Übungsblatt 10

Dr. Matthias Frank, Dr. Matthias Wübbeling

Ausgabe Mittwoch, 13. Dezember 2023
Abgabe bis
Freitag, 05. Januar 2024, 23:59 Uhr
Vorführung vom 8. bis zum 12. Januar 2023

Alle Programme müssen unter **Ubuntu 22.04** kompilierbar bzw. lauffähig und ausreichend kommentiert und mit **Makefile** (C, Assembler) versehen sein, um Punkte zu erhalten. Als Compiler sollen **clang** (C) und **nasm** (Assembler) verwendet werden. Die Lösungen sind bei Ihrem Tutor während Ihrer Übungsgruppen vorzuführen. Alle Gruppenmitglieder sollten die Abgabe erklären können. Die Abgabe erfolgt mittels Ihres Git-Repositories und der vorgegebenen Ordnerstruktur.

Die Punkte der Aufgaben sind relevant für die Zulassung. Die Punkte der Bonusaufgaben werden auf Ihren Punktestand addiert, werden aber nicht auf die für die Zulassung benötigten Punkte addiert.

Abgabestruktur: Jede Abgabe, die die folgende Struktur nicht umsetzt, wird **NICHT** bepunktet bzw. mit 0 Punkten bewertet

- die zu korrigierenden Lösungen müssen bis zur Deadline auf dem `master`-Branch liegen. Lösungen auf anderen Branches werden nicht gewertet
- alle Lösungen müssen in der vorgegebenen Ordnerstruktur (**blattXX/aufgabeYY**) abgelegt werden, wobei `XX` und `YY` durch die jeweiligen Nummern des Zettels und der Aufgaben ersetzt werden sollen. Der Name soll exakt nur aus diesen Zeichen bestehen und achtet auf Kleinschreibung
- sämtliche Aufgaben, mit Ausnahme der theoretischen Aufgaben, die eine PDF erfordern, benötigen zwingend ein **Makefile**. Abgaben ohne dieses werden nicht gewertet

Hinweis:	Manche Browser sind nicht dazu in der Lage die in die PDFs eingebetteten Dateien anzuzeigen. Mittels eines geeigneten Readers, wie dem Adobe Reader, lassen sich diese jedoch anzeigen und verwenden.

Aufgabe 1. Praktische IPC – Shellprogrammierung mit Pipes (4 Punkte)

Dem ambitionierten Linux-Benutzer begegnet ein IPC-Mechanismus im Alltag immer wieder: Pipes. Dies liegt daran, dass sich auf der Kommandozeile einzelne Programme sehr elegant miteinander koppeln lassen, indem man ihre Eingabe- und Ausgabestreams mit Hilfe von Pipes verbindet. Außer Pipes kann man auf der Kommandozeile aber auch FIFOs verwenden. Das Ziel dieser Aufgabe besteht darin, Sie mit den Mechanismen von Pipes und FIFOs auf der Kommandozeile anhand eines Beispiels vertraut zu machen.

a)

Machen Sie sich mit Pipes auf der Kommandozeile vertraut.
Beispiel: `ls -l | less`

- Übergibt die Ausgabe des `ls`-Befehls als Eingabe an das Betrachtungstool `less`
- `less` ermöglicht die komfortable Navigation in der Auflistung des Verzeichnisinhalts, die von `ls -l` generiert wurde.

b)

Machen Sie sich mit FIFOs auf der Kommandozeile vertraut.
Beispiele:

- Erzeugung eines FIFOs: `mkfifo /tmp/testfifo`
- Schreiben in eine FIFO: `cat test.txt > /tmp/testfifo`
- Lesen aus einer FIFO: `grep 'Donald Duck' < /tmp/testfifo`
- Löschen eines FIFOs: `rm /tmp/testfifo`

c)

Kombinieren Sie verschiedene Linux-Kommandozeilentools mit Pipes und FIFOs, so dass die folgende Aufgabe gelöst wird: Geben Sie eine Liste der zuletzt auf dem System angemeldeten Personen aus

1. in die Datei **lastusers.txt**. Sie soll eine Liste der Benutzernamen komplett in Großschreibung enthalten, in der Benutzernamen zwar mehrfach aber nicht mehrfach direkt hintereinander auftauchen können.
2. in die Datei **lasthosts.txt**. Sie soll eine Liste mit den IP-Adressen der Rechner enthalten, von denen die Benutzer aus eingeloggt waren. Diese Liste soll numerisch sortiert sein und keine Einträge von Benutzern enthalten, die noch eingeloggt sind.

Beachten Sie dabei:

- Das Tool `last` darf für beide Ausgaben zusammen nur einmal aufgerufen werden! (Tipp: FIFOs to the rescue . . .)
- Sie dürfen keine temporären Dateien verwenden. Erlaubt ist dagegen die Verwendung von FIFOs.
- Bei der Lösung dürfen Sie nur folgende Programme verwenden:
  - `last`
  - `grep`
  - `tee`
  - `mkfifo`
  - `sort`
  - `tr`
  - `head`
  - `rev`
  - `uniq`
  - `cut`

Darüber hinaus werden Sie eventuell den `&`-Operator benötigen, um einen Befehl im Hintergrund auszuführen.

Eine Dokumentation der Funktionaliät der Befehle finden Sie auf der jeweiligen man page. (Hinweis: Sie müssen nicht unbedingt alle dieser Tools verwenden!)

Aufgabe 2. Pipes in C (6 Punkte)

a)

Schreiben Sie ein C-Programm, dass eine Pipe erstellt. Nachdem die Pipe geöffnet wurde, soll ein Kindprozess erzeugt werden. Im Kindprozess können Sie nun das Schreib-Ende der Pipe schließen, im Elternprozess das Lese-Ende.

Schreiben Sie im Kindprozess eine Schleife, in der die Pipe mit `read` ausgelesen wird. Das, was Ihnen gesendet wurde, sollen Sie mit `write` ausgeben. Sie können dabei den Filedeskriptor der Standardausgabe erhalten, indem Sie `fileno(stdout)` verwenden. Lesen Sie so lange weiter, bis Sie eine Rückgabe ≤ 0 bekommen. Dies kann entweder einen Fehler bedeuten (bei einer negativen Rückgabe), oder es bedeutet, dass die andere Seite der Pipe geschlossen wurde.

Im Elternprozess schicken Sie mit `write` eine Nachricht in das Schreib-Ende der Pipe. Denken Sie daran, nach dem Schreiben mit `wait` auf den Kindprozess zu warten, damit das Programm nicht sofort terminiert!

b)

Erweitern Sie Ihr Programm jetzt so, dass Sie im Elternprozess mit `open` Ihre Quellcodedatei lesend öffnen. Der Inhalt soll in einer Schleife in einen von Ihnen anzulegenden Puffer gelesen und anschließend an den Kindprozess weitergeschickt werden, der den Inhalt der Datei ausgibt.

c)

Öffnen Sie nun im Kindprozess eine Datei mit dem Namen `copy.c`, in der Sie schreiben können. Falls diese Datei nicht existiert, soll Sie beim Öffnen erstellt werden. Nachdem Sie die Datei geöffnet haben, rufen Sie `dup2` auf. Dabei soll der erste Parameter der Deskriptor Ihrer neuen Datei sein; der zweite Parameter ist der Wert, den Sie von `fileno` für `stdout` bekommen haben. Was passiert mit Ihrer Ausgabe? Wie verändert sich die Datei? Erklären Sie, was hier passiert ist!

Aufgabe 3. Signale und sleep (4 Punkte)

Erstellen Sie ein C-Programm, das einen Signal-Handler für das Signal `SIGINT` registriert. Nachdem der Handler registriert ist, soll das Programm einige Sekunden mit `nanosleep` warten.

Während das Programm wartet, senden Sie über die Konsole das Signal `SIGINT`. Das können Sie mit der Tastenkombination Strg + C machen.

Das Programm soll einerseits angeben, dass der Signal-Handler ausgeführt wurde und welches Signal angekommen ist, und andererseits die verbleibende Zeit, die `nanosleep` noch hätte warten sollen. Anschließend soll das Warten für die verbleibende Zeit fortgesetzt werden.

Bonusaufgabe IV. Weihnachts-Bonus-Aufgabe (4 Punkte)

Schreiben Sie ein C-Programm, das einen Weihnachtsbaum in ASCII-Art auf der Kommandozeile ausgibt. Dabei sollen als Kommandozeilenparameter die Höhe des Baumes und die Anzahl der Kugeln übergeben werden. Der Aufruf in der Konsole erfolgt mit

```
./weihnachtsbaum 14 15
```

, wobei 14 die Höhe des Baums und 15 die Anzahl der Kugeln ist. Die Breite des Baums soll entsprechend der angegebenen Höhe skaliert werden. Die Kugeln sollen zufällig in dem Baum platziert werden. Ihr Baum für diesen Aufruf könnte zum Beispiel wie folgt aussehen:

```
          \ /
        -->*<--
          /o\
         /_\_\
        /_/_0_\
       /_o_\_\_\
      /_/_/_/_/o\
     /@\_\_\@\_\_\
    /_/_/0/_/_/_/_\
   /_\_\_\_\_\o\_\_\
  /_/0/_/_/_0_/_/@/_\
 /_\_\_\_\_\_\_\_\_\_\
/_/o/_/_/@/_/_/o/_/0/_\
          [___]
```

- Sie erhalten bis zu 3 Punkte, wenn die Aufgabe bis zur regulären Abgabefrist gelöst und ins Git-Repository eingespielt wird.
- Sie erhalten einen weiteren Punkt, wenn sie zwischen dem 24. – 26.12 zwei Lösungen (ASCII-Ausgaben) für zwei verschiedene Parametersets an die Sys-Prog-Mailingliste

  `vl-sys-prog@lists.iai.uni-bonn.de`

  schicken, mit dem Betreff „Weihnachtsbaum“. Die Bäume sollen als Anhang mitgeschickt werden, um Formatierungprobleme zu vermeiden.