ESCOM-IPN

Proyecto Final
Minibash

Sistemas Operativos

Ivan Aldavera Gallaga
Laura Andrea Morales López
Erick Francisco Vázquez Nuñez

Noviembre 2019

Índice

1. Objetivo

Diseñar y desarrollar, en lenguaje C y en un sistema operativo basado en UNIX, un programa que funcione como un interprete de comandos (minishell). En la realización de éste programa se verán reflejados conocimientos de comunicación entre procesos y de llamadas al sistema como fork, exec, pipe, dup.

2. Introducción

Shell script es un intérprete de comandos de UNIX, provee una interfaz entre el usuario y el kernel para que se ocupen funciones del sistema.

El objetivo de cualquier intérprete de comandos es ejecutar los programas que el usuario teclea en el prompt del mismo. El prompt es una indicación que muestra el intérprete para anunciar que espera una orden del usuario. Cuando el usuario escribe una orden, el intérprete ejecuta dicha orden.

3. Desarrollo

La manera en que procesaremos los comandos ingresados en nuestro minibash es de la siguiente manera.

- Imprimir el directorio actual.
- Obtienes la cadena
- Dividimos la cadena de entrada en comandos
- Comprobamos si existen tuberías, direccionamiento a archivos, etc.
- Si hay tuberías hay que manejarlas.
- Ejecutamos los comandos del sistema llamando a execvp.

4. Resultados

5. Conclusiones

En este proyecto el manejo de tuberías es la parte más complicada del mismo, el uso de execvp es el que nos permitió realizar la funcionalidad de los comandos solicitados, y

cortar la cadena de entrada nos permite analizar comando por comando en la entrada, para los archivos se realizo un manejo de entrada y salida de los comandos.

Generar este proyecto nos permitió ver como podemos usar funciones para llamar al sistema y manipular la visualización de la terminal para que tenga la apariencia que deseas.

Anexos

```c
/*═══════════════════════════════════════════════════════════════════════*/
/*═══════════════ INSTITUTO POLITÉCNICO NACIONAL ═══════════════════════*/
/*═══════════════ ESCUELA SUPERIOR DE CÓMPUTO ══════════════════════════*/
/*═══════════════════════════════════════════════════════════════════════*/
/*═══════════════ Ivan Aldavera Gallaga ═══════════════════════════════*/
/*═══════════════ Erick Francisco Vázquez Nuñez ═══════════════════════*/
/*═══════════════ Laura Andrea Morales López ══════════════════════════*/
/*═══════════════════════════════════════════════════════════════════════*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <readline/readline.h>
#include <readline/history.h>
#include <fcntl.h>

#define MAXCOM 1000                                                    // Número máximo de letras
#define MAXLIST 100                                                    // Número máximo de comandos
#define TAM 60


//═══════════════════════ Mensaje de bienvenida ═══════════════════════
void init_shell()
{
    printf("\n\n\n\n═══════════════════════════════════════════════════");
    printf("\n\n\n\t****MINIBASH****");
    printf("\n\n\n\n═══════════════════════════════════════════════════");

    char* username = getenv("USER");
    printf("\n");
    sleep(1);
}
//═══════════════════════ Manejo de archivos "<" y " >" ═══════════════
void redirSal(char cad[TAM]){
    char *cadPtr;
    cadPtr=cad;                                                        //puntero a la cadena
    close(1);                                                          //cerramos la salida estándar
    open(cadPtr,O_CREAT | O_WRONLY,0777);                              //Se asigna la salida al fichero, tambien se le asignan permisos totales

}
void redirEnt(char cad[TAM]){
    char *cadPtr;
    int f;
    cadPtr = cad;                                                      //puntero a la cadena
    f=open(cadPtr,O_RDONLY);                                           // se asigna la salida al fichero
    close(0);                                                          //cerramos la salida estándar
    dup(f);
}
//═══════════════════════ Lectura de entrada ══════════════════════════

int takeInput(char* str)
{
    char* buf;
```

```
58      buf = readline(">>> ");
59      if (strlen(buf) != 0) {
60          add_history(buf);
61          strcpy(str, buf);
62          return 0;
63      } else {
64          return 1;
65      }
66  }
67  //========================= Imprime la dirección ==========================
68
69  void printDir()
70  {
71      char cwd[1024];
72      getcwd(cwd, sizeof(cwd));
73      printf("\nDir: %s", cwd);
74  }
75  //========================= Ejecuta comandos ==========================
76
77
78  void execArgs(char** parsed)
79  {
80
81      pid_t pid = fork();
82      if (pid == -1) {
83          printf("\nFailed forking child..");
84          return;
85      } else if (pid == 0) {
86          if (execvp(parsed[0], parsed) < 0) {
87              printf("\nCould not execute command..");
88          }
89          exit(0);
90      } else {                                                // Esperando al hijo
91          wait(NULL);
92          return;
93      }
94  }
95  //========================= Ejecuta comandos en tuberías ==========================
96
97  void execArgsPiped(char** parsed, char** parsedpipe)
98  {
99      int pipefd[2];
100     pid_t p1, p2;
101     if (pipe(pipefd) < 0) {
102         printf("\nPipe could not be initialized");
103         return;
104     }
105     p1 = fork();
106     if (p1 < 0) {
107         printf("\nCould not fork");
108         return;
109     }
110     if (p1 == 0) {
111                                                             // Hijo 1
112         close(pipefd[0]);
113         dup2(pipefd[1], STDOUT_FILENO);
114         close(pipefd[1]);
115         if (execvp(parsed[0], parsed) < 0) {
116             printf("\nCould not execute command 1..");
117             exit(0);
118         }
119
120     } else {
121                                                             //Código del padre
122         p2 = fork();
123         if (p2 < 0) {
124             printf("\nCould not fork");
125             return;
126         }
127                                                             // Hijo 2
128
129         if (p2 == 0) {
130             close(pipefd[1]);
131             dup2(pipefd[0], STDIN_FILENO);
132             close(pipefd[0]);
133             if (execvp(parsedpipe[0], parsedpipe) < 0) {
134                 printf("\nCould not execute command 2..");
135                 exit(0);
136             }
137         } else {
138                                                             //Espera a los hijos
139             wait(NULL);
140             wait(NULL);
141         }
142     }
```

```
}
//========================= Manejo de tuberias ==========================
void pipeline(char ***cmd)
{
    int fd[2];
    pid_t pid;
    int fdd = 0;

    while (*cmd != NULL) {
        pipe(fd);
        if ((pid = fork()) == -1) {
            perror("fork");
            exit(1);
        }
        else if (pid == 0) {
            dup2(fdd, 0);
            if (*(cmd + 1) != NULL) {
                dup2(fd[1], 1);
            }
            close(fd[0]);
            execvp((*cmd)[0], *cmd);
            exit(1);
        }
        else {
            wait(NULL);
            close(fd[1]);
            fdd = fd[0];
            cmd++;
        }
    }
}

//========================= Manejo de comandos ==========================

int ownCmdHandler(char** parsed)
{
    int NoOfOwnCmds = 2, i, switchOwnArg = 0;
    char* ListOfOwnCmds[NoOfOwnCmds];
    char* username;
    ListOfOwnCmds[0] = "exit";
    ListOfOwnCmds[1] = "cd";
    for (i = 0; i < NoOfOwnCmds; i++) {
        if (strcmp(parsed[0], ListOfOwnCmds[i]) == 0) {
            switchOwnArg = i + 1;
            break;
        }
    }
    switch (switchOwnArg) {
    case 1:
        exit(0);
    case 2:
        chdir(parsed[1]);
        return 1;

    default:
        break;
    }
    return 0;
}

//========================= Manejo de cadena (tubería) ==========================

int parsePipe(char* str, char** strpiped)
{
    int i = 0;

    while( (strpiped[i] = strsep(&str,"|")) != NULL ){

        i++;
    }

    if (strpiped[1] == NULL)
        return 0;                                          //Regresa cero si
     no encuntra una tubería
    else {
        return i;
    }
}

 //========================= Manejo de cadena espacios ==========================

void parseSpace(char* str, char** parsed)
{
    int i;
```

```
226     char *aux;
227     for (i = 0; i < MAXLIST; i++) {
228         aux = strsep(&str, " ");
229         if (aux == NULL){
230             parsed[i] = aux;
231 
232             break;
233         }
234         else if (strlen(aux) == 0)
235             i--;
236         else{
237             parsed[i] = aux;
238         }
239     }
240 }
241 //========================== Manejo de cadena espacios tubería ==========================
242 
243 char ** parseSpacePipes(char* str, char** parsed)
244 {
245     char *aux;
246     char **parsed2;
247     parsed2 = (char**)malloc(sizeof(char*) * 100);
248     for (int i = 0; i < MAXLIST; i++) {
249         aux = strsep(&str, " ");
250         if (aux == NULL){
251             parsed[i] = aux;
252             parsed2[i] = aux;
253             return parsed2;
254         }
255         else if (strlen(aux) == 0)
256             i--;
257         else{
258             parsed[i] = aux;
259             parsed2[i] = aux;
260         }
261     }
262 }
263 
264 
265 
266 //========================== Procesar la cadena PRINCIPAL==========================
267 
268 int processString(char* str, char** parsed, char** parsedpipe)
269 {
270     char* strpiped[100];
271     int piped = 0, i = 0, y, k = 0, ejecutar = 0;
272     piped = parsePipe(str, strpiped);
273     char entrada[MAXLIST], salida[TAM];
274 
275     char ** cmd[(piped+1)];
276 
277     if (piped) {
278 
279         for(int j = 0; j < piped; j++){
280         i = 0;
281         while(i <= strlen(strpiped[j])){
282 
283 
284             if(strpiped[j][i] =='<'){                                   // si encuentra un simbolo de redireccionamiento de entrada, entrara en el if
285                     strpiped[j][i] = ' ';
286                     i++;
287                     if(strpiped[j][i] !=' '){                           //debe de haber separaciones entre cada letra o simbolo
288                         ejecutar=1;
289                     }else{
290                         i++;                                            //se lee despues del espacio
291                             for(y = 0; strpiped[j][i] !='\0' && strpiped[j][i] !=' ' && strpiped[j][i] !='|' && strpiped[j][i] !='>'; y++){
292                                 entrada[y] = strpiped[j][i];                    //vamos formando el argumento
293                                 strpiped[j][i] = ' ';
294                                 i++;
295                             }
296                             entrada[y]='\0';                            // se asigna un terminador de cadena
297                             if(strpiped[j][i]!='\0') i++;               //avanzamos a lo que sigue del comando
298                             redirEnt(entrada);                          //mandamos el argumento a la funcion para que se pueda procesar el fichero que se abrirá
299                         }
300             }
301 
302             if (strpiped[j][i] == '>') {                                // si encuentra un > corta la cadena que será el fichero que se usará para la salida
303                     strpiped[j][i] = ' ';
```

```
304                     i++;
305                     if (strpiped[j][i] != ' '){
306                         ejecutar=1;                                     //esto es para confirmar
    posteriormente que existe un erros de sintaxis
307                     }else{
308                             i++;                                        // se lee despues del
    espacio
309                                 for(y = 0; strpiped[j][i] != '\0';y++){
310                                     salida[y] = strpiped[j][i];          // vamos formando el
    argumento de la funcion redirSal que será el fichero de salida
311                                     strpiped[j][i] = ' ';
312                                     i++;
313                                 }
314                             salida[y] = '\0';                           // se le asigana el
    terminador de cadena
315                             redirSal(salida);                           //mandamos el argumento
    creado a la funcion para que sea tomado como el fichero de salida
316                     }
317             }
318
319             i++;
320
321         }
322     }
323
324     if(ejecutar!=0) printf("Error en la sintáxis\n");
325
326         parseSpace(strpiped[0], parsed);
327
328
329         for(i = 0; i < piped; i++){
330             cmd[i] = parseSpacePipes(strpiped[i], parsedpipe);
331
332         }
333         cmd[i] = NULL;
334
335
336     } else {
337
338         i = 0;
339         while(i <= strlen(str)){
340             if(str[i] =='<'){                                            // si encuentra un simbolo
    de redireccionamiento de entrada, entrara en el if
341                     str[i] = ' ';
342                     i++;
343                     if(str[i] !=' '){                                    //debe de haber
    separaciones entre cada letra o simbolo
344                         ejecutar=1;
345                     }else{
346                         i++;                                             //se lee despues del
    espacio
347                             for(y = 0; str[i] !='\0' && str[i] !=' ' && str[i] !='|' && str[i]
    !='>'; y++){
348                                 entrada[y] = str[i];                     //vamos formando el
    argumento
349                                 str[i] = ' ';
350                                 i++;
351                             }
352                             entrada[y]='\0';                             // se asigna un terminador
    de cadena
353                             if(str[i]!='\0') i++;                        //avanzamos a lo que sigue
    del comando
354                             redirEnt(entrada);                           //mandamos el argumento a
    la funcion para que se pueda procesar el fichero que se abrirá
355                         }
356             }
357
358             if (str[i] == '>') {                                         // si encuentra un > corta
    la cadena que será el fichero que se usará para la salida
359                     str[i] = ' ';
360                     i++;
361                     if (str[i] != ' '){
362                         ejecutar=1;                                     //esto es para confirmar
    posteriormente que existe un erros de sintaxis
363                     }else{
364                             i++;                                        // se lee despues del espacio
365                                 for(y = 0; str[i] != '\0';y++){
366                                     salida[y] = str[i];                  // vamos formando el
    argumento de la funcion redirSal que será el fichero de salida
367                                     str[i] = ' ';
368                                     i++;
369                                 }
370                             salida[y] = '\0';                           // se le asigana el
    terminador de cadena
371                             redirSal(salida);                           //mandamos el argumento
    creado a la funcion para que sea tomado como el fichero de salida
372                     }
373             }
```

```
374
375                 i++;
376
377             }
378
379              if(ejecutar!=0) printf("Error en la sintáxis\n");
380
381
382             parseSpace(str, parsed);
383
384         }
385         if (ownCmdHandler(parsed))
386             return 0;
387         else{
388             pipeline(cmd);
389             return 1 + piped;
390         }
391     }
392
393     int main()
394     {
395
396         char inputString[MAXCOM], *parsedArgs[MAXLIST];
397         char* parsedArgsPiped[MAXLIST];
398         int execFlag = 0;
399         int stdout = dup(1), stdin = dup(0);
400
401         init_shell();
402
403         while (1) {
404             close(1);                                                   // Se cierra la salida que
        tenga
405             dup(stdout);
406                 close(0);                                               //Se cierra la salida,
        tambien cierra el fichero cuando se ha guardado en el
407             dup(stdin);
408
409             printDir();                                                 //imprime
410
411             if (takeInput(inputString))
412                 continue;
413
414             execFlag = processString(inputString, parsedArgs, parsedArgsPiped);
415
416             if (execFlag == 1)
417                 execArgs(parsedArgs);                                 //Ejecuta si hay comandos
418
419         }
420         return 0;
421     }
```