

6. Übung zu Betriebssysteme (Prof. Dr. Kremer)

Aufgabe 1: Schreiben Sie ein C-Programm,

1. das eine beliebige Umgebungsvariable einliest und deren Inhalt auf dem Bildschirm ausgibt.
2. das sämtliche Aufrufparameter und Umgebungsvariablen auf dem Bildschirm ausgibt.
3. das ein beliebiges Shell-Kommando ausführt.
4. das das Arbeitsverzeichnis anzeigt und nach Eingabe eines Zielpfades in ein anderes Verzeichnis verzweigt und dann das Arbeitsverzeichnis erneut anzeigt.

Aufgabe 2: Schreiben Sie ein C-Programm, das mit `fork()` einen Kind-Prozess erzeugt und anschließend endet, während der Kind-Prozess weiterläuft und von `init` adoptiert wird. Zeigen Sie diese Adoption mit dem Kommando `ps`.

Aufgabe 3: Schreiben Sie ein C-Programm, das mit `fork()` einen Kind-Prozess erzeugt, der sofort endet, während der Eltern-Prozess weiterläuft aber keinen `wait()`-Aufruf auf das Ende des Kindes macht. Zeigen Sie den entstandene Zombie-Prozess mit `ps` an und versuchen Sie, ihn oder den Elternprozess mit `kill` zu beenden.

Aufgabe 4: Schreiben Sie ein C-Programm, das mit `fork()` einen Kind-Prozess erzeugt und anschließend auf dessen Ende mit `wait()` wartet. Hierbei soll der `exit`-Kode des Kindprozesses vom Eltern-Prozess ausgegeben werden.

Aufgabe 5: Schreiben Sie ein C-Programm (ohne `fork()`), das sich mit einem Aufruf aus der `exec`-Familie mit einem anderen Programm überlagert. Dabei soll das Programm versuchen bei Eingabe von 0 `who`, 1 `ls`, 2 `date` und von 3 `xxx` ein Programm, das es nicht gibt, zu starten. Bei 3 soll eine Fehlermeldung Ihres Programms erscheinen.

Aufgabe 6: Benutzen Sie das Programm der vorigen Aufgabe, um mit `fork()`, `exec()` und `wait()` ein kleine Menü-Shell zu erzeugen, wobei der Elternprozess in einer Schleife immer wieder die Auswahl 0 `who`, 1 `ls`, 2 `date`, 3 `xxx` und 4 Ende anbietet. Bei einer falschen Eingabe endet die Menü-Shell mit dem `Exit`-Kode 1. Der Elternprozess gibt jeweils den `Exit`-Status des Kind-Prozesses aus.

Aufgabe 7: Schreiben Sie ein C-Programm mit einem Interrupt-Handler für die Signale `SIGINT` und `SIGQUIT` mit folgendem Verhalten: Das erste Senden von `SIGINT` (bzw. `SIGQUIT`) wird vom Interrupt-Handler abgefangen (Ausgabe mit `printf()`). Weitere `SIGINT` (bzw. `SIGQUIT`) Signale werden ignoriert. Ein Aufruf von `SIGQUIT` aktiviert den Interrupt-Handler für `SIGINT` wieder und umgekehrt.