Universität Konstanz
Informatik & Informationswissenschaft
Prof. Dr. Marcel Waldvogel / Klaus Herberth

Programmierkurs 3
WS 2016/2017

11. Übungsblatt

(Ausgabe: 19. Januar 2017 — Abgabe bis: 26. Januar 2017, 4:00)

Aufgabe 1: Rudimentäre Shell, Teil 2 (60 Punkte)

Erweitern Sie die Shell um die Fähigkeit andere Programme zu starten, und deren Ein-/Ausgabe umzuleiten.

Als Gerüst für die Shell können Sie die Musterlösung zum letzten Übungsblatt verwenden.

Den Parser für die Benutzereingaben zu schreiben ist arg komplex und wenig interessant. Verwenden Sie einfach den vorbereiteten Parser, dazu müssen Sie den Header `parser.h` einbinden und die Datei `parser.c` mit Ihrem Programm zusammen kompilieren. Die verwendeten Datenstrukturen sind in `parser.h` dokumentiert, und so entworfen dass sich die Aufgabe möglichst einfach lösen lässt. Die Verwendung des Parsers wird in `testparser.c` demonstriert. Das vorbereitete `Makefile` erzeugt ein kleines Programm `testparser` mit dem Sie den Parser testen können, Beispieleingaben sind in `testparser.txt`.

Während `testparser` die gelesenen Befehle nur ausgibt, müssen Sie sich noch darum kümmern die Programme tatsächlich zu starten, und die Filedescriptoren anzupassen. Am einfachsten ist es in der gegebenen Reihenfolge vorzugehen:

1. Passen Sie die `rsh` so an, dass sie den vorgeschlagenen Parser verwendet, und damit die bisherige Aufgabe löst (eingebaute Kommandos, z.B., `cd` oder `help`, . . .). (5)

2. *Ohne Pipelines und Redirection:* Externe Programme (also Kommandos die nicht in die `rsh` eingebaut sind) sollen ausgeführt werden, *cf.* `exec`(3). Die Shell wartet bis das aufgerufene Programm terminiert, und gibt eine Meldung darüber aus. Erst dann wird die nächste Zeile gelesen. (15)

```
$ sleep 5
Forked 24177: sleep
Child 24177 returned 0.             # das dauert etwas
$ echo hello world how are you
Forked 24191: echo
hello world how are you             # Die Ausgabe von echo
Child 24191 returned 0.
$ false
Forked 24201: false
Child 24201 returned 1.             # Rückgabewert 1
```

3. *Pipelines einbauen:* Wenn mehrere Kommandos durch `|` getrennt eingegeben werden, dann wird die Standardausgabe des vorderen Programmes mit der Standardeingabe des Nachfolgenden verbunden. Entsprechend müssen zwischen den Prozessen mit `pipe`(2) Pipelines angelegt, und die Filedescriptoren entsprechend auf 0 oder 1 gesetzt werden. (15)

```
$ ls -l | wc
Forked 24165: wc
Forked 24166: ls
Child 24166 returned 0.
14     119     669             # Die Ausgabe von wc
Child 24165 returned 0.
$ date | sed s/1/x/g | rev
Forked 24154: date
Forked 24153: sed
```

```
Forked 24152: rev
Child 24154 returned 0.
6x02 TEC 34:52:6x 52 naJ noM        # Die Ausgabe von rev
Child 24153 returned 0.
Child 24152 returned 0.
```

4. *Redirections einbauen:* Jedes der durch `|` getrennten Kommandos kann seine eigenen Umleitungen einrichten — das geschieht *nachdem* die Pipeline konstruiert wurde. Der Parser versteht die folgenden Konstruktionen, die Sie implementieren sollen. Die Syntax ist and die `bash`(1) angelehnt: (25)

   - `n<file` — Öffne Datei file zum Lesen. Mit `dup2`(3) muss sichergestellt werden, dass die geöffnete Datei den Filedescriptor n bekommt. Fehlt n, so wird 0 angenommen.
   - `n>file` — Wie `<`, die Datei wird aber zum Schreiben geöffnet, erstellt falls sie nicht existiert, und trunkiert sonst, *cf.* `open`(2). Fehlt n, so wird 1 angenommen.
   - `n>&m`, oder `n<&m` — Der (neue) Filedescriptor `n` ist ein Duplikat von `m`.

   Wichtig ist die Reihenfolge der Umleitungen (*cf.* `bash`(1)), der Parser liefert sie in der richtigen Reihenfolge.

```
$ ls -l Makefile /nope >stdout 2>stderr
Forked 29765: ls
Child 29765 returned 2.
$ cat stdout
Forked 29800: cat
-rw------- 1 sk users 526 Jan 25 19:02 Makefile
Child 29800 returned 0.
$ cat stderr
Forked 29803: cat
ls: cannot access /nope: No such file or directory
Child 29803 returned 0.
$ ls -l Makefile /nope 3>&1 1>&2 2>&3 | rev
Forked 29735: rev
Forked 29736: ls
yrotcerid ro elif hcus oN :epon/ ssecca tonnac :sl
-rw------- 1 sk users 526 Jan 25 19:02 Makefile
Child 29736 returned 2.
Child 29735 returned 0.
```

5. Wenn Sie die eingebauten Kommandos (`help`, *etc.*) von einem abgespaltenen Kindprozess der `rsh` erledigen lassen, der sofort danach terminiert, dann können sogar diese Kommandos über Pipelines Daten an externe Kommandos schicken. (5 Bonuspunkte)

```
$ help | tr a-z A-Z | sed /^$/d
Forked 29992: sed
Forked 29993: tr
Forked 29994: help
Child 29994 returned 0.
Child 29993 returned 0.
BUILTIN COMMANDS:
HELP
DISPLAY INFORMATION ABOUT BUILTIN COMMANDS.
ECHO
WRITE ARGUMENTS TO THE STANDARD OUTPUT.
PWD
PRINT THE NAME OF THE CURRENT WORKING DIRECTORY.
CD
CHANGE THE SHELL WORKING DIRECTORY.
Child 29992 returned 0.
```

Im Idealfall können Sie alle Kommandos ausführen, die vom Parser akzeptiert werden. Sinnvolle Eingaben sollten sich verhalten wie bei der Eingabe an der `bash`.

Hinweis: Achten Sie darauf, alle nicht mehr benötigten Filedescriptoren so bald wie möglich zu schließen, ein Prozess wartet sonst evtl. auf Eingabe die nie erzeugt wird.