Operating Systems

Grado en Informática 2024/2025

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 (refference shell) for the workings and exact syntax for the commands**. (You can use the help command, "command -?" or "command -help" to get help):

getuid		views the process's credentials (real and efective)
setuid	**[-l] id**	set the process efective credential (-l for login)
showvar	**v1 v2 ...**	shows the value and address of environment variables v1 v2 .... access must be by main() third argument, environ and library function getenv
changevar	**[-a|-e|-p] var val**	changes the value of an environment variable. A new variable might be created ONLY when accesing with putenv (-p)
subsvar	**[-a|-e] v1 v2 val**	changes one environmet (v1) variable for other (v2 with value val)
environ	**[-environ|-addr]**	shows the process environment
fork		the shell does the fork system call and waits for its child to end
search		shows or modifies the search list (the list of directories where the shell looks for executables)
	-add dir	adds a dirrectory to the search list
	-del dir	deletes a directory from the search list
	-clear	clears the search list
	-path	imports de directories in the PATH to the seach list
exec	***progspec***	executes, without creating a new process, the program described by *progspec* (see explanation below).
exec-pri	**prio *progspec***	executes, without creating a new process and with its priority changed to prio, a program described by *progspec* (see explanation below).
fg	***progspec***	creates a process tha executes in the foreground the program described by *progspec (see explanation below).*
fgpri	**prio *progspec***	creates a process that executes in the foreground, with its priority changed to prio, the program described by *progspec (see explanation below).*
back	***progspec***	creates a process tha executes in the background the program described by *progspec (see explanation below).*
backpri	**prio *progspec***	creates a process tha executes in the background, with its priority changed to prio, the program described by *progspec (see explanation below).*
listjobs		lists background processes (executed with *back* or *backpri*)
deljobs	**-term|-sig**	deletes background processes from the list
*****		For anything that is not a shell command, the shell will asume it is an

external program (in the **format for execution** described below). This is equivalent to fg *progspec* with **** being *progspec*

IMPORTANT TOPICS:

I-LISTS

We have to implement (list implementation free) two lists: a list of processes executing in the background and a list of directories where the shell looks for executable files: the search list

1) list of processes executing in the background (processes created with the shell commands *back* or *backpri*). 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 ***jobs*** and ***deljobs***, show and manipulate that list. Only processes launched from the shell to execute in the background (with commands *back* or *backpri)* will be added to the list.

We insert processes in the list as ACTIVE. We should update, with the *waitpid* system call, the status of processes before printing. Note that the waitpid system call reports status changes, not the state itself, in fact, it only reports once a process has finished.

```
pid_t waitpid (pid_t pid, int * wstatus, int options);
```

The parameter *wstatus* ONLY has a meaninful value in the case waitpid returs the pid.

Priority (as it can change) should be obtained at the time of printing so it is not necessary to store it it the list.

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.

2) list of directories where the shell looks for executable files.

This is a simple list of directories where the shell will look for executable files to execute. It is analog to the PATH for the system's shell, except that we implement it with a list and not in an environment variable as the system'ss shell. The system call *execvp()* searches for executables in the list of directories contained in the PATH environment variable, we are not using that system call but *execve()* instead which doesn't look in the directories in the PATH and allows us to pass an alternate environment.

The Ejecutable function in the table below (also avalilable in the a*yudaP3-25.c.txt* file), takes care of finding an executable in the search list (provided we have implemented the SearchListFirst(), and SearchListNext() funcions to get the directories in our searchlist

.

```
char * Ejecutable (char *s)
{
        static char path[MAXNAME];
        struct stat st;
        char *p;

        if (s==NULL || (p=SearchListFirst())==NULL)
                return s;
        if (s[0]=='/' || !strncmp (s,"./",2) || !strncmp (s,"../",3))
                return s;        /*s is an absolute pathname*/

        strncpy (path, p, MAXNAME-1);strncat (path,"/",MAXNAME-1); strncat(path,s,MAXNAME-1);
        if (lstat(path,&st)!=-1)
                return path;
        while ((p=SearchListNext())!=NULL){
            strncpy (path, p, MAXNAME-1);strncat (path,"/",MAXNAME-1); strncat(path,s,MAXNAME-1);
            if (lstat(path,&st)!=-1)
                   return path;
        }
        return s;
}
```

With that in mind, we could easily construct a function that deals with executing a program changing (or not) the environment and changing (or not) the priority.

```
int Execpve(char *tr[], char **NewEnv, int * pprio)
{
    char *p;                /*NewEnv contains the address of the new environment*/
                            /*pprio the address of the new priority*/
                            /*NULL indicates no change in environment and/or priority*/
    if (tr[0]==NULL || (p=Ejecutable(tr[0]))==NULL){
        errno=EFAULT;
        return-1;
        }
    if (pprio !=NULL  && setpriority(PRIO_PROCESS,getpid(),*pprio)==-1 && errno){
                        printf ("Imposible cambiar prioridad: %s\n",strerror(errno));
                        return -1;
        }

        if (NewEnv==NULL)
                return execv (p,tr);
        else
                return execve (p, tr, NewEnv);
}
```

II-progspec: format for execution

The format for creating process that executes a program is

[VAR1 VAR2 VAR3 ....] executablefile [arg1 arg2.....]

items inside brackes [] are optional

- **executablefile** is the name of the executable file to be executed
- **arg1 arg2 ...** are the arguments passed to the executable. (number of them is undefined)
- **VAR1 VAR2 VAR3 ...**if present, means that execution is to be with an environment consisting only of the variables VAR1, VAR2, VAR3 (their values to be taken from *environ)*

The rule is simple: the first name that is not an environment variable (found in environ) is the executable file

Examples

`-> ls`

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

`-> fg ls -l -a /home`

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

`-> fgpri 12 xterm -fg yellow -e /bin/bash`

executes, creating a process in the foreground, 'xterm -fg yellow -e /bin/bash' with its priority set to 12

-> **back xterm -fg green -bg black -e /usr/local/bin/ksh**

executes creating a process in the background 'xterm -fg green -bg black -e /usr/local/bin/ksh'

`-> execpri 9 xterm -fg white`

executes without creating a new process 'xterm -fg white' with its priority set to 9

`-> fgpri 12 TERM HOME DISPLAY xterm -fg yellow -e /bin/bash`

executes, creating a process in the foreground, 'xterm -fg yellow -e /bin/bash' with its priority set to 12 in an environment that only contains environment variables TERM, HOME and DISPLAY. Their values are taken from environ.

`-> execpri 9 TERM HOME DISPLAY xterm -fg white`

executes without creating a new process 'xterm -fg white' with its priority set to 9 in an environment that only contains environment variables TERM, HOME and DISPLAY. Their values are taken from environ.

`-> TERM HOME DISPLAY xterm -fg yellow -e /bin/bash`

executes, creating a process in the foreground, 'xterm -fg yellow -e /bin/bash' in an environment that only contains environment variables TERM, HOME and DISPLAY. Their values are taken from environ.

Although the reference shell allows also to specify **&** for background execution, **@** for priority changes and implements redirection. THERE'S NO NEED to do add that functionality to this lab assignment.

III-CREDENTIALS

We'll use the *setuid* systemt call to change the user credential. Under normal circunstances the real and effective credentials will be the same, so no changes in the proccess credentials are allowed. To check this part of the lab assigment we'll have to

- Give the executable the setuid execution bit rwsr-xr-x (4755)

- Have it executed by another user so that the real credential is that of the user executing the file and the saved and efective credentials are those of the owner of the file. (It's a good idea to place the executable file on a writable directory for both of those users: `/tmp`.)
- The *setuid* system calls behave direrently for the `root` user, so **we must not use that account to check this part of the lab assigment**

REMEMBER:

- Information on the system calls and library functions needed to code these programs is available through man: fork, exec, waitpid, getenv, putenv, getpriority,....
- 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 treated as in the previous lab assignments
- An additional C file is provided with some useful functions

WORK SUBMISSION

- Work must be done in pairs.
- Moodle will be used to submit the source code: a zipfile containing a directory named P3 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, **Saturday December the 14th, 2024**