Technische Informatik 2 | WS 2016/17 | C05
Tutor: Marc/Bingbin

Tabea Eggers
Jan Fiedler
Florian Pflüger
Jonas Schmutte

Übungsblatt 4

1	2	3	4	5	$\sum$

Lösungsvorschlag
Abgabe: 28.11.2016

Aufgabe 1

Erstmal haben wir den Signalhandler geschrieben, da wir auf dem letzten Blatt bereits einen geschrieben haben, bot sich dies an.
Bevor wir uns an die Ausführung der Prozesse machten, haben wir die Methode “findPathToCommand“ geschrieben, damit wir die richtigen Pfade bekommen.
Als dies richtig funktionierte, haben wir den Ausführungsteil geschrieben.
Danach haben wir diesen getestet und als funktionierend befunden.
Allerdings benutzen wir weder “wait()“ noch “waitpid()“, da wir mit dem Befehl ”pause()”, die richtige Funktionalität erzielten und mit den anderen beiden genannten, mehrere Bugs vorhanden waren.

```
1  #include <stddef.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string>
5  #include <sys/wait.h>
6  #include <unistd.h>
7  #include <iostream>
8  #include <dirent.h>
9  #include "parser.h"
10 #include <string.h>
11 using namespace std;
12
13
14 /*
15  * Die Methode die bestimmt wie mit den Signalen umgegangen wird.
16  * Sie sollen nur gesammelt werden, es braucht nichts anderes getan
       werden.
17  * @param int - Signalnummer, wird hier nicht benoetigt
18  */
19 static void handel(int){}
20
21
22 /*
23  * Gibt den Pfad zu dem uebergebenen Kommando zurueck.
24  * @param command - Kommando zu dem der Pfad ermittelt werden soll
25  * @return std::string - Pfad zum Kommando, falls das Kommando nicht
       existiert
```

```
26  *  wird "" zurueckgegeben.
27  */
28 static string findPathToCommand(char* command){
29     // Beginnt das mit Kommando mit '.' oder '/' ist es eine
       Pfadangabe
30     if(command[0] == '.' || command [0] == '/')
31         return command;
32     else{
33         //Holt den Inhalt der environment variable PATH
34         char* path = getenv("PATH");
35         //Position in path
36         int ppos = 0;
37         while (path[ppos]){
38
39             //Ist das aktuelle Zeichen in path kein '/' ist
40             //dieser Eintrag ungueltig und die Schleife kann enden.
41             if(path[ppos] != '/') break;
42
43             //String indem der Pfad zum Kommando aufgebaut wird
44             string temp = "";
45             //position in temp
46             int tpos = 0;
47
48
49             //Solange path nicht am Ende ist und das aktuelle Zeichen
                   nicht das
50             // Trennzeichen ':' ist, koennen wir temp weiter aufbauen
                  .
51             while(path[ppos] && path[ppos] != ':'){
52                 temp += path [ppos];
53                 ppos++;
54                 tpos++;
55             }
56             //Es muss noch hinter das ':' gesprungen werden
57             ppos +=1;
58
59             //in temp steht jetzt der Path zu einem Directory in der
                  PATH
60             DIR* dirStream = opendir(temp.c_str());
61
62             struct dirent* dir;
63             //Dieses gilt es jetzt nach unserem kommando zu
                  durchsuchen
64             while (((dir = readdir(dirStream))!= NULL)){
65                 char* dname = dir->d_name;
66                 //Gibt es dieses kann '/' und das Kommando an temp
                      gehangen
67                 //werden und wir erhalten unseren Pfad zum Kommando
68                 if (strcmp(dname, command)== 0){
69                     temp += '/';
```

```
70                      temp += command;
71                      closedir(dirStream);
72                      return temp;
73                  }
74              }
75
76          }
77          //Das Kommando existiert nicht
78          return "";
79      }
80  }
81
82
83  /*
84   * Setzt den Signalhandler
85   */
86  static void setUpSigHandler() {
87      // Legt unsere sigaction an
88      struct sigaction sigact;
89      // Setzt die Methode zum umgehen mit Signalen auf handel()
90      sigact.sa_handler = &handel;
91      // Setzt den SA_Restart flag
92      sigact.sa_flags = SA_RESTART;
93      // Die sigaction sigact soll benutzt werden um SIGCHLD Signale zu
            handeln
94      if (sigaction(SIGCHLD, &sigact, NULL) == -1) perror("sigaction:
            ");
95  }
96
97
98  int main(){
99      setUpSigHandler();
100     for (;;){
101         struct command cmd = read_command_line();
102         cout << "command: " << cmd.argv[0]
103              << ", background: " << (cmd.background ? "ja" : "nein")
                    << endl;
104         //Der Pfad zu unserem Kommando
105         string path = findPathToCommand(cmd.argv[0]);
106         // Ab jetzt brauchen wir Kind- und Vaterprozess
107         pid_t pid;
108         if ((pid = fork()) == -1){
109             perror("fork :");
110         }
111         if(pid > 0){
112             //Der Vaterprozess muss auf die Terminierung seines
                    Kindes warten,
113             //falls dieses nicht im Hintergrund ausgefuehrt werden
                    soll.
114             if(cmd.background==0){pause();}
```

```
115
116             } else if (pid == 0){
117                 //Der Kindprozess muss nur noch das Kommando ausfuehren
118                 if ((execv(path.c_str(), cmd.argv)) == -1){
119                     perror("execv: ");
120                 }
121             } else {
122                 cerr << "Fork failed!" << endl;
123             }
124         }
125 }
```

Tests

Führt die Beispielabfragen aus den Tutoriumsfolien aus.
Alle werden wie erwartet umgesetzt

```
jonas@jonas-ThinkPad-T400 ~/Git/ti2-c05/04_C05/aufgabe01 $ make
g++ -g -Wall -Wextra -std=c++0x   -c -o ti2sh.o ti2sh.cc
g++ -g -Wall -Wextra -std=c++0x    ti2sh.o r.o parser.h   -o ti2sh
jonas@jonas-ThinkPad-T400 ~/Git/ti2-c05/04_C05/aufgabe01 $ ./ti2sh
ti2sh$ ls -1
command: ls, background: nein
Makefile
parser.h
r.l
r.o
ti2sh
ti2sh.cc
ti2sh.o
ti2sh$ sleep 10 &
command: sleep, background: ja
ti2sh$ echo foo
command: echo, background: nein
foo
ti2sh$ bla/fasel
command: bla/fasel, background: nein
execv: : No such file or directory
ti2sh$ sleep 100
command: sleep, background: nein
ls
```

Während des “sleep 100”wird ls eingegeben, wie man sieht wird dieser erstmal nicht verarbeitet.
Nach ablaufen des sleeps wird der eben eingegebene ls-Befehl richtig verarbeitet.

```
ti2sh$ sleep 100
command: sleep, background: nein
ls
ti2sh$ command: ls, background: nein
Makefile  parser.h  r.l  r.o  ti2sh  ti2sh.cc  ti2sh.o
ti2sh$
```

Hier wird nochmal ls erst im Hintergrund und dann im Vordergrund ausgeführt, man sieht deutlich, dass dies richtig funktioniert.

```
ti2sh$ ls &
command: ls, background: ja
ti2sh$ Makefile  parser.h  r.l  r.o  ti2sh  ti2sh.cc  ti2sh.o
ls
command: ls, background: nein
Makefile  parser.h  r.l  r.o  ti2sh  ti2sh.cc  ti2sh.o
ti2sh$
```

Hier wird getestet, ob ein Befehl auch mit mehreren Optionen ausgeführt werden kann, auch dies funktioniert.

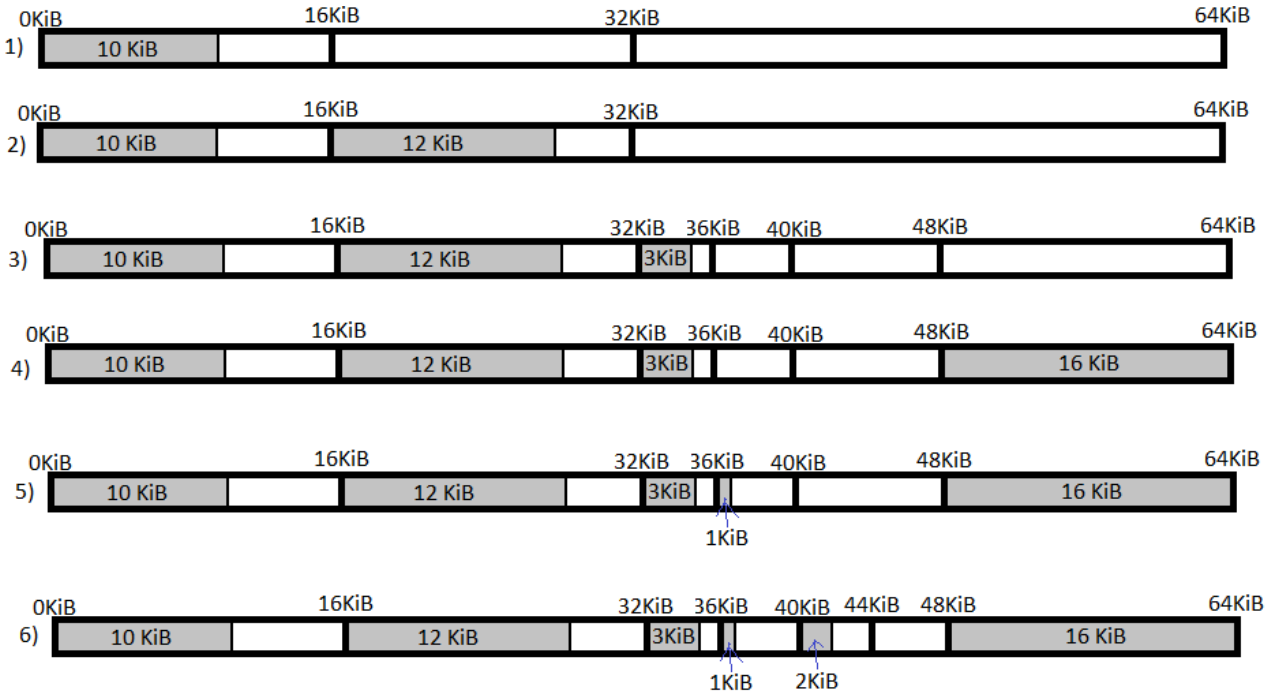
```
ti2sh$ ls -al
command: ls, background: nein
insgesamt 216
drwxr-xr-x 2 jonas jonas  4096 Nov 28 20:46 .
drwxr-xr-x 3 jonas jonas  4096 Nov 27 11:59 ..
-rw-r--r-- 1 jonas jonas   206 Nov 27 11:59 Makefile
-rw-r--r-- 1 jonas jonas   160 Nov 27 11:59 parser.h
-rw-r--r-- 1 jonas jonas  1029 Nov 27 11:59 r.l
-rw-r--r-- 1 jonas jonas 26160 Nov 27 12:59 r.o
-rwxr-xr-x 1 jonas jonas 79008 Nov 28 20:46 ti2sh
-rw-r--r-- 1 jonas jonas  4074 Nov 28 20:38 ti2sh.cc
-rw-r--r-- 1 jonas jonas 83768 Nov 28 20:46 ti2sh.o
ti2sh$
```

Hier wird getest, ob absolute Pfadangaben richtig funktionieren, auch dies ist der Fall.

```
ti2sh$ /bin/ls
command: /bin/ls, background: nein
Makefile  parser.h  r.l  r.o  ti2sh  ti2sh.cc  ti2sh.o
ti2sh$ /bin/ls &
command: /bin/ls, background: ja
ti2sh$ Makefile  parser.h  r.l  r.o  ti2sh  ti2sh.cc  ti2sh.o
```

Aufgabe 2

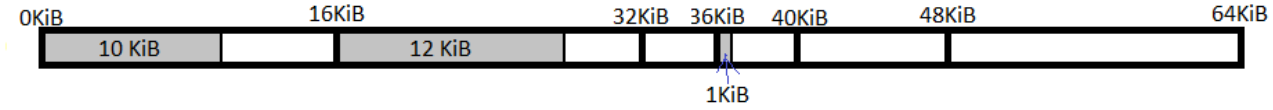
zu a)
Einfügen der Anforderungen:



Anforderung für 20 KiB ist nicht erfüllbar, da kein Platz groß genug ist (siehe von Schritt 5) auf 6)). Deshalb wird in Schritt 6 auch die Anforderung von 2 KiB erfüllt (dafür ist genug Platz). Somit bleiben 6 freie Blöcke übrig:

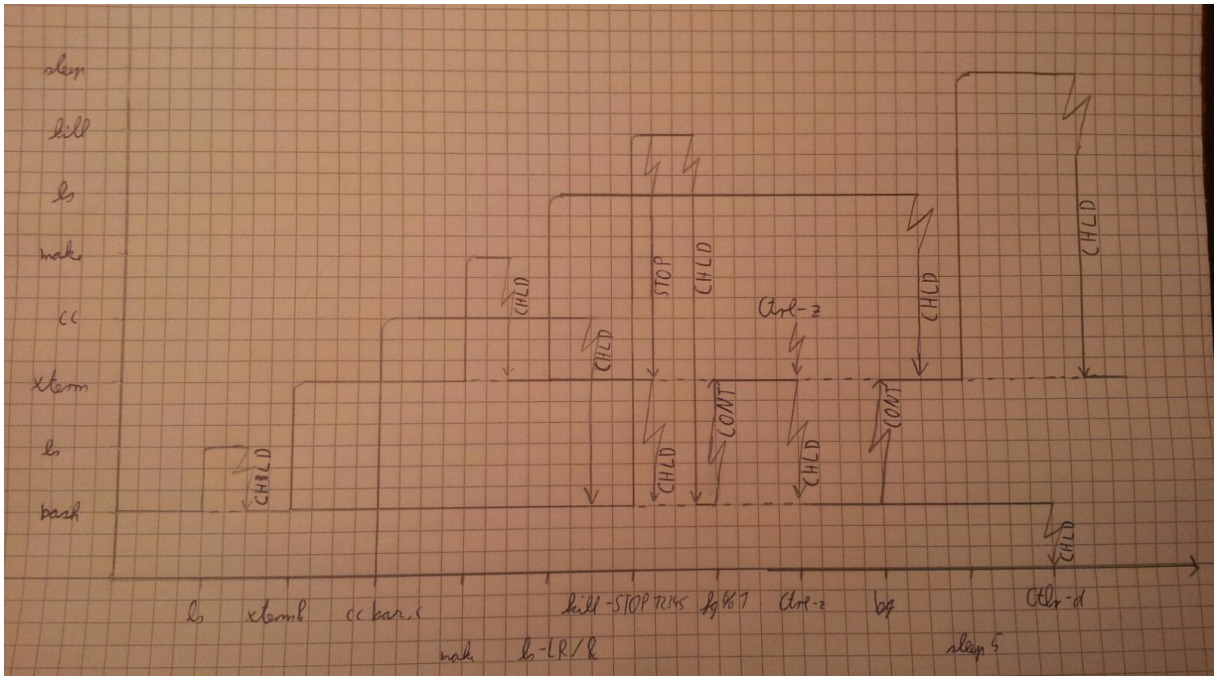
Block	Anfangsadresse	Endadresse	Größe
1	10 KiB	16 KiB	6 KiB
2	28 KiB	32 KiB	4 KiB
3	35 KiB	36 KiB	1 KiB
4	37 KiB	40 KiB	3 KiB
5	42 KiB	44 KiB	2 KiB
6	44 KiB	48 KiB	4 KiB

zu b)
Es gibt nun noch folgende Blöcke:



Nach der Freigabe kann keine Anforderung von 21 KiB erfüllt werden, da kein Speicherplatz groß genug ist. Wenn nun noch die Anforderung von 1 KiB freigegeben werden würde, würde die Anforderung von 21 KiB erfüllt werden können.

Aufgabe 3



Die x-Achse stellt die Befehle dar, die y-Achse die Prozesse. Prozesse mit einem &, die im Hintergrund laufen und irgendwann fertig sind, senden, beim Beenden, bei uns ein CHLD an ihren Vater, falls der beendet worden wäre, würde es an dessen Vater gesendet werden.
Am Ende gehört das in Schritt 2 gestartete Terminal zum Vater-Prozess der Bash aus Schritt 0, da diese an diesen vererbt wird, wenn die Bash beendet wird.