Betriebssysteme - Blatt 5

|  |  |
| --- | --- |
| Name: | Ümit Karadeniz |
| Gruppe: | 5 (ehemals Gruppe 7) |
| Datum: | 14. Mai 2014 |
| Funktion: | Aufgabe 5.1 Erläutern des Kommandos „cpio –ivt“  Aufgabe 5.2 Programmierung einer eigenen Shell |
| Funktionen: | write() - <http://linux.die.net/man/2/write>  read() - <http://linux.die.net/man/2/read>  waitpid() - <http://linux.die.net/man/2/waitpid>  dup2() - <http://linux.die.net/man/2/dup2>  execlp() - <http://linux.die.net/man/3/execlp>  pipe() - <http://linux.die.net/man/2/pipe>  scanf() - <http://linux.die.net/man/3/scanf> |
| Benutzung: | Aufgabe 5.2  Normale Kompilierung über gcc mit *„gcc mysh2.c –o mysh2“.* Nach dem Ausführen des Programms erscheint eine Shell, die einfache Befehle entgegen nimmt. |

Aufgabe 5-1 : cpio -ivt

Cpio ist ein Archivierungsprogramm unter Linux und dient zur Komprimierung von Daten. Mit den übergebenen Parametern „-ivt“ liest cpio eine Archivedatei nach der angebenen Ausgabe und listet diese einzeln den Dateinamen und Abhängigkeiten.

Aufgabe 5-2 : mysh2.c

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <wait.h>

#define MAX 1024

#define READ 0

#define WRITE 1

#define NUM\_OF\_PIPES 2

#define PARENT 1

#define CHILD 0

// Erstelle zwei Pipes zur bidirektionalen Kommunikation zwischen

// Eltern- und Kindprozess.

int pipes [NUM\_OF\_PIPES][NUM\_OF\_PIPES];

// Deklariere die Pipes. 0 definiert Lesen und 1 das Schreiben.

#define PARENT\_READ (pipes[PARENT][READ])

#define PARENT\_WRITE (pipes[CHILD][WRITE])

#define CHILD\_READ (pipes[CHILD][READ])

#define CHILD\_WRITE (pipes[PARENT][WRITE]) // wait

// Funktion, ohne Rückgabewert, zur Ausführung eines Befehls,

// das mittels Aufruf übergeben wurde.

void parser(char \*cmd){

pid\_t pid;

int status;

// Pipe für Eltern und Kind

pipe(pipes[PARENT]);

pipe(pipes[CHILD]);

if( (pid = fork()) == 0){

// Dupliziere alten File Descriptor auf neuen FD.

dup2(CHILD\_READ, STDIN\_FILENO);

dup2(CHILD\_WRITE, STDOUT\_FILENO);

// Nicht benutzte Pipes werden geschlossen.

close(CHILD\_WRITE);

close(PARENT\_READ);

close(PARENT\_WRITE);

// Auslesen des Befehls aus dem Pipe und ausführen mit execlp

char buffer[MAX];

int out = read(CHILD\_READ, buffer, sizeof(buffer)-1);

execlp(buffer,buffer,NULL);

} else if(pid > 0) {

// Nicht benutzte Pipes werden geschlossen.

close(CHILD\_READ);

close(CHILD\_WRITE);

// Elternprozess schreibt Befehl für Kind in die Pipe.

write(PARENT\_WRITE, cmd, 20);

// Wartet auf die Terminierung vom Kindprozess.

waitpid(pid, &status, 0);

// Lese Ausgabe aus der Pipe, das vom Kind geschrieben wurde.

char readBuffer[MAX];

int out = read(PARENT\_READ, readBuffer, sizeof(readBuffer)-1);

if (out >= 0) {

readBuffer[out] = 0;

printf("%s", readBuffer);

} else {

printf("IO Error\n");

}

}

}

int main(){

// Endlos-Schleife mit Befehlabfrage

while(1){

char command[16];

printf("%s@%s$ ", getenv("USER"),getenv("TERM"));

scanf("%s", command);

parser(command);

}

return 0;

}