Dr. M. Grabert
Johannes Mayer

Systemnahe Software (SS 2001)

Abgabetermin: 5. Juni 2001

Beispiellösung

8 Let it grow – Midi-Shell wird erwachsen

Sie haben mal so zum Spass die Funktionalität der Bash-Shell mit der Funktionalität der *Midi-Shell* aus der Vorlesung verglichen und sind ganz enttäuscht. „So winzig und nennt sich schon Shell“, denken Sie. Da gibt's nur Eines. Sie kennen ja jetzt einige Funktionalitäten der Bash, von denen Sie ganz begeistert sind. Diese wollen Sie unbedingt in die Midi-Shell aus der Vorlesung einbauen. Gesagt getan! Sie setzen sich an den Rechner und erweitern die Midi-Shell um die folgenden Fähigkeiten.

8.1 Zombie-Vermeidung (2 Punkte)

Wenn sie in der Midi-Shell bisher ein Hintergrundkommando starten, geistert nach der Termination des zugehörigen Prozessen ein Zombie herum. Sie beschließen, dass es durch Ihre Erweiterung erst gar nicht zu Zombies kommen wird.

8.2 Tilde-Expansion (1 Punkt)

Damit sie nicht immer wieder Ihren Heimatkatalog eintippen müssen, ersetzen sie alle Token, die nur aus dem Zeichen „`~`“ bestehen, direkt nach dem „Zerlegen“ durch den Heimatkatalog, wie er in der Umgebungsvariablen `HOME` zu finden ist.

Beispiel:

```
turing$ echo ~
/home/turing/jmayer
turing$
```

8.3 Variablen-Substitution (2 Punkte)

Ebenso wollen Sie, dass jedes Token der Form „`$Variablenname`“ durch den Wert der jeweiligen Umgebungsvariablen ersetzt wird, sofern es eine Umgebungsvariable mit diesem Namen gibt. Andernfalls wird ein solches Token durch den leeren String ersetzt.

Beispiel:

```
turing$ echo $HOME $TTY $x
/home/turing/jmayer /dev/pts/85
turing$
```

8.4 Dateinamen-Expansion (2 Punkte)

Alles klar so weit? Dann wird's jetzt spannend! Das Token „`*`“ wird von der Bash nämlich durch mehrere Token ersetzt, falls im aktuellen Katalog eine Datei (auch Verzeichnis!) existiert, die nicht mit einem Punkt („.“) beginnt; nämlich für jede solche Datei ein eigenes Token. Gibt es keine solche Datei, dann bleibt das Token „`*`“ unverändert.

Beispiel:

```
turing$ ls
midish.c midish test.txt
turing$ echo *
test.txt midish.c midish
turing$
```

8.5 Built-In-Kommandos (3 Punkte)

Sie haben bei der Midi-Shell auch mal `cd` gefolgt von `pwd` eingegeben. Tja, leider ist der aktuelle Katalog danach nicht Ihr Heimatkatalog. Da kommt es Ihnen! „Klar, für manche Kommandos darf nicht geforkt werden!“ Erklären Sie das mal Ihrem Tutor anhand eines solchen Kommandos. Nun implementieren Sie die Kommandos `set` und `cd` als Funktionsaufruf (ohne zu forken!) in der Midi-Shell. So etwas nennt sich dann „Built-In-Kommando“. `set` soll einfach die Umgebungsvariablen auf die Standardausgabe schreiben. `cd` soll bei einem Argument in das angegebene Verzeichnis wechseln oder in den Heimatkatalog, falls kein Argument angegeben ist. Bei Built-In-Kommandos soll in unserer Midi-Shell keine Hintergrundverarbeitung und keine Ein-/Ausgabeumlenkung stattfinden. (Wir machen uns das Leben leicht. :-)))

Beispiel:

```
turing$ set
http_proxy=http://www-proxy.uni-ulm.de:3128/
TMPDIR=/tmp
...
turing$
turing$ pwd
/home/turing/jmayer/html/feedback
turing$ cd
turing$ pwd
/home/turing/jmayer
turing$
```

Tipps: Auf die Umgebungsvariablen dürfen Sie jetzt natürlich mit Hilfe von `getenv()` zugreifen. Will man alle Dateien eines Katalogs aufspüren, sind die Funktionen `opendir()`, `readdir()` und `closedir()` nützlich. Und das aktuelle Verzeichnis wechselt man am besten mit `chdir()`.

Lösung:

```
/*
 * Midishell via System
 * basierend auf Loesung von Blatt 3 AI4 SS01
 * Vater nimmt Kommando entgegen und Sohn fuehrt dieses via execvp() aus
 * Signale SIGINT und SIGQUIT werden ggf. abgefangen und an den Sohn gesandt
 * Eingabeumlenkung "<" und ">" und Hintergrundkommandos "&" werden unterstuetzt
 * (c) mg sai 2001
 * Erweiterung zur Loesung von Blatt 5 AI4 SS01
 * folgende zusaetzlichen "Features" stehen zur Verfuegung
 *  - Zombie-Prozesse entstehen nicht
 *  - Token "~" wird durch Homeverzeichnis ersetzt
 *  - Token "*" wird durch die Namen aller Dateien im aktuellen
 *    Katalog ersetzt, wobei jeder Dateiname ein EIGENES (!) Token bildet
 *  - Token "$Variablenname" wird durch den Wert der Umgebungsvariablen
 *    ersetzt, falls es eine Umgebungsvariable mit diesem Namen gibt
 *  - Built-Ins "cd" und "set" sind implementiert
 * (c) jm sai 2001
 */
#include <stdio.h>
#include <strings.h>
#include <signal.h>
#include <wait.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <dirent.h>

static int pid;         // pid des Sohnes
                        // das ist uebrigens auch ein Kommentar :-))

/*
 * wird bei Eintreffen von SIGINT oder SIGQUIT vom Vater hier abgefangen
 * und an den Sohn gesandt
 */

void sendtoson(int sig)
{
   if (!pid)            // kein Sohn aktiv ...
      return;
```

```
   printf("midish gets signal %d ... forwarding to %d\n", sig, pid);
   kill(pid, sig);      // forward signal to son
}

/*
 * Built-In-Kommandos wie cd, set, ... werden von dieser
 * Funktion interpretiert.
 * Rueckgabewert: != 0, falls das Kommando ein Built-In ist
 *                      und somit schon bearbeitet wurde
 *                == 0, falls das Kommando kein Built-In ist
 *                      und noch ausgefuehrt werden muss
 */

int builtin(char *args[]) {
   if (args[0] != NULL) {
      if (!strcmp(args[0], "set")) {         // built-in set
         if (args[1] != NULL)
            fprintf(stderr, "usage: set\n");
         else {
            extern char **environ;
            int i;
            for (i = 0; environ[i] != NULL; i++)
               puts(environ[i]);
         }
         return 1;
      }
      else if (!strcmp(args[0], "cd")) {     // built-in cd
         if (args[1] != NULL && args[2] != NULL)
            fprintf(stderr, "usage: cd [<directory>]\n");
         else if (args[1] == NULL) {
            if (!getenv("HOME"))
               fprintf(stderr, "cd: HOME isn't defined\n");
            else if (chdir(getenv("HOME")) < 0)
               perror("cd");
         }
         else {
            if (chdir(args[1]) < 0)
               perror("cd");
         }
         return 1;
      }
   }
   return 0;            // kein built-in
}

int main(int argc, char **argv)
```

```
{  char buf[200];     // Kommandozeile
   char *tok;         // braucht strtok ... argv
   char *args[200];   // Argumentvektor fuer Kommando
   char *bg;          // gesetzt, falls "&" in Kommandozeile
   char infile[100];  // hier steht ggf. die Ausgabedatei
   char outfile[100]; // hier steht ggf. die Eingabedatei
   char *out;         // gesetzt, falls > kam
   char *in;          // gesetzt, falls < kam
   int i = 0;
   int stat;          // fuer waitpid
   struct sigaction new;

   new.sa_handler = sendtoson;
   new.sa_flags = SA_RESTART;        // Einlesen nicht unterbrechen
   sigaction(SIGINT, &new, NULL);    // SIGINT und SIGQUIT abfangen
   sigaction(SIGQUIT, &new, NULL);

   new.sa_handler = NULL;            // kein Handler, aber ...
   new.sa_flags = SA_RESTART
                | SA_NOCLDWAIT;      // ... keine Zombies erzeugen
   sigaction(SIGCHLD, &new, NULL);

   printf("midish > ");
   while (fgets(buf, sizeof(buf), stdin)) {
      buf[strlen(buf)-1] = 0;           // Newline am Ende entfernen
      if (!strcmp(buf, "exit"))         // "exit" beendet Shell
         break;
      if (*buf==0) {                    // leere Kommandozeile
         printf("midish > ");
         continue;
      }
      if ((bg = strchr(buf, '&')))      // Hintergrundstart
         *bg = ' ';                     // & loeschen

      in = strchr(buf, '<');            // Eingabeumlenkung ???
      if ((out = strchr(buf, '>'))) {   // Ausgabeumlenkung ???
         *out = 0;                      // > loeschen
         sscanf(out+1, "%s", outfile); // Dateinamen lesen
      }
      if (in) {
         *in = 0;                       // < loeschen
         sscanf(in+1, "%s", infile);    // Dateinamen lesen
      }
      tok = strtok(buf, " ");           // Kommandozeile bauen
      i = 0;
      while (tok) {
```

```
   if (*tok == '$')                // Variablen-Substitution
      args[i++] = getenv(tok+1) ? strdup(getenv(tok+1)) : "";
                                   // Tilde-Substitution
   else if (!strcmp(tok, "~") && getenv("HOME"))
      args[i++] = strdup(getenv("HOME"));
   else if (!strcmp(tok, "*")) {  // Dateinamen-Expansion
      DIR *dp;
      struct dirent *entry;
      int ialt = i;               // Wert von i sichern
      // alle Dateien im aktuellen Katalog
      // in args aufnehmen ...
      if (!(dp = opendir(".")))
         continue;
      while ((entry = readdir(dp)))
         if (*(entry->d_name) != '.')
            args[i++] = strdup(entry->d_name);
      closedir(dp);
      if (i == ialt)              // gab es keine Datei?
          args[i++] = strdup(tok);// => "*" bleibt!
   }
   else
      args[i++] = strdup(tok);    // "normales" Argument

   tok = strtok(NULL, " ");
}
args[i] = NULL;

if (!builtin(args))   // nur forken falls kein Built-In ...
switch (pid = fork()) {
   case -1:   /* error */
      break;
   case 0:    /* Sohn */
      // Signale zuruecksetzen
      new.sa_handler = SIG_DFL;
      sigaction(SIGINT, &new, NULL);
      sigaction(SIGQUIT, &new, NULL);
      if (out) {   // Ausgabeumlenkung ??
         close(1); // Stdout schliessen
            // open nimmt 1 als Filedes-
            // kriptor (kleinster freier)
         if (open(outfile, O_CREAT|O_TRUNC|O_WRONLY, 0644)==-1)
            perror(outfile),
            exit(1);
      }
      if (in) {
```

```
            close(0); // Stdin schliessen
              // open nimmt 0 als Filedes-
              // kriptor (kleinster freier)
            if (open(infile, O_RDONLY)==-1)
              perror(infile),
              exit(1);
          }
          execvp(args[0], args); // Kommando starten !
          printf("midish: cmd >%s< not found\n", args[0]);
          exit(0);
        default:
          if (!bg)   // nicht im Hintergrund: warten
            waitpid(pid, &stat, 0);
      }
      printf("midish > ");
      pid = 0;         // braucht Fktn. sendtoson
    }
    return 0;
}
```