

|          |                                       |               |
|----------|---------------------------------------|---------------|
| BAI3 BSP | Praktikum Betriebssysteme             | Hübner        |
| WS 2020  | Aufgabe 2 – Unix-Shell / Java-Threads | Seite 1 von 2 |

## 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     |
| 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/BS - Was willst du, Franz? quit
... und tschüß!
$
```

1.2. **Testen** Sie Ihr hawsh-Programm bzgl. aller Built-In-Befehle, der UNIX-Befehle `ps`, `ls` und `env` sowie eines nicht existierenden Befehls (z.B. `abcde`), und zwar jeweils mit Ausführung im Vordergrund und im Hintergrund (&)!

|          |                                       |               |
|----------|---------------------------------------|---------------|
| BAI3 BSP | Praktikum Betriebssysteme             | Hübner        |
| WS 2020  | Aufgabe 2 – Unix-Shell / Java-Threads | Seite 2 von 2 |

**Tipps:**

- Verwenden Sie den Beispielcode aus der Vorlesung (auf Folie) als Vorlage!
- Folgende Bibliotheksfunktionen sind zur Lösung hilfreich (Dokumentation über `man`-Befehl): `getenv`, `strcmp`, `strlen`, `fork`, `waitpid`, `exit`, `execlp`
- Der Aufruf von `execlp` sollte so aussehen: `execlp (command, command, NULL) ;`
- Der `man`-Dokumentation können Sie entnehmen, welche zusätzlichen C-Bibliotheksfunktionen explizit in Ihren Programmcode eingebunden werden müssen (`#include`).

## 2. Threads (JAVA-Programmierung)

Schreiben Sie ein JAVA-Programm **SimRace**, welches mit Hilfe von Threads ein Autorennen simuliert. Das Programm soll als Konstanten die Anzahl an Autos sowie die Länge der Strecke (in Runden) definieren. Direkt nach Erzeugung eines Auto-Threads (Klasse **Car**) kann dieser gestartet werden. Die Fortbewegung der simulierten Autos 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“ ( $0 \leq \text{Rundenzeit} < 100 \text{ ms}$ ) und der Auto-Thread solange angehalten („sleep“). Jedes Auto soll seine eigene Gesamtfahrzeit messen.

Sobald alle Autos im Ziel sind (nicht vorher!), soll auf der Konsole eine Gesamtergebnistabelle ausgegeben werden, in der die Autos in der Platzierungs-Reihenfolge mit der entsprechend benötigten Gesamtfahrzeit ausgegeben werden.

Beispiel für ein Rennen mit 5 Autos:

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

**Tipps:**

- Orientieren Sie sich am Beispielcode aus der Vorlesung (`sleep()`, `join()`, ..)!
- Jeder Thread kann auf alle `public`-Variablen aller Klassen zugreifen und die `public`-Methoden aller Klassen ausführen (falls nicht durch Synchronisationsmechanismen verhindert).
- Die veralteten Methoden `stop`, `suspend` und `resume` dürfen nicht verwendet werden, ebenso sollten Sie die Synchronisationsfunktionen `wait` und `notify` **nicht** einsetzen.

Viel Spaß!! ☺