Sistemas Operativos (75.08): Lab Shell

Matias Rozanec (#97404)
rozanecm@gmail.com

Abril 2018

Índice

Parte I
Resolución

1. Parte 1

Para resolver la primera parte de este Lab, se alteraron las líneas correspondientes al caso de ejecución de comandos en el switch de la función `void exec_cmd(struct cmd* cmd)` del archivo `exec.c` (líneas 6 a 16 del fragmento de código recuadrado).

Para lograr invocar programas y permitir pasarles argumentos, lo primero que se hizo fue castear el `struct cmd* cmd` a `(struct execcmd*) full_cmd`; de esta forma logramos tener acceso de forma directa a todos los datos necesarios del comando recibido. Por último, invocamos `execvp` pasándole como primer argumento `argv[0]`, o sea el nombre del programa a ejecutar, y como segundo argumento `argv`, o sea toda la lista de argumentos.

Para poder expandir variables de entorno, lo que primero que se hizo fue trabajar sobre la detección de las mismas. Esto se implementó leyendo el primer caracter de cada uno de los argumentos. En caso de que el caracter coincida con `$`, se trata de una variable de entorno, por lo que se procede a su expansión mediante `getenv()`. A dicha función hay que pasarle el argumento correspondiente pero sin el caracter `$`, por lo que antes de realizar la llamada a la función, se altera el `char *` de forma que apunte al segundo char del argumento en cuestión, quedando así el argumento listo para ser procesado por `getenv()`.

```
1  void exec_cmd(struct cmd* cmd) {
2
3      switch (cmd->type) {
4
5          case EXEC: {
6              // spawns a command
7              struct execcmd* full_cmd = (struct execcmd*)cmd;
8              for(int i = 0; i < full_cmd->argc; ++i){
9                  if(full_cmd->argv[i][0] == '$'){
10                     full_cmd->argv[i] += sizeof(char);
11                     full_cmd->argv[i] = getenv(full_cmd->argv[i]);
12                 }
13             }
14             execvp(full_cmd->argv[0], full_cmd->argv);
15             break;
16         }
17
18         case BACK: {
19             // runs a command in background
20             //
21             // Your code here
22             printf("Background process are not yet implemented\n");
23             _exit(-1);
24             break;
25         }
26
27         case REDIR: {
28             // changes the input/output/stderr flow
29             //
30             // Your code here
31             printf("Redirections are not yet implemented\n");
32             _exit(-1);
33             break;
34         }
35
36         case PIPE: {
37             // pipes two commands
38             //
39             // Your code here
40             printf("Pipes are not yet implemented\n");
41
42             // free the memory allocated
43             // for the pipe tree structure
44             free_command(parsed_pipe);
45
```

```
46                 break;
47             }
48         }
49 }
```