# Mini-Lab 2: Signal

COMP3230, The University of Hong Kong

Sept. 2023

## 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));
```

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.

Note: Not all signals are redefinable, *e.g.*, `sigkill(9)`.

## 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;
}
```