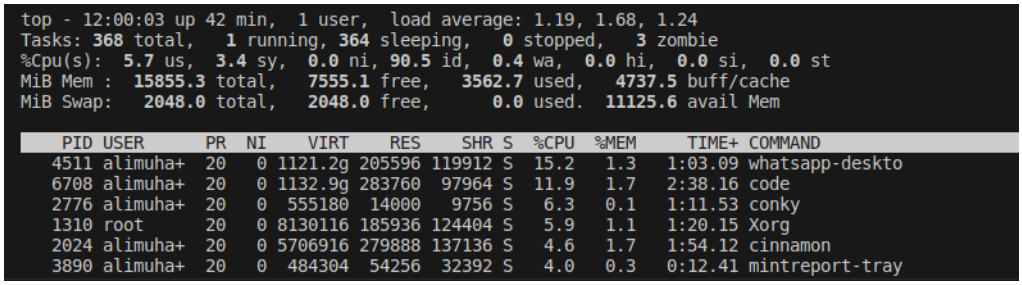
Operating Systems - CS/CE 232L/324L
Lab 13: Using Signls to Communicate Between Processes

Ali Muhammad Asad - aa07190

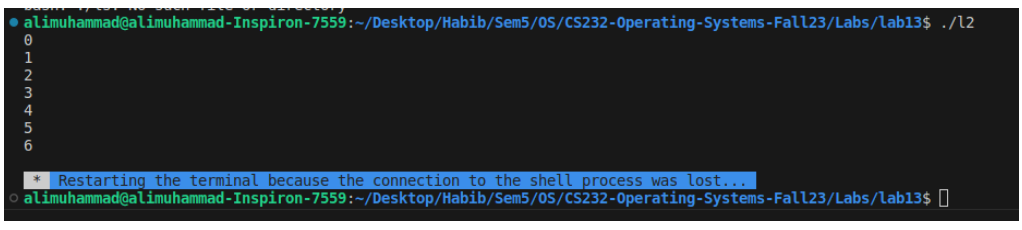
Question:	1	2	3	4	Total
Points:	0	0	0	0	0
Score:					

Question 1: How did you stop the code in section 3?

Solution: Since the process was going on forever, and `Ctrl+C` wasn't working, I opened a new terminal, and ran the ``top'' command. This effectively showed me a list of the processes running at that time. I then found the process ID of the process I wanted to terminate, and then ran the command ``kill -9 <PID>'' to kill the process. The command ``kill -SIGTERM <PID>'' could also be used.



The image depicts a list of processes when ``top'' was run [the process in question vanished by the time since sleep had been called, but it was noted]. The command was then run; command in question ``kill -9 16503''. When the command was run, the process was terminated as shown below:



Question 2: How do we use signal function in section 3?

Solution: The `signal(arg1, arg2)` function in section 3 is used to define how a program handles different signals, essentially interrupts or notifications sent to a process by the OS. The first argument is of type `int`: the signal that we want to handle. For eg, `SIGINT` is the signal number for the interrupt signal generated when we press `Ctrl+C`. The second argument is of type `void (*func)(int)`: the function that we want to run when the signal is received. In our section, `SIG_IGN` is used to ignore the signal. The return type is of type `void (*func)(int)`: a pointer to the previous signal handler for the specified signal.

Question 3: Try converting void signal_handler function in section 4 to a return function. What happened if you do that. Explain why did that happen.

Solution: I changed the function to a static int to return an integer, a character type to return a char, and a double to return a double. My compiler did issue some warnings when I compiled the program, however, the behavious did not change when I ran the program. This could be since the expected signature for a signal handler function is void. So by changing the return type, it no longer matches the expected signature as per convention. Signal handling is meant to be a quick response, so signal handlers are designed to return 'void'. The fact that the behaviour didn't change is an example of how undefined behaviour can manifest in C. The return type is not used by the system. Since the operating system's signal dispatching mechanism doesn't do anything with a return value from a signal handler, returning an int instead of void has no practical effect.

Question 4: In lab exercise, try interrupting the terminal before the quit/exit is called in the code.

Solution: This keeps on running since the child process keeps on ignoring the interrupt signal, much as it was happening in section 4 (same reasoning).