Aufgabenblatt 2

Shell- und Threadprogrammierung

1 Prozesserzeugung in UNIX (C-Programmierung)

1.1 Erstellen der HAW-Shell

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). Der Pfadname beginnt immer im Homeverzeichnis (Zugriff über Umgebungsvariable $HOME).
`help`	Anzeige der möglichen Built-In-Befehle mit Kurzbeschreibung

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

```
$ hawsh
/home/Franz - hawsh$ ps
4702 ..... bash
4718 ..... hawsh
/home/Franz - hawsh$ ps&
/home/Franz - hawsh$
4702 ..... bash
4718 ..... hawsh
/home/Franz - hawsh$ version
HAW-Shell Version 0.987 Autor: Max Muster
/home/Franz - hawsh$ ~/dsadBS
Fehler: Verzeichnis nicht gefunden
/home/Franz - hawsh$ ~/Franz/BS
Neues Arbeitsverzeichnis: /home/Franz/BS
/home/Franz/BS - hawsh$ quit
... und tschüss!
$
```

1.2 Testen der HAW-Shell

Testen Sie Ihr hawsh-Programm bzgl. aller Built-In-Befehle, der UNIX-Befehle `date`, `ls`, `ps` und `env` sowie eines nicht existierenden Befehls (z.B. `abcde`), und zwar jeweils mit Ausführung im Vordergrund und im Hintergrund (&)! Warum kann die Ausgabe bei Befehlen mit & Suffix von der im Beispiel gegebenen Ausgabe abweichen?

Tipps:

- Orientieren Sie sich am Beispielcode aus der Vorlesung
- Beachten Sie die Tipps zur String-Behandlung aus Aufgabe 1
- Folgende Bibliotheksfunktionen sind ggf. für die Lösung hilfreich (Dokumentation über man-Befehl): `getenv, setenv, chdir, getcwd, strcmp, strlen, strncpy, fork, waitpid, execlp, exit`
- **Achtung:** `system` darf **nicht** benutzt werden!)

2 Threads mit Java

2.1 Erstellen des Programms

Schreiben Sie ein Java-Programm **PodRacer**, welches mit Hilfe von Threads ein futuristisches Pod-Rennen simuliert (siehe https://starwars.fandom.com/wiki/Podracing). Das Programm soll als Konstanten die Anzahl an teilnehmenden Fahrer*innen sowie die Länge der Strecke (in Runden) definieren. Direkt nach Erzeugung eines Podracer-Threads (Klasse **Pod**) kann dieser gestartet werden. Die Fortbewegung der simulierten Rennfahrzeuge 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 Podracer-Thread so lange angehalten („sleep“). Jeder Pod soll seine eigene Gesamtlaufzeit messen.

Sobald alle Podracer im Ziel sind (nicht vorher!), soll auf der Konsole eine Gesamtergebnistabelle ausgegeben werden, in der die Podracer in der Platzierungs-Reihenfolge mit der entsprechend benötigten Gesamtlaufzeit ausgegeben werden. Beispiel für ein Rennen mit 5 Rennfahrzeugen:

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

2.2 Erweiterung: Crashs

Erweitern Sie das Programm dahingehend, dass Crashs 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 Podracer 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 Pods im Ziel sind, muss das Rennen sofort abgebrochen werden (d.h. alle Pods, die noch im Rennen sind, müssen spätestens bei Beendigung der aktuellen Runde anhalten). Eine Ergebnisausgabe erfolgt dann nicht und das Programm wird beendet.

*Tipp: Verwenden Sie einen weiteren Thread (z.B. **accident**)!*

Tipps:

- Verwenden Sie die in Java vorgesehenen Methoden zur Thread-Steuerung: `interrupt(), join(), ..`
- Jeder Thread kann auf alle public-Variablen aller Klassen zugreifen (die sollten Sie aber nicht verwenden) und die public-Methoden aller Klassen ausführen (falls nicht durch Synchronisationsmechanismen verhindert – die Verwendung von Synchronisationsmechanismen ist in dieser Aufgabe aber **nicht** notwendig!)

ACHTUNG: Die Methoden `stop`, `suspend` und `resume` dürfen **nicht** verwendet werden, da Sie als **deprecated** markiert sind.