ITF22519 - Innføring i Operativsystemer

Lab 5

Exercise 1.

```
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$ ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000    5649    5631  0  80   0 -  6269 do_wai pts/0    00:00:02 bash
0 S  1000    6251    5649  0  80   0 - 223295 do_pol pts/0   00:00:34 evince
0 S  1000   16291    5649  0  80   0 -   693 hrtime pts/0    00:00:00 print_pid
0 S  1000   16292    5649  0  80   0 -   693 hrtime pts/0    00:00:00 print_pid
0 R  1000   16293    5649  0  80   0 -  6050 -      pts/0    00:00:00 ps
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$ kill 16291
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$ kill 16292
[2]-  Terminated              ./print_pid
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$ ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000    5649    5631  0  80   0 -  6269 do_wai pts/0    00:00:02 bash
0 S  1000    6251    5649  0  80   0 - 223295 do_pol pts/0   00:00:34 evince
0 R  1000   16310    5649  0  80   0 -  6050 -      pts/0    00:00:00 ps
[3]   Terminated              ./print_pid
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$
```

Figure 1: Running two instances of print_pid, then killing them.

Process name = CMD
Process state = S
Process ID = PID
Process ID of parent = PPID

The process which spawned my print_pid processes is the bash process with PID = 5649. We can see this by looking at the PPID column of the print_pid processes.

When we kill the print_pid processes they stop running and are no longer listed in the process table.

I have used ps -l instead of ps -el to avoid printing irrelevant lines.

```
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
jrlundqv   14478   14460  0 13:00 pts/1    00:00:00 bash
jrlundqv   17455   14478  0 13:47 pts/1    00:00:19 evince lab5.pdf
jrlundqv   21847   14478  0 15:53 pts/1    00:00:00 ./print_pid
jrlundqv   21848   14478  0 15:53 pts/1    00:00:00 ./print_pid
jrlundqv   21850   14478  0 15:53 pts/1    00:00:00 ps -f
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$
```

Figure 2: ps -f to get STIME

Time being created = STIME

Exercise 2.

```
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$ ./fork_ex3
Process 17807's parent process ID is 5649
Process 17810's parent process ID is 17807
Process 17809's parent process ID is 17807
Process 17811's parent process ID is 17809
Process 17808's parent process ID is 17807
Process 17812's parent process ID is 17808
Process 17813's parent process ID is 17808
Process 17814's parent process ID is 17812
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$
```

Figure 3: Running fork_ex3

```
bash─┬─fork_ex3─┬─fork_ex3─┬─fork_ex3──fork_ex3
     │          │          └─fork_ex3
     │          ├─fork_ex3──fork_ex3
     │          └─fork_ex3
     └─pstree
```

Figure 4: Showing the process tree by the "pstree" command

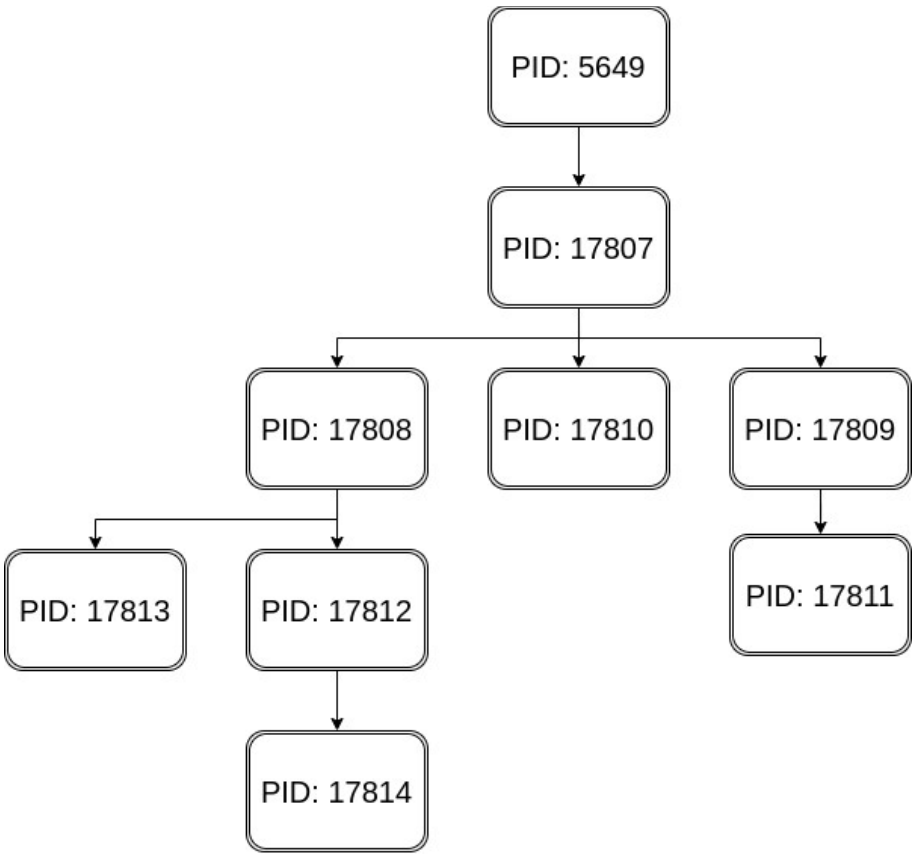

Figure 5: Process tree

PID 5649 is the bash shell, PIDs 17807-17814 are instances of the fork_ex3 program.

Calling fork() n amounts of times will result in 2^n processes. In the fork_ex3 example code we have called fork() three times resulting in $2^3 = 8$ processes. This is because after each call to fork(), a new process has been created which will also execute the next fork() call.

```
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$ ./fork_ex3
Process 19854's parent process ID is 14478
Process 19857's parent process ID is 19854
Process 19856's parent process ID is 1794
Process 19855's parent process ID is 1794
Process 19859's parent process ID is 19856
Process 19860's parent process ID is 19855
Process 19858's parent process ID is 19855
Process 19861's parent process ID is 19858
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$ ps -p 1794 -o comm=
systemd
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$
```

Figure 6: Running fork_ex3 without call to sleep()

By removing the sleep() statement, some of the parent processes will finish before their child processes. When this happens the child processes will be assigned to a new parent process. In this case systemd with PID 1794 has become the parent of two fork_ex3 processes.

Exercise 3.

```
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$ ./Ex3
a.out                  Ex3            execl.c     no_wait.c                    wait.c
Ex1.png                Ex3.c          fork_ex1.c  no_wait.txt                  waitpid
Ex1_STIME.png          Ex3_cat.png    fork_ex2.c  OS_Lab5_JonathanLundqvist.odt  waitpid.c
Ex2_no_sleep.png       Ex3.png        fork_ex3    print_pid                    waitpid.txt
Ex2.png                Ex3_ps_-l.png  fork_ex3.c  print_pid.c                  wait.txt
ex2_process_tree.png   Ex3_ps.png     lab5.pdf    task3.txt
Ex2_pstree.png         Ex3_pwd.png    no_wait     wait
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$
```

Figure 7: execl("bin/ls", "ls", NULL);

```
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$ ./Ex3
/home/jrlundqv/OneDrive/22høst/OS/labs/lab5
```

Figure 8: execl("bin/pwd", "pwd", NULL);

```
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$ ./Ex3
Hello Mr. Studass
Hello Mr. Studass
I hope you are having a good day, and that the retting is going greit :)
I hope you are having a good day, and that the retting is going greit :)
^C
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$
```

Figure 9: execl("bin/cat", "cat", NULL);

```
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$ ./Ex3
    PID TTY          TIME CMD
  14478 pts/1    00:00:00 bash
  17455 pts/1    00:00:21 evince
  22879 pts/1    00:00:00 ps
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$
```

Figure 10: execl("bin/ps", "ps", NULL);

```
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$ ./Ex3
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   14478   14460  0  80   0 -  5896 do_wai pts/1    00:00:00 bash
0 S  1000   17455   14478  0  80   0 - 233160 do_pol pts/1   00:00:21 evince
0 R  1000   22939   14478  0  80   0 -  6050 -      pts/1    00:00:00 ps
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab5$
```

Figure 11: execl("bin/ps", "ps", "-l", NULL);

```c
int main()
{
        execl("/bin/ps", "ps", "-U", "root", "-u", "root", "u", NULL);
        printf("This will only be printed if the execl call fails");
}
```

If the execl call is successful, the current process image will be replaced by what is passed as arguments into the execl function. That would result in the above program never reaching the printf() function.

If the execl call fails for some reason, the current process will continue executing, and our printf() function would be reached and called.