Университет ИТМО

Кафедра прикладной математики и информатики

Системное программное обеспечение

Лабораторная работа № 5

Выполнил:

Сорокин Юрий

Группа Р3217

Преподаватель:

Зыков А.Г.

СПб

2016 г.

# Лабораторная работа № 5

#include "stdafx.h"

#define THREADS\_COUNT 4

using namespace std::chrono;

CRITICAL\_SECTION critSect;

HANDLE SyncMutex;

HANDLE SyncSemaphore;

HANDLE SyncEvent;

HANDLE hThreads[THREADS\_COUNT];

int AskedValue = INT\_MAX;

volatile int ThreadsFinishedSort = 0;

size\_t ThreadsTimeStats[THREADS\_COUNT];

std::vector < int > data\_storage;

DWORD WINAPI Thread1\_Work ( CONST LPVOID lpParam )

{

size\_t ThreadID = GetThreadId ( hThreads[0] );

std::stringstream os;

os << "Thread with ID: " << ThreadID << " started." << std::endl;

std::cout << os.str ();

os.str ( std::string () );

os.clear ();

// Mutex

WaitForSingleObject ( SyncMutex, INFINITE ); // wait until SyncMutex is free

SyncMutex = CreateMutex ( NULL, TRUE, NULL );

os << "Thread with ID: " << ThreadID << " started sorting by insertion." << std::endl;

std::cout << os.str ();

os.str ( std::string () );

os.clear ();

auto start = high\_resolution\_clock::now ();

// Insertion sort

for (unsigned int uns = 1; uns < data\_storage.size(); ++uns)

{

int next = data\_storage[uns];

unsigned int idx;

for (idx = uns; idx > 0 && data\_storage[idx - 1] > next; --idx)

{

data\_storage[idx] = data\_storage[idx - 1];

}

data\_storage[idx] = next;

}

auto end = high\_resolution\_clock::now ();

os << "Thread with ID: " << ThreadID << " finished sorting." << std::endl;

std::cout << os.str();

os.str(std::string());

os.clear();

ReleaseMutex ( SyncMutex );

// Critical section.

// Increment can be non-atomic operation, sync. required.

EnterCriticalSection ( &critSect );

++ThreadsFinishedSort;

LeaveCriticalSection ( &critSect );

WaitForSingleObject ( SyncSemaphore, INFINITE );

bool ValueFound = std::binary\_search ( data\_storage.begin (), data\_storage.end (), AskedValue );

os << "Thread with ID: " << ThreadID << (ValueFound ? " found asked value." : " not found asked value.") << std::endl;

std::cout << os.str ();

os.str ( std::string () );

os.clear ();

ThreadsTimeStats[0] = duration\_cast<milliseconds>(end - start).count ();

return TRUE;

}

DWORD WINAPI Thread2\_Work ( CONST LPVOID lpParam )

{

size\_t ThreadID = GetThreadId ( hThreads[1] );

std::stringstream os;

os << "Thread with ID: " << ThreadID << " started." << std::endl;

std::cout << os.str ();

os.str ( std::string () );

os.clear ();

// Mutex

WaitForSingleObject ( SyncMutex, INFINITE ); // wait until SyncMutex is free

SyncMutex = CreateMutex ( NULL, TRUE, NULL );

os << "Thread with ID: " << ThreadID << " started sorting by bubble." << std::endl;

std::cout << os.str ();

os.str ( std::string () );

os.clear ();

auto start = high\_resolution\_clock::now ();

// Bubble sort

bool swapp = true;

while (swapp)

{

swapp = false;

for (size\_t i = 0; i < data\_storage.size () - 1; ++i)

{

if (data\_storage[i] > data\_storage[i + 1])

{

data\_storage[i] += data\_storage[i + 1];

data\_storage[i + 1] = data\_storage[i] - data\_storage[i + 1];

data\_storage[i] -= data\_storage[i + 1];

swapp = true;

}

}

}

auto end = high\_resolution\_clock::now ();

os << "Thread with ID: " << ThreadID << " finished sorting." << std::endl;

std::cout << os.str();

os.str(std::string());

os.clear();

ReleaseMutex ( SyncMutex );

// Critical section.

// Increment can be non-atomic operation, sync. required.

EnterCriticalSection ( &critSect );

++ThreadsFinishedSort;

LeaveCriticalSection ( &critSect );

WaitForSingleObject ( SyncSemaphore, INFINITE );

bool ValueFound = std::binary\_search ( data\_storage.begin (), data\_storage.end (), AskedValue );

os << "Thread with ID: " << ThreadID << (ValueFound ? " found asked value." : " not found asked value.") << std::endl;

std::cout << os.str ();

os.str ( std::string () );

os.clear ();

ThreadsTimeStats[1] = duration\_cast<milliseconds>(end - start).count ();

return TRUE;

}

DWORD WINAPI Thread3\_Work ( CONST LPVOID lpParam )

{

size\_t ThreadID = GetThreadId ( hThreads[2] );

std::stringstream os;

os << "Thread with ID: " << ThreadID << " started." << std::endl;

std::cout << os.str ();

os.str ( std::string () );

os.clear ();

// Mutex

WaitForSingleObject ( SyncMutex, INFINITE ); // wait until SyncMutex is free

SyncMutex = CreateMutex ( NULL, TRUE, NULL );

os << "Thread with ID: " << ThreadID << " started sorting by quick search." << std::endl;

std::cout << os.str ();

os.str ( std::string () );

os.clear ();

auto start = high\_resolution\_clock::now ();

// STL quicksort

std::sort ( data\_storage.begin (), data\_storage.end () );

auto end = high\_resolution\_clock::now ();

os << "Thread with ID: " << ThreadID << " finished sorting." << std::endl;

std::cout << os.str();

os.str(std::string());

os.clear();

ReleaseMutex ( SyncMutex );

// Critical section.

// Increment can be non-atomic operation, sync. required.

EnterCriticalSection ( &critSect );

++ThreadsFinishedSort;

LeaveCriticalSection ( &critSect );

WaitForSingleObject ( SyncSemaphore, INFINITE );

bool ValueFound = std::binary\_search ( data\_storage.begin (), data\_storage.end (), AskedValue );

os << "Thread with ID: " << ThreadID << (ValueFound ? " found asked value." : " not found asked value.") << std::endl;

std::cout << os.str ();

os.str ( std::string () );

os.clear ();

ThreadsTimeStats[2] = duration\_cast<milliseconds>(end - start).count ();

return TRUE;

}

DWORD WINAPI Thread4\_Work ( CONST LPVOID lpParam )

{

size\_t ThreadID = GetThreadId ( hThreads[3] );

std::stringstream os;

os << "Thread with ID: " << ThreadID << " started." << std::endl;

std::cout << os.str ();

os.str ( std::string () );

os.clear ();

WaitForSingleObject ( SyncEvent, INFINITE );

os << "Thread with ID: " << ThreadID << " prints program execution results." << std::endl;

std::cout << os.str ();

os.str ( std::string () );

os.clear ();

for (size\_t i = 0; i < THREADS\_COUNT-1; ++i)

{

std::cout << "Thread #" << i + 1

<< " finished sorting work in " << ThreadsTimeStats[i] << " milliseconds." << std::endl;

}

return TRUE;

}

int main(int argc, char \* argv[])

{

if (argc != 5)

{

std::cout << "Wrong usage pattern.\n \tSynopsys: -file %FILENAME% -search\_for %VALUE" << std::endl;

exit ( 10 );

}

std::string filename;

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

{

if ( !strcmp ( argv[i], "-file" ) && argc > i )

{

filename = argv[i + 1];

}

if (!strcmp ( argv[i], "-search\_for" ) && argc > i)

{

AskedValue = atoi ( argv[i + 1] );

}

}

InitializeCriticalSection ( &critSect );

SyncSemaphore = CreateSemaphore ( NULL, THREADS\_COUNT, THREADS\_COUNT, NULL );

SyncEvent = CreateEvent ( NULL, TRUE, FALSE, NULL );

std::ifstream input ( filename );

int tmp;

while ( !input.eof () && input >> tmp )

{

data\_storage.push\_back ( tmp );

}

// Threads start, sync work is done inside of them.

hThreads[0] = CreateThread ( NULL, 0, &Thread1\_Work, NULL, 0, NULL );

hThreads[1] = CreateThread ( NULL, 0, &Thread2\_Work, NULL, 0, NULL );

hThreads[2] = CreateThread ( NULL, 0, &Thread3\_Work, NULL, 0, NULL );

hThreads[3] = CreateThread ( NULL, 0, &Thread4\_Work, NULL, 0, NULL );

WaitForMultipleObjects ( THREADS\_COUNT - 1, hThreads, TRUE, INFINITE );

SetEvent ( SyncEvent );

WaitForSingleObject ( hThreads[3], INFINITE ); // printstats thread.

CloseHandle ( SyncSemaphore );

CloseHandle ( SyncMutex );

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

{

CloseHandle ( hThreads[i] );

}

system ( "pause" );

}}

**Вывод**

При выполнении данной работы я ознакомился с работой с потоками, а именно с созданием и синхронизацией потоков разными способами: критическая секция, мьютекс, семафор, событие.