Aufgabenblatt #R2.1 zur Rechnerübung Betriebssysteme

Nachdem Sie in den Rechnerübungen zum Aufgabenblatt R1 Hilfsfunktionen für eine Shell implementiert haben sollen sie jetzt eine eigene Shell namens *myshell* implementieren. Hierbei soll der grundlegende Umgang mit dem UNIX-Prozessmodell das Sie in der Vorlesung kennen gelernt haben geübt werden. Für die Lösung der folgenden Aufgaben ist es Hilfreich wenn Sie sich die Manpages zu **fork(2)**, **waitpid(2)**, **exec(3)** und **exit(3)** anschauen.

Es wird davon ausgegangen, dass Sie das Aufgabenblatt R1 abgearbeitet haben und mit den Hilfsfunktionen in shellutils vertraut sind.

Die folgenden Aufgaben bauen aufeinander auf und sollten (wenn nicht anders angegeben) in der gegebenen Reihenfolge bearbeitet werden.

2.1 Eingabeaufforderung

Schreiben sie in der Datei `myshell.c` ein Programm names *myshell* das als Prompt das aktuelle Arbeitsverzeichnis ausgibt und auf eine Eingabeaufforderung wartet (s. **fgets(3)**).
Besteht die Eingabe nur aus einem `End-of-File (EOF)` (=strg+d) so soll in der nächsten Zeile eine nette Abschiedsmeldung ausgegeben und die *myshell* mittels **exit(3)** beendet werden. Prüfen sie hierfür ob **fgets(3)** als Rückgabewert `NULL` zurückgibt und prüfen sie anschließend mit **feof(3)** ob auf dem Standardeingabekanal (stdin) ein `EOF` liegt. In allen anderen Fällen tut die *myshell* zunächst nichts und gibt erneut einen Prompt gefolgt von einer Eingabeaufforderung aus.

2.2 Ausführen von Vordergrundprozessen

Erweitern sie nun die *myshell* um die Ausführung von Programmen. Besteht die Eingabe des Benutzers nicht aus `EOF` so geht die *myshell* davon aus, das die Eingabe den Start eines Programms bewirkt soll. Erzeugen Sie hierfür einen neuen Kind-Prozess (**fork(2)**) und warten sie dann auf die Terminierung dieses Kind-Prozesses (**waitpid(2)**).
Führen sie das Programm mittels `execvp()` (s. **exec(3)**) im neu erzeugten Kind-Prozess aus. Kann das Programm nicht ausgeführt werden soll eine sinnvolle Fehlermeldung mittels **perror(3)** ausgegeben und der Kindprozess beendet werden (**exit(3)**).
Geben sie nach Beendigung des Kindprozesses mit der Funktion `printStat()` den Exitstatus aus.

2.3 Ausführung von Hintergrundprozessen

Wenn die Ausführung von Vordergrundprozessen funktioniert erweitern sie die *myshell* um die Ausführung von Prozessen im Hintergrund.
Endet eine Eingabe mit dem Token „`&`“, d.h. in der Kommandostruktur ist *background==1*, so wird das ausgeführte Programm in einem Hintergrundprozess ausgeführt. In diesem Fall wartet die *myshell* nicht auf die Terminierung des Prozesses, sondern zeigt sofort einen neuen Prompt zur Entgegennahme weiterer Eingaben an.

2.4 Behandlung von Zombies

Da jetzt mehrere Prozesse parallel laufen und irgendwann unbemerkt Terminieren, ohne das ihr Exit-Status abgefangen wird, entstehen Zombies. Das soll natürlich nicht so bleiben, also ist es nun Ihre Aufgabe sich dem Zombieproblem zu stellen.

- Merken Sie sich hierfür beim Erzeugen eines Hintergrundprozesses dessen PID und die Kommandoeingabe in einer verketteten Liste. Hierfür ist eine fertig Implementierte Verkettete Liste vorgegeben (s. plist.h und plist.c).
- Prüfen Sie vor der Anzeige eines neuen Prompts welche Hintergrundprozesse bereits Terminiert sind.
- Fangen sie den Exit-Status dieser Prozesse ab (s. **waitpid(2)**).
- Entfernen sie den entsprechenden Prozess aus der Verketteten Liste.
- Geben sie analog zu den Vordergrundprozessen den Exit-Status aus.

Testen mit wait:
Damit Sie ihre *myshell* testen können lässt sich mit dem Befehl *make wait* ein Programm *wait* kompilieren welches bei Ausführung eine gegebene Menge an Sekunden wartet und danach Terminiert. Aufrufen können sie das Programm aus ihrer *myshell* heraus wie folgt: **./wait [Sekunden]**
[Sekunden] ist ein optionales Argument das die Menge an Sekunden angibt die *wait* warten soll. Ohne die Zeitangabe wartet *wait* eine Sekunde.

2.5 Verzeichniswechsel

Diese Aufgabe kann auch direkt nach Aufgabe 2.1 bearbeitet werden.

Bisher kann die *myshell* Prozesse ausführen und sich mit Zombies herumschlagen. Allerdings immer nur in dem gleichen, statischen Arbeitsverzeichnis aus dem die *myshell* gestartet wurde. Das soll sich jetzt ändern.
Implemtieren sie ein internes Kommando „`cd`“(cd für **c**hange **d**irectory).
Prüfen sie hierfür ob in der Kommando-Eingabe das erste Wort „`cd`“ ist (**strcmp(3)**) und prüfen Sie im erfolgsfall ob exakt ein Parameter übergeben wurde. Ist dieses der Fall führen sie einen Wechsel des Arbeitsverzeichnisses mittels **chdir(2)** durch.

Bei einem Verzeichniswechsel kann es passieren, dass dieses nicht durchgeführt werden kann weil z.B. das Verzeichnis nicht existiert. Sorgen sie bitte in diesem Fall für eine angemessene Fehlerausgabe mittels **perror(3)**.

2.6 *Anzeige Laufender Hintergrundprozesse*

Implementieren sie analog zum Kommando „`cd`“aus Aufgabe 2.5 ein weiteres internes Kommando namens „`jobs`“, welches beim Aufruf in einer Zeile PID und Kommandoeingabe aller aktuell laufen Hintergrund Prozesse auf der Standardausgabe ausgibt. Dazu ist es notwendig, das vorgegebene plist.c-Modul um die Funktion
`void walkList( int(*callback)(pid_t, const char *) )` zu erweitern, welche jedes Element der verketten Liste abläuft und eine Callback-Funktion aufruft. Die Callback-Funktion soll dabei die Argumente PID und Kommandoeingabe schön formatiert auf der Konsole ausgeben.

Erklärung zur parameterliste von `walklist()`: Die Funktion erwartet in den Parametern einen Pointer auf eine beliebige Funktion die zwei Parameter vom Typ `pid_t` und `const char*` hat und ein Integer-Wert zurückgibt.

Abgabe: bis 07.12.2012