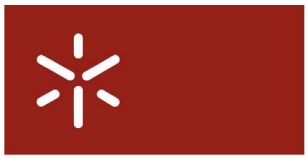
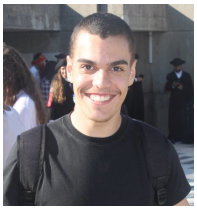


Universidade do Minho

Sistemas Operativos

MIEI - 2º ano - 2º semestre - Grupo 53
Universidade do Minho

Trabalho Prático



João Leal
A75569

Bernardo Viseu
A74618

António Lopes
A74357

2 de Junho de 2018

Conteúdo

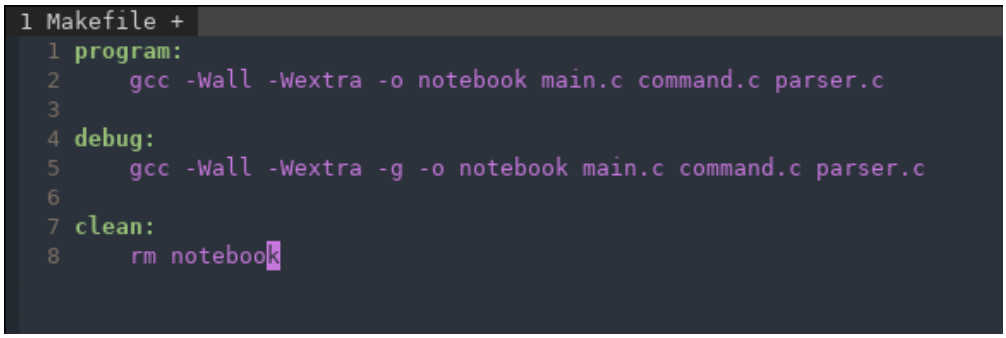
1 Introdução

No âmbito da unidade curricular de Sistemas Operativos, foi-nos proposto a elaboração de um trabalho prático que visa desenvolver um sistema para processamento de notebooks, que misturam fragmentos de código, resultados da execução, e documentação. No contexto deste trabalho prático um notebook é um ficheiro texto (por exemplo notebook.nb) que depois de processado é modificado de modo a incorporar resultados da execução de código ou comandos nele embebidos. Para o desenvolvimento deste sistema usamos a linguagem C, com a utilização de uma Makefile que compila os ficheiros de código e faz um executável (denominado "notebook"). Para utilizar este software basta correr o executável dando como argumento o ficheiro notebook que se quer processar (por exemplo "./notebook testeNotebook.nb").

Para fazermos este projecto aplicamos os conhecimentos ensinados nas aulas práticas de Sistemas Operativos, utilizando também práticas de cadeiras estudadas anteriormente, como Programação Imperativa e Laboratórios de Informática 2/3.

2 Makefile

A Makefile permite-nos compilar todo o nosso programa para que depois seja possível executá-lo.



2.1 Comandos

program - Comando que compila os varios ficheiros .c e gera um executável.

debug - Comando que compila os varios ficheiros .c, utilizando o parametro -g.

clean - Comando que remove o executável.

3 Funcionalidade do programa

3.1 Estrutura do comando

```
 6 struct command{
 7     char* comment;
 8     char* execution;
 9     char* output;
10 };
11
12 Command new_command(){
13     Command com = malloc(sizeof(struct command));
14     com->comment = malloc(1024);
15     com->execution = malloc(64);
16     com->output = malloc(1024);
17
18     return com;
19 }
```

Esta estrutura é utilizada para guardar os vários dados que nos são importantes para a execução do programa. Nós acessamos a esta estrutura para ter acesso aos outputs e lista de comandos realizados anteriormente, de maneira a poder utilizar os outputs como input para outros comandos do notebook.

comment - Sequência de caracteres onde se encontra a breve descrição do comando que se vai executar.

execution - Sequência de caracteres onde se encontra os comandos que se vao executar.

output - O output da execução dos comandos.

3.2 Diferentes tipos de comandos

**"$"** -> Linhas começadas com "$" são interpretadas como comandos que serão executados sendo o output inserido imediatamente a seguir, delimitado por ">>>" e "<<<".

**"$|"** -> Linhas começadas por "$|" executam comandos que tem como stdin o output do comando anterior.

**"$N|"** -> Linhas começadas por "$N|" podem ser generalizados para terem como stdin o resultado o N-ésimo comando anterior.

3.3 Exemplo de um ficheiro Notebook (.nb)

Um Notebook pode ser, por exemplo:

```
1 teste2.nb +
  1 Este comando lista os ficheiros
  2 $ ls
  3 Estado do repositorio
  4 $ git status
  5 Head
  6 $| head -1
  7 Cowsay
  8 $| cowsay
```

Depois de executar o programa, o notebook fica:

```
1 teste2.nb +
  1 Este comando lista os ficheiros
  2 $ ls
  3 >>>
  4 command.c
  5 command.h
  6 main.c
  7 Makefile
  8 notebook
  9 parser.c
 10 parser.h
 11 README.md
 12 teste1.nb
 13 teste2.nb
 14 <<<
 15 Estado do repositorio
 16 $ git status
 17 >>>
 18 On branch master
 19 Your branch is up to date with 'origin/master'.
 20
 21 Changes not staged for commit:
 22   (use "git add <file>..." to update what will be committed)
 23   (use "git checkout -- <file>..." to discard changes in working directory)
 24
 25     modified:   command.c
 26     modified:   main.c
 27     modified:   notebook
 28     modified:   teste1.nb
 29
 30 no changes added to commit (use "git add" and/or "git commit -a")
 31 <<<
 32 Head
 33 $| head -1
 34 >>>
 35 On branch master
 36 <<<
 37 Cowsay
 38 $| cowsay
 39 >>>
 40  __________________
 41 < On branch master >
 42  ------------------
 43         \   ^__^
 44          \  (oo)\_______
 45             (__)\       )\/\
 46                 ||----w |
 47                 ||     ||
 48 <<<
```

3.4 O Código do Programa:

Para o realizamento deste trabalho utilizamos conhecimentos abordados nas aulas práticas de SO.
O programa funciona com a utilização de 3 ficheiros de código C (main.c, parser.c e command.c), ficheiros header (parser.h e command.h) e uma Makefile.

No ficheiro command.c definimos a nossa estrutura Command que é como nós guardamos os dados importantes de um notebook, aqui também se encontram os getters, setters e free's de cada variável da estrutura (char* comment, char* execution e char* output). Aqui também temos a função *char** *trim_ execution(Command c)* que devolve o comando que se quer executar, num conjunto de strings, se, por exemplo, o comando for "git status", esta função retorna {"git","status"}.

No ficheiro parse.c definimos as funções *Command* parse_file(int fd, int* i)* e *ssize_ t read_ line(int fd, char* buf, size_ t nbyte)*. A função parse_file é responsável por receber o ficheiro notebook e retornar um array de Commands, que vai ser utilizado na main, para processar o notebook. A função read_line é utilizada para ler linha a linha o ficheiro.

Assim sendo, na nossa main, o ficheiro dado como argumento é primeiramente parsado, retirando deste o Comentário e o Comando a ser executado, ignorando qualquer Output (delimitado entre ">>>" e "<<<"), sendo cada uma destas informações guardadas para um comando que é posteriormente inserido num array de Comandos.

Tendo todos os comandos guardados, iteramos este mesmo array e verificamos que tipo de comando se trata, através de um *strlen()* em que, se este valor for 1, temos que o comando a ser executado é um comando simples ($), se for 2, temos que o comando a ser executado é um comando cujo input é o output do anterior ($|) e se este for maior que 2, é um comando cujo input é o output do N-ésimo comando anterior ($N|). Sabendo que tipo de comando se trata, prosseguimos a fazer um *fork()* de modo a criar um novo processo, e através do funcionamento do *pipe* e da função *dup2* garantimos o bom funcionamento destes mesmo processos, retirando também o seu output e posteriormente guardando-o na nossa estrutura. Chegando ao fim do nosso ciclo e tendo em posse todas as informações relevantes dos comandos, presentes no ficheiro passado como argumento, resta-nos apenas escrever para o ficheiro em questão essas mesmas informações, usufruindo da função *write(int fildes, const void *buf, size_ t nbyte)* .

4 Conclusão

Para a realização deste projecto baseamo-nos naquilo que temos feito e desenvolvido nas aulas práticas. Uma das maiores dificuldades deste projecto foi pensar no método em que íamos guardar os dados que cada notebook tem, acabamos por utilizar a estrutura Command, porque foi a que achamos mais conveniente para realizar os processos necessários nos notebooks.

O facto de ser possível abordar este projecto de várias maneiras diferentes, deu-nos a possibilidade de sermos criativos no realizamento deste trabalho, e mostrou-nos melhor o quão "maleável" é a linguagem de programação C.

Desenvolvemos, desta forma, o nosso conhecimento com base no funcionamento do sistema operativo que mais utilizamos (Linux) e percebemos o quanto podemos explorar. Este projecto foi bastante útil na consolidação dos conhecimentos e funcionalidades de liguagem de programação C já conhecidos, como as *struct's*, entre outros, e mostrou-nos o verdadeiro potencial que esta linguagem consegue oferecer, também melhoramos os conhecimentos de aplicaçoes de debugging como *valgrind* e *gdb*. Desta forma, de certeza que o resultado e empenho que demonstramos neste trabalho, nos virá a ser bastante útil num futuro profissional.