Abgabefrist Übungsaufgabe 1

Die Lösung muss abhängig von der Tafelübung abgegeben werden, an der ihr teilnehmt:

- **W1** – eigene Übung U1 in KW17 (25.-27.04.): Abgabe bis **Donnerstag, den 05.05.2022 12:00**
- **W2** – eigene Übung U1 in KW18 (02.-04.05.): Abgabe bis **Montag, den 10.05.2022 12:00**

In darauffolgenden Tafelübungen werden teilweise einzelne abgegebene Lösungen besprochen, teilweise auch ein Lösungsvorschlag aus dem Tutorenteam.

Allgemeine Hinweise zu den BS-Übungen

- Ab jetzt ist es *nicht* mehr möglich, Einzelabgaben in AsSESS zu tätigen. Falls ihr (statt in einer Dreiergruppe) zu zweit oder zu viert abgeben möchtet, klärt das bitte *vorher* mit eurem Übungsleiter!
- Die Gruppenmitglieder **sollten nach Möglichkeit** gemeinsam an der gleichen Tafelübung teilnehmen. Es sind aber auch Abgaben mit Studierenden aus **verschiedenen Übungsgruppen** zulässig. Es gilt dabei immer die früheste Abgabefrist. Die Lösung wird jeweils komplett bewertet und den Gruppenmitgliedern gleichermaßen angerechnet.
- Die abgegebenen Antworten/Programme werden automatisch auf Ähnlichkeit mit anderen Abgaben überprüft. Wer beim Abschreiben[1] erwischt wird, verliert ohne weitere Vorwarnung die Möglichkeit zum Erwerb der Studienleistung in diesem Semester!
- Die Zusatzaufgaben sind ein Stück schwerer als die „normalen“ Aufgaben und geben zusätzliche Punkte.

Aufgabe 1: Prozessverwaltung und fork (10 Punkte)

Lernziel dieser Aufgabe ist die Verwendung der UNIX-Systemschnittstelle zum Erzeugen und Verwalten von Prozessen.

Theoriefragen: Prozessverwaltung (5 Punkte)

Bitte gebt diesen Aufgabenteil, wie auch in Aufgabe 0, in der Datei antworten.txt ab. Die Antworten sind in eigenen Worten zu formulieren.

1. Erklärt den Unterschied zwischen **fork(2)** und **vfork(2)**? Gibt es zwischen den beiden heutzutage effektiv noch einen Unterschied?
2. Betrachtet die folgenden beiden Kommandozeilen:

   ```
   ls -l > sort
   ls -l | sort
   ```

   Erklärt zunächst, was die beiden Kommandos jeweils tun. Worin besteht nun der Unterschied?
3. Welchen für ein Betriebssystem wichtigen Zweck erfüllt der Systemaufruf **wait(2)**?
4. Betrachtet folgendes C-Code-Schnipsel: **(!) nicht ausführen (!)**

   ```
   for (;;) fork();
   ```

[1] Da wir im Regelfall nicht unterscheiden können, wer von wem abgeschrieben hat, gilt das für Original **und** Plagiat.

Was ist das Problem bei der Ausführung dieses Codes? Beschreibt das Programmverhalten in der 1., 2., 3. und n. „Generation“. Legt zu Grunde, dass alle Prozesse (hier insbesondere obige for-Schleife) parallel ausgeführt werden. Betrachtet die Gesamtheit der parallelen Schleifendurchläufe als eine „Generation“.

Programmierung: Menü / Shell (5 Punkte)

In dieser Aufgabe soll ein einfaches Menü implementiert werden, das die Auswahl eines Parameters für das Programm *ls* ermöglicht. Gebt eure Lösung in der Datei shell_menue.c ab. Um euch die Arbeit zu erleichtern, haben wir die Aufgaben in kleine Schritte unterteilt.

Achtet bei jedem System- bzw. Bibliotheksaufruf auf eine geeignete Fehlerbehandlung – sofern erforderlich. **Dies schließt auch den korrekten Umgang mit falschen Nutzereingaben ein.** Lest euch im Zweifel die Man-Page dazu durch oder schaut in die Übungsfolien.

1. **Einlesen der Standardeingabe (1 Punkt)** Implementiert zunächst ein einfaches Menü, das eine Liste an nummerierten Argumenten auf der Kommandozeile ausgibt. Das Menü soll den Benutzer auffordern, eine Zahl einzugeben, um einen von drei möglichen Menüeinträgen auszuwählen. Anschließend soll lediglich ausgegeben werden, welche Wahl getroffen wurde. Wählbare Parameter können z.B. *-l*, *-a* und *-t* sein.

   Hinweis: Zur Ausgabe soll **printf(3)** und für die Eingabe **scanf(3)** verwendet werden.

2. **Starten von Programmen (3 Punkte)** Nun sollen nicht mehr nur die Argumente ausgegeben werden, sondern das Programm *ls* mit dem jeweiligen Argument auch gestartet werden. Die Ausführung des Programms soll in einem neuen Kindprozess geschehen. Der Elternprozess gibt nach der Terminierung des Kindes die PID des Kindes aus.

   Hinweis: Zum Erzeugen der Prozesse soll **fork(2)** und **execlp(3)** verwendet werden. Achtet darauf, keine verwaisten Prozesse oder Zombie-Prozesse zu hinterlassen.

3. **Schleife (1 Punkt)** Es soll jetzt möglich sein, das Programm *ls* mehrfach nacheinander mit unterschiedlichen Argumenten zu starten. Nach dem Ende eines Kindprozesses soll dazu erneut das Menü mit der Auswahl erscheinen. Zum Beenden des Menüs soll ein vierter „Eintrag“ *exit* dienen.

Beispiel für Programmaufruf:

```
$ ./shell_menue
1. -l
2. -a
3. -t
4. exit
Auswahl: 1
Es wurde -l gewählt
total 853
-rw-r--r-- 1 studi studi      853 Apr 20 09:42 bar.txt
PID von pwd: 139
1. -l
2. -a
3. -t
4. exit
Auswahl: 4
Es wurde exit gewählt
```

Zusatzaufgabe 1: Ausführen eines beliebigen Arguments (2 Sonderpunkte)

Fügt zu euren bestehenden Menüeinträgen einen weiteren Eintrag hinzu, welcher die Möglichkeit bietet *ls* mit *einem* beliebigen Argument auszuführen. Dazu müsst ihr, nachdem der Menüeintrag ausgewählt wurde, mithilfe von **scanf(3)** einen String einlesen, was durch das Formatzeichen „%s“ möglich ist. Beachtet dabei die Größe eures Puffers und die Länge der Eingabe. Überprüft nach dem Aufruf von **execlp(3)**, ob die Ausführung erfolgreich war und gebt dies entsprechend aus. Die Verarbeitung von Optionen und Argumenten ist **nicht** erforderlich.

Tipps zu den Programmieraufgaben:

- Kommentiert euren Quellcode ausführlich, so dass wir auch bei Programmierfehlern im Zweifelsfall noch Punkte vergeben können!
- Denkt daran, dass viele Systemaufrufe fehlschlagen können! Fangt diese Fehlerfälle ab (die Aufrufe melden dies über bestimmte Rückgabewerte, siehe die jeweiligen man-Pages), gebt geeignete Fehlermeldungen aus (z.B. unter Zuhilfenahme von **perror(3)**) und beendet euer Programm danach ordnungsgemäß.
- Die Programme sollen sich mit dem gcc auf den Linux-Rechnern im IRB-Pool übersetzen lassen. Der Compiler ist dazu z.B. mit folgenden Parametern aufzurufen:
  `gcc -std=c11 -Wall -o shell_menue shell_menue.c`
  Alternativ *könnt* ihr die Programme auch in C++ schreiben, der Compiler ist dazu z.B. mit folgenden Parametern aufzurufen:
  `g++ -Wall -o shell_menue shell_menue.c`
  Weitere (nicht zwingend zu verwendende) Compilerflags, die dafür sorgen, dass man sich näher an die Standards hält, sind: `-Wpedantic -Werror`