System-Level Programming

28 Programs and Processes – UNIX

J. Kleinöder, D. Lohmann, V. Sieh, P. Wägemann

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Summer Term 2024

http://sys.cs.fau.de/lehre/ss24

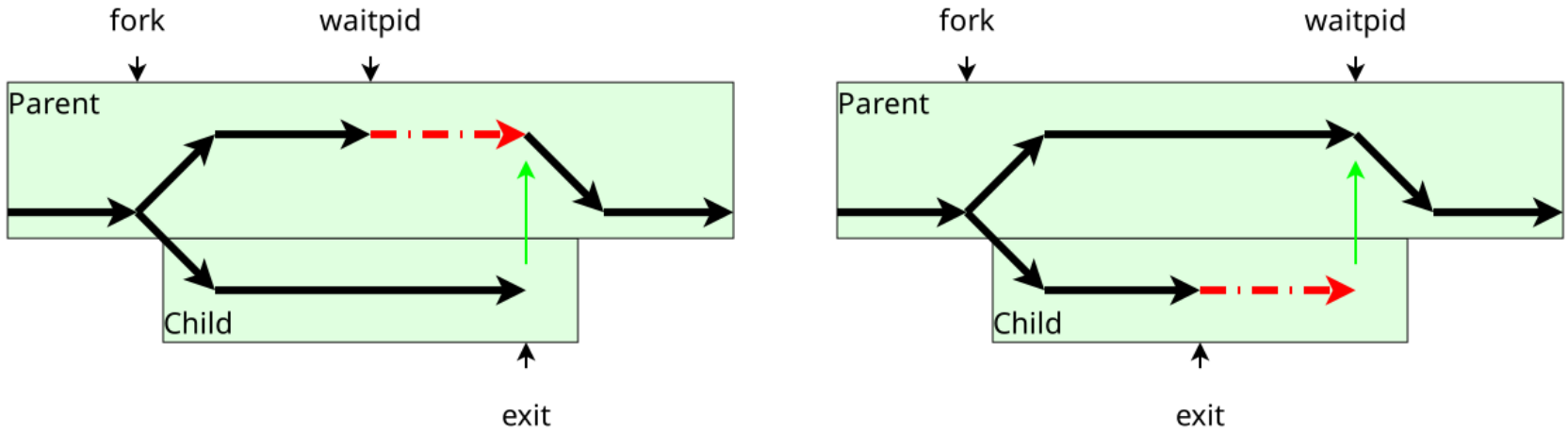
Example: UNIX

- Overview:

fork: creates new processes

exit: terminates a process

waitpid: waits for the termination of a process



Programming Interface

- Duplicating the currently running process

```
#include <unistd.h>

pid_t fork(void);
```

- Termination of the currently running process

```
#include <stdlib.h>

void exit(int status);
```

- Waiting for the termination of a different process

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

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


Example

```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;
}
```

Process Creation

- The child process is a copy of the parent process
  - same program
  - same data (contents of variables)
  - same program counter
  - same current and root directories
  - same opened files
- Only difference
  - different process IDs
  - returned value of `fork`

Execution of Programs

- The program that is executed by a process can be replaced by an other program:

```
#include <unistd.h>

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

Example:

```
...                 /* 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[])
{
    ...
}
```

The previously running program is terminated, the new one is started.

Only the program is replaced.

It is still the same process that is running!

Starting a Program

- Example: Starting of the program `./prog` with parameters `-a` and `-b`

... as a foreground process:

```
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;
}
```

... as a background process:

```
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;
}
```