Lab Session 1: Processes
21653-Sistemes Operatius

1 Introduction

The objective of this seminar is to implement and understand process creation as we saw in class mainly using the commands: fork, exit, wait, exec. We will be programming in C. You will need to deliver only one file with the code of the program (also including the background option of section 3) and a short document (explained in section 5.1) in a zip file. We provide help files:

- readsplit.c : contains the function readsplit which reads from an open file or pipe or keyboard and returns when the split character is found and saves the split part into a buffer.
- computetime.c : example file on how to compute the elapsed milliseconds between
- folderutils.c : example file on how to read and create folders.

The methods "sprintf" and "sscanf" can also be used if necessary. To print on the screen or wait for keyboard input we will be using the system calls read and write. You can also use if wanted methods from the "string.h" header. Here follows an example of substring matching:

```
char *str1 = "Input string chain", *str2 = "cha", *ptr;
ptr = strstr(str1, str2);
printf("The substring is: %s\n", ptr);
```

Remember to edit your files in a ".c" file and compile using the line "gcc -o outputexec myfile.c".

2 Back up shell system

We will be creating a backup shell system. When the command runs if no parameter is given it prints a symbol ">" and waits for keyboard input. The shell can interpret two special commands "backup", "ls" and "sleep".

```
./timemachine
> backup c
...
> ls
...
```

- When "backup c" is introduced as input, it creates a folder (look at folderutils.c) that has as name the current hour-min-sec (look at computetime.c) and copies all the current folder files that end with extension "c" (to extract files look at folderutils.c) to the new directory. To copy files we will create a process and call execlp (that remember substitutes the code of a process) with the command "cp".
- When "ls" is introduced it creates a process and executes the command "ls" showing in the screen the result of "ls" and the parent process wait for it to finish with the system call "waitpid"
- When "sleep" is introduced it creates a process and executes the command "sleep 1" which waits for a second.

For example, to execute a code from the current process and execute the command: "ls -l /" just do:

```
execlp("ls", "ls", "-l", "/", (char *)NULL);
```

The input and output of the shell will be done with the system calls read and write. To help you processing the input and output we provide you with a sample program that reads lines from a file and writes them into the screen. This program is called readsplit.c.

3 Executing commands in parallel (also called in the background

We are now going to implement an additional feature which exists also in normal shells. When we include an ampersand (the symbol &) at the end of a command it is executed without waiting for it to finish; something that is called execution in background. If we write:

```
> backup pdf &
>
```

The prompt is returned immediately, the program executed in background and you can continue working. We are going to do the same in our shell, if we write:

```
> ls &
>
```

the execution of the backup procedure must be done in the background by creating a process that executes it.

4 Standard input execution

If the input of our shell is read with the standard input (channel 0), remember that we can also pass to our program a list of commands from a file (in the commands line of bash shell):

```
./timemachine < commands.txt
```

Being commands a file like:

```
backup pdf &
backup c
sleep 5
backup pdf &
backup c
```

5 Benchmarking the execution

We are going to add now to our shell a way of computing the ellapsed time of the execution. We have provided the file computetime.c to compute the elapsed milliseconds between two points of code. When our shell is launched, it can compute the total time elapsed in the execution: when starting and just before exit.

5.1 Document delivery

In the delivered document you have to explain what is the main structure of calling a series of commands in sequence or in parallel. Present a minimal piece of code that explains what are the main structure of the two schemas. Then relate it to your code.

Do some benchmarking of several executions of your command files, that include commands in background and discuss the results.