Weitere Hinweise zu den Übungen Betriebssysteme

- 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 Abschreiben[1] erwischt wird, verliert ohne weitere Vorwarnung die Möglichkeit zum Erwerb der Studienleistung in diesem Semester!
- Die optionalen Aufgaben (davon wird es jeweils eine auf den Aufgabenblättern 1–3 geben) sind ein Stück schwerer als die „normalen“ und geben *keine* zusätzlichen Punkte für das jeweilige Aufgabenblatt – aber jeweils ein „Bonus-Sternchen“ ★. Wenn ihr alle drei Sternchen sammelt, müsst ihr das letzte Aufgabenblatt (A4) nicht bearbeiten!

Aufgabe 1: Prozesse verwalten (10 Punkte)

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

Theoriefragen: Prozessverwaltung (4 Punkte)

Bitte gebt diesen Aufgabenteil, wie auch in Aufgabe 0, in der Datei `antworten.txt` ab.

1) Beschreibt in eigenen Worten den Unterschied zwischen **fork(2)** und **vfork(2)**.

2) Wozu dient der ">"-Operator in einer UNIX-Shell?

3) Was ist ein verwaister Prozess? Was ist der Unterschied zu einem Zombie-Prozess?

4) Was ist der Unterschied zwischen leichtgewichtigen und schwergewichtigen Prozessen?

Programmierung: Menü (6 Punkte)

In dieser Aufgabe soll ein einfaches Menü implementiert werden. Hierbei sollt ihr für jede Teilaufgabe eine eigene Datei anlegen und die vorherige erweitern. Legt dazu die Datei `menue_a.c` an und erstellt darin euer Programm für Teilaufgabe a). Sobald ihr diese Aufgabe gelöst habt, kopiert ihr `menue_a.c` in die Datei `menue_b.c` und erweitert den Code um Teilaufgabe b). Bei den restlichen Teilaufgaben geht ihr analog vor, so dass `menue_d.c` am Ende das vollständige Programm enthält.

a) Benutzung von scanf Implementiert zunächst ein einfaches Menü. Das Programm soll den Benutzer auffordern, einen Menüpunkt auszuwählen und anschließend, abhängig von der Eingabe, einen Programmnamen mittels **printf(3)** ausgeben. Das Menü soll die folgenden Eingaben akzeptieren: Bei der Eingabe von `'t'`, `'p'` oder `'x'` sollen die Programmnamen (Strings) „`thunar`“, „`ps`“ oder „`xterm`“ ausgeben werden. Wenn ein Programmname ausgegeben wurde, soll der Benutzer erneut zu einer Eingabe aufgefordert werden, bis er das Programm mit <**Ctrl**+**c**> abbricht. Ein Wert vom Typ *char* kann mit Hilfe des **scanf(3)**-Formatzeichen „%c“ in eine Variable vom Typ *char* eingelesen werden.

Zusätzlich soll eine Fehlerfunktion `void error(char *str)` implementiert werden, die einen String entgegen nimmt und als minimalistische Fehlerbehandlung **perror(3)** und **exit(3)** aufruft. Benutzt diese Fehlerfunktion bei allen eingesetzten Systemaufrufen zur Fehlerbehandlung in allen Teilaufgaben.

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

b) Programme starten Nun sollen nicht mehr die Programmnamen ausgegeben, sondern die jeweiligen Programme gestartet werden. Euer Quellcode soll nun unter dem Namen `menue_b.c` so abgeändert werden, dass nach der Eingabe von `'t'`, `'p'` oder `'x'` das entsprechende Programm ausgeführt wird (**thunar(1)**, **ps(1)** oder **xterm(1)**)[2]. Um das zu erreichen, soll nach der Eingabe ein neuer Prozess erzeugt werden (**fork(3)**), der das Ausführen des Programms übernimmt (**execlp(3)**). Der Vaterprozess soll nach dem Aufruf von **fork(3)** die *Prozess-ID* des Kindprozesses ausgeben und dann wieder auf eine Eingabe warten während parallel dazu der Kindprozess das gewählte Programm abarbeitet.

c) Prozesse wegräumen Sobald Teilaufgabe b) funktioniert, beobachtet die mit eurer *User-ID* gestarteten Prozesse (**ps(1)** oder **top(1)**). Die gestarteten Programme, die bereits ausgeführt und terminiert sind, werden hier immer noch gelistet: Sie verbleiben im System, solange sie nicht mit **wait(2)** bzw. **waitpid(2)** weggeräumt werden.

Löst das Problem in `menue_c.c` und gebt jeweils die *Prozess-ID* des dahingeschiedenen Kindes aus.

d) Signal-Handler (optional) ★ In dieser Zusatzaufgabe (`menue_d.c`) sollt ihr einen Signal-Handler implementieren, der im Vaterprozess aufgerufen wird, wenn ein Kindprozess das Signal `SIGCHLD` sendet. Benutzt dazu die in der Übung vorgestellte Funktion **sigaction(2)**. Der von euch zu implementierende Signal-Handler soll dann mittles **wait(2)** den Prozess wegräumen, die *Prozess-ID* ausgeben und über die in der man-page zu **wait(2)** definierten Macros ausgeben, ob der Prozess normal oder durch ein Signal terminiert ist.

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 dem ANSI-C- und POSIX-Standard entsprechen und sich mit dem gcc auf den Linux-Rechnern im FBI-Pool übersetzen lassen. Der Compiler ist dazu mit folgenden Parametern aufzurufen: *gcc -Wall -o ziel datei1.c*
  Weitere (nicht zwingend zu verwendende) Compilerflags, die dafür sorgen, dass man sich enger an die Standards hält, sind: `-ansi -pedantic -Werror`
- Alternativ kann auch der GNU C++-Compiler (*g++*) verwendet werden.

Abgabe: bis spätestens Donnerstag, 16. Mai 12:00 (Übungsgruppen 1/3/5/...) bzw. Dienstag, 21. Mai 12:00 (Übungsgruppen 2/4/6/...).

[2] Falls eines der Programme bei euch nicht installiert ist, könnt ihr auch ein beliebiges anderes Programm verwenden.