Hochschule RheinMain
Fachbereich DCSM - Informatik
Prof. Dr. Reinhold Kröger
Prof. Dr. Martin Gergeleit

Betriebssysteme und Rechnerarchitektur
WS 2015/16
LV 3142

Übungsblatt 1
Bearbeitungszeit 2 Wochen

Aufgabe 1.1 (Repository):

In diesem Praktikum werden die Abgaben in einem zentralen SVN-Repository verwaltet. Die Praktikumsleiter legen dazu für jedes Übungsblatt eine Vorlage an, die Sie am Anfang der Übung auschecken müssen. Machen Sie sich mit der Handhabung von SVN vertraut. Weiterführende Hilfe finden Sie dazu im Wiki des Studiengangs und im Internet, z.B.:

- https://www-intern.cs.hs-rm.de/publicwiki/index.php/Hauptseite

In den Vorlagen finden Sie eine einheitliche Projektstruktur vor, die Sie nicht verändern dürfen. Das Projekt beinhaltet i.d.R. Templates für alle Quelldateien und die anzulegende Dokumentation, sowie ein Makefile, um das Projekt mit den gewünschten Compiler-Settings unter Linux zu kompilieren:

https://svn.cs.hs-rm.de/svn/bs15_mmust001/1/	Vorlage Aufgabenblatt 1
https://svn.cs.hs-rm.de/svn/bs15_mmust001/2/	Vorlage Aufgabenblatt 2
https://svn.cs.hs-rm.de/svn/bs15_mmust001/3/	Vorlage Aufgabenblatt 3

Initialer Checkout der Aufgaben aus dem Repository:

```
$ svn checkout https://svn.cs.hs-rm.de/svn/bs15_mmust001/
```

Aktualisierung des Repositorys:

```
$ svn update
```

Anzeigen der lokalen Änderungen:

```
$ svn diff
```

Speichern der lokalen Änderungen im Repository (wichtig!):

```
$ svn commit
```

Bei Problemen mit dem SVN-Zugang wenden Sie sich bitte an die Laboringenieure.

Aufgabe 1.2 (UNIX-Kommandos):

Wiederholen Sie den Umgang mit den allgemeinen UNIX-Kommandos. Im Weiteren wird die Kenntnis des praktischen Umgangs mit dem UNIX-System auf Kommandoebene als bekannt vorausgesetzt. Sie können sich z.B. orientieren an:

- Online-Dokumentationsprojekt SelfLinux (http://www.selflinux.org/selflinux/)
- Single Unix Specification (siehe Materialliste oder http://www.opengroup.org)

Machen Sie sich weiterhin mit den sogenannten Manual-Pages in einem Unix-System vertraut. Die Manual-Pages sind in Gruppen organisiert. Gruppe 1 behandelt UNIX Kommandos, Gruppe 2 die Systemdienstschnittstelle, Gruppe 3 die Dokumentation der C-Bibliotheken, etc. Der Befehl `apropos` auf der Kommandozeile zeigt für ein Stichwort die vorhandenen Manual-Pages an. Das Kommando `man` zeigt die jeweilige Manual-Page an:

```
$ apropos printf
$ man 3 printf
```

Aufgabe 1.3 (Programmierstil):

Im Weiteren des Praktikums wird davon ausgegangen, dass Sie sich einen auch für andere Personen lesbaren Programmierstil angewöhnt haben (vgl. Prof. Reith: C-Programmierrichtlinie zu "Algorithmen und Datenstrukturen", SS 2014). Falls Sie noch unsicher sein sollten, beachten Sie bevorzugt diesen Style Guide. Alternativ kann auch z.B. der kleine Style Guide für C/C++ genutzt werden, der unter Materialien ebenfalls bereitgestellt wird.

Aufgabe 1.4 (Aufwärmübung):

Wissen Sie noch, wie Programmentwicklung unter UNIX geschieht? Wenn Sie in dieser Hinsicht noch oder wieder unsicher sein sollten, können Sie die Notizen "Übersicht zur Programmentwicklung unter UNIX" durcharbeiten. Diese enthalten auch Hinweise bezüglich der Vorgehensweise zur Bearbeitung der weiteren Teilaufgaben. Die Notizen sind als PDF-Datei `C_dev_kz.pdf` über die Materialienseite zur Vorlesung erhältlich.

(a) Schauen Sie sich die Datei `rechner.c` [vormals `gesamt.c`] an. Verstehen Sie das Programm. Übersetzen Sie es in die ausführbare Datei `rechner1` und führen Sie alle Wahlmöglichkeiten aus. Aus funktionaler Sicht ist das Programm sehr einfach. Betrachten Sie zwischendurch den C-Quellcode und ergründen Sie in Zweifelsfällen das Laufzeitverhalten aus dem Quellcode heraus und umgekehrt.

(b) Erzeugen Sie unter Verwendung des Präprozessors das expandierte Hauptprogramm und unter Verwendung dieser Datei eine neue Version `rechner2`. Was können Sie in der expandierten Datei beobachten?

(c) Erzeugen Sie aus dem Programm-Code mit Hilfe des Compilers eine Assembler-Datei `rechner3.s` und assemblieren Sie diese, so dass Sie anschließend unter Verwendung der erzeugten Objektdatei `rechner3.o` eine neue Version `rechner3` erzeugen können. Schauen Sie ruhig mal in die Assembler-Datei hinein!

(d) Wie verändern sich Assemblercode, Objektdatei und das fertige Programm, wenn Sie eine neue Version `rechner4` mit den Compiler-Optimierungen mit `-O2` übersetzen?

(e) Disassemblieren Sie das erzeugte Programm `rechner4` mit Hilfe des Dienstprogramms `objdump` und speichern Sie die Assembler-Ausgabe in der Datei `rechner4.dis`. Was hat sich im Vergleich mit `rechner4.s` aus Aufgabe (d) geändert?

(f) Finden Sie mit dem Dienstprogramm `nm` heraus, wie der Linker Abhängigkeiten auf Bibliotheksfunktionen wie `printf()` in Objektdateien auflöst.

Aufgabe 1.5 (Ungepufferte Dateiein-/ausgabe):

In dieser Übung wird der Umgang mit Dateien mittels UNIX-Systemaufrufen geübt. Unterscheiden Sie zwischen Systemaufrufen und vergleichbaren Bibliotheksfunktionen der ANSI C Standard-I/O-Bibliothek. Beachten Sie die notwendigen Header-Dateien. Zur Beschreibung der Systemaufrufe sei auf die Manual Pages, Kapitel 2, verwiesen. Beachten Sie, dass Systemaufrufe i.d.R. mit den Rückgabeparametern auch Fehler anzeigen. Diese müssen überprüft werden, ansonsten wird das Programm in der Beurteilung durch den Praktikumsleiter abgewertet.

int open(const char *pathname, int oflag); bzw. int open(const char *pathname, int oflag, mode_t mode);	Öffnen einer Datei
int creat(const char *pathname, mode_t mode); analog int open(pathname, O_WRONLY|O_CREAT|O_TRUNC, mode)	Erzeugen einer neuen Datei
int close(int filedes);	Schließen einer Datei
ssize_t read(int filedes, void *buf, size_t nbytes);	Lesen aus einer Datei
ssize_t write(int filedes, const void *buf, size_t nbytes);	Schreiben in eine Datei
off_t lseek(int filedes, off_t offset, int whence);	Positionieren in einer Datei

(a) Schreiben Sie ein C-Programm `mybcp.c`, das eine beliebige Datei byteweise kopiert. Der Name der Quelldatei und der Name der Zieldatei sollen beim Programmaufruf über die Kommandozeile übergeben werden, d.h. ein Aufruf des Programms lautet `mybcp` *`myfromfile mytofile`*. Zunächst soll die erzeugte Datei Lese- und Schreibrecht für den Eigentümer besitzen, keine Rechte für alle anderen.

(b) Modifizieren Sie Ihr Programm aus (a) so zu einem Programm `myrevbcp.c`, dass die erzeugte Datei die Folge der Bytes in umgekehrter Reihenfolge enthält.

(c) Schreiben Sie ein Programm `mybappend.c`, das den Inhalt einer Datei byteweise an eine bestehende Datei anfügt. Die Namen der Ausgangs-/Zieldatei sowie der anzufügenden Datei sollen wieder beim Programmaufruf über die Kommandozeile übergeben werden.

(d) Modifizieren Sie Ihr Programm aus (a) so zu einem Programm `mycp.c`, dass die Anzahl der in einem Systemaufruf kopierten Bytes (Puffergröße) über die Kommandozeile wählbar ist.

(e) Benutzen Sie das Utility `time` zur Ermittlung der Ausführungszeit eines Programms (elapsed real time). Ermitteln Sie mittels `time` für eine mehrere Megabyte große Datei Ausführungszeiten Ihres Programms für unterschiedliche Puffergrößen (z.B. Zweierpotenzen an Bytes) und stellen Sie die Ergebnisse in einer Tabelle zusammen. Was beobachten Sie? Was bedeuten die anderen Zeitangaben, die Sie mittels `time` ermitteln können?

Vergessen Sie nicht, Ihre erzeugten Dateien im Projekt abzulegen!

Aufgabe 1.6 (Einfache Dateiattribute):

int stat(const char *pathname, struct stat *buf); bzw. int fstat(int filedes, struct stat *buf);	Ermitteln von Dateiattributen

(a) Schreiben Sie ein C-Programm `filelength.c`, das die Länge einer Datei ausgibt, deren Namen über die Kommandozeile übergeben wird. Vergleichen Sie Ihre Ausgabe mit der Ausgabe von `ls`.

(b) Modifizieren Sie Ihr Programm `mycp.c` aus Aufgabe 1.5 so, dass die Rechtefestlegungen von Originaldatei und kopierter Datei identisch sind.

(c) Modifizieren Sie Ihr Programm aus (b) so, dass eine entsprechende Fehlerausgabe erfolgt, wenn es nicht auf reguläre Dateien angewendet wird.

Aufgabe 1.7 (Zeitfunktionen):

In dieser Übung erstellen Sie eine Bibliothek, mit der Sie Ausführungsdauern von Programmabschnitten ermitteln können. Diese Bibliothek sowie eine weitere werden im Laufe des Praktikums noch mehrfach zur Leistungsbeurteilung eingesetzt werden. Die Bibliothek soll wie folgt implementiert werden: das Modul `time/mytime.c` beinhaltet den Programmcode der Funktionen, während die Header-Datei `include/mytime.h` das Interface dieses Moduls (insbesondere Typen und Funktionen) für die Programmierung beschreibt. Die kompilierte Bibliothek wird als `lib/libmytime.a` anderen Modulen für den Link-Vorgang zur Verfügung gestellt. (vgl. Aufgabe 1.4)

(a) Klären Sie die Bedeutung der Systemdienste `getrusage()`, `times()` und `gettimeofday()`.

(b) Ermitteln Sie experimentell, z.B. durch wiederholtes Auslesen der Uhrzeit, die Granularität der Zeitmarken von `gettimeofday()`, d.h. was ist der kleinste (von 0 verschiedene) Abstand zweier aufeinander folgender Zeitmarken. Wie erklären Sie sich diesen Wert?

(c) Erstellen Sie basierend auf `gettimeofday()` ein C-Modul mit den folgenden Funktionen:

```
void start(struct tstamp *t);
void stop(struct tstamp *t);
unsigned long extime(struct tstamp *t);
```

Dabei soll die Timestamp-Struktur `tstamp` aus zwei `timeval`-Strukturen bestehen, die jeweils zur Speicherung einer Zeitmarke durch `gettimeofday()` dienen. Mit der Funktion `start()` wird die erste `timeval`-Struktur gefüllt, mit `stop()` die zweite. Die Funktion `extime()` gebe die zwischen den beiden gespeicherten Zeitmarken verstrichene Zeit (execution time) in Mikrosekunden zurück.

(d) Erzeugen Sie für eine wiederverwendbare Bibliothek aus der Quell-Datei `time.c` zunächst eine Objekt-Datei `time.o`. Fügen Sie diese mit dem Kommando `ar` in eine Objekt-Bibliothek `libsp.a` ein. (vgl. Aufgabe 1.4)

(e) Testen Sie Ihre Bibliothek zunächst mit dem Programm `timetest.c`, indem Sie die Ausführungsdauer einer Schleife ermitteln, in der die Zahlen 1..100 mit `printf()` ausgegeben werden. Man sagt auch, dass das Programm "instrumentiert" wird, d.h. es werden Messinstrumente hinzugefügt.

(f) Ermitteln Sie nun die Ausführungsdauer des eigentlichen Kopierabschnittes im Programm `mycp.c` aus Aufgabe 1.5 (d) mit Ihrer eigenen Bibliothek und wiederholen Sie damit die Messreihe in Aufgabe 1.5 (e).

Aufgabe 1.8 (Testhilfen):

In dieser Übung lernen Sie einige wichtige Hilfsmittel kennen, die auf Linux-Systemen zur Verfügung stehen und für das Testen von Programmen sehr hilfreich sein können.

(a) Klären Sie die Bedeutung der Utilities `ltrace` und `strace` (nicht Bestandteil der Single Unix Specification). Benutzen Sie sie z.B. mit der instrumentierten Version von `mycp` aus Aufgabe 1.7 (f). Was können Sie aus den Ausgaben ablesen?

(b) Vielleicht kennen Sie den auf UNIX-Systemen verwendeten Debugger `gdb` und sein graphisches Frontend `ddd` bereits, andernfalls haben Sie hier die Gelegenheit, damit erste Schritte zu tun.

- Welche Option müssen Sie für die Kompilation verwenden, damit ein anschließendes Debugging möglich ist? Was passiert in diesem Fall mit Ihrem Programm? Kompilieren Sie das instrumentierte Programm `mycp.c` entsprechend.
- Starten Sie den Debugger über dem Programm `mycp`.
- Führen Sie das Programm Statement für Statement aus.
- Setzen Sie einen Breakpoint vor die Ausführung des Aufrufs von `stop()` und lassen Sie das Programm bis dorthin laufen.
- Stellen Sie nun die verwendete Struktur `tstamp` mit ihren Komponenten graphisch dar.
- Ermitteln Sie die Inhalte der Zeitmarken in der verwendeten Struktur `tstamp` vor und nach dem Aufruf von `stop()`.
- Üben Sie den Umgang mit dem Debugger weiter und nutzen Sie ihn zukünftig zur Analyse von Programmierfehlern fehlerhafter Programme.