Derek Lee, Shine Li Computer Operating Systems Fall 2020

Professor Hakner PSET #4

Problem 1

a) ctrl + \ produces a core dump and terminates the process. This key sequence controls the SIGQUIT signal.

b) SIGTTOU would be sent to a process when that process is in the background (as opposed to the foreground) and it attempts to write to the terminal or set the modes of the terminal. It is only sent if the TOSTOP output mode is set. It causes the process to stop until it receives the SIGCONT signal to resume.

c) Signal #40 is a real-time signal, so it will be queued while it is blocked. When the signal is unblocked, the signals are delivered separately and in the order they were sent. The handler receives the signal and blocks the signal bit. If the handler returns, the signal bit gets unblocked, and all three signals are handled by the handler. Otherwise, the handler will receive the signal one time.

d) The program prints:
“In handler instance 1
the other side”.
The char pointer points to NULL. SIGSEGV is handled by the handler. setjmp() sets the return position from longjmp(). On the first call, it returns 0, so the contents of the IF statement are skipped. *p-- results in a segmentation fault (because it is attempting to modify a NULL), which is handled by the handler. The handler increments i, so the current value is 1. The handler blocks the signal bit, but does not unblock it, because it does not return from the handler function. Instead, it uses longjmp(), which returns 1 to setjmp(). The contents of the IF statement are now executed, which causes another segmentation fault. At this point, the signal bit is still blocked, so the program should return with 1. However, during testing, the program terminates with 139 (128+11).

This makes some sense, as blocking a segfault seems absurd, so the kernel may forcibly exit the program. It is possible that SIGSEGV is blocked from the custom handler, and goes to the default handler instead. According to the man page, ignoring SIGSEGV produces undefined behavior. However, it is being blocked, not ignored, so this should not apply. Changing the longjmp() to a return causes an infinite loop. This is because the original cause of the SIGSEGV was not resolved and the signal is no longer blocked after the handler returns. The program attempts to re-execute the line that caused the SIGSEGV, which results in an infinite loop.

Problem 2

a) No, this will never produce ABA. The write request size is 1024, which is less than 4k, meaning that the write will be handled atomically, so even if there are multiple writes to the same pipe, they will never interleave. It does not matter the order in which the children execute.
b) The read system call will be in a sleep state forever, because there is no EoF condition, which would normally be produced by closing the write side of the pipe.