**Part 1: Signal Research**

Each signal has a disposition that determines how the process behaves when the signal is delivered.

A signal handler is a function that that is executed in response to a signal being sent. They are used so that the programmer can create custom behaviors to respond to each signal.

Term: Terminates the process

Ign: Ignores the signal

Core: Terminate and core dump

Stop: Stop the process

Cont: Continue process if currently stopped

One way to send a signal programmatically is to use the kill() function, which takes in a pid and a signal as parameters. So a code snippet to do this would be:

pid\_t target = 1111;

if (kill(target, SIGTERM) == 0) {

printf(“Signal sent to process\n”);

} else {

Printf(“Failed to send signal\n”);

}

One way to send a signal through the command line is also the kill command line argument. This is done using a command that looks like kill -signal pid. An example of this would be

kill -SIGTERM 1111

SIGINT:

Description: Signal Interrupt, interrupt from keyboard

Disposition: TERM

Override: Yes, the disposition can be overridden, so that resources can be cleaned up before exiting the process.

SIGTERM

Description: Signal Terminate, requests termination of process

Disposition: TERM

Override: Yes, the disposition can be overridden so that resources can be cleaned up before terminating the process.

SIGUSR1

Description: User defined signal 1, the developers can define the meaning.

Disposition: TERM

Override: Yes, the disposition can be overridden since the whole point of the signal is that the developer determines what it does.

SIGKILL

Description: Kill signal, used to immediately terminate a process

Disposition: TERM

Override: No, the disposition cannot be overridden since the signal wants to immediately end the process, with nothing done before that.

SIGSTOP

Description: Stop signal, Stops the process

Disposition: STOP

Override, No, the disposition cannot be override since the signal wants to immediately sotp the process.

**Part 2: Working with Signal Handler**

Two was to send the SIGINT signal to the process is to use CTRL-C while it is running and use the kill command through the command line, which sends the signal to the specific pid. (kill -SIGINT <pid>)

SIGINT was sent to the process using the signal() function, which catches the SIGINT signal and goes to the signal\_handler() function.

The code was modified to use SIGKILL instead of SIGINT, since SIGKILL’s disposition can not be overridden. This means that it never goes to the signal\_handler() function, and instead just ends the process then and there.

**Part 3: Signals sent from the operating system**

signal\_alarm: on GITHUB

signal\_segfault: on GITHUB

For signal\_segfault, there is an infinite loop. This is because when handling the segmentation fault, the process just goes right back to where the signal stopped it. This means it will continue to try and dereference the null pointer, and will continuously print out the error message in the handler function.

**Part 4: Getting details from a received signal**

signal\_sigaction: on GITHUB

**Part 5: Sending Data with a Signal**

recv\_signal: on GITHUB

send\_signal: on GITHUB