```
Worksheet - Signal
/* CPSC 457 (Winter 2019)
 * Week 5 - 2
 * Tutorial 1 and 2
 * Sina Keshvadi
 *
 * Notes: No error handling!
 */
============================================================
Linux Signals are:

Signal Name Number  Description
SIGHUP   1   Hangup (POSIX)
SIGINT   2   Terminal interrupt (ANSI)
SIGQUIT 3   Terminal quit (POSIX)
SIGILL   4   Illegal instruction (ANSI)
SIGTRAP 5   Trace trap (POSIX)
SIGIOT   6   IOT Trap (4.2 BSD)
SIGBUS   7   BUS error (4.2 BSD)
SIGFPE   8   Floating point exception (ANSI)
SIGKILL 9   Kill(can't be caught or ignored) (POSIX)
SIGUSR1 10  User defined signal 1 (POSIX)
SIGSEGV 11  Invalid memory segment access (ANSI)
SIGUSR2 12  User defined signal 2 (POSIX)
SIGPIPE 13  Write on a pipe with no reader, Broken pipe (POSIX)
SIGALRM 14  Alarm clock (POSIX)
SIGTERM 15  Termination (ANSI)
SIGSTKFLT   16  Stack fault
SIGCHLD 17  Child process has stopped or exited, changed (POSIX)
SIGCONT 18  Continue executing, if stopped (POSIX)
SIGSTOP 19  Stop executing(can't be caught or ignored) (POSIX)
SIGTSTP 20  Terminal stop signal (POSIX)
SIGTTIN 21  Background process trying to read, from TTY (POSIX)
SIGTTOU 22  Background process trying to write, to TTY (POSIX)
SIGURG  23  Urgent condition on socket (4.2 BSD)
SIGXCPU 24  CPU limit exceeded (4.2 BSD)
SIGXFSZ 25  File size limit exceeded (4.2 BSD)
SIGVTALRM   26  Virtual alarm clock (4.2 BSD)
SIGPROF 27  Profiling alarm clock (4.2 BSD)
SIGWINCH    28  Window size change (4.3 BSD, Sun)
SIGIO   29  I/O now possible (4.2 BSD)
SIGPWR  30  Power failure restart (System V)
============================================================

Signals are a way of sending simple messages to processes.
Most of these messages are already defined and can be found in <linux/signal.h>.

Signal is a notification sent to a process to notify it of some event.
Signal has an integer number that represents it, and symbolic name.

Example:
signal SIGALRM (numeral value is 14), caused (also) by alarm clock.
Tip: you can get a list with all available signals using  $kill –l  shell command (they will appear
without the prefix SIG).

============================================================
What can user do to send a signal to a process via command prompt (shell)?
 - use CTRL+C to send an SIGINT signal to the running process
 - use CTRL+Z to send a SIGTSTP signal to the running process, etc.
============================================================
In order to catch a signal, we should build a special function that will be executed when a signal
arrives.
Such a function is called a signal handler.

Example:
void catch_alarm (int sig_num) {
        printf ( “Operation time out. Exiting. \n”);
        exit (0);
}
============================================================
How would the operating system know that there is a specific signal handler handles a specific signal?
We should connect them, using signal() C library function:

    signal (SIGALRM, catch_alarm_&_setitimer);
============================================================
// Catch the SIGINT signal

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

void sig_handler(int signo)
{
  if (signo == SIGINT)
    printf("OUCH! - I got signal\n");
}

int main(void)
{
  if (signal(SIGINT, sig_handler) == SIG_ERR)
  printf("\ncan't catch SIGINT\n");

  while(1)
{
    printf("Hi \n");
    sleep(1);
}
  return 0;
}
============================================================
A function sig_handler is used as a signal handler.
This function is registered to the kernel by passing it as the second argument of the system call
‘signal’ in the main() function.
The first argument to the function ‘signal’ is the signal we intend the signal handler to handle which is
SIGINT in this case.
============================================================
// You can not catch SIGKILL (9) and SIGSTOP (19) signals.

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

void sig_handler(int signo)
{
    if (signo == SIGINT)
        printf("I can catch you :) \n");
    else if (signo == SIGKILL)
        printf("received SIGKILL\n");
    else if (signo == SIGSTOP)
        printf("received SIGSTOP\n");
}

int main(void)
{
    if (signal(SIGINT, sig_handler) == SIG_ERR)
        printf("\ncan't catch SIGKILL\n");
    if (signal(SIGKILL, sig_handler) == SIG_ERR)
        printf("\ncan't catch SIGKILL\n");
    if (signal(SIGSTOP, sig_handler) == SIG_ERR)
        printf("\ncan't catch SIGSTOP\n");
    // A long long wait so that we can easily issue a signal to this process
    while(1)
    sleep(1);
    return 0;
```

```
}

- run your code
- press ctrl+c (you can catch this signal)
- open another terminal
- run this command: ps -axu
- find above PID
- kill the process by:
kill -9 {PID}
or STOP it by
kill -19 {PID}

============================================================
Trap Command

trap defines and activates handlers to be run when the shell receives signals or other special conditions.

- trap 'cal' 2
now press ctrl + c

- trap
Display a list of the currently-set signal traps.

trap -l
Display a list of signal names and their corresponding numbers.

- trap '' 2
If the command listed for trap is null, the specified signal will be ignored when received.

- trap 2
it comes back to the default action
============================================================
//This is for next tutorial. using signals by threads

#include <sys/types.h>
#include <stdlib.h>
#include <signal.h>
#include <stdio.h>
#include <unistd.h>

static int alarm_fired = 0;
void ding(int sig)
{
    alarm_fired = 1;
}

int main ()
{
    pid_t pid;
    printf("alarm application starting \n");

    pid = fork();
    switch(pid){
        case -1:
            /*Failure*/
            perror("fork failed");
            exit(1);
        case 0:
            /*child*/
            sleep(5);
            kill(getppid(), SIGALRM);
            exit(0);
    }
    /*if we get here we are the parent process*/
    printf("waiting for alarm to go off \n");
    (void) signal(SIGALRM, ding);
    pause();
    if(alarm_fired) printf("Ding! \n");

    printf("done\n");
    exit(0);
}
============================================================
// Search in big array by using thread

#include <iostream>
#include <pthread.h>
#include<cstdlib>
using namespace std;

// Max size of array
#define max 100000000

// Max number of threads to create
#define thread_max 4

long a[max];
long key = 99999990;

// Flag to indicate if key is found in a[] or not.
long f = -1;

int current_thread = 0;

// Linear search function which will run for all the threads
void* ThreadSearch(void* args)
{
   int num = current_thread++;

   for (long i = num * (max / thread_max);
   i < ((num + 1) * (max / thread_max)); i++)
   {
      if (a[i] == key)
         f = i;
   }
}

int main()
{
   for (long j=0; j<max; j++)
      a[j] = j;

   pthread_t thread[thread_max];

   for (int i = 0; i < thread_max; i++) {
      pthread_create(&thread[i], NULL, ThreadSearch, (void*)NULL);
   }

   for (int i = 0; i < thread_max; i++) {
      pthread_join(thread[i], NULL);
   }

   if (f == -1)
      cout << "Key not present" << endl;
   else
      cout << "Key element found in index " <<f<< endl;
   return 0;
}
```