**ResourcesApp.cpp Source Code File for Assignment Question 3**

#include <windows.h>

#include <tchar.h>

#include <strsafe.h>

#include <math.h>

#include <time.h>

#include <iostream>

#define MAX\_THREADS 16

#define BUF\_SIZE 255

DWORD WINAPI MyThreadFunction(LPVOID lpParam);

void ErrorHandler(LPTSTR lpszFunction);

// Sample custom data structure for threads to use.

// This is passed by void pointer so it can be any data type

// that can be passed using a single void pointer (LPVOID).

typedef struct MyData {

int val1;

int val2;

} MYDATA, \*PMYDATA;

int \_tmain()

{

unsigned int numberThreads;

std::cout << "This is the ResourcesApp for question 3: " << std::endl;

std::cout << "Enter the number of threads: ";

std::cin >> numberThreads;

if (numberThreads > MAX\_THREADS) numberThreads = 1;

PMYDATA pDataArray[MAX\_THREADS];

DWORD dwThreadIdArray[MAX\_THREADS];

HANDLE hThreadArray[MAX\_THREADS];

int t = (int)time(NULL);

// Create numberThreads worker threads.

for (unsigned int i = 0; i < numberThreads; i++)

{

// Allocate memory for thread data.

pDataArray[i] = (PMYDATA)HeapAlloc(GetProcessHeap(), HEAP\_ZERO\_MEMORY,

sizeof(MYDATA));

if (pDataArray[i] == NULL)

{

// If the array allocation fails, the system is out of memory

// so there is no point in trying to print an error message.

// Just terminate execution.

ExitProcess(2);

}

std::cout << "Thread " << i+1 << " started" << std::endl;

// Generate unique data for each thread to work with.

pDataArray[i]->val1 = i;

pDataArray[i]->val2 = i + 100+rand()%100;

// Create the thread to begin execution on its own.

hThreadArray[i] = CreateThread(

NULL, // default security attributes

0, // use default stack size

MyThreadFunction, // thread function name

pDataArray[i], // argument to thread function

0, // use default creation flags

&dwThreadIdArray[i]); // returns the thread identifier

// Check the return value for success.

// If CreateThread fails, terminate execution.

// This will automatically clean up threads and memory.

if (hThreadArray[i] == NULL)

{

ExitProcess(3);

}

} // End of main thread creation loop.

// Wait until all threads have terminated.

WaitForMultipleObjects(numberThreads, hThreadArray, TRUE, INFINITE);

std::cout << numberThreads << " threads completed in " << time(NULL)-t << " seconds" << std::endl;

// Close all thread handles and free memory allocations.

for (unsigned int i = 0; i < numberThreads; i++)

{

CloseHandle(hThreadArray[i]);

if (pDataArray[i] != NULL)

{

HeapFree(GetProcessHeap(), 0, pDataArray[i]);

pDataArray[i] = NULL; // Ensure address is not reused.

}

}

return 0;

}

DWORD WINAPI MyThreadFunction(LPVOID lpParam)

{

HANDLE hStdout;

PMYDATA pDataArray;

TCHAR msgBuf[BUF\_SIZE];

size\_t cchStringSize;

DWORD dwChars;

int t = time(NULL);

// Make sure there is a console to receive output results.

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

if (hStdout == INVALID\_HANDLE\_VALUE)

return 1;

pDataArray = (PMYDATA)lpParam;

// ------------- processing code --------------

double d;

for (int a = 0; a < 10; a++)

{

for (int b = 0; b < 100000000; b++) d = sqrt(a\*b);

}

// Print the time the thread took to complete the processing.

StringCchPrintf(msgBuf, BUF\_SIZE, TEXT("Thread #%d completed in %d s\n"),

pDataArray->val1+1, (time(NULL)-t));

StringCchLength(msgBuf, BUF\_SIZE, &cchStringSize);

WriteConsole(hStdout, msgBuf, (DWORD)cchStringSize, &dwChars, NULL);

return 0;

}