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

Programmierkurs 3
WS 2016/2017

10. Übungsblatt

(Ausgabe: 12. Januar 2017 — Abgabe bis: 19. Januar 2017, 4:00)

Aufgabe 1: ??? (10 Punkte)

Betrachten Sie das Shellscript `splitfile`, und sein Verhalten.

```
$ cat splitfile
#!/bin/bash
head >/dev/null;
head;
$ seq 100000 | ./splitfile

1861
1862
1863
1864
1865
1866
1867
1868
1869
```

Rechnen Sie *genau* nach weshalb gerade diese Ausgabe erscheint. Eine der für die Rechnung benötigten Zahlen finden Sie mit `strace`(1). Wie? (Falls Sie andere Zahlen beobachten, rechnen Sie mit Ihren).

Aufgabe 2: Rudimentäre Shell (50 Punkte)

Über die nächsten Übungsblätter werden wir eine rudimentäre Shell implementieren. Die Shell liest die Eingabe ein und teilt sie in Token auf. Die Token sind durch Leerzeichen getrennt, Anführungszeichen und Backslash sollen nicht besondes beachtet werden (kein Quoting). Nützliche Funktionen: `getline`(3), `strtok`(3).

- Wenn das erste Token ein internes Kommando benennt, wird dieses mit den restlichen Token als Argumente aufgerufen.
- Sonst wird eine Fehlermeldung ausgegeben, die Shell aber nicht beendet. Das Starten externer Prozesse wird Aufgabe einer späteren Übung.
- Implementieren Sie die Shell-internen Kommandos `help` und `echo`, die wie im Beispiel gezeigt funktionieren sollen.
- Jedem Unix-Prozess ist ein *"current working directory"* zugeordnet, das Verzeichnis auf welches sich relative Pfadangaben beziehen. Die `rsh` soll dieses Verzeichnis mit den Kommandos `pwd` und `cd` ausgeben und wechseln können. Ohne Argumente führt `cd` ins `$HOME`-Verzeichnis der Benutzers. Nützliche Funktionen: `getcwd`(2), `chdir`(2).

Beispiel:

```
kh@tauhou:~/prg/experiments/rsh$ ./rsh
$ help
Builtin commands:
help
Display information about all builtin commands.
echo
Write arguments to the standard output.
pwd
Print the name of the current working directory.
cd
Change the shell working directory.
$ echo   foo  bar
foo bar
$ pwd
/home/sk/prg/experiments/rsh
$ cd
$ pwd
/home/sk
$ cd /tmp
$ pwd
/tmp
$ ls -l
rsh: ls: unknown
$ ^D[exit by end of file]
```

Hinweise:

- Es muss ein `Makefile` mitgeliefert werden.
- Achten Sie auf saubere und flexible Speicherverwaltung. Es ist aber nicht nötig Speicher explizit freizugeben, kurz bevor die Shell terminiert.
- Versuchen Sie möglichst sinnvoll Fehler abzufangen. Die Shell sollte nicht wegen jeder Kleinigkeit terminieren, sondern hilfreiche Fehlermeldungen ausgeben (*z.B.* `err`(3), `warn`(3), ...).
- Achten Sie darauf, den Aufbau von `rsh` leicht erweiterbar zu halten. Vorschläge:
  - Die Struktur `builtin_func` stellt ein internes Kommando dar.
  - Alle internen Kommandos sind in einem Array `builtin_funcs` gelistet.
  - Man könnte alle internen Kommandos in einer separaten Datei definieren.

  Ein entsprechender Vorschlag für `builtins.h`:

```
struct builtin_func {
        /* Name des Kommandos */
        const char *name;
        /* Kurze Beschreibung */
        const char *help;
        /* Funktion die aufgerufen werden soll.  Argumente analog zu  ‘main‘: ‘argc‘ ist die Zahl der Token.
         * ‘argv‘ enthält alle Token, ‘argv[argc]‘ ist immer ‘NULL‘.
         */
        void (*func)(int argc, char **argv);
};
extern struct builtin_func builtin_funcs[];
```