Prof. Dr. M. Jäger
FB MNI

28. November 2011

Hausübung 1 - Betriebssysteme I

Bearbeitungszeit: bis 23.12.2011 24 Uhr

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

Programmieren Sie in C eine Shell, die folgende Kommandotypen korrekt interpretiert:

1. Einfacher Programmaufruf, Syntax: *Programmpfad Parameter ...*

   Aufruf als Subprozess der Shell. Die Shell wartet auf die Terminierung und merkt sich die Terminierungsinformation (exit-Wert bzw. Signalnummer) bis zur Ausführung des nächsten Befehls.

2. Hintergrundausführung, Syntax: *Programmaufruf* &

   Aufruf eines Programms in einem Subprozess der Shell. Die Shell wartet nicht auf die Terminierung. Die Shell schreibt die Subprozess-PID auf die Standardfehlerausgabe.

   Zombies werden durch *waitpid*-Aufrufe in einem Signalhandler für SIGCHLD vermieden. Die Terminierungsinformation wird so lange aufbewahrt, bis der Benutzer sie mit dem *status*-Kommando abgefragt hat.

3. Status der Subprozesse ausgeben, Syntax: status

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

   Beispiel:

```
>> pwd
/home/jaeger
>> /opt/firefox3.6/bin/firefox &
PID=1454
>> xterm &
PID=1455
>> kill -15 1455
>> ls -l /xyz
Datei nicht gefunden
>> status
PID      STATUS       PROG
1412     exit(0)      pwd
1454     running      /opt/firefox3.6/bin/firefox
1455     signal(15)   xterm
1461     exit(0)      kill
1464     exit(1)      ls
>> status
PID      STATUS       PROG
1454     running      /opt/firefox3.6/bin/firefox
>>
```

4. Verzeichnis wechseln

   Syntax: cd [*Dateipfad*]

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

5. 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.

6. Ausgabeumlenkung mit Anfügen, Syntax: *Programmaufruf* >> *Dateipfad*

   Wie oben, aber falls die Datei schon vorhanden ist, wird der neue Inhalt hinter den alten geschrieben.

7. Eingabeumlenkung, Syntax: *Programmaufruf* < *Dateipfad*

   Die Datei wird zur Standardeingabe des Programms.

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

   Die Programme werden in Shell-Subprozessen nacheinender ausgeführt.

9. Ausführung bei Erfolg,

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

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

10. Ausführung bei Misserfolg

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

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

11. Pipeline

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

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

12. Bedingte Ausführung

    Syntax:

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

Hinweise

1. Das Programm muss sich fehlerfrei auf `naiade.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 `naiade.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. Durch Signale (SIGCHLD) kommt es während der Ausführung der Shell immer wieder zum Aufruf von Signal-Handlern. Sowohl die reguläre Verarbeitung als auch der Signalhandler für SIGCHLD verändern die Prozessliste, so dass diese wegen der Race-Gefahr durch geeignete Synchronisationsmaßnahmen abzusichern ist. Definieren Sie dazu einen *mutex* mit folgenden Operationen:

   - Erzeugen eines neuen Mutex: `mutex mutex_init(void)`
     liefert den neuen Mutex zurück
   - Sperren mit Blockade: `int mutex_lock(mutex m)`
     liefert 0 bei Erfolg, -1 bei Fehler
   - Sperre freigeben / Aufwecken: `int mutex_unlock(mutex m)`
     liefert 0 bei Erfolg, -1 bei Fehler
   - Mutex entfernen: `int mutex_destroy(mutex m)`
     liefert 0 bei Erfolg, -1 bei Fehler

   Es bleibt Ihnen überlassen, wie Sie den Mutex implementieren. Einige Möglichkeiten: *pthread_mutex*, *pipe/read/write*, Dateisperren (*fcntl*), Semaphore (*semget*). Testen Sie sorgfältig, ob der gegenseitige Ausschluss funktioniert!

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 „naiade.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 „gmake“ getestetes Makefile zur Erzeugung der Shell und eine auf „naiade“ 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 „naiade“ 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 Naiade 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 Naiade 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
```