Operating Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: CO20-320202
Date: 2016-09-13
Deadline: 2016-09-20

Problem Sheet #1

Problem 1.1: *processes* (2+2 = 4 points)

a) Briefly explain (i) what happens when the parent process of a child process terminates before the child process terminates and (ii) what happens if a child process terminates and the parent process did not yet call one of the `wait()` system calls?

b) When a process waits for a child process using one of the `wait()` system calls, it may obtain a status code. Read the documentation of the `wait()` system calls. What does the status code contain?

Problem 1.2: *watch - execute a program periodically* (6 points)

Write a C program called `watch` that executes a command periodically (e.g., every 2 seconds), showing the output on the standard output (terminal). Your implementation of `watch` does not have to clear the screen like other implementations of `watch` do. Your program should implement a command line option `-n` to set the number of seconds that `watch` sleeps between each repeated execution of the command. The option `-b` causes the special value
`a` to be written to the standard output if an execution of the command ends with a non-zero exit code (this usually rings the terminal bell). The option `-e` terminates your `watch` program when the execution of a command fails. (If `-e` is not given on the command line, the execution continues irrespective of any failures of the command execution.)

Your program must use the `fork()`, `execvp()`, and `waitpid()` system calls. You are not allowed to use the `system()` library call. You can let your `watch` program sleep by calling the `sleep()` library function.

```
$ ./watch date
Tue Sep 13 13:51:33 CEST 2016
Tue Sep 13 13:51:35 CEST 2016
Tue Sep 13 13:51:37 CEST 2016
Tue Sep 13 13:51:39 CEST 2016
```

```
$ ./watch -e ls -l /foo
ls: /foo: No such file or directory
$
```

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 occured.