

## *Hilfsblatt zu Praktikumsaufgabe 4*

### **Grundlagen der Prozessverwaltung:**

Dieses Hilfsblatt beschreibt einige Grundlagen zur Behandlung von Prozessen unter POSIX in dem Umfang, wie sie zur Bearbeitung von Aufgabe 4 benötigt werden.  
Schauen Sie sich die benutzten Systemaufrufe zusätzlich in der Dokumentation an!

#### ***Erzeugen eines neuen Prozesses:***

In einem POSIX konformen System werden Prozesse mit dem `fork( )` Befehl erzeugt.

Syntax

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

```
pid_t fork(void);
```

`fork( )` erzeugt einen neuen Prozess (Kindprozess).

Einige Eigenschaften des Kindprozesses:

- Der Kindprozess erhält eine eindeutige Prozess Id.
- Der Kindprozess erbt alle offenen Dateien des Elternprozesses.
- Der Kindprozess erhält eine vollständige Kopie des Speichers des Elternprozesses.
- Eltern- und Kindprozess laufen in ihrer eigenen Prozessumgebung jeweils nach dem `fork( )` Aufruf weiter.

Rückgabewert:

- 1: falls `fork()` scheitert. Muss immer überprüft werden! Der Fehlergrund kann `errno` entnommen werden.
- 0: Der Kindprozess läuft weiter. Der neu erzeugte Prozess bekommt in seiner Umgebung die 0 zurückgeliefert, hieran kann der neue Prozess erkannt werden.
- >0: Der Elternprozess erhält die Prozess Id des Kindprozesses zurück.

#### ***Synchronisation des Elternprozesses mit den Kindprozessen***

```
#include <sys/wait.h>
```

```
pid_t waitpid(pid_t pid, int* stat_loc, int options);
```

Parameter:

`pid`:

- >0: Warte auf den Prozess mit der ID `pid`.
- = 0: Warte auf einen Prozess aus der gleichen Prozessgruppe. (Eine Prozessgruppe ist eine Menge von Prozessen, die zusammen Signale empfangen können. Ein Kindprozess ist in derselben Prozessgruppe wie sein Elternprozess. Prozesse können ihre eigene Prozessgruppe starten.)
- -1: Warte auf einen (beliebigen) Kindprozess.
- < -1: Warte auf einen Prozess, dessen Gruppen ID dem Betrag von `pid` entspricht.

`stat_loc`:

Ein Zeiger auf eine int Variable, in der der Status des Prozesses, auf die gewartet wurde, geschrieben wird.

Der Wert ist implementationsabhängig, deshalb muss die Auswertung über Makros erfolgen. Es stehen unter anderem folgende Makros zur Verfügung (für mehr: siehe man pages ...):

- `WIFEXITED(stat_loc)` ; : Liefert TRUE, wenn der Kindprozess sich normal beendet hat.
- `WEXITSTATUS(stat_loc)` ; : Liefert den exit-Code des Kindprozesses (untere 8 Bit), falls dieser sich normal beendet hat.
- `WIFSIGNALED(stat_loc)` ; : Liefert TRUE, wenn der Kindprozess durch ein Signal (z.B. durch CTRL-C oder durch `kill <pid>`) beendet wurde.
- ...

Braucht der Rückgabewert nicht berücksichtigt werden, kann ein NULL Zeiger übergeben werden.

`options`:

Dieser Parameter beschreibt, wie sich der `waitpid()` Befehl verhält. Die möglichen Optionen sind Bitwerte, die oder-verknüpft werden können. Z.B.:

`WNOHANG`: `waitpid()` suspendiert nicht den Elternprozess, sondern kehrt direkt wieder zurück. (Könnte benutzt werden, um Prozesse im Hintergrund zu starten, die Kontrolle wird an den Elternprozess zurückgegeben.)

Weitere Optionen können den man pages oder der Literatur entnommen werden, für dieses Praktikum brauchen keine Optionen benutzt werden (es wird eine 0 übergeben.).

Rückgabewert:

Die Prozess Id des Prozesses, auf den gewartet wurde.

(Alternativ kann auch der `wait()` Befehl benutzt werden, der auf einige Parameter verzichtet und damit weniger flexibel ist.)

## ***Beenden eines Prozesses***

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

```
void exit(int status);
```

Beendet einen Prozess.

Parameter:

status: Wird an den wartenden (Eltern-)Prozess zurückgeliefert.

Achtung: Sollte kein Elternprozess mit `waitpid()` auf den Aufruf von `exit` warten, wird der terminierende Prozess zu einem Zombie Prozess, er kann nicht korrekt beendet werden und seine Ressourcen freigeben!

## ***Austausch des Speicherbildes (Starten eines Programms)***

```
#include <unistd.h>
extern char **environ;

int exec<l|v>[e|p] (const char *path, parameter [,
environment]);
```

(-> 6 Möglichkeiten des Aufrufes)

Die exec – Befehlsfamilie lädt ein neues Programm. Der Speicherinhalt wird ausgetauscht. Wird das neue Programm erfolgreich gestartet, so läuft es mit der Prozess Id des Prozesses, der exec aufgerufen hat.

Konventionen:

execl[e|p]: Die Übergabeparameter, die der main-Funktion des zu starten Programms übergeben werden, werden in Listenform angegeben. Der letzte Parameter der Liste muss ein Null- Zeiger sein.

(Beispiel: execl(const char\* path, char \*arg0, char \*arg1, ...);)

execv[e|p]: Die Übergabeparameter, die der main-Funktion des zu starten Programms übergeben werden, werden als Vektor angegeben. Das letzte Vektorelement muss ein Null-Zeiger sein.

(Beispiel: execv(const char\* path, char \*argv[]);

vergleiche argv[] als Parameter der main() Funktion.)

exec<l|v>: Nur der Pfad und die Argumente werden übergeben.

exec<l|v>e: Als letzter Parameter wird ein Vektor auf die Umgebungsvariablen übergeben.

Die externe Variable environ ist ein Vektor mit den Umgebungsvariablen!

exec<l|v>p: Der Pfad des übergebenen Befehls wird in der Umgebungsvariable \$PATH gesucht.

Beispiele:

```
char * cmd[] = {"ls", "-l", (char *)NULL}; /* gilt für alle */
                                           /* Beispiele      */
```

```
/* Beispiel 1: */
```

```
execl("/bin/ls", "ls", "-l", (char *)NULL);
```

```
/* Beispiel 2: */
```

```
execv("/bin/ls", cmd);
```

```
/* Beispiel 3: */
```

```
execve("/bin/ls", cmd, environ);
```

```
/* Beispiel 4: */
```

```
execvp("ls", cmd);
```

```
/* Beispiel 5: */
```

```
execle("/bin/ls", "ls", "-l", (char *)NULL, environ);
```