Aufgabe 1

(a) Erstellen Sie einen Daemon Prozess, welcher jede Sekunde HELLO WORLD in eine Datei schreibt. (siehe man 2 write, man 2 open)

Ein Beispielquellcode:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
int main() {
  int pid, fd;
  //>>>>>>>> fork <<<<<<<<//
  if ((pid = fork()) == -1) {
    perror("fork call");
    exit(1);
  }
  else {
    if (pid != 0) {
        exit(0);
    }
  }
  //>>>>>>>> setsid <<<<<<<<//
  if (setsid() == -1) {
    perror("setsid call");
    exit(2);
  }
  //>>>>>>>> change dir to root <<<<<<<<//
  if (chdir("/") == -1) {
    perror("chdir call");
    exit(3);
  }
  //>>>>>>>> set umask <<<<<<<<//
  umask(0);
  //>>>>>>>> close stdio /stderr
  close(0); close(1); close(2);
  //>>>>>>>> open file <<<<<<<<//
  fd = open("/tmp/hello.txt",O_CREAT|O_WRONLY|O_APPEND,0644);
  if (fd == -1) {
    perror("open call");
    exit(4);
  }
  //>>>>>>>> write to file <<<<<<<<//
  while(1) {
    write(fd, "Hello World\n", 11);
    sleep(1);
  }
  return 0;
}
```

Alternative Lösung zu (a), die Streams anstatt von Filedeskriptoren verwendet
(auf eine Überprüfung der Rückgabewerte wurde hier verzichtet)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(){
  int pid=fork();
  switch(pid){
    case 0: // Kind
      setsid();
      chdir("/");
      umask(0);
      fclose(stdin);
      fclose(stdout);
      fclose(stderr);
      FILE* x=fopen("/tmp/hello","a");
      while(1==1){
        fprintf(x,"HELLO WORLD\n");
        fflush(x);
        sleep(1);
      }
      break;
    case -1: // Fehler
      return 1;
      break;
    default: // Vater
      return 0;
  }
}
```

(b) Welche exec-Funktionen gibt es, und wie unterscheiden sich diese?

execl, execlp, execle:

Bekommen als zusätzliche(n) Parameter eine Liste der Argumente für das auszuführende Programm (alle Argumente sind Null-terminierte Strings).

Zum Abschluss der Liste nutzen Sie (char*) NULL.

execv, execvp:

Bekommen als zusätzlichen Parameter ein Array von Pointern auf Null-terminierte Strings, die ebenfalls als Argumente für das auszuführende Programm dienen.

Erstes Element im Array muss der Dateiname der auszuführenden Datei sein, das letzte wieder ein NULL-Pointer.

(c) Wie unterscheidet sich die exec-Funktionen Familie vom system-Befehl?

exec() überschreibt den Programmcode bzw. das ganze Prozessimage des eigenen Prozesses im Hauptspeicher und führt den neuen Programmcode aus. Programmcode der nach dem exec-Aufruf steht wird also nicht ausgeführt.

system() startet das übergebene Programm als neuen Prozess und der Programmfluss im eigenen Prozess geht anschließend weiter. Dadurch, dass ein neuer Prozess gestartet wird, ist system() auch langsamer als exec().

Aufgabe 2

(a) Führen Sie in einer Shell das Programm top aus und suchen Sie nach der Anzeige für die Anzahl von Zombieprozessen. Lassen Sie diese Shell geöffnet.

Die Anzahl der Zombies befindet sich in der zweiten Zeile der top-Ausgabe:

```
top - 18:49:02 up 6 days, 45 min,  1 user,  load average: 0.00, 0.03, 0.03
Tasks: 118 total,   2 running, 116 sleeping,   0 stopped,   0 zombie
```

(b) Schreiben Sie ein C-Programm, in dem ein Kindprozess erzeugt wird. Der Vaterprozess wartet 2 Minuten (man 3 sleep) bevor er waitpid aufruft (man waitpid). Führen Sie das Programm aus und beobachten Sie innerhalb von 2 Minuten die Ausgabe von top (oder auch STAT Z in ps u).

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int main(){
  int pid, childpid, mypid;
  pid=fork();
  switch(pid){
    case 0:
      childpid = getpid();
      printf("Ich bin das Kind mit der PID %d.\n", childpid);
      break;
    case -1:
      printf("Ich konnte kein Kind erzeugen.\n");
      exit(1);
      break;
    default:
      mypid = getpid();
      printf("Ich bin der Vater mit der PID %d.\n", mypid);
      sleep(120); // Kind bleibt Zombie
      waitpid(pid,NULL,0); // Der Zombie verschwindet
  }
}
```

In top sollte ein Zombie Prozess zu sehen sein. Alternative Beobachtung mit ps u:

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
user     23010  0.0  0.0      0     0 pts/3    Z+   18:51   0:00 [for] <defunct>
```

(c) Wenn Sie waitpid auskommentieren, bleibt dann das Kind für immer ein Zombie?

Das Kind bleibt kein Zombie. Sobald sich der Vaterprozess beendet, erbt der init-Prozess dessen Kinder und liest die Prozessstati aus.

(d) Richtet ein Zombieprozess Schaden an? Wenn ja, welchen?

Ja:

- braucht Speicher für Eintrag in der Prozesstabelle .
- braucht PID
- bei massenhaftem Auftreten können Zombies durch das Verbrauchen aller für Prozesse vorgesehenen Ressourcen das Starten weiterer Prozesse verhindern.

Aber:

- maximale Anzahl an Prozessen pro UID kann festgelegt werden.
- wenn der Elternprozess den Exit-Status nicht abfragt, wird das nach Beenden des Elternprozesses automatisch vom init-Prozess übernommen.

(e) Modifizieren Sie das Programm so, dass es ständig neue Zombies erzeugt. Nachdem kein weiteres Kind erzeugt werden kann, gibt es eine entsprechende Meldung aus und bleibt noch 2 Minuten aktiv bevor es sich schließlich selbst beendet. Versuchen Sie nach der Misserfolgsmeldung weitere Prozesse zu starten, z.B. ein einfaches ls in einer bereits geöffneten Konsole.

Falls top noch geöffnet ist, zeigt es jede Menge Zombies bei der Ausführung folgendes Programmes an.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(){

  int pid, childpid, mypid;
  int cont=1;

  while (cont==1){
    pid=fork();

    switch(pid){

      case 0:
        childpid = getpid();
        printf("Ich bin das Kind mit der PID %d.\n", childpid);
        cont=0;
        break;

      case -1:
        printf("Ich konnte kein Kind erzeugen.\n");

        sleep(120);
        exit(1);
        break;

      default:
        mypid = getpid();
        printf("Ich bin der Vater mit der PID %d.\n", mypid);
    }
  }

}
```

Das Starten weiterer Prozesse funktioniert nicht mehr ohne weiteres.

Beispiel:

```
    $ ps u
bash: fork: Cannot allocate memory
```

Es können jedoch Prozesse gestartet werden, wenn stattdessen andere beendet werden. Den kill Befehl kann man verwenden, da er als shell builtin keinen neuen Prozess startet. Das Schließen von Fenstern beendet ebenfalls Prozesse.

(f) Wie kann man Zombies aus dem System endgültig entfernen?

Das kann der Vaterprozess selbst, wenn er wait bzw. waitpid ausführt. Alternativ kann er sich beenden (exit, return) oder beendet werden (kill), so dass der init-Prozess den Zombie beseitigt.

Hinweis. Ein Reboot löscht alle Zombie-Prozesse.