Санкт-Петербургский государственный политехнический университет

Факультет технической кибернетики

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе №6**

«Синхронизация потоков в ОС WINDOWS»

Работу выполнил студент группы № 4081/12

Дорофеев Юрий Владимирович

Работу принял преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Малышев Игорь Алексеевич

г. Санкт-Петербург

2012

**Содержание работы**

[1. Цель работы 3](#_Toc343622331)

[2. Программа работы 3](#_Toc343622332)

[2.1. Необходимо создать два потока, которые выполняются в одном адресном пространстве (в одном процессе): поток – производитель  и поток – потребитель. 3](#_Toc343622333)

[2.2. Мьютексы 5](#_Toc343622334)

[2.3. Семафоры 6](#_Toc343622335)

[2.4. Критические секции 8](#_Toc343622336)

[2.5. Объекты события 9](#_Toc343622337)

[2.6. Условные переменные 11](#_Toc343622338)

[2.7. Функции ожидания 13](#_Toc343622339)

[3. Создадим аналогичные программы для множества потоков, количество которых можно задать из командной строки 13](#_Toc343622340)

[3.1. Мьютексы 13](#_Toc343622341)

[3.2. Семафоры 15](#_Toc343622342)

[3.3. Критическая секция 17](#_Toc343622343)

[3.4. Объекты события 19](#_Toc343622344)

[3.5. Условные переменные 21](#_Toc343622345)

[3.6. Функции ожидания 23](#_Toc343622346)

[4. Предусмотрим условие окончания работы программы semap.cpp 24](#_Toc343622347)

[4.1. По таймеру 24](#_Toc343622348)

[4.2. Интерактивно от оператора 26](#_Toc343622349)

[5. Выводы 27](#_Toc343622350)

1. Цель работы

Изучить средства синхронизации потоков в ОС Windows.

1. Программа работы
   1. Необходимо создать два потока, которые выполняются в одном адресном пространстве (в одном процессе): поток – производитель  и поток – потребитель.

Потоки разделяют целочисленный массив, в который заносятся  производимые и извлекаются потребляемые данные. Для наглядности и контроля за происходящим целесообразно в буфер помещать наращиваемое значение, однозначно идентифицирующее производителя и номер его очередной посылки.

Код должен удовлетворять трем требованиям:

* потребитель не должен пытаться извлечь значение из буфера, если буфер пуст;
* производитель не должен пытаться поместить значение в буфер, если буфер полон;
* состояние буфера должно описываться общими переменными (индексами, счетчиками, указателями связанных списков и т.д.).

Задание необходимо выполнить различными способами, применив следующие средства синхронизации доступа к разделяемому ресурсу:

* мьютексы;
* семафоры;
* критические секции;
* объекты события;
* условные переменные;
* функции ожидания.

Реализация буфера, удовлетворяющая требованиям:

buffer.h:

#ifndef \_BUFER\_H

#define \_BUFER\_H

#define BUF\_LEN 10

struct buf{

int arr[BUF\_LEN];

int counter;

//конструктор

buf(void);

int GetData(void);

int SetData(int data);

};

#endif

buffer.cpp:

#include "bufer.h"

//конструктор

buf::buf(void){

for(int i=0; i<BUF\_LEN; i++){ //обнуляем буфер

arr[i] = 0;

}

counter = 0; //обнуляем счетчик

}

//считывание из буфера, возвращает -1 если буфер пуст или первый элемент

//буфера (положительное число) если буфер не пуст

int buf::GetData(void)

{

int data;

if(counter == 0) //буфер пуст?

{

return -1;

}

else //если не пуст то:

{

data = arr[0]; //считываем первый элемент

counter --; //декрамент счетчика

for(int i=0; i<BUF\_LEN-1; i++)

{

arr[i] = arr[i+1]; //сдвигаем все элементы буфера на 1 влево

}

return data;

}

}

//запись в буфер, возвращает -1 если буфер полон или 0 если в буфере есть свободное место

//запись производится в конец буфера

int buf::SetData(int data){

if(counter == BUF\_LEN) //буфер полон?

{

return -1;

}

else //если нет, то:

{

arr[counter] = data; //записываем данные в конец буфера

counter ++; //инкремент счетчика

return 0;

}

}

Метод, реализующий чтение из буфера, проверяет наличие в буфере данных и если буфер не пуст, то считывает первый элемент массива и возвращает его, а так же декрементирует счетчик.

Метод записи данных проверяет наличие свободного места в буфере и если буфер не полон, то записывает указанные данные в массив на первое свободное место и инкрементирует счетчик.

* 1. Мьютексы

mutex.cpp:

#include <stdio.h>

#include <windows.h>

#include "bufer.h"

DWORD WINAPI Producer(LPVOID); //производитель

DWORD WINAPI Comsumer(LPVOID); //потребитель

buf Buffer; //буфер

***HANDLE hMutex; //описатель мьютекса***

int main(void)

{

DWORD ProducerId, ComsumerId; //идентификаторы потоков

HANDLE hProducer, hComsumer; //дескрипторы потоков

char st;

***hMutex = CreateMutex(NULL, FALSE, NULL); //создаем мьютекс***

//создаем потоки

hProducer = CreateThread(NULL, 0, Producer, NULL, 0, &ProducerId);

hComsumer = CreateThread(NULL, 0, Comsumer, NULL, 0, &ComsumerId);

scanf("%c", &st); //ожидаем какого-либо ввода

printf("Streams are closed and the mutex is completed\n");

//закрываем потоки

CloseHandle(hProducer);

CloseHandle(hComsumer);

***CloseHandle(hMutex); //закрываем мьютекс***

}

//производитель

DWORD WINAPI Producer(LPVOID)

{

int currentNumber = 0; //число, записываемое в буфер

int res;

while(true)

{

***WaitForSingleObject(hMutex, INFINITE); //ожидаем освобождения мьютекса***

res = Buffer.SetData(currentNumber); //запись числа в буфер

if(res != -1) //если буфер не полон

{

printf("Write in the buffer: %d\n", currentNumber);

currentNumber ++; //инкрементируем текущее число

}

***ReleaseMutex(hMutex); //освобождаем мьютекс***

Sleep(300); //засыпаем на 300 милисекунд

}

return 0;

}

//потребитель

DWORD WINAPI Comsumer(LPVOID)

{

int currentNumber = 0; //число, считанное из буфера

while(true)

{

***WaitForSingleObject(hMutex, INFINITE); //ожидаем освобождения мьютекса***

currentNumber = Buffer.GetData(); //чтение числа из буфера

if(currentNumber != -1) //если буфер не пуст

{

printf("Read from the buffer: %d\n", currentNumber);

}

***ReleaseMutex(hMutex); //освобождаем мьютекс***

Sleep(200); //засыпаем на 200 милисекунд

}

return 0;

}

Синхронизация работы потоков производится с помощью мьютекса. Потоки в бесконечном цикле ожидают освобождения мьютекса, и после получения мьютекса выполняют свои действия.

Базовый поток процесса после создания мьютекса и дополнительных потоков, переходит в режим ожидания ввода от пользователя, а затем закрывает потоки и мьютекс и завершается.

Запустим программу:

E:\mutex\Debug>mutex.exe

Write in the buffer: 0

Read from the buffer: 0

Write in the buffer: 1

Read from the buffer: 1

Write in the buffer: 2

Read from the buffer: 2

Write in the buffer: 3

Read from the buffer: 3

Streams are closed and the mutex is completed

Видно, что сначала производитель заносит данные в буфер, а потребитель их читает. Программа работает верно.

* 1. Семафоры

semap.cpp:

#include <stdio.h>

#include <windows.h>

#include "bufer.h"

DWORD WINAPI Producer(LPVOID); //производитель

DWORD WINAPI Comsumer(LPVOID); //потребитель

buf Buffer; //буфер

***HANDLE hSemap; //описатель семафора***

int main(void)

{

DWORD ProducerId, ComsumerId; //идентификаторы потоков

HANDLE hProducer, hComsumer; //дескрипторы потоков

char st;

***hSemap = CreateSemaphore(NULL, 1, 1, NULL); //создаем семафор***

//создаем потоки

hProducer = CreateThread(NULL, 0, Producer, NULL, 0, &ProducerId);

hComsumer = CreateThread(NULL, 0, Comsumer, NULL, 0, &ComsumerId);

scanf("%c", &st); //ожидаем какого-либо ввода

printf("Streams are closed and the mutex is completed\n");

//закрываем потоки

CloseHandle(hProducer);

CloseHandle(hComsumer);

***CloseHandle(hSemap); //закрываем семофор***

}

//производитель

DWORD WINAPI Producer(LPVOID)

{

int currentNumber = 0; //число, записываемое в буфер

int res;

while(true){

***WaitForSingleObject(hSemap, INFINITE); //ожидаем освобождения семафора***

res = Buffer.SetData(currentNumber); //запись числа в буфер

if(res != -1) //если буфер не полон

{

printf("Write in the buffer: %d\n", currentNumber);

currentNumber ++; //инкрементируем текущее число

}

***ReleaseSemaphore(hSemap, 1, NULL); //освобождаем семафор***

Sleep(300); //засыпаем на 300 милисекунд

}

return 0;

}

//потребитель

DWORD WINAPI Comsumer(LPVOID)

{

int currentNumber = 0; //число, считанное из буфера

while(true){

***WaitForSingleObject(hSemap, INFINITE); //ожидаем освобождения семафора***

currentNumber = Buffer.GetData(); //чтение числа из буфера

if(currentNumber != -1) //если буфер не пуст

{

printf("Read from the buffer: %d\n", currentNumber);

}

***ReleaseSemaphore(hSemap, 1, NULL); //освобождаем семафор***

Sleep(200); //засыпаем на 200 милисекунд

}

return 0;

}

Запустим программу:

E:\semap\Debug>semap.exe

Write in the buffer: 0

Read from the buffer: 0

Write in the buffer: 1

Read from the buffer: 1

Write in the buffer: 2

Read from the buffer: 2

Write in the buffer: 3

Read from the buffer: 3

Streams are closed and the semaphore is completed

Как видно, результаты выполнения аналогичны результатам при использовании мьютекса.

* 1. Критические секции

crs.cpp:

#include <stdio.h>

#include <windows.h>

#include "bufer.h"

DWORD WINAPI Producer(LPVOID); //производитель

DWORD WINAPI Comsumer(LPVOID); //потребитель

buf Buffer; //буфер

***CRITICAL\_SECTION cr\_s; //созд. критической секции***

int main(void)

{

DWORD ProducerId, ComsumerId; //идентификаторы потоков

HANDLE hProducer, hComsumer; //дескрипторы потоков

char st;

***InitializeCriticalSection(& cr\_s); //инициализация критической секции***

//создаем потоки

hProducer = CreateThread(NULL, 0, Producer, NULL, 0, &ProducerId);

hComsumer = CreateThread(NULL, 0, Comsumer, NULL, 0, &ComsumerId);

scanf("%c", &st); //ожидаем какого-либо ввода

printf("Streams are closed\n");

//закрываем потоки

CloseHandle(hProducer);

CloseHandle(hComsumer);

***DeleteCriticalSection(&cr\_s); //удаление критической секции***

}

//производитель

DWORD WINAPI Producer(LPVOID)

{

int currentNumber = 0; //число, записываемое в буфер

int res;

while(true){

***EnterCriticalSection(&cr\_s); //вызов критической секции***

res = Buffer.SetData(currentNumber); //запись числа в буфер

if(res != -1) //если буфер не полон

{

printf("Write in the buffer: %d\n", currentNumber);

currentNumber ++; //инкрементируем текущее число

}

***LeaveCriticalSection(&cr\_s); //освобождение критической секции***

Sleep(300); //засыпаем на 300 милисекунд

}

return 0;

}

//потребитель

DWORD WINAPI Comsumer(LPVOID)

{

int currentNumber = 0; //число, считанное из буфера

while(true){

***EnterCriticalSection(&cr\_s); //вызов критической секции***

currentNumber = Buffer.GetData(); //чтение числа из буфера

if(currentNumber != -1) //если буфер не пуст

{

printf("Read from the buffer: %d\n", currentNumber);

}

***LeaveCriticalSection(&cr\_s); //освобождение критической секции***

Sleep(200); //засыпаем на 200 милисекунд

}

return 0;

}

Запустим программу:

E:\crs\Debug>crs.exe

Write in the buffer: 0

Read from the buffer: 0

Write in the buffer: 1

Read from the buffer: 1

Write in the buffer: 2

Read from the buffer: 2

Write in the buffer: 3

Read from the buffer: 3

Streams are closed

Как видно, программа работает правильно. Результаты совпадают с полученными ранее.

* 1. Объекты события

event.cpp:

#include <stdio.h>

#include <windows.h>

#include "bufer.h"

DWORD WINAPI Producer(LPVOID); //производитель

DWORD WINAPI Comsumer(LPVOID); //потребитель

buf Buffer; //буфер

***HANDLE hEvFirst, hEvSecond; //описатели событий***

int main(void)

{

DWORD ProducerId, ComsumerId; //идентификаторы потоков

HANDLE hProducer, hComsumer; //дескрипторы потоков

char st;

//создание событий

***hEvFirst = CreateEvent(NULL, FALSE, FALSE, NULL);***

***hEvSecond = CreateEvent(NULL, FALSE, TRUE, NULL);***

//создаем потоки

hProducer = CreateThread(NULL, 0, Producer, NULL, 0, &ProducerId);

hComsumer = CreateThread(NULL, 0, Comsumer, NULL, 0, &ComsumerId);

scanf("%c", &st); //ожидаем какого-либо ввода

printf("Streams are closed\n");

//закрываем потоки

CloseHandle(hProducer);

CloseHandle(hComsumer);

//закрываем события

***CloseHandle(hEvFirst);***

***CloseHandle(hEvSecond);***

}

//производитель

DWORD WINAPI Producer(LPVOID)

{

int currentNumber = 0; //число, записываемое в буфер

int res;

while(true){

***WaitForSingleObject(hEvSecond, INFINITE); //ожидание события от потребителя***

res = Buffer.SetData(currentNumber); //запись числа в буфер

if(res != -1) //если буфер не полон

{

printf("Write in the buffer: %d\n", currentNumber);

currentNumber ++; //инкрементируем текущее число

}

***SetEvent(hEvFirst); //установка события от производителя***

Sleep(300); //засыпаем на 300 милисекунд

}

return 0;

}

//потребитель

DWORD WINAPI Comsumer(LPVOID)

{

int currentNumber = 0; //число, считанное из буфера

while(true){

***WaitForSingleObject(hEvFirst, INFINITE); //ожидание события от производителя***

currentNumber = Buffer.GetData(); //чтение числа из буфера

if(currentNumber != -1) //если буфер не пуст

{

printf("Read from the buffer: %d\n", currentNumber);

}

***SetEvent(hEvSecond); //установка события от потребителя***

Sleep(200); //засыпаем на 200 милисекунд

}

return 0;

}

Запустим программу:

E:\event\Debug>event.exe

Write in the buffer: 0

Read from the buffer: 0

Write in the buffer: 1

Read from the buffer: 1

Write in the buffer: 2

Read from the buffer: 2

Write in the buffer: 3

Read from the buffer: 3

Streams are closed

В этой программе для взаимодействия друг с другом в потоках использован механизм событий. Производитель и потребитель ждут наступления события от потребителя и производителя соответственно. После прихода события и его получения в потоке оно автоматически сбрасывается, а поток начинает выполнения кода. По завершению работы поток устанавливает свое событие.

Как видно, программа работает правильно. Результаты совпадают с полученными ранее.

* 1. Условные переменные

variable.cpp:

#include <stdio.h>

#include <windows.h>

#include "bufer.h"

DWORD WINAPI Producer(LPVOID); //производитель

DWORD WINAPI Comsumer(LPVOID); //потребитель

buf Buffer; //буфер

***bool first, second; //условные переменные***

int main(void)

{

DWORD ProducerId, ComsumerId; //идентификаторы потоков

HANDLE hProducer, hComsumer; //дескрипторы потоков

char st;

//инициализация переменных

***first=false;***

***second=true;***

//создаем потоки

hProducer = CreateThread(NULL, 0, Producer, NULL, 0, &ProducerId);

hComsumer = CreateThread(NULL, 0, Comsumer, NULL, 0, &ComsumerId);

scanf("%c", &st); //ожидаем какого-либо ввода

printf("Streams are closed\n");

//закрываем потоки

CloseHandle(hProducer);

CloseHandle(hComsumer);

}

//производитель

DWORD WINAPI Producer(LPVOID)

{

int currentNumber = 0; //число, записываемое в буфер

int res;

while(true){

***if(second)***

{

***second=false;***

res = Buffer.SetData(currentNumber);//запись числа в буфер

if(res != -1) //если буфер не полон

{

printf("Write in the buffer: %d\n", currentNumber);

currentNumber ++; //инкрементируем текущее число

}

Sleep(300); //засыпаем на 300 милисекунд

***first=true; //установка переменной от производителя***

}

}

return 0;

}

//потребитель

DWORD WINAPI Comsumer(LPVOID)

{

int currentNumber = 0; //число, считанное из буфера

while(true){

***if(first)***

{

***first=false;***

currentNumber = Buffer.GetData(); //чтение числа из буфера

if(currentNumber != -1) //если буфер не пуст

{

printf("Read from the buffer: %d\n", currentNumber);

}

Sleep(200); //засыпаем на 200 милисекунд

***second=true; //установка переменной от потребителя***

}

}

return 0;

}

В данной программе для синхронизации работы потоков используются глобальные булевы переменные. Принцип их работы аналогичен работе событий.

Запустим программу:

E:\variable\Debug>variable.exe

Write in the buffer: 0

Read from the buffer: 0

Write in the buffer: 1

Read from the buffer: 1

Write in the buffer: 2

Read from the buffer: 2

Write in the buffer: 3

Read from the buffer: 3

Streams are closed

Как видно, программа работает правильно. Результаты совпадают с полученными ранее.

* 1. Функции ожидания

Функции ожидания использовались в предыдущих программах за исключением условных переменных.

1. Создадим аналогичные программы для множества потоков, количество которых можно задать из командной строки
   1. Мьютексы

mutex.cpp:

#include <stdio.h>

#include <windows.h>

#include "bufer.h"

#define MAX\_THREADS 64

DWORD WINAPI Producer(LPVOID num); //производитель

DWORD WINAPI Comsumer(LPVOID num); //потребитель

buf Buffer; //буфер

int numProd, numComs; //количество потоков производителя и потребителя

***HANDLE hMutex; //описатель мьютекса***

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

DWORD ThId; //идентификатор потоков

HANDLE hProducer[MAX\_THREADS];//массив дескрипторов потоков произв.

HANDLE hComsumer[MAX\_THREADS];//массив дескрипторов потоков потреб.

char st;

if(argc < 3) //проверка, что указаны все параметры

{

numProd = 2; //значения по умолчанию

numComs = 2;

}

else{

numProd = atoi(argv[1]);

numComs = atoi(argv[2]);

}

//создание потоков

if((numProd > 0) && (numComs > 0))

{

for(int i=0; i<numProd; i++){ //производители

hProducer[i] = CreateThread(NULL, 0, Producer, (LPVOID)i, 0, &ThId);

}

for(int i=0; i<numComs; i++){ //потребители

hComsumer[i] = CreateThread(NULL, 0, Comsumer, (LPVOID)i, 0, &ThId);

}

}

else

{

printf("Error!");

return -1;

}

***hMutex = CreateMutex(NULL, FALSE, NULL); //создаем мьютекс***

scanf("%c", &st); //ожидаем какого-либо ввода

printf("Streams are closed and the mutex is completed\n");

//закрываем потоки

for(int i=0; i<numProd; i++){ //производители

CloseHandle(hProducer[i]);

}

for(int i=0; i<numComs; i++){ //потребители

CloseHandle(hComsumer[i]);

}

**CloseHandle(hMutex); //закрываем мьютекс**

}

//производитель

DWORD WINAPI Producer(LPVOID num)

{

int currentNumber = (int)num; //число, записываемое в буфер

int res;

while(true){

***WaitForSingleObject(hMutex, INFINITE); //ожидаем освобождения мьютекса***

res = Buffer.SetData(currentNumber); //запись числа в буфер

***ReleaseMutex(hMutex); //освобождаем мьютекс***

if(res != -1) //если буфер не полон

{

printf("The producer %d write in the buffer: %d\n", (int)num, currentNumber);

currentNumber ++; //инкрементируем текущее число

}

Sleep(300); //засыпаем на 300 милисекунд

}

return 0;

}

//потребитель

DWORD WINAPI Comsumer(LPVOID num)

{

int currentNumber = 0; //число, считанное из буфера

while(true){

***WaitForSingleObject(hMutex, INFINITE); //ожидаем освобождения мьютекса***

currentNumber = Buffer.GetData(); //чтение числа из буфера

***ReleaseMutex(hMutex); //освобождаем мьютекс***

if(currentNumber != -1) //если буфер не пуст

{

printf("The comsumer %d read from the buffer: %d\n", (int)num, currentNumber);

}

Sleep(200); //засыпаем на 200 милисекунд

}

return 0;

}

В программе в соответствии с указанными параметрами в цикле создаются потоки производителей и потребителей.

Запустим программу:

E:\mutex\Debug>mutex.exe 3 2

The producer 1 write in the buffer: 1

The comsumer 0 read from the buffer: 1

The producer 2 write in the buffer: 2

The producer 0 write in the buffer: 0

The comsumer 1 read from the buffer: 0

The comsumer 0 read from the buffer: 2

The producer 1 write in the buffer: 2

The producer 0 write in the buffer: 1

The producer 2 write in the buffer: 3

The comsumer 1 read from the buffer: 2

The comsumer 0 read from the buffer: 1

The producer 1 write in the buffer: 3

The comsumer 1 read from the buffer: 3

The comsumer 0 read from the buffer: 3

The producer 0 write in the buffer: 2

The producer 2 write in the buffer: 4

Streams are closed and the mutex is completed

Видно, что производители и потребители записывают и извлекают в произвольном порядке.

* 1. Семафоры

semap.cpp:

#include <stdio.h>

#include <windows.h>

#include "bufer.h"

#define MAX\_THREADS 64

DWORD WINAPI Producer(LPVOID num); //производитель

DWORD WINAPI Comsumer(LPVOID num); //потребитель

buf Buffer; //буфер

int numProd, numComs; //количество потоков производителя и потребителя

***HANDLE hSemap; //описатель семафора***

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

DWORD ThId; //идентификатор потоков

HANDLE hProducer[MAX\_THREADS];//массив дескрипторов потоков произв.

HANDLE hComsumer[MAX\_THREADS];//массив дескрипторов потоков потреб.

char st;

if(argc < 3) //проверка, что указаны все параметры

{

numProd = 2; //значения по умолчанию

numComs = 2;

}

else{

numProd = atoi(argv[1]);

numComs = atoi(argv[2]);

}

//создание потоков

if((numProd > 0) && (numComs > 0))

{

for(int i=0; i<numProd; i++){ //производители

hProducer[i] = CreateThread(NULL, 0, Producer, (LPVOID)i, 0, &ThId);

}

for(int i=0; i<numComs; i++){ //потребители

hComsumer[i] = CreateThread(NULL, 0, Comsumer, (LPVOID)i, 0, &ThId);

}

}

else

{

printf("Error!");

return -1;

}

***hSemap = CreateSemaphore(NULL, 1, 1, NULL); //создаем семафор***

scanf("%c", &st); //ожидаем какого-либо ввода

printf("Streams are closed and the semaphore is completed\n");

//закрываем потоки

for(int i=0; i<numProd; i++){ //производители

CloseHandle(hProducer[i]);

}

for(int i=0; i<numComs; i++){ //потребители

CloseHandle(hComsumer[i]);

}

***CloseHandle(hSemap); //закрываем семафор***

}

//производитель

DWORD WINAPI Producer(LPVOID num)

{

int currentNumber = (int)num; //число, записываемое в буфер

int res;

while(true){

***WaitForSingleObject(hSemap, INFINITE); //ожидаем освобождения семафора***

res = Buffer.SetData(currentNumber); //запись числа в буфер

***ReleaseSemaphore(hSemap, 1, NULL); //освобождаем семафора***

if(res != -1) //если буфер не полон

{

printf("The producer %d write in the buffer: %d\n", (int)num, currentNumber);

currentNumber ++; //инкрементируем текущее число

}

Sleep(300); //засыпаем на 300 милисекунд

}

return 0;

}

//потребитель

DWORD WINAPI Comsumer(LPVOID num)

{

int currentNumber = 0; //число, считанное из буфера

while(true){

***WaitForSingleObject(hSemap, INFINITE); //ожидаем освобождения семафора***

currentNumber = Buffer.GetData(); //чтение числа из буфера

***ReleaseSemaphore(hSemap, 1, NULL); //освобождаем семафор***

if(currentNumber != -1) //если буфер не пуст

{

printf("The comsumer %d read from the buffer: %d\n", (int)num, currentNumber);

}

Sleep(200); //засыпаем на 200 милисекунд

}

return 0;

}

Запустим программу:

E:\semap\Debug>semap.exe 3 2

The producer 0 write in the buffer: 0

The producer 2 write in the buffer: 2

The comsumer 1 read from the buffer: 0

The producer 1 write in the buffer: 1

The comsumer 0 read from the buffer: 2

The comsumer 1 read from the buffer: 1

The producer 0 write in the buffer: 1

The producer 2 write in the buffer: 3

The producer 1 write in the buffer: 2

Streams are closed and the semaphore is completed

Результаты выполнения программы аналогичны предыдущей программе.

* 1. Критическая секция

crs.cpp:

#include <stdio.h>

#include <windows.h>

#include "bufer.h"

#define MAX\_THREADS 64

DWORD WINAPI Producer(LPVOID num); //производитель

DWORD WINAPI Comsumer(LPVOID num); //потребитель

buf Buffer; //буфер

int numProd, numComs; //количество потоков производителя и потребителя

***CRITICAL\_SECTION cr\_s; //критическая секция***

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

DWORD ThId; //идентификатор потоков

HANDLE hProducer[MAX\_THREADS];//массив дескрипторов потоков произв.

HANDLE hComsumer[MAX\_THREADS];//массив дескрипторов потоков потреб.

char st;

if(argc < 3) //проверка, что указаны все параметры

{

numProd = 2; //значения по умолчанию

numComs = 2;

}

else{

numProd = atoi(argv[1]);

numComs = atoi(argv[2]);

}

//создание потоков

if((numProd > 0) && (numComs > 0))

{

for(int i=0; i<numProd; i++){ //производители

hProducer[i] = CreateThread(NULL, 0, Producer, (LPVOID)i, 0, &ThId);

}

for(int i=0; i<numComs; i++){ //потребители

hComsumer[i] = CreateThread(NULL, 0, Comsumer, (LPVOID)i, 0, &ThId);

}

}

else

{

printf("Error!");

return -1;

}

***InitializeCriticalSection(&cr\_s); //инициализация критической секции***

scanf("%c", &st); //ожидаем какого-либо ввода

printf("Streams are closed and the semaphore is completed\n");

//закрываем потоки

for(int i=0; i<numProd; i++){ //производители

CloseHandle(hProducer[i]);

}

for(int i=0; i<numComs; i++){ //потребители

CloseHandle(hComsumer[i]);

}

***DeleteCriticalSection(&cr\_s); //удаление критической секции***

}

//производитель

DWORD WINAPI Producer(LPVOID num)

{

int currentNumber = (int)num; //число, записываемое в буфер

int res;

while(true){

***EnterCriticalSection(&cr\_s); //вход в критическую секцию***

res = Buffer.SetData(currentNumber); //запись числа в буфер

***LeaveCriticalSection(&cr\_s); //освобождение критической секции***

if(res != -1) //если буфер не полон

{

printf("The producer %d write in the buffer: %d\n", (int)num, currentNumber);

currentNumber ++; //инкрементируем текущее число

}

Sleep(300); //засыпаем на 300 милисекунд

}

return 0;

}

//потребитель

DWORD WINAPI Comsumer(LPVOID num)

{

int currentNumber = 0; //число, считанное из буфера

while(true){

***EnterCriticalSection(&cr\_s); //вход в критическую секцию***

currentNumber = Buffer.GetData(); //чтение числа из буфера

***LeaveCriticalSection(&cr\_s); //освобождение критической секции***

if(currentNumber != -1) //если буфер не пуст

{

printf("The comsumer %d read from the buffer: %d\n", (int)num, currentNumber);

}

Sleep(200); //засыпаем на 200 милисекунд

}

return 0;

}

Запустим программу:

E:\Soft\crs\Debug>crs.exe 1

The producer 0 write in the buffer: 0

The producer 1 write in the buffer: 1

The comsumer 0 read from the buffer: 0

The comsumer 1 read from the buffer: 1

The producer 1 write in the buffer: 2

The producer 0 write in the buffer: 1

The comsumer 1 read from the buffer: 2

The comsumer 0 read from the buffer: 1

Streams are closed

Результаты выполнения программы аналогичны предыдущей программе.

* 1. Объекты события

event.cpp:

#include <stdio.h>

#include <windows.h>

#include "bufer.h"

#define MAX\_THREADS 64

DWORD WINAPI Producer(LPVOID num); //производитель

DWORD WINAPI Comsumer(LPVOID num); //потребитель

buf Buffer; //буфер

int numProd, numComs; //количество потоков производителя и потребителя

***HANDLE hEvent; //событие***

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

DWORD ThId; //идентификатор потоков

HANDLE hProducer[MAX\_THREADS];//массив дескрипторов потоков произв.

HANDLE hComsumer[MAX\_THREADS];//массив дескрипторов потоков потреб.

char st;

if(argc < 3) //проверка, что указаны все параметры

{

numProd = 2; //значения по умолчанию

numComs = 2;

}

else{

numProd = atoi(argv[1]);

numComs = atoi(argv[2]);

}

//создание потоков

if((numProd > 0) && (numComs > 0))

{

for(int i=0; i<numProd; i++){ //производители

hProducer[i] = CreateThread(NULL, 0, Producer, (LPVOID)i, 0, &ThId);

}

for(int i=0; i<numComs; i++){ //потребители

hComsumer[i] = CreateThread(NULL, 0, Comsumer, (LPVOID)i, 0, &ThId);

}

}

else

{

printf("Error!");

return -1;

}

//создаем событие в активном состоянии с автоматическим сбросом

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

scanf("%c", &st); //ожидаем какого-либо ввода

printf("Streams are closed and the semaphore is completed\n");

//закрываем потоки

for(int i=0; i<numProd; i++){ //производители

CloseHandle(hProducer[i]);

}

for(int i=0; i<numComs; i++){ //потребители

CloseHandle(hComsumer[i]);

}

***CloseHandle(hEvent);//закрываем событие***

}

//производитель

DWORD WINAPI Producer(LPVOID num)

{

int currentNumber = (int)num; //число, записываемое в буфер

int res;

while(true){

***WaitForSingleObject(hEvent, INFINITE);//ожидаем события и сбрасывает его***

res = Buffer.SetData(currentNumber); //запись числа в буфер

***SetEvent(hEvent);//снова устанавливаем событие***

if(res != -1) //если буфер не полон

{

printf("The producer %d write in the buffer: %d\n", (int)num, currentNumber);

currentNumber ++; //инкрементируем текущее число

}

Sleep(300); //засыпаем на 300 милисекунд

}

return 0;

}

//потребитель

DWORD WINAPI Comsumer(LPVOID num)

{

int currentNumber = 0; //число, считанное из буфера

while(true){

***WaitForSingleObject(hEvent, INFINITE);//ожидаем события и сбрасывает его***

currentNumber = Buffer.GetData(); //чтение числа из буфера

***SetEvent(hEvent);//снова устанавливаем событие***

if(currentNumber != -1) //если буфер не пуст

{

printf("The comsumer %d read from the buffer: %d\n", (int)num, currentNumber);

}

Sleep(200); //засыпаем на 200 милисекунд

}

return 0;

}

Запустим программу:

E:\event\Debug>event.exe 2 3

The producer 0 write in the buffer: 0

The producer 1 write in the buffer: 1

The comsumer 0 read from the buffer: 0

The comsumer 2 read from the buffer: 1

The producer 0 write in the buffer: 1

The producer 1 write in the buffer: 2

The comsumer 2 read from the buffer: 2

Streams are closed

Результаты выполнения программы аналогичны предыдущей программе

* 1. Условные переменные

#include <stdio.h>

#include <windows.h>

#include "bufer.h"

#define MAX\_THREADS 64

DWORD WINAPI Producer(LPVOID num); //производитель

DWORD WINAPI Comsumer(LPVOID num); //потребитель

buf Buffer; //буфер

int numProd, numComs; //количество потоков производителя и потребителя

bool first; //переменная

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

DWORD ThId; //идентификатор потоков

HANDLE hProducer[MAX\_THREADS];//массив дескрипторов потоков произв.

HANDLE hComsumer[MAX\_THREADS];//массив дескрипторов потоков потреб.

char st;

if(argc < 3) //проверка, что указаны все параметры

{

numProd = 2; //значения по умолчанию

numComs = 2;

}

else{

numProd = atoi(argv[1]);

numComs = atoi(argv[2]);

}

//создание потоков

if((numProd > 0) && (numComs > 0))

{

for(int i=0; i<numProd; i++){ //производители

hProducer[i] = CreateThread(NULL, 0, Producer, (LPVOID)i, 0, &ThId);

}

for(int i=0; i<numComs; i++){ //потребители

hComsumer[i] = CreateThread(NULL, 0, Comsumer, (LPVOID)i, 0, &ThId);

}

}

else

{

printf("Error!");

return -1;

}

***first = true; //инициализация переменной***

scanf("%c", &st); //ожидаем какого-либо ввода

printf("Streams are closed\n");

//закрываем потоки

for(int i=0; i<numProd; i++){ //производители

CloseHandle(hProducer[i]);

}

for(int i=0; i<numComs; i++){ //потребители

CloseHandle(hComsumer[i]);

}

}

//производитель

DWORD WINAPI Producer(LPVOID num)

{

int currentNumber = (int)num; //число, записываемое в буфер

int res;

while(true){

***if(first)***

{

***first=false;***

res = Buffer.SetData(currentNumber);//запись числа в буфер

***first=true;***

if(res != -1) //если буфер не полон

{

printf("The producer %d write in the buffer: %d\n", (int)num, currentNumber);

currentNumber ++; //инкрементируем текущее число

}

Sleep(300); //засыпаем на 300 милисекунд

}

}

return 0;

}

//потребитель

DWORD WINAPI Comsumer(LPVOID num)

{

int currentNumber = 0; //число, считанное из буфера

while(true){

***if(first)***

{

***first=false;***

currentNumber = Buffer.GetData(); //чтение числа из буфера

***first=true;***

if(currentNumber != -1) //если буфер не пуст

{

printf("The comsumer %d read from the buffer: %d\n", (int)num, currentNumber);

}

Sleep(200); //засыпаем на 200 милисекунд

}

}

return 0;

}

Запустим программу:

E:\variable\Debug>variable.exe 3 2

The producer 0 write in the buffer: 0

The producer 1 write in the buffer: 1

The producer 2 write in the buffer: 2

The comsumer 1 read from the buffer: 0

The comsumer 0 read from the buffer: 1

The comsumer 1 read from the buffer: 2

The producer 0 write in the buffer: 1

Streams are closed

Результаты выполнения программы аналогичны предыдущей программе.

* 1. Функции ожидания

Функции ожидания использовались в предыдущих программах за исключением условных переменных.

1. Предусмотрим условие окончания работы программы semap.cpp

* 1. По таймеру

semap.cpp:

#include <stdio.h>

#include <windows.h>

#include "bufer.h"

#define MAX\_THREADS 64

#define SEC 10000000

DWORD WINAPI Producer(LPVOID num); //производитель

DWORD WINAPI Comsumer(LPVOID num); //потребитель

buf Buffer; //буфер

int numProd, numComs; //количество потоков производителя и потребителя

HANDLE hSemap; //описатель семафора

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

{

DWORD ThId; //идентификатор потоков

HANDLE hProducer[MAX\_THREADS];//массив дескрипторов потоков произв.

HANDLE hComsumer[MAX\_THREADS];//массив дескрипторов потоков потреб.

***HANDLE Tim;***

***\_\_int64 end\_time;***

***LARGE\_INTEGER end\_time2;***

SYSTEMTIME st;

***int time;***

if(argc < 4) //проверка, что указаны все параметры

{

numProd = 2; //значения по умолчанию

numComs = 2;

***time=2;***

}

else{

numProd = atoi(argv[1]);

numComs = atoi(argv[2]);

***time=atoi(argv[3]);***

}

***Tim = CreateWaitableTimer(NULL,FALSE,NULL); //создаем таймер***

***end\_time=-1 \* time \* SEC; //определяем время завершения программы***

***end\_time2.LowPart = (DWORD) (end\_time & 0xFFFFFFFF);***

***end\_time2.HighPart = (LONG) (end\_time >> 32);***

GetLocalTime(&st); //получаем системное время

printf("Time: %d:%d:%d\n",st.wHour,st.wMinute,st.wSecond);

//создание потоков

if((numProd > 0) && (numComs > 0))

{

for(int i=0; i<numProd; i++){ //производители

hProducer[i] = CreateThread(NULL, 0, Producer, (LPVOID)i, 0, &ThId);

}

for(int i=0; i<numComs; i++){ //потребители

hComsumer[i] = CreateThread(NULL, 0, Comsumer, (LPVOID)i, 0, &ThId);

}

}

else

{

printf("Error!");

return -1;

}

hSemap = CreateSemaphore(NULL, 1, 1, NULL); //создаем семафор

***SetWaitableTimer(Tim,&end\_time2,0,NULL,NULL,FALSE); //устанавливаем таймер***

***WaitForSingleObject(Tim,INFINITE); //ожидаем таймер***

GetLocalTime(&st);

printf("\nProcess stopped at: %d:%d:%d\n", st.wHour, st.wMinute, st.wSecond);

***CloseHandle(Tim); //закрываем таймер***

//закрываем потоки

for(int i=0; i<numProd; i++){ //производители

CloseHandle(hProducer[i]);

}

for(int i=0; i<numComs; i++){ //потребители

CloseHandle(hComsumer[i]);

}

CloseHandle(hSemap); //закрываем семафор

}

//производитель

DWORD WINAPI Producer(LPVOID num)

{

int currentNumber = (int)num; //число, записываемое в буфер

int res;

while(true){

WaitForSingleObject(hSemap, INFINITE); //ожидаем освобождения семафора

res = Buffer.SetData(currentNumber); //запись числа в буфер

ReleaseSemaphore(hSemap, 1, NULL); //освобождаем семафор

if(res != -1) //если буфер не полон

{

printf("The producer %d write in the buffer: %d\n", (int)num, currentNumber);

currentNumber ++; //инкрементируем текущее число

}

Sleep(300); //засыпаем на 300 милисекунд

}

return 0;

}

//потребитель

DWORD WINAPI Comsumer(LPVOID num)

{

int currentNumber = 0; //число, считанное из буфера

while(true){

WaitForSingleObject(hSemap, INFINITE); //ожидаем освобождения семафора

currentNumber = Buffer.GetData(); //чтение числа из буфера

ReleaseSemaphore(hSemap, 1, NULL); //освобождаем семафор

if(currentNumber != -1) //если буфер не пуст

{

printf("The comsumer %d read from the buffer: %d\n", (int)num, currentNumber);

}

Sleep(200); //засыпаем на 200 милисекунд

}

return 0;

}

Время выполнения программы указывается в качестве третьего аргумента командной строки. На указанное время устанавливается таймер и по его сигналу программа завершается.

Запустим программу:

E:\semap\Debug>semap.exe 2 3 1

Time: 22:1:36

The producer 0 write in the buffer: 0

The comsumer 0 read from the buffer: 0

The producer 1 write in the buffer: 1

The comsumer 1 read from the buffer: 1

The producer 1 write in the buffer: 2

The producer 0 write in the buffer: 1

<…>

Process stopped at: 22:1:37

* 1. Интерактивно от оператора

Во всех предыдущих программах, кроме последней реализуется этот способ завершения программы.

1. Выводы

Мьютексы позволяют координировать взаимное исключение доступа к разделяемому ресурсу. Сигнальное состояние мьюткса соответствует моменту времени, когда объект не принадлежит ни одному потоку и его можно "захватить". Сброшенное состояние соответствует моменту, когда какой-либо поток уже владеет этим объектом. Доступ к мьютексу разрешается, когда поток, владеющий объектом, освободит его.

Семафор позволяет захватить себя нескольким потокам, после чего захват будет невозможен, пока один из ранее захвативших семафор потоков не освободит его. Семафоры применяются для ограничения количества потоков, одновременно работающих с ресурсом. Сигнальному состоянию соответствует значение счетчика больше нуля. Когда счетчик равен нулю, семафор считается не установленным.

Семафор позволяет остановить выполнение потока до тех пор, пока он не получит сигнала от другого потока о том, что работу можно продолжать, т. е. позволяет блокировать одновременное выполнение потоков. Мьютекс - служит для совместного использования неких данных, т. е. он запрещает прерывать текущий поток до выполнения некоторых действий. Задача мьютекса — защита объекта от доступа к нему других потоков, отличных от того, который завладел мьютексом.

Критическая секция помогает выделить участок кода, где поток получает доступ к разделяемому ресурсу, и предотвратить одновременное использование ресурса. Перед использованием ресурса поток входит в критическую секцию. Если после этого другой поток попытается войти в ту же самую критическую секцию, его выполнение приостановится, пока первый поток не покинет секцию. Критические секции используется только для потоков одного процесса, в отличии от мьютексов.

События используются для уведомления ожидающих потоков о наступлении какого-либо события. Существует два вида событий - с ручным и автоматическим сбросом. События с ручным сбросом используются для уведомления сразу нескольких потоков. При использовании события с автоматическим сбросом уведомление получит и продолжит свое выполнение только один ожидающий поток, остальные будут ожидать дальше.

Практически все средства синхронизации (кроме критических секций) используются вместе с ожидающими функциями.