NTNU | Norwegian University of Science and Technology

Operating Systems

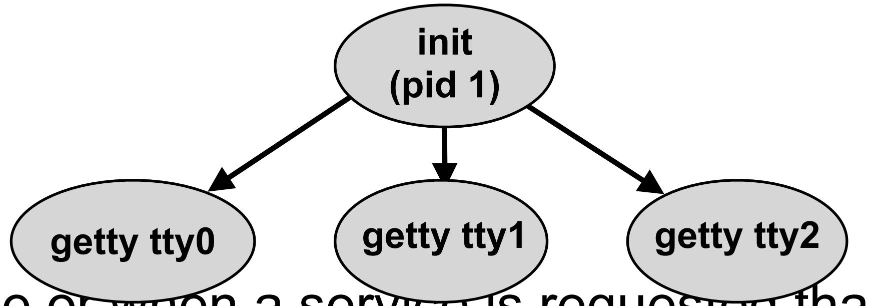
Discussion of TE2 – 11.02.2021

Michael Engel

2.1 Unix processes

a. Init is the process at the top of the Unix process hierarchy. Describe the two cases in which a process is a child process of init.

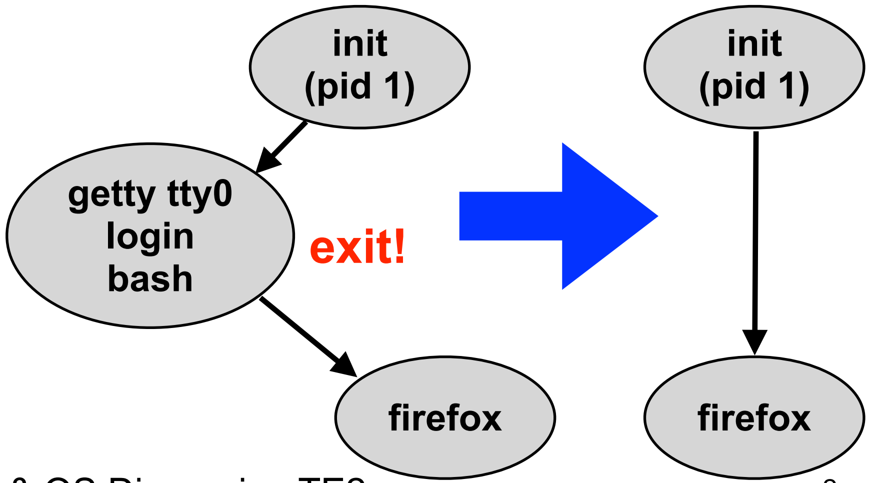
Case 1: a process is *directly* created by the **init** process using fork



This happens at system startup time or when a service is requested that is defined in `/etc/inittab` (e.g. a newly attached terminal)

Case 2: a process "loses" its parent (parent terminates before child using exit/_exit or a crash/signal)

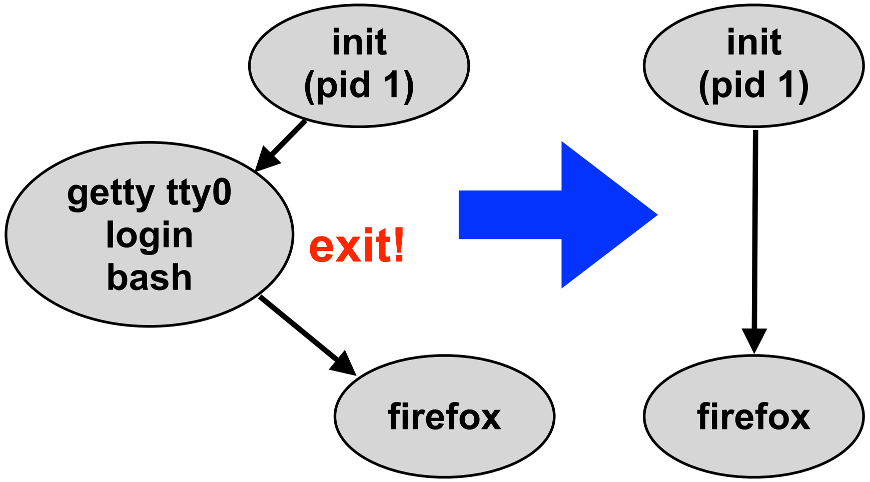
In this case, **init** takes over the role of the parent

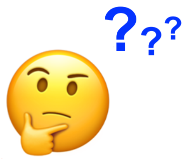


2.1 Unix processes

Case 2: a process "loses" its parent (parent terminates before child using exit/_exit or a crash/signal)

In this case, **init** takes over the role of the parent





What happens if init exits?

An older Linux `kill(2)` manpage stated:

"The only signals that can be sent to process ID 1, the init process, are those for which init has explicitly installed signal handlers. This is done to assure the system is not brought down accidentally"

Accordingly, killing init would shut down the system. This is usually prohibited.

2.1 Unix processes

b. Describe the function of `execvp` in your own words
   (Hint: read the man page)

There is a whole family of exec… functions which are part of the C library (libc). All ultimately use the system call execve(2) to ~~create a new process~~ *execute a different process in the current context*, but are more convenient to use.

execvp(3) allows to pass command line arguments as parameter:

```
int execvp(const char *file, char *const argv[]);
```

The executable file is searched for in the path specified in the environment by the `$PATH` variable. If this is not set, a default search path is assumed (which one depends on your system).

2.1 Unix processes

b. Describe the function of `execvp` in your own words
(Hint: read the man page)

Example for the use of execvp(3):

The argument list array has the **program name as element 0** and the command line parameters as separate strings starting from element 1, terminated by a NULL pointer

```
// Define NULL terminated array of char* strings
char* argument_list[] = {"ls", "-l", NULL};

// Now execute the command "ls -l" (ls is searched in $PATH)
execvp("ls", argument_list);
```

2.1 Unix processes

c. Explain the output of the following command in your own words.
(Hint: assume the current directory has at least one pdf file)

```
ls | grep -c .pdf
```

The `ls` command lists the contents of the current directory, one line per filename.

Grep filters its input according to the given pattern.
From the grep(1) manpage:
"-c, --count
    Only a count of selected lines is written to standard output."

Thus, the command prints the number of pdf files in the current directory

2.1 Unix processes

c. Which data is transferred through the pipeline and what operation does grep perform here?

```
ls | grep -c .pdf
```

The output of `ls` is passed through the pipeline as text, one filename per line, separated by a newline (0x0a = '\n') character

The `grep` command then first ***filters*** its input and removes all the lines ***not*** matching the search pattern "`.pdf`".

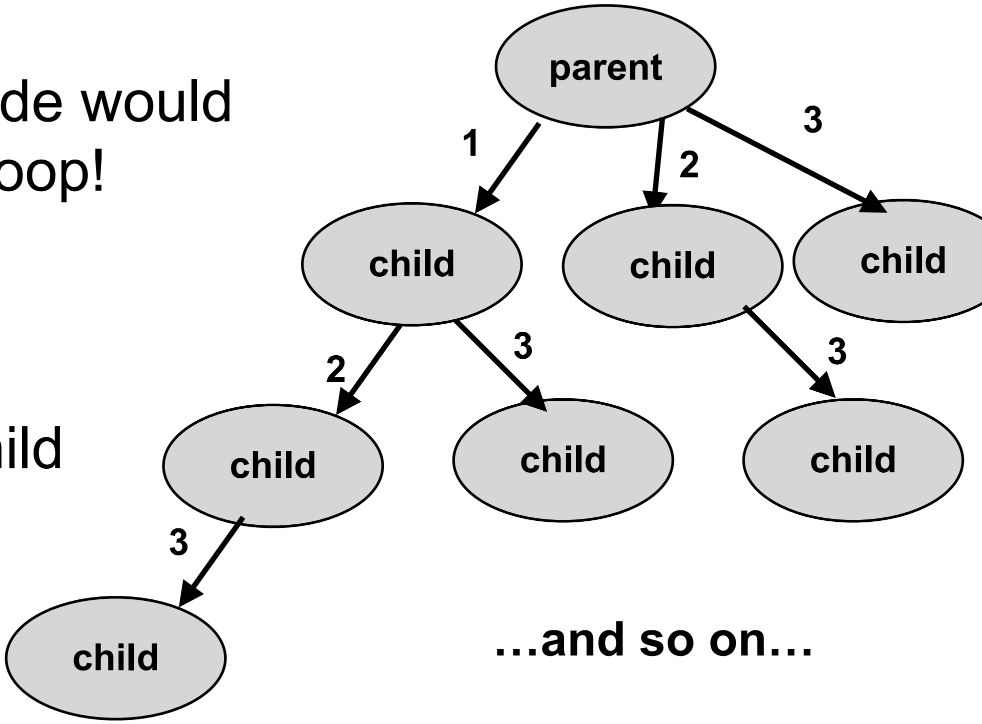
Following, the `-c` option `grep` tells the command to only print the ***number*** of matching lines ➛ number of pdf files

2.2 fork

Consider the following line of C code: (Caution: Do not try to execute this!): `for (;;) fork();`

a. Describe the program behavior after 1, 2, 3 and n iterations of the for loop. Assume that all processes (especially the for loop) are executed in parallel

   The program executing this code would execute fork(2) in an endless loop!
   After iteration 1:
   1 parent, 1 child process
   After iteration 2:
   1 parent, 2 children, 1 grandchild
   After iteration 3:
   1 parent, 3 children, …



2.2 fork

Consider the following line of C code:
(Caution: Do not try to execute this!): `for (;;) fork();`

b. The behavior of a program like this can lead to problems. Describe the problems that can occur.

   With each iteration of the loop, the number of newly created processes increases. This does not use more memory for the processes itself (due to ***copy-on-write***), but for their page tables (since each process has its own) and the process table entries.

   Eventually, the OS will run out of resources and will possibly crash.

2.2 fork

Consider the following line of C code:
(Caution: Do not try to execute this!): `for (;;) fork();`

- Try to find a way to avoid the problem in Unix (without changing the program above).

  The maximum number of processes in a system can be quite large. On Linux, you can find this using

  ```
  $ cat /proc/sys/kernel/pid_max
  4194304
  ```

  In most systems, there is no (or a large) limit for the number of processes a user is allowed to create. You can check and set the current limit using the `ulimit(1)` command (either with parameter `-a` for "all" or only `-u` to only print the number of processes):

  ```
  $ ulimit -a
  …
  max user processes                 (-u) 254718
  ```