Operating Systems

Homework Assignment #3

Due 14.4.2016, 23:59

Part 1

In this part of the assignment, you will implement inter-process communication using FIFO files.

Write a program **`writer_pipe.o`**. It will accept one command-line parameter: path to a FIFO file (e.g., `/home/moshe/pipefile`). The program will:

- Open the input FIFO file.
  - If it doesn't exist, create it (`man 2 mkfifo`) and open it for writing.
  - If it exists, verify it is a FIFO file (`man 2 stat`) – using the `stat` function **only**!
    If it isn't, remove it (`man 2 unlink`), then create it as in the previous step.
    *You may assume it is not a directory and exit (with error) if **`unlink`** fails.*
- Write user input to the FIFO file.
  - Read the standard input for input lines from the user (you may use `fgets`).
  - Write each line to the FIFO file – using system calls **only**.
- When finished (e.g., EOF from the console), remove the FIFO file and **<u>only then</u>** close it, then terminate.

Write a program **`reader_pipe.o`**. It will accept one command-line parameter: path to a FIFO file (e.g., `/home/moshe/pipefile`). The program will:

- Open the input FIFO file.
  - If it exists, **before opening it** verify it is a FIFO file (`man 2 stat`) – using the `stat` function **only**!
    If it isn't, exit with a proper error message.
  - If it doesn't exist, sleep for 1 second and re-check (i.e., polling), repeatedly.
    (Note that it might be deleted between the *stat* and *open* calls, you should handle it)
  - *Exit with a proper error message on any <u>other</u> error opening the file.*
- Write input from the FIFO file to the console.
  - Read data from the FIFO file – using system calls **only**.
  - Write the data as text to the console (standard output) – you may use `printf`.
- When finished (e.g., EOF from the file), close the FIFO file and return to the first bullet.
  - The reader program constantly waits for a writer by opening the FIFO file, reading its content until EOF, then closing it and starting all over again.
  - The reader program terminates by pressing CTRL+C, when not "connected" to a writer.

Check your code by running these 2 programs together.

Open 2 terminals, one for the reader and one for the writer. Execute various scenarios and make sure the programs work together properly in all of the scenarios. Both programs should be able to start and stop repeatedly and still each should continue to function properly.

Several issues with the executions should arise, which we will fix using *signal handlers*:

1. To register signal handlers in UNIX, use the **`sigaction`** system call (`man 2 sigaction`).
   - Read about sigaction and how to properly use it. Do not assume its usage, look for examples!
   - Read about `SIGINT`, `SIGTERM`, and `SIGPIPE`.
2. The writer program can terminate without exiting properly
   - Register a **single** signal handler for both signals `SIGTERM` and `SIGINT`.
   - Modify your code to terminate properly – removing <u>and then</u> closing the FIFO file.
   - You should `unlink` and `close` the FIFO file.
3. The writer program fails when writing after the reader program was opened & closed.
   - While the writer is running, if a reader is started (thus opening the input file) and then terminated, the writer holds an open `FIFO` file with no reader – and writing to it causes a failure.
   - Register a signal handler for this possible pipe error.
   - When writing to a closed pipe, output a proper error message and continue regularly.
     To allow the user to continue writing, you should close & re-open the file!
   - The writer should now be able to handle the error without fault, and continue.
4. The reader program can terminate while still "connected" (reading from an open FIFO file)
   - After opening the FIFO file – use `sigaction` to ignore the two termination signals (`SIGTERM`, `SIGINT`) – <u>**without** a signal handler!</u>
   - After closing the FIFO file (after reading EOF) – use `sigaction` to <u>**restore**</u> whatever handlers were previously used (to restore, you'll need to store the handlers from the original calls).
   - This does **not** solve issue 3 - the reader can still be killed by `KILL` or `STOP`.

<u>**Before**</u> fixing each issue, make sure you can reproduce it and cause the specified problem. Only then, start handling it as instructed, and finally make sure it's fixed.

<u>Guidelines</u>

- Use CTRL+D to send EOF to the console.
- Use only system calls (learned in the recitations) to access files, unless otherwise stated.
- Be careful with system calls – check everything!

Part 2

In this part of the assignment you will create a file copy utility using memory-mapped files.

Write a program **copy_tool.o**. It will accept two command-line parameters: path to a source file, and path to a destination file. The program will:

- Open the **source file**.
  - *If it doesn't exist, output a proper error message and exit the program.*
- Open the **destination file**, and truncate it to the appropriate size (**man 2 truncate**).
  - *Even if the file already exists.*
- Copy the source file's contents into the destination files using memory mapping **only**.
  - *Use no system calls or C functions / macros.*
- If an error occurs during the program, do not remove the destination file (**man 2 unlink**).
- Make sure the destination file is created with **proper permissions!**

Guidelines

- Use only system calls (**learned in the recitations**) to access the files. **Do not use read/write!**
- Read the guidelines carefully - especially the part about memory usage.
- **Handle all errors and exit gracefully** – properly *unmap* and *close* all files, **even on errors!**
  This is in contrary to the guidelines, in this case only.

Submit:

- Three C files: *reader_pipe.c*, *writer_pipe.c*, *copy_tool.c*