MYY502 – SYSTEMS PROGRAMMING

Lab #8
8/12/2015

I. PROCESSES

Create a "launcher"-type program, which shows a menu with application names and the user selects which one to execute each time. For example, use the following menu:

```
1. run firefox
2. run firefox (and wait)
3. run gedit
4. run gedit (and wait)
5. run gnome-calculator
6. run gnome-calculator (and wait)
0. exit
```

(a) **Operation**:
For the execution of each application, you should create a child process, (using `fork()`), which should then use `execl()` to execute the required application. For choices 2, 4 and 6, the parent process should wait till the required application finishes, while for choices 1, 3 and 5, the parent is free to continue its execution immediately and independently of the child.

(b) **Stucture**:
The menu should be implemented as an array of 7 `structs`. Each `struct` should contain:

- the text to display (e.g. "`2. run firefox (and wait)`")
- the full path of the corresponding application (e.g. "`/usr/bin/firefox`")
- the name of the corresponding application (e.g. "`firefox`")

so that the fields of the `struct` are used to display the menu choices as well as for passing the necessary arguments to `execl()`. To find the full path of an application, you can use the "`which`" terminal command.

II. PROCESSES AND SIMPLE PIPES

Design a program which:

(α) Creates a child process with which it will communicate through a pipe.

(β) The **child process** reads integers from the user and passes them to the parent process through the pipe.

(γ) The parent process checks the numbers it receives from the child and write the positive ones to a *text* file named "`positive.txt`".

The program ends when the user gives -1. *Make sure that the processes finish correctly and smoothly*, and check that everything is saved in `positive.txt` as it should.

III. PROCESSES AND ADVANCED PIPES

The `gnuplot` application is among the most popular ones when it comes to creating function or data plots of various kinds. On a terminal, if you execute:

```
$ gnuplot
```

`gnuplot` starts (the `gnuplot>` prompt is displayed), and waits for simple commands from the user to do the job. For example, if you type:

```
gnuplot> plot sin(x);
```

you will get a plot of the $\sin(x)$ function. Type:

```
gnuplot> help plot;
```

to get some help on the commands `gnuplot` understands. You can terminate the application with the `exit` command:

```
gnuplot> exit;
$
```

You are asked to implement a program which executes `gnuplot` in order to draw any plot you like. More specifically:

a) Your program should create a child process and a pipe in order to communicate with it.

b) The child process should use `execl()` to execute `gnuplot` and employ `dup()`-style functions so as to feed whatever it receives from the pipe to the standard input of `gnuplot`.

c) The parent should write the commands (for `gnuplot` to execute) to the pipe. At the end, the parent should wait for user input so as to send the `exit` command and terminate `gnuplot`.

IV. MORE PRACTICE WITH PIPES

Design a program which:

(a) Creates a child process and communicates with it using **two (2)** pipes—the parent writes to the first one and the child writes to the second one.

(b) The parent process reads integers from the user and passes them to the child, using the first pipe, until the user gives -1.

(c) The child process gets the numbers from the parent and calculates *their sum and their average* (the latter should be a `double`).

(d) In the end, the child process sends the sum and the average to the parent process, using the *second* pipe, The parent prints them and exits.

V. EVEN MORE PRACTICE WITH ADVANCED PIPES

Try to control other applications, in the spirit of Problem III. For example, practice with `matlab` or `octave`, if you know them.