Operating Systems

Grado en Informática. Course 2021-2022

OPTIONAL Lab Assignment

THIS LAB ASSIGNMENT IS OPTIONAL: the maximum score can be reached without doing it, with just the previous three lab assigments

CONTINUE the coding of the shell started in previous lab assigments. In this lab assignment we'll add to the shell the capability to execute external programs both in foreground and background and without creating process **with a reduced environment**. The shell will keep track (using a list) of the processes created in background with the reduced environment as it did before.

MODIFY the following shell commands, that is, the ones that execute a program with its arguments (without process, foreground process or background process) so that we can have execution with a reduced enviroment. The new syntax is

ejec VAR1 VAR2 VAR3 ... prog arg1 arg2 ... Executes program *prog* with its args in an environment that contains ONLY environment variables VAR, VAR2 VAR2 .... If no variables are given it executes prog with its arguments, exactly as the previous lab assignment. Execution is without creating process.

ejecas us VAR1 VAR2 VAR3 ... prog arg1 arg2 ... Executes, as user *us*, program *prog* with its args in an environment that contains ONLY environment variables VAR, VAR2 VAR2 .... If no variables are given it executes prog with its arguments, exactly as the previous lab assignment. Execution is without creating process.

ejecpri prio VAR1 VAR2 VAR3 ... prog arg1 arg2 ... Executes, changing the priority to *prio*, program *prog* with its args in an environment that contains ONLY environment variables VAR, VAR2 VAR2 .... If no variables are given it executes prog with its arguments, exactly as the previous lab assignment. Execution is without creating process.

fg VAR1 VAR2 VAR3 ... prog arg1 arg2 ... Executes program *prog* with its args in an environment that contains ONLY environment variables VAR, VAR2 VAR2 .... If no variables are given it executes prog with its arguments, exactly as the previous lab assignment. Execution is in a new process in the foregound.

fgas us VAR1 VAR2 VAR3 ... prog arg1 arg2 ... Executes, as user

us, program *prog* with its args in an environment that contains ONLY environment variables VAR, VAR2 VAR2 .... If no variables are given it executes prog with its arguments, exactly as the previous lab assignment. Execution is in a new process in the foregound.

fgpri **prio VAR1 VAR2 VAR3 ... prog arg1 arg2 ...** Executes, changing the priority to *prio*, program *prog* with its args in an environment that contains ONLY environment variables VAR, VAR2 VAR2 .... If no variables are given it executes prog with its arguments, exactly as the previous lab assignment. Execution is in a new process in the foregound.

back **VAR1 VAR2 VAR3 ... prog arg1 arg2 ...** Executes program *prog* with its args in an environment that contains ONLY environment variables VAR, VAR2 VAR2 .... If no variables are given it executes prog with its arguments, exactly as the previous lab assignment. Execution is in a new process in the background and the process should be added to the list of processes in the background.

backas **us VAR1 VAR2 VAR3 ... prog arg1 arg2 ...** Executes, as user *us*, program *prog* with its args in an environment that contains ONLY environment variables VAR, VAR2 VAR2 .... If no variables are given it executes prog with its arguments, exactly as the previous lab assignment. Execution is in a new process in the background and the process should be added to the list of processes in the background.

backpri **prio VAR1 VAR2 VAR3 ... prog arg1 arg2 ...** Executes, changing the priority to *prio*, program *prog* with its args in an environment that contains ONLY environment variables VAR, VAR2 VAR2 .... If no variables are given it executes prog with its arguments, exactly as the previous lab assignment.Execution is in a new process in the background and the process should be added to the list of processes in the background.

- **VAR1 VAR2 VAR3 ... prog arg1 arg2 ...** [**&**] Executes program *prog* with its args in an environment that contains ONLY environment variables VAR, VAR2 VAR2 .... If no variables are given it executes prog with its arguments, exactly as the previous lab assignment. Execution is in a new process in the foreground, unless the symbol '&' appears, in which case is in background and the process should be added to the list of processes in the background.
- **Notes on the syntax, applicable to all commands**
  - Only the variable names are given (the value is obtained from *environ*)
  - if first variable name is `NULLENV` the program is executed with an

empty environment.

Examples

```
#) fg TERM DISPLAY HOME ./shell
#) fgas visita TERM DISPLAY HOME ./shell
#) backpri 15 TERM DISPLAY HOME USER LANG xterm -fg green -e /bin/bash
#) ejec NULLENV ./shell
```

Information on the system calls and library functions needed to code this program is available through `man`: (*setpriority, getpriority, fork, execvpe, execvp, waitpid . . .)*.

- Work must be done in pairs.
- The source code will be submitted using moodle in a **ZIP FILE**, *p4.zip* containing a directory **P4** where all the files reside
- A `Makefile` must be supplied so that the program can be compiled with just `make`. The executable produced must be named shell
- Only one of the members of the workgroup will submit the source code. The names and logins of all the members of the group should be in the source code of the main program (at the top of the file)

DEADLINE: December 10 , 2021, 23:00

Assesment: During lab hours

NOTES ON EXECUTING WITH AN ALTERNATE ENVIRONMENT:

To execute with an alternate environment we use

```
int execvpe(const char *file, char *const argv[], char *const envp[])
```

which is excatly the same as *execvp* except that it requires a third parameter, a NULL terminated array of pointers, *envp* with the environment to be passed to the program we are going to execute (`man execvpe` for the details).

It is the programmers task to construct this array by getting the variables from *envirom* (the previously given function `BuscarVariable` might come in handy). For practical purposes we can asume that the environmmet can have up to 128 variables.

The *execvpe* libray function exists only in linux and we have to define `_GNU_SOURCE`. If we are programming on another system, or we don't want to define `_GNU_SOURCE`

in our code (for the sake of portability) we can make our own version

```
int OurExecvpe(const char *file, char *const argv[], char *const envp[])
```

using the *execve()* system call

```
char * Ejecutable (char *s)
{
  char path[MAXPATHLENGTH];
  static char aux2[MAXNAMELEN];
  struct stat st;
  char *p;
  if (s==NULL || (p=getenv("PATH"))==NULL)
      return s;
  if (s[0]=='/' || !strncmp (s,"./",2) || !strncmp (s,"../",3))
      return s;       /*is an absolute pathname*/
  strncpy (path, p, MAXPATHLENGTH);
  for (p=strtok(path,":"); p!=NULL; p=strtok(NULL,":")){
      sprintf (aux2,"%s/%s",p,s);
      if (lstat(aux2,&st)!=-1)
         return aux2;
  }
  return s;
}

int OurExecvpe(const char *file, char *const argv[], char *const envp[])
{
    return (execve(Ejecutable(file),argv, envp);
}
```

Please note that if the previously lab assignment (lab assignment 3) was correctly programmed, adding this new functionality does not require more than 25 o 30 lines of code, (50 or 60 if we choose to use `OurExecvpe()`