E 1: Systemprogrammierung und parallele Prozesse

Lernziele

- Mit den Beschreibungen von Systemfunktionen eigene systemnahe Programme entwickeln können
- Prozessverdoppelung und Prozessverkettung unter Unix praktisch durchführen
- Mit den Beschreibungen von Systemfunktionen arbeiten lernen

Übungsumgebung

- Diese Übung kann unter Unix oder Linux ausgeführt werden.
- GCC (GNU Compiler Collection) installiert
- Vorlagedateien (C-Quellcode): **`forktest.c`**, **`demoexec.c`**

Allgemeine Hinweise

- Eine besser aufbereitete Version der Unix-Handbuchseiten ist auf dem Web unter:
  `http://www.opengroup.org/susv3/`
  zu finden. Auf dieser Webseite ist zweimal "System Interfaces" zu wählen, um eine Übersichtsliste aller Systemaufrufe zu erhalten. Über diese können detaillierte Beschreibungen abgefragt werden. Übrigens, diese Liste enthält auch die Funktionen der C-Standardbibliothek.
- Für diese Übung ist ein Linux-VMware-Image mit dem Namen *"Betriebssysteme-Linux"* vorbereitet. Der Benutzername ist: "hsr" und das Kennwort: "welcome". Benutzen Sie bitte das Verzeichnis **`/tmp`** um die Übungen durchzuführen, da die Anbindung an das HSR-Homeverzeichnis mangelhaft ist (nur für Kopieren geeignet).

Aufgaben

E 1.1: Anfangsparameter für Prozesse.

Beim Starten eines C-Programms können dem Programm zusätzliche Argumente übergeben werden. Diese Argumente werden der **`main()`**-Funktion des Programms als Parameter übergeben. Die vollständige Signatur der **`main(`**-Funktion lautet:

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

Als erster Parameter wird die Anzahl Argumente, als zweiter Parameter die Argumentliste in Form eines zweidimensionalen Vektors übergeben.

Aufgabe: Schreiben Sie ein Programm **`ArgPrinter`**, welches beim Aufruf die Anzahl Argumente und die Argumente selbst auf der Konsole ausgibt. Wenn beispielsweise in der Konsole

```
ArgPrinter www.hsr.ch 443 fmuster@hsr.ch
```

eingegeben wird, soll die Ausgabe wie folgt lauten:

```
Anzahl Argumente: 3
www.hsr.ch
443
fmuster@hsr.ch
```

E 1.2 Umgebungsvariablen

Umgebungsvariablen können in Programmen für unterschiedliche Zwecke benutzt werden. Bei der Java Programmentwicklungsumgebung (SDK) beispielsweise dient die Umgebungsvariable mit dem Namen **`CLASSPATH`** dazu, eines oder mehrere Verzeichnisse anzugeben, in denen nach Dateien des Typs **`*.class`** gesucht werden soll.

a) Die Umgebungsvariablenliste enthält eine Anzahl von Paaren:
{Umgebungsvariablenname, Umgebungsvariableninhalt}
in der Form von einzelnen Zeichenketten der Art (Beispiel):
`"LOGNAME=fmueller"`
bzw. allgemein: *"Umgebungsvariablenname=Umgebungsvariableninhalt"*
Auf diese Liste kann über die globale Variable **`environ`** zugegriffen werden. Diese ist in eigenen Programmen wie folgt zu deklarieren:
`extern char **environ`
Es handelt sich genau genommen um einen Vektor, der eine Menge von Zeigern des Typs **`char *`** enthält. Die einzelnen Zeichenketten selbst sind woanders abgelegt – über die Zeiger können diese aber abgerufen werden.
Der Zugriff auf den i-ten Eintrag in der Liste erfolgt mittels **`environ[i]`**. Die Liste wird übrigens durch einen **`NULL`**-Zeiger abgeschlossen (= letztes gültiges Vektorelement).

Aufgabe: Schreiben Sie ein kleines Programm, das den gesamten Inhalt der Umgebungsvariablenliste auf die Konsole ausgibt. Überprüfen Sie Ihr Programm, indem Sie die Programmausgabe mit der Ausgabe des Kommandozeilenbefehls **`printenv`** vergleichen.

b) Üblicherweise will man in einem Programm gezielt auf eine bestimmte Umgebungsvariable zugreifen. Dazu stehen die Systemfunktionen **`getenv()`** und **`putenv()`** zur Verfügung.

Aufgabe: Schreiben Sie ein kleines Programm, das eine Umgebungsvariable mit dem Namen **`EU`** und dem Inhalt **`Europa`** anlegt. Zusätzlich soll mittels der Funktion **`getenv()`** der aktuelle Inhalt der Umgebungsvariablen **`EU`** zur Kontrolle auf die Konsole ausgegeben werden. Ihr Code sollte auch den Fehlerfall, zB. fehlende Umgebungsvariable **`EU`**, behandeln.

c) Wenn Sie nach erfolgreicher Ausführung des Programms aus der vorangegangenen Teilaufgabe mit dem Kommandozeilenbefehl **`printenv`** die Umgebungsvariablenliste abfragen, so werden Sie feststellen, dass der Eintrag **`EU=...`** fehlt. Wieso ist dies so?

.....

.....

E 1.3 Prozessverdoppelung mit **`fork()`**.

Testen Sie die Prozessverdoppelung durch **`fork()`** mit dem Beispielsprogramm **`forktest.c`**.
Der Quellcode kann mit **`gcc forktest.c -o forktest`** in die ausführbare Datei **`forktest`** übersetzt werden.

Für die Beobachtung der Prozesse ein anderes Fenster öffnen und vorhandene Prozesse mit **`ps -la`** (auch **`ps -lea`** verwenden) ermitteln. Prozesstabelle vor, während des Parallelablaufs und nach Beendigung des ersten Prozesses anschauen (*Linux*: unter Gnome steht das komfortable Programm "Systemmonitor" zur Verfügung). Anschließend Programm starten mit **`forktest`** (falls dies nicht geht mit **`./forktest`**) und Ablauf beobachten (inklusive **`ps ...`**).

a) Gibt es Zombies? (Eintrag **<defunct>** bzw. "**z**" in Prozesstabelle (Anzeige mit Befehl: **ps**)

.....

.....

b) Ändern Sie das Programm **forktest.c** ab, indem Sie die Zeile mit dem **wait()**-Aufruf auskommentieren. Wie steht es nun mit den Zombies?

.....

.....

E 1.4 Prozessverkettung mit **execl()**.

a) Testen Sie das Chaining von Programmen (**execl**) mit dem Beispiel **demoexec.c**.
Studieren Sie dazu den Quellcode des Programms, übersetzen Sie ihn anschliessend und bringen Sie das Programm zur Ausführung. Läuft es wie erwartet?
NB: Falls nein, bitte Nachbar oder Betreuer fragen.

b) Unter welchen Umständen könnte die Ausgabe **"This should not happen!\n"** erscheinen?

.....

.....

c) Wieso wird der Text **"Child just died"** zweimal ausgegeben, wenn das Programm **date** nicht gestartet werden kann?

.....

.....

d) Bekommt der mit **execl** gestartete Prozess eine neue PID?

.....

.....

e) Kann der durch **execl** neu erzeugte Prozess weiterlaufen, auch wenn der **execl** aufrufende Prozess terminiert?
Falls ja: welches ist sein Elternprozess?
Hinweis: Evtl. Programm abändern, so dass diese Fragen experimentell beantwortet werden können.

.....

.....

f) Das Programm soll **0** im Normalfall bzw. **-1** im Fehlerfall zurückgeben. Erweitern Sie das Programm derart, dass mögliche Fehler behandelt werden.

NB: Sie können den Rückgabewert eines Programms auf der Konsole zur Anzeige bringen, indem Sie nach der Programmbeendigung den Befehl **echo $?** eingeben.
Mit **echo $?** erhalten Sie stets den Rückgabewert des zuletzt auf der gleichen Konsole ausgeführten Programms. Beispiel (**$** für Eingabeaufforderung bei Zeilenbeginn):

```
$ demoexec
$ echo $?
0
$
```

Musterlösung

E 1.1: Anfangsparameter für Prozesse.
Siehe Lösungsmuster auf Skriptablage: E1/loesung/ArgPrinter.c

E 1.2: Umgebungsvariablen.

a) Siehe Lösungsmuster auf Skriptablage: /E1/loesung/Environment.c

b) Siehe Lösung a)

c) Die Umgebungsvariablenliste gehört zur aktuellen Prozessumgebung. Wird der Prozess beendet, so verschwindet diese. Lediglich ein durch den Prozess neu erzeugter Prozess würde (im Regelfall) die Umgebungsvariablenliste erben.

E 1.3 Prozessverdoppelung mit **`fork()`**.

a) Da im Elternprozess ein Aufruf von **`wait()`** vorhanden ist, können höchstens temporär Zombieprozesse entstehen, die aber auf jeden Fall wieder verschwinden (wenn Elternprozess den erweiterten Beendigungsstatus entgegennimmt).

b) Ein Zombie wird in der Prozessliste sichtbar.

E 1.4: Chaining mit **`execl()`**.

a) Ja.

b) Wenn **`execl`**-Aufruf nicht erfolgreich war, z.B. weil Pfad der auszuführenden Datei ungültig ist.

c) Wie b), da die **`printf()`**-Anweisung nicht in einem **`else`**-Block der **`if`** -Anweisung steht.

d) Nein, erbt PID des **`execl`** aufrufenden Prozesses.

e) Ja. Als Elternprozess wird die PID=1 aufgeführt. Dies ist die Kennung des **`init`**-Systemprozesses. Er adoptiert alle verwaisten Kindprozesse, die man als Waisenprozesse (orphans) bezeichnet.

f) Es kommen drei Systemaufrufe vor, bei denen Fehler auftreten können. Im Fehlerfall beenden wir das Programm mit einem Rückgabewert von -1 anstatt von 0. Folgende Änderungen sind zu machen:
(1) Falls Rückgabewert von **`fork() == -1`** dann **`exit(-1)`**
(2) Falls Programm direkt nach **`execl`** weiterfährt, dann **`exit(-1)`**
(3) Falls Rückgabewert von **`wait() == -1`** dann **`exit(-1)`**
Siehe auch Lösungsmuster im Lösungsverzeichnis.