J. Gutenberg-Universität Mainz
Prof. Dr. Reinhold Kröger
Fabian Meyer M.Sc.

Betriebssysteme
WS 2012/13

Übungsblatt 2
Praktische Übungen

Aufgabe 2.1 (Ungepufferte Dateiein-/ausgabe):

In dieser Übung wird der Umgang mit Dateien mittels UNIX-Systemaufrufen durchgeführt. 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?

Aufgabe 2.2 (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 2.1 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 2.3 (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.

(a) Klären Sie die Bedeutung der Systemdienste `rusage()`, `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()` eine Bibliothek 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) Testen Sie Ihre Bibliothek zunächst, 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.

(e) Ermitteln Sie nun die Ausführungsdauer des eigentlichen Kopierabschnittes im Programm `mycp.c` aus Aufgabe 2.1 (c) mit Ihrer eigenen Bibliothek und wiederholen Sie damit die Messreihe in Aufgabe 2.1 (e).

Aufgabe 2.4 (Testhilfen):

In dieser Übung lernen Sie zwei 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 des Utilities `strace` (nicht Bestandteil der Single Unix Specification). Benutzen Sie es z.B. mit der instrumentierten Version von `mycp` aus Aufgabe 2.3 (e). 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.