Jacob Tyree
Lab10

## POSIX Signals:

1. What is a Signal Disposition?
    a. It is the action a process takes when it receives a particular signal. Dispositions include certain actions like ignoring the signal, terminating the process, stopping the process, or calling a specific function to handle the signal.
2. What is a Signal Handler? What is it Used for?
    a. A signal handler is a user-defined function in C that specifies what a process should do when it receives a specific signal. A program can register a signal handler to execute custom code when the signal is received.
    b. Signal Handlers are commonly used to clean up resources before termination, handle specific events like catching SIGINT to prevent a user from prematurely stopping a process, and to restart or reconfigure parts of the program based on received signals.
3. Name and describe each of the 5 default dispositions.
    a. Terminate: This is when the process ends immediately, this is the default disposition for many signals.
    b. Ignore: This is when the signal is ignored, and the process continues as if it wasn't received.
    c. Stop: This pauses the process until it receives a CONT (continue) signal.
    d. Continue: If the process was previously stopped, this disposition resumes execution.
    e. Core Dump: This terminates the process and produces a core dump file, which contains a memory snapshot for debugging.
4. One way to programmatically send a signal to a process:
    a. One way to do it is by using the kill() function.
    b. An example of this is: int kill(pid_t pid, int sig);
    This is where pid is the process ID of the target process, and sig is the signal to send to request termination.
5. One way to send a signal to a process via the command line:
    a. From the command line, you can simply use the kill command to send a signal to a process by specifying its PID.
    b. An example of this is: kill -SIGTERM 1234
    This sends the SIGTERM signal to the process with PID 1234.

## POSIX Signal Types:

1. SIGINT: (Signal Interrupt). This is used when a user interrupts a process from the terminal, typically by pressing Ctrl+C. The default disposition is terminate if there is

not a custom disposition. This signal can be overridden, because you can define a signal handler to catch SIGINT and customize the behavior, allowing the program to handle an interrupt more gracefully.

2. SIGTERM: (Signal Termination). This disposition requests a process to terminate, which allows for graceful shutdowns. The default disposition is terminate as well. This signal can be overridden, because you can set a custom signal handler to perform custom cleanup operations before exiting.
3. SIGUSR1: (User-defined Signal 1). This is a signal reserved for user-defined purposes. It doesn't have a specific predefined behavior, so it's often used for custom purposes. The default disposition is also terminate. The signal can be overridden, since SIGUSR1 is designed to be customizable and you can set a signal handler to define any action that you want.
4. SIGKILL: (Signal Kill). This signal immediately terminates a process. It cannot be caught, blocked, or ignored, and it's used as a "force quit". The default disposition is also terminate. The signal cannot be overridden because the kernel immediately kills the process upon receiving the signal.
5. SIGSTOP (Signal Stop): This signal stops or pauses a process, similar to hitting pause on an application. This can later be resumed with SIGCONT. The default disposition is Stop. Additionally, the signal cannot be overridden because it directly stops the process without any handler.

## Part 2:

When running signal_handler.c, it prints "Sleeping" until the user registers a SIGINT signal.

One way to register a SIGINT signal is to hit Ctrl+C.

The second way is to issue a kill command in the terminal, where you would type in "ps aux" to find the PID of the program, and then issue the command "kill -SIGTERM <PID>" or kill -SIGINT <PID>.

How I sent the signal to the process is by typing in the command line the command I listed above. The result of doing kill -SIGINT <PID> stopped the program and displayed "Received a Signal". If I used the command "kill -SIGTERM <PID>", the program would also stop, but it would display "Terminated" instead of "Received a Signal".

### After Modifying the Code:

Using Ctrl + C does not terminate the program, instead it just displays "Received a Signal" and continues the program. One way to terminate the program is by finding the PID of the program, and then inputting into the command line the command "kill -SIGKILL <PID>".

Jacob Tyree
Lab10

## Part 3:

SIGALRM: Used to indicate that a timer has expired, and is sent to a process when the timer set by an alarm function reaches 0.

SIGSEGV: Commonly used with segmentation faults, which typically cause the program to terminate.

What is noticed in signal_segfault.c:

Upon running the program, when the signal is sent to the signal handler, the printout message gets repeatedly printed to the console until the user does a force terminate. This is happening because the program is going right back to where it left off after sending the signal to the signal handler. This point that it left off at is right when it sends the signal to the handler, and so it will keep sending the signal to the handler until it is forced to quit.

Sigaction: Used in POSIX-complaint systems to define a custom handler function ro a specific signal. This helps provide more flexibility and reliability than other system calls.