Software Entwurf 4
Übung 2

HSD
FH-HAGENBERG

Sommersemester 2013

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

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

Abgabetermin: KW 15 - **Mi, 10.4.2013**
Punkte: _____
korrigiert: _____

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.*

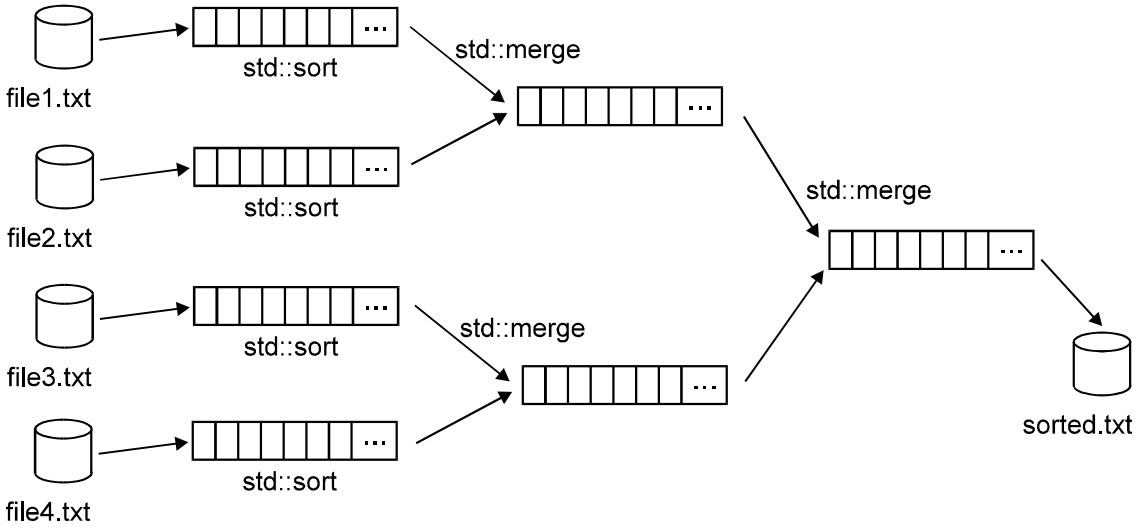

Abb 1: Prinzip einer parallelen Dateisortierung

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 Bibilotheken** (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 Zugriffrechte 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 CloseHanlde() 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, &
           ThreadId);
13
14     assert(ThreadHdl != 0);
15 }
16
17 ThreadBase::~ThreadBase() {
18     CloseHandle(ThreadHdl);
19 }
20
21 unsigned long WINAPI ThreadBase::ThreadFunc(void * Param) {
22     assert(Param != 0);
23
24     ThreadBase * pObj = static_cast <ThreadBase *> (Param);
25     pObj->Run();
26
```

```
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,
          GetCurrentProcess(),
52                         &hdlDup, 0, FALSE, DUPLICATE_SAME_ACCESS);
53    assert(res != 0);
54    return hdlDup;
55 }
```

../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,
          HANDLE hdl);
20
21    virtual int Run();
22
23 private:
24    bool IsInitialized;
25    std::string  mFilename;
26    TStringList* mL1;
27    HANDLE mHdl;
28 };
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 *
       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
          stream";
43       throw (error);
44    }
45
46    WaitForSingleObject(mHdl, INFINITE);
47    CloseHandle(mHdl);
48
49    TStringList::const_iterator itor = mL1->begin();
50    for(;itor != mL1->end(); ++itor)
51    {
52       fs << *itor << std::endl;
53    }
54
55    fs.close();
56
57    delete this;
58
59    return 0;
60 }
```

../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
```

```
21    virtual int Run();
22
23 private:
24    bool IsInitialized;
25    TStringList * mL1;      // pointer to list that is sorted
26    HANDLE mHdl;
27 };
28
29 #endif
```

../Uebung02/Bsp01/ThreadSort.cpp

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : ThreadSort.cpp
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 09.04.2013
5  // Description : Thread that sorts a list
6  ////////////////////////////////////////////////////////////////////////
7
8  #include <cassert>
9  #include <algorithm>
10 #include "ThreadSort.h"
11
12 ThreadSort::ThreadSort() {
13    assert(ThreadBase::IsThreadCreated());
14    IsInitialized = false;
15 }
16
17 ThreadSort::~ThreadSort() {
18
19 }
20
21 void ThreadSort::Init(TStringList * const l1, HANDLE hdl)
22 {
23    mL1 = l1;
24    mHdl = hdl;
25    IsInitialized = true;
26 }
27
28 int ThreadSort::Run() {
29    assert(IsInitialized);
30    if (!IsInitialized) {
31       return 0;
32    }
33
34    WaitForSingleObject(mHdl, INFINITE);
35    CloseHandle(mHdl);
36    mL1->sort();
37
38    delete this;
39
40    return 0;
41 }
```

../Uebung02/Bsp01/ThreadMerge.h

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : ThreadMerge.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 09.04.2013
5  // Description : Header of ThreadMerge.cpp
6  ////////////////////////////////////////////////////////////////////////
```

```
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     std::for_each(mHandles.begin(), mHandles.end(), CloseHandle);
38     mL1->merge(*mL2);
39
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
             filenameOutput";
24       throw (error);
25    }
26
```

```
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>
```

```cpp
#include <iostream>
#include "SortMergeST.h"
#include "StopWatch.h"

typedef std::list<std::string> TStringList;

void SortMergeST(std::vector<std::string> const& filenames,
             std::string const& filenameOutput)
{
   if (filenameOutput.empty())
   {
      std::string error = "Error in SortMergeST::SortMergeST: no valid
         filenameOutput";
      throw (error);
   }

   size_t const numFiles = 4;
   if (filenames.size() != numFiles)
   {
      std::string error = "Error in SortMergeST::SortMergeST: no valid
         vector";
      throw (error);
   }

   for (size_t i = 0; i < filenames.size(); ++i)
   {
      if (filenames[i].empty())
      {
         std::stringstream error;
         error << "Error in SortMergeST::SortMergeST: no valid string at
            index " << i;
         throw (error.str());
      }
   }

   // file to list
   TStringList strList;
   for (size_t i = 0; i < numFiles; ++i)
   {
      std::ifstream fs;
      fs.open(filenames[i]);

      if (fs == 0)
      {
         std::string error = "Error in SortMergeST::SortMergeST: could not
            open input stream";
         throw (error);
      }

      std::string line;

      while(!fs.eof())
      {
         std::getline(fs, line);
         strList.push_back(line);
      }

      fs.close();
   }

```

```
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
              open output stream";
77        throw (error);
78     }
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 occured";
36    }
```

```
}

int main() {
   std::cout << "Start timer." << std::endl;
   stw::Start();

   vector<string> strVec;
   Test(strVec, "sorted.txt");         // test empty vector

   strVec.push_back("");
   strVec.push_back("");
   strVec.push_back("");
   strVec.push_back("");
   Test(strVec, "sorted.txt");         // test empty strings

   strVec.clear();
   strVec.push_back("dutch.txt");
   strVec.push_back("finnish.txt");

   Test(strVec, "sorted.txt");         // test wrong size vector

   strVec.push_back("japanese.txt");
   strVec.push_back("polish.txt");

   Test(strVec, "");             // test empty output name

   Test(strVec, "sortedST.txt");    // test normal

   std::cout << "End timer. Time: " << stw::Stop() << std::endl;
   cout << "Press a key..." << endl;   // poor man's sync
   _getch();                     // so you can see the output better

   return 0;
}
```

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

2.2 Sourcecode

2.3 Testausgabe