Universidade do Minho
Escola de Engenharia

TRABALHO PRÁTICO

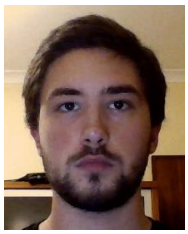
Mestrado Integrado em Engenharia Informática

Sistemas Operativos

(2º semestre, 2º ano)

Grupo 10

A78322 André Filipe Ferreira de Mira Vieira **A77048** Eduardo Gil Ribeiro Rocha



A78764 Ricardo André Araújo Neves

Data:

2 de junho de 2018

Índice

Resumo

Este relatório serve como complemento ao trabalho prático da Unidade Curricular de Sistemas de Software.

Aqui iremos expressar a nossa linha de pensamento sobre como realizamos o trabalho proposto, seguida de uma breve conclusão crítica sobre o projeto realizado pelo grupo.

Para a realização deste projeto foi necessário aplicar a matéria abordada nas aulas de Sistemas de Software, tendo o projeto ter sido feito na linguagem C.

Introdução

Este projeto final teve em vista a criação de uma aplicação capaz de processar ficheiros Notebook, que contêm dentro de si comandos, assinalados por ‘$’ no início da sua linha.

O programa deverá ser capaz de extrair os comandos contidos no ficheiro, executar os comandos e imprimir os seus resultados no ficheiro.

O programa deve também ser capaz de reprocessar ficheiros, substituindo outputs já colocados previamente no ficheiro.

Finalmente, o último requisito básico requer que o programa seja capaz de detetar erros na execução de argumentos e, através de sinais, interromper o programa, deixando o ficheiro de input inalterado.

Estrutura dos Ficheiros Notebook

Os ficheiros notebook têm uma estrutura que permite conter pedaços de texto independetes dos comandos armazenados dentro do ficheiro. O programa deverá ser capaz de processar os comandos, colocando os seus outputs dentro do ficheiro de input, delimitando esses outputs por “>>>” quando o output começa e “<<<” quando o output acaba.

```
Este comando lista os ficheiros:
$ ls -l
```

Figura 1 – Exemplo de um notebook antes de ser processado

```
Este comando lista os ficheiros:
$ ls -l
>>>
total 116
-rw-rw-r-- 1 d3 d3 74687 Abr 27 17:58 enunciado-so-2017-18.pdf
-rw-rw-r-- 1 d3 d3   363 Mai 30 23:43 exemplo.nb
-rwxrwxr-x 1 d3 d3 14080 Mai 30 23:43 main
-rw-rw-r-- 1 d3 d3    31 Mai 18 19:52 Makefile
-rw------- 1 d3 d3     0 Mai 30 23:43 pipe.txt
-rw-rw-r-- 1 d3 d3  5268 Mai 30 23:43 processador.c
-rw-rw-r-- 1 d3 d3   460 Mai 18 19:57 processador.h
-rw------- 1 d3 d3    41 Mai 30 23:43 temp.txt
<<<
```

Figura 2 – Exemplo de um notebook depois de ter sido processado

Primeiro Requisito

O primeiro requisito pede que o programa seja capaz de processar um ficheiro notebook, ou seja, ler um ficheiro de input, detetar comandos incluídos no ficheiro, processá-los e depois colocar o output desses comandos de volta no ficheiro.

Para isso, o programa gera um ficheiro de texto temporário com o filename “temp.txt” que irá conter o output do programa até o ficheiro de input ter sido todo processado, ponto a que o programa irá enviar o output colocado no ficheiro temporário de volta para o ficheiro original de input.

```
int input = open(filename, O_CREAT | O_RDWR, 0600);
int output = open("temp.txt", O_CREAT | O_TRUNC | O_RDWR, 0600);
```

Figura 1 – Abertura de ficheiros usados para input e output

O programa vai processando o texto do ficheiro, tendo atenção para se encontrar carateres ‘$’ no início de uma linha. Caso encontre, vai começar a extrair o texto encontrado nessa linha para um vector que irá conter os argumentos, para depois ser usado numa chamada da função “execvp”.

O programa tem também atenção a se o caracter ‘|’ segue o caracter ‘$’. Isto significa que será necessário incluir as funcionalidades de *pipe* na execução dos argumentos. Dentro do contexto do programa, isto significa que será chamada a função *pipeAux* em vez da função *execAux*.

```
if(flagPipe == 1){
    pipeAux(output, args);
    flagPipe = 0;
}
else execAux(output, args);
```

Figura 2 – Decisão de qual função de execução será chamada

No final do processo, o programa irá enviar o output guardado até esse ponto no ficheiro temp.txt para o ficheiro de input, através da função *terminar*.

```
reader(input, output);
terminar(filename);
```

Figura 3 – Chamada da função terminar depois da chamada de reader

Segundo Requisito

O segundo requisito do trabalho requer que o programa seja capaz de reprocessar um ficheiro notebook, substituindo outputs já colocados com novos outputs, baseados nos comandos que se encontram no ficheiro.

As funcionalidades deste requisito encontram-se todas na função *terminar*, que é a função que trata de reenviar os conteúdos do ficheiro *temp.txt* de volta para o ficheiro de input.

Para satisfazer este requisito, a função terminar passa a reconhecer a existência de carateres “>>>” e “<<<” no ficheiro, para indicar o início e fim de secções de output.

Usamos também flags para detetar onde está o primeiro output criado pelo programa, que é o único output que é reenviado para o ficheiro de input. O segundo output detetado é ignorado pelo processo de reenvio de texto.

```
if(buffer[0] == '>'){
    read(input, buffer+1, 3);
    if(buffer[0] == '>' && buffer[1] == '>' &&
       buffer[2] == '>' && buffer[3] == '\n'){
        if(flagPrim == 0){
            write(output, buffer, 4);
            read(input, buffer, 1);
            flagPrim = 1;
        }
        else flagPrim = 0;
        flag = 1;
    }
    else{
        write(output, buffer, 4);
        read(input, buffer, 1);
    }
}
```

Figura 4 – Secção responsável por detetar início de output

Terceiro Requisito

O terceiro requisito do trabalho requer que o programa seja capaz de processar certos sinais recebidos pelo processo pai. Estes sinais devem indicar quando uma chamada *exec* realizada por um processo filho corre mal ou então quando é executado *Ctrl-C* no terminal, pelo utilizador.

Os sinais deverão dizer ao processo pai para interromper o processo, apagando os ficheiros temporários criados, deixando o ficheiro Notebook de input inalterado.

Para permitir isto, o processo pai passa a estar atento à chegada de sinais *SIGINT*, que são enviados pelos filhos caso os seus *execs* corram mal ou então quando o utilizador envie o sinal através de *Ctrl-C*.

Figura 5 – Chamada à função signal, com o sinal SIGINT e apontador à função handlerChild

Quando um sinal *SIGINT* é recebido, é chamada então a função *handlerChild*, que por sua vez chamará a função *remFiles*, que tratará de remover os ficheiros temporários criados antes de terminar o processo.

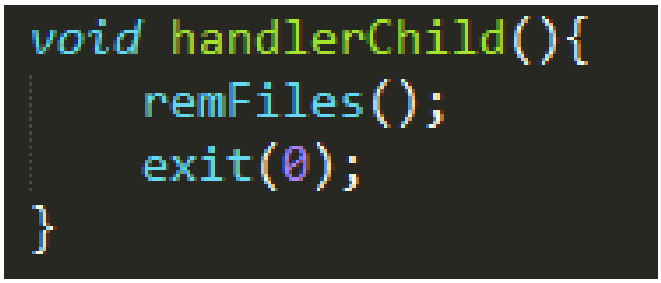


Figura 6 – Função que é chamada quando um sinal SIGINT é recebido

A função *remFiles* já é utilizada a partir do primeiro requisito. É a função que trata de remover os ficheiros auxiliares criados durante a execução do programa. A função é assistida pela variável global *flagFile*, que é a *flag* que indica se o ficheiro *pipeAux* foi criado. Caso essa *flag* tenha o valor 1, iremos apagar também o ficheiro *pipeAux*.

```
void remFiles(){
    int x = fork();
    if(x == 0){
        execlp("rm","rm","temp.txt",NULL);
        kill(getppid(), SIGINT);
        exit(-1);
    }
    x = fork();
    if(x == 0){
        execlp("rm","rm","pipe.txt",NULL);
        kill(getppid(), SIGINT);
        exit(-1);
    }
    if(flagFile){
        x = fork();
        if(x == 0){
            execlp("rm","rm","pipeAux.txt",NULL);
            kill(getppid(), SIGINT);
            exit(-1);
        }
    }
}
```

Figura 7 – Função remFiles

Instruções

Para correr o programa, basta utilizar o comando *make* para gerar o executável a ser utilizado, fornecendo também o nome do ficheiro a ser processado.

```
all: processador.c
	gcc processador.c -o notebook
```

Figura 8 – Ficheiro Makefile desenvolvido

```
d3@pc:~/Documents/SO/TP$ make
gcc processador.c -o notebook
```

Figura 9 – Chamada make usada para compilar o programa

```
d3@pc:~/Documents/SO/TP$ ./notebook exemplo.nb
d3@pc:~/Documents/SO/TP$ ./notebook exemplo2.nb
```

Figura 10 – Exemplos de uso do executável gerado

Conclusão

Achamos que o grupo realizou um bom trabalho. Estamos contentes com os requisitos que realizamos. Este trabalho foi muito útil em aprofundar os nossos conhecimentos relativos à matéria da disciplina, e em relembrar conhecimentos relativos à linguagem C, aprendidos em outras disciplinas.

Porém, gostaríamos de ter realizado os requisitos extras referidos no enunciado. O grupo não se encontrou com tempo suficiente para implementar esses requisitos, embora tenhamos passado algum tempo a discutir qual seria a nossa abordagem ao problema.

Em conclusão, enquanto estamos satisfeitos com o trabalho realizado, continuamos a querer ter feito os requisitos adicionais.