6 Übungsreihe „Windows: Nebenläufigkeit & Co.“

6.1 Übung 1: Kennenlernen von einzelnen Kommandos in der Shell

6.1.1 Lernziel

- ☞ Kennenlernen der Shell Kommandos im Command Prompt in Windows
- ☞ Kennenlernen der Shellscripte im Command Prompt in Windows

6.1.2 Literatur im WWW

- ☞ http://technet.microsoft.com/de-de/library/cc778084%28WS.10%29.aspx
- ☞ http://msdn.microsoft.com/en-us/library/windows/desktop/hh448382%28v=vs.85%29.aspx
- ☞ http://ss64.com/nt/
- ☞ Dokument im moodle

6.1.3 Aufgaben

6.1.1 Öffnen Sie in Windows das *Command Prompt*. Gehen Sie mit dem Kommando *d:* in die Partition. Legen Sie mit *mkdir mein_name* Ihren eigenen Folder an und gehen Sie mit cd *mein_name* in den Folder. x
Rufen Sie *edit test* auf und tragen Sie in das File ein:

```
Katze
Maus
Hund
Ratte
Elefant
```

Speichern Sie und schließen Sie das File wieder.

6.1.2 Suchen Sie jeweils das Gegenstück in Linux-Kommandos in Windows und probieren Sie sie aus. Sie können dafür die Tabelle „Befehle Windows Shell“ verwenden: x

LINUX	**WINDOWS**
ls	dir
cd	cd
date	date
mkdir	mkdir
rmdir	rmdir
who	whoami
cat test	type
move test test1	move
rm test1	del
echo	echo

6.1.3 Probieren Sie die folgenden Befehle aus und erklären Sie, was sie tun: x

route print	Routingtabelle
tree	Verzeichnisbaum

ver	
start	
cls	
getmac	
type test | sort	
type test | sort > susi	
sort < test > hans	
ping 192.168.1.0	
find „Kat“ test	

6.1.4 Öffnen Sie ein File test1.bat und speichern Sie x

```
echo Hallo World
```

Starten Sie den File in der Shell mit test1.
Was ist hier eher überflüssig?

Erzeugen Sie einen File test2.bat mit folgendem Inhalt: x

```
@echo off
echo Hallo World
```

Starten Sie den File in der Shell mit test2.

Erzeugen Sie einen File test3.bat mit folgendem Inhalt: x

```
@echo off
FOR /L %%i IN (3 20 104) DO (
   echo ***
   echo %%i
   echo ***
)
```

Starten Sie den File in der Shell mit test3. Was bedeutet der Ausdruck (3 20 104)?

Erzeugen Sie einen File test4.bat mit folgendem Inhalt: x

```
@echo off
if [%1] GTR [%2]  echo %1 ist größer als %2
if [%1] LSS [%2]  echo %1 ist kleiner als %2
if [%1] EQU [%2]  echo %1 ist gleich als %2
```

Dann starten Sie ihn mit z.B. test4 5 8 oder test4 3 3

Erzeugen Sie einen File test5.bat mit folgendem Inhalt (Alle Zeichen werden benötigt): x

```
rem -- Ausgabedatei
set htmlout=%temp%\logfile.html

rem -- HTML generieren
echo ^<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"^> >%htmlout%
echo ^<HTML^>^<HEAD^>                                    >>%htmlout%
echo ^<TITLE^>HTML-Ausgabe-Test^</TITLE^>                >>%htmlout%
echo ^</HEAD^>^<BODY^>                                   >>%htmlout%
echo ^<H1^>HTML-Test^</H1^>                              >>%htmlout%
date /t                                                  >>%htmlout%
time /t                                                  >>%htmlout%
echo ^<HR /^>                                            >>%htmlout%
echo Hello world.^<HR /^>                                >>%htmlout%
echo ^</BODY^>^</HTML^>                                  >>%htmlout%

rem -- Browser starten
start %htmlout%
```

Starten Sie den File in der Shell mit test5. Was passiert?

Es ist unklar, ob die Shell unter Windows genauso mächtig ist wie die shells unter Linux. Eventuell liegt es daran, dass es weniger Freaks gibt, die es ausprobieren und ihre Ergebnisse ins Netz stellen. Gerade das letzte Beispiel zeigt, dass viel möglich ist, wenn man weiß, wie es geht.

6.2 Übung 2: Erzeugung von Prozessen und Threads

6.2.1 Lernziel

☞ Erzeugung von Prozessen und Threads

6.2.2 Aufgaben

6.2.1 Öffnen Sie Microsoft Visual Studio x
Öffnen Sie Datei/Neu/Project
Öffnen Sie Visual C++/Win32-Konsolenanwendung mit
Name: u6_01
Ort: D:/mein_Name
Fertigstellen
Füllen Sie u6_01.cpp mit

```
#include "stdafx.h"
#include "stdio.h"


int main(int argc, _TCHAR* argv[])
{
    printf("Hallo World");
    return 0;
}
```

Kompellieren Sie das Programm mit Erstellen/Projektmappe erstellen.
Führen Sie das Programm in der Shell (Command mit u6_01 aus. Sie finden das Program als …exe unter Debug.

6.2.2 Erzeugen eines neues Projekt u6_02 mit: x

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

```
int main(int argc, _TCHAR* argv[])
{

    STARTUPINFO                 SUInfo;
    PROCESS_INFORMATION         PInfo;

    // Start-Up-Info auf 0
    ZeroMemory( &SUInfo, sizeof( SUInfo ));
    SUInfo.cb=sizeof( SUInfo );

    printf("Folgender Prozess wurde gestartet: \n ");

    CreateProcess(
    _T("u6_01.exe"),      // Modulname
    NULL,    // Kommando-Zeile
    NULL,    // Prozess-Handle nicht vererbbar
    NULL,    // Thread-Handle nicht vererbbar
    FALSE,   // Handle.Verebung auf FALSE
    0,       // Normale Priorität
    NULL,    // Eltern-Evironment-Block verwenden
    NULL,    // Elternverzeichnis als Startverzeichnis
    &SUInfo, // Zeiger auf Startup-Struktur
    &PInfo   // Zeiger auf Process-Informations-Struktur
    );

    printf("Alles ok, PID=%d\n", PInfo.dwProcessId);

     // Close process and thread handles.
    CloseHandle( PInfo.hProcess );
    CloseHandle( PInfo.hThread );

    return 0;
}
```

Kopieren Sie die u6_01.exe in den Debug-Folder von u6_02.
Führen Sie u6_02 aus.
Mehr Information zu der Funktion zur Prozesserzeugung z.B. unter
https://msdn.microsoft.com/en-us/library/aa908775.aspx

6.2.3 Fügen Sie jetzt in die Programme u6_01 und u6_02 die Zeilen (jeweils nach den printf) x

```
int a = 5;
while (a == 5);
```

ein und kompellieren Sie den Code. Kopieren Sie die u6_01.exe in den Debug-Folder von u6_02. Führen Sie u6_02 aus.

Rufen Sie den Windows Task Manager auf und suchen Sie die Prozesse u6_01 und u6_02.
Notieren Sie die folgenden Größen

-	u6_01	u6_02
CPU Auslastung in %		
Arbeitspeicher		
Handles		
Threads		
Basispriorität		

Beenden Sie die beiden Prozesse im Task Manager.

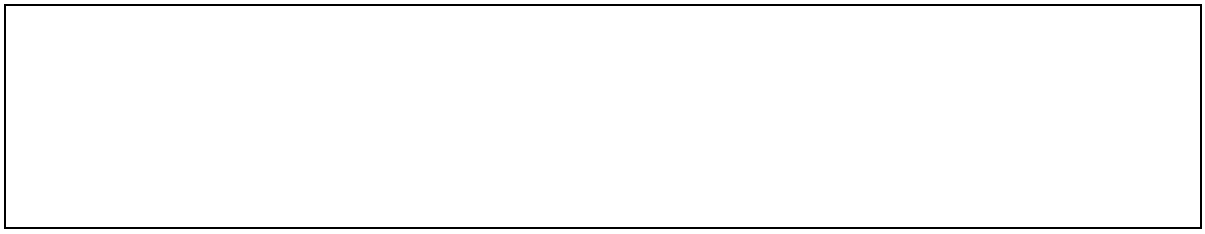
6.2.4 Erzeugen eines neues Projekt u6_03 mit dem Inhalt von Projekt u6_02.
Löschen Sie den Teil „Close process and thread handles“.
Kopieren Sie

```
PROCESS_INFORMATION          PInfo;
```

5mal und benennen Sie PInfo um zu PInfo1 … PInfo6
Kopieren Sie die Prozesserzeugung mit dem nachfolgenden printf 5mal hintereinandern.
Ändern Sie jeweils PInfo zu PInfo1 bzw. PInfo2 u.s.w.
Setzen Sie in einem Prozess die Normale Priorität „0“ auf „HIGH_PRIORITY_CLASS“
Führen Sie u6_03 aus.

Kopieren Sie die Prozesserzeugung
Rufen Sie den Windows Task Manager auf und suchen Sie die Prozesse u6_01 und u6_03.

Was beobachten Sie:



Beenden Sie die Prozesse im Task Manager.

6.2.5 Erzeugen eines neues Projekt u6_04 mit: x

```
#include "stdafx.h"
#include <iostream>
#include <windows.h>
using namespace std;

int a;
int b;

DWORD WINAPI ThreadProc( LPVOID pvoid )
{
    // Übergebenen int-Parameter holen
    int* pid = (int*)pvoid;
    int id = *pid;

    for( int i = 0; i < 1000000; i++ ) {
        a = (int) id;
        b = (int) id;
        if ( a != b)
        {
             printf("Fehler: a = %d und b = %d \n", a, b);
        }
    }
    return(0);
}

void main()
{
    // Threads erzeugen
    cout << "Thread1 startet..."<<endl;
    DWORD dwThreadParam1 = 1;
    HANDLE hThread1 = CreateThread (NULL,0,ThreadProc,&dwThreadParam1,0,NULL);

    cout << "Thread2 startet..."<<endl;
    DWORD dwThreadParam2 = 2;
    HANDLE hThread2 = CreateThread (NULL,0,ThreadProc,&dwThreadParam2,0,NULL);

    // Auf Threads warten
    WaitForSingleObject( hThread1, INFINITE );
    WaitForSingleObject( hThread2, INFINITE );

    // Thread-Handles schließen
    cout << "Thread1 schliesst..."<<endl;
    CloseHandle( hThread1 );
    cout << "Thread2 schliesst..."<<endl;
```

```
    CloseHandle( hThread2 );
}
```

Starten Sie den Prozess. Was beobachten Sie? (Sie können den Prozess notfalls mit *Strg C* abbrechen.)

Ersetzen Sie in der Schleife i++ durch i (damit die Schleife nicht abbricht). Rufen Sie den Windows Task Manager auf und suchen Sie die Prozesse u6_04. Notieren Sie die folgenden Größen

-	**u6_04**
CPU Auslastung in %	
Arbeitspeicher	
Handles	
Threads	

Beenden Sie den Prozess im Task Manager.

6.3 Übung 3: Synchronisation

6.3.1 Lernziel

☞ Kennenlernen von Mutexen, Semaphoren und Wartefunktionen

6.3.2 Aufgaben

6.3.1 Erzeugen Sie ein neues Projekt u6_05 und fügen den Code aus u6_04 (incl. i++) ein. x

Fügen Sie vor *ThreadProc* ein

```
    // Mutex-Handle definieren und Mutex erzeugen
    HANDLE mutex = CreateMutex( NULL, FALSE, NULL );
```

Fügen Sie am Ende des Hauptprogramms ein

```
    // Mutex zerstören
    CloseHandle( mutex );
```

Schützen Sie die Zuweisung von a und b und den print-Befehl durch eine Mutex.

```
    // Mutex setzen
    WaitForSingleObject( mutex, INFINITE );
.....
.....
.....
    // Mutex wieder freigeben
    ReleaseMutex( mutex );
```

Starten Sie den Prozess. Was beobachten Sie jetzt?

6.3.2 Erzeugen Sie ein neues Projekt u6_06 und führen es aus. x

```
// Es gibt in der Semaphor 8 freie Plätze, //
// sodass von den 12 Threads 4 warten müssen //

#include "stdafx.h"
#include <iostream>
#include <windows.h>
using namespace std;

#define MAX_SEM_COUNT 8
#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

    // 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,        // thread function arguments
                     0,          // default creation flags
                     &ThreadID); // receive thread identifier
    }

    // Wait for all threads to terminate

    WaitForMultipleObjects(THREADCOUNT, aThread, TRUE, INFINITE);

    // Close thread and semaphore handles

    printf("Alle Threads fertig \n");

    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: arbeitet jetzt\n", GetCurrentThreadId());
                    bContinue=FALSE;

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

                 printf(" Thread %d: ist fertig mit arbeiten\n", GetCurrentThreadId());
                    // Release the semaphore when task is finished

                    ReleaseSemaphore(
                           ghSemaphore,  // handle to semaphore
                           1,            // increase count by one
                           NULL);        // not interested in previous count

                    break;

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

Versuchen Sie das Programm im Detail zu verstehen. Führen Sie eine Art Strichliste, welche Threads laufen können, welche warten müssen, welche fertig sind und welche später laufen.
Falls Sie nicht alles verstehen, spielen Sie z.B. an der Anzahl der Threads und MAX_SEM_COUNT.

6.3.3 Wenn Sie noch Zeit haben, gehen Sie auf
http://msdn.microsoft.com/en-us/library/windows/desktop/ms687008%28v=vs.85%29.aspx
Dort finden Sie viele weitere Beispiele für Synchronisation unter Windows.

6.4 Übung 4: Kommunikation

6.4.1 Lernziel

☞ Verwendung von Pipes

6.4.2 Aufgaben

6.4.1 Erstellen Sie ein neues Projekt u6_07a und kompellieren Sie es: x

```
#include "stdafx.h"
#include <iostream>
#include <windows.h>

int main(void)
{
    HANDLE          hIn;
    DWORD           dwBytesRead;
    char            buf[100];
    LPTSTR lpszPipename = TEXT("\\\\.\\pipe\\mynamedpipe");

    hIn = CreateNamedPipe(lpszPipename,       // Name
        PIPE_ACCESS_DUPLEX | WRITE_DAC, // OpenMode
        PIPE_TYPE_BYTE | PIPE_READMODE_BYTE | PIPE_WAIT, // PipeMode
        2, // MaxInstances
        1024, // OutBufferSize
        1024, // InBuffersize
        2000, // TimeOut
        NULL); // Security

    if (hIn == INVALID_HANDLE_VALUE)
    {
```

```
            printf("Could not create the pipe\n");
            exit(1);
        }
        printf("hIn=%p\n", hIn);
        printf("connect...\n");
        ConnectNamedPipe(hIn, NULL);
        printf("...connected\n");

        for (;;)
        {
            if (!ReadFile(hIn, buf, sizeof(buf), &dwBytesRead, NULL))
            {
                        printf("ReadFile failed -- probably EOF\n");
                        break;
            }

        buf[dwBytesRead] = '\0';
        printf("read [%s]\n", buf);
       }

       DisconnectNamedPipe(hIn);

       CloseHandle(hIn);
    }
```

6.4.2 Erstellen Sie ein neues Projekt u6_07b und kompellieren Sie es: x

```
#include "stdafx.h"
#include <iostream>
#include <stdio.h>
#include <windows.h>

void main()
{
    HANDLE          hOut;
    char            buf[1024];
    DWORD           len;
    DWORD           dwWritten;
    LPTSTR lpszPipename = TEXT("\\\\.\\pipe\\mynamedpipe");

    printf_s("pwrite: waiting for the pipe...\n");
    if (WaitNamedPipe(lpszPipename, NMPWAIT_WAIT_FOREVER) == 0)
    {
             printf("WaitNamedPipe failed. error=%d\n", GetLastError());
        return;
    }
    printf("pwrite: the pipe is ready\n");

    hOut = CreateFile(lpszPipename,
        GENERIC_WRITE,
        0,
        NULL, OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);
    if (hOut == INVALID_HANDLE_VALUE)
    {
        printf("CreateFile failed with error %d\n", GetLastError());
        return;
    }
    printf("Opened the pipe\n");

    for (int i = 0; i < 5; i++)
    {
        sprintf_s(buf, "This is test line %d so there.", i);
        len = strlen(buf);
        printf("Sending [%s]\n", buf);
        if (!WriteFile(hOut, buf, len, &dwWritten, NULL))
        {
             printf("WriteFile failed\n");
             break;
        }

        Sleep(1000);
    }
```

```
    CloseHandle(hOut);
    printf("pwrite: done\n");
}
```

6.4.3 Öffnen Sie einen Command Prompt und starten Sie u6_07a.
Öffnen Sie einen zweiten Command Prompt und starten Sie u6_07b und beobachten Sie das Ergebnis.
Versuchen Sie das Programm im Detail zu verstehen. Versuchen Sie zu verstehen, wie die Pipe funktioniert. Was wird abgeschickt und was kommt an?
Falls Sie nicht alles verstehen, modifizieren Sie das Program.

6.5 Übung 5: Selbst zuentwickelte Programme

6.5.1 Aufgaben

6.5.1 Legen das Projekt u6_08 an. Hier sollten Sie noch eine eigene Idee verwirklichen. x