Weitere Hinweise zu den Betriebssysteme-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 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 0–4 geben) sind ein Stück schwerer als die „normalen“ und geben *keine* zusätzlichen Punkte für das jeweilige Aufgabenblatt – aber jeweils ein „Bonus-Sternchen“ ★. Wenn ihr drei Sternchen sammelt, müsst ihr das letzte Aufgabenblatt (A5) nicht bearbeiten.
- Am Besten arbeitet ihr in diesem Blatt zuerst die Theoriefragen durch; die Programmieraufgaben bauen z.T. stark auf dem Theorieteil auf.
- Führt den C-Code-Ausschnitt aus Theoriefrage 4 nicht aus! Eine theoretische Betrachtung ist zum Lösen der Aufgabe mehr als ausreichend.

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 in eigenen Worten die Ausgabe des folgenden Kommandos. **Tipp:** Führt es in einem Verzeichnis mit mindestens einer PDF-Datei aus.

```
ls | grep -c .pdf
```

2) Erklärt in eigenen Worten, was mit Zombies im Kontext von Betriebssystemen gemeint ist und was man dagegen tun kann.

3) Weshalb kann man in C den Inhalt von Strings nicht mit **==** vergleichen?

4) Betrachtet folgendes C-Code-Schnipsel: **/!\ nicht ausführen /!**

for(;;) fork();

a) Beschreibt das Programmverhalten nach 1, 2, 3 und n „Generationen“. Legt zu Grunde, dass alle Prozesse (hier insbesondere obige for-Schleife) parallel ausgeführt werden. Betrachtet die Gesamtheit der parallelen Schleifendurchläufe als eine „Generation“.

b) Das Verhalten eines solchen Programms kann zu Problemen führen und mit Hilfe der Datei */etc/security/limits.conf* beschränkt werden. Welche Probleme sind dies und warum sind Gegenmaßnahmen sinnvoll? Gebt ferner beispielhaft Einträge für diese Datei an, die den Benutzer „tux“ hart auf 100 und die Gruppe „guests“ hart auf 2000 Prozesse limitieren (siehe auch `limits.conf(5)`).

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

Programmierung: Euer eigenes Unix-Kommando (2+1+1+1 Punkte)

An dieser Stelle wollen wir das Unix-Kommando **watch(1)** in einer leicht vereinfachten Art nachprogrammieren. Dazu solltet ihr euch zunächst einmal **watch(1)** durchlesen und euch (auf der Kommandozeile) mit dem Programm vertraut machen. Am einfachsten geht das, indem ihr das Programm **ls** überwacht und währenddessen im gleichen Verzeichnis eine neue Datei erstellt.

a) Einlesen des zu überwachenden Befehls Implementiert zunächst eine Abfrage, welche den Benutzer ein Kommando und genau einen Parameter eingeben lässt. Gebt daraufhin zum testen sowohl das Kommando als auch den Parameter aus und beendet das Programm anschließend.

Eine Beispieldurchlauf des Programms könnte folgendermaßen aussehen:

```
luke@deathstar ~$ ./watch
Zu beobachtenes Programm mit einem Parameter eingeben:  ls -l
Kommando: ls, Parameter: -l
luke@deathstar ~$
```

Hinweise:

- Setzt für alle Strings eine maximale Größe; ein guter Wert dafür sind 256 Zeichen. Denkt daran, dass Strings in C nullterminiert sind
- Fügt geeignete Fehlerabfragen hinzu! *(Dies gilt auch für alle weiteren Aufgaben)*
- Die Implementierung soll in der Datei `aufgabe1_a.c` abgegeben werden

b) Starten von Programmen Erweitert euer Programm soweit, dass die Shell den Befehl mit dem übergebenen Argument mittels **execlp(3)** ausführt.

Hinweis:

- Wenn ihr die erste Teilaufgabe nicht gelöst habt, könnt ihr davon ausgehen, dass als Befehl `ls` und als Argument `-l` übergeben wurde.
- Die Implementierung soll in der Datei `aufgabe1_b.c` abgegeben werden

c) Periodische Ausgabe des Kommandos Bisher wird das eingegebene Kommando nur einmal gestartet und das Programm danach beendet. Sorgt mittels **fork(2)**, **sleep(3)** und **wait(2)** oder **waitpid(2)** dafür, dass es nun alle **5** Sekunden ausgeführt wird. Anders als bei **watch(1)** sollt ihr allerdings nicht den Inhalt des Terminals nach jeder Ausführung löschen, sondern die jeweiligen Ausgaben z.B. durch `printf("----\n")` voneinander trennen.

- Die Implementierung soll in der Datei `aufgabe1_c.c` abgegeben werden

d) Variable Periode

Beim Starten eures Programms soll es nun optional möglich sein, mit dem Parameter `-n` *Zahl* die Zeit zwischen den Ausführungen festzulegen. Sollte kein solcher Parameter übergeben werden, soll wie in Teilaufgabe c) das Intervall auf 5 Sekunden gesetzt werden.

Zum Beispiel soll der folgende Aufruf das eingegebene Programm alle zehn Sekunden ausführen:

```
luke@deathstar ~$ ./watch -n 10
```

Hinweise:

- **atoi(3)** wandelt einen String in eine Zahl um
- Die Implementierung soll in der Datei `aufgabe1_d.c` abgegeben werden

e) Ausgabe des Programms umleiten (optional) (★) Bisher wird die Ausgabe des Programms auf der Standardausgabe ausgegeben und so in eurem Terminalfenster dargestellt. Nun sollt ihr die Standardausgabe in die Datei `out.txt` im aktuellen Arbeitsverzeichnis umleiten. Gebt dem Benutzer trotzdem noch ein Feedback, wenn ein Durchlauf des Programms beendet wurde. Darüber hinaus soll das Programm nun nicht mehr endlos laufen, sondern nach einer bestimmten Anzahl von Aufrufen beendet werden. Dazu soll der Parameter `-i` mit der Anzahl der gewünschten Iterationen eurem Programm übergeben werden können. Den Parameter **-n** aus der vorherigen Aufgabe braucht ihr **nicht** mehr einlesen (ihr könnt also wieder ein festes Intervall wählen).
Beispielaufruf für sieben Programmaufrufe:

```
luke@deathstar ~$ ./watch -i 7
```

Hinweise:

- Schaut euch die Funktion **dup2(2)** an
- Ihr sollt zur Lösung dieser Aufgabe **keine** Shellkommandos verwenden
- Die Implementierung soll in der Datei `aufgabe1_e.c` abgegeben werden

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`
- Alternativ kann auch der GNU C++-Compiler (*g++*) verwendet werden.

Abgabe: bis spätestens Donnerstag, den 10. Mai 8:00 (Übungen in gerader Kalenderwoche) bzw. Dienstag, den 15. Mai 8:00 (Übungen in ungerader Kalenderwoche).