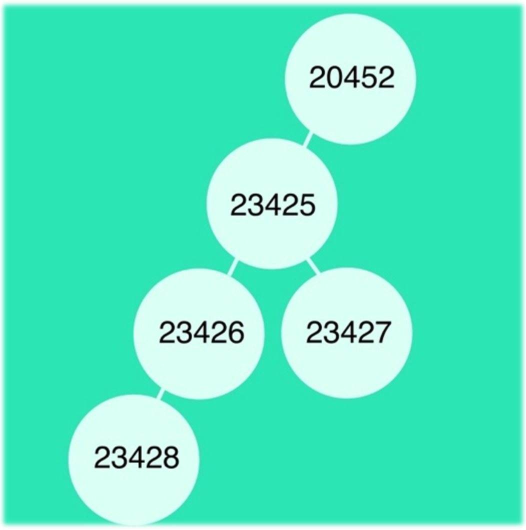
Lab 5 – steinhs (@github)

Exercise 3.1



Running fork_ex1:

```
Process 23425's parent process ID is 20452
Process 23426's parent process ID is 23425
Process 23428's parent process ID is 23426
Process 23427's parent process ID is 23425
```

The tree starts on top with process 20452 which is the main process and 23425 which is its child.

Calling the first fork() will create child process 23426 on parent 23425.

Calling the second fork() will create child 23427 on parent 23425 and child 23428 on parent 23426.

Removing sleep(2):

```
Process 29747's parent process ID is 29196
Process 29748's parent process ID is 29747
:~/Introduction20S/labs/lab5$ Process 29750's parent process ID is 1
Process 29749's parent process ID is 1
```

Running the program without sleep() will cause the parent process to finish before the child gets to ask for the parents PID. The children are then reassigned to a new parent PID which is most likely the initial process 1.

Exercise 3.2

```
$ cat output_no_wait.txt
```

```
I am the parent!
Child 7181 is done, status is 0
I am the child!
```

Parent is executed before child is complete, which prompts it to return status of 0 since it haven't been changed to 42 when the child is executed.

```
$ cat output_wait.txt
```

```
I am the child!
I am the parent!
Child 7172 is done, status is 42
```

The child is created and returns "I am the child!" message, then wait() is used to block the parent from executing before its child has completed, which takes 5 seconds. We can also tell that this has been done by the status which is changed to 42 in the program.

```
$ cat output_waitpid.txt
```

```
I am the child!
I am the parent!
Child 7157 is done, status is 42
```

Uses the same principle as wait() but using waitpid() we can specify which child the parent should wait for. The rest follows as mentioned in wait().

Exercise 3.3

I assume the program will keep looping until the usleep() statement get signaled by kill() which is after 10 seconds. The loop “usleeps” for 10000 microseconds (0,01 seconds), which adds up to counting and printing 1000 times.

```
$ ./kill
```

```
Parent sleeping
Child is counting at 1
Child is counting at 2
...
Child is counting at 986
Child is counting at 987
Child has been killed!!!!!!!!!!
Parent done!
```

The infinite loops stops because of the process running it gets killed off.

The counter almost hit 1000 but didn’t quite get there. From what I have read in the *man pages* is that it is either because of the small delay before *usleep* start to count, or because of system activities that delays it a bit.

Exercise 3.4

A zombie process is a process that has executed its task and gets a “zombie” status and signals the parent that it has finished or died. The parent is then supposed to execute wait() so that the child is removed from memory, but as long as its parent-process doesn’t call wait() the zombie process will stay in memory until removed or cleaned up.

Creating the program *zombie.c* which creates a child and put it to sleep for 20 seconds. While it sleeps I also kill the parent. While the program is sleeping, we can use *ps -l* to print out running processes and display its statuses. The first print shows before running the program, while the last print shows during the 20 seconds. The zombie process is marked as *zombie*.

```
$ ps –l
```

```
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  2219 14911 14910  0  80   0 -  2737 -      pts/14   00:00:00 bash
0 R  2219 22159 14911  0  80   0 -  3346 -      pts/14   00:00:00 ps
```

```
$ ./zombie
$ ps –l
```

```
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  2219 14911 14910  0  80   0 -  2737 -      pts/14   00:00:00 bash
1 S  2219 22167     1  0  80   0 -   536 hrtime pts/14   00:00:00 zombie

0 R  2219 22168 14911  0  80   0 -  3346 -      pts/14   00:00:00 ps
```