Name: SGT Schoenwald-Oberbeck, Jesse
Date: 12MAY2017
Current Module: Operating Systems
Project Name: Relay

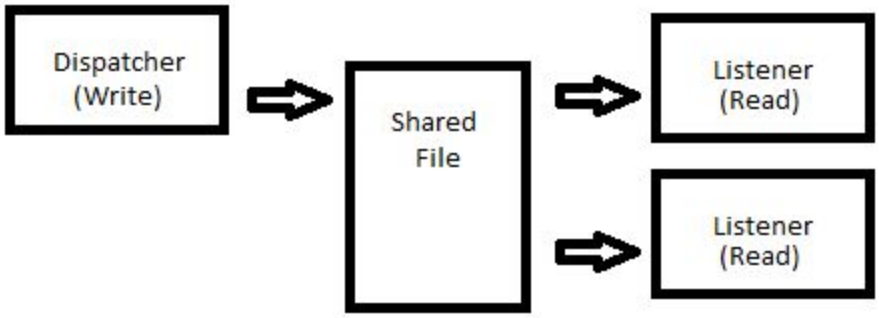
Project Goals:
Create a dispatcher and listener. Text typed into dispatcher should appear in any number of running listeners.

Considerations:
o Ctrl+D and Ctrl+C should close both the dispatcher and all listeners when entered in dispatcher.
o Make should create two binaries.
o Neither binary should require arguments.
o Should be runnable from different directories/users.

Initial Design:
Dispatcher.c, and listener.c will be compiled to their respective binaries on invocation of make. Both files are self contained/monolithic, as they are short enough that refactoring was completely unnecessary, and would do nothing but hurt readability.

Data Flow:
Dispatcher will use a set file as shared memory with each listener, meaning as text is entered in the dispatcher, that same text is immediately available to each listener, as they are actively accessing a single file.
Dispatcher loops, taking user input *non-canonically* and writing it to the shared memory space.
The listener(s) read in a loop the same memory and print it to the open terminal.
Upon receipt of EOT or SIGINT, the listeners will close all at once, and the dispatcher will close once enter is pressed.

Communication Protocol:
IPC via shared memory. A single file is actively used by multiple processes simultaneously.



Potential Pitfalls:

o Shared memory.

o Non canonical transmission/raw-terminal use is tricky, relatively.

Test Plan:

User Test:

Multiple runs of the program, using every variation/combination of options and inputs the user can think of.

Test Cases:

All test cases completed with correct output.

Conclusion:

Shared memory space was interesting to work with, and could have come with some huge issues, such as overflow. In testing, with proper use of limits (hard coded to limit stack usage) rather than overflowing out of the buffer, input circles back to the beginning of the buffer and begins to overwrite itself. So in effect, it doesn't limit the user's entry. The user can step on their own toes, so to speak, if they wanted to keep a history, but history is a small sacrifice for security in this case. Shared memory was an interesting concept to learn and work with, and I'm curious if and how it will come up in the future.

Non canonical/raw entry is a bit of a pain, and generally I wouldn't use it. Might one day be useful information, but were it not for the challenge I would not see a point.

Shared memory was actually not my first approach. Initially I went with UDP broadcast over a network socket on loopback. This worked, however it has the same issue as other pipes, in that one listener would remove the input from the buffer. I thought about having each listener register with the dispatcher, incrementing a counter, and having the dispatcher broadcast the same input that many times, so that each listener would get a copy, but this is too dependent on the OS's scheduling, or having the dispatcher keep a list of registered listeners via file descriptor, but ultimately I thought shared memory was a better (or at least more interesting) solution.