NWI-IBC019: Operating system concepts / Exercises: week 2 / 2022 v0

NOTE *These weekly exercises are individual (unless marked otherwise).*

READING **10TH EDITION** *chapter 3 up to and incl. 3.6 + 3.7.4 + 3.8; chapter 4 up to and incl. 4.3 + 4.5 + 4.6.*

HAND IN *Please hand in the exercises in PDF format in Brightspace.*

Exercise 1

Using the program in below, identify the values of pid at lines A, B, C, and D. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)

```c
int main() {
  pid_t pid, pid1;
  /* fork a child process */ pid = fork();
  if (pid < 0) { /* error occurred */
    fprintf(stderr, "Fork Failed");
    return 1;
  }
  else if (pid == 0) { /* child process */
    pid1 = getpid();
    printf("child: pid = %d",pid); /* A */
    printf("child: pid1 = %d",pid1); /* B */
  }
  else { /* parent process */
    pid1 = getpid();
    printf("parent: pid = %d",pid); /* C */
    printf("parent: pid1 = %d",pid1); /* D */
    wait(NULL);
  }
  return 0;
}
```

Exercise 2

What are the benefits and the disadvantages of each of the following? Consider both the system level and the programmer level.

- Synchronous and asynchronous communication
- Automatic and explicit buffering
- Send by copy and send by reference
- Fixed-sized and variable-sized messages

Exercise 3

In chapter 3, modern browsers are discussed and its practice of opening each new website in a separate process. Would the same benefits have been achieved if instead the browser had been designed to open each new website in a separate thread? Explain.

Exercise 4

A process without threads has a turnaround time of 10 seconds (time required to go from the 'new' state to the 'terminated' state). Two seconds are spent in the running state, two in the ready state, and six seconds are spent in the blocked state due to disk I/O. Somebody suggests to decrease the turnaround time by using threads in the process.

- Will this make a difference on a computer with only one execution unit (core/processor)? If yes, what is the most time that can be saved (best case scenario)?
- Will this make a difference on a computer with four execution units (core/processor)? If yes, what is the most time that can be saved (best case scenario)?

Exercise 5

What is the output of the C++ program below? Give reason (for example, you could draw a figure which clarifies creating the process and giving the ouput on the screen)

```
int main() {
    int i = 0;
    i = i + 1;
    if (fork() == 0) {
        i = i + 1;
        if (fork() == 0) {
            i = i + 1;
        }
    }
    i = i + 1;
    std::cout << "i is " << i << std::endl;
    return 0;
}
```

```
// framework for "date | tail -c 10"
int shell(bool showPrompt) {
  // make a communication channel between the two processes
  pid_t child1 = fork();
  if (child1 == 0) {
    // guarantee that the standard output (stdout) is redirected to the shared communication channel
    // don't clean the used resources
    Command cmd = {{string("date")}};
    executeCommand(cmd);
    abort(); // if the executable isn't found, the child process must be aborted
  }
  pid_t child2 = fork();
  if (child2 == 0) {
    // guarantee that the standard input (stdint) reads from the shared channel
    // don't clean the used resources
    Command cmd = {{string("tail"), string("-c"), string("5")}};
    executeCommand(cmd);
    abort(); // if the executable isn't found, the child process must be aborted
  }
  // don't clean the used resources
  // wait for the child processes
  waitpid(child1, nullptr, 0);
  waitpid(child2, nullptr, 0);
  return 0;
}
```

listing 1 Framework for letting two processes communicate.

Extra: feedback opportunity for assignment 1

HAND IN *For this, you can work with your assignment pair.* ***Only one of the two should hand this in.*** *This extra bit is not graded.*

To get an understanding on how system calls and file descriptors function, you can try a small proof of concept described by the function in listing 1. This function executes the command `date | tail -c 5` and if `date` returns for example `Wed Sep 2 10:42:37 CET 2021` (followed by a newline), the output of the command described below is `2021` (followed by a newline). The main aspect is executing two commands where the input of the second command is the output of the first command, by using a shared communication channel. We will use `pipe()` for this. By getting the function in listing 1 to work, you get a good idea which system calls will be executed by your shell.

We will work in the same project as the first assignment, as this is actually part of the assignment. So please read the first assignment. The function is given in the function `step1()` in the file `shell.cpp` (be sure to edit the function `shell()` in the same file to start), and you can edit this file and execute the desired command.