**CSC-3044 Operating Systems & System Programming**

**Laboratory 2 Report**

**NAME:** Joseph Leskey

1. **Do you see a connection to the above description of strace?**  
     
   Sort of, yes.
2. **If you see a connection, what is it?**

The standard streams are represented by file descriptors, which means that they have roughly the same interface as a stored file. In this way, they are treated as the default I/O files and redirection to some stored file can occur quite smoothly.

1. **Figure out where each symbolic link takes you using the ls -li command and list the result of each in your lab report.**  
     
   /dev/stdin → /proc/self/fd/0 → /dev/pts/3  
   /dev/stdout → /proc/self/fd/1 → /dev/pts/3  
   /dev/stderr → /proc/self/fd/2 → /dev/pts/3  
     
   They link to the first three file descriptors in the context of the current terminal.
2. **Do the three symbolic links for stdin, stdout, and stderr, all point back to one device or different devices?**  
     
   The same device. When I ran the commands, the device was represented by /dev/pts/3.
3. **Where does the device file(s) point to?**  
     
   It points to the current terminal interface. In this case, probably due to my SSH connection, it was pseudoterminal 3.
4. **What type of device is it (character or block) and how would you know?**  
     
   It’s a character device. The file type in the mode string is “c.”
5. **What is the count of system calls before the “clone” system call?**  
     
   It appears to be 27.
6. **What call does clone most closely resemble from the Win32 API?**  
     
   There is no single Win32 API call that resembles `clone` due to its flexibility. It very roughly maps to a combination of `CreateThread` and `CreateProcess`, but in order to more closely approach its full functionality, one would have to dip into Native API calls.
7. **When would you want to use clone instead of fork?**  
     
   In Linux systems, `clone` is used to implement the POSIX `fork` syscall behind the scenes. Clone has considerably more options when it comes to the allocation and sharing of resources, so it is appropriate to use it whenever more flexibility is desired.
8. **What do you notice about the parent ID value for the child and parent branches?**  
     
   The parent ID of the child process matches the process ID of the parent process, while the parent ID of the parent process matches the process ID of the shell (its parent).
9. **From your reading, what type of structure is being formed from this collective group of process identifier associations?**  
     
   A process tree.
10. **What does this modification represent?**  
      
    It keeps the parent alive indefinitely, creating a state where the parent can maintain its child process(es) and/or provide continuous functionality. It could be taken to represent a lower-level shell (if I’m understanding the question correctly).
11. **What does this modification represent?**  
      
    It effectively creates a command line, similar to that of a shell. The child process can receive input indefinitely, as long as it is kept alive (and error-free).
12. **How does the information from the status command compare with what you might get from top or htop?**  
      
    The status files go into great detail about a specific process, whereas top and htop offer performance metric summaries across several processes.
13. **Add these observations to your report document.**  
      
    The child process had a relatively high number of voluntary context shifts and a relatively low number of nonvoluntary context shifts, reflecting the fact that the child process often yielded control of the CPU as it waited for input. The parent process is just the opposite. It spent most of its time executing continuously, which prompted the scheduler to frequently suspend it and led to a large number of nonvoluntary context shifts.
14. **What changes do you observe from gathering information using the status file and top/htop programs?**  
      
    This change means that the parent process rests instead of looping continuously. Accordingly, it is subject to far fewer nonvoluntary context shifts and the processing burden it places on the CPU is greatly reduced.
15. **What can you observe about the CPU utilization of the parent and child process?**  
      
    Each is down to a low level of CPU utilization since they are both resting when not in active use. This is considerably more efficient than the continual processing incurred by an infinite loop.