# Лабораторная работа № 4. Семафоры

## Функции для работы с семафорами

**Семафор** – это объект синхронизации процессов или потоков, который может находиться в двух состояниях – открыт или закрыт. В общем и целом использование семафоров аналогично использованию мьютексов. Более того, можно утверждать, что мьютексы являются частным случаем реализации семафоров – бинарными семафорами.

Главным отличием семафоров от мьютексов с точки зрения их использования для синхронизации потоков является то, что семафоры могут использоваться для организации доступа к делимым ресурсам, т.к. в состав семафора входит специальная переменная – счетчик. Счетчик может принимать значения от 0 до n, где n – количество единиц разделяемого ресурса, доступом к которому управляет монитор. Если n = 1, то семафор считается бинарным, в противном случае семафор принято называть считающими. Для считающих семафоров состояние «открыт» характеризуется значением счетчика больше 0, а состояние «закрыт» возникает, когда счетчик равен 0.

Семафор, как и мьютекс, является системным объектом Microsoft Windows, поэтому его можно использовать для синхронизации потоков, определенных в контексте разных процессов. При этом, как и в случае с мьютексом, для синхронизации потоков, принадлежащих разным процессам, используются поименованные семафоры. Анонимные семафоры используются для синхронизации потоков в рамках одного процесса и/или его дочерних процессов, которые унаследовали указатель (handle) данного семафора.  
Для создания семафора используется функция **CreateSemaphore**:

HANDLE CreateSemaphore(

LPSECURITY\_ATTRIBUTES *lpSemaphoreAttributes*,

LONG *lInitialCount*,

LONG *lMaximumCount*,

LPCTSTR *lpName*

);

Параметры функции CreateSemaphore:

Параметры lpSemaphoreAttributes и lpName аналогичны параметрам функции CreateMutex. Параметры lInitialCount и lMaximumCount используются для настройки счетчика семафора. Параметр lMaximumCount определяет максимальное значение счетчика, а параметр lInitialCount – текущее значение счетчика, или, вернее, то значение, с которым семафор будет создан. Если параметр lInitialCount равен нулю, то семафор создается в закрытом состоянии, в противном случае – в состоянии «открыт».

Возвращаемые значения функции CreateSemaphore:

1. Если создание семафора прошло успешно, то функция вернет значение указателя (handle) созданного семафора.

2. Если во время создания семафора произошла ошибка, то функция вернет значение NULL, а более подробно об ошибке можно узнать, обратившись к функции GetLastError.

3. Если при создании поименованного семафора окажется, что семафор с таким именем уже существует, то функция все равно вернет его указатель (handle), а значением функции GetLastError будет код ошибки ERROR\_ALREADY\_EXISTS.

Открыть уже существующий семафор можно при помощи функции **OpenSemaphore**:

HANDLE OpenSemaphore(

DWORD *dwDesiredAccess*,

BOOL *bInheritHandle*,

LPCTSTR *lpName*

);

Функция позволяет определить права доступа к семафору, возможность наследования указателя (handle) семафора потомками процесса, открывшего семафор, и имя открываемого семафора. В случае успеха, функция возвращает указатель (handle) семафора, иначе – значение NULL.

Дождаться освобождения семафора и захватить его перед входом в критическую секцию можно при помощи функции **WaitForSingleObject**:

DWORD WaitForSingleObject(

HANDLE *hHandle*,

DWORD *dwMilliseconds*

);

Если же потоку необходимо дождаться освобождения и захватить несколько семафоров (или один из нескольких семафоров), то можно воспользоваться функцией **WaitForMultipleObjects**:

DWORD WaitForMultipleObjects(

DWORD *nCount*,

const HANDLE\* *lpHandles*,

BOOL *bWaitAll*,

DWORD *dwMilliseconds*

);

Значение параметров данных функций аналогично тем, которые использовались при работе с потоками (см. лаб. работу № 1), за исключением того, что в качестве дескриптора hHandleиспользуется дескриптор семафора, и вместо массива дескрипторов потоков lpHandlesиспользуется массив дескрипторов семафоров.

При работе с семафорами данные функции выполняют следующие действия:

- проверка состояния семафора;

- если мьютекс открыт, то уменьшить значение его счетчика на 1 и войти в критическую секцию. При этом, если после уменьшения счетчика он примет значение 0, то семафор перейдет в состояние «закрыт»;

- в противном случае (счетчик семафора уже равен нулю) – приостановить выполнение потока до того момента, пока семафор не окажется в состоянии «открыт», т.е. пока его счетчик не превысит значение 0.

Освобождение семафора и увеличение значения его счетчика можно выполнить функцией **ReleaseSemaphore**:

BOOL ReleaseSemaphore(

HANDLE *hSemaphore*,

LONG *lReleaseCount*,

LPLONG *lpPreviousCount*

);

Параметры функции ReleaseSemaphore:

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| hSemaphore | Указатель (handle) изменяемого семафора |
| lReleaseCoun | Значение, на которое будет изменено значение счетчика семафора |
| lpPreviousCount | Указатель на переменную типа LONG, в которую будет сохранено предыдущее значение счетчика семафора – значение до изменения |

Функция ReleaseSemaphore возвращает значение true, если после изменения семафор перешел в состояние «открыт», и значение false в случае, когда счетчик остался в нулевом состоянии. Эта ситуация считается ошибкой. Получить большую информацию можно, как всегда, воспользовавшись функцией GetLastError.

После использования семафоры можно освободит функцией CloseHandle.

## Пример программы

***Задача****.* В качестве примера использования семафоров используем другой классический пример синхронизации процессов – задачу «Читатели-писатели». В целом задача является развитием задачи «Производитель-потребитель». В системе имеется *n* потоков писателей, которые генерируют данные в буфере длиной *k* единиц. Также имеется *m* потоков читателей, которые считывают и обрабатывают информацию из указанного буфера. Писатель может записать информацию в буфер только в том случае, если в буфере есть хотя бы одна свободная ячейка, в противном случае – все потоки-писатели должны быть приостановлены. Читатель может считать и обработать информацию только тогда, когда она есть в буфере. Если буфер пуст, то все потоки-читатели должны быть приостановлены.

В качестве частного примера рассмотрим задачу, в которой *n* потоков-писателей генерируют записи об изменении счета клиента в банке, а *m* потоков-читателей обрабатывают эту информацию и формируют баланс счета. Каждая операция характеризуется видом (приход/расход), которые будем обозначать операциями +/−, и значением операции. Баланс должен содержать графы «сумма прихода», «сумма расхода» и «оборот по счету».

***Решение****.* Для решения задачи потребуется два считающих семафора: один для подсчета пустых ячеек в буфере, другой – для заполненных, и одни мьютекс, который блокирует доступ к разделяемым между всеми потоками ресурсам – буфером и переменными для накопления сумм.

#include <windows.h>

#include <iostream>

const int n = 20, m = 10, k = 7;

const int oper\_number = 100;

// Описание структуры буфера и самого буфера

struct s\_operation

{

char oper;

float amount;

} buffer[k];

// Текущая позиция буфера

int cur\_pos;

// Значения для итоговых переменных

float income = 0,

outcome = 0,

total = 0;

HANDLE hSemFull,

hSemEmpty,

hMutex;

// Функция писателя

DWORD WINAPI writer(LPVOID param)

{

for(int i = 0; i<oper\_number; i++)

{

// Дождаться освобождения ячеек

WaitForSingleObject(hSemEmpty,INFINITE);

// Захватить критический ресурс и сформировать новую операцию

WaitForSingleObject(hMutex,INFINITE);

std::cout << "Введите вид операции: ";

std::cin >> buffer[cur\_pos].oper;

std::cout << "Введите сумму операции: ";

std::cin >> buffer[cur\_pos].amount;

cur\_pos++;

// Освободить критический ресурс

ReleaseMutex(hMutex);

// Увеличить число занятых ячеек

ReleaseSemaphore(hSemFull,1,NULL);

}

return 0;

}

// Функция читателя

DWORD WINAPI reader(LPVOID param)

{

for(int i = 0; i<oper\_number; i++)

{

// Дождаться появления заполненных ячеек

WaitForSingleObject(hSemFull,INFINITE);

// Захватить критический ресурс и обработать операцию

WaitForSingleObject(hMutex,INFINITE);

cur\_pos--;

switch (buffer[cur\_pos].oper)

{

case '+': income += (buffer[cur\_pos].amount);

break;

case '-': outcome -= (buffer[cur\_pos].amount);

break;

default:

std::cout << "Ошибочная операция\n";

}

// Освободить критический ресурс

ReleaseMutex(hMutex);

// Увеличить число пустых ячеек

ReleaseSemaphore(hSemEmpty,1,NULL);

}

return 0;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

// Массивы дескрипторов потоков

HANDLE hReaders[m], hWriters[n];

cur\_pos = 0;

// Создание семафоров

// семафор занятых ячеек создаем закрытым

hSemFull = CreateSemaphore(NULL,0,k,NULL);

// семафор пустых ячеек создаем открытым

hSemEmpty = CreateSemaphore(NULL,k,k,NULL);

hMutex = CreateMutex(NULL,false,NULL);

// Запуск потоков

for (int i = 0; i < m; i++)

{

hReaders[i] = CreateThread(NULL,0,reader,NULL,0,NULL);

}

for (int i = 0; i < n; i++)

{

hWriters[i] = CreateThread(NULL,0,writer,NULL,0,NULL);

}

// Необходимо дождаться завершения всех потоков

WaitForMultipleObjects(n,hWriters,true,INFINITE);

WaitForMultipleObjects(m,hReaders,true,INFINITE);

// Подсчет итоговой суммы и вывод результатов

total = income - outcome;

std::cout << "Приход = " << income << '\t' << "Расход = " << outcome << '\t' << "Оборот = " << total << '\n';

for(int i = 0; i<n+m; i++)

CloseHandle(hThread[i]);

CloseHandle(hSemFull);

CloseHandle(hSemEmpty);

CloseHandle(hMutex);

return 0;

}

## Задания

1. Напишите программу управления пулом печати. В системе, имеется пул печати объемом *n* байт, поток управления пулом и m потоков, осуществляющих печать.

Поток, получивший доступ к пулу может записать в него либо весь текст, предназначенный для печати, либо часть текста, если места в пуле недостаточно. Если места в пуле недостаточно, то поток должен быть приостановлен до тех пор, пока поток управления пулом не освободит место в пуле. Поток, начавший вывод информации в пул не может быть прерван другими потоками вывода текста до тех пор, пока не осуществит вывод всего текста. Запись текста всегда осуществляется в конец пула.

Поток управления пулом может получить доступ к пулу только тогда, когда в пуле имеются заполненные ячейки. Чтение из пула осуществляется из начала. При этом остальное содержимое пула сдвигается к началу – выведенный на печать текст удаляется.

Для имитации работы пула используйте потоки, читающие текстовые файлы, и поток управления, выполняющий запись текста в файл.