Реализация потоков Windows

Создание и завершение потоков

- Функция, создающая поток, который выполняется в пределах виртуального адресного пространства вызывающего процесса.

```
HANDLE CreateThread
(
      LPSECURITY_ATTRIBUTES lpThreadAttributes,   // дескриптор защиты
      SIZE_T dwStackSize,                         // начальный размер стека
      LPTHREAD_START_ROUTINE lpStartAddress,      // функция потока
      LPVOID lpParameter,                         // параметр потока
      DWORD dwCreationFlags,                      // опции создания
      LPDWORD lpThreadId                          // идентификатор потока
);
```

- Функция, заканчивающая работу потока

```
VOID ExitThread
(
      DWORD dwExitCode  // код выхода для этого потока
);
```

`ExitThread` – предпочтительный метод завершения работы потока. Когда эта функция вызывается (или явно, или при помощи возврата из процедуры потока), стек текущего потока освобождается, а поток завершает работу. Функция точки входа всех связанных с потоком динамически подключаемых библиотек (DLL) вызывается со значением, указывающим, что поток отключается от DLL.

- Функция, проверяющая текущее состояние указанного объекта. Если данный объект находится в неотмеченном состоянии, выполнение потока приостанавливается.

```
DWORD WaitForSingleObject
(
      HANDLE hHandle,          //дескриптор объекта
      DWORD dwMilliseconds     //интервал времени
);
```

Функция `WaitForSingleObject` возвращает свое значение в двух случаях: когда указанный объект устанавливается в отмеченное состояние, и когда истекает время ожидания. В процессе ожидания поток практически не использует процессорное время. Перед завершением своей работы функция изменяет состояние некоторых объектов синхронизации. Изменения происходят только в том случае, если изменение состояния объекта привело к выходу из функции. Например, счетчик семафора уменьшается на единицу.

Функция WaitForSingleObject может использоваться со следующими объектами:

- извещениями об изменениях;
- вводом с системной консоли;
- объектами событий;
- заданиями;
- мютексами;

- процессами;
- семафорами;
- потоками;
- таймерами ожидания.

Средства синхронизации

Мьютекс

- Функция для инициализации мьютекса.

```
HANDLE CreateMutex
(
      LPSECURITY_ATTRIBUTES lpMutexAttributes, // атрибут безопасности
      BOOL bInitialOwner,     // флаг начального владельца
      LPCTSTR lpName    // имя объекта
);
```

Результатом будет дескриптор нового объекта Mutex, а если такое имя уже есть, то дескриптор существующего. В последнем случае функция `GetLastError()` при вызове будет выдавать значение `ERROR_ALREADY_EXISTS`.

- Функция, открывающая (запрашивающая) мьютекс.

```
HANDLE OpenMutex
(
      DWORD   fdwAccess,      // требуемый доступ
      BOOL    fInherit,       // флаг наследования
      LPCTSTR lpszMutexName   // адрес имени объекта Mutex
);
```

С помощью функции `OpenMutex` несколько задач могут открыть один и тот же объект Mutex и затем выполнять одновременное ожидание для этого объекта.

- Функция, освобождающая мьютекс.

```
BOOL ReleaseMutex
(
      HANDLE hMutex     // дескриптор mutex
);
```

Данная функция при успешном выполнении вернет ненулевое значение.

Пример создания потока при помощи функции CreateThread

```
#include <windows.h>
#include <iostream.h>
volatile int n;
DWORD WINAPI Add(LPVOID iNum)
{
      cout << "Thread is started." << endl;
      n += (int)iNum;
      cout << "Thread is finished." << endl;
```

```
	return 0;
}
int main()
{
	int inc = 10;
	HANDLE hThread;
	DWORD IDThread;
	cout << "n = " << n << endl;
	hThread = CreateThread(NULL, 0, Add, (void*)inc, 0, &IDThread);
	if (hThread == NULL)
		return GetLastError();
	// ждем пока поток Add закончит работу
	WaitForSingleObject(hThread, INFINITE);
	// закрываем дескриптор потока Add
	CloseHandle(hThread);
	cout << "n = " << n << endl;
	return 0;
}
```

Отметим, что в этой программе используется функция `WaitForSingleObject`, которая ждет завершения потока `Add`.

Семафор

- Функция для создания семафора.

```
HANDLE CreateSemaphore
(
    LPSECURITY__ATTRIBUTES lpSemaphoreAttributes, //атрибут безопасности
    LONG lInitialCount, //начальное значение счетчика
    LONG lMaximumCount, //максимальное значение счетчика
    LPCTSTR lpName      //адрес имени объекта Semaphore
);
```

После того как семафор успешно создан, поток может обратиться к ресурсу, защищенному семафором, с помощью одной из **wait-функций**. При этом **wait-функции** передается дескриптор семафора.

- Функция, увеличивающая значение счетчика.

```
BOOL ReleaseSemaphore
(
    HANDLE hSemaphore,  //дескриптор семафора
    LONG lReleaseCount, //увеличение счетчика
    LPLONG lpPreviousCount    //предыдущее значение счетчика
);
```

Пример работы с семафорами в ОС MS Windows

```
#include <windows.h>
#include <stdio.h>

#define MAX_SEM_COUNT 10
#define THREADCOUNT 12
```

```
HANDLE ghSemaphore;

DWORD WINAPI ThreadProc( LPVOID );

int main( void )
{
    HANDLE aThread[THREADCOUNT];
    DWORD ThreadID;
    int i;

    // Create a semaphore with initial and max counts of MAX_SEM_COUNT

    ghSemaphore = CreateSemaphore(
        NULL,           // default security attributes
        MAX_SEM_COUNT,  // initial count
        MAX_SEM_COUNT,  // maximum count
        NULL);          // unnamed semaphore

    if (ghSemaphore == NULL)
    {
        printf("CreateSemaphore error: %d\n", GetLastError());
        return 1;
    }

    // Create worker threads

    for( i=0; i < THREADCOUNT; i++ )
    {
        aThread[i] = CreateThread(
                     NULL,       // default security attributes
                     0,          // default stack size
                     (LPTHREAD_START_ROUTINE) ThreadProc,
                     NULL,       // no thread function arguments
                     0,          // default creation flags
                     &ThreadID); // receive thread identifier

        if( aThread[i] == NULL )
        {
            printf("CreateThread error: %d\n", GetLastError());
            return 1;
        }
    }

    // Wait for all threads to terminate

    WaitForMultipleObjects(THREADCOUNT, aThread, TRUE, INFINITE);

    // Close thread and semaphore handles

    for( i=0; i < THREADCOUNT; i++ )
        CloseHandle(aThread[i]);

    CloseHandle(ghSemaphore);

    return 0;
```

```
}

DWORD WINAPI ThreadProc( LPVOID lpParam )
{

    // lpParam not used in this example
    UNREFERENCED_PARAMETER(lpParam);

    DWORD dwWaitResult;
    BOOL bContinue=TRUE;

    while(bContinue)
    {
        // Try to enter the semaphore gate.

        dwWaitResult = WaitForSingleObject(
            ghSemaphore,   // handle to semaphore
            0L);           // zero-second time-out interval

        switch (dwWaitResult)
        {
            // The semaphore object was signaled.
            case WAIT_OBJECT_0:
                // TODO: Perform task
                printf("Thread %d: wait succeeded\n", GetCurrentThreadId());
                bContinue=FALSE;

                // Simulate thread spending time on task
                Sleep(5);

                // Release the semaphore when task is finished

                if (!ReleaseSemaphore(
                        ghSemaphore,  // handle to semaphore
                        1,            // increase count by one
                        NULL) )       // not interested in previous count
                {
                    printf("ReleaseSemaphore error: %d\n", GetLastError());
                }
                break;

            // The semaphore was nonsignaled, so a time-out occurred.
            case WAIT_TIMEOUT:
                printf("Thread %d: wait timed out\n", GetCurrentThreadId());
                break;
        }
    }
    return TRUE;
}
```