Operating Systems Writeup

Name: Joshua Abraham

Date: 2 November 2017

Current Module: Operating Systems

Project Name: ”Relay”

Project Goals:

The relay project aims to create two programs that communicate via Interprocess Communication. This IPC will be one-way, sending input from a ”**dispatcher**” program that takes input from standard input to a ”**listener**” program which then prints text sent to the first program.

Considerations:

- **Dispatcher** must take text from stdin and exit when it receives an EOF.
- **Listener** should exit if the listener is closed or not running.
- Multiple **listeners** should be able to run simultaneously from different users/directories.

Initial Design:

The project is composed of the following files:

· *Makefile*: The main makefile for the project.

· *dispatcher.c*: The source code for the **dispacher**.

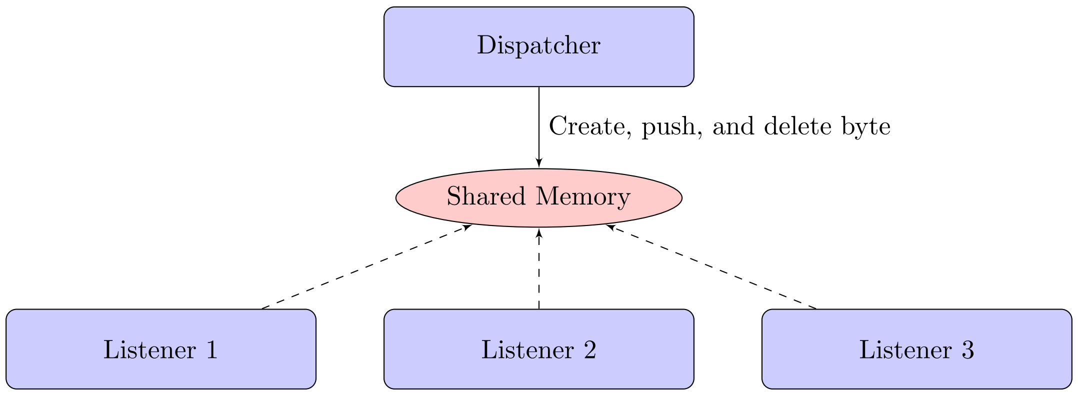
· *listener.c*: The source code for the **listener**.

· *relay.c*: The source code with a helper function used by both programs.

· *relay.h*: The header file for the project.

Data Flow:

The data flow from **dispatcher** to **listener** is as follows:



(1)

My project operates character by character, byte by byte on input from the user. Input is given to the running dispatcher which has changed the terminal settings to non-cannonical mode in order to get data unbuffered. Each character is copied to the one byte shared memory segment then the dispatcher sleeps for 5000 nanoseconds. Then, the shared memory segment is set to NUL. Before the shared memory is set to NUL, the listener prints the data and then sleeps for 5000 nanoseconds. This loop is performed until the user gives a control-D to the dispatcher. In order to prevent a control-C from prematurely ending the dispatcher (which would result in the shared memory segment not being deleted), a tiny SIGINT handler is used at the beginning of the program. The listeners test for the existence of the shared memory segment and if it does not exist, the listener exits cleanly. Once the user sends a control-D to the dispatcher, the dispatcher sets the shared memory to EOT which is the magical number that closes the listeners.

Communications Protocol:

The dispatcher and listener performed IPC via System V shared memory. This communication was one-way and used exactly one byte of memory.

Potential Pitfalls:

- Listeners re-printing data in shared memory before it is deleted.
- Listeners closing when user gives a control-D to the dispatcher.
- Dispatcher not prematurely closing before sending EOT to listeners.

Test Plan:

User Tests:

· Performed test specified in the project supplement manual.

· Sent dispatcher a control-C in the middle of execution.

· Checked system IPC mechanisms via the "ipcs" command to ensure that all shared memory is deleted after successful execution.

Test Cases:

· Tested SIGINT handler and shared memory deletion.

Conclusion:

The relay project was a learning experience for me. I learned about the various IPC mechanisms that UNIX provides such as message queues, semaphores, shared memory, etc. I decided to utilize shared memory after reading the textbook and deciding that this method was best for one way, multicast communication. I initially planned on using semaphores for the dispatcher to keep track of all listeners, however, I decided that it added unneeded complexity and simply used the EOT character to terminate the listeners. Throughout the development of the project, I chose to prioritize efficiency and simplicity in order to produce a fast, accurate program. This is why I chose to operate character by character and to only use one byte of shared memory. I am very happy with the outcome. The only possible change for the future would be to use a faster mechanism for getting data from the shared memory.