1. Prozesserzeugung in UNIX (C-Programmierung)

1.1 Schreiben Sie ein C-Programm **hawsh**, das die Funktionalität einer (stark eingeschränkten) Shell besitzt. Die HAW-Shell soll dabei folgende Eigenschaften aufweisen:

1. Die Shell gibt einen Prompt-String aus, in dem das aktuelle Arbeitsverzeichnis und der Name des aktuellen Benutzers enthalten sind.
2. Der Benutzer kann danach den Namen eines in die Shell eingebauten "Built-In" -Befehls (Liste s.u.) oder den Namen einer beliebigen Programmdatei eingeben (ohne Optionen und ohne Argumente!)
3. Die Shell interpretiert anschließend den angegebenen Befehl und führt ihn aus. Falls kein "Built-In" - Befehl erkannt wurde, erzeugt die Shell einen neuen Prozess und veranlasst das Laden der Programmdatei anhand des übergebenen Namens.
4. Falls das letzte Zeichen eines (Nicht-Builtin-)Befehls ein „&“ ist (ohne Leerzeichen als Zwischenraum!), wartet die Shell nicht auf die Beendigung des Befehls, sondern meldet sich sofort zurück (d.h. der Befehl wird im Hintergrund ausgeführt).
5. Weiter bei Punkt 1.

Es ist dafür zu sorgen, dass bei Fehlersituationen der Benutzer ausreichend informiert und ein stabiler Zustand erreicht wird.

Folgende "Built-In" - Befehle soll die Shell selbst bereitstellen:

Name des Befehls	**Wirkung des Befehls**
`Quit`	Beenden der HAW-Shell
`version`	Anzeige des Autors und der Versionsnummer der HAW-Shell
`/[Pfadname]`	Wechsel des aktuellen Arbeitsverzeichnisses (analog zu `cd`). Es muss immer ein kompletter Pfadname eingegeben werden.
`Help`	Anzeige der möglichen Built-In-Befehle mit Kurzbeschreibung

Beispiel-Dialog (Benutzereingaben sind *kursiv* dargestellt, Systemrückmeldungen sind nur Beispiele):

```
$ hawsh
/home/Franz - Was willst du, Franz? ps
4702 ..... bash
4718 ..... hawsh
/home/Franz - Was willst du, Franz? ps&
/home/Franz - Was willst du, Franz?
4702 ..... bash
4718 ..... hawsh
version
HAW-Shell Version 0.987 Autor: Max Muster
/home/Franz - Was willst du, Franz? /home/Franz/BS
Neues Arbeitsverzeichnis: /home/Franz/BS
/home/Franz/BS - Was willst du, Franz? quit
... und tschüß!
$
```

1.2. **Testen** Sie Ihr hawsh-Programm bzgl. aller Built-In-Befehle, der UNIX-Befehle `date`, `ls` und `env` sowie eines nicht existierenden Befehls (z.B. `abcde`), und zwar jeweils mit Ausführung im Vordergrund und im Hintergrund (&)!
Zur Analyse und Fehlersuche können Sie auch geeignete Linux-Befehle aus Versuch 1 anwenden.

Tipps:

- Verwenden Sie den Beispielcode aus der Vorlesung (auf Folie) als Vorlage!
- Beachten Sie die Tipps zur String-Behandlung aus Aufgabe 1!
- Folgende Bibliotheksfunktionen sind zur Lösung hilfreich (Dokumentation über `man`−Befehl): `getenv, setenv, chdir, getcwd, strcmp, strlen, fork, waitpid,` **`execlp`**`, exit` (`system` darf **nicht** benutzt werden!)

2. Threads (JAVA-Programmierung)

2.1 Schreiben Sie ein JAVA-Programm **CrossRace**, welches mit Hilfe von Threads ein Motorrad-Crossrennen simuliert. Das Programm soll als Konstanten die Anzahl an Motorradfahrern sowie die Länge der Strecke (in Runden) definieren. Direkt nach Erzeugung eines Motorrad-Threads (Klasse **`Motorbike`**) kann dieser gestartet werden. Die Fortbewegung der simulierten Motorräder soll so erfolgen, dass die Zeit, die für eine Runde benötigt wird, jeweils durch eine Zufallszahl („random“) bestimmt wird, d.h. nach jeder „gefahrenen“ Runde wird die Zeit für die nächste Runde „ausgewürfelt“ (0 <= Rundenzeit < 100 ms) und der Motorrad-Thread solange angehalten („sleep“). Jedes Motorrad soll seine eigene Gesamtlaufzeit messen.
Sobald alle Motorräder im Ziel sind (nicht vorher!), soll auf der Konsole eine Gesamtergebnistabelle ausgegeben werden, in der die Motorräder in der Platzierungs-Reihenfolge mit der entsprechend benötigten Gesamtlaufzeit ausgegeben werden. Beispiel für ein Rennen mit 5 Motorrädern:

```
**** Endstand ****
1. Platz: Motorrad 1 Zeit: 376
2. Platz: Motorrad 0 Zeit: 478
3. Platz: Motorrad 3 Zeit: 546
4. Platz: Motorrad 2 Zeit: 611
5. Platz: Motorrad 4 Zeit: 626
```

2.2 Erweitern Sie das Programm dahingehend, dass Stürze auftreten können. Ein Sturz ist zu jeder Zeit während des Rennens möglich und führt dann zu einem sofortigen Rennabbruch. Ein Sturz braucht dabei nicht einem bestimmten Motorrad zugeordnet werden. Simulieren Sie Stürze deshalb z.B. so, dass nach einer zufälligen Zeit ein Rennabbruch wegen Sturz ermittelt wird. Sollte dies geschehen, bevor alle Motorräder im Ziel sind, muss das Rennen sofort abgebrochen werden (d.h. alle Motorräder, die noch im Rennen sind, müssen sofort anhalten). Eine Ergebnisausgabe erfolgt dann nicht und das Programm kann beendet werden.
*Tipp: Verwenden Sie einen weiteren Thread (Klasse **`Accident`**)!*

Tipps:

- Orientieren Sie sich am **Beispielcode aus der Vorlesung** (`interrupt`(), `join`(), ..)!
- Jeder Thread kann auf alle public-Variablen aller Klassen zugreifen und die public-Methoden aller Klassen ausführen (falls nicht durch Synchronisationsmechanismen verhindert – die Verwendung von Synchronisationsmechanismen ist in dieser Aufgabe aber **nicht** notwendig!).
- Die Methoden `stop`, `suspend` und `resume` dürfen **nicht** verwendet werden, ebenso sollten Sie **noch keine Synchronisationsfunktionen** (`wait`, `notify, ...`) einsetzen.

Viel Spaß!! ☺