

Weitere 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 abgegebenen Antworten/Programme werden automatisch auf Ähnlichkeit mit anderen Abgaben überprüft. Wer beim Abschreiben¹ 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: Prozesse / Shell (4 Punkte)

1. Wie unterscheiden sich die folgenden Kommandos?
 - a) `cat a.txt > b.txt`
 - b) `cat a.txt >> b.txt`
2. Erklärt in eigenen Worten die Ausgabe des folgenden Kommandos. Tipp: Führt es in einem Verzeichnis mit mindestens einer PDF-Datei aus.

`ls | grep -c .pdf`
3. Was versteht man unter einem „verwaisten“ Prozess? Wie verändert sich beim „Verwaisen“ die Prozesshierarchie?
4. Betrachtet folgendes C-Code-Schnipsel: /!\ **nicht ausführen** /!\

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

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“.

⇒ `antworten.txt`

Programmierung: Menü / Shell (6 Punkte)

In dieser Aufgabe soll ein einfaches Menü implementiert werden. Gebt eure Lösung in einer Datei 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.

⇒ `shell_menu.c`

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

a) Einlesen der Standardeingabe (1 Punkt) Implementiert zunächst ein einfaches Menü, das nummerierte Programmnamen 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. Die Programmeinträge können z.B. *ps*, *date* und *ls* sein.

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

b) Starten von Programmen (3 Punkte) Nun sollen nicht mehr nur die Programmnamen ausgegeben werden, sondern die jeweiligen Programme auch gestartet werden. Die Ausführung des ausgewählten 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 Zombies zu hinterlassen.

c) Schleife (1 Punkte) Es soll jetzt möglich sein, mehrere Programme nacheinander zu starten. Nach dem Ende eines Kindprozesses soll dazu erneut das Menü mit der Auswahl erscheinen. Zum Beenden des Menüs soll ein vierter „Programmeintrag“ *exit* dienen.

d) Exit-Status abfragen und ausgeben (1 Punkt) Sobald Teilaufgabe c) funktioniert, erweitert die Funktionalität so, dass der Exit-Status eines Kindprozesses durch den Elternprozess abgefragt wird. Dieser soll anschließend mit einer Meldung ausgegeben werden. Sollte der Kindprozess nicht ordnungsgemäß terminiert worden sein, gebt dies mit einer zusätzlichen Meldung aus.

Zusatzaufgabe: Argumente (2 Sonderpunkte)

Ändert das Programm so ab, dass nach Auswahl eines Befehls zusätzlich Argumente angegeben werden können. Mehrere Argumente werden dabei durch je ein Leerzeichen getrennt angegeben. Es müssen nicht mehr als 100 Zeichen oder 10 Argumente verarbeitet werden.

Hinweis: Schaut euch dazu die Funktionen **fgets(3)**, **strtok(3)** und **execvp(3)** an. Wenn die Argumente mit **fgets(3)** eingelesen werden, muss evtl. `'\n'` am Ende entfernt werden.

⇒ `shell_menu_extended.c`

Beispiel: Programmausgabe Aufgabe b)

```
$ ./1b
1. ps
2. date
3. ls
Auswahl: 2
Mo 22. Apr 12:34:56 CEST 2019
PID 1234
```

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 mit folgenden Parametern aufzurufen:
`gcc -std=c11 -Wall -o ziel datei.c`
Weitere (nicht zwingend zu verwendende) nützliche Compilerflags sind:
`-Wpedantic -Werror -D_POSIX_SOURCE`
- Achtet darauf, dass sich der Programmcode ohne Warnungen übersetzen lässt; z.B. durch Nutzung von `-Werror`.
- Alternativ kann auch der GNU C++-Compiler (*g++*) verwendet werden.

Abgabe: bis spätestens Freitag, den 22. Mai 08:00 (Übungen in gerader Kalenderwoche) bzw. Dienstag, den 26. Mai 08:00 (Übungen in ungerader Kalenderwoche).