Operating Systems

Grado en Informática 2022/2023

Lab Assignment 3: Processes

We continue to code the shell we started in the first lab assignment. We'll add the following commands. Check the supplied shell to check the workings and exact sintax for the commands. (You can use the help command, "command -?" or "command -help" to get help):

priority	views (or changes) the priority of a process
showvar	shows the value of an environment variable
changevar	changes the value of an environment variable
showenv	shows the process environment
fork	the shell does the fork system call and waits for its child to end
execute	executes a program (with arguments) without creating a new process
listjobs	lists background processes
deljobs	deletes background processes from the list
job	shows info on a background process. Can change it to foreground
*****	For anything that is not a shell command, the shell will asume it is an external program (in the **format for execution** described below) and will create a process to execute it

IMPORTANT:

We have to implement (same list type as we used before) a list of processes executing in the background. For each process we must show

- Its PID
- Date and time of launching
- Status (FINISHED, STOPPED, SIGNALED or ACTIVE) (with return value/signal when relevant)
- Its command line
- Its priority

The shell commands ***listjobs*** and ***deljobs***, show and manipulate that list. Only processes lauched from the shell to execute in the background will be added to the list.

Priority can be obtained at the time of printing. We insert processes in the list as ACTIVE, and use the **waitpid** system call to check for status changes.

Execution in foreground means the parent process **waits** for the child to finish before continuing.

Execution in the background means the parent continues to execute concurrently with the child: i**t does not wait** for its child to finsh.

******* FORMAT FOR EXECUTION**

The format for creating process that executes a program is

[VAR1 VAR2 .....] executable arg1 arg2.....[@pri] [&]

items inside brackes [] are optional

- **[VAR1 VAR2...]** if present mean that the program being executed has an environment consisting only of the variables VAR1, VAR2...(VAR1, VAR2 are the names of the variables, their values are taken from environ in the current process). The number of variables is undefined and they need not be capitalized
- **executable** is the name of the executable file to be executed
- **arg1 arg2 ...** are the arguments passed to the executable. (number of them is undefined)
- **[@pri]** if present means that the new program being executed has its priority set (with setpriority) to pri
- **[&]** if present means that execution is to be in the background
- This format, with the execption of the &, is the same to be used with the ***execute*** command to execute without creating process

Examples

`-> ls`

executes, creating a process in the foreground, the program ls

`-> ls -l -a /home`

executes, creating a process in the foreground, 'ls -l -a /home '

`-> TERM HOME DISPLAY ./a.out`

executes, creating a process in the foreground, './a.out' in an environ that only contains the variables TERM HOME DISPLAY

`-> xterm -fg yellow -e /bin/bash @15`

executes, creating a process in the foreground, 'xterm -fg yellow -e /bin/bash' having its priority set to 15

-> **TERM HOME DISPLAY USER xterm -fg green -bg black -e /usr/local/bin/ksh @12 &**

executes creating a process in the background 'xterm -fg green -bg black -e /usr/local/bin/ksh' with its priority set to 12 and in an environ that contains only variables TERM HOME DISPLAY and USER

`-> execute xterm -fg white @10`

executes without creating a new process 'xterm -fg white' with its priority set to 10

To execute programs in the system's PATH with its environment changed we can use the **execvpe** function. This function only exists in linux and requires that we define _GNU_SOURCE. Alternatively we could use the function **OurExecvpe** provided in the ayudaP3.c file, which relies on the standard **execve** system call.

REMEMBER:

- Information on the system calls and library functions needed to code these programs is available through man: fork, exec, waitpid, getenv, putenv, getpriority, setpriority....

- A reference shell is provided (for various platforms) for students to check how the shell should perform. This program should be checked to find out the syntax and behaviour of the various commands. **PLEASE DOWNLOAD THE LATEST VERSION**
- The program should compile cleanly (produce no warnings even when compiling with **gcc -Wall)**
- These programs can have no memory leaks (you can use valgrind to check)
- When the program cannot perform its task (for whatever reason, for example, lack of privileges) it should inform the user
- All input and output is done through the standard input and output
- Errors should be trated as in the previous lab assignments

WORK SUBMISSION

- Work must be done in pairs.
- Moodle will be used to submit the source code: a zipfile containing a directory named P2 where all the source files of the lab assignment reside
- The name of the main program will be p3.c, Program must be able to be compiled with gcc p3.c, Optionally a Makefile can be supplied so that all of the source code can be compiled with just make. Should that be the case, the compiled program should be called p3
- ONLY ONE OF THE MEMBERS OF THE GROUP will submit the source code. The names and logins of all the members of the group should appear in the source code of the main programs (at the top of the file)
- Works submited not conforming to these rules will be disregarded.
- DEADLINE: 23:00, **THURSDAY December the 15th, 2022**