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.*

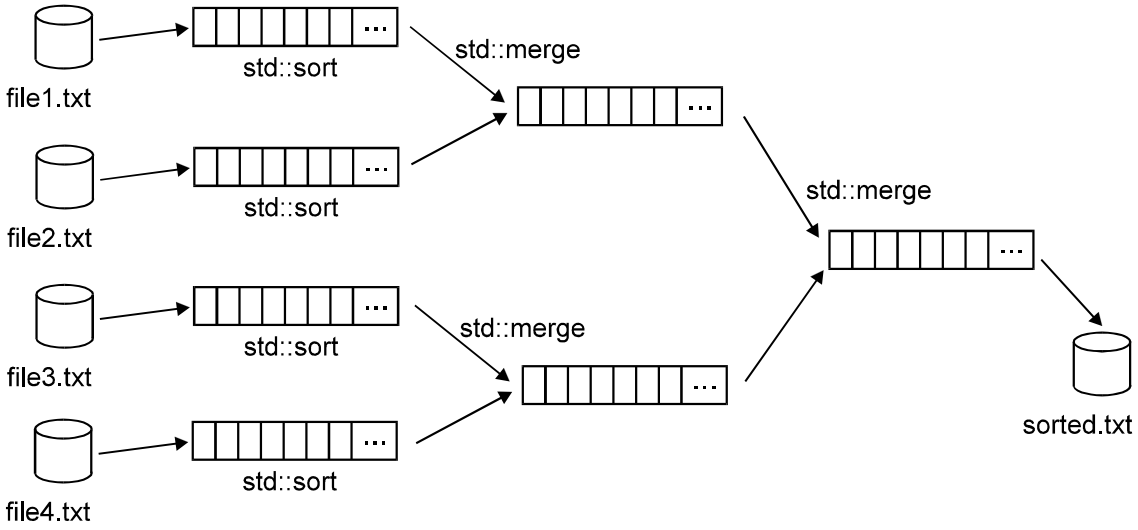

Abb 1: Prinzip einer parallelen Dateisortierung

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

1.2 Sourcecode

1.3 Testausgabe

2 Beispiel 2

2.1 Funktionsbeschreibung

2.2 Sourcecode

2.3 Testausgabe