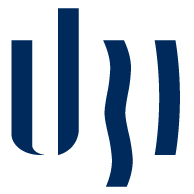
**Ficha Prática P6**



Sistemas Operativos 2023

SoShell implementação do “Redireccionamento” e “Pipes”

Objectivos: Adicionar as funcionalidades de ***redireccionamento*** e ***pipes*** para o projeto das aulas praticas de criar um interpretador de comandos

* Deve ver antes o sintaxe das funções : exec(), dup() e pipe()
  + - <http://man7.org/linux/man-pages/man2/dup.2.html>
    - <http://man7.org/linux/man-pages/man3/exec.3.html>
    - <https://man7.org/linux/man-pages/man2/pipe.2.html>

**Exercício 1:**

* Estude e Implemente o seguinte programa “testredirect.c”
* Este programa **simula** a execução do : cat <input.txt > output.txt

#define FILE\_MODE ( S\_IRUSR | S\_IWUSR )

int main(void)

{

int fd;

char nome[]="cat";

char \*args[2];

args[0] = nome;

args[1] = NULL;

fd=open("input.txt",O\_RDONLY);

if (fd<0) { fprintf(stderr,"open error"); return (1);}

//close (STDIN\_FILENO) //followed by dup(fd); is equivalent to dup2()

dup2(fd, STDIN\_FILENO);

close (fd);

fd=creat("output.txt",FILE\_MODE);

if (fd<0) { fprintf(stderr,"creat error"); return (1);}

dup2(fd, STDOUT\_FILENO);

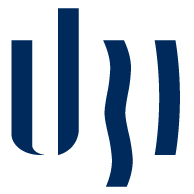
close (fd);

execvp(nome,args);

return(0);

}

Redireccionamento



O seu shell deverá implementar redireccionamento do :

* stdin (<) STDIN\_FILENO
* stdout ( > >> ) STDOUT\_FILENO
* standard error (2>) STDERR\_FILENO

O sintaxe da utilização é : comando [< file] [ [> file] | [>> file ] ] [2> file].

Os operadores têm que estar nesta ordem da utilização e só pode haver > ou >> e não ambos.

Exemplos:

* sosshell$ ls > f.txt
* soshelll$ ls nãoexiste 2> errors.txt
* soshell$ cat < fin.txt >> fout.txt 2> errors.txt

**Exercício 2** Implementação num ficheiro “redirects.c” da função redirects() que alterará a tabela de ficheiros do processo atual. O exemplo em baixo mostra como pode ser tratado o redireccionamento do stdout.

// redirects.c

// redirects tratamento dos simbolos por ordem inversa: i) 2> ii) > OU > iii) <

int redirects(int numargs, char \*args[])

{

// i tratar do redirect do 2>

if ( numargs <3) return numargs; //Guard the following section

if ( strcmp(args[numargs -2] , "2>") ==0) {

int fd= creat ( args[numargs -1], FILE\_MODE )

if (fd<0) { perror( NULL ) ; exit (1) ; /\*indicar um erro\*/ }

dup2(fd, STDERR\_FILENO) ;

close(fd)

args[numargs -2]=NULL ;

numargs = numargs -2 ;

}

//ii tratar do redirect do >

//verificar casos de (>) ou append (>>)

//tratar do > creat ou >> open O\_WRONLY | O\_APPEND

// iii tratar do < open O\_RDONLY

return numargs; //devolver o numero de argumentos a passar para execvp

}

**Notas:** A chamada para esta função **tem** que ser inserida no processo filho criado pelo shell mas antes de fazer um exec() do novo programa. Se a função devolver -1 então houve um erro e não deverá ser chamado execvp() mas exit() para terminar o processo filho. Altere o Makefile, shell.h, execute.c etc. conforme as necessidades.

TESTES soshell $ ls > tmp1

soshell $ cat tmp1

soshell $ cat < tmp1 > tmp2

soshell $ cat tmp2

soshell $ ls naoexists 2> tmp3

soshell $ cat tmp3

soshell $ ls > tmp1 &

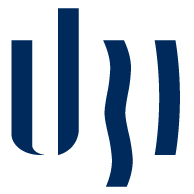
sohhell $ cat tmp1;

soshell $ ls >> tmp1

soshell $ cat < tmp1

soshell $ rm tmp1 tmp2 tmp3

PIPES



O seu shell deverá implementar **um** pipe *Não Recursivo*. ( Recursivo é um exericio ! )

Exemplos

* soshell> ls –l | wc –l -c
* soshell> cat /etc/passwd | grep root

Antes : Consultar a manual das chamadas ao sistema pipe() <http://man7.org/linux/man-pages/man2/pipe.2.html>

**Exercício 3**

Este exercício contém um programa para **simular** o processo dum pipe num shell. Em vez de ler uma linha de texto e utilizar o processo de parsing/particionamento para arranjar um vetor de apontadores (feito em qualquer interpretador de comandos – no soshell é a função parse()), faremos um processo mais simples, nomeadamente um vetor de apontadores embutido (hard coded) no programa. Faça uma diagrama que ilustra o funcionamento do programa com o pipe e implemente o programa completando o código em falta.

#include <stdio.h> <stdlib.h> <sys/stat.h> <sys/types.h> <fcntl.h> <unistd.h>

/\* Detect PIPE SYMBOL in array of Strings return its index or -1 if it does not exist \*/

int containsPipe **(**int numArgs**,** char **\*\***args**)**

**{**

int index**;**

**for** **(**index **=** 0**;** index **<** numArgs**;** index**++)**

**if** **(**args**[**index**][**0**]** **==** '|'**)**

**{**

**return** index**;**

**}**

**return** **-**1**;**

**}**

/\* Program to run Two Commands connected via a PIPE \*/

int main **()**

**{**

int indice**,** pidFilho**,** fd**[**2**],** numArgs**;**

//char \*myargs[]= { "ls", "-l" , "-a", NULL};

//char \*myargs[] = { "ls", "-l" , "-a", "|" ,"wc", "-c", NULL};

//char \*myargs[] ={ "cat", "-t", "/etc/passwd", "|", "grep", "-v", "root", NULL };

char **\*\***args **=** myargs**;**

numArgs **=** //3,6 ou 7

**indice** **=** containsPipe **(**numArgs**,** args**);**

**if** **(**indice **==** **-**1**)**

execvp **(\***args**,** args**);**

if (indice > 0 ) {

printf **(**"pipe detected at index %d\n"**,** indice**);**

printf **(**"Remove PIPE symbol. Create Pipe. Fork(). Exec in 2 Processes\n"**);**

args**[**indice**]** **=** **NULL;**

pipe **(** fd **);**

pidFilho **=** fork **();**

**if** **(** pidFilho **==** 0**)**

**{** //write

numArgs **=** indice**;**

fprintf **(stderr,**"cmd write to pipe: %s numArgs=%d\n"**,** args**[**0**],** numArgs**);**

dup2 **(**fd**[**1**],** STDOUT\_FILENO**);**

close **(**fd**[**0**]);** close **(**fd**[**1**]);**

**}**

**else**

**{** //read

args **=** args **+** indice **+** 1**;**

numArgs **=** numArgs **-** indice **-** 1**;**

fprintf **(stderr,**"cmd read from pipe: %s numArgs=%d\n"**,** args**[**0**],** numArgs**);**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//duplicar o descritor de ficheiro de leitura do PIPE para a posição na tabele de FD do STDIN

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//fechar o descritor do ficheiro do pipe que este processo não necessita.

**}**

\_\_\_\_\_\_\_\_\_\_\_\_\_ //Chamar a função execvp() para executar os comandos agora ligados via um pipe.

}

**return** 0**;**

**}**

**Exercício 4** Implementar a funcionalidade do pipe no *SoShell*

* Rever o ficheiro execute.c do *SoShell* Este é o ficheiro que será alterado.

**Procedimento:**

No Processo Filho criado no *SoShell* na função execute()

---Se a linha do comando não contêm o símbolo pipe 🡪 execvp() no processo “filho”.

ELSE Se a linha do comando contêm o símbolo pipe()

* + - Criar um pipe
    - Efectuar mais um fork para criar dois processos : Assim temos “Filho” e “Neto”
    - No processo apropriado fechar stdin/stdout
    - Fazer dup() do descritor do ficheiro do pipe apropriado
    - execvp() dos dois processos “filho” e “neto”

Testes

* Soshell > ls | wc
* Soshell > ls -l -a | grep \.c
* Soshell > cat < main.c | wc > lixo
* Soshell > ls | wc &

Exercício

* Rever o ficheiro execute.c do *SoShell*
* Fazer uma diagrama dos processo envolvidos e a comunicação entre eles via pipe()

## Conclusão do Exercício

O seu *SoShell* já implemente redireccionamento e pipes simples ?.Bom Trabalho