In this exercise we familiarize ourselves to interprocess communication (IPC).

Exercise 9a (Pipe, 1p)

A process first creates a child process. The child process sends digits 0, 1, 2, 3,..., 9 in one second intervals to the parent via a pipe. When all digits have been sent, the child closes the write end of the pipe and exits.

Parent process reads the pipe and always immediately displays the received digit to the screen. The parent process terminates when it has read all characters from the pipe (the parent does not know in advance how many characters the child process sends). The end can be recognized because reading from an empty pipe returns 0 (end of file), if the pipe is not anymore open for writing (this is the reason we close the pipe in the child when all characters are written).

Exercise 9b (Extra exercise, Pipe, 0.25p)

Note: This is a tricky one, therefore think carefully before you write the code (e.g. draw a communication diagram etc.).

Write a program that consists of six processes. The program that is started from the command line is running as a main process of the system. This process then creates five child processes. The child processes are "sisters", because they all have the same parent process. Now you need to construct a pipeline from sister to sister. The child that is created last (the fifth child) is the beginning of the pipeline. The fifth child sends information to the fourth child. The fourth child sends information to the third child and so on. The first child has the end of the pipe line. When the processes have written the information to the pipe, they close the write end of the pipe and terminate.

Children that read the pipe (1st, 2nd, 3rd and 4th child) read the pipe until end of file is received.

Now we only carry a single character through the pipeline (from the last child to the first child). The last (youngest) child process sends a character A to its older sister (and closes the write end of the pipe and terminates). The sister that received the character increments the code of the character by one and forwards it to its big sister. The same procedure continues until the first child has received the character. The first (oldest) child process only displays the character that it has received from the second child process.

Remark 1. You have to use pipe to pass information between child processes.

Remark 2. When the parent process has created all children, it only waits until all children have terminated.

Remark 3. To make the program behavior easier to follow, put a delay of half a second between creations of children.[1]

Exercise 9c (Extra exercise, FIFO, 0.25p)

Write two separate programs that communicate with each other using FIFO. Both programs are started from the command line. The reader program is started first on the background. Then the writer program is started.

The writer program sends continuously the digits 0, 1, 2, 3, ..., 9, 0, 1, 2,… to the reader through the FIFO. It sleeps one second after sending each digit.

The reader process creates the FIFO. Then it reads the FIFO and displays immediately the digits it has received. The termination is done so that the reader program terminates after reading digit 6. The termination of the writer process is based on the fact that it detects that reader process has closed its read end. This means that the writer process terminates smoothly and displays the message "Need to terminate because no reader anymore"

Remark. Observe also how open blocks in the writer if there is no reader yet. You can do this by letting the reader to sleep 5 seconds after creating the FIFO and before opening it. Use suitable outputs to see the blocking of open function call.

[1] This can be done using the `nanosleep`:

```
struct timespec nano_sleep;
nano_sleep.tv_sec = 0;
nano_sleep.tv_nsec = 500000000;
nanosleep(&nano_sleep, NULL);
```