ShellSO

Alunos: Josemar Rocha e Pedro Aleph

ShellSO

Interpretador de comandos do Unix/Linux.

Linguagem de programação: C.

Comandos principais:

- cd: acesso de diretórios.
- status: informa se o comando foi executado com sucesso.
- fim: finaliza e sai da shell.

Parsing

A string do input é iterada do começo ao fim, onde cada caractere é colocado em uma variável char “line” que será depois utilizada para a análise de comandos.

A análise consiste em:

Analisar a string, verificando por exemplo os 2 primeiros caracteres, caso ache por exemplo, “cd”.

Analisar o último, caso ache por exemplo, & que define que aquele processo seja executado em segundo plano.

```c
//le o input de string de quantidade desconhecida e retorna um array char
char *inputString(FILE* fp, size_t size){
    char *str;
    int ch;
    size_t len = 0;
    str = realloc(NULL, sizeof(char)*size);
    if(!str)return str;
    while(EOF!=(ch=fgetc(fp)) && ch != '\n'){
        str[len++]=ch;
        if(len==size){
            str = realloc(str, sizeof(char)*(size+=16));
            if(!str)return str;
        }
    }
    str[len++]='\0';
    return realloc(str, sizeof(char)*len);
}
```

Leitura de input

```
if(strcmp(line, "fim")==0){
    exitv++;
}else if(strcmp(firstTwoLetters, "cd")==0){
    if(strlen(line) == 2){
        const char* s = getenv("HOME");
        chdir(s);
    }else{
        char dir[strlen(line)-3];
        memcpy(dir, &line[3], strlen(line));
        chdir(dir);
    }
}else if(strcmp(args2[0], "status")==0){
    if(lastforeground != 1){
        printf("valor de saida %d\n", exitstatus);
```

É isso aí...

Lidando com crianças

Para lidar com os processos filhos, cada vez que um comando é executado com sucesso a função fork() é invocada. O processo pai espera o processo criança terminar e em seguida o mata.

Lidando com crianças

```
//coloca o que sera executado no comando execv em um vetor chamado exec_string

        char *exec_string[size];
        i = 0;
        if(ignored == 1){
            background = 1;
        }
        for(i=0; i < size-background; i++){
            exec_string[i] = OS_string[i];
        }
        exec_string[size-background] = NULL;
        if(ignored == 1){ background = 0; }
        pid_t spawnpid = -5;
        int childExitMethod = -5;
        int ten = 10;
        result = 0;
        int backgroundpid = 99;
        spawnpid = fork();
        int spaces2 = 0;
        if(skip == 1){
            sourceFD = open("/dev/null", O_WRONLY);
        }
```

Lidando com crianças

```
default: //classe pai
    if(background == 1 ){
        printf("background pid eh %d\n", spawnpid);
    }else if(background == 0){
        waitpid(spawnpid, &childExitMethod, 0);
        kill(spawnpid, SIGKILL);
    }
    break;
```

Pipe e redirecionamento

Para a implementação de pipes, por exemplo:

>> comando | outro comando

Foi utilizado dup2:

dup2 é basicamente um system call que duplica um descritor de arquivo, isso é bem útil no redirecionamento de output, pois automaticamente fecha o novo descritor, fazendo o redirecionamento elegantemente.

```
//acha i_place (onde <= ocorre") e o_place (onde => ocorre)

            i = 0;
            int i_place = 0;
            int o_file = 0;
            int o_place = 0;
            int OS_string_end = 0;
            for(i=0; i<= spaces; i++){
                if(strcmp(args2[i], "<=")==0){
                    i_place = i;
                }else if(strcmp(args2[i], "=>")==0){
                    o_place = i;
                    o_file = i+1;
                }
            }
```

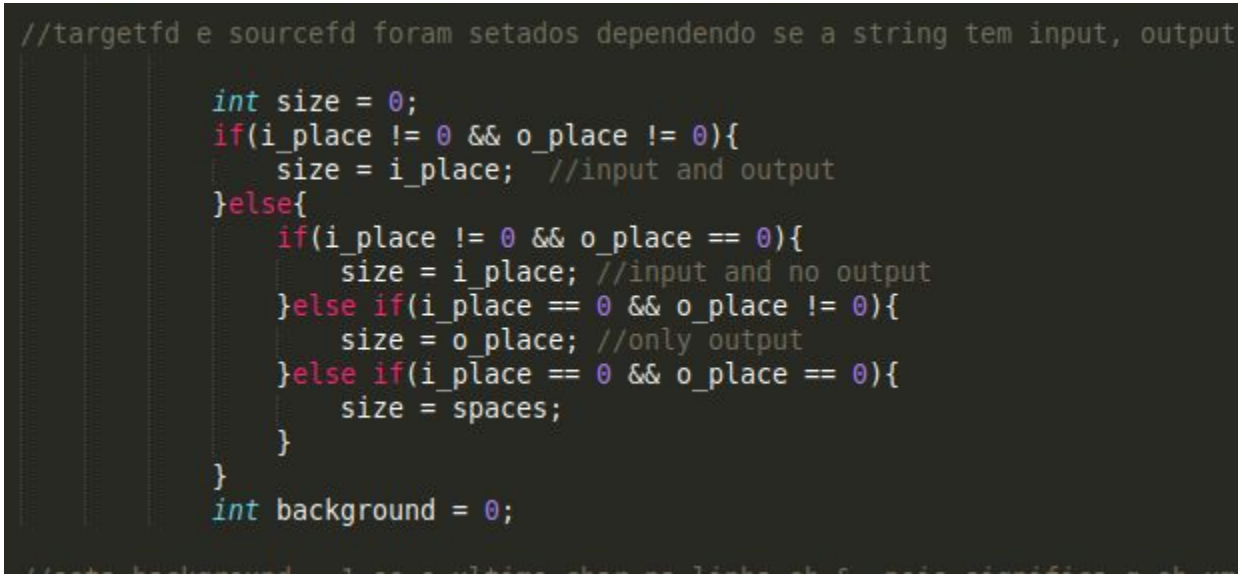
Defino como é recebido o redirecionamento no terminal

```c
//faz redirecionamento de i/o se necessario
            int sourceFD = 0;
            int targetFD = 0;
            int result = 0;
            int stat = 0;
            if(i_place != 0){

                if(access(OS_string[i_place+1], F_OK)!= -1){
                }else{
                    printf("nao pode abrir %s para o input\n", OS_string[i_place+1]);
                    skip = 1;
                }

                stat = 1;
                sourceFD = open(OS_string[i_place+1], O_RDONLY);
                OS_string_end = i_place;
                if(o_place != 0){
                    stat = 2;
                    targetFD = open(OS_string[o_place+1], O_WRONLY | O_CREAT | O_TRUNC, 0644);
                }
            }else{
                if(o_place == 0 && i_place == 0){
                    stat = 3;
                }else{
                if(o_place != 0){
                        stat = 4;
                            targetFD = open(OS_string[o_place+1], O_WRONLY | O_CREAT | O_TRUNC, 0644);
                    }
                }
            }
```

Caso necessário, o redirecionamento é colocado em um source e um target.



```
//targetfd e sourcefd foram setados dependendo se a string tem input, output

            int size = 0;
            if(i_place != 0 && o_place != 0){
                size = i_place;  //input and output
            }else{
                if(i_place != 0 && o_place == 0){
                    size = i_place; //input and no output
                }else if(i_place == 0 && o_place != 0){
                    size = o_place; //only output
                }else if(i_place == 0 && o_place == 0){
                    size = spaces;
                }
            }
            int background = 0;
```

Aqui é setado se o redirecionamento é somente de input, output, combinação.

```
case 0:   //caso filho

if(background==0){
    spaces2 = spaces;
    if(stat == 1){ //somente redirecionamento de inputs
        result = dup2(sourceFD, 0);
        if(skip == 1){

        }
        if(result ==-1){
            perror("source dup2()");
            exitstatus = 1;
        }
    }
    if(stat == 2){ //redirecionamento de ambos input e output
        result = dup2(sourceFD, 0);
        if(result ==-1){
            perror("source dup2()");
            exitstatus = 1;
        }
        result = dup2(targetFD, 1);
        if(result ==-1){
            perror("target dup2()");
            exitstatus = 1;
        }
        OS_string[1] = NULL;
        if(sourceFD == -1){
            perror("source open()");
            exitstatus = 1;
        }
        if(targetFD == -1){
            perror("target open()");
            exitstatus = 1;
        }
```

Execução do redirecionamento no fork do processo filho

Alguns dos comandos disponíveis

`cd`

`ls`

`pwd`

`cat`

`wc`

`fim`

Ze End