In this exercise we familiarize ourselves to signals.

Exercise 7a (Signals, 1p)

Phase 1. (The child process does not inherit the alarm)

Show yourself first that the alarm is not inherited by the child process. You can do this by

1. Installing signal handler before the fork
2. Request alarm signal before the fork (to be delivered after 5 sec)
3. Observe that the signal is delivered only to the parent.

Phase 2. (The new program receives the alarm)

Show yourself that the alarm is inherited by the new program created by exec. You can do this by

1. Requesting alarm in the child (to be delivered after 5 sec)
2. Starting a new program in the child (exec)
3. Installing signal handler in the new program. Signal handler only displays "Signal received".
4. Observe that the signal is delivered to the new program in the child.

Exercise 7b (Extra, Child processes run their own program files and signals are used to inform the parent, 0.25p)

Use the file excer7B_starting_point.c as a starting point. It helps you to understand the problem description.

Phase 1

Write a program that creates five children. All children start to run a separate program file, which is the same for all children. The parent process waits for the children. Waiting is now based on the signal SIGCHLD so that signal handler increments the global variable n that indicates how many children have terminated. The parent remains in the waiting loop as long as the n is less 5. There is a pause in the loop to save processor power. Each time pause returns (signal is delivered, child has terminated), the parent displays a message “Child terminated x”, where x is the current value on n. When all children are terminated, the parent comes out from the loop and displays the contents of the file that children have written. The parent itself terminates after that.

Remark. Use for loop to fork five children (instead of repeating the code).

The children write n times the string xxxx to the file. The number of writes n and the string xxxx can be different for each child, because they are passed as parameters for each child. In the parent program use string array to store strings. Initialize the string array with the strings “AAAA”, “BBBB”, etc. The file descriptor is also passed as a parameter.

Parent process opens the file before the fork using system call open, so that children inherit the opened file descriptor and can use it without opening. It is necessary to pass the file descriptor number to the new program. This is done using command line parameters. Do not use library functions for file operations, please!

In the test run use the value 5 for parameter n so that the first child writes text "AAAA" 5 times to the file. The second child writes text lines "BBBB" 5 times to the file, and so on.

Run the program as many times that you see two different problems that can happen. What are they? What are the reasons for these problems?

Phase 2.

The first problem is that at least sometimes, the parent process does not come out from the waiting loop. Fix this problem. You are not expected to correct another problem yet, but you need to explain what it is and what the reason for it is.