Prof. Dr. M. Jäger
FB MNI

27. November 2014

Hausübung 1 - Betriebssysteme I

Bearbeitungszeit: bis 20.12.2014 24 Uhr

Präsentation der Lösung: in den Übungsstunden UND per Upload (siehe unten)

Programmieren Sie in C eine Shell, die verschiedene Kommandotypen interpretiert und Vordergrund- und Hintergrundausführung von Kommandos unterstützt. Die Shell soll interaktive Eingriffe in die Programme unterstützen:

- Abbruch mit STRG-C
- Anhalten mit STRG-Z
- Fortsetzen eines angehaltenen Programms mit *fg* oder *bg*

Prozessgruppen, Vordergrund- und Hintergrundausführung

Zu jeder interaktiven Sitzung gehört ein Kontrollterminal, das im Dateisystem unter dem Pfad */dev/tty* zu finden ist. Ein Programm läuft im „Vordergrund“, wenn ihm die Tastatur des Kontrollterminals zugeordnet ist.

Dazu müssen Prozessgruppen verwaltet werden: Für jeden Programmaufruf wird eine neue Prozessgruppe erzeugt (`setpgid`). Die Prozessgruppen-ID (PGID) ist identisch mit der PID des Prozesses. Damit die richtigen Prozesse abgebrochen bzw. unterbrochen werden, muss die Shell dem Systemkern eine Änderung der Vordergrundprozessgruppe mitteilen (`tcsetpgrp`). Nach Ende eines Vordergrund-Kommandos ist die Shell selbst wieder im Vordergrund. Eine Pipeline wird als ein Kommando, d.h. als eine Gruppe implementiert: Die PID des ersten Pipeline-Teilnehmers ist die PGID aller Pipeline-Teilnehmer. Mit STRG-Z wird also die gesamte Pipeline angehalten.

Kommandotypen und Syntax

1. Programmaufruf im Vordergrund, Syntax: *`Programmpfad Parameter ...`*

   Aufruf eines Programms als Subprozess der Shell. Die Shell wartet auf die Terminierung.

2. Programmaufruf im Hintergrund, Syntax: *`Programmpfad Parameter ...`* &

   Aufruf eines Programms als Subprozess der Shell. Die Shell wartet *nicht* auf die Terminierung. Die Shell schreibt die PID und PGID des neuen Prozesses auf die Standardfehlerausgabe.

3. Status der Subprozesse ausgeben, Syntax: status

   Die Shell zeigt für alle terminierten Subprozesse, deren Status noch nicht angezeigt wurde, folgendes an: PID, PGID, Terminierungsinformation und Programmname an. Danach werden diese Daten gelöscht. Für alle noch laufenden Hintergrundprozesse werden PID, PGID, Programmname und “running” angezeigt.

   Beispiel:

```
>> pwd
/home/jaeger
>> /opt/firefox/bin/firefox &
PID=1454 PGID=1454
>> xterm &
```

```
PID=1455 PGID=1455
>> kill -15 1455
>> ls -l /xyz
Datei nicht gefunden
>> status
PID      PGID   STATUS       PROG
1412     1412   exit(0)      pwd
1454     1454   running      /opt/firefox/bin/firefox
1455     1455   signal(15)   xterm
1461     1461   exit(0)      kill
1464     1464   exit(1)      ls
>> status
PID      STATUS       PROG
1454     1454   running      /opt/firefox/bin/firefox
>>
```

Erweiterungsmöglichkeit (optional): Führen Sie den Zustand *stopped* für angehaltene Prozesse ein.

4. Terminieren der Shell

   Syntax: `exit [` *Exit-Wert* `]`

   Wenn kein Exit-Wert angegeben ist, soll die Shell mit *exit(0)* terminieren.

5. Verzeichnis wechseln

   Syntax: `cd [` *Dateipfad* `]`

   Die Shell ändert ihr aktuelles Verzeichnis. Bei fehlendem Pfad wird das Login-Verzeichnis des Benutzers verwendet. (Systemaufruf *chdir*)

6. Fortsetzung eines angehaltenen Kommandos im Vordergrund

   Syntax: `fg` *PGID*

   Der Prozessgruppe wird dazu mit *kill* das Fortsetzungs-Signal SIGCONT geschickt.

7. Fortsetzung eines angehaltenen Kommandos im Hintergrund

   Syntax: `bg` *PGID*

8. Umlenkungen der Ein- und Ausgabe

   Umlenkungen der Eingabe und Ausgabe sind einzeln oder kombiniert möglich.

   - Ausgabeumlenkung, Syntax: *Programmaufruf* $>$ *Dateipfad*
     Die Standardausgabe des Programms wird in die angegebene Datei umgelenkt. Diese wird bei Bedarf erzeugt. Falls sie schon vorhanden ist, wird der alte Inhalt gelöscht.
   - Ausgabeumlenkung mit Anfügen, Syntax: *Programmaufruf* $>>$ *Dateipfad*
     Wie oben, aber falls die Datei schon vorhanden ist, wird der neue Inhalt hinter den alten geschrieben.
   - Eingabeumlenkung, Syntax: *Programmaufruf* $<$ *Dateipfad*
     Die Datei wird zur Standardeingabe des Programms.

9. Sequenz, Syntax: *Programmaufruf* ; *Programmaufruf* ; ... ; *Programmaufruf*

   Die Programme werden in Shell-Subprozessen nacheinander ausgeführt.

10. Ausführung bei Erfolg,

    Syntax: *Programmaufruf* `&&` *Programmaufruf* `&& ... &&` *Programmaufruf*

    Wie bei Sequenz, aber Abbruch, falls ein Programm nicht mit `exit(0)` terminiert.

11. Ausführung bei Misserfolg

    Syntax: *Programmaufruf* `||` *Programmaufruf* `|| ... ||` *Programmaufruf*

    Wie bei Sequenz, aber Abbruch, falls ein Programm mit `exit(0)` terminiert.

12. Pipeline

    Syntax: *Programmaufruf* `|` *Programmaufruf* `| ... |` *Programmaufruf*

    Alle Programme werden in Shell-Subprozessen nebenläufig ausgeführt. Die Standardausgabe eines Pipeline-Teilnehmers wird zur zur Standardeingabe des nächsten.

13. Bedingte Ausführung

    Syntax:

    `if` *Programmaufruf* `then` *Programmaufruf* `else` *Programmaufruf* `fi`
    `if` *Programmaufruf* `then` *Programmaufruf* `fi`

Signalbehandlung

Der Shell-Hauptprozess soll sich nicht mit STRG-C abbrechen lassen, sondern stattdessen einen Hinweis ausgeben, dass er mit dem *exit*-Kommando beendet wird.

Hinweise

1. Das Programm muss sich fehlerfrei auf `saturn.mni.thm.de` übersetzen und ausführen lassen.
2. Als Basis können Sie ein Shell-Skelett verwenden, in dem schon das komplette Front-End, d.h. Syntaxanalyse und Zerlegung der Kommandozeile, sowie einfache Kommandos und Sequenzen realisiert sind. Sie finden das Skelett auf dem Rechner `saturn.mni.thm.de` im Verzeichnis `~hg52/bs/shellsource`. Der Zugriff über das Internet ist beispielsweise per *rsync* oder *scp* möglich.
3. Bei allen Kommandos ist eine Fehlerbehandlung für fehlgeschlagene Systemaufrufe durchzuführen.
4. Für das status-Kommando muss eine Prozessliste geführt werden. Vor Anzeige der Prozessliste muss diese durch Abruf der aktuellen Prozesszustände aller Subprozesse aktualisiert werden.
5. Testen Sie die Shell sorgfältig. Testen Sie bei der Pipeline vor allem auch die Szenarien „Terminierung des Lesers bei blockiertem Schreiber“, z.B.

   ```
   $ od -x /bin/bash | head -1
   ```

   und „Terminierung des Schreibers bei blockiertem Leser“, z.B.

   ```
   $ echo hallo | cat
   ```

Abgabe:

Die Hausübung muss in Einzelarbeit oder in Zweiergruppen bearbeitet und abgegeben werden. Die Lösungen werden mit moss (http://theory.stanford.edu/ aiken/moss) auf kopierten Code geprüft. Abgegebene Lösungen mit gemeinsamen Code-Abschnitten werden nicht gewertet!

Die Hausübung muss wie folgt abgegeben werden:

- Sie erzeugen auf dem Rechner „saturn.mni.thm.de“ ein Verzeichnis, dessen Name mit der Matrikelnummer des (bzw. eines) Bearbeiters übereinstimmt. Dort gibt es ein Unterverzeichnis „prog“ und eine Datei „autor.txt“. In „autor.txt“ steht der Autor (bzw. die beiden Autoren) mit folgenden Angaben: Nachname, Vorname(n), Matrikelnummer.
- Im Verzeichnis „prog“ stehen die Quelltexte, ein auf Naiade mit „make“ getestetes Makefile zur Erzeugung der Shell und eine auf „saturn“ lauffähige und getestete Shell mit dem Programmnamen „shell“.
- Das Verzeichnis wird in eine mit komprimierte tar-Archivdatei (*Matrikelnummer*`.tgz`) gepackt, die nur für den Besitzer lesbar ist, und dann auf den Rechner „saturn“ in das Verzeichnis „/home/hg52/bs/ha1/moss’ kopiert.
- Beispiel:

  Hansi Hacker und Gundula Guru bearbeiten die Aufgabe als Zweiergruppe. Hansi hat die Matrikelnummer 123456, Gundula die Matrikelnummer 765432. Hansi bietet an, die Aufgabe unter seinem Benutzerkonto abzugeben.

  Er meldet sich auf Saturn an und legt die für die Abgabe vorgesehenen Verzeichnisse in seinem Home-Verzeichnis an:

  ```
  $ cd
  $ mkdir 123456
  $ mkdir 123456/prog
  ```

  Inhalt von „123456/autor.txt“:

  ```
  Guru, Gundula, 765432
  Hacker, Hansi, 123456
  ```

  Inhalt von „123456/prog“: Quelltextmodule: `Makefile, shell.c, shell.h, ...`
  Auf Saturn lauffähiges Programm: `shell`

  Archiv erzeugen, Zugriffsrechte setzen, abgeben:

  ```
  cd
  tar cvfz 123456.tgz 123456
  chmod 600 123456.tgz
  cp 123456.tgz ~hg52/bs/ha1/moss
  ```