

Software Entwurf 4

Übung 2

Sommersemester 2013

(1) Name: _____
Mat.Nr.: _____

Abgabetermin: KW 15 - **Mi, 10.4.2013**

Punkte: _____

korrigiert: _____

(2) Name: _____
Mat.Nr.: _____
Übungsgruppe: _____
Aufwand in Mh: _____

Beispiel 1 (12 Punkte) Parallelisierung: Erstellen Sie ein Multithreaded-Programm, mit dem Sie vier beliebig große Textdateien in eine einzelne, sortierte Gesamtdatei zusammenführen können. Abb 1 veranschaulicht den prinzipiellen Ablauf: Die Dateien werden eingelesen, in Teilen sortiert und die Gesamtdatei wieder auf die Festplatte geschrieben.

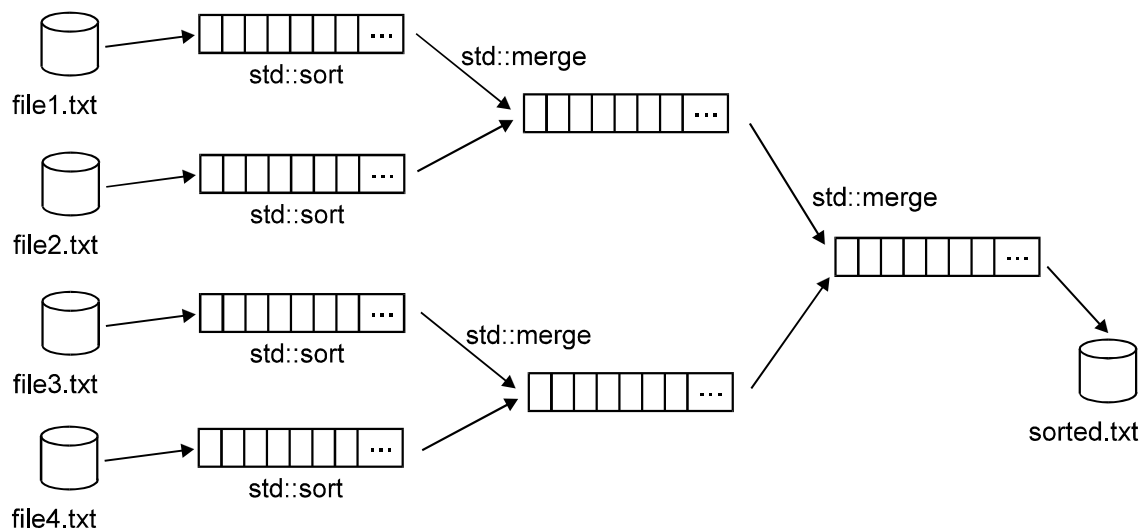


Abb 1: Prinzip einer parallelen Dateisortierung

Überlegen Sie sich ein effizientes Parallelisierungsmodell, welches durchaus abweichend von Abb 1 arbeiten kann. Wählen Sie geeignete Container und Sortieralgorithmen (`std::sort`, `std::merge`, `std::inplace_merge`). Dokumentieren Sie ihr Modell in der Funktionsbeschreibung.

Implementieren Sie zum Vergleich eine *Singlethreaded-Version*, die ebenso die 4 Dateien effizient sortiert und das Ergebnis wieder speichert.

Testen Sie beide Versionen mit den Textdateien auf dem LVA-Laufwerk und stellen Sie die Ergebnisse (hinsichtlich *Geschwindigkeit* und *Speicherbedarf*) übersichtlich dar! Geben Sie den *Speedup* an und diskutieren Sie das Ergebnis.

Hinweise:

- Führen Sie die Geschwindigkeitstests jeweils mit der **Release-Version** durch.
- Ändern Sie den Umfang der Textdateien wenn nötig, damit Sie **Laufzeiten** im Sekundenbereich bekommen.
- Übersetzen Sie ihr paralleles Programm mit den **Multithreaded Bibliotheken** (Project Properties: C/C++ -> Code Generation -> Runtime Library: Multi-threaded DLL).
- Die **Terminierung von Threads** können Sie mit den API-Calls `WaitForSingleObject()` bzw. `WaitForMultipleObjects()` abwarten (siehe dazu die MSDN-Hilfe).
- Verwenden Sie die **Threadklasse** aus der Übung!

Beispiel 2 (12 Punkte) Admin-Tool: Implementieren Sie ein Administrationswerkzeug, das über folgende Funktionen verfügt:

1 Applikationen starten: Via Menüauswahl im Testtreiber können ein *Editor* und der *Taschenrechner* gestartet werden - auch mehrmals, zB. `notepad.exe` und `calc.exe`.

2 Prozesse auflisten: Alle Prozesse im System werden formatiert (`setw()`, `setfill()`, `left`, etc) aufgelistet (sofern die Zugriffsrechte existieren), zB:

Process	PID	BasePriority	Thread Count
-----	-----	-----	-----
calc.exe	123	normal	1
notepad.exe	233	high	1
Mcshield.exe	215	low	18
calc.exe	317	low	1
Acrobat.exe	127	normal	4
...			

3 Bestimmte Prozesse terminieren: Die über die PID angegebene Applikation wird terminiert, sofern die Rechte bestehen.

4 Systeminformation anzeigen: Hier wird eine Information über das aktuelle System ausgegeben (Hardware und Software). Sie dürfen sich hier auf die Intel-Architektur beschränken. Geben Sie also *ProcessorLevel* (Family) und *ProcessorRevision* (Model und Stepping) nur aus, wenn es sich um Intel-CPUs handelt, ansonsten „unknown“. Bei der Software beschränken Sie sich auf den Computernamen und die Betriebssystemversion. Eine Ausgabe kann etwa wie folgt aussehen:

```
System Information <HARDWARE>:
-----
Processor Type           : 586
Architecture            : Intel CPU or compatibles
Processor Level (Family): 6
Model and Stepping MMSS : 0x905
Number of Processors     : 1
```

```
Features:
    Physical Address Extension not supported.
    Virtualization enabled.
```

```
System Information <SOFTWARE>:
-----
Computer Name:   Zeus
OS Version      : Major: 6 Minor: 1
OS Revision     : Service Pack 1
```

Zur Realisierung:

Analyse: Analysieren Sie zuerst die Anforderungen und berücksichtigen Sie dabei, wo und wie ihnen die unten angeführten API-Calls helfen.

Design: Diskutieren Sie ihr Design und legen Sie die Datenstrukturen und Schnittstellen fest (Module & Funktionen bzw. Klassen & Methoden inkl. Parameter).

Implementierung: Implementieren Sie ihre Lösung entweder mit Modulen - oder besser - mit Klassen (OOP).

Test: Erstellen Sie weiters einen Testtreiber, der diese Funktionen über ein einfaches Menü anbietet.

Dokumentation: Verfassen Sie eine Funktionsbeschreibung in LaTeX. Konzentrieren Sie sich dabei auf das Wesentliche.

Hinweise: Dieses Beispiel hat den Zweck, sich mit der Funktionsweise der Win32/64-Schnittstelle vertraut zu machen. Benützen Sie die MSDN (Dokumentation zur Platform SDK), um sich mit den API-Calls und den Strukturen vertraut zu machen. Beginnen Sie bei folgenden API-Calls:

```
GetCurrentProcess(), GetSystemInfo(), GetComputerName(), GetVersionEx(),
Process32First(), Process32Next(), OpenProcess(), TerminateProcess()
```

Testen Sie ihr Programm ausführlich und geben Sie diese Tests auch ab!

Hinweise zur Ausarbeitung dieser Übung:

Lösen Sie diese Übung im Teamwork (2 Personen) und geben Sie eine gemeinsame Ausarbeitung ab. Setzen Sie dabei an den entscheidenden Stellen **Pair Programming** ein, einer Technik aus dem Software Engineering Konzept *Extreme Programming* (XP) von *Kent Beck*. Dabei sitzt das Teammitglied mit der geringeren Erfahrung am PC und codiert, während das zweite Teammitglied unterstützend und erklärend einwirkt und auch prüft, ob der geschriebene Code funktionieren kann (permanentes Review).

Sie können nach eigenem Ermessen im konventionellen Software Lifecycle arbeiten, oder weiter im XP-Ansatz vorgehen.

- In XP schreiben Sie **zuerst den Testtreiber** und werden sich dadurch bewusst, *welche Funktionalität* Sie benötigen und *in welcher Weise* Sie diese anwenden wollen.
- Danach beginnt das Design mit dem Fokus auf **Einfachheit** („*the simplest thing that could possibly work*“).
- Die Implementierung wird in möglichst **kurzen Iterationen** immer ins Gesamtsystem integriert, um fehlende Übereinstimmungen möglichst früh zu erkennen. Sie können damit keine Funktionalitäten vergessen, weil Sie spätestens der nächste Testdurchlauf darauf hinweist.
- Da Testen zu hoher Produktstabilität führt, wird zwischendurch **immer wieder getestet** (*unit testing* durch SW-Entwickler und *functional testing* durch Kunden).

Allgemeine Hinweise: Legen Sie bei der Erstellung Ihrer Übung großen Wert auf eine saubere Strukturierung und auf eine sorgfältige Ausarbeitung! Dokumentieren Sie alle Schnittstellen und versehen Sie Ihre Algorithmen an entscheidenden Stellen ausführlich mit Kommentaren! Testen Sie ihre Implementierungen ausführlich! Geben Sie den Testoutput mit ab! Erstellen Sie eine Funktionsbeschreibung in LaTeX und dokumentieren Sie, **wer im Team welche Bereiche ausgearbeitet hat** - pro Beispiel!

1 Beispiel 1

1.1 Funktionsbeschreibung

Es gibt 4 verschiedene Arten von Workerthreads:

- Datei zu Liste
- Liste zu Datei
- Liste sortieren
- 2 Listen zusammenfügen

Es gibt für jede Arbeit einen eigenen Thread, dadurch ist die Software übersichtlicher und leichter erweiterbar. Zum Beispiel beliebige Anzahl von Eingangsdateien oder auch Ausgangsdateien. Der Thread der die letzte Liste in die Datei schreibt hätte auch wie die singlethreaded Version implementiert werden können und wäre dadurch auch schneller gewesen. Im Mainthread werden die Handles in Vektoren verwaltet. Von dort werden sie den Threads gegeben. Die Threads warten auf die Threads von denen sie die Handles haben und schließen diese danach.

Die Daten der Dateien werden in 4 Listen gespeichert. Diese 4 Listen werden sortiert. Danach werden 2 mal 2 Listen zusammengefügt und wieder danach werden die letzten 2 Listen zusammengefügt. Die letzte Liste wird wieder in die Datei geschrieben.

Anmerkung: Es wurden keine Fehlerfälle im Testtreiber getestet, weil diese durch Exceptions abgefangen werden würden.

Aufgabenaufteilung: Beispiel 1: Bernhard Selymes, Beispiel 2: Reinhard Penn

1.2 Schnittstellen

Name: SortMergeMT(...) und SortMergeST(...)

Parameter: vector<string> const& filenames, string const& filenameOutput

Vektor wegen Erweiterbarkeit und einfacherer Handhabung.

Schnittstellen der Threads: Bei der Init Funktion werden jeweils die benötigten Listen und Handles mitgegeben. Die Listen via Pointer, weil sonst die ganze Liste kopiert werden müsste und auch die falsche Liste verändert werden würde. Diese Parameter werden in den Threads jeweils gespeichert.

1.3 Singlethreaded Version

Bei dieser Version werden die Daten aller Dateien gleich in eine Liste geschrieben. Diese Liste wird dann sortiert und wieder in eine Datei geschrieben.

Speedup:

Speedup = $5.885/5.695 = 1.0334$

Die längste Zeit dauert das Schreiben auf die Festplatte.

1.4 Sourcecode

../Uebung02/Bsp01/StopWatch.h

```
1 #ifndef STOPWATCH_H
2 #define STOPWATCH_H
3
4
5 namespace stw {
6     void Start ();
7     double Stop (); // Returns seconds.
8 }
9
10
11 #endif // STOPWATCH_H
```

../Uebung02/Bsp01/StopWatch.cpp

```
1 #include <ctime>
2 #include "StopWatch.h"
3
4 clock_t StartTime = -1;
5
6 void stw::Start () {
7     StartTime = clock ();
8 }
9
10 double stw::Stop () {
11     clock_t Elapsed = 0;
12     clock_t Stop = 0;
13
14     if (StartTime > -1) {
15         Stop = clock ();
16         Elapsed = Stop - StartTime;
17         StartTime = -1;
18     }
19
20     return static_cast <double> (Elapsed) / static_cast <double> (
        CLOCKS_PER_SEC);
21 }
```

../Uebung02/Bsp01/ThreadBase.h

```
1 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2 // Workfile : ThreadBase.h
3 // Author : Reinhard Penn, Bernhard Selymes
4 // Date : 09.04.2013
5 // Description : Header of ThreadBase.cpp
6 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
7
8 #ifndef ThreadBase_INCLUDED
9 #define ThreadBase_INCLUDED
10
11 #include <Windows.h>
12 #include <list>
13 #include <string>
14
15 typedef std::list<std::string> TStringList;
16
17 class ThreadBase {
18 public:
19     ThreadBase();
```

```

20     virtual ~ThreadBase();
21
22     // THE thread function
23     virtual int Run() = 0;
24     // control thread behaviour
25     virtual void Start() const;
26     virtual void Stop() const;
27
28     // state check:
29     virtual bool IsThreadCreated() const;
30
31     // this handle is even valid after destroying this thread object
32     // call CloseHandle() after usage!
33     virtual HANDLE GetDuplicateHdl() const;
34
35 protected:
36     // internal thread handle => derived classes only
37     virtual HANDLE GetThreadHdl() const;
38
39     ThreadBase(ThreadBase const&);
40     ThreadBase & operator = (ThreadBase const &);
41
42 private:
43     static unsigned long WINAPI ThreadFunc(void * Param);
44
45     HANDLE ThreadHdl;
46     unsigned long ThreadId;
47
48 };
49
50 #endif

```

../Uebung02/Bsp01/ThreadBase.cpp

```

1  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2  // Workfile : ThreadBase.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 09.04.2013
5  // Description : Threadbase
6  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
7
8  #include <cassert>
9  #include "ThreadBase.h"
10
11 ThreadBase::ThreadBase() {
12     ThreadHdl = CreateThread(0, 0, ThreadFunc, this, CREATE_SUSPENDED, &
13         ThreadId);
14
15     assert(ThreadHdl != 0);
16 }
17
18 ThreadBase::~ThreadBase() {
19     CloseHandle(ThreadHdl);
20 }
21
22 unsigned long WINAPI ThreadBase::ThreadFunc(void * Param) {
23     assert(Param != 0);
24
25     ThreadBase * pObj = static_cast <ThreadBase *> (Param);
26     pObj->Run();

```

```

27     return 0;
28 }
29
30 void ThreadBase::Start() const {
31     ResumeThread(ThreadHdl);
32 }
33
34 void ThreadBase::Stop() const {
35     SuspendThread(ThreadHdl);
36 }
37
38 bool ThreadBase::IsThreadCreated() const {
39     return ThreadHdl != 0;
40 }
41
42 // internal handle
43 HANDLE ThreadBase::GetThreadHdl() const {
44     return ThreadHdl;
45 }
46
47 // duplicate handle
48 // closehandle!!!
49 HANDLE ThreadBase::GetDuplicateHdl() const {
50     HANDLE hdlDup = 0;
51     BOOL res = DuplicateHandle(GetCurrentProcess(), ThreadHdl,
52                               GetCurrentProcess(),
53                               &hdlDup, 0, FALSE, DUPLICATE_SAME_ACCESS);
54     assert(res != 0);
55     return hdlDup;
56 }

```

../Uebung02/Bsp01/ThreadFileToList.h

```

1  //////////////////////////////////////////////////
2  // Workfile : ThreadFileToList.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 09.04.2013
5  // Description : Header of ThreadFileToList.cpp
6  //////////////////////////////////////////////////
7
8  #ifndef THREADFILETOLIST_H
9  #define THREADFILETOLIST_H
10
11 #include "ThreadBase.h"
12
13 class ThreadFileToList : public ThreadBase {
14
15 public:
16     ThreadFileToList();
17     virtual ~ThreadFileToList();
18
19     virtual void Init(std::string const& filename, TStringList * const l1);
20
21     virtual int Run();
22
23 private:
24     bool IsInitialized;
25     std::string mFilename;
26     TStringList* mL1;
27 };
28

```


29 **#endif**

../Uebung02/Bsp01/ThreadFileToList.cpp

```
1  ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2  // Workfile : ThreadFileToList.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 09.04.2013
5  // Description : Thread that writes data from a file into a list
6  ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
7
8  #include <cassert>
9  #include <fstream>
10 #include "ThreadFileToList.h"
11
12 ThreadFileToList::ThreadFileToList() {
13     assert(ThreadBase::IsThreadCreated());
14     IsInitialized = false;
15 }
16
17 ThreadFileToList::~ThreadFileToList() {
18 }
19 }
20
21 void ThreadFileToList::Init(std::string const& filename, TStringList *
    const l1)
22 {
23     mFilename = filename;
24     mL1 = l1;
25     IsInitialized = true;
26 }
27
28 int ThreadFileToList::Run() {
29     assert(IsInitialized);
30     if (!IsInitialized) {
31         return 0;
32     }
33
34     std::ifstream fs;
35     fs.open(mFilename);
36
37     if (fs == 0)
38     {
39         std::string error = "Error in ThreadFileToList::Run: could not open
            stream";
40         throw (error);
41     }
42
43     std::string line;
44
45     while (!fs.eof())
46     {
47         std::getline(fs, line);
48         mL1->push_back(line);
49     }
50
51     fs.close();
52
53     delete this;
54
55     return 0;
```

56 }

../Uebung02/Bsp01/ThreadListToFile.h

```
1 //////////////////////////////////////////////////
2 // Workfile : ThreadListToFile.h
3 // Author : Reinhard Penn, Bernhard Selymes
4 // Date : 09.04.2013
5 // Description : Header of ThreadListToFile.cpp
6 //////////////////////////////////////////////////
7
8 #ifndef THREADLISTTOFILE_H
9 #define THREADLISTTOFILE_H
10
11 #include "ThreadBase.h"
12
13 class ThreadListToFile : public ThreadBase {
14
15 public:
16     ThreadListToFile();
17     virtual ~ThreadListToFile();
18
19     virtual void Init(std::string const& filename, TStringList * const l1,
20                     HANDLE hdl);
21
22     virtual int Run();
23
24 private:
25     bool IsInitialized;
26     std::string mFilename;
27     TStringList* mL1;
28     HANDLE mHdl;
29 };
30 #endif
```

../Uebung02/Bsp01/ThreadListToFile.cpp

```
1 //////////////////////////////////////////////////
2 // Workfile : ThreadListToFile.cpp
3 // Author : Reinhard Penn, Bernhard Selymes
4 // Date : 09.04.2013
5 // Description : Thread that writes a list into a file
6 //////////////////////////////////////////////////
7
8 #include <cassert>
9 #include <fstream>
10 #include <iostream>
11 #include <algorithm>
12 #include "ThreadListToFile.h"
13
14 ThreadListToFile::ThreadListToFile() {
15     assert(ThreadBase::IsThreadCreated());
16     IsInitialized = false;
17 }
18
19 ThreadListToFile::~ThreadListToFile() {
20
21 }
22
23 void ThreadListToFile::Init(std::string const& filename, TStringList *
24                             const l1, HANDLE hdl)
```

```

24 {
25     mFilename = filename;
26     mL1 = l1;
27     mHdl = hdl;
28     IsInitialized = true;
29 }
30
31 int ThreadListToFile::Run() {
32     assert(IsInitialized);
33     if (!IsInitialized) {
34         return 0;
35     }
36
37     std::ofstream fs;
38     fs.open(mFilename);
39
40     if (fs == 0)
41     {
42         std::string error = "Error in ThreadListToFile::Run: could not open
43             stream";
44         throw (error);
45     }
46
47     WaitForSingleObject(mHdl, INFINITE);
48     CloseHandle(mHdl);
49
50     TStringList::const_iterator itor = mL1->begin();
51     for(; itor != mL1->end(); ++itor)
52     {
53         fs << *itor << std::endl;
54     }
55
56     fs.close();
57
58     delete this;
59
60     return 0;
61 }

```

../Uebung02/Bsp01/ThreadSort.h

```

1  //////////////////////////////////////
2  // Workfile : ThreadSort.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 09.04.2013
5  // Description : Header of ThreadSort.cpp
6  //////////////////////////////////////
7
8  #ifndef THREADSORT_H
9  #define THREADSORT_H
10
11 #include "ThreadBase.h"
12
13 class ThreadSort : public ThreadBase {
14
15 public:
16     ThreadSort();
17     virtual ~ThreadSort();
18
19     virtual void Init(TStringList * const l1, HANDLE hdl);
20

```



```

7
8 #ifndef THREADMERGE_H
9 #define THREADMERGE_H
10
11 #include <vector>
12 #include "ThreadBase.h"
13
14 class ThreadMerge : public ThreadBase {
15
16 public:
17     ThreadMerge();
18     virtual ~ThreadMerge();
19
20     virtual void Init(TStringList * const l1, TStringList * const l2, HANDLE
        h1, HANDLE h2);
21
22     virtual int Run();
23
24 private:
25     bool IsInitialized;
26     TStringList * mL1; // two lists get merged into this one
27     TStringList * mL2;
28     std::vector<HANDLE> mHandles; // threads to wait for
29 };
30
31 #endif

```

../Uebung02/Bsp01/ThreadMerge.cpp

```

1 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2 // Workfile : ThreadMerge.cpp
3 // Author : Reinhard Penn, Bernhard Selymes
4 // Date : 09.04.2013
5 // Description : Thread that merges 2 lists
6 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
7
8 #include <cassert>
9 #include <algorithm>
10 #include "ThreadMerge.h"
11
12 ThreadMerge::ThreadMerge() {
13     assert(ThreadBase::IsThreadCreated());
14     IsInitialized = false;
15 }
16
17 ThreadMerge::~ThreadMerge() {
18 }
19
20
21 void ThreadMerge::Init(TStringList * const l1, TStringList * const l2,
    HANDLE h1, HANDLE h2)
22 {
23     mL1 = l1;
24     mL2 = l2;
25     mHandles.push_back(h1);
26     mHandles.push_back(h2);
27     IsInitialized = true;
28 }
29
30 int ThreadMerge::Run() {
31     assert(IsInitialized);

```

```

32     if (!IsInitialized) {
33         return 0;
34     }
35
36     WaitForMultipleObjects(mHandles.size(), mHandles.data(), true, INFINITE)
37     ;
38     std::for_each(mHandles.begin(), mHandles.end(), CloseHandle);
39     mL1->merge(*mL2);
40     delete this;
41
42     return 0;
43 }

```

../Uebung02/Bsp01/SortMergeMT.h

```

1  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2  // Workfile : SortMergeMT.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 09.04.2013
5  // Description : Header of SortMergeMT.cpp
6  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
7
8  #ifndef SORTMERGEMT_H
9  #define SORTMERGEMT_H
10
11 #include <string>
12
13 void SortMergeMT(std::vector<std::string> const& filenames,
14                 std::string const& filenameOutput);
15
16 #endif

```

../Uebung02/Bsp01/SortMergeMT.cpp

```

1  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2  // Workfile : SortMergeMT.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 09.04.2013
5  // Description : Multithreaded sort and merge of lists
6  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
7
8  #include <list>
9  #include <vector>
10 #include <sstream>
11 #include <fstream>
12 #include "SortMergeMT.h"
13 #include "ThreadFileToList.h"
14 #include "ThreadListToFile.h"
15 #include "ThreadMerge.h"
16 #include "ThreadSort.h"
17
18 void SortMergeMT(std::vector<std::string> const& filenames,
19                 std::string const& filenameOutput)
20 {
21     if (filenameOutput.empty())
22     {
23         std::string error = "Error in SortMergeMT::SortMergeMT: no valid
24                             filenameOutput";
25         throw (error);
26     }
27 }

```

```

27     size_t const numFiles = 4;
28     if (filenames.size() != numFiles)
29     {
30         std::string error = "Error in SortMergeMT::SortMergeMT: no valid
            vector";
31         throw (error);
32     }
33
34     for (size_t i = 0; i < filenames.size(); ++i)
35     {
36         if (filenames[i].empty())
37         {
38             std::stringstream error;
39             error << "Error in SortMergeMT::SortMergeMT: no valid string at
                index " << i;
40             throw (error.str());
41         }
42     }
43
44     // file to list
45     std::vector<TStringList*> StringLists;
46     std::vector<HANDLE> ThreadFtLHandles;
47     for (size_t i = 0; i < numFiles; ++i)
48     {
49         TStringList* sl = new TStringList;           // allocate list
50         StringLists.push_back(sl);
51         ThreadFileToList* tFtL = new ThreadFileToList; // allocate
            thread
52         tFtL->Init(filenames[i], StringLists[i]);     // init thread
53         ThreadFtLHandles.push_back(tFtL->GetDuplicateHdl()); // save handle
54         tFtL->Start();
55     }
56
57     // sort
58     std::vector<HANDLE> ThreadSortHandles;
59     for (size_t i = 0; i < numFiles; ++i)
60     {
61         ThreadSort* tS = new ThreadSort;           // allocate thread
62         tS->Init(StringLists[i], ThreadFtLHandles[i]); // init thread
63         ThreadSortHandles.push_back(tS->GetDuplicateHdl()); // save
            handle
64         tS->Start();
65     }
66
67     // merge: first level
68     std::vector<HANDLE> ThreadMergeHandles;
69     ThreadMerge* tM1 = new ThreadMerge;
70     ThreadMerge* tM2 = new ThreadMerge;
71     tM1->Init(StringLists[0], StringLists[1], ThreadSortHandles[0],
        ThreadSortHandles[1]);
72     tM2->Init(StringLists[2], StringLists[3], ThreadSortHandles[2],
        ThreadSortHandles[3]);
73     ThreadMergeHandles.push_back(tM1->GetDuplicateHdl());
74     ThreadMergeHandles.push_back(tM2->GetDuplicateHdl());
75     tM1->Start();
76     tM2->Start();
77
78     // merge: second level
79     ThreadMerge* tM3 = new ThreadMerge;

```

```

80     tM3->Init(StringLists[0], StringLists[2], ThreadMergeHandles[0],
      ThreadMergeHandles[1]);
81     ThreadMergeHandles.push_back(tM3->GetDuplicateHdl());
82     tM3->Start();
83
84     // list to file
85     ThreadListToFile* tLtF1 = new ThreadListToFile;
86     tLtF1->Init(filenameOutput, StringLists[0], ThreadMergeHandles[2]);
87     HANDLE hdlListToFile = tLtF1->GetDuplicateHdl();
88     tLtF1->Start();
89
90     WaitForSingleObject(hdlListToFile, INFINITE);
91
92     for (size_t i = 0; i < StringLists.size(); ++i)
93     {
94         delete StringLists[i]; StringLists[i] = 0;
95     }
96 }

```

../Uebung02/Bsp01/main.cpp

```

1  ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2  // Workfile : main.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 09.04.2013
5  // Description : Testdriver for SortMergeMT(...)
6  ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
7
8  #include <iostream>
9  #include <vector>
10 #include <conio.h>
11 #include "StopWatch.h"
12 #include "SortMergeMT.h"
13 // #include <vld.h>
14
15 using namespace std;
16
17 void Test(std::vector<std::string> const& filenames,
18          std::string const& filenameOutput)
19 {
20     try
21     {
22         SortMergeMT(filenames, filenameOutput);
23     }
24     catch(std::bad_alloc& ex)
25     {
26         cerr << ex.what() << endl;
27     }
28     catch(string const& ex)
29     {
30         cerr << ex << endl;
31     }
32     catch(...)
33     {
34         cerr << "Unhandled exception occured";
35     }
36 }
37
38 int main() {
39     std::cout << "Start timer." << std::endl;
40     stw::Start();

```



```

41
42     vector<string> strVec;
43     Test(strVec, "sorted.txt");           // test empty vector
44
45     strVec.push_back("");
46     strVec.push_back("");
47     strVec.push_back("");
48     strVec.push_back("");
49     Test(strVec, "sorted.txt");           // test empty strings
50
51     strVec.clear();
52     strVec.push_back("dutch.txt");
53     strVec.push_back("finnish.txt");
54
55     Test(strVec, "sorted.txt");           // test wrong size vector
56
57     strVec.push_back("japanese.txt");
58     strVec.push_back("polish.txt");
59
60     Test(strVec, "");                     // test empty output name
61
62     Test(strVec, "sortedMT.txt");         // test normal
63
64     std::cout << "End timer. Time: " << stw::Stop() << std::endl;
65     cout << "Press a key..." << endl;    // poor man's sync
66     _getch();                             // so you can see the output better
67
68     return 0;
69 }

```

../Uebung02/Bsp01_SingleThreaded/SortMergeST.h

```

1  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2  // Workfile : SortMergeST.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 09.04.2013
5  // Description : Header of SortMergeST.cpp
6  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
7
8  #ifndef SORTMERGEST_H
9  #define SORTMERGEST_H
10
11  #include <string>
12  #include <vector>
13
14  void SortMergeST(std::vector<std::string> const& filenames,
15                  std::string const& filenameOutput);
16
17  #endif

```

../Uebung02/Bsp01_SingleThreaded/SortMergeST.cpp

```

1  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2  // Workfile : SortMergeST.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 09.04.2013
5  // Description : Singlethreaded Version of SortMergeMT(...)
6  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
7
8  #include <sstream>
9  #include <list>
10 #include <fstream>

```

```

11 #include <iostream>
12 #include "SortMergeST.h"
13 #include "StopWatch.h"
14
15 typedef std::list<std::string> TStringList;
16
17 void SortMergeST(std::vector<std::string> const& filenames,
18                 std::string const& filenameOutput)
19 {
20     if (filenameOutput.empty())
21     {
22         std::string error = "Error in SortMergeST::SortMergeST: no valid
23                             filenameOutput";
24         throw (error);
25     }
26
27     size_t const numFiles = 4;
28     if (filenames.size() != numFiles)
29     {
30         std::string error = "Error in SortMergeST::SortMergeST: no valid
31                             vector";
32         throw (error);
33     }
34
35     for (size_t i = 0; i < filenames.size(); ++i)
36     {
37         if (filenames[i].empty())
38         {
39             std::stringstream error;
40             error << "Error in SortMergeST::SortMergeST: no valid string at
41                     index " << i;
42             throw (error.str());
43         }
44     }
45
46     // file to list
47     TStringList strList;
48     for (size_t i = 0; i < numFiles; ++i)
49     {
50         std::ifstream fs;
51         fs.open(filenames[i]);
52
53         if (fs == 0)
54         {
55             std::string error = "Error in SortMergeST::SortMergeST: could not
56                                 open input stream";
57             throw (error);
58         }
59
60         std::string line;
61         while (!fs.eof())
62         {
63             std::getline(fs, line);
64             strList.push_back(line);
65         }
66
67         fs.close();
68     }
69 }

```

```

67     // sort
68     strList.sort();
69
70     // list to file
71     std::ofstream fs;
72     fs.open(filenameOutput);
73
74     if (fs == 0)
75     {
76         std::string error = "Error in SortMergeST::SortMergeST: could not
77             open output stream";
78         throw (error);
79     }
80     TStringList::const_iterator itor = strList.begin();
81     for(; itor != strList.end(); ++itor)
82     {
83         fs << *itor << std::endl;
84     }
85
86     fs.close();
87 }

```

../Uebung02/Bsp01_SingleThreaded/main.cpp

```

1  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2  // Workfile : main.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 09.04.2013
5  // Description : Testdriver for SortMergeST(...)
6  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
7
8  #include <iostream>
9  #include <vector>
10 #include <string>
11 #include <conio.h>
12 #include "SortMergeST.h"
13 #include "StopWatch.h"
14 // #include <vld.h>
15
16 using namespace std;
17
18 void Test(std::vector<std::string> const& filenames,
19         std::string const& filenameOutput)
20 {
21     try
22     {
23         SortMergeST(filenames, filenameOutput);
24     }
25     catch(std::bad_alloc& ex)
26     {
27         cerr << ex.what() << endl;
28     }
29     catch(string const& ex)
30     {
31         cerr << ex << endl;
32     }
33     catch(...)
34     {
35         cerr << "Unhandled exception occurred";
36     }

```

```

37 }
38
39 int main() {
40     std::cout << "Start timer." << std::endl;
41     stw::Start();
42
43     vector<string> strVec;
44     Test(strVec, "sorted.txt");           // test empty vector
45
46     strVec.push_back("");
47     strVec.push_back("");
48     strVec.push_back("");
49     strVec.push_back("");
50     Test(strVec, "sorted.txt");           // test empty strings
51
52     strVec.clear();
53     strVec.push_back("dutch.txt");
54     strVec.push_back("finnish.txt");
55
56     Test(strVec, "sorted.txt");           // test wrong size vector
57
58     strVec.push_back("japanese.txt");
59     strVec.push_back("polish.txt");
60
61     Test(strVec, "");                     // test empty output name
62
63     Test(strVec, "sortedST.txt");         // test normal
64
65     std::cout << "End timer. Time: " << stw::Stop() << std::endl;
66     cout << "Press a key..." << endl;   // poor man's sync
67     _getch();                             // so you can see the output better
68
69     return 0;
70 }

```

1.5 Testausgabe

Multithreaded(Release):

```
Start timer.  
End timer. Time: 5.695  
Press a key...
```

Multithreaded(Debug ohne Ende(dauert ewig)):

```
Start timer.  
Error in SortMergeMT::SortMergeMT: no valid vector  
Error in SortMergeMT::SortMergeMT: no valid string at index 0  
Error in SortMergeMT::SortMergeMT: no valid vector  
Error in SortMergeMT::SortMergeMT: no valid filenameOutput
```

Singlethreaded(Release):

```
Start timer.  
End timer. Time: 5.885  
Press a key...
```

Singlethreaded(Debug ohne Ende(dauert ewig)):

```
Start timer.  
Error in SortMergeST::SortMergeST: no valid vector  
Error in SortMergeST::SortMergeST: no valid string at index 0  
Error in SortMergeST::SortMergeST: no valid vector  
Error in SortMergeST::SortMergeST: no valid filenameOutput
```

2 Beispiel 2

2.1 Funktionsbeschreibung

Das AdminTool besteht aus einer Klasse die nur statische Funktionen enthält. Die Funktionen wurden statisch erstellt, da die Funktionalität des AdminTools nicht vom jeweiligen Objekt abhängig ist. Das AdminTool kann Prozesse starten und beenden. Zum Prozess starten muss der jeweilige Name des Programms übergeben werden, zum beenden wird die Prozess-Id übergeben. Eine weitere Funktion des AdminTools ist es alle laufenden Prozesse aufzulisten. Dabei wird Prozessname, ProzessId, BasePriority und ThreadCount ausgegeben. Die letzte Funktion des AdminTools ist es Systeminformationen auszugeben. Zum Beispiel: Prozessorarchitektur, Anzahl der Prozessorkerne, Computernamen, OS-Version,... . Die Fehlerbehandlung erfolgt mittels Exception die im Testtreiber abgefangen werden. Im Testtreiber wird ein einfaches Menü zur Verwaltung der Funktionen des AdminTools erzeugt.

Schnittstelle:

Name: StartProgram

Returnwert: void

Parameter:

size_t ProgramName: Name des zu startenden Programms

Name: ListProcesses

Returnwert: void

Name: KillProcess

Returnwert: void

Parameter:

size_t ProcessId: Prozess Id, des Prozesses der terminiert wird

Name: PrintSystemInfo

Returnwert: void

2.2 Sourcecode

../Uebung02/./PSS4_UE02_AdminTool/PSS4_UE02_AdminTool/AdminTool.h

```
1  #ifndef ADMINTOOL_H
2  #define ADMINTOOL_H
3
4  #include <string>
5  #include <Windows.h>
6
7  class AdminTool
8  {
9  public:
10     static void StartProgram(std::string const& ProgramName);
11     static void ListProcesses();
12     static void KillProcess(DWORD const& ProcessId);
13     static void PrintSystemInfo();
14 };
15
16 #endif
```

../Uebung02/../../PSS4_UE02_AdminTool/PSS4_UE02_AdminTool/AdminTool.cpp

```
1 #include <iostream>
2 #include <iomanip>
3 #include <Windows.h>
4 #include <TlHelp32.h>
5 #include "Hlp.h"
6 #include "AdminTool.h"
7
8 void AdminTool::StartProgram(std::string const& ProgramName)
9 {
10     STARTUPINFO si; //specifies how the main window of the new process
        should appear
11     PROCESS_INFORMATION pi; //filled by CreateProcess (IDs and so on)
12
13     ZeroMemory(&si, sizeof(si));
14     si.cb = sizeof(si);
15
16     ZeroMemory(&pi, sizeof(pi));
17
18     //convert the string to char*
19     char* pProgramName = new char[ProgramName.size()+1];
20     const char* pTemp = ProgramName.c_str();
21     strcpy_s(pProgramName, ProgramName.size()+1, pTemp);
22
23     //start new process
24     if (!CreateProcess(0, pProgramName, 0, 0, false, 0, 0, 0, &si, &pi))
25     {
26         throw("CreateProcess failed: " + Hlp::ErrMsg(GetLastError()));
27     }
28     else
29     {
30         //close process handle and thread handle (decoupling)
31         CloseHandle(pi.hProcess);
32         CloseHandle(pi.hThread);
33     }
34 }
35
36 void AdminTool::ListProcesses()
37 {
38     HANDLE hProcessSnap;
39     PROCESSENTRY32 pe32;
40
41     int const ProcessNameWidth = 25;
42     int const PIDWidth = 10;
43     int const PriorityWidth = 20;
44     int const ThreadCountWidth = 15;
45     char const FillSign = ' ';
46     char const LineDivider = '-';
47
48     // Take a snapshot of all processes in the system.
49     hProcessSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
50     if(hProcessSnap == INVALID_HANDLE_VALUE)
51     {
52         throw("CreateToolhelp32Snapshot failed: " + Hlp::ErrMsg(GetLastError()));
53     }
54
55     // Set the size of the process structure.
56     pe32.dwSize = sizeof(PROCESSENTRY32);
57
```



```

58 // Retrieve information about the first process,
59 // and exit if unsuccessful
60 if( !Process32First( hProcessSnap, &pe32 ) )
61 {
62     CloseHandle( hProcessSnap ); // clean the snapshot object
63     throw("Process32First failed: " + Hlp::ErrMsg(GetLastError()));
64 }
65
66 //Print header info
67 std::cout << std::setw(ProcessNameWidth) << std::setfill(FillSign) <<
    std::left << "Process";
68 std::cout << std::setw(PIDWidth) << std::setfill(FillSign) << std
    ::left << "PID";
69 std::cout << std::setw(PriorityWidth) << std::setfill(FillSign) <<
    std::left << "BasePriority";
70 std::cout << std::setw(ThreadCountWidth) << std::setfill(FillSign) <<
    std::left << "Thread Count";
71 std::cout << std::endl;
72 std::cout << std::string(ProcessNameWidth+PIDWidth+PriorityWidth+
    ThreadCountWidth, LineDivider) << std::endl;
73
74 // Now walk the snapshot of processes, and
75 // display information about each process in turn
76 do
77 {
78     //Print process info
79     std::cout << std::setw(ProcessNameWidth) << std::setfill(FillSign)
        << std::left << pe32.szExeFile;
80     std::cout << std::setw(PIDWidth) << std::setfill(FillSign) <<
        std::left << pe32.th32ProcessID;
81     std::cout << std::setw(PriorityWidth) << std::setfill(FillSign)
        << std::left << pe32.pcPriClassBase;
82     std::cout << std::setw(ThreadCountWidth) << std::setfill(FillSign)
        << std::left << pe32.cntThreads;
83     std::cout << std::endl;
84 } while(Process32Next(hProcessSnap, &pe32));
85
86 CloseHandle(hProcessSnap);
87 }
88
89 void AdminTool::KillProcess(DWORD const& ProcessId)
90 {
91     DWORD DesiredAccess = PROCESS_TERMINATE;
92     BOOL InheritHandle = FALSE;
93
94     HANDLE hProcess = OpenProcess(DesiredAccess, InheritHandle, ProcessId);
95     if (hProcess == NULL)
96     {
97         throw("OpenProcess failed: " + Hlp::ErrMsg(GetLastError()));
98     }
99
100    if (!TerminateProcess(hProcess, 0))
101    {
102        throw("TerminateProcess failed: " + Hlp::ErrMsg(GetLastError()));
103    }
104    CloseHandle(hProcess);
105 }
106
107 void AdminTool::PrintSystemInfo()
108 {

```

```

109  _SYSTEM_INFO sysInfo;
110  GetSystemInfo(&sysInfo);
111
112  int const HeaderWidth      = 30;
113  int const HardwareWidth    = 25;
114  int const SoftwareWidth    = 13;
115  char const FillSign        = ' ';
116  char const LineDivider     = '-';
117
118
119  //Hardware
120  std::cout << "System Information <HARDWARE>:" << std::endl;
121  std::cout << std::string(HeaderWidth, LineDivider) << std::endl;
122
123  std::cout << std::setw(HardwareWidth) << std::setfill(FillSign) << std::
    left << "Processor Type";
124  std::cout << ": " << sysInfo.dwProcessorType << std::endl;
125
126  std::cout << std::setw(HardwareWidth) << "Architecture";
127  if (sysInfo.wProcessorArchitecture == PROCESSOR_ARCHITECTURE_INTEL)
128  {
129      std::cout << ": " << "Intel CPU or compatibles" << std::endl;
130
131      std::cout << std::setw(HardwareWidth) << "Processor Level (Family)";
132      std::cout << ": " << sysInfo.wProcessorLevel << std::endl;
133
134      std::cout << std::setw(HardwareWidth) << "Model and Stepping MMSS";
135      std::cout << ": 0x" << std::hex << sysInfo.wProcessorRevision << std
        ::dec << std::endl;
136  }
137  else
138  {
139      std::cout << ": " << "Unknown architecture" << std::endl;
140  }
141
142  std::cout << std::setw(HardwareWidth) << "Number of Processors";
143  std::cout << ": " << sysInfo.dwNumberOfProcessors << std::endl;
144  std::cout << std::endl;
145
146
147  //TODO Features:
148  std::cout << std::endl;
149
150
151  //Software
152  std::cout << "System Information <SOFTWARE>:" << std::endl;
153  std::cout << std::string(HeaderWidth, LineDivider) << std::endl;
154
155  char computerName[MAX_COMPUTERNAME_LENGTH + 1];
156  DWORD size = sizeof(computerName) / sizeof(computerName[0]);
157
158  if (!GetComputerName(computerName, &size))
159  {
160      throw("GetComputerName failed: " + Hlp::ErrMsg(GetLastError()));
161  }
162  std::cout << std::setw(SoftwareWidth) << "Computer Name";
163  std::cout << ": " << computerName << std::endl;
164
165  OSVERSIONINFO osVersionInfo;
166  ZeroMemory(&osVersionInfo, sizeof(osVersionInfo));

```

```
167     osVersionInfo.dwOSVersionInfoSize = sizeof(osVersionInfo);
168
169     if (!GetVersionEx(&osVersionInfo))
170     {
171         throw("GetVersionEx failed: " + Hlp::ErrMsg(GetLastError()));
172     }
173     std::cout << std::setw(SoftwareWidth) << "OS Version";
174     std::cout << ": " << "Major: " << osVersionInfo.dwMajorVersion
175         << "  Minor: " << osVersionInfo.dwMinorVersion << std::endl;
176
177     std::cout << std::setw(SoftwareWidth) << "OS Revision";
178     std::cout << ": " << osVersionInfo.szCSDVersion << std::endl;
179 }
```

../Uebung02/./PSS4_UE02_AdminTool/PSS4_UE02_AdminTool/main.cpp

```
1  #include <iostream>
2  #include <vld.h>
3  #include "AdminTool.h"
4
5  using namespace std;
6
7
8  int main(int argc, char **argv)
9  {
10     char command = 0;
11     string ProgramName = " ";
12     DWORD ProcessId = -1;
13
14     while(command != 'e')
15     {
16
17         cout << "Enter command: ";
18         cin >> command;
19         try
20         {
21             switch (command)
22             {
23                 case '?':
24                     cout << "StartProgram: 0" << endl;
25                     cout << "ListProcesses: 1" << endl;
26                     cout << "Remove: 2" << endl;
27                     cout << "KillProcess: 3" << endl;
28                     cout << "PrintSystemInfo: 4" << endl;
29                     cout << "Exit: e" << endl;
30                     cout << endl;
31                     break;
32
33                 case '0':
34                     cout << "Enter program name: ";
35                     cin >> ProgramName;
36                     AdminTool::StartProgram(ProgramName);
37                     cout << endl;
38                     break;
39
40                 case '1':
41                     AdminTool::ListProcesses();
42                     cout << endl;
43                     break;
44
45                 case '2':
46                     cout << "Enter process Id of process to kill: ";
47                     cin >> ProcessId;
48                     AdminTool::KillProcess(ProcessId);
49                     cout << endl;
50                     break;
51
52                 case '3':
53                     AdminTool::PrintSystemInfo();
54                     cout << endl;
55                     break;
56             }
57         }
58         catch(std::bad_alloc& ex)
59         {
```

```
60         cerr << ex.what() << endl;
61     }
62     catch (std::string const& error)
63     {
64         std::cerr << error << std::endl;
65     }
66     catch (...)
67     {
68         cerr << "Unknown exception caught.";
69     }
70 }
71 return 0;
72 }
```

2.3 Testausgabe

```
Enter command: ?
StartProgram: 0
ListProcesses: 1
Remove: 2
KillProcess: 3
PrintSystemInfo: 4
Exit: e
```

```
Enter command: 0
Enter program name: not_existing
CreateProcess failed: Das System kann die angegebene Datei nicht finden.
```

```
Enter command: 0
Enter program name: notepad
```

```
Enter command: 1
```

Process	PID	BasePriority	Thread Count

[System Process]	0	0	4
System	4	8	114
smss.exe	296	11	3
csrss.exe	392	13	9
wininit.exe	464	13	3
csrss.exe	492	13	13
services.exe	524	9	7
lsass.exe	540	9	8
lsm.exe	548	8	10
svchost.exe	656	8	10
svchost.exe	740	8	8
atiesrxx.exe	800	8	6
winlogon.exe	848	13	3
svchost.exe	900	8	23
svchost.exe	932	8	34
svchost.exe	972	8	21
svchost.exe	1008	8	39
UMVPFSrv.exe	324	8	3
svchost.exe	1124	8	16
atieclxx.exe	1228	8	10
spoolsv.exe	1396	8	13
sched.exe	1424	8	7
svchost.exe	1448	8	19
armsvc.exe	1564	8	4
Fuel.Service.exe	1588	8	6
avguard.exe	1612	8	35
bitkinexsvc.exe	1648	8	4
jtagserver.exe	1736	8	4
sqlservr.exe	1760	8	35
mysqld.exe	1780	8	22
sqlwriter.exe	1928	8	4

TeamViewer_Service.exe	1104	8	6
VC9SecS.exe	1496	8	5
WLIDSVC.EXE	960	8	8
WLIDSVCM.EXE	2272	8	3
avshadow.exe	2660	8	3
conhost.exe	2668	8	1
avwebgrd.exe	2700	8	10
SearchIndexer.exe	2808	8	13
svchost.exe	2960	8	16
wmpnetwk.exe	2984	8	13
svchost.exe	2540	8	26
dwm.exe	1952	13	5
explorer.exe	3008	8	25
ddm.exe	3384	8	9
avgnt.exe	3708	8	11
acrotray.exe	3808	8	2
MOM.exe	3876	8	15
svchost.exe	3436	8	6
CCC.exe	5084	8	27
svchost.exe	5636	8	11
dllhost.exe	5344	8	5
PresentationFontCache.exe	4128	8	6
texmaker.exe	6796	8	4
devenv.exe	7896	8	26
vcpkgssrv.exe	6092	8	5
mspdbsrv.exe	3868	8	3
wuauclt.exe	4056	8	3
MpCmdRun.exe	6440	8	8
vcpkgssrv.exe	1696	8	8
cmd.exe	7760	8	1
conhost.exe	3884	8	2
PSS4_UE02_AdminTool.exe	7964	8	1
notepad.exe	6640	8	1

Enter command: 2

Enter process Id of process to kill: 123123123

OpenProcess failed: Falscher Parameter.

Enter command: 2

Enter process Id of process to kill: 0

OpenProcess failed: Falscher Parameter.

Enter command: 2

Enter process Id of process to kill: 4

OpenProcess failed: Zugriff verweigert

Enter command: 2

Enter process Id of process to kill: 6640

Enter command: 3

System Information <HARDWARE>:

Processor Type : 586
Architecture : Intel CPU or compatibles
Processor Level (Family) : 16
Model and Stepping MMSS : 0x402
Number of Processors : 4

System Information <SOFTWARE>:

Computer Name: REINHARD-PC
OS Version : Major: 6 Minor: 1
OS Revision : Service Pack 1

Enter command: e

Drücken Sie eine beliebige Taste . . .