Introduction to Computer Science
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: CH08-320101
Date: 2018-11-22
Due: 2018-11-29

ICS 2018 Problem Sheet #10

Problem 10.1: *fork system call* (1+3 = 4 points)

Consider the following C (C++) program (let me call the source file `foo.c`).

```
#include <unistd.h>

int main(int argc, char *argv[])
{
    for (; argc > 1; argc--) {
        if (0 == fork()) {
            (void) fork();
        }
    }
    return 0;
}
```

a) Assume the program has been compiled into `foo` and that all system calls succeed at runtime. How many child processes are created for the following invocations of the program? Explain

   - `./foo`
   - `./foo a`
   - `./foo a b`
   - `./foo a b c`
   - `./foo a b c d`

b) A process returns a number to its parent when it exits and a process stays around until this has happened. If the parent process is not interested in picking up the numbers from its child processes, then the child processes stay around as "zombies". Write a C program that creates for every command line argument one zombie process. Then use utilities like `top` or `ps` to find the zombies in the process list and document that you managed to create zombies.

Problem 10.2: *recursive directory tree walk* (6 points)

Write a C (or C++) programm, lets call it `lsr`, that displays (recursively) the files in a directory tree. Your program should use the `opendir()`, `closedir()` and `readdir()` C library functions. Make sure you handle all relevant runtime errors.

To learn how to use these functions, you can simply read the manual pages. (Type `man opendir` into a terminal to obtain and read the manual page for `opendir()`).

Your program should recursively show the content of the current working directory if no arguments are passed to the `main()` function of your program. Otherwise, start a recursive listing for each of the arguments that are passed to the `main()` function.

A sample execution is shown below. We first create a simple directory tree:

```
$ mkdir -p a/b/c
$ touch a/x a/y a/b/z
```

Then we can run `lsr` as follows:

```
$ ./lsr a/b
a/b/z
a/b/c
$ ./lsr a
a/x
a/b
a/b/z
a/b/c
a/y
```