pi.dwProcessId << endl;

cout << "TID" << pi.dwThreadId << endl;

}

nCreateProcess = CreateProcessW(

L"C:\\Windows\\System32\\notepad.exe",

NULL,

NULL,

NULL,

FALSE,

0,

NULL,

NULL,

&si,

&pi

);

if (nCreateProcess == FALSE)

{

cout << "Failed";

}

else {

cout << "Process Creation Successful!";

cout << "PID" << pi.dwProcessId << endl;

cout << "TID" << pi.dwThreadId << endl;

}

}

int main()

{

for (size\_t i = 0; i < 5; i++)

{

thread t1(task);

Sleep(1000);

t1.join();

if (i == 3)

{

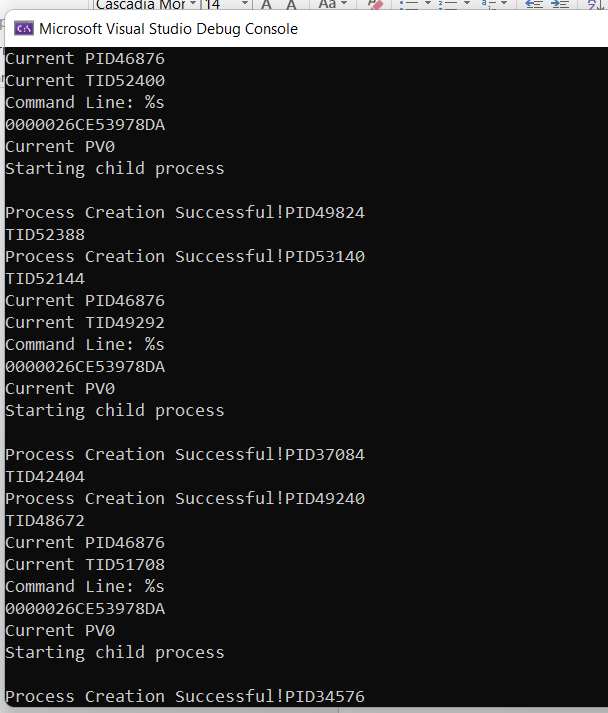
abort();

}

}

}

**OUTPUT:**

****

****

**TASK 2:**

**EVENT:**

#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,

TRUE,

FALSE,

TEXT("WriteEvent")

);

if (ghWriteEvent == NULL)

{

printf("CreateEvent failed (%d)\n", GetLastError());

return;

}

for (i = 0; i < THREADCOUNT; i++)

{

ghThreads[i] = CreateThread(

NULL,

0,

ThreadProc,

NULL,

0,

&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;

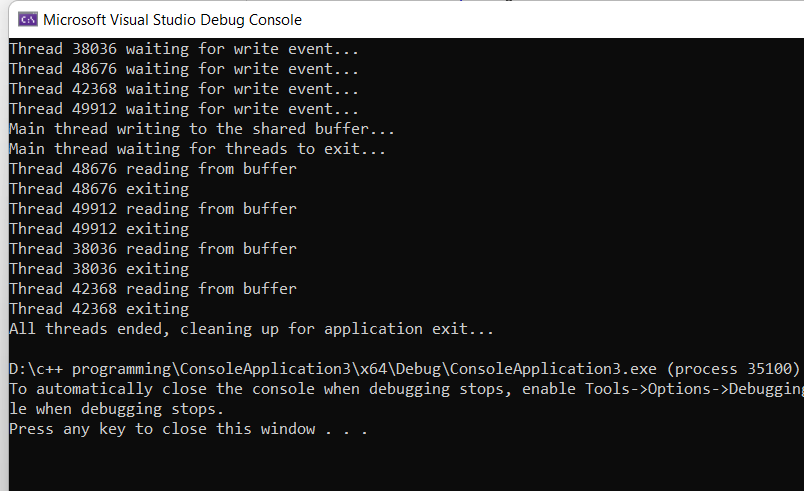
}

printf("Thread %d exiting\n", GetCurrentThreadId());

return 1;

}

**OUTPUT:**

****

**Semaphore:**

#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;

ghSemaphore = CreateSemaphore(

NULL,

MAX\_SEM\_COUNT,

MAX\_SEM\_COUNT,

NULL);

if (ghSemaphore == NULL)

{

printf("CreateSemaphore error: %d\n", GetLastError());

return 1;

}

// Create worker threads

for (i = 0; i < THREADCOUNT; i++)

{

aThread[i] = CreateThread(

NULL,

0,

(LPTHREAD\_START\_ROUTINE)ThreadProc,

NULL,

0,

&ThreadID);

if (aThread[i] == NULL)

{

printf("CreateThread error: %d\n", GetLastError());

return 1;

}

}

WaitForMultipleObjects(THREADCOUNT, aThread, TRUE, INFINITE);

for (i = 0; i < THREADCOUNT; i++)

CloseHandle(aThread[i]);

CloseHandle(ghSemaphore);

return 0;

}

DWORD WINAPI ThreadProc(LPVOID lpParam)

{

UNREFERENCED\_PARAMETER(lpParam);

DWORD dwWaitResult;

BOOL bContinue = TRUE;

while (bContinue)

{

dwWaitResult = WaitForSingleObject(

ghSemaphore,

0L);

switch (dwWaitResult)

{

case WAIT\_OBJECT\_0:

printf("Thread %d: wait succeeded\n", GetCurrentThreadId());

bContinue = FALSE;

Sleep(5);

if (!ReleaseSemaphore(

ghSemaphore,

1,

NULL))

{

printf("ReleaseSemaphore error: %d\n", GetLastError());

}

break;

case WAIT\_TIMEOUT:

printf("Thread %d: wait timed out\n", GetCurrentThreadId());

break;

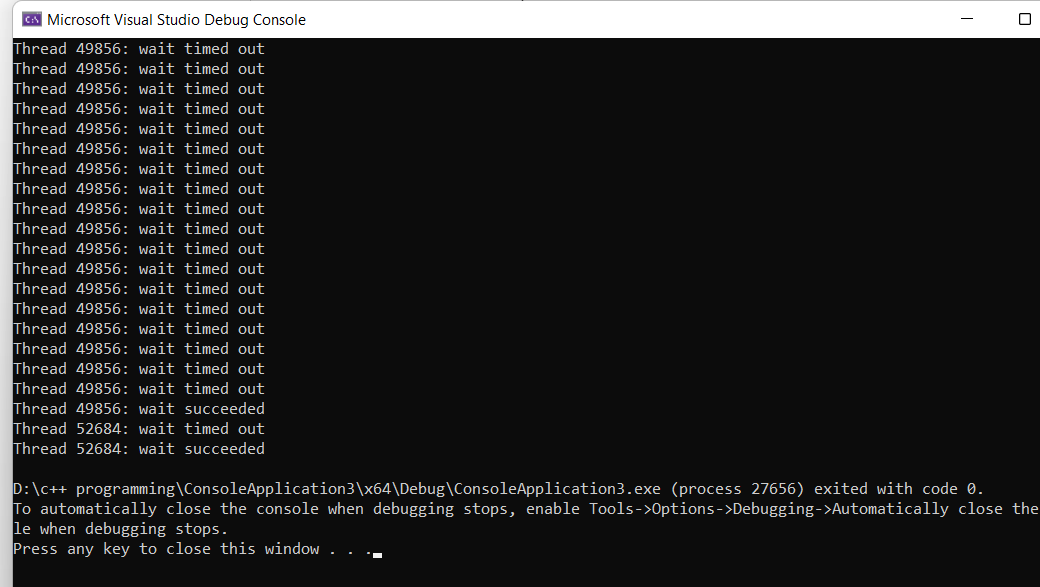
}

}

return TRUE;

}

**Output:**

****