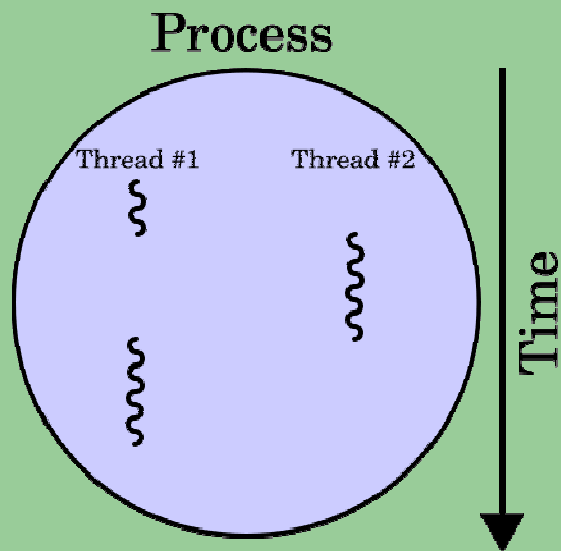


ΕΠΛ 428

Windows Threads and Concurrency



Ορέστης Σπανός
Χρίστος Κυριάκου

Ιστορική Αναδρομή

- Windows 3.1
 - Cooperative Multithreading
 - Η διεργασία είναι που καθορίζει το πότε θα φύγει από το CPU και να ελευθερώσει τους πόρους
 - ΑΡΝΗΤΙΚΑ: Μπορεί να δημιουργήσει πρόβλημα σε περίπτωση που περιμένει κάποιο πόρο να γίνει διαθέσιμος

Ιστορική Αναδρομή (Συνέχεια)

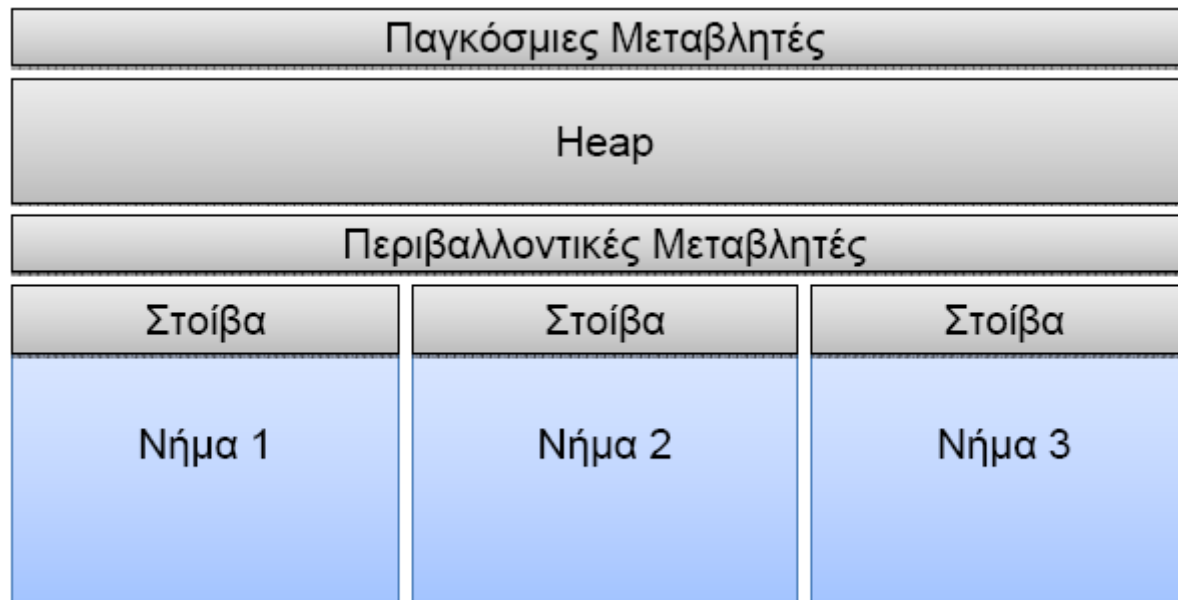
- Windows 98 Single CPU
- Windows 2000: Multi CPU
 - Preemptive multithreading
 - Το Λειτουργικό Σύστημα ελέγχει τη χρονοδρομολόγηση
 - ΑΡΝΗΤΙΚΑ: Το ΛΣ μπορεί να αλλάξει τη διεργασία που τρέχει στο CPU σε ακατάλληλη στιγμή και να έχουμε λάθος αποτελέσματα.
 - Ανάγκη για έλεγχο των κρίσιμων τμημάτων κώδικα

Τι είναι ένα Thread (Νήμα)

- Ανεξάρτητη ροή ελέγχου μέσα σε μια διεργασία
 - Έχει δική του στοίβα, Program Counter, registers
 - Μοιράζεται τη μνήμη με τη διεργασία στην οποία ανήκει.
- Πλεονεκτήματα χρήσης νημάτων
 - Αν κάποιο νήμα γίνει block π.χ. I/O, η διεργασία μπορεί να συνεχίσει κανονικά (αν υπάρχουν άλλα νήματα)

Τι είναι ένα Thread (Συνέχεια)

Διεργασία



Εγκατάσταση

- Χρήση της βιβλιοθήκης <windows.h>
- Δυναμικός σύνδεσμος με το kernel32.dll με χρήση του #pragma
- Το <windows.h> το βρίσκουμε στο Platform SDK στη διεύθυνση
<http://www.microsoft.com/downloads/details.aspx?familyid=E15438AC-60BE-41BD-AA14-7F1E0F19CA0D&displaylang=en>
- Έρχεται μαζί με το Visual Studio

Δημιουργία Νήματος

- HANDLE WINAPI CreateThread(
LPSECURITY_ATTRIBUTES *lpThreadAttributes*,
SIZE_T *dwStackSize*,
LPTHREAD_START_ROUTINE *lpStartAddress*,
LPVOID *lpParameter*,
DWORD *dwCreationFlags*, LPDWORD *lpThreadId*);
Επιστρέφει handle στο νέο thread / NULL σε λάθος

lpThreadAttributes: Παράμετροι ασφαλείας

dwStackSize: Αρχικό μέγεθος στοίβας για το νήμα

lpStartAddress: Συνάρτησης που θα εκτελεί το νήμα

lpParameters: Παράμετροι που θα περάσουν στη συνάρτηση

dwCreationFlags: Παράμετροι δημιουργίας νήματος

lpThreadId: Μεταβλητή η οποία θα παραλάβει το ID του thread.

Δημιουργία Νήματος (Συνέχεια)

- Μεταβλητές Volatile

Οι μεταβλητές που θέλουμε να είναι κοινές μεταξύ των νημάτων πρέπει να τις δηλώνουμε volatile που βασικά αναγκάζει τον compiler να κρατά συνεχώς ενήμερη τη μνήμη με τα περιεχόμενα της μεταβλητής.

Παράδειγμα 1

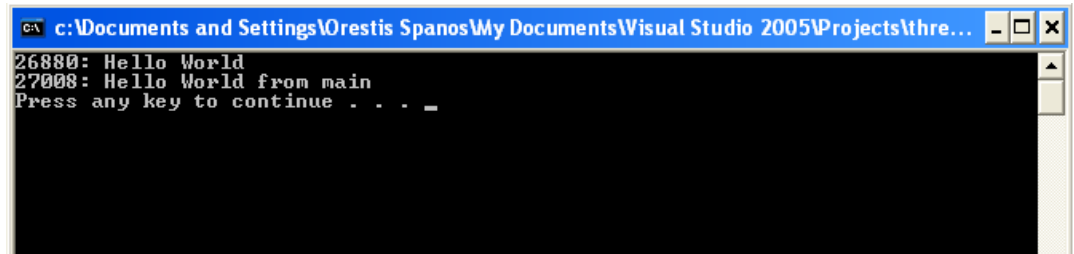
```
#include <windows.h>
#include <iostream>
#pragma comment(lib,"kernel32.lib")

using namespace std;

volatile UINT count = 0;

void CountThread() {
    cout<<GetCurrentThreadId()<<": Hello World"<<endl;
}

int main() {
    HANDLE countHandle;
    DWORD threadID;
    countHandle = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) CountThread, 0, 0, &threadID);
    if (countHandle==0)
        cout << "Cannot create thread: " << GetLastError() << endl;
    Sleep(10);
    cout<<GetCurrentThreadId()<<": Hello World from main"<<endl;
    system("pause");
}
```



```
c:\Documents and Settings\Orestis Spanos\My Documents\Visual Studio 2005\Projects\thre...
26880: Hello World
27008: Hello World from main
Press any key to continue . . . _
```

GetCurrentThreadId()

Επιστρέφει το μοναδικό αριθμό του thread

GetLastError()

Όπως την perror στα UNIX

Αναμονή Νημάτων

- Αν μια διεργασία που έχει δημιουργήσει νήματα πεθάνει τότε πεθαίνουν και αυτά

DWORD WaitForSingleObject(HANDLE hHandle, DWORD dwMilliseconds)

hHandle: Το Handle του νήματος

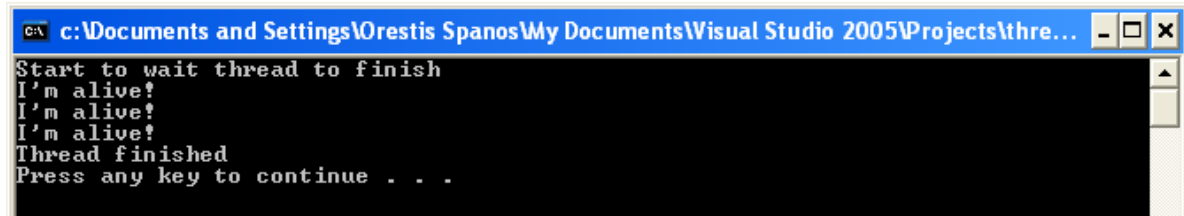
dwMilliseconds: Μέγιστος χρόνος για τον οποίο θα περιμένουμε (millisecond)

Επιστρέφει το γεγονός για το οποίο σταμάτησε ή κωδικό λάθους σε περίπτωση αποτυχίας

Παράδειγμα 2

```
#include <windows.h>
#include <iostream>
#pragma comment(lib,"kernel32.lib")
using namespace std;
volatile UINT count = 0;
void CountThread(LPVOID iter) {
    for (DWORD i = 0; i < iter; i++) {
        Sleep(1000);
        cout << "I'm alive!\n";
    }
}
int main() {
    HANDLE countHandle;
    DWORD threadID;
    DWORD iterations = 3;
    int count=0;
    countHandle=CreateThread(0, 0, (LPTHREAD_START_ROUTINE) CountThread, (LPVOID *) iterations, 0, &threadID);
    if (countHandle==0)
        cout << "Cannot create thread: " << GetLastError() << endl;

    cout << "Start to wait thread to finish "<<endl;
    WaitForSingleObject(countHandle, INFINITE);
    cout << "Thread finished "<<endl;
    system("pause");
}
```



```
c:\Documents and Settings\Orestis Spanos\My Documents\Visual Studio 2005\Projects\thre...
Start to wait thread to finish
I'm alive!
I'm alive!
I'm alive!
Thread finished
Press any key to continue . . .
```

Αναμονή Πολλαπλών Νημάτων

DWORD WaitForMultipleObjects(

DWORD nCount,

const HANDLE* lpHandles,

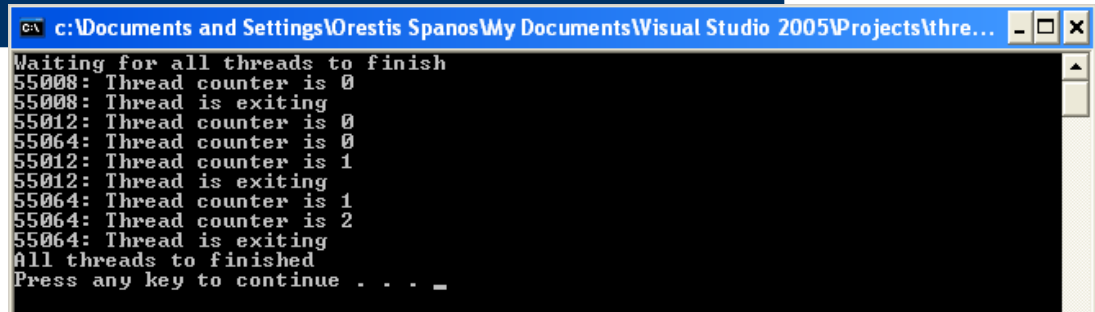
BOOL bWaitAll,

DWORD dwMilliseconds)

- nCount: μέγεθος πίνακα
- lpHandles: πίνακας των handles των threads
- bWaitAll: Boolean αν θέλουμε να επιστρέψει αν τελιώσουν όλα τα thread ή όταν τελιώσει ένα νήμα από αυτά
- dwMilliseconds: Μέγιστος χρόνος για τον οποίο θα περιμένουμε (ms)

Παράδειγμα 3

```
#include <windows.h>
#include <iostream>
#pragma comment(lib,"kernel32.lib")
using namespace std;
void CountThread(LPVOID *iter) {
    int iterNo=(int)iter;
    for (int i = 0; i < iterNo; i++) {
        Sleep(1000);
        cout << GetCurrentThreadId()<<": Thread counter is " << i << endl;
    }
    cout<<GetCurrentThreadId()<<": Thread is exiting"<<endl;
}
int main() {
    HANDLE countHandles[3];
    DWORD threadID;
    for (int i = 0; i < 3; i++) {
        countHandles[i] = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) CountThread,(LPVOID *) (i+1), 0,
            &threadID);
        if (countHandles[i]==0) cout << "Cannot create thread: " << GetLastError() << endl;
        Sleep(100);
    }
    cout<<"Waiting for all threads to finish"<<endl;
    WaitForMultipleObjects(3, countHandles, TRUE, INFINITE);
    cout<<"All threads to finished"<<endl;
    system("pause");
}
```



```
c:\Documents and Settings\Orestis Spanos\My Documents\Visual Studio 2005\Projects\thre...
Waiting for all threads to finish
55008: Thread counter is 0
55008: Thread is exiting
55012: Thread counter is 0
55064: Thread counter is 0
55012: Thread counter is 1
55012: Thread is exiting
55064: Thread counter is 1
55064: Thread counter is 2
55064: Thread is exiting
All threads to finished
Press any key to continue . . . _
```

Άλλες Συναρτήσεις

- `DWORD SuspendThread (HANDLE hThread)`
Σταματά την εκτέλεση ενός νήματος
- `DWORD ResumeThread (HANDLE hThread)`
Συνεχίζει την εκτέλεση ενός νήματος
- `VOID ExitThread (DWORD dwExitCode)`
Αναγκάζει ένα νήμα να τερματίσει με κωδικό `dwExitCode`

Παράδειγμα 4

```
#include <windows.h>
#include <iostream>
#pragma comment(lib,"kernel32.lib")
using namespace std;
volatile INT count;
const INT numThreads=4;
void CountThread(INT iterations)
{
    int temp;
    LONG semaCount;
    for (int i=0; i<iterations; i++) { temp=count; Sleep(10); temp++; count=temp; }
}

int main()
{
    HANDLE handles[numThreads];
    DWORD threadID;
    for (int i=0; i<numThreads; i++)
        handles[i]=CreateThread(0, 0, (LPTHREAD_START_ROUTINE) CountThread,(LPVOID *) 25, 0,
        &threadID);
    WaitForMultipleObjects(numThreads, handles,TRUE, INFINITE);
    cout << "Count = " << count << endl;
    system("pause");
    return 0;
}
```

Θεωρητικά άγνωστο αποτέλεσμα

ΑΝΑΓΚΗ ΓΙΑ ΣΥΓΧΡΟΝΙΣΜΟ

Συγχρονισμός Νημάτων

- Interlocked Συναρτήσεις
- CRITICAL_SECTION
- MUTEX
- SEMAPHORE
- EVENT

Interlocked Συναρτήσεις

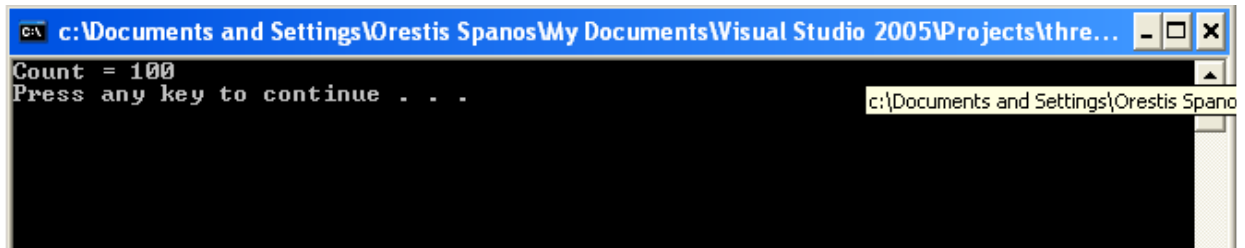
- Είναι υλοποιημένες σε user space
- Λειτουργούν σαν ατομικές εντολές
 - LONG InterlockedIncrement (LONG volatile* Addend);
 - Αύξηση της μεταβλητής Addend κατά 1
 - Επιστρέφει την νέα τιμή της Addend
 - LONG InterlockedDecrement (LONG volatile* Addend);
 - Μείωση της μεταβλητής Addend κατά 1
 - Επιστρέφει την νέα τιμή της Addend
 - LONG InterlockedExchange (LPLONG Target, LONG Value)
 - Ανάθεση Target=Value
 - LONG InterlockedExchangeAdd (PLONG Addend, LONG Increment)
 - Addend=Addend+Increment;
 - LONG InterlockedCompareExchange(LONG volatile* Destination, LONG Exchange, LONG Comparand);
 - If (Destination==Comparand) Destination=Exchange;

CRITICAL_SECTION

- Υλοποιούνται σε User Space
- Αρχικοποίηση **CRITICAL_SECTION**
 - VOID InitializeCriticalSection (LPCRITICAL_SECTION lpCriticalSection)
- Είσοδος στο **CRITICAL_SECTION**
 - VOID EnterCriticalSection (LPCRITICAL_SECTION lpCriticalSection)
- Έξοδος από το **CRITICAL_SECTION**
 - VOID LeaveCriticalSection (LPCRITICAL_SECTION lpCriticalSection)
- Διαγραφή **CRITICAL_SECTION**
 - VOID DeleteCriticalSection (LPCRITICAL_SECTION lpCriticalSection)

Παράδειγμα 5 CRITICAL_SECTION

```
CRITICAL_SECTION cs;
volatile INT count;
const INT numThreads=4;
void CountThread(INT iterations)
{
    int temp;
    LONG semaCount;
    for (int i=0; i<iterations; i++) {
        EnterCriticalSection (&cs); //Enter cs
        temp=count; Sleep(10); temp++; count=temp;
        LeaveCriticalSection(&cs); //Leave cs
    }
}
int main()
{
    HANDLE handles[numThreads];
    DWORD threadID;
    count =0;
    InitializeCriticalSection (&cs); //Initialise cs
    for (int i=0; i<numThreads; i++)
        handles[i]=CreateThread(0, 0, (LPTHREAD_START_ROUTINE) CountThread,(LPVOID *) 25, 0,
        &threadID);
    WaitForMultipleObjects(numThreads, handles,TRUE, INFINITE);
    DeleteCriticalSection(&cs); //Delete cs
    cout << "Count = " << count << endl;
    system("pause");
    return 0;
}
```



```
c:\Documents and Settings\Orestis Spanos\My Documents\Visual Studio 2005\Projects\thre...
Count = 100
Press any key to continue . . .
```

CRITICAL_SECTION (Συνέχεια)

- ΑΡΝΗΤΙΚΑ

- Δεν έχουν ώρα λήξης. Δηλαδή αν ένα νήμα δεν καλέσει το `LeaveCriticalSection` για οποιοδήποτε λόγο τότε τα υπόλοιπα νήματα θα μείνουν μπλοκάρισμένα
- Δεν μπορούν να χρησιμοποιηθούν μεταξύ διεργασιών

MUTEX

- Είναι υλοποιημένα στον πυρήνα
- Μοιράζονται μεταξύ διεργασιών
- Έχουν ώρα λήξης
- Όταν ένα thread που κλείδωσε ένα mutex πεθάνει χωρίς να το ξεκλιδώσει, τότε ξεκλιδώνεται αυτόματα

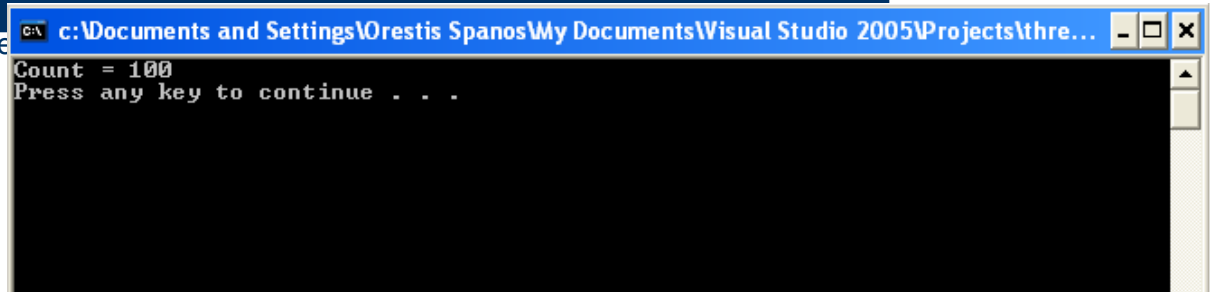
MUTEX (Συνέχεια)

- **Δημιουργία MUTEX**
 - HANDLE CreateMutex (LPSECURITY_ATTRIBUTES lpsa, BOOL bInitialOwner, LPCTSTR lpMutexName)
lpsa: Παράμετροι ασφάλειας mutex
bInitialOwner: TRUE για να αποκτήσουμε τον έλεγχο του mutex μόλις δημιουργηθεί
lpMutexName: Προαιρετικό όνομα του mutex, 0 αν δεν θέλουμε
- **Για να πάρουμε το handle ενός mutex**
 - HANDLE OpenMutex (DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR lpName)
dwDesiredAccess: Δικαιώματα που θέλουμε για το mutex
bInheritHandle: TRUE για να μπορεί το handle να κληρονομηθεί από άλλα process
lpName: Το όνομα του mutex που θέλουμε
- **Για να αποκτήσουμε τον έλεγχο σε ένα Mutex**
 - WaitForSingleObject και WaitForMultipleObjects όπως τα είδαμε πριν
- **Για να ελευθερώσουμε ένα mutex**
 - BOOL ReleaseMutex (HANDLE hMutex)

Παράδειγμα 6 MUTEX

```
HANDLE mutex;           //define the mutex
volatile INT count;
const INT numThreads=4;
void CountThread(INT iterations)
{
    int temp;
    LONG semaCount;
    for (int i=0; i<iterations; i++) {
        WaitForSingleObject (mutex, INFINITE);    //get control of the mutex
        temp=count; Sleep(10); temp++; count=temp;
        ReleaseMutex(mutex);    //release the mutex
    }
}

int main()
{
    HANDLE handles[numThreads];
    DWORD threadID;
    count =0;
    mutex = CreateMutex(0, FALSE, 0);    //create & handle mutex
    for (int i=0; i<numThreads; i++)
        handles[i]=CreateThread(0, 0, (LPTHREAD_START_ROUTINE) CountThread,(LPVOID *) 25, 0,
        &threadID);
    WaitForMultipleObjects(numThreads, handles,TRUE, INFINITE);
    CloseHandle(mutex);    //release the handler
    cout << "Count = " << count << endl;
    system("pause");
    return 0;
}
```



The screenshot shows a Windows command prompt window with the following text:

```
c:\Documents and Settings\Orestis Spanos\My Documents\Visual Studio 2005\Projects\thre...
Count = 100
Press any key to continue . . .
```

SEMAPHORE

- Είναι υλοποιημένα στον πυρήνα
- Δημιουργία **SEMAPHORE**
 - HANDLE CreateSemaphore (LPSECURITY_ATTRIBUTES lpsa, LONG lSemInitial, LONG lSemMax, LPCTSTR lpSemName)
lpsa: Παράμετροι ασφάλειας semaphore
lSemInitial: Αρχική τιμή / lSemMax: Μέγιστη τιμή
lpSemName : Προαιρετικό όνομα του semaphore, 0 αν δεν θέλουμε
- Για να αποκτήσουμε τον έλεγχο σε ένα **SEMAPHORE**
 - WaitForSingleObject και WaitForMultipleObjects όπως τα είδαμε πριν
- Για να ελευθερώσουμε ένα **SEMAPHORE**
 - BOOL ReleaseSemaphore (HANDLE hSemaphore, LONG cReleaseCount, LPLONG lpPreviousCount)
hSemaphore: Ο semaphore που θέλουμε να ελευθερώσουμε
cReleaseCount: Αριθμό semaphores που θέλουμε να ελευθερώσουμε (συνήθως 1)
lpPreviousCount: Επιστρέφει τον προηγούμενο αριθμό του semaphore (άν θέλουμε)

Περιορισμοί σε SEMAPHORE

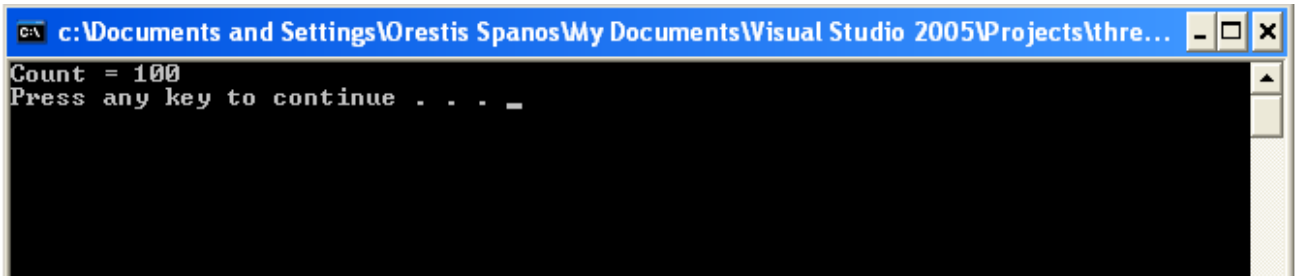
- Μείωση του count του semaphore περισσότερο από 1
 - Δεν μπορούμε με το WaitForMultipleObjects
 - αν στο array υπάρχει το ίδιο handle περισσότερο από μια φορά παίρνουμε error.
 - Αναγκαστικά πρέπει να χρησιμοποιήσουμε διαδοχικές κλήσεις στην συνάρτηση WaitForSingleObject
 - πρόβλημα: το νήμα μπορεί να γίνει preempt, πριν προλάβει να δεσμεύσει τον σωστό αριθμό.
 - Πιθανή λύση: συνδυασμός CRITICAL_SECTION με Semaphores. π.χ.:
 - EnterCriticalSection (&csSem);
 - WaitForSingleObject (hSem, INFINITE);
 - WaitForSingleObject (hSem, INFINITE);
 - LeaveCriticalSection (&csSem);

Παράδειγμα 7 SEMAPHORE

```
HANDLE sema;           //define the handler semaphore
volatile INT count;
const INT numThreads=4;
void CountThread(INT iterations)
{
    int temp;
    LONG semaCount;
    for (int i=0; i<iterations; i++)

        WaitForSingleObject (sema, INFINITE);    //get control of the semaphore
        temp=count; Sleep(10); temp++; count=temp;
        ReleaseSemaphore(sema, 1, &semaCount );    //release the semaphore
    }
}

int main()
{
    HANDLE handles[numThreads];
    DWORD threadID;
    count =0;
    sema = CreateSemaphore(0, 1, 1, 0); //create & handle semaphore
    for (int i=0; i<numThreads; i++)
        handles[i]=CreateThread(0, 0, (LPTHREAD_START_ROUTINE) CountThread,(LPVOID *) 25, 0, &threadID);
    WaitForMultipleObjects(numThreads, handles,TRUE, INFINITE);
    CloseHandle(sema);           //release the handler
    cout << "Count = " << count << endl;
    system("pause");
    return 0;
}
```



The screenshot shows a Windows command prompt window with the following text:

```
c:\Documents and Settings\Orestis Spanos\My Documents\Visual Studio 2005\Projects\thre...
Count = 100
Press any key to continue . . . _
```

EVENT

- Είναι υλοποιημένα στο πυρήνα
- Χρήση για ενημέρωση κάποιου thread που περιμένει για ένα γεγονός
- Δυο είδη
 - manual-reset: ειδοποιούν πολλά threads ταυτόχρονα και μένουν ενεργά μέχρι να απενεργοποιηθούν με κλήση συστήματος
 - auto-reset: ειδοποιούν ένα thread κάθε φορά και απενεργοποιούνται αυτόματα

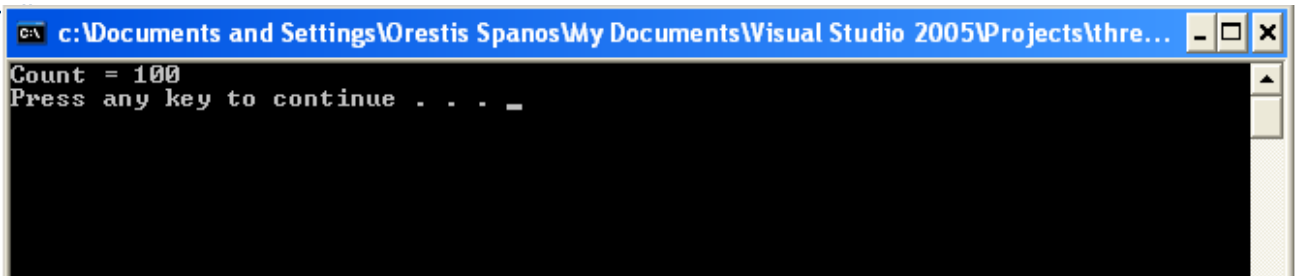
EVENT (Συνέχεια)

- **Δημιουργία event**
 - HANDLE CreateEvent (LPSECURITY_ATTRIBUTES lpsa, BOOL bManualReset, BOOL bInitialState, LPTCSTR lpEventName)
 - lpsa: Παράμετροι ασφάλειας event
 - bManualReset : TRUE για manual-reset event
 - bInitialState: TRUE για ενεργοποιημένο event
 - lpEventName : Προαιρετικό όνομα του event, 0 αν δεν θέλουμε
- **Για να πάρουμε το handle ενός event**
 - HANDLE OpenEvent (DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR lpName)
 - dwDesiredAccess: Δικαιώματα που θέλουμε για το event
 - bInheritHandle: TRUE για να μπορεί το handle να κληρονομηθεί από άλλα process
 - lpName: Το όνομα του event που θέλουμε
- **Ενεργοποίηση / Απενεργοποίηση event**
 - BOOL SetEvent (HANDLE hEvent) / BOOL ResetEvent (HANDLE hEvent)

Παράδειγμα 8 EVENT

```
HANDLE evnt; //define the handle
volatile INT count;
const INT numThreads=4;
void CountThread(INT iterations)
{
    int temp;
    LONG semaCount;
    for (int i=0; i<iterations; i++)
        WaitForSingleObject (evnt, INFINITE); //get control of the event
        temp=count; Sleep(10); temp++; count=temp;
        SetEvent(evnt); //release the event
    }
}

int main()
{
    HANDLE handles[numThreads];
    DWORD threadID;
    count =0;
    evnt = CreateEvent(0, FALSE, TRUE, 0); //create & handle event
    for (int i=0; i<numThreads; i++)
        handles[i]=CreateThread(0, 0, (LPTHREAD_START_ROUTINE) CountThread,(LPVOID *) 25, 0,
        &threadID);
    WaitForMultipleObjects(numThreads, handles,TRUE, INFINITE);
    CloseHandle(evnt); //release the handler
    cout << "Count = " << count << endl;
    system("pause");
    return 0;
}
```



The screenshot shows a Windows command prompt window with the following text:

```
c:\Documents and Settings\Orestis Spanos\My Documents\Visual Studio 2005\Projects\thre...
Count = 100
Press any key to continue . . . _
```

Βιβλιογραφία

- Wikipedia
- Microsoft Co., (2007). MSDN Documentation.
<http://msdn2.microsoft.com/en-us/default.aspx>
- Windows Threads and Concurrency Presentation
 - Vassos Tziongouros
 - Christos Constantinou