САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

КАФЕДРА ИНФОРМАТИКИ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Отчет по лабораторной работе №4
по курсу «Операционные системы»

Вариант 7

Выполнил:
студент гр. P3318
Петкевич Константин Вячеславович
Преподаватель:
Лаздин А.В.

2017

Задание

Написать программы для консольного процесса **Administrator** и консольных процессов **Reader и Writer**.

Для моделирования передачи сообщений ввести специальные события, которые обозначают сообщение “A” , сообщение “B”, и конец сеанса для процессов **Reader и Writer**.

Одновременно принимать и отправлять сообщения могут **только два процесса Writer** и **два процесса Reader**, передача остальных сообщений от других процессов должна блокироваться с помощью мьютексов;

Процесс **Administrator**:

запрашивает у пользователя количество процессов **Reader и Writer**, которые он должен запустить;
запрашивает у пользователя кол-во отправленных сообщений для процесса **Writer**. Кол-во принятых сообщений для процесса **Reader** вычислить**. (соответствие сообщений проверить и подкорректировать по формуле)**;
запускает заданное количество процессов **Reader и Writer**;
принимает от каждого процесса **Reader и Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке.
завершает свою работу.

Процесс **Writer**:

запрашивает с консоли сообщения, и передает их (по одному) процессу **Reader**; передает сообщение о завершении сеанса процессу **Administrator**;
завершает свою работу.

Процесс **Reader**:

принимает сообщение от процесса **Writer**; выводит на консоль сообщение;
передает сообщение о завершении сеанса процессу **Administrator**; завершает свою работу.

Листинг программ

Administrator.cpp

```
#include <windows.h>
#include <iostream>
#include <string>
using namespace std;

int main()
{
	HANDLE hMsgA, hMsgB, hReaderEnd, hWriterEnd;
	HANDLE hMutexR1, hMutexR2, hMutexW1, hMutexW2;


	int pCount, msgCount;
	cout << "Enter counts of \"Readers\" and \"Writers\": ";
	cin >> pCount;
	cout << "Enter count of messages for sending: ";
	cin >> msgCount;


	hMutexR1 = CreateMutex(NULL, FALSE, L"MutexR1");
	hMutexR2 = CreateMutex(NULL, FALSE, L"MutexR2");
```

```
	hMutexW1 = CreateMutex(NULL, FALSE, L"MutexW1");
	hMutexW2 = CreateMutex(NULL, FALSE, L"MutexW2");

	hMsgA = CreateEvent(NULL, FALSE, FALSE, L"MsgA");
	hMsgB = CreateEvent(NULL, FALSE, FALSE, L"MsgB");

	hReaderEnd = CreateEvent(NULL, FALSE, FALSE, L"ReaderEnd");
	hWriterEnd = CreateEvent(NULL, FALSE, FALSE, L"WriterEnd");


	TCHAR msgCountStr[8];
	_itow(msgCount, msgCountStr, 10);

	for (int i = 0; i < pCount; i++)
	{
		STARTUPINFO si1, si2;
		PROCESS_INFORMATION pi1, pi2;
		ZeroMemory(&si1, sizeof(STARTUPINFO));
		si1.cb = sizeof(STARTUPINFO);
		ZeroMemory(&si2, sizeof(STARTUPINFO));
		si2.cb = sizeof(STARTUPINFO);

		if (!CreateProcess(L"Reader.exe", msgCountStr, NULL, NULL, FALSE,
CREATE_NEW_CONSOLE, NULL, NULL, &si2, &pi2))
		{
			cout << "Can't start Reader.exe" << endl;
			return GetLastError();
		}

		if (!CreateProcess(L"Writer.exe", msgCountStr, NULL, NULL, FALSE,
CREATE_NEW_CONSOLE, NULL, NULL, &si1, &pi1))
		{
			cout << "Can't start Writer.exe" << endl;
			return GetLastError();
		}
	}

	HANDLE hEnds[2];
	hEnds[0] = hReaderEnd;
	hEnds[1] = hWriterEnd;
	for (int i = 0; i < pCount; i++)
	{
		WaitForMultipleObjects(2, hEnds, true, INFINITE);
		printf_s("Session %d of Writer and Reader finished\n", i);
	}

	CloseHandle(hMutexR1);
	CloseHandle(hMutexR2);
	CloseHandle(hMutexW1);
	CloseHandle(hMutexW2);

	CloseHandle(hReaderEnd);
	CloseHandle(hWriterEnd);

	CloseHandle(hMsgA);
	CloseHandle(hMsgB);
```

```
	cout << "The End!" << endl;
	cin >> pCount;

	return 0;
}
```

Reader.cpp

```
#include <windows.h>
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{
	HANDLE hMutex[2];
	HANDLE hMsg, hEnd;

	hMutex[0] = OpenMutex(SYNCHRONIZE, false, L"MutexR1");
	hMutex[1] = OpenMutex(SYNCHRONIZE, false, L"MutexR2");

	if (hMutex[0] == NULL || hMutex[1] == NULL)
	{
		cout << "Can't open Mutex" << endl;
		Sleep(10000);
		return GetLastError();
	}

	FILE* fp;
	char filename[64];

	cout << "Waiting for ending other Readers..." << endl;
	while (true)
	{
		if (WaitForSingleObject(hMutex[0], 1000) == WAIT_OBJECT_0)
		{
			hMsg = OpenEvent(EVENT_ALL_ACCESS, false, L"MsgA");
			strcpy(filename, "MsgA.txt");
			break;
		}
		if (WaitForSingleObject(hMutex[1], 1000) == WAIT_OBJECT_0)
		{
			hMsg = OpenEvent(EVENT_ALL_ACCESS, false, L"MsgB");
			strcpy(filename, "MsgB.txt");
			break;
		}
	}

	hEnd = OpenEvent(EVENT_MODIFY_STATE, false, L"ReaderEnd");
	if (hMsg == NULL || hEnd == NULL)
	{
		cout << "Can't open Event" << endl;
		Sleep(10000);
		return GetLastError();
	}

	LPWSTR msgCountStr = GetCommandLine();
	int msgCount = _wtoi(msgCountStr);
	char buf[256] = { 0 };
```

```
	for (int i = 0; i < msgCount; i++)
	{
		cout << "Waiting for message..." << endl;

		DWORD dwRes = WaitForSingleObject(hMsg, INFINITE);
		if (dwRes != WAIT_OBJECT_0)
		{
			cout << "Wait for single object failed" << endl;
			return GetLastError();
		}

		fp = fopen(filename, "rt");
		memset(buf, 0, sizeof(buf));
		fread(buf, sizeof(char), 256, fp);
		fclose(fp);

		printf_s("Message received: %s\n", buf);
	}

	SetEvent(hEnd);
	cout << endl << "Close in 5 sec..." << endl;
	Sleep(5000);

	ReleaseMutex(hMutex[0]);
	ReleaseMutex(hMutex[1]);
	CloseHandle(hMutex[0]);
	CloseHandle(hMutex[1]);
	CloseHandle(hMsg);
	CloseHandle(hEnd);

	return 0;
}
```

Writer.cpp

```
#include <windows.h>
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{
	HANDLE hMutex[2];
	HANDLE hMsg, hEnd;

	hMutex[0] = OpenMutex(SYNCHRONIZE, false, L"MutexW1");
	hMutex[1] = OpenMutex(SYNCHRONIZE, false, L"MutexW2");

	if (hMutex[0] == NULL || hMutex[1] == NULL)
	{
		cout << "Can't open Mutex" << endl;
		Sleep(10000);
		return GetLastError();
	}

	FILE* fp;
```

```
char filename[64];

cout << "Waiting for ending other Writers..." << endl;
while (true)
{
	if (WaitForSingleObject(hMutex[0], 1000) == WAIT_OBJECT_0)
	{
		hMsg = OpenEvent(EVENT_ALL_ACCESS, false, L"MsgA");
		strcpy(filename, "MsgA.txt");
		break;
	}
	if (WaitForSingleObject(hMutex[1], 1000) == WAIT_OBJECT_0)
	{
		hMsg = OpenEvent(EVENT_ALL_ACCESS, false, L"MsgB");
		strcpy(filename, "MsgB.txt");
		break;
	}
}

hEnd = OpenEvent(EVENT_MODIFY_STATE, false, L"WriterEnd");
if (hMsg == NULL || hEnd == NULL)
{
	cout << "Can't open Event" << endl;
	Sleep(10000);
	return GetLastError();
}

LPWSTR msgCountStr = GetCommandLine();
int msgCount = _wtoi(msgCountStr);
char buf[256] = { 0 };

for (int i = 0; i < msgCount; i++)
{
	cout << "Type message " << i + 1 << ":";
	cin >> buf;

	fp = fopen(filename, "wt");
	fwrite(buf, sizeof(char), 256, fp);
	fclose(fp);

	cout << "Sending message..." << endl;
	SetEvent(hMsg);
	Sleep(1000);
}

SetEvent(hEnd);
cout << endl << "Close in 5 sec..." << endl;
Sleep(4000);

ReleaseMutex(hMutex[0]);
ReleaseMutex(hMutex[1]);
CloseHandle(hMutex[0]);
CloseHandle(hMutex[1]);
CloseHandle(hMsg);
CloseHandle(hEnd);

return 0; }
```