UNIVERSITÄT OSNABRÜCK

marcel.koeppen [☺] uni-osnabrueck.de
marcel.luetkedreimann [☺] uni-osnabrueck.de

Übung zur Vorlesung
Betriebssysteme
Wintersemester 2021/22

Programmierblatt 1: Prozesse
May the fork be with you

➪ **Abgabe der Lösungen bis Mittwoch, 08. November 14:00 im AsSESS**

In dieser ersten Programmieraufgabe sollen Sie schrittweise eine eigene kleine Shell entwickeln, die Kommandos vom Benutzer entgegennimmt und die entsprechenden Programme oder Operationen ausführt.

Geben Sie Ihre Entwicklungsschritte jeweils in den angegebenen Dateien ab.

1 Eingabeaufforderung (4 Punkte)

Zunächst soll die Basisfunktionalität entwickelt werden. Schreiben Sie hierfür ein Programm, das den Usernamen, den Hostnamen und das Arbeitsverzeichnis mittels **printf(3)** wie folgt ausgibt:

```
username@hostname arbeitsverzeichnis $
```

Zusätzlich soll der Nutzer genau einen Befehl mit genau einem Argument eingeben können. Der eingegebene Befehl soll zunächst einfach ausgegeben werden. Dies soll in einer Endlosschleife realisiert werden.
Eine beispielhafte Ausführung Ihres Programms würde dann folgende Ausgabe liefern:

```
studi@bsvm /home/studi$ ls -la
Befehl: ls
Argument: -la
studi@bsvm /home/studi$
```

Um Zeichen von der Konsole einzulesen, lässt sich **scanf(3)** nutzen. Hierfür wird **scanf(3)** ein Formatstring übergeben. In diesem spezifizieren Sie, wie die Eingabe interpretiert werden soll. Als weitere Argumente erwartet die Funktion Pointer zu Variablen, in denen die Eingabe gespeichert werden soll.
Hinweise:

- Die Ausführung des Programms kann mit der Tastenkombination <**Strg**>+**C** abgebrochen werden.
- Nutzername, Rechnername und das aktuelle Verzeichnis lassen sich mit den folgenden Funktionen ermitteln:
  - **getpwuid(3)**
  - **gethostname(2)**
  - **getcwd(3)**
- Die Buffer für die einzulesenden Eingaben sollten ausreichend groß dimensioniert werden. Als sinnvolle Größe bieten sich hier z.B. 256 Byte an.
- In der System-Include-Datei `limits.h` finden Sie maximale Größe für den Hostnamen und den aktuellen Pfad. (`HOST_NAME_MAX` bzw. `PATH_MAX`)

- Um einen Bufferoverflow bei der Eingabe zu vermeiden, sollten Sie **scanf(3)** im Formatstring die maximale Länge Ihres Buffers übergeben. Bedenken Sie dabei aber, dass Strings in C nullterminiert sind. Der Formatstring ‘‘%10s‘‘ beschreibt einen maximal 10 Byte langen String.
- Es kann notwendig sein die Compileroption `-D_GNU_SOURCE` anzugeben, um **gethostname(2)** verwenden zu können.

→ `shell_a1.c`

2 Einen Befehl ausführen (2 Punkte)

Nun soll der eingelesene Befehl auch ausgeführt werden. Nutzen Sie hierfür **execlp(3)**. Übergeben Sie auch das eingelesene Argument, sodass der Befehl nun wie im „richtigen“ Terminal ausgeführt wird.

Hinweise:

- Wenn Sie Aufgabe 1 nicht bearbeitet haben, können Sie davon ausgehen, dass der Nutzer nur `ls -l` eingeben würde.

→ `shell_a2.c`

3 Mehrere Befehle ausführen (3 Punkte)

Aktuell wird die Shell beendet, sobald der aufgerufene Befehl beendet ist. Sorgen Sie mittels **fork(2)** dafür, dass nach der Ausführung eines Befehls wieder zur Eingabeaufforderung gesprungen wird. Achten Sie hierbei darauf, dass keine Zombie-Prozesse entstehen.

→ `shell_a3.c`

4 Eingebaute Befehle (2 Punkte)

Zusätzlich soll die Shell nun eingebaute Befehle bekommen, für die kein externes Programm gestartet wird. Implementieren Sie die Befehle **cd**, mit dem das Verzeichnis abhängig vom Argument mittels **chdir(2)** gewechselt werden soll, und **exit**, mit dem sich die Shell mit dem als Argument übergebenen Rückgabewert beenden soll.

Hinweise:

- **strncmp(3)** vergleicht die ersten *n* Bytes von zwei Strings.
- **strtol(3)** wandelt einen String in eine Zahl um.

→ `shell_a4.c`

5 Befehle im Hintergrund ausführen (3 Punkte)

Außerdem soll es nun möglich sein, Befehle im Hintergrund auszuführen. Sobald ein Befehl mit einem „!“ beginnt, soll die Shell nicht auf die Beendigung des Prozesses warten, sondern die PID des Hintergrundprozesses ausgeben und anschließend sofort neue Eingaben entgegennehmen. Auch hierbei sollen Sie darauf achten, dass keine Zombie-Prozesse entstehen.

→ `shell_a5.c`

Tipps zu den Programmieraufgaben:

- Kommentieren Sie Ihren Quellcode ausführlich, so dass wir auch bei Programmierfehlern im Zweifelsfall noch Punkte vergeben können!
- Denken Sie daran, dass viele Systemaufrufe fehlschlagen können! Fangen Sie diese Fehlerfälle ab (die Aufrufe melden dies über bestimmte Rückgabewerte, siehe die jeweiligen man-Pages), geben Sie geeignete Fehlermeldungen aus (z.B. unter Zuhilfenahme von **perror(3)**) und beenden Ihr Programm danach ordnungsgemäß.
- Die Programme sollen sich mit dem gcc auf aktuellen Linux-Rechnern wie denen im CIP-Pool oder der BSVM übersetzen lassen. Der Compiler ist mit den folgenden Parametern aufzurufen:
  `gcc -Wall -D_GNU_SOURCE -o shell shell.c`
  Weitere (nicht zwingend zu verwendende) nützliche Compilerflags sind: `-ansi -Wextra -Wpedantic -Werror -D_POSIX_SOURCE`
- Achten Sie darauf, dass sich der Programmcode ohne Warnungen übersetzen lässt, z.B. durch Nutzung von `-Werror`.