Operating Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder

Module: CO-562
Date: 2022-09-15
Due: 2022-09-22

OS 2022 Problem Sheet #2

Problem 2.1: *process creation using `fork()`* (1+1 = 2 points)

Consider the following C program. Assume that all system calls succeed at runtime, that no other processes are created during the execution of the program, and that process identifiers are allocated sequentially.

```
1  #include <stdio.h>
2  #include <unistd.h>
3
4  static int x = 0;
5
6  int main(int argc, char *argv[])
7  {
8      pid_t p = getpid();
9
10     x++;
11     fork();
12     if (! fork()) {
13         if (fork()) {
14             x++;
15         }
16         x++;
17     }
18
19     printf("p%d: x = %d\n", getpid() - p, x);
20     return 0;
21 }
```

Try to solve this question on paper and not by typing the code into your computer. During an exam, you will have to answer questions like this on paper as well.

a) How many processes does the program create during its execution. Draw the process tree and indicate the value of x on the edges whenever it changes in a process.

b) What is the output produced by the program?

Problem 2.2: *xargs - execute a programs with constructed argument lists* (8 points)

Write a C program called `xargs`, which is a simplified version of the Unix xargs utility. Your program reads lines from the standard input and constructs argument lists for a command to be executed. If more lines are available than can fit on the argument list, then additional commands are executed with the arguments that did not fit on the first command line. Your program continues constructing argument lists and executing commands until the end of the standard input has been reached. The command to execute is specified as part of the `xargs` arguments or if none are provided, then `/bin/echo` is used. Your program should support the following options:

`-n` the (maximum) number of input lines added to the constructed argument lists

`-t` show the argument list (on stderr) before the command is executed

If you want to challenge yourself, then you implement the following additional option:

`-j` the (maximum) number of processes (jobs) executed concurrently (default is 1, which means processes are executed sequentially)

By implementing this option correctly, you may earn up to two bonus points.

Here are some example invocations:

```
$ echo "hello world" | xargs
hello world
$ seq 0 10 | xargs
0 1 2 3 4 5 6 7 8 9 10
$ seq 0 10 | xargs -t
/bin/echo 0 1 2 3 4 5 6 7 8 9 10
0 1 2 3 4 5 6 7 8 9 10
$ seq 0 10 | xargs -n 3
0 1 2
3 4 5
6 7 8
9 10
$ seq 0 10 | xargs -n 3 -t
/bin/echo 0 1 2
0 1 2
/bin/echo 3 4 5
3 4 5
/bin/echo 6 7 8
6 7 8
/bin/echo 9 10
9 10
$ seq 1 4 | xargs -t -n 1 printf "foo-%02d\n"
printf foo-%02d\n 1
foo-01
printf foo-%02d\n 2
foo-02
printf foo-%02d\n 3
foo-03
printf foo-%02d\n 4
foo-04
```

Make sure your program properly handles all possible runtime errors and that it returns an error status to its parent process (usually the shell) in case a runtime error occurred.

Your program must use the `fork()`, `execvp()`, and `waitpid()` system calls. Use the `getopt()` function of the C library for the command line option parsing. You may want to use the `getline()` library function for reading the lines from standard input.