Operating Systems

Laboratory 5

Learning goals: In this laboratory activity, you will learn how to use `system` and `exec` system calls and signals. You will understand how to execute commands in your own programs.

Exercise 1

Write a C program, **using the system call `system`**, which takes as its argument a number **`k`**.

It must produce two text files, **`F1.txt`** and **`F2.txt`**, each including **`k`** random generated integer numbers, in range [**`0`** – **`1000000],`** one per line. Initialize the random seed with your ID.

It then generates a child. Parent and child must sort, in ascending order, **`F1.txt`** and **`F2.txt`**, respectively, using the shell command **`sort.`** The parent process must wait the end of its child.

(Command **`sort –n –o fname fname`** sorts in ascending order the content of **`fname`**, and by means of the **`–o`** option rewrites the content of file **`fname`** with the sorted numbers . Option **`–n`** indicates numeric rather than alphabetic ordering.)

Then, the parent process reads both files, **`F1.txt`** and **`F2.txt,`** and convert them to **`binary`** format, producing the files **`F1.bin`** and **`F2.bin`**.

Finally, the parent process will merge these sorted files, **properly reading one integer at a time from the two files**, and writing the smallest on the output file **`F12.sorted`**.

Before terminating, the parent process removes the files **`F1.bin`**, and **`F2.bin`**.

You cannot read in memory the content of the two files for sorting, but you can look at this function to inspire the final part of your main program. Function **`merge`** merges two sorted vectors, **`v1`** and **`v2`**, and produces the sorted vector **`v3`**.

```
void merge(int *, int *v2,  int *v3, int N1, int N2){
i=j=k=0;
  while(i<N1 && j<N2){
    if(v1[i] < v2[j])
      v3[k++] = v1[i++];
    else
      v3[k++] = v2[j++];
  }
  // Copy the remaining values of one of the two vectors
  while(i<N1)
    v3[k++] = v1[i++];
  while(j<N2)
    v3[k++] = v2[j++];
}
```

Exercise 2

Implement the solution for Exercise 1 using system call **`execlp`**. Can you solve it using a single child?

Exercise 3

Implement the solution for Exercise 1 using system call **`execv`**.

Exercise 4

Write a C program that generate a child. The parent opens a file given as an argument of 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 a random number of times [**`10-20]`** a **`SIGUSR1`** signal has been received, the next **`5 SIGUSR1`** signals must be ignored. Finally, and after receiving another **`SIGUSR1`** signal both parent and child must terminate.

Before its termination, the parent must print the number of **`SIGUSR1`** signals that have been sent.

Summary

At the end of this laboratory activity, you should have understood how to use **`system`** and **`exec`** system calls, and signals.