Built your own Shell

Project 1

Sergi Fígols

Introduction - Basic Features

The shell implemented runs continuously, displaying a prompt and reading a line one at a time.

As it is mentioned in the statement, first of all we look if the command is built in, and then if it is not, we create a child process that runs execvp. An error is displayed if execvp does not find the entered file.

There is a background execution permitted, where the parent process does not wait the child to end.

Built in Commands

In this implementation I have used the system call indicated in each case:

- ic: getcwd()
- cd: chdir()
- cm: chmod()
- co: chown()

In the implementation, we define an array of chars containing the built in commands, and an array of functions containing its functions. These functions are sorted the same way so that we can execute them in the execute section.

```c
// definition of built in commands
char *built_ins[] = {
    "ic",  // Print current dir
    "cd",  // Change directory
    "cm",  // Change the file's permissions into given mode
    "co",  // Change owner giving user and file path
    "surt" // Exits Shell
};

typedef void (*function_t)(char**);

function_t built_in_functions[BUILT_IN_COMMANDS] = {
    &get_actual_path,
    &change_dir,
    &change_mode,
    &change_owner,
    &surt
};
```

```c
/* ----------- BUILT IN SECTION ------------- */
for (int i = 0; i < BUILT_IN_COMMANDS; i++) {

    if (strcmp(args[0], built_ins[i]) == 0) { // args[0] == any_builtin
        built_in_functions[i](args);
        printf("\n");
        return 0;
    }
}
```

Not Built in Commands

First of all, a child is created. After that if there is the '&' symbol we quit it as it does not interfere in the execution. Then we run execvp, and if no command is found the error is printed, and child ends.
Secondly, the parent looks if there is an '&' in the argument. If it is there is no need to wait the child.

```
    /* ---------- NOT BUILT IN SECTION ----------- */
    int pid = fork();

    if (pid<0) {
        printf("Fork Failed.\n");
    }

    else if (pid == 0) {
        if (strcmp(args[get_last_arg(args)], "&") == 0) {
            args[get_last_arg(args)] = NULL;
        }

        if (execvp(args[0], args) < 0) {
            perror("Error");
            exit(1);
        }

    }

    else {
        // In case there is not background execution
        if (strcmp(args[get_last_arg(args)], "&") != 0) {
            wait(NULL);
        }
    }
    return 0;
}
```

Captures

In this slide it can be seen some built in commands execution. First of all we can see that cd works as planed and printing an error when it occurs. After that I show the cm execution and some error displaying.

```
sergi@/home/sergi/term/SO/shell> % cd

sergi@/home/sergi> % cd term/SO/shell

sergi@/home/sergi/term/SO/shell> % cd hola
chdir() error: Not a directory

sergi@/home/sergi/term/SO/shell> %
```

```
sergi@/home/sergi/term/SO/shell> % ic
/home/sergi/term/SO/shell
sergi@/home/sergi/term/SO/shell> % cm simple_shell.c 000

sergi@/home/sergi/term/SO/shell> % ls -l
total 948
-rw-r--r-- 1 root  root      20 de des.  12 13:37 '&~'
-rwxrwxr-x 1 sergi sergi  16464 de des.  12 13:47  hola
-rw-rw-r-- 1 sergi sergi     63 de des.  12 13:46  hola.c
-rwxrwxrwx 1 sergi sergi 912920 de des.  11 10:55  Prj1.pdf
-rwxrwxr-x 1 sergi sergi  18240 de des.  12 22:27  shell
---------- 1 sergi sergi   5176 de des.  12 22:27  simple_shell.c
sergi@/home/sergi/term/SO/shell> % cm simple_shell.c 666

sergi@/home/sergi/term/SO/shell> % ls -l
total 948
-rw-r--r-- 1 root  root      20 de des.  12 13:37 '&~'
-rwxrwxr-x 1 sergi sergi  16464 de des.  12 13:47  hola
-rw-rw-r-- 1 sergi sergi     63 de des.  12 13:46  hola.c
-rwxrwxrwx 1 sergi sergi 912920 de des.  11 10:55  Prj1.pdf
-rwxrwxr-x 1 sergi sergi  18240 de des.  12 22:27  shell
-rw-rw-rw- 1 sergi sergi   5176 de des.  12 22:27  simple_shell.c
sergi@/home/sergi/term/SO/shell> %
```

```
ergi@/home/sergi/term/SO/shell> % cm
hmod() error: Bad address

ergi@/home/sergi/term/SO/shell> % cm test
hmod() error: No such file or directory
```

Captures 2

Here we can see some testing with co. First of all we can see that the user is changed. Then if user does not exist or if there are missing parameters an error is printed. In case that there is no user name added, the command does not change anything as its uid_t variable it stays -1.

```
sergi@/home/sergi/term/SO/shell> % co simple_shell.c newuser

sergi@/home/sergi/term/SO/shell> % ls -l
total 948
-rw-r--r-- 1 root    root      20 de des.  12 13:37 '&~'
-rwxrwxr-x 1 sergi   sergi  16464 de des.  12 13:47  hola
-rw-rw-r-- 1 sergi   sergi     63 de des.  12 13:46  hola.c
-rwxrwxrwx 1 sergi   sergi 912920 de des.  11 10:55  Prj1.pdf
-rwxrwxr-x 1 sergi   sergi  18240 de des.  12 22:27  shell
---------- 1 newuser sergi   5176 de des.  12 22:27  simple_shell.c
sergi@/home/sergi/term/SO/shell> % co simple_shell.c hola
User not found.
sergi@/home/sergi/term/SO/shell> % co
chown() error: Bad address

sergi@/home/sergi/term/SO/shell> % co simple_shell.c

sergi@/home/sergi/term/SO/shell> % ls -l
total 948
-rw-r--r-- 1 root    root      20 de des.  12 13:37 '&~'
-rwxrwxr-x 1 sergi   sergi  16464 de des.  12 13:47  hola
-rw-rw-r-- 1 sergi   sergi     63 de des.  12 13:46  hola.c
-rwxrwxrwx 1 sergi   sergi 912920 de des.  11 10:55  Prj1.pdf
-rwxrwxr-x 1 sergi   sergi  18240 de des.  12 22:27  shell
---------- 1 newuser sergi   5176 de des.  12 22:27  simple_shell.c
sergi@/home/sergi/term/SO/shell> %
```

Captures 3

With execvp() we can execute the files staying in any path of the environment path variable, as shown with ls in previous slides.

As we can see, background execution is implemented correctly too. The hola executable is not ending because it is an infinite loop, we can run it in background too.

And finally, as we can see the prompt displaying, shows the user name actually working, with the actual path.

```
sergi@/home/sergi/term/SO/shell> % emacs hola.c&
sergi@/home/sergi/term/SO/shell> % emacs hola.c &
sergi@/home/sergi/term/SO/shell> % ./hola
```

```
int main(void) {
  while(1);
  return 0;
}
-:---  hola.c          All
Welcome to GNU Emacs, one c
```

Conclusion

To conclude this project, I think that it is a great way to understand how a shell is implemented. I think that it is great to do some projects individually as in most of subjects it is not like this. The part that I got stucked the most where in the cm and co commands, where a bit of research needed to be done for example, know how to find a user id with its name or how to work with the mode bits in a mode_t variable. A structure I did not now and it would have been good to know before is the passwd structure, which gives you a lot of information about a user.