Operating Systems. Home Work #2.

Due by April 25, 2018, 23:55.

In this exercise we continue practicing inter process communication. This time, we use shared memory and data pipes. You need to implement two C programs (Parts I,II).

Remark

If you put your work in your TAU home directory, set file and directory permissions to be accessible only by you, i.e. the owner.

Part I

Write a C program `sym_count.c` (Counter) that counts instances of a symbol in a data file. Each time the program finds the symbol it increases its internal counter.
Command line arguments:

1. Full path to a data file (string)
2. Symbol to find (character)

The flow:

1. Use mmap to load the data file into virtual memory.
2. Go over the data.
3. Update the instance counter every time the symbol (argv[2]) is seen .
4. When all the data is processed report the obtained result to the manager by sending a message "Process %d finishes. Symbol %c. Instances %d.\n".
5. Exit gracefully.

Part II

Write a C program `sym_mng.c` (Manager) that manages the execution of sym_count processes. Each time Manager detects a finished counter it prints to standard output the data received from the counter.

Command line arguments:

1. Full path to a data file (string)
2. Pattern. Symbols to find (string)

The flow:

1. (Initialization) For every symbol in the Pattern string launch a Counter process with that symbol to count.
2. Sleep for 1 second. Iterate through the launched processes.
3. If a process is finished:
   - Exclude the process form the list of managed processes
   - Print to standard output the data reported by that process.
4. When all the processes are finished, exit gracefully.

Communication between Counter and Manager

You decide what kind of data pipe to use. There are several options shown in the class.

1. Manager uses pipe system call and "fork"s a Counter child processes.
2. Manager creates (mkfifo) per child process, and provides the FIFO file name in the command line, or by setting a custom environmental variable. All FIFO files are removed at the end.
3. Manager creates a FIFO file with a predefined name, defined at compile time. The FIFO file is removed at the end.

Handling SIGPIPE

Take care of SIGPIPE. Register your custom SIGPIPE signal handler both in Manager and in Counter.

Counter

If a Counter process receives SIGPIPE, then it:

- Prints a message in the format: "SIGPIPE for process {PID}. Symbol {SYMBOL}. Counter {COUNTER}. Leaving.", where PID, SYMBOL, COUNTER are the values, relevant to this process.
- Releases all the allocated resources.
- Exits.

Manager

If a Manager process receives SIGPIPE (can it receive it? Think about it and why it happened if you encounter SIGPIPE in the manager), then it:

- Prints a message in the format: "SIGPIPE for Manager process {PID}. Leaving."
- Terminates (SIGTERM) all the Counters.
- Releases all the allocated resources.
- Exits.

General assumptions:

1. The command line arguments are supplied correctly in every executable. i.e. their order and types are as stated above.

Submission Guidelines

1. Submissions are in Moodle.
2. Submit one ZIP file that contains two source files: `sym_count.c, sym_mng.c`
3. Name the ZIP file `hw2_012345678.zip`, where 012345678 is your ID number.
   For example, in BASH:
   ```
   zip hw2_012345678.zip sym_count.c sym_mng.c
   ```
4. Attention for Mac users: Mac pushes some hidden subfolders into zip archives. Remove them.

We check

1. Programs implement the specified behavior.
2. Error handling in the C code.
3. **Do not** use variable-length arrays (https://en.wikipedia.org/wiki/Variable-length_array).
4. If a buffer is allocated dynamically then it should be released correctly.
5. File reading **shall not** be done using read/fread, it should be done using mmap.
6. You can use printf/puts for STDOUT outputs.