Tworzenie i zarządzanie wątkami

utworzenie wątku

```
HANDLE CreateThread(
    LPSECURITY_ATTRIBUTES lpThreadAttributes,   //pointer to thread security attributes
    DWORD dwStackSize,        //initial thread stack size, in bytes
    LPTHREAD_START_ROUTINE lpStartAddress,      //pointer to thread function
    LPVOID lpParameter,       //argument for new thread
    DWORD dwCreationFlags,    //creation flags
    LPDWORD lpThreadId        //pointer to returned thread identifier
   );
```

wstrzymanie, wznowienie i przerwanie pracy wątku

```
DWORD SuspendThread(HANDLE hThread);
DWORD ResumeThread(HANDLE hThread);  //returned value – previous suspend count
BOOL TerminateThread( HANDLE hThread, DWORD dwThreadExitCode );
```

pobranie informacji o wątku

```
HANDLE GetCurrentThread(VOID);
DWORD  GetCurrentThreadId(VOID);

BOOL GetExitCodeThread(
    HANDLE hThread,           // handle to the thread
    LPDWORD lpExitCode        // address to receive termination status
   );
```

zamknięcie pracy z wątkiem - sprzątanie

```
BOOL CloseHandle( HANDLE hObject );
```

przykład:

```
class A {...};
HANDLE handle1,handle2;

long WINAPI MyThread(){...}
long WINAPI MyThreadWithParam(A * parameter){...}
...
handle1 = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)MyThread,0,CREATE_SUSPENDED,0);
ResumeThread(handle1);
...
A param;
Handle2 = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) MyThreadWithParam,&param,0,0);
...
CloseHandle(handle1);
CloseHandle(handle2);
```

Oczekiwanie na zakończenie pracy wątku

```
DWORD WaitForSingleObject(
    HANDLE hHandle,           // handle of object to wait for
    DWORD dwMilliseconds      // time-out interval in milliseconds
   );

DWORD WaitForMultipleObjects(
    DWORD nCount,             // number of handles in the object handle array
    CONST HANDLE *lpHandles,  // pointer to the object-handle array
    BOOL bWaitAll,            // wait flag
    DWORD dwMilliseconds      // time-out interval in milliseconds
   );
```

przykład:

```
long WINAPI MyThread(){...}
...

handle = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)MyThread,0,0,0);
WaitForSingleObject(handle,1000);
...
CloseHandle...


HANDLE hThreads[10];

for (int i=0; i<10; i++) {
  hThreads[i] = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)MyThread,0,0,0);
}
WaitForMultipleObjects(10,hThreads,TRUE,INFINITE);
...
CloseHandle...
```

Synchronizacja pracy wątków

Zdarzenia

stworzenie [obiektu] zdarzenia

```
HANDLE CreateEvent(
    LPSECURITY_ATTRIBUTES lpEventAttributes,  // pointer to security attributes
    BOOL bManualReset,                        // flag for manual-reset event
    BOOL bInitialState,                       // flag for initial state
    LPCTSTR lpName                            // pointer to event-object name
   );
```

pobranie, otwarcie [obiektu] zdarzenia

```
HANDLE OpenEvent(
    DWORD dwDesiredAccess,       // access flag  (EVENT_ALL_ACCESS - default)
    BOOL bInheritHandle,         // inherit flag
    LPCTSTR lpName               // pointer to event-object name
   );
```

ustawienie, wywołanie zdarzenia

```
BOOL SetEvent( HANDLE hEvent );
BOOL PulseEvent( HANDLE hEvent );
```

ręczne zresetowanie, zgaszenie zdarzenia

```
BOOL ResetEvent( HANDLE hEvent );
```

oczekiwanie na zdarzenie

```
DWORD WaitForSingleObject( HANDLE hHandle, DWORD dwMilliseconds );
```

przykład:

```
HANDLE hEvent;

long WINAPI MyThread(){
    ...
    WaitForSingleObject(hEvent,INFINITE); //wait for event
    ...
}

...
//somewhere else before creating thread
hEvent = CreateEvent (0, FALSE, FALSE, 0);

...
//somewhere else – to launch event
SetEvent(hEvent); //or PulseEvent(hEvent);
...
CloseHandle...
```

Mutexy

stworzenie mutexu z określeniem jego posiadania – tylko jeden obiekt może posiadać mutex

```
HANDLE CreateMutex(
  LPSECURITY_ATTRIBUTES lpMutexAttributes,    // pointer to security attributes
  BOOL bInitialOwner,                         // flag for initial ownership
  LPCTSTR lpName                              // pointer to mutex-object name
);
```

otwarcie mutexu z określeniem jego posiadania

```
HANDLE OpenMutex(
  DWORD dwDesiredAccess,       // access flag
  BOOL bInheritHandle,         // inherit flag
  LPCTSTR lpName               // pointer to mutex-object name
);
```

zwolnienie mutexu

```
BOOL ReleaseMutex( HANDLE hEvent );
```

oczekiwanie na przyznanie (wzięcie w posiadanie) mutexu – jeśli jesteś w posiadaniu to funkcja wraca od razu

```
DWORD WaitForSingleObject( HANDLE hHandle, DWORD dwMilliseconds );
```

przykład:

```
HANDLE hMutex;

long WINAPI MyThread(){
    ...
    WaitForSingleObject(hMutex,INFINITE); //wait for mutex ownership
    ...
    ReleaseMutex(hMutex);                 // !!! release ownership
    ...
}

...
//somewhere else before creating thread
hMutex=CreateMutex(NULL, TRUE, “name_of_my_mutex”);
...
//somewhere else – to give access to mutex
ReleaseMutex(hMutex);
...
CloseHandle...
```

Semafory

stworzenie semafora i inicjacja

```
HANDLE CreateSemaphore(
    LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,  // pointer to security attributes
    LONG lInitialCount,                           // initial count
    LONG lMaximumCount,                           // maximum count
    LPCTSTR lpName                                // pointer to semaphore-object name
   );
```

otwarcie semafora – pobranie uchwytu

```
HANDLE OpenSemaphore(
    DWORD dwDesiredAccess,     // access flag
    BOOL bInheritHandle,       // inherit flag
    LPCTSTR lpName             // pointer to semaphore-object name
   );
```

wejście do sekcji chronionej (dekrementacja semafora o 1)

```
DWORD WaitForSingleObject( HANDLE hHandle, DWORD dwMilliseconds );
```

wyjście z sekcji chronionej – zwolnienie semafora

```
BOOL ReleaseSemaphore(
    HANDLE hSemaphore,        // handle of the semaphore object
    LONG lReleaseCount,       // amount to add to current count
    LPLONG lpPreviousCount    // address of previous count
   );
```

przykład:

```
HANDLE hSemaphore;

long WINAPI MyThread(){
    ...
    WaitForSingleObject(hSemaphore,INFINITE); // wait for access
    ...
    ReleaseSemaphore(hSemaphore,1,NULL);      // release semaphore
    ...
}

...
//somewhere else before creating thread
hSemaphore=CreateSemaphore(NULL, 0, 1, “FILE_EXISTS”);
...
//somewhere else – to give access
ReleaseSemaphore(hSemaphore,1,NULL);
...
CloseHandle...
```

Sekcje krytyczne

inicjalizacja i usunięcie sekcji krytycznej

```
VOID InitializeCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
VOID DeleteCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
```

wejście do i opuszczenie sekcji krytycznej

```
VOID EnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
BOOL TryEnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection);

VOID LeaveCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
```

przykład:

```
CRITICAL_SECTION cs;

long WINAPI MyThread(){
    ...
    EnterCriticalSection(&cs);
    ...
    LeaveCriticalSection(&cs);
    ...
}

...
//before creating threads
InitializeCriticalSection(&cs);
...
//after work
DeleteCriticalSection(&cs);
```

Oczekujące zegary

```
CreateWaitableTimer(), OpenWaitableTimer(), SetWaitableTimer(), CancelWaitableTimer()
```