Technische Informatik 2 WS 2016/17 C08
Tutor: Marc Hildebrandt

Timo Jasper (Inf, 3.FS.)
Thomas Tannous (Inf, 3.FS.)
Moritz Gerken (Inf, 3.FS.)

Übungsblatt 4

1	2	3	4	5	$\sum$

Lösungsvorschlag
Abgabe: 28.11.2016

Gruppe

Oliver Hilbrecht hat sich entschlossen TI2 in diesem Semester abzubrechen.

Aufgabe 1

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include "string.h"
#include <sys/wait.h>
#include <unistd.h>
#include <iostream>

#include "parser.h"

using namespace std;

/*
Diese Methode kriegt einen command namen
und sucht durch PATH nach dem entsprechenden Programm ab
und gibt den absoluten Pfad zum Programm zurueck
   */
char* getPathToExec(const char* cmdname) {

    char *path = getenv("PATH");
    char* pathtok = strtok(path, ":");
    char* patharr[100];
    int k=0;
    while(pathtok != NULL) {
        pathtok = strtok(NULL, ":");
        patharr[k] = pathtok;
        k++;
    }
    char* finalpath;
```

```
    for(int i=0; i< (sizeof(patharr)/sizeof(patharr[0])); ++i ) {
          asprintf(&finalpath,"%s%s%s", patharr[i], "/",cmdname);
          if(access(finalpath,X_OK) == 0) {
             printf("%s", finalpath);
             return finalpath;
          }
    }
    printf("no command with this name");
    return NULL;
}


void handle_sigchld(int signum, siginfo_t *siginfo, void*) {
    int temp_errno = errno;
    while(waitpid((pid_t)(-1), 0, WNOHANG) > 0) {}
    errno = temp_errno;
}

int main(){
    for (;;){
        struct command cmd = read_command_line();
        cout << "command: " << cmd.argv[0]
             << ", background: " << (cmd.background ? "ja" : "nein")
                << endl;
       pid_t pid;

       //signal handler
       struct sigaction sigact;
       sigact.sa_flags = SA_RESTART;
       sigact.sa_sigaction = &handle_sigchld;



       switch( pid=fork() ) { //fork erzeugt kopie des Prozesses.
           case -1:
               printf("Fehler bei fork()\n");
               fflush(stdout);
               break;
           case 0: // im Kindprozess
               {
                   const char* completecmd = getPathToExec(cmd.argv
                      [0]);
                   execv(completecmd,cmd.argv); //fuehrt das Programm
                        completecmd im Kindprozess aus
                   fflush(stdout);
                   break;
               }
           default: // im Elternprozess
               if(cmd.background == 0)
                   wait(0);
```

```
                else if (sigaction(SIGCHLD, &sigact, NULL) == -1) {
                    perror("Error, cannot handle");
                    exit(1);
                }
                fflush(stdout);
                break;
            }
        }
        return 0;
}
```

Wir haben uns für den Signal Händler des Signals SIGCHLD entschieden. Dieser sorgt dafür das Hintergrundprozesse der Shell, sobald sie in den Zombie-Status wechseln, direkt aus der Prozess-Tabelle vom Parent entfernt werden. Im Handler selbst wird waitpid mit -1 aufgerufen was bedeutet, dass wait auf den Kindprozess warten soll. WNOHANG sorgt dafür, dass waitpid sofort returned, wenn kein Kind sich schließt. Um die Umgebungsvaribale PATH zu parsen, haben wir die extra Funktion getPathToExec geschrieben, welche aus einem command Namen, wie z.B. "ls"den Pfad zum binären Programm angibt, indem es alle Pfade, die in PATH enthalten ist durchläuft und mit access checkt, ob die Datei vorhanden ist. X_OK checkt zu dem zusätzlich für executable permissions. Falls ein Programm nicht im Hintergrund laufen soll warten im Elternrprozess mit wait auf die Terminierung des Kindprozesses.

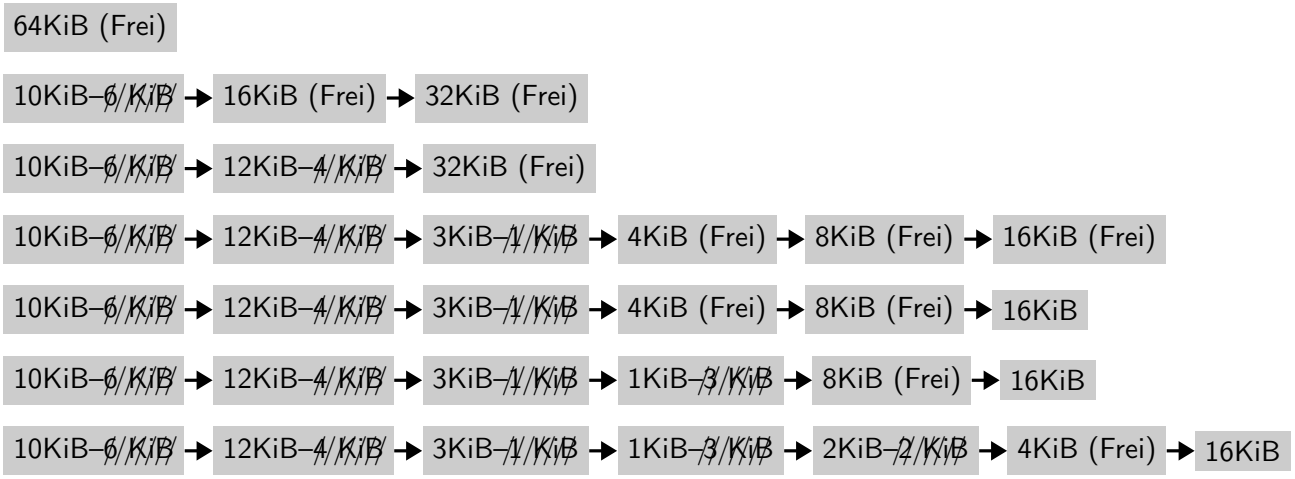
Tests

Wir haben Tests entsprechend aller Anforderungen des Übungsblattes ausgeführt.
Wir haben verschiedene Kommandos mit verschieden vielen Parametern getestet.
Wir haben Prozesse im Vorder- und Hintergrund laufen lassen, zum teil Parallel.(sleep 10, während sleep 60 & im Hintergrund läuft)
Wir haben Befehle mit absoluten Pfaden ausgeführt.
Verzeichnisswechsel d. Shellstatus war nicht erfolgreich, die Aufgabenstellung verlangt nur das arbeiten mit absoluten Pfaden, daher ist auch nicht relevant, welches Verzeichnis im Shell Status ist.
Das gleichzeitige ausführen mehrerer Kommandos mit einer Zeile wurde bereits als nicht implementiert vorgegeben.

Eingabe	Erw. Ausgabe/Reaktion	Ergbenis
ti2sh$ ls	Inhalte aktuelles Verzeichnis	erfolgreich
ti2sh$ ls -l	die selben Inhalte+weitere Infos in vertikaler Liste	erfolgreich
ti2sh$ ls -l -i	nochmal das selbe + Inode Nummer	erfolgreich
ti2sh$ echo Test	Ausgabe von "Test"	erfolgreich
ti2sh$ sleep 60 &	Prozess soll im Hintergrund laufen	erfolgreich
ti2sh$ sleep 10	Prozess sollte für 10sec im Vordergrund laufen	erfolgreich
ti2sh$ cd ./src/	Verzeichnis der Shell wechseln	nicht erfolgreich
ti2sh$ evince ~/home/.../ueb4.pdf	pdf anzeige d. 4 Übungsblattes	erfolgreich
ti2sh$ sleep 5 & jobs	Ausgabe des Kindprozesses (sleep)	nicht impl.
ti2sh$ ctrl. + d	Shell schließt sich	erfolgreich

Aufgabe 2

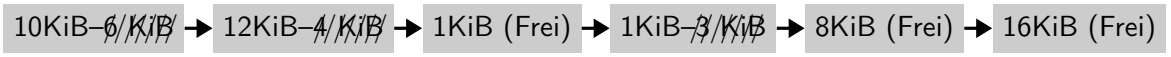
2 a)



Die Anforderung von 20KiB kann nicht erfüllt werden, da zu dem Zeitpunkt bereits kein Block von außreichender Größe mehr Frei ist.

Der in der Abb. markierte Block mit der Größe 4KiB ist frei geblieben. Die Adresse haben wir ermittelt, indem wir von der Adresse: 0 an, die Blockgrößen aufaddiert haben, bis zum Freien 4 KiB Block. Der Freie Block befindet sich also an der Adresse: 45056.

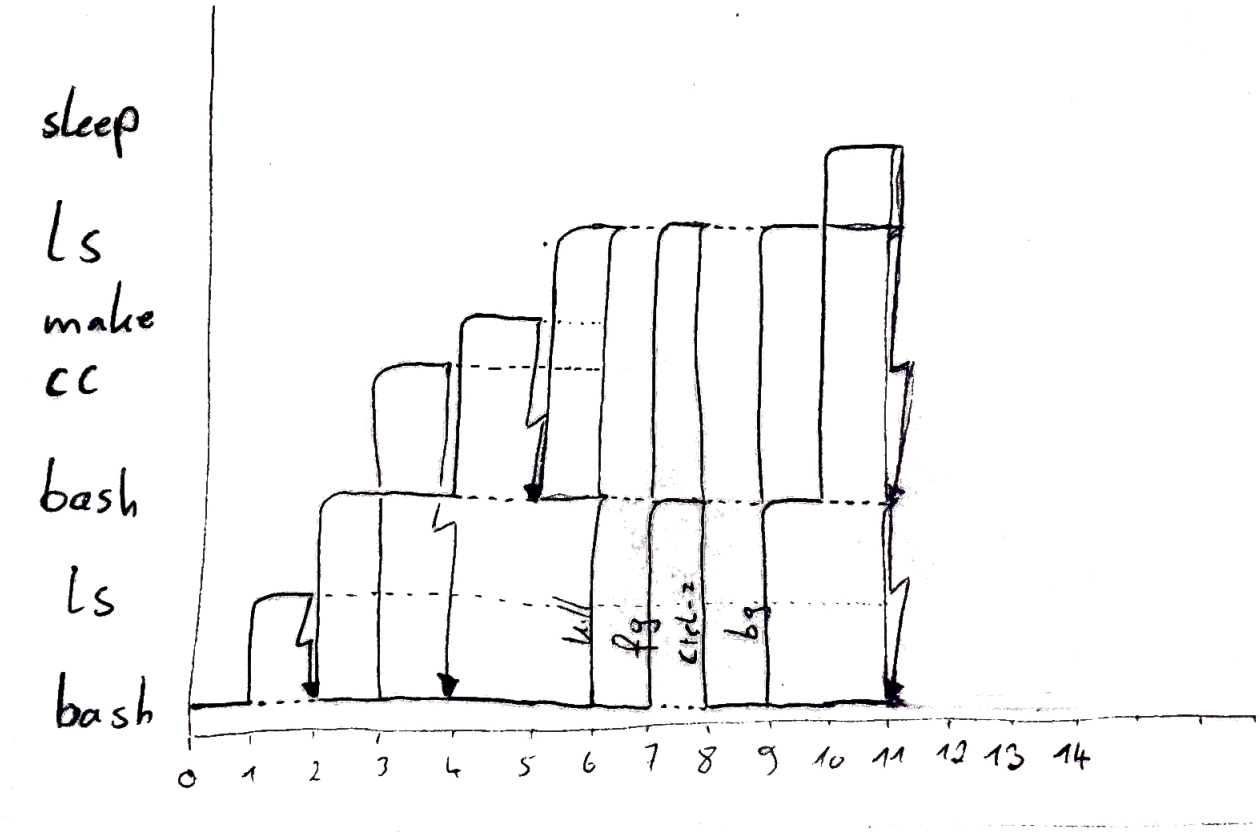
2 b)



Der Freigewordene 2KiB Block wird mit dem noch aus Aufgabe 2a) Freien 4 KiB Block verschmolzen und bilden einen 8 KiB Block, weitere Verschmelzungen sind nicht möglich.

Da der größte freie Block nur 16 KiB groß ist, kann die Anforderung von 21 KiB nicht erfüllt werden, außerdem muss auch noch die nicht erfüllte Anforderung von 20 KiB aus Aufgabe 2 a) berücksichtigt werden.

Aufgabe 3



Das Terminal von Schritt 2 gehört am Ende zum sleep Prozess.