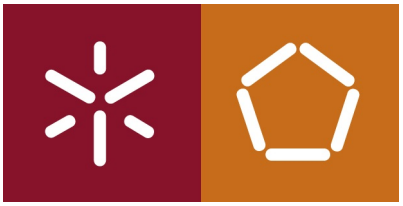

UNIVERSIDADE DO MINHO

Processamento de Notebooks

Sistemas Operativos
Ano letivo 2017/2018

Grupo 36
A71509 Cláudia Marques
A73236 Humberto Vaz
A72062 Ricardo Lopes

Braga
2 de Junho de 2018

Resumo

`Processamento de Notebooks` é um projeto realizado no âmbito da unidade curricular de Sistemas Operativos do 2º ano do Mestrado Integrado em Engenharia Informática da Universidade do Minho.

O objetivo deste projeto é construir um sistema para processamento de notebooks, que misturam fragmentos de código, resultados de execução e documentação.

O processador de *notebooks* é um comando que ao receber um nome de ficheiro, executa os comandos nele embebidos. Ou seja, sempre que encontra um $, executa o comando que se encontra a seguir e imprime o *output* do programa.

Ao longo deste relatório, serão abordados todos os objetivos propostos no enunciado.

Por fim, serão demonstrados alguns testes feitos ao sistema.

Conteúdo

Capítulo 1

Introdução

1.1 Descrição do Problema

Como já foi referido anteriormente, este trabalho tem como objetivo criar um sistema para processamento de *notebooks*, implementado em C, usando as primitivas do sistema operativo e os programos utilitários Unix indicados.

Este sistema deverá ser capaz de, ao receber um nome de um ficheiro, interpretar as linhas começadas por $ como comandos que serão executados, sendo o resultado produzido inserido imediatamente a seguir, delimitado por >>> *e* <<< .

As linhas começadas por $| executam comandos que têm como *stdin* o resultado do comando anterior.

Por exemplo, um ficheiro `exemplo.nb` com o conteúdo:

```
Este comando lista os ficheiros:
\$ ls
Agora podemos ordenar estes ficheiros:
$| sort
E escolher o primeiro:
$| head -1
```

O ficheiro ficará com o conteúdo do género:

```
Este comando lista os ficheiros:
$ ls
>>>
coisa.c
a.out
batata.c
<<<
Agora podemos ordenar estes ficheiros:
$| sort
```

```
>>>
a.out
batata.c
coisa.c
<<<
E escolher o primeiro:
$| head -1
>>>
a.out
```

Este trabalho possui ainda alguns requisitos de deteção de erros e interrupção da execução, nomeadamente caso algum dos comandos não consiga ser executado, não termine com sucesso, ou escreva algo para o *stderr*, o processamento deve ser anulado, ficando o notebook inalterável. Eventualmente, se o utilizador quiser interromper um processamento em curso, deverá fazer `Ctrl-C`, ficando o *notebook* inalterado.

Capítulo 2

Desenvolvimento

2.1 Script de Instalação

2.2 Makefile

2.3 Estruturas Utilizadas

No começo, foram criadas duas estruturas: *Estrut* e *Command*.

2.3.1 Estrut

Criou-se a estrutura *Estrut*, na qual definiu-se dois campos: o número de linhas do output e um apontador de apontadores *data*. É usada para armazenar o conteúdo produzido a seguir ao comando respetivo, que não é intepretado como comandos.

```
typedef struct estrutura
{
    int nrlinhas;
    char **data;
} * Estrut;
```

2.3.2 Command

Também se criou a estrutura *Command*, para armazenar a informação dos comandos recebidos. Esta estrutura é uma matriz em que cada palavra ocupa uma coluna.

```
typedef struct command
{
    char **command;
    char **out;
    int nrcoms;
} * Command;
```

2.4 Processador de Notebooks

A estratégia do grupo foi criar dois pipes anónimos, cujos *file descriptors* são `fd` e `fdPipe`. O primeiro será responsável por fazer *parsing* para passar para uma estrutura, e o segundo será responsável por conter o *output* do comando anterior.

Inicialmente, começou-se por usar a função **strtok** para quebrar a string original numa sequência de *tokens* delimitados por espaços, usando uma estrutura temporária *str*.

```
while ((readFile = fgets(temp, 100, file)) != NULL) {
    temp[strlen(temp) - 1] = '\0';
    linha = strdup(temp);
    char *s = strdup(" ");
    token = strtok(temp, s);
    int i = 0;
    int j = 0;
    while (token != NULL)
    {
        str[i] = strdup(token);
        token = strtok(NULL, s);
        i++;
    }
```

Após o ficheiro ser processado, caso as linhas não comecem por $, o programa imprime essas linhas "não modificadas". Caso contrário, são interpretadas como comandos, sendo adicionados ao array da estrutura `command` primeiramente criada.

```
if (str[0][0] != '$'){
        estrut->data[estrut->nrlinhas++] = strdup(linha); }
    else {
        Command addingCommand;
        addingCommand = initComs();
        for (j = 0; str[j] != NULL; j++) {
            addingCommand->command[j] = strdup(str[j]); }
        addingCommand->command[j] = NULL;
```

```
        commands = addCommand(commands, addingCommand,
            indexCommands++, &sizeCommands);

        estrut->data[estrut->nrlinhas++] = strdup(linha);
    }
```

2.4.1 Execução de Comandos sem pipe

No caso das linhas começarem só por $, são interpretadas como comandos, e o resultado gerido é inserido no ficheiro. Para isso, criou-se um processo filho que será responsável por escrever no *output* esse resultado.

Depois do *parsing* ter sido efetuado, é executado através deste processo:

```
for (k = 0; k < indexCommands; k++){
   int fd[2];
   int r = pipe(fd);
   if (r < 0)
       perror("Erro no Pipe\n");
   x = fork();
        if (x == 0 && commands[k]->command[0][1] != '|') {
        close(fd[0]);
        dup2(fd[WRITE_END], STDOUT_FILENO);
        execvp(commands[k]->command[1],
            &(commands[k]->command[1]));
        perror("N[U+FFFD]o devia imprimir isto\n");
       exit(-1);
       }
}
```

2.4.2 Execução de Comandos com pipe

Já no caso das linhas começarem por $ |, o comando tem como *stdin* o resultado do comando anterior. Assim, criou-se um processo filho que irá escrever no *output* do comando anterior para o *stdin* do processo quue vai executar o comando.

```
if (x == 0 && commands[k]->command[0][1] == '|'){
...
   int fdPipe[2];
   pipe(fdPipe);

   int y;
   y = fork();
   if (y == 0) {
       close(fdPipe[0]);
```

```
        dup2(fdPipe[WRITE_END], STDOUT_FILENO);
        int l;
        for (l = 0; commands[k - 1]->out[l] != NULL; l++) {
            write(fdPipe[WRITE_END], commands[k - 1]->out[l],
                strlen(commands[k - 1]->out[l]));
            write(fdPipe[WRITE_END], "\n", 1);
        }
        close(fdPipe[WRITE_END]);
        exit(0);
    }
    ...
}
```

Também foi criado um processo pai, que ao receber informação do pipe guarda-a para a estrutura **Command**.

```
...
close(fd[WRITE_END]);
int status;
char buf[100];
int counter = 0;
wait(0);
WEXITSTATUS(status);
if (status == 0) {
        line = 0;
        while ((readPipe = lerLinha(fd[READ_END])) != NULL {
                commands[k]->out[line++] = strdup(readPipe);
        }
        int i;
}
...
```

2.4.3 Deteção de erros

Nos casos de erros, é enviada uma mensagem de erro, que é escrita no *stderr*. Por exemplo:

```
        ...
        perror("Error on waiting for child process\n");
        ...
```

2.4.4 Acesso a resultados de comandos anteriores arbitrrios

Capítulo 3

Testes

Capítulo 4

Conclusão

Neste trabalho, o principal objetivo era a utilização da linguagem de programação C, para a criação de um programa de *Processamento de Notebooks*.

Em relação ao resultado final do trabalho, o grupo foi capaz de alcançar todos os objetivos propostos, e ainda satisfazer, de forma correta, todas as funcionalidades básicas enunciadas, e uma parte das funcionalidades avançadas.