Loren Howard, Yujian Zhang, Ken Wakaba

Professor Ahmed Ezzat

COEN 383

27 November 2017

UNIX I/O Project Report

Within this project, our task was to practice UNIX I/O system calls by reading from multiple files and writing to the file system. This was done by first creating 5 pipes and attaching them to five child processes (made by calling the fork() command). Four of these pipes were to generate messages with time stamps and sleep for a few seconds after generation, while the fifth process was to wait for user input from the terminal, which would then be written to its pipe. All these processes should be completed within a 30 second period.

When implementing this project, a few complications came about that we were able to overcome. The first was figuring out the use of pipes as well as the **select()** system call. select() allows a program to monitor multiple file descriptors, waiting until one becomes "ready" for an I/O operation. A file descriptor is considered ready if it is possible to perform the an operation such as a read or write without blocking. In this instance, we create these file descriptors by calling the pipe() system call which essentially creates a pair of file descriptors, pointing to a pipe inode, and places them in the array. The first index of the array is for reading (array[0]), while the second index is for writing (array[1]). Once we understood how to incorporate all this within the project, implementation became much simpler.

Another obstacle we ran into was how to incorporate the time stamps as well as how to signal when the 30 second interval was over. To do so we had to use a few system calls. The main call we used was the **signal()** system call. Signals are generated when an event occurs

that requires attention. signal() is a function that accepts two arguments and returns a pointer to a function that takes one argument, the signal handler, and returns nothing. In this case, we used signal() with SIGALRM which signals an alarm clock. We also utilized **gettimeofday()** which returns the current time expressed as seconds and microseconds since the Epoch, and store it in the timeval structure. With the signal() call, we are able to send a signal after the 30 second period has been completed so that we know when to end the program.

With these problems resolved, we were able to fully implement our project successfully.