

COP4610 / CGS5765 Operating Systems Project 3

Process Synchronization

Important Notes

1. This is a **single-person** project. Do your own work!
2. The project is based on the Linux operating system, not the Xv6.
3. Make sure your program works on `linprog.cs.fsu.edu` because that is where it will be graded.

Programming Task

In this assignment, you will learn how to use `fork` to create processes, and how to use Linux pipes and shared memory for process synchronization. Specifically, you will create a program called `trans`. The program takes two command line parameters, an input file and an output file. It copies the input file to the output file using the shared memory. For example, to invoke the program, type

```
trans input-file output-file
```

at the command line prompt.

Here is how the program works: your program will create two processes, a parent process and a child process, using `fork`. These two processes share two pipes, one pipe carries data from the parent to the child and the other carries data in the opposite direction (a regular pipe is unidirectional.) The parent and the child also shares a 4KB memory. To transfer the file, the parent first reads a block of 4KB data from the input file into the shared memory (the last block of the data might be less than 4KB if the file size is not a multiple of 4KB), and sends the block number and the block length to the child through its pipe. After receiving the block number and the block length, the child writes the data from the shared memory to the output file, and the sends the block number back to the parent as an acknowledgment through the other pipe. After receiving the correct block number from the child, the parent continues to transfer the next block. This process continues until the whole input file has been written to the output file. The following are some specific requirements:

1. Put all your code in a single file, called `trans.c`.
2. Use Linux pipe to create pipes. You can find the API and an example program at <http://man7.org/linux/man-pages/man2/pipe.2.html>.
3. Use the POSIX shared memory, **NOT** SysV shared memory. That is, your program should NOT use SysV APIs such as `shmget` and `shmat`. Instead, use `shm_open` and `mmap`. POSIX shared memory is the preferred way to use shared memory on Linux.

You can find the man pages for POSIX shared memory at http://man7.org/linux/man-pages/man7/shm_overview.7.html, and some example code at <http://www.cse.psu.edu/~deh25/cmpsc473/notes/OSC/Processes/shm.html>. Note that programs using the POSIX shared memory API must be compiled with `cc -lrt` to link against the real-time library, `librt`. You can use the following command to compile your code:

```
gcc -std=c99 -Wall -Wextra -D_XOPEN_SOURCE=700 -o trans trans.c -lrt
```

The shared memory must be created after the process has forked.

4. The block numbers and the block lengths should be sent and received as the standard C int. On `linprog`, it will be a 64-bit integer since `linprog` uses 64-bit linux. The block number starts from 1. On a 32-bit Linux system, they will be 32-bit integers.

After all the blocks have been transferred, the parent process sends block number 0 and length 0 to the child. After receiving block number 0, the child replies the parent with block number 0 and then exit. The parent receives the child's acknowledgment to block 0 and exits too.

5. Your code should include necessary error handling. For example, it should check the errors for `fork`, pipes, and shared memory. Both examples given previously have the necessary error handling.