Operating Systems. Home Work #1.

Due to April 8, 2018, 23:55.

In this exercise we practice some basic Bash scripting and interaction between a Bash script and a C program. You will need to implement some C programs (Parts I,II) and a script that runs it (Part III).

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 suspends itself.
Command line arguments:

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

The flow:
Signal handlers

1. Define signal handler for SIGTERM. The program prints the number of counted instances and terminates. The message is in the format: “Process %d finishes. Symbol %c. Instances %d.\n”, with PID, symbol, and the instance counter as the arguments.
2. Define signal handler for SIGCONT. The program prints a message in the format “Process %d continues”, with PID as an argument, and continues the processing.

Main

1. Read the data file in chunks (choose 512/1024/2048 bytes).
2. Every time the symbol (argv[2]) is seen:
   - Update the instance counter
   - Print a message in the format “Process %d, symbol %c, going to sleep\n”, with PID and symbol as the arguments.
   - Suspend itself (raise SIGSTOP)
3. If EOF is reached then invoke SIGTERM handler and terminate gracefully.

Part II

Write a C program `sym_mng.c` (Manager) that manages the execution of sym_count processes. Each time Manager detects a stopped counter it tries to resume it.

Command line arguments:

1. Full path to a data file (string)
2. Pattern. Symbols to find (string)
3. Termination bound (integer)

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 suspended/stopped:
   - Increment the Stop Counter for that process.
   - If the Stop Counter is equal to the Termination bound, then:
     - Send SIGTERM to that process.
     - Exclude the process form the list of managed processes
   - If the Stop Counter is less than the Termination bound, then send SIGCONT signal.
4. If a process is finished, then exclude the process from the list of managed processes.
5. If all the processes are finished, terminate.

Part III

Write a script `sym_mng_launch.sh` that launches the Manager. The script assumes the existence of the following environmental variables:

1. ${FULL_EXE_NAME} – full path to the Manager executable file. Example: “/some/where/sym_mng”.
2. ${PATH_TO_DATA} – a path to some folder. Example “/some/where/DATA”. No terminating “/”.
3. ${DATA_FILE} – a data file name. Example “test1.dat”
4. ${PATTERN} – a pattern, string of symbols to search. Example “abcd@~_”
5. ${BOUND} – termination bound. Example “20”.

The flow:

1. Concatenate correctly PATH_TO_DATA and DATA_FILE.
2. Set full access rights to the user and cancel any rights to group and others for the data file.
3. Launch the Manager at the background (Hint: use terminating “&”) and quit.

General assumptions:

1. The script will be run from the working directory.
2. 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, sym_mng_launch.sh`
3. Name the ZIP file `hw1_012345678.zip`, where 012345678 is your ID number.
   For example, in BASH:
   `zip hw1_012345678.zip sym_count.c sym_mng.c sym_mng_launch.sh`
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. The script shall be a Bash script. You can start it with "#!/bin/bash" but we will not take points if you don't.
6. File reading **shall not** be done in byte-after-byte manner.
7. Use system calls for file access.
8. You can use printf/puts for STDOUT outputs.