Operating Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder

Module: CO-562
Date: 2021-09-23
Due: 2021-09-30

OS 2021 Problem Sheet #3

Problem 3.1: *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 3.2: *readers / writers problem* (1+1+1 = 3 points)

Below are three incorrect solutions of the readers-writers problem. Explain in which situations the solutions fail to work correctly. The solutions use the following common definitions:

```
shared object data;
shared int readcount = 0;
semaphore mutex = 1, writer = 1;
```

a)
```
void reader()
{
    down(&mutex);
    readcount = readcount + 1;
    if (readcount == 1) down(&writer);
    up(&mutex);
```

```
void writer()
{
    down(&writer);
    write_shared_object(&data);
    up(&writer);
}
```

```
        read_shared_object(&data);
        down(&mutex);
        readcount = readcount - 1;
        up(&mutex);
        if (readcount == 0) up(&writer);
    }
```

b)
```
void reader()
{
    down(&mutex);
    readcount = readcount + 1;
    if (readcount == 1) down(&writer);
    up(&mutex);
    read_shared_object(&data);
    down(&mutex);
    readcount = readcount - 1;
    if (readcount == 0) {
        up(&mutex);
        up(&writer);
    } else {
        up(&mutex);
    }
}
```

```
void writer()
{
    down(&writer);
    write_shared_object(&data);
    up(&writer);
}
```

c)
```
void reader()
{
    down(&mutex);
    readcount = readcount + 1;
    if (readcount == 1) down(&writer);
    up(&mutex);
    read_shared_object(&data);
    down(&mutex);
    readcount = readcount - 1;
    if (readcount == 0) up(&writer);
    up(&mutex);
}
```

```
void writer()
{
    down(&writer);
    down(&mutex);
    write_shared_object(&data);
    up(&mutex);
    up(&writer);
}
```

Problem 3.3: *running competition* (5 points)

A sports club is organizing a running competition. The club has a total of N electronic chips that are used to measure the performance of the runners between the start and the finish line. Unfortunately, the number of runners registered for the running event by far exceeds the number of chips owned by the club. As such, only some runners can start together while the others have to wait for other runners to finish before they can pick up a chip.

The competition also supports relay running teams. An arriving relay runner joins the next possible team. Once a team of T relay runners is complete, the team leader (e.g., the last member of the team) picks up T chips and then the relay runners proceed together to the start.

Write a solution (in pseudo code like on the course slides) for this problem using semaphores. Individual runners execute the function `runner()` to obtain a chip, run, and return the chip. Relay runners execute the function `relay()` to join a team of relay runners. Once a team is complete, the team leader obtains T chips and then the team members proceed to run. Every relay runner returns his chip independently after finishing.