

SO Cheat Sheet

Thread-uri - Windows

`HANDLE CreateThread (LPSECURITY_ATTRIBUTES lpThAttr, SIZE_T dwStackSize, LPTHREAD_START_ROUTINE lpStartAddr, LPVOID lpParameter, DWORD dwCreatFlags, LPDWORD lpThreadId)` – creează un fir de execuție

- **lpThAttr** - pointer la o structură de tip SECURITY_ATTRIBUTES, dacă e NULL handle-ul nu poate fi moștenit
- **dwStackSize** - mărimea inițială a stivei, în bytes; 0 - mărimea implicită
- **lpStartAddr** - pointer la funcția ce trebuie executată
- **lpParameter** - opțional - pointer la o variabilă
- **dwCreatFlags** - opțiuni: 0, CREATE_SUSPENDED, STACK_SIZE_PARAM_IS_A_RESERVATION
- **lpThreadId** - pointer unde va fi întors identificatorul
- **întoarce** - handle către threadul creat

`HANDLE OpenThread (DWORD dwDesireAccess, BOOL bInheritHandle, DWORD dwThreadId)` – deschide un obiect de tip Thread existent

- **dwDesireAccess** - drepturile de acces
- **bInheritHandle** - TRUE - handle-ul poate fi moștenit
- **dwThreadId** - identificatorul thread-ului
- **întoarce** - handle către thread

`DWORD WaitForSingleObject (HANDLE hHANDLE, DWORD dwMilliseconds)` – așteaptă terminarea unui fir de execuție

- **hHandle** - handle către obiect
- **dwMilliseconds** - intervalul de timeout (0 până la INFINITE)

`void ExitThread (DWORD dwExitCode)` – terminarea threadului curent cu specificarea codului de terminare

`BOOL TerminateThread (HANDLE hThread, DWORD dwExitCode)` – terminarea unui thread hThread cu specificarea codului de terminare

`BOOL GetExitCodeThread (HANDLE hThread, LPDWORD lpExitCode)`

- **hThread** - handle-ul threadului ce trebuie să aibă dreptul de acces THREAD_QUERY_INFORMATION
- **lpExitCode** - pointer la o variabilă în care va fi plasat codul de terminare al firului. Dacă firul nu și-a terminat execuția, această valoare va fi STILL_ACTIVE

`DWORD SuspendThread (HANDLE hThread)` – suspendă execuția unui thread

`DWORD ResumeThread (HANDLE hThread)` – reia execuția unui thread

Aceste funcții nu pot fi folosite pentru sincronizare, dar sunt utile pentru programe de debug.

`void Sleep (DWORD dwMilliseconds)` – suspendă execuția unui thread pe o perioadă de dwMilliseconds

`BOOL SwitchToThread (void)` – firul de execuție renunță doar la timpul pe care îl avea pe procesor în momentul respectiv
întoarce TRUE dacă procesorul este cedat unui alt thread și FALSE dacă nu există alte thread-uri gata de execuție

`HANDLE GetCurrentThread (void)` – întoarce un pseudo-handle pentru firul curent ce nu poate fi folosit decât de firul apelant

`DWORD GetCurrentThreadId (void)` – întoarce identificatorul threadului

`DWORD GetThreadId (HANDLE Thread)` – întoarce identificatorul threadului ce corespunde handle-ului Thread

Sincronizare

Funcții de așteptare

Funcțiile de așteptare sunt cele din familia WaitForSingleObject și au fost prezentate, în detaliu, în laboratorul de comunicație interproces.

Mutex Win32

Funcțiile de mai jos au fost prezentate, în detaliu, în cadrul laboratorului de comunicație interproces.

`HANDLE CreateMutex (LPSECURITY_ATTRIBUTES lpMutexAttr, BOOL bInitialOwner, LPCTSTR lpName)` – creează un mutex

`HANDLE OpenMutex (DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR lpName)` – deschide un mutex

`BOOL ReleaseMutex (HANDLE hMutex)` – cedează posesia mutexului

Semafoare Win32

Funcțiile de mai jos au fost prezentate, în detaliu, în cadrul laboratorului de comunicație interproces.

`HANDLE CreateSemaphore (LPSECURITY_ATTRIBUTES lpSemAttr, LONG lInitialCount, LONG lMaximumCount, LPCTSTR lpName)` – creează/deschide un semafor

`HANDLE OpenSemaphore (DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR lpName)` – deschide un semafor existent

`BOOL ReleaseSemaphore (HANDLE hSemaphore, LONG lReleaseCount, LPLONG lpPreviousCount)` – incrementează semaforul

Evenimente

`HANDLE CreateEvent (LPSECURITY_ATTRIBUTES lpEventAttributes, BOOL bManualReset, BOOL bInitialState, LPCTSTR lpName)` – creează un eveniment

• **lpEventAttributes** - pointer la o structură de tip SECURITY_ATTRIBUTES, dacă e NULL handle-ul nu poate fi moștenit

• **bManualReset** - TRUE - pentru manual-reset, FALSE - auto-reset

• **bInitialState** - TRUE - evenimentul e creat în starea signaled

• **lpName** - numele evenimentului sau NULL dacă se dorește a fi anonim

• **întoarce** - handle către evenimentul creat

`HANDLE SetEvent (HANDLE hEvent)` – setează evenimentul în starea signaled

`HANDLE ResetEvent (HANDLE hEvent)` – setează evenimentul în starea non-signaled

`HANDLE PulseEvent (HANDLE hEvent)` – semnalează toate thread-urile care așteaptă la un eveniment manual-reset

Secțiuni critice

`void InitializeCriticalSection (LPCRITICAL_SECTION pcrit_sect)` – inițializează o secțiune critică cu un contor de spin implicit egal cu 0

`BOOL InitializeCriticalSectionAndSpinCount (LPCRITICAL_SECTION pcrit_sect, DWORD dwSpinCount)` – permite specificarea contorului de spin

`DWORD SetCriticalSectionSpinCount (LPCRITICAL_SECTION pcrit_sect, DWORD dwSpinCount)` – permite modificarea contorului de spin al unei secțiuni critice

`void DeleteCriticalSection (LPCRITICAL_SECTION pcrit_sect)` – distruge o secțiune critică

`void EnterCriticalSection (LPCRITICAL_SECTION lpCriticalSection)` – similar cu pthread_mutex_lock() pentru mutexuri RECURSIVE

`void LeaveCriticalSection (LPCRITICAL_SECTION lpCriticalSection)` – similar cu pthread_mutex_unlock() pentru mutexuri RECURSIVE

`BOOL TryEnterCriticalSection (LPCRITICAL_SECTION lpCritSect)` – similar cu pthread_mutex_trylock() pentru mutexuri RECURSIVE

lpCritSect secțiunea critică
întoarce FALSE dacă secțiunea critică este ocupată de alt fir de execuție

Interlocked Variable Access

LONG InterlockedIncrement(LONG volatile *lpAddend) – incrementează, atomic, valoarea indicată de lpAddend și întoarce valoarea incrementată

LONG InterlockedDecrement(LONG volatile *lpDecend) – decrementează, atomic, valoarea indicată de lpAddend și întoarce valoarea decrementată

LONG InterlockedExchange(LONG volatile *Target, LONG Value) – scrie valoarea întreagă Value în zona indicată de Target și întoarce vechea valoarea a lui Target

LONG InterlockedExchangeAdd(LPLONG volatile Addend, LONG Value) – adaugă valoarea întreagă Value la zona indicată de Addend și întoarce vechea valoarea a lui Addend

PVOID InterlockedExchangePointer(PVOID volatile *Target, PVOID Value) – atribuie pointerul Value pointerului indicat de pointerul Target

LONG InterlockedCompareExchange(LONG volatile *dest, LONG exchange, LONG comp) – compară valoarea de la adresa dest cu valoarea comp și, dacă sunt egale, setează atomic valoarea de la adresa dest la valoarea exchange

PVOID InterlockedCompareExchangePointer(PVOID volatile *dest, PVOID exchange, PVOID comp) – compară pointerul de la adresa dest cu pointerul comp și, dacă sunt egale, setează atomic pointerul de la adresa dest la pointerul exchange

Timer Queues

HANDLE CreateTimerQueue(void) – întoarce un handle la coada de timere

BOOL DeleteTimerQueue(HANDLE TimerQueue) – marchează coada pentru ștergere, dar NU așteaptă ca toate callbackurile asociate cozii să se termine

BOOL DeleteTimerQueueEx(HANDLE TimerQueue, HANDLE CompletionEv) – eliberează resursele alocate cozii, oferind facilități suplimentare

- **TimerQueue** - coada
- **CompletionEv** - una din valorile NULL, INVALID_HANDLE_VALUE SAU un handle de tip Event
- **întoarce** - non-zero pentru succes

BOOL CreateTimerQueueTimer(PHANDLE phNewTimer, HANDLE TimerQueue, WAITORTIMERCALLBACK Callback, PVOID Parameter, DWORD DueTime, DWORD Period, ULONG Flags) – creează un timer

- **phNewTimer** - aici întoarce un HANDLE la timerul nou creat
- **TimerQueue** - coada la care este adăugat timerul. Dacă e NULL se folosește o coadă implicită
- **Callback** - callback de executat
- **Parameter** - parametru trimis callbackului

- **DueTime** - intervalul, în milisecunde, după care va expira, prima dată, timerul
- **Period** - perioada, în milisecunde, a timerului
- **Flags** - tipul callbackului: IO/NonIO, EXECUTEONLYONCE
- **întoarce** - non-zero pentru succes

VOID WaitOrTimerCallback(PVOID lpParameter, BOOLEAN TimerOrWaitFired) – semnătura unui callback

BOOL ChangeTimerQueueTimer(HANDLE TimerQueue, HANDLE Timer, ULONG DueTime, ULONG Period) – modifică timpul de expirare a unui timer

BOOL CancelTimerQueueTimer(HANDLE TimerQueue, HANDLE Timer) – dezactivează unui timer

BOOL DeleteTimerQueueTimer(HANDLE TimerQueue, HANDLE Timer, HANDLE CompletionEvent) – dezactivează ȘI distruge unui timer. CompletionEvent e similar cu cel din DeleteTimerQueueEx.

Registered Wait Functions

BOOL RegisterWaitForSingleObject(PHANDLE phNewWaitObject, HANDLE hObject, WAITORTIMERCALLBACK Callback, PVOID Context, ULONG dwMilliseconds, ULONG dwFlags) – înregistrează în thread pool o funcție de așteptare, al cărei callback va fi executat când expiră timeout-ul SAU când obiectul la care se așteaptă, hObject, trece în starea SIGNALED

- **phNewWaitObject** - aici întoarce un HANDLE la timerul nou creat
- **hObject** - obiectul de sincronizare la care se așteaptă
- **Callback** - callback de executat
- **Context** - parametru trimis callbackului
- **dwMilliseconds** - timeout
- **dwFlags** - EXECUTEONLYONCE etc.
- **întoarce** - non-zero pentru succes

VOID CALLBACK WaitOrTimerCallback(PVOID lpParameter, BOOLEAN TimerOrWaitFired) – semnătura unui callback

BOOL UnregisterWait(HANDLE WaitHandle) – anulează înregistrarea unei funcții de așteptare

BOOL UnregisterWaitEx(HANDLE WaitHandle, HANDLE CompletionEv) – anulează înregistrarea unei funcții de așteptare

- **WaitHandle** - handle-ul funcției
- **CompletionEv** - asemănător cu parametrul lui DeleteTimerQueueEx
- **întoarce** - non-zero pentru succes