In this exercise we familiarize ourselves to process creating and control in GNU/Linux OS.

Excercise 5a (Process creation, 1,0p)

Part A (Fan of processes)

Write a program that first finds out how many processes can be created. The program displays the maximum number of processes.

Next the program creates 5 child processes. The first child displays character A and waits for one second after displaying the character. This is repeated 5 times. The second child displays character B and wait for one second and repeats this 5 times. There is no other output from the child processes (not even newlines) so that all characters are at the same line. Make sure that characters go immediately to the screen so that they are not left in the buffer.

The parent process waits for all children. Always when a child has terminated the parent display the message "A child xxxx has terminated", where xxxx is the process id of the child process. When all children have terminated, the parent terminates.

Run the program a few times and examine the output. Find the differences in the output.

Part B (Two processes)

Phase 1 (Child process and exit status)

You have already solved the "real-time problem" in chat program in two different ways. The first method was polling and the second was dividing the tasks in separate processes. In the latter solution you needed to start the chat application by entering two program names from the command line. This is not an acceptable way if we had several inputs to respond meaning several separate programs to start.

This problem can now be solved so that we write one program and that program creates an another process (or more if we had more input to respond to).

Write once again the chat application using this method. The original process (the parent process) creates a child process that takes care of reading the input from the chat fellow descriptor. The parent process itself reads input from the keyboard. The application still needs to work in real-time, so that it gives immediate answer to each input.

Add the following new features to the application:

1. The child process, in addition to sending characters from the chat fellow directly to the screen, writes them also to the file. When end of file has been received from the chat fellow, the child process terminates.

2. When the user enters Q or q from the keyboard the parent process stops reading the keyboard and starts to wait for the child. When the child process has terminated, the parent process displays the contents from the file (written by the child). The output is formatted as lines each containing 7 characters. See remark 1 below.

Remark 1. Use system calls in the file handling and do the output of the file in generic way so that there is no in advance assumption about the size of the file. The output always consists of lines containing 7 characters. The last line of course can have less than 7 characters.

Remark 2. Remember that the child process inherits the opened file descriptors. For this reason, open the file only once in the parent process for writing and reading. You need to open it before the fork Remember also that the file table entry (and thus the file offset) is common for the parent and for the child.

Only the child process uses the chat fellow descriptor. So open it only in the child process.

Phase 2 (Zombie process)

Run the program so that you do not enter q or Q and wait until the chat fellow has sent z character. When z is sent it also means that the child process has terminated but the parent process has not yet called wait. In this situation the child process is a zombie process. You can see this by opening another terminal and by entering the command `ps -a`. Zombie process is denoted by <defunc>.

After that, enter ‘q’ or ‘Q’ to your application. It means that the parent process proceeds and calls wait and terminates. After that you cannot anymore see the child process even as a zombie.

Extra exercise 5b (Return status of the process, 0,25p)

Modify the previous chat program in such a way that the child process also returns the number of characters it has written to the file. The parent process uses this value so that it only displays the value to the screen. This return value is not used to make conclusions about the size of the file.