**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. **Код основной части программы**

В качестве задачи для синхронизации была выбрана разработка простейшей базы данных. Далее приводится описание всех случаев использования каждого объекта синхронизации.

**2.1 Мьютекс**

Подзадачей использования мьютексов в данном случае является чтение и запись данных зарегистрировавшегося пользователя в «базу данных», представленную текстовым файлом, содержащим логины и пароли. Поскольку мьютекс является именованным объектом синхронизации, что обеспечивает возможность его использования в разных процессах, а также позволяет допустить до единовременной работы с разделяемым ресурсом только один поток исполнения, его использование в данном контексте является наиболее практичным решением задачи синхронизации. Далее приведены фрагменты кода, связанные непосредственно с использованием мьютекса.

HANDLE getFileMutex(){

HANDLE hMutex = OpenMutex(SYNCHRONIZE, FALSE, "fileMutex");

if (hMutex == NULL){

hMutex = CreateMutex(NULL, FALSE, "fileMutex");

}

return hMutex;

}

int writeUser(HANDLE fileMutex, const char\* fileName, string login, string password)

{

DWORD dwWaitResult = WaitForSingleObject(fileMutex,INFINITE);

ofstream fout(fileName, ios\_base::app);

fout <<"$"<<login<<"/"<<password;

fout.close();

ReleaseMutex(fileMutex);

return 0;

}

int admitUser(int data)

{

HANDLE fileMutex = getFileMutex();

string login = getData("Please, enter your login and press Enter : ", "$/", 32);

string password = getData("Please, enter your password and press Enter : ", "$/", 32);

if (checkUser(fileMutex,"users.txt",login,password) == 0){

ReleaseMutex(fileMutex);

CloseHandle(fileMutex);

passToDataBase(data);

} else {

ReleaseMutex(fileMutex);

CloseHandle(fileMutex);

cout << "Invalid login or password" << endl;

}

return 0;

}

int checkUser(HANDLE fileMutex, const char\* fileName, string login, string password)

{

string rLogin;

string rPassword;

DWORD dwWaitResult = WaitForSingleObject(fileMutex,INFINITE);

char buff[50];

ifstream fin(fileName);

int stop = 2;

int del = 1;

int index = 0;

char ch;

while ((stop!=0) && fin.get(ch)) {

//cout << ch << endl;

if (ch == '$'){

stop--;

if (stop == 0){

stop = 1;

del = 1;

//cout << "login : " << rLogin << endl;

//cout << "password : " << rPassword << endl;

//cout << "-" << rLogin << "-" << rPassword << "-" << endl;

if ((login.compare(rLogin) == 0) && (password.compare(rPassword) == 0)) return 0;

//cout << "login is " << login.compare(rLogin) << "password is " << password.compare(rPassword) << endl;

rLogin.clear();

rPassword.clear();

}

continue;

}

if (ch == '/'){

del--;

continue;

}

if ((stop == 1) && (del == 1)){

rLogin.append(1, ch);

continue;

}

if ((stop == 1) && (del == 0) && (ch != '\n')){

rPassword.append(1, ch);

continue;

}

}

//cout << "login : " << rLogin << endl;

//cout << "password : " << rPassword << endl;

//cout << "-" << rLogin << "-" << rPassword << "-" << endl;

//cout << "login is " << login.compare(rLogin) << "password is " << password.compare(rPassword) << endl;

if ((login.compare(rLogin) == 0) && (password.compare(rPassword) == 0)) return 0;

fin.close();

return 1;

}

**2.2 Семафор**

Для использования семафора была представлена ситуация, при которой могло бы стать необходимым ограничение числа пользователей, которые могут иметь доступ к БД одновременно. В данном случае имеем дело с задачей, в рамках которой требуется позволить выполнение программы только для определенного количества потоков исполнения, остальные же должны «встать в очередь» и ждать, пока кто-нибудь из игроков не покинет сервер. Трудно представить в данном случае более элегантное решение, чем введение семафора.

HANDLE getDataBaseSemaphore()

{

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

if (hSemaphore == NULL)

{

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

}

return hSemaphore;

}

int passToDataBase(int championship)

{

HANDLE hSemaphore = getDataBaseSemaphore();

HANDLE hEvent = NULL;

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

DWORD dwWaitResult = WaitForSingleObject(hSemaphore, INFINITE);

cout << "Welcome to the DataBase!" << endl;

if (championship == 1){

hEvent = getSharingStartEvent(0);

cout << "Waiting for starting sharing..." << endl;

WaitForSingleObject(hEvent, INFINITE);

}

string word;

string computerNum;

srand(time(NULL));

while (true)

{

if ((championship == 1) && (WaitForSingleObject(hEvent,0) != WAIT\_OBJECT\_0)){

cout << "Sharing ended." << endl;

CloseHandle(hEvent);

break;

}

cout << "Type the word or num to share or type \\exit" << endl;

getline(cin, word);

if (word.compare("\\exit") == 0)

{

cout << "Thank you for using our DataBase!" << endl;

break;

}

SaveData(word);

computerNum = rand() % 1000;

cout << "You've shared " << word << " with me and I've shared "

<< computerNum << " with you. We're the best team'" << endl;

}

ReleaseSemaphore(hSemaphore, 1, NULL);

CloseHandle(hSemaphore);

return 0;

}

**2.3 Событие синхронизации**

Для использования данного объекта было создано допущение, что все пользователи должны закончить работу в одно время. Для подобного арбитра как нельзя лучше подходит объект события синхронизации, который может быть либо в сигнальном состоянии, либо в несигнальном.

HANDLE getSharingStartEvent(int access)

{

HANDLE hEvent;

if (access == 1)

{

hEvent = OpenEvent(EVENT\_ALL\_ACCESS, FALSE, "StartShare");

}

else

{

hEvent = OpenEvent(SYNCHRONIZE, FALSE, "StartShare");

}

if (hEvent == NULL)

{

cout << "creating event" << endl;

hEvent = CreateEvent(NULL, TRUE, FALSE, "StartShare");

}

return hEvent;

}

int StartShare()

{

string password;

HANDLE hEvent = getSharingStartEvent(1);

if (WaitForSingleObject(hEvent,0) == WAIT\_OBJECT\_0){

cout << "Sharing already started" << endl;

return 0;

}

do{

cout << "Please, type password to start sharing: " << endl;

getline(cin, password);

} while (password.compare("hexademical") != 0);

if (WaitForSingleObject(hEvent,0) == WAIT\_OBJECT\_0){

cout << "Sharing already started" << endl;

return 0;

}

cout << "before" << endl;

//ResetEvent(hEvent);

SetEvent(hEvent);

cout << "after" << endl;

if (WaitForSingleObject(hEvent,0) == WAIT\_OBJECT\_0){

cout << "Sharing have started !" << endl;

return 0;

}

CloseHandle(hEvent);

return 0;

}

int StopShare()

{

string password;

HANDLE hEvent = getSharingStartEvent(1);

if (WaitForSingleObject(hEvent,0) != WAIT\_OBJECT\_0){

cout << "Sharing already stopped" << endl;

return 0;

}

do{

cout << "Please, type password to stop sharing : " << endl;

getline(cin, password);

} while (password.compare("hexademical") != 0);

if (WaitForSingleObject(hEvent,0) != WAIT\_OBJECT\_0){

cout << "Sharing already stopped" << endl;

return 0;

}

cout << "before" << endl;

//SetEvent(hEvent);

ResetEvent(hEvent);

cout << "after" << endl;

if (WaitForSingleObject(hEvent,0) != WAIT\_OBJECT\_0){

cout << "Sharing have stopped !" << endl;

return 0;

}

CloseHandle(hEvent);

return 0;

}

int passToDataBase(int championship)

{

HANDLE hSemaphore = getDataBaseSemaphore();

HANDLE hEvent = NULL;

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

DWORD dwWaitResult = WaitForSingleObject(hSemaphore, INFINITE);

cout << "Welcome to the DataBase!" << endl;

if (championship == 1){

hEvent = getSharingStartEvent(0);

cout << "Waiting for starting sharing..." << endl;

WaitForSingleObject(hEvent, INFINITE);

}

string word;

string computerNum;

srand(time(NULL));

while (true)

{

if ((championship == 1) && (WaitForSingleObject(hEvent,0) != WAIT\_OBJECT\_0)){

cout << "Sharing ended." << endl;

CloseHandle(hEvent);

break;

}

cout << "Type the word or num to share or type \\exit" << endl;

getline(cin, word);

if (word.compare("\\exit") == 0)

{

cout << "Thank you for using our DataBase!" << endl;

break;

}

SaveData(word);

computerNum = rand() % 1000;

cout << "You've shared " << word << " with me and I've shared "

<< computerNum << " with you. We're the best team'" << endl;

}

ReleaseSemaphore(hSemaphore, 1, NULL);

CloseHandle(hSemaphore);

return 0;

}

**2.4 Критическая секция**

Помимо всего прочего, для данного случая была разработана модификация базы данных: два пользователя могут получать доступ к одним и тем же данным. Это можно наиболее естественно реализовать с использованием объекта критической секции.

void SaveData(std::string data)

{

std::cout << "Saving data..." << std::endl;

EnterCriticalSection(&criticalSection);

std::cout << "Saved!" << std::endl;

LeaveCriticalSection(&criticalSection);

}

int main()

{

InitializeCriticalSection(&criticalSection);

cout << "Welcome to the DataBase!" << endl;

string word = "";

while(true)

{

system("cls");

cout << "Menu : \n - Login \n - SignIn \n - SignInSharing \n - StartShare \n - StopShare \n - Exit" << endl;

getline(cin, word);

if (word.compare("SignIn") == 0)

{

admitUser(0);

}

else if (word.compare("SignInSharing") == 0)

{

admitUser(1);

}

else if (word.compare("StartShare") == 0)

{

StartShare();

}

else if (word.compare("StopShare") == 0)

{

StopShare();

}

else if (word.compare("Login") == 0)

{

registerUser();

}

else if (word.compare("Exit") == 0)

{

cout << "Good buy!";

return 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.