**ИТМО Кафедра Информатики и прикладной математики**

Отчет по лабораторной работе №5

«Объекты синхронизации»

**Выполнил: студент группы P3217**

**Плюхин Дмитрий**

**Преподаватель: Зыков А. Г.**

**2017 год**

1. **Задание**

Исследование на конкретном примере следующих объектов синхронизации:

1. критические секции

2. мьютексы

3. семафоры

4. события

Задачу для синхронизации выбрать на свое усмотрение.

Задачи для каждого метода синхронизации должны быть различными. Задачи должны наглядно демонстрировать выбранный метод синхронизации и учитывать его особенности. Студент, сдающий работу должен аргументированно обосновать задачу, выбранную для синхронизации и метод синхронизации.

1. **Задачи, выбранные для синхронизации**

В качестве задачи для синхронизации была выбрана работа с файлом – синхронизированная сортировка, запись и поиск в файле. Далее приводится краткое описание задач.

**2.1 Мьютекс**

Подзадачей использования мьютексов в данном случае является исключение одновременной сортировки и записи в файл. Это реализуется при помощи создания единого мьютекса для некоторого файла, который используется в приложении, сортирующем файл и в приложении, выполняющем запись в файл по запросу пользователя. Без использования мьютекса попытка записи в сортируемый файл не завершается успехом, поскольку наиболее вероятно добавление информации во время сортировки, однако после окончания сортировки содержимое файла все равно переписывается. Необходимо дождаться завершения сортировки – без синхронизации тут не обойтись.

**2.2 Семафор**

Для использования семафора была представлена гипотетическая ситуация, в которой требуется ограничить количество потоков, одновременно осуществляющих поиск в файле (например, во избежание перегрузки разделяемого ресурса). Демонстрируется наиболее простая ситуация – одновременно поиск в файле могут осуществлять два потока, остальные «встают в очередь» и дожидаются разрешения семафора начать работу с файлом.

**2.3 Событие синхронизации**

Использование событий синхронизации как нельзя лучше демонстрируется на многопоточной сортировке – главный поток порождает дочерние и должен дождаться их завершения перед тем, как продолжить работу – иначе он будет иметь дело с еще не отсортированными кусками данных, как результат – на выходе файл будет сформирован хаотическим образом.

**2.4 Критическая секция**

Использование критической секции демонстрируется в контексте многопоточной сортировки, синхронизируемой событиями, более конкретно, критические секции позволяют осуществить корректный вывод на экран информации о запускающихся потоках – в случае пренебрежения критическими секциями в консоли появляется нелицеприятная последовательность символа, не имеющая смысла.

1. **Листинг основной части программы**

**//Файл main.cpp**

#include <fstream>

#include <windows.h>

#include <iostream>

#include <process.h>

#include <stdlib.h>

#include <time.h>

#include <string>

#include <sstream>

#include <array>

#include <stdio.h>

using namespace std;

#define MAX\_LENGTH 256;

CRITICAL\_SECTION criticalSection;

typedef struct \_ThreadInfo

{

HANDLE hFinishEvent;

char\*\* a;

char\*\* b;

int low;

int high;

} ThreadInfo, \*ThreadInfoPointer;

int showThreadInfo(ThreadInfo threadInfo){

cout << "Thread :" << endl;

cout << " low - " << threadInfo.low << endl;

cout << " hight - " << threadInfo.high << endl;

return 0;

}

HANDLE getFileMutex(char\* fileName){

char fileMutexName[256] = "fileMutex";

strcat(fileMutexName,fileName);

cout << "get mut" << endl;

HANDLE hMutex = OpenMutex(SYNCHRONIZE, FALSE, fileMutexName);

cout << "got mut" << endl;

if (hMutex == NULL){

hMutex = CreateMutex(NULL, FALSE, fileMutexName);

}

return hMutex;

}

unsigned \_\_stdcall mergesort(void\* threadArg) {

ThreadInfoPointer threadInfo = (ThreadInfoPointer)threadArg;

if ((threadInfo->low) < (threadInfo->high))

{

HANDLE hEvents[2];

hEvents[0] = CreateEvent(NULL, TRUE, FALSE, NULL);

hEvents[1] = CreateEvent(NULL, TRUE, FALSE, NULL);

EnterCriticalSection(&criticalSection);

//showThreadInfo(\*threadInfo);

LeaveCriticalSection(&criticalSection);

int ridge = ((threadInfo->low)+(threadInfo->high))/2;

ThreadInfo firstSubThreadInfo;

ThreadInfo secondSubThreadInfo;

firstSubThreadInfo.a = threadInfo->a;

firstSubThreadInfo.b = threadInfo->b;

firstSubThreadInfo.low = threadInfo->low;

firstSubThreadInfo.high = ridge;

firstSubThreadInfo.hFinishEvent = hEvents[0];

secondSubThreadInfo.a=threadInfo->a;

secondSubThreadInfo.b=threadInfo->b;

secondSubThreadInfo.low=ridge+1;

secondSubThreadInfo.high=threadInfo->high;

secondSubThreadInfo.hFinishEvent = hEvents[1];

EnterCriticalSection(&criticalSection);

//cout << "Creating thread with " << firstSubThreadInfo.low << " and " << firstSubThreadInfo.high << endl;

LeaveCriticalSection(&criticalSection);

\_beginthreadex(NULL, 0, mergesort, &firstSubThreadInfo, 0, 0);

\_beginthreadex(NULL, 0, mergesort, &secondSubThreadInfo, 0, 0);

WaitForMultipleObjects(2,hEvents,true,INFINITE);

EnterCriticalSection(&criticalSection);

//cout << "Subthreads finished" << endl;

LeaveCriticalSection(&criticalSection);

int h,i,j,k,lowBound, highBound;

lowBound = threadInfo->low;

highBound = threadInfo->high;

h=lowBound;

i=lowBound;

j=ridge+1;

while((h <= ridge) && (j <= highBound))

{

if(strcmp(threadInfo->a[h],threadInfo->a[j]) <= 0)

{

threadInfo->b[i] = threadInfo->a[h];

h++;

} else {

threadInfo->b[i] = threadInfo->a[j];

j++;

}

i++;

}

if(h > ridge)

{

for (k = j; k <= highBound; k++)

{

threadInfo->b[i]=threadInfo->a[k];

i++;

}

} else {

for ( k = h; k <= ridge; k++)

{

threadInfo->b[i]=threadInfo->a[k];

i++;

}

}

for(k = lowBound; k <= highBound; k++) threadInfo->a[k]=threadInfo->b[k];

}

if (threadInfo->hFinishEvent != INVALID\_HANDLE\_VALUE) SetEvent(threadInfo->hFinishEvent);

return 0;

}

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

{

char identifier[256];

DWORD dwWaitResult;

HANDLE fileMutex = getFileMutex(argv[1]);

while (true){

dwWaitResult = WaitForSingleObject(fileMutex,INFINITE);

ifstream identifiers(argv[1]);

if (!identifiers){

return;

}

int numOfElements = 0;

while (identifiers){

identifiers.getline(identifier, 256);

//cout << identifier << endl;

numOfElements++;

}

identifiers.close();

cout << numOfElements << endl;

char\*\* idents = (char\*\*)malloc(numOfElements\*sizeof(char\*));

char\*\* identstmp = (char\*\*)malloc(numOfElements\*sizeof(char\*));

ifstream identifierss(argv[1]);

if (!identifierss){

return;

}

int index = 0;

while (identifierss){

idents[index] = (char\*)malloc(256\*sizeof(char\*));

identifierss.getline(idents[index], 256);

//cout << idents[index] << endl;

//cout << identifier << endl;

index++;

}

identifierss.close();

InitializeCriticalSection(&criticalSection);

ThreadInfo threadInfo;

int inputArr[12] = {12,10,43,23,-78,45,123,56,98,41,90,24};

int copiedArr[12]={0,0,0,0,0,0,0,0,0,0,0,0};

threadInfo.a=idents;

threadInfo.b=identstmp;

//for(int i=0; i<numOfElements; i++) cout<<threadInfo.a[i]<<" ";

//cout<<endl;

HANDLE hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

threadInfo.low = 0;

threadInfo.high = numOfElements - 1;

threadInfo.hFinishEvent = hEvent;

\_beginthreadex(NULL, 0, mergesort, &threadInfo, 0, 0);

WaitForSingleObject(hEvent, 5000);

//cout << "oss" << endl;

cout<<"After run the result is "<<endl;

for(int i=0; i<numOfElements; i++) cout<<threadInfo.a[i]<<" ";

cout<<endl;

ofstream ofs (argv[1], std::ofstream::out);

for(int i=0; i<numOfElements; i++) ofs<<threadInfo.a[i]<<"\n";

ofs.close();

cout << "Releasing mutex" << endl;

ReleaseMutex(fileMutex);

cout << "Mutex released" << endl;

}

}

**//Файл writer.cpp**

#include <fstream>

#include <windows.h>

#include <iostream>

#include <process.h>

#include <stdlib.h>

#include <time.h>

#include <string>

#include <sstream>

#include <array>

#include <stdio.h>

using namespace std;

HANDLE getFileMutex(char\* fileName){

char fileMutexName[256] = "fileMutex";

strcat(fileMutexName,fileName);

cout << "get mut" << endl;

HANDLE hMutex = OpenMutex(SYNCHRONIZE, FALSE, fileMutexName);

cout << "got mut" << endl;

if (hMutex == NULL){

hMutex = CreateMutex(NULL, FALSE, fileMutexName);

}

return hMutex;

}

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

{

HANDLE fileMutex = getFileMutex(argv[1]);

DWORD dwWaitResult;

string command = "";

while(true){

cout << "Type identifier to write to file (type exit to stop) : " << endl;

getline(cin, command);

if (command.compare("exit") == 0){

break;

}

dwWaitResult = WaitForSingleObject(fileMutex,INFINITE);

ofstream ofs (argv[1], std::ofstream::app);

ofs<<command<<"\n";

ofs.close();

cout << "Releasing mutex" << endl;

ReleaseMutex(fileMutex);

cout << "Mutex released" << endl;

}

}

**//Файл searcher.cpp**

#include <fstream>

#include <windows.h>

#include <iostream>

#include <process.h>

#include <stdlib.h>

#include <time.h>

#include <string>

#include <sstream>

#include <array>

#include <stdio.h>

using namespace std;

#define MAX\_LENGTH 256;

#define MAX\_SEM\_COUNT 2

CRITICAL\_SECTION criticalSection;

typedef struct \_ThreadInfo

{

char\* fileName;

char\* identifier;

int delay;

HANDLE hSemaphore;

} ThreadInfo, \*ThreadInfoPointer;

HANDLE getSemaphore(char\* fileName){

char semaphoreName[256] = "Semaphore";

strcat(semaphoreName,fileName);

HANDLE hSemaphore = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, semaphoreName);

if (hSemaphore == NULL){

hSemaphore = CreateSemaphore(NULL, MAX\_SEM\_COUNT, MAX\_SEM\_COUNT, semaphoreName);

}

return hSemaphore;

}

unsigned \_\_stdcall find(void\* threadArg) {

ThreadInfoPointer threadInfo = (ThreadInfoPointer)threadArg;

HANDLE hSemaphore = threadInfo->hSemaphore;

cout << "Waiting for semaphore..." << endl;

WaitForSingleObject(hSemaphore, INFINITE);

cout << "search started for identifier " << threadInfo->identifier << endl;

char identifier[256];

DWORD dwWaitResult;

//HANDLE fileMutex = getFileMutex(argv[1]);

//dwWaitResult = WaitForSingleObject(fileMutex,INFINITE);

ifstream identifiers(threadInfo->fileName);

if (!identifiers){

cout << "error" << endl;

return 1;

}

int index = 0;

while (identifiers){

identifiers.getline(identifier, 256);

//cout << "-<" << threadInfo->identifier << ">-" << endl;

//cout << "-<" << identifier << ">-" << endl;

if (strcmp(identifier,threadInfo->identifier) == 0){

cout << "identifier \'" << threadInfo->identifier << "\' " << "found at the position " << index << endl;

ReleaseSemaphore(hSemaphore, 1, NULL);

return 0;

}

index++;

Sleep(threadInfo->delay);

//cout << ".";

}

identifiers.close();

cout << "Wasn't found" << endl;

ReleaseSemaphore(hSemaphore, 1, NULL);

return 1;

}

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

{

DWORD dwWaitResult;

HANDLE hSemaphore = getSemaphore(argv[1]);

char identifier[256];

string command = "";

while(true){

cout << "Type identifier to search in file (type exit to stop) : " << endl;

getline(cin, command);

if (command.compare("exit") == 0){

break;

}

ThreadInfo\* threadInfo = (ThreadInfo\*)malloc(sizeof(ThreadInfo));

threadInfo->fileName = argv[1];

threadInfo->identifier = (char\*)malloc(256\*sizeof(char));

strcpy(threadInfo->identifier, command.c\_str());

threadInfo->delay = 100;

threadInfo->hSemaphore = hSemaphore;

\_beginthreadex(NULL, 0, find, threadInfo, 0, 0);

}

}

1. **Вывод**

Таким образом, в данной лабораторной работе были применены на практике знания об объектах синхронизации, области применения каждого из них и предоставляемых преимуществах. Безусловно, приведенная версия программы не может найти практического применения в том состоянии, в котором находится на момент сдачи лабораторной работы, однако впоследствии может быть значительно расширена при необходимости, на данный же момент она является хорошим наглядным примером использования объектов синхронизации, демонстрируя те случаи, в которых без них просто не обойтись.