CPSC 351, Operating Systems Concepts

Homework, Processes and IPC

Inspired by Exercise 3.19 in the textbook

1) (*20 points*) Write a C program (time_shm.c and time_pipe.c) that determines the amount of time necessary to run a command from the command line. This program will be run from the command line as

   ```
   ./time <command [args...]>
   ```

   and will report the amount of elapsed time to run the specified command. This will involve using `fork()` and `execvp()` functions, as well as the `gettimeofday()` function to determine the elapsed time. It will also require the use of two different IPC mechanisms.

   The general strategy is to fork a child process that will execute the specified command. However, before the child executes the command, it will record a timestamp of the current time (which we term "starting time"). The parent process will wait for the child process to terminate. Once the child terminates, the parent will record the current timestamp for the ending time. The difference between the starting and ending times represents the elapsed time to execute the command. The example output below reports the amount of time to run the command `ls`:

   ```
   ./time ls -l

   total 156
   -rwxr-xr-x 1 thomas users 25616 Feb  4 15:59 a.out
   -rw-r--r-- 1 thomas users   252 Feb  4 16:00 output.txt
   -rwxr-xr-x 1 thomas users 86024 Feb  2 20:58 project.exe
   -rwxr-xr-x 1 thomas users 25616 Feb  2 21:00 time
   -rw-r--r-- 1 thomas users  5144 Feb  2 20:58
   time.c

   Elapsed time: 0.001448 seconds
   ```

   As the parent and child are separate processes, they will need to arrange how the starting time will be shared between them. **You will write two versions of this program, each representing a different method of IPC.** time_shm.c and time_pipe.c have been provided to get you started.

   a) (*10 points*) The first version, time_shm.c, will have the child process write the starting time to a region of shared memory before it calls `execvp()`. After the child process terminates, the parent will read the starting time from shared memory. The region of shared memory should be established before the child process is forked, allowing both the parent and child processes access to the region of shared memory.

   b) (*10 points*) The second, time_pipe.c, version will use a pipe. The child will write the starting time to the pipe, and the parent will read from it following the termination of the child process.

   Hints:

   1. For IPC between the child and parent processes, the contents of the shared memory pointer can be assigned the `struct timeval` representing the starting time. When pipes are used, a pointer to a `struct timeval` can be written to - and read from - the pipe.

2. The child process replaces its current process image with a new process image using the array version of exec, called `execvp`. `execvp` takes two arguments, a pointer to a file name containing the program to execute, and a pointer to an array of strings used as arguments of the program to execute. The first string in the array, by convention, should (again) point to the filename associated with the file being executed. See https://linux.die.net/man/3/execvp. For example, you can forward the arguments passed to your timer program (e.g., `ls -l`) to `execvp` like this:

   ```
   int main( int argc, char *argv[] )
   {
     ...
     if( fork() == 0 ) // child process
     {
       ...
       execvp( argv[1], argv + 1 );
     }
     ...
   }
   ```

   argv[1] ***is*** a pointer to a file name containing the program to execute

   argv+1 ***is*** an array of strings used as arguments of the program to execute

3. Using the Build.sh script from your first homework, compile and link your program like this:

   `./Build.sh time`

   Notice the argument `time` specifies the name of your executable file. Remember, Build.sh attempts to compile and link everything in the folder. If you receive an error that says something like "`multiple definition of `main'`", make sure that time_pipe.c and time_shm.c are not both present at the same time in the same directory when building.

4. `./time ls -l | tee -a time_shm_output.txt` is an example command to run your program while collecting your program's output to a text file and seeing the output on the console. Test your program with different commands, such as:

   a. `./time pwd | tee output.txt`

   b. `./time cat time.c output.txt`

   c. `./time Build.sh "my executable file"`

   The output of any command with at least one argument (`ls -l`, for example) can be captured to output.txt and submitted, but the same command must be used for both versions (shared memory and pipe).

Deliverable Artifacts: Submit (only) the following 4 files to Canvas.

- time_shm.c, time_shm_output.txt (both files must be present to earn credit)
- time_pipe.c, time_pipe_output.txt (both files must be present to earn credit)