Mini-Lab 2: Signal

COMP3230, The University of Hong Kong

Sept. 2024

Total 1 point

Objective

At the end of this mini-lab, you will be able to:

- Gain hands-on experience in using signals as an inter-process communication method.

Instructions

Signals serve as a limited form of inter-process communication (IPC), acting as software interrupts. In this task, you will customize the `SIGUSR1` signal to terminate the process. To achieve this, we can utilize the `signal` function, and its signature is as follows.

```
signal(int sig , void (*handler)(int));
```

and a simple example is:

```
void sigusrHandler(int sig_num)
{
    // Custom code to handle the signal
}

int main()
{
    signal(sig, sigusrHandler);
}
```

Note:In C, `void` and `void*` have distinct purposes: `void`: Indicates that a function does not return a value. For example, `void function()` means the function returns nothing. `void*`: A generic pointer that can point to any data type. It is often used in memory management functions like `malloc()`, where the return type is `void*`, and the pointer can later be cast to a specific data type.

1. Open two terminals in VS Code under the same directory with `lab2-signal.c`.

2. In Terminal 1, compile and run `lab2-signal.c` without any modification. The program will run a dead-loop and write to local file `./pid.txt` its *process id* for us to terminate it.

3. In Terminal 2, execute `kill -10 $(cat ./pid.txt)` to send an SIGUSR1 signal (`sigusr1(10)`) using the command `kill` to the *process id* stored in `./pid.txt` or press `Ctrl` + `c` in Terminal 1.

   Before redefining SIGUSR1 handler: the program in Ternimal 1 will be terminated and print out "<pid> user-defined signal 1 <file>", which is the default behavior of SIGUSR1.

4. Complete TODO1&TODO2 in `lab2-signal.c` to terminate the process immediately without any message once SIGUSR1 is received. Compile, and run it again in Terminal 1.

5. In Terminal 2, execute `kill -10 $(cat ./pid.txt)` again.

   After redefining SIGUSR1 handler : The default handler will be replaced to terminating process immediately and the dead loop will exit.

6. In Terminal 2, terminate the dead loop using `kill -10 $(cat ./pid.txt)`.

   Customize signal handler: Some useful handler behaviors are: 1)ignoring some control signals like SIGINT, 2) ignoring exception, like SIGFPE (float-point error) and continuing.

Here's a summary table of some common signals in Linux, along with their default actions: You can learn more from Linux manuals [Man7].

Signal	Action	Description
`SIGKILL (9)`	Terminate	Forcefully kills a process (non-redefinable).
`SIGTERM (15)`	Terminate	Termination signal, can be caught or ignored.
`SIGINT (2)`	Terminate	Interrupt from keyboard (Ctrl+C).
`SIGSEGV (11)`	Core dump	Invalid memory reference (Segmentation fault).
`SIGUSR1 (10)`	Terminate	User-defined signal 1, can be handled by the process.
`SIGSTOP (19)`	Stop	Stops process execution (non-redefinable).
`SIGCONT (18)`	Continue	Resumes a stopped process.

Table 1: Common Linux Signals and Their Actions

Note: Not all signals are redefinable, *e.g.*, `sigkill(9)` sends the `SIGKILL` signal, which cannot be caught, blocked, or ignored by a process. This ensures immediate and forceful termination without giving the process a chance to handle the signal or clean up resources. Unlike other signals, such as SIGTERM or SIGUSR1, which can be redefined or handled by the process, SIGKILL is non-redefinable, making it the signal of last resort when a process needs to be forcibly stopped..

Submission

(1 pt) Complete all TODO sections and submit your code as `lab2-signal_<your_student_id>.c`.

Appendix

```c
// filename: lab2-signal.c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void sigusrHandler(int sig_num)
{
    // TODO2: Terminate the program when the SIGUSR1 signal is caught. (~1 line)
}

int main()
{
    // TODO1: Set the signal handler for SIGUSR1 to the function sigusrHandler using the signal function. (~1 line)

    // Write pid to a local file named pid.txt
    FILE *fp;
    fp = fopen("pid.txt", "w");
    fprintf(fp, "%d", getpid());
    fclose(fp);

    /* An infinite loop. */
    while(1) {
        printf("Still running...\n");
        sleep(1);
    }

    return 0;
}
```