Laborator 9

Thread-uri - Windows

Sisteme de Operare

14 - 20 aprilie 2011

Windows Threads API

- CreateThread
- ThreadProc
- WaitForSingleObject
- ExitThread
- TlsAlloc
- TlsFree
- TlsGetValue
- TlsSetValue
- GetCurrentThread

Mecanisme de sincronizare

- Mutex (POSIX, Win32)
- Semafor (POSIX, Win32)
- Secțiune critică (Win32)
- Variabilă de condiție (POSIX)
- Barieră (POSIX)
- Operații atomice cu variabile partajate (Win32)
- Thread pooling (Win32)

Mutex (Win32)

- Tratate in laboratorul IPC
- Creare
  - `HANDLE CreateMutex(LPSECURITY_ATTRIBUTES lpMutexAttributes, BOOL bInitialOwner, LPCTSTR lpName)`
- Deschidere
  - `HANDLE OpenMutex(DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR lpName)`
- Așteptare
  - Funcțiile din familia `WaitForSingleObject`
- Eliberare
  - `BOOL ReleaseMutex(HANDLE hMutex)`

Semafor (Win32)

- Tratate in laboratorul IPC
- Creare
  - `HANDLE CreateSemaphore(LPSECURITY_ATTRIBUTES semattr, LONG initial_count, LONG maximum_count, LPCTSTR name)`
- Deschidere
  - `HANDLE OpenSemaphore(DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR name)`
- Așteptare
  - Funcțiile din familia `WaitForSingleObject`
- Incrementare, cu `lReleaseCount`
  - `BOOL ReleaseSemaphore(HANDLE hSemaphore, LONG lReleaseCount, LPLONG lpPreviousCount)`

- `CRITICAL_SECTION`
- Sincronizare DOAR între firele de execuție ale **aceluiași proces**
- Inițializare/Distrugere
  - `void InitializeCriticalSection(LPCRITICAL_SECTION pcrit_sect)`
  - `void DeleteCriticalSection(LPCRITICAL_SECTION pcrit_sect)`
- Intrare/Ieșire
  - `void EnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection)`
  - `BOOL TryEnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection)`
  - `void LeaveCriticalSection(LPCRITICAL_SECTION lpCriticalSection)`

Operații atomice cu variabile partajate

- Incrementare/Decrementare
  - `LONG InterlockedIncrement(LONG volatile *lpAddend)`
  - `LONG InterlockedDecrement(LONG volatile *lpDecend)`
- Atribuire atomică
  - `LONG InterlockedExchange(LONG volatile *Target, LONG Value)`
  - `LONG InterlockedExchangeAdd(LPLONG volatile Addend, LONG Value)`
  - `PVOID InterlockedExchangePointer(PVOID volatile *Target, PVOID Value)`
- Atribuire atomică condiționată
  - `LONG InterlockedCompareExchange(LONG volatile *dest, LONG exchange, LONG comp)`
  - `PVOID InterlockedCompareExchangePointer(PVOID volatile *dest, PVOID exchange, PVOID comp)`

Thread pooling

- Fiecare task primește un thread din pool
- Eliminare overhead creare/terminare fire de execuție
- Task-urile pot fi:
  - Executate **imediat**
  - Executate **mai târziu** (operații de așteptare + funcție callback asociată)
    - Așteptarea terminării unei operații I/O asincrone
    - Așteptarea expirarii unui TimerQueue
    - Funcții de așteptare înregistrate

- Este posibil ca un thread să modifice o variabilă de pe stiva altui thread?
- Care este diferența între while(!conditie); și while (!conditie) sleep(1); ?
- Care este diferența dintre deadlock și livelock?