Systemnahe Programmierung in C

28 Programme und Prozesse – UNIX

J. Kleinöder, D. Lohmann, V. Sieh

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Sommersemester 2025

http://sys.cs.fau.de/lehre/ss25

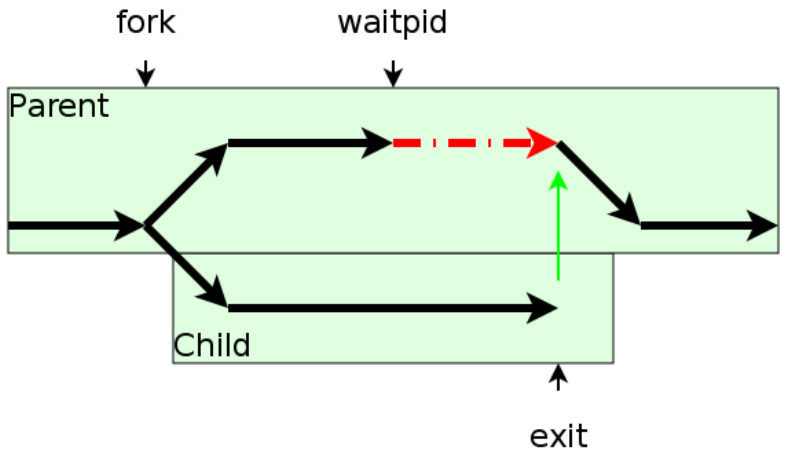
Beispiel: UNIX

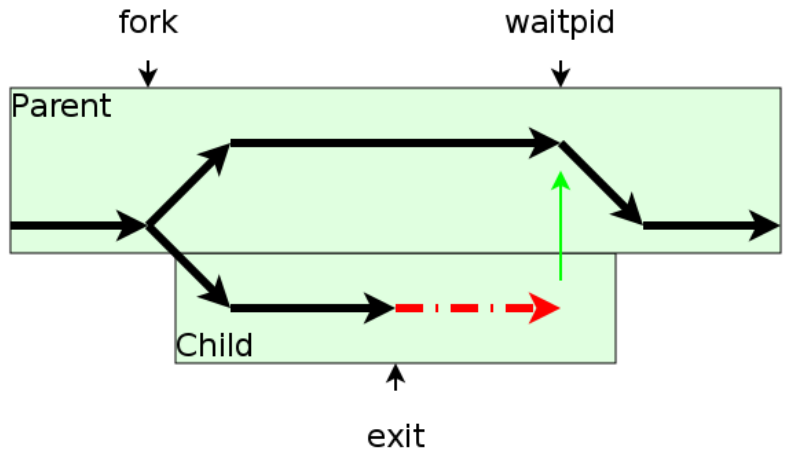
- Überblick:

fork: Erzeugung von neuen Prozessen

exit: Terminieren von Prozessen

waitpid: Warten auf das Terminieren von Prozessen





Programmierschnittstelle

- Duplizieren des gerade laufenden Prozesses

```
#include <unistd.h>

pid_t fork(void);
```

- Terminieren des aktuellen Prozesses

```
#include <stdlib.h>

void exit(int status);
```

- Warten auf das Terminieren eines anderen Prozesses

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

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


Beispiel

```c
pid_t pid, ret;
int status;

pid = fork();
switch (pid) {
case -1: /* Error */
    perror("fork");
    exit(1);

case 0: /* Child */
    do_child_work();
    exit(13);

default: /* Parent */
    do_parent_work();
    ret = waitpid(pid, &status, 0);
    /*
     * In case of no error:
     * ret == pid
     * WIFEXITED(status) == 1
     * WEXITSTATUS(status) == 13
     */
    break;
}
```

Prozesserzeugung

- Der Kind-Prozess ist Kopie des Eltern-Prozesses
  - gleiches Programm
  - gleiche Daten (Variablen-Inhalte)
  - gleicher Programmzähler
  - gleiches aktuelles Verzeichnis, Wurzelverzeichnis
  - gleiche geöffnete Dateien
- einzige Unterschiede
  - verschiedene Prozess-IDs
  - Rückgabewert von `fork`

Ausführung von Programmen

- Das von einem Prozess ausgeführte Programm kann durch ein neues Programm ersetzt werden:

```
#include <unistd.h>

int execv(const char *path, char *argv[]);
int execl(const char *path, char *arg0, ...);
```

Beispiel:

```
...                 /* Process A */
argv[0] = "ls";
argv[1] = "-l";
argv[2] = NULL;
execv("/bin/ls", argv);
/* Should not be reached. */
```

$\Rightarrow$

```
...                 /* Process A */
int
main(int argc, char *argv[])
{
    ...
}
```

Das zuvor laufende Programm wird beendet, das neue gestartet.

Es wird nur das Programm ausgetauscht.

Der Prozess läuft weiter!

Start eines Programms

- Beispiel: Start des Programms `./prog` mit Parametern `-a` und `-b`

... als Vordergrund-Prozess:

```
pid_t pid;

pid = fork();
switch (pid) {
case -1: /* Error */
    perror("fork");
    exit(EXIT_FAILURE);

case 0: /* Child */
    execl("./prog", "prog",
          "-a", "-b", NULL);
    perror("./prog");
    exit(EXIT_FAILURE);

default: /* Parent */
    waitpid(pid, NULL, 0);
    break;
}
```

... als Hintergrund-Prozess:

```
pid_t pid;

pid = fork();
switch (pid) {
case -1: /* Error */
    perror("fork");
    exit(EXIT_FAILURE);

case 0: /* Child */
    execl("./prog", "prog",
          "-a", "-b", NULL);
    perror("./prog");
    exit(EXIT_FAILURE);

default: /* Parent */
    /* No "waitpid" here! */
    break;
}
```