Kam Bielawski - 2972621 - EECS678 Lab 6

1. There are two special signals (KILL and STOP) which are not handled by the process they are sent to. When a KILL or STOP signal is generated, the operating system itself handles this signal and kills or stops the appropriate process. Considering what you learned in today's lab, speculate as to why the system designers chose to include signals which are handled solely by the operating system.

We need the ability to terminate arbitrary processes from the OS because those processes could be running someone else's (ie. an attacker's) code. If there was a way to handle every single signal you send to a process, you have the potential for an unkillable process, which could be executing malicious code.

2. What benefit do we gain from using the pause system call as opposed to an infinite while loop?

The process won't eat up the processor's resources (ie. the process won't be sitting in the "ready" queue for the scheduler). An infinite while loop will cause the processor to execute useless code for much of its CPU time.

3. Why do we mask other signals while inside the signal handler?

A signal handler's code could be important to the proper execution of the program. If it is interrupted for the purpose of *another* signal, then the handler's code would stop executing.

But then the incoming signal's handler would be masked and never executed, so I guess we have to be careful what we mask and don't mask and if some signal's handlers take precedence over others.

4. When we implement the time out, we do not mask the SIGALRM signal. Why?

We have to unmask SIGALRM because we call alarm(5) from *inside* the check_count() function, which is called from the SIGINT handler. If SIGALRM was masked and and SIGALRM is sent to the process, then it wouldn't be handled by catch_alarm() - it would be masked.