1. exec-Varianten

a) Betrachten Sie das folgende Programm `exec-beispiele.c`:

```
1   // exec-beispiele.c
2   // Betriebssysteme 1, v0.1, 2024-04-23
3
4   #include <stdlib.h>
5   #include <unistd.h>
6   #include <sys/types.h>
7   #include <sys/wait.h>
8   #include <stdio.h>
9
10  void execv_test () {
11    char *path = "/usr/bin/head";
12    char *args[] = {
13      "head",
14      "-1",
15      "/etc/os-release",
16      NULL
17    };
18    execv (path, args);
19  }
20
21  void execl_test () {
22    execl ("/usr/bin/head",
23      "head",
24      "-1",
25      "/etc/os-release",
26      NULL);
27  }
28
29  void execle_test () {
30    char *s = getenv("HOME");
31    printf ("HOME = %s\n", s);
32
33    char *env[] = { "HOME=/test/xyz", NULL };
34    execle ("./printhome", "printhome", NULL, env);
35  }
36
37  int main () {
38    int pid;
39
40    // execv-Test
41    pid=fork(); if (pid==0) execv_test(); else wait(NULL);
42
43    // execl-Test
44    pid=fork(); if (pid==0) execl_test(); else wait(NULL);
45
46    // execle-Test
47    pid=fork(); if (pid==0) execle_test(); else wait(NULL);
48  }
```

Ab Zeile 40 erzeugt das Programm jeweils mit `fork()` einen Kindprozess, führt in diesem eine der Testfunktionen (`execv_test`, `execl_test`, `execle_test`) aus und wartet im Parent-Prozess mit `wait(NULL)` darauf, dass der gerade erzeugte Kindprozess sich beendet.

Es gibt die folgenden Zeilen aus:

```
PRETTY_NAME="Ubuntu 22.04.3 LTS"
PRETTY_NAME="Ubuntu 22.04.3 LTS"
HOME = /home/swf
HOME = /test/xyz
```

In den letzten beiden Zeilen der Ausgabe erscheint jeweils der mit `getenv()` abgefragte Wert der Umgebungsvariable `HOME` – die erste Zeile per direktem Aufruf, die zweite Zeile durch Start des Programms `printhome` (das auch `getenv()` nutzt). Der Code zu `printhome` ist dieser:

```
1       #include <stdlib.h>
2       #include <unistd.h>
3       #include <stdio.h>
4
5       int main() {
6         char *s = getenv("HOME");
7         printf ("HOME = %s\n", s);
8       }
```

b) Wechseln Sie (wie in der letzten Übung) in der Shell in den Ordner, in dem Sie Übungsdateien zu Betriebssysteme 1 ablegen, und starten Sie einen Ubuntu-Docker-Container. Laden Sie dann die Beispieldateien herunter und testen Sie die Ausführung:

```
cd /realworld
wget swf.hgesser.de/bs-b1/prakt/bs1-ue03.zip
unzip bs1-ue03.zip
cd ue03
make
./exec-beispiele
```

Testen Sie auch das Programm `printhome`:

```
./printhome
```

(Achten Sie bei den Aufrufen jeweils auf die Zeichen `./` unmittelbar vor dem Programmnamen.)

c) Lesen Sie die Manpages zu `exec`, `wait` (dort nur den Anfang, bis vor der Auflistung der Konstanten), `getenv`. Erklären Sie, warum in der letzten Zeile der Ausgabe `HOME = /test/xyz` erscheint.

d) Ändern Sie in Zeile 33 den ersten Array-Eintrag von `"HOME=/test/xyz"` in `"TEST=/test/xyz"` und übersetzen (`make`) und starten Sie das Programm erneut. Was ändert sich, und woran liegt das?

2. spawn-Implementierung

Im Quellcode-Ordner `ue03/` ist noch eine weitere Datei `spawn-beispiele.c` enthalten. Sie ähnelt der ersten Quellcode-Datei, aber hier werden Funktionen `spawnv()` und `spawnve()` aufgerufen, die es noch nicht gibt. Sie sollen ähnlich arbeiten wie `execv()` und `execve()`, dabei aber zunächst einen Kindprozess erzeugen, in dem dann ein Programm nachgeladen wird.

a) Aktivieren Sie im `Makefile` die Übersetzung der dritten Quelldatei, indem Sie die führende Raute (`#`) in Zeile 4 entfernen. Das führende Tabulatorzeichen muss (!) erhalten bleiben. Erzeugen Sie mit `make` das Programm `spawn-beispiele` und testen Sie es. (Es sollte nur eine Zeile ausgeben.)

b) Implementieren Sie `spawnv()` und `spawnve()`. Die Funktionen sollen im Erfolgsfall die Prozess-ID des erzeugten Kindprozesses zurückgeben, im Fehlerfall (= `fork()` nicht erfolgreich) den Wert `-1`.

3. Logrun (Zusatzaufgabe)

Entwickeln Sie ein Programm `logrun.c`, das seine Parameter als auszuführendes Kommando interpretiert. Es soll also mit einem geeigneten Aufruf einer Funktion aus der `exec*`-Familie das angegebene Programm nachladen und mit den Parametern laufen lassen.

Beispiel:

```
swf@7e1d87ae5b0c:~/host$ printf "%5.2f\n" "3,1415"
3,14
swf@7e1d87ae5b0c:~/host$ ./logrun printf "%5.2f\n" "3,1415"
3,14
```

Nutzen Sie dazu aus, dass Sie im Hauptprogramm mit

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

die Anzahl der Aufrufargumente (`argc`) und ein Array von Aufrufargumenten (`argv`) auswerten können. Die Struktur von `argv` passt dabei gut zu einer der `exec*`-Funktionen – es gibt nur am Anfang „ein Argument zu viel“.

Vor dem Nachladen des Programms soll `logrun` in einer Logdatei das vollständige Kommando mit Datum und Uhrzeit im Format

```
2024/04/23 06:40 printf "%5.2f\n" "3,1415"
```

protokollieren. Dabei hilft folgender Code[1]:

```
#include <stdio.h>
#include <time.h>
int main() {
  time_t t = time(NULL);
  struct tm tm = *localtime(&t);
  printf("now: %d-%02d-%02d %02d:%02d:%02d\n",
    tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday, tm.tm_hour, tm.tm_min, tm.tm_sec);
}
```

(Die Datei liegt auch als `gettime.c` im Ordner `ue03/`.)

1 Quelle: https://stackoverflow.com/questions/1442116/how-to-get-the-date-and-time-values-in-a-c-program