SFWRENG 3SH3: Operating Systems
Lab1: Unix Signals

Dr. Borzoo Bonarkdarpour

1 Concept

Signals are one of the principle operating system mechanisms for interprocess communication (IPC) as well as communication between the OS kernel and processes. Each Unix system has a fixed set of signals that one process can *raise* to cause an interrupt in another process that responds to a particular signal, whose response the original process can *catch*. A signal is a software mechanism for informing a process of the occurrence of an asynchronous event. Some signals (e.g., `SIGINT`, `SIGBUS`) are associated with hardware interrupts, while other signals are initiated from software. Unlike hardware interrupts, signals are not prioritized and have no concept of ordering.

For instance, when pressing `CTRL+C` (respectively, `CTRL+Z`)

- Keyboard generates a hardware interrupt
- The hardware interrupt is handles by the OS
- The OS sends a `SIGINT` (respectively, `SIGTSTP`) signal.

Each signal is represented as a single bit flag, and thus signals of the same type cannot be queued. When a process is scheduled for execution, it checks these signal flags and performs corresponding actions for posted signals before returning to user level. The signals can be handled by predefined (default) actions (e.g., termination for `SIGTERM`), can be caught by user-defined signal handlers, or (in some cases) can be ignored by the process. User-defined signal handler functions execute at user level and in the context of the target process.

2 Signal System Calls

A process may respond to a signal in one of three ways:

- take the default action with `SIG_DFL`
- ignore the signal with `SIG_IGN`

- catch the signal with an address of a programmer's function.

To make a signal handler operational, it must be *registered* with the kernel by using the `signal()` or `sigaction()` system call. For example:

```
signal(SIGALRM, SIG_IGN) /* ignore alarms from another process */
signal(SIGALRM, SIG_DFL) /* reset to default:  enable */
```

To define a customized response to a signal (third option) requires that the programmer must define a function whose signature appears as the second parameter of the signal function, namely,

```
void ( * sighandler )( int )
```

The function `sighandler` returns `void` and has one `int` parameter which is the signal number. The `signal()` function responds to its signal number parameter by calling a function named `sighandler`. The call requires knowledge of the address of `sighandler`, hence the `*`.

```
signal(SIGALRM, alarm_handler);
```

Some characteristics of Unix signals:

- The default action of a `SIGKILL` signal is terminating the process and cannot be overridden.
- A signal handler takes exactly one parameter, the signal number. Any additional arguments need to be passed to the signal handler through some other mechanisms.
- Signals can be sent from one process to another using the `kill()` system call. (Typically, the processes must be owned by the same user for the signal to take effect.)
- A process can also send signals to itself using `raise()` library routine, which invokes `kill()`.

3 Example

The following is a simple example C program that handles the `SIGALRM` signal with a user-defined signal handler.

```
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
```

```
#include<stdlib.h>

 // user-defined signal handler for alarm.
void alarm_handler(int signo){
     if (signo == SIGALRM){
         printf("alarm goes off\n");
     }
 }

 int main(void){
     // register the signal handler
     if (signal(SIGALRM, alarm_handler) == SIG_ERR){

         printf("failed to register alarm handler.");
         exit(1);
     }

     alarm(3);               // set alarm to fire in 3 seconds.
     while(1){ sleep(10); } // wait until alarm goes off
 }
```

4 Assignment

You are to write a C program that handles three signals:

- The `SIGALRM` that handles a timer that wakes up each 2 seconds. You may use the `sleep` and `raise` systems calls in an infinite `while` loop to create a timer. When your program receives `SIGALRM`, it should print "`Alarm`".
- The `SIGINT` signal by printing "`CTRL+C pressed!`"
- The `SIGTSTP` signal by printing "`CTRL+Z pressed!`" and then exiting the program.

An alternative way to test your program is by sending a signal from the command like by using the command `kill`. For example,

```
$kill -9 1234
```

sends `SIGKILL` to the process whose id is 1234. You may use the `ps` command to identify the process id of your program.

5 Guidelines

You will

- work on this assignment individually
- implement this assignment using C or C++
- present your implementation and output to your lab TA.

You may present your program in a different section in the same week **only once** throughout the term.