Systemprogrammierung - AIN/2

Sommersemester 2022

Übungsaufgabe 7: POSIX Funktionen und Linux Prozesse

Abgabe bis 30.6./1.7.2022

Programmierung

- Erstellen Sie ein C-Programm `filesize`, das als Kommandozeilenargumente beliebig viele Dateinamen erwartet und für jede der Dateien die Größe in Byte auf die Standardausgabe schreibt.

  Bei einem Aufruf ohne Kommandozeilenargumente soll das Programm mit der POSIX-Funktion `read` byteweise von der Standardeingabe lesen und dabei die Anzahl der Bytes zählen. Andernfalls soll das Programm für die angegebenen Dateien zur Größenbestimmung die POSIX-Funktion `stat` verwenden. Siehe `man 2 read` und `man 2 stat` oder die POSIX-Dokumentation im Internet. Achten Sie auf eine korrekte Fehlerbehandlung mit Ausgabe der zugehörigen Systemmeldung. Damit die Systemmeldung in der auf dem Rechner (bzw. der beim Aufruf verwendeten Konsole) eingestellten Sprache erscheint, müssen Sie am Anfang des Programms folgenden Aufruf einfügen (siehe auch `man 3 setlocale`):

  ```
  setlocale(LC_ALL, "");
  ```

  Hinweise:
  Verwenden Sie das Vorlesungsbeispiel `count.c` als Vorlage. Ersetzen Sie im Fall der Standardeingabe den C-Bibliotheksaufruf `fgetc` durch den Aufruf der POSIX-Funktion `read` und bei Dateien die ganze Lese-/Zähl-Schleife durch den Aufruf der POSIX-Funktion `stat`. Den C-Bibliotheksaufruf `fopen` brauchen Sie nicht mehr (Warum?). Beachten Sie außerdem, dass POSIX für ganzahlige Datentypen in der Regel Aliasnamen verwendet. Ihr Programm muss damit richtig umgehen.

- Erstellen Sie ein C-Programm `filecopy`, das als Kommandozeilenargumente zwei Dateinamen erwartet und die erstgenannte Datei in die zweitgenannte kopiert.

  Das Programm soll zur Größenbestimmung die POSIX-Funktion `fstat` verwenden, daraufhin mit `malloc` einen Puffer in Dateigröße allokieren, die Quelldatei mit einem einzigen Aufruf der POSIX-Funktion `read` in den Puffer kopieren und schließlich den Puffer mit einem einzigen Aufruf der POSIX-Funktion `write` in die Zieldatei schreiben (siehe `man 2 fstat`, `man 2 read`, `man 2 write` oder die POSIX-Dokumentation im Internet).

  Achten Sie auf eine korrekte und vollständige Fehlerbehandlung mit für den Benutzer aussagekräftigen Fehlermeldungen. Stellen Sie für die Systemmeldungen die deutsche Sprache ein, um ein sprachliches Durcheinander zwischen Systemmeldungen und eigenen Fehlertexten zu vermeiden:

  ```
  setlocale(LC_MESSAGES, "de_DE.UTF-8");
  ```

  Geben Sie eine Warnung aus, wenn die Plattform die deutschen Systemmeldungen nicht unterstützt.

  Hinweise:
  Verwenden Sie das Vorlesungsbeispiel `copy.c` als Vorlage und ersetzen Sie die byteweise Kopierschleife durch die genannten Aufrufe von POSIX-Funktionen.
  Beachten Sie auch hier, dass die POSIX-Funktionen mit Aliasnamen für ganzzahlige Datentypen definiert sind. Ihr Programm muss damit richtig umgehen.

- Erstellen Sie ein möglichst einfaches `Makefile`, das die beiden Programme baut und die gewohnten Stilregeln einhält. Sie dürfen die eingebauten Variablen und Musterregeln von make verwenden (d.h. `make -R` braucht nicht zu funktionieren).

- Prüfen Sie Ihre beiden Programme mit `cppcheck --enable=warning,style --std=c11` und mit `valgrind` auf Probleme.

Test

Vergleichen Sie die Ausgaben Ihres Programms `filesize` mit denen des Linux-Kommandos `ls`:

```
./filesize xxx *
ls -l xxx *
```

Hinweise:
Statt `xxx` können Sie auch einen anderen Dateinamen verwenden, der im aktuellen Arbeitsverzeichnis nicht vorkommt.
*Der * ist das sogenannte Wildcard-Symbol und wird von der Shell durch alle Dateinamen des aktuellen Arbeitsverzeichnisses ersetzt.*

Kommt für die nicht existierende Datei `xxx` die gleiche Fehlermeldung?
In welcher Sprache erscheint die Systemmeldung?

Die aktuellen Spracheinstellungen können Sie in der Konsole mit dem Linux-Kommando `locale` abfragen (siehe `man 1 locale`)). Mit der Option `-a` zeigt das Kommando statt der Einstellungen die verfügbaren Locales an. Ändern Sie mal die Einstellung für die Systemmeldungen wie folgt:

```
export LC_MESSAGES=C
```

In welcher Sprache erscheinen jetzt die Fehlermeldungen von `filesize` und `ls`?

Testen Sie, ob Ihr Programm `filecopy` Dateien wirklich vollständig kopiert:

```
./filecopy filecopy.c filecopy-kopie.c
diff filecopy.c filecopy-kopie.c
```

Wie verhält sich `filecopy`,
wenn die Quelldatei nicht existiert?
wenn Sie kein Leserecht auf der Quelldatei haben?
wenn die Zieldatei bereits existiert?
wenn Sie kein Schreibrecht im Zielverzeichnis haben?

Arbeiten Sie auf der lokalen Festplatte, damit Sie die Rechte Ihrer Dateien und Verzeichnisse beliebig manipulieren dürfen:

```
cd
mkdir tmp
cd tmp
```

Kopieren Sie Ihre im Rahmen von Aufgabe 7 erstellten Dateien in das neue Verzeichnis `tmp` und sehen Sie sich die Zugriffsrechte der Dateien an:

```
ls -l
```

Hinweise:
Bei jeder Linux-Datei gibt es die Rechte `r` für Lesen, `w` für Schreiben und `x` für Ausführen (gewöhnliche Dateien) bzw. Durchsuchen (Verzeichnisse). Die Rechte gibt es jeweils drei Mal, für den Eigentümer, die Gruppe des Eigentümers und für sonstige Benutzer. Bei Kommandos und Funktionen werden die Rechte übrigens statt mit den Buchstaben oft oktal kodiert angegeben. Wie das funktioniert, können Sie z.B. im Wikipedia-Artikel Unix-Dateirechte nachlesen.

Ihr Leserecht auf einer Datei entfernen bzw. setzen Sie mit dem Kommando

```
chmod -r Datei
chmod +r Datei
```

Ihr Schreibrecht auf einem Verzeichnis entfernen bzw. setzen Sie mit dem Kommando

```
chmod -w Verzeichnis
chmod +w Verzeichnis
```

Protokoll

Erstellen Sie ein Protokoll Ihrer Tests. Gehen sie dazu so vor wie in **Aufgabe 1** beschrieben. Nennen Sie die Protokolldatei `protokoll-aufgabe7.txt` und ergänzen Sie darin Ihre Antworten auf alle im Testabschnitt gestellten Fragen.

Abgabe

Verpacken Sie alle Dateien Ihrer Lösung in ein Archiv:

```
tar cvzf aufgabe7.tar.gz Makefile filesize.c filecopy.c protokoll-aufgabe7.txt
```

Laden Sie das Archiv dann in Moodle hoch (siehe dort).

Hinweis:
Der Compiler `gcc` darf für Ihre Programme keine Fehler oder Warnungen mehr ausgeben.
Ihre Programme müssen außerdem ordentlich formatiert sein. Bessern Sie die Formatierung gegebenenfalls mit `astyle` nach:

```
astyle -p -H --style=ansi filesize.c filecopy.c
```

Umgang mit Linux-Prozessen *(Bearbeitung empfohlen, aber freiwillig)*

Sie brauchen für die Experimente Ihr Programm `filesize`:

```
make "CC=gcc -g" filesize
```

Hinweis: das Setzen der Variablen `CC` brauchen Sie im `make`-Aufruf nur, wenn Ihr Makefile die Debugoption `-g` nicht enthält

Linux verwaltet die gerade laufenden Programme als Prozesse. Die Prozesse können über eine fortlaufende Nummer, die PID (= Process Identifier), eindeutig identifiziert und manipuliert werden. Machen Sie dazu die folgenden Experimente. Sie benötigen zwei Konsolenfenster mit Arbeitsverzeichnis `Aufgabe7/`. Für `PID` müssen Sie in der folgenden Anleitung jeweils die mit `ps` ermittelte Prozessnummer einsetzen:

Konsole 1

```
# Programm filesize starten
```

Konsole 2

```
./filesize



# Programm filesize neu starten
./filesize
```

```
# die PID von filesize herausfinden
ps -a
# filesize gewaltsam beenden
kill -9 PID


# die PID von filesize herausfinden
ps -a
# Debugger mit dem laufenden Programm verbinden
gdb filesize PID
# analysieren Sie mit dem gdb den Programmstatus
# und setzen Sie das Programm dann schrittweise fort
```

Konsole 1

Konsole 2

```
# core-Dateien erlauben
ulimit -c unlimited
# Programm filesize neu starten
./filesize



# wie gross ist die core-Datei?
ls -l core

# Finden Sie mit dem gdb heraus,
# bei welcher Anweisung filesize beendet wurde
gdb filesize core

# core-Dateien brauchen viel Plattenplatz
rm core
```

```
# die PID von filesize herausfinden
ps -a
# filesize beenden und core-Datei erstellen
kill -6 PID
```

Hinweis: Der gdb hat ein Kommando `backtrace`. Sie können sich damit den Zustand des Aufrufstacks anzeigen lassen.

Prof. Dr. H. Drachenfels
Hochschule Konstanz - Impressum - Datenschutzerklärung

Letzte Änderung: 24.2.2022