Evan Nichols

EECS 678

Lab 4 Report

1. Signals are used to send a short message to a process or a group of processes. After receiving a signal, processes can choose one of the following actions after a signal is delivered:

- perform the default action associated with the signal.
- ignore the signal.
- catch the signal with a signal handler, a pre-defined function that is automatically called upon signal delivery

Now, in the case of KILL and STOP, the operating system itself handles this signal directly, killing or stopping the appropriate process. Consider what would happen if a these signals were NOT handled directly by the OS. A malicious programmer could choose to ignore the KILL or STOP signals until their process is completed, or perhaps define a handler that spawns a new process upon receiving the KILL or STOP! Simply put, having control over the KILL or STOP signals means potentially having control of an entire computer. Malicious programs (copying user data to a remote location, monitoring keystrokes) would be able to run without any ability to stop them.

2. Using the pause( ) system call causes the calling process to wait until any signal is received, usually placed within a loop in the main function. For this lab in particular, it is ideal to use the pause( ) within an infinite while loop because we want to keep track of the number of keyboard interrupt signals received.

3. The other signals are masked because we do not want any interference with the execution of our handler functions. Our program deals only with the Ctrl+Z and Ctrl+C, and we want them to be handled by the functions we created.

4. While implementing the timeout, we do not mask the SIGALRM signal because we want it to be handled normally if a response is not received. If we blocked the SIGALRM signal, then there would be no way to implement the timeout function.