Operating Systems

Lab 02 Exercises – signals and thread creation and join

1. **Learning goals: this laboratory activity is useful to practice with signals, and to understand how to create threads using Pthreads creation.**

Exercise 1

Write a C program that generate a child. The parent opens a file given as argument in the command line, then loops forever, 1) reading each line, 2) printing the line number and the line content 3) rewinding the file. The child process sends a **`SIGUSR1`** signal to the parent at random intervals between **`1`** and **`10`** seconds. The first received signal makes the parent skip the print statement. It will restart printing after receiving the next **`SIGUSR1`** signal. This behavior is repeated for all the received **`SIGUSR1`** signals. After **`60`** seconds, the child must send a **`SIGUSR2`** signal to the parent, and then terminate. Receiving this signal, also the parent will terminate.

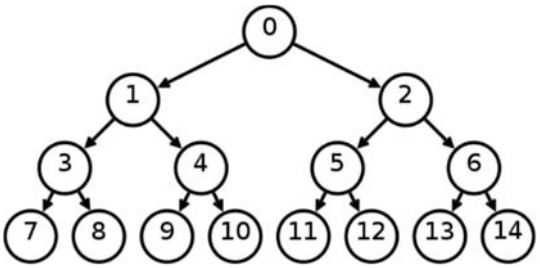
Hint: the child forks a process that sleeps 60 seconds then send to it a **`SIGUSR2`**, which the child catches and "forward" to the parent.

Exercise 2

Implement a C program, **`thread_generation_tree`**, which receives a command line parameter: **`n`**.
The main thread creates two other "children" threads and terminates.
Each child thread creates another two children threads, and terminates.

Thread creation stops after 2^n leave threads have been created.

For example, if **`n=3`**, the main thread creates two children, and each child creates another two threads for a total number of 8 leaves threads. At this point process creation stops.
Each leaf thread must print its generation tree, i.e., the sequence of thread identifiers from the main thread. Example:




```
> thread_generation_tree 3
Tread tree: 3077879488 3069483888 3052575600 3077876592
Tread tree: 3077879488 3069483888 3052575600 3035790192
Tread tree: 3077879488 3069483888 3060968304 3052575600
Tread tree: 3077879488 3069483888 3060968304 3027397488
Tread tree: 3077879488 3077876592 3044182896 3060968304
Tread tree: 3077879488 3077876592 3044182896 3019004784
Tread tree: 3077879488 3077876592 3069483888 3044182896
Tread tree: 3077879488 3077876592 3069483888 3010612080
```

Exercise 3

Implement a C program that

- takes from the command line two integer numbers **n1**, **n2,**
- allocates two vectors **v1** and **v2**, of dimensions **n1** and **n2**, respectively,
- fills **v1** with **n1** random even integer numbers between **10-100**,
- fills **v2** with **n2** random odd integer numbers between **21-101**,
- save the content of vectors **v1** and **v2** in two **binary** files **fv1.b** and **fv2.b**, respectively,

Use command **od** for verifying the content of files **fv1.b**, **fv2.b**.

You can use **Standard ANSI C**

```
FILE *fpw;         // file pointer
if ((fpw = fopen (filename, "wb")) == NULL){  // w for write, b for
binary
    fprinf(stderr," error open %s\n", filename);
    return(1);
}
fwrite(buffer,sizeof(buffer),1,  fpw);  //  write  sizeof(buffer)  bytes
from buffer
```

Or **Unix system calls**

```
#include <unistd.h>
#include <fcntl.h>

int fdo;           // file descriptor
if ((fdo = open(filename, O_CREAT | O_WRONLY, 0777)) < 0){
    fprinf(stderr," error open %s\n", filename);
    return 1;
}
write(fdo, buffer, sizeof(buffer));
```

Implement a concurrent program in C language, using Pthreads, which creates two client threads, then it acts as a server.
A client thread loops reading the next number from the binary file (**fv1.b** and **fv2.b**, respectively), and storing it in a global variable **g**. Then, it performs a signals on a semaphore to indicate to the server that the variable is ready, and it waits on a semaphore a signal from the server indicating that the number has been processed (simply multiplied by **3**), finally, it **prints the result and its identifier**.
The server loops waiting the signal of the clients, doing the multiplication, storing the results on the same global variable **g**, and signalling to the client that the string is ready to be printed.
The main thread waits the end on the threads, and prints the total number of served requests.