Interpretador de comandos

LA1/LI2

Aula 1
1 de Março de 2015

Alguns comandos Linux

cd	muda de directoria
ls	lista o conteúdo de uma directoria
cp	copia um ficheiro ou um conjunto de ficheiros
mv	move um ficheiro ou conjunto de ficheiros
rm	remove um ficheiro ou conjunto de ficheiros
cat	mostra o conteúdo de um ficheiro
more	mostra o conteúdo de um ficheiro (com pausa)
mkdir	cria uma directoria
rmdir	apaga uma directoria
gcc	Compilador de C
gdb	Debuggger
make	Comando `make`

Exemplo de utilização da linha de comandos

```
mkdir code
cp *.c code
cd code
ls -la
cat *.c
```

Compilação

- O código `C` é texto
- O codigo `C` passa por vários processos até ao executável
- Esse processo de transformação chama-se compilação
- A compilação é feita utilizando o comando `gcc`
- Um projecto em `C` é composto por muitos ficheiros
- Cada um desses ficheiros é compilado para código objecto
- Após isso, os vários ficheiros de código objecto são ligados num único executável

- O comando `make` serve para facilitar a tarefa de compilação
- Compila automáticamente todo o projecto
- Só compila o que foi modificado desde a última compilação

```
CFLAGS=-Wall -Wextra -ansi -pedantic -O2
OBJS=$(patsubst %.c,%.o,$(wildcard *.c))
LIBS=-lreadline

prot: $(OBJS)
	$(CC) $(CFLAGS) -o prot $(OBJS) $(LIBS)

limpar:
	rm prot *.o
```

O meu primeiro programa

```
/**
Funcao principal
@return Devolve zero se tudo correu bem
**/
int main() {
    return 0;
}
```

O meu segundo programa

```c
#include <stdio.h>

/**
Funcao principal
@return Devolve zero se tudo correu bem
**/
int main() {
    printf("ola\n");
    return 0;
}
```

O meu terceiro programa

```c
#include <stdio.h>
#define MAX_SIZE 1024
/**
Funcao principal
@return Devolve zero se tudo correu bem
**/
int main() {
    /* Variavel que vai armazenar a linha */
    char buffer[MAX_SIZE];
    /* Ler uma linha */
    fgets(buffer, MAX_SIZE, stdin);
    /* Imprimir a linha */
    printf("%s\n", buffer);
    return 0;
}
```

O meu quarto programa

```c
#include <stdio.h>
#include <string.h>
#define MAX_SIZE 1024

/* Prototipo da funcao */
char *nome_comando(char *str);
/** Funcao principal
@return Devolve zero se tudo correu bem
**/
int main() {
    /* Variavel que vai armazenar a linha */
    char linha[MAX_SIZE];
    /* Ler uma linha */
    fgets(linha, MAX_SIZE, stdin);
    /* Imprimir o nome do comando */
    printf("%s\n", nome_comando(linha));
    imprime_comando_e_argumento(linha);
    return 0;
}
```

O meu quarto programa

```
/** Devolve o nome do comando que foi passado
@param str a linha de texto passada como parametro
@return A string que contem o nome do comando
*/
char *nome_comando(char *str) {
    static char buf[MAX_SIZE];
    sscanf(str, "%s", buf);
    return buf;
}
/**
Imprime o nome do comando e o primeiro argumento
@param str a linha de texto passada como parametro
*/
void imprime_comando_e_argumento(char *str) {
    static char cmd[MAX_SIZE];
    static char arg[MAX_SIZE];
    sscanf(str, "%s %s", cmd, arg);
    printf("comando: %s\narg: %s\n", cmd, arg);
}
```

O que falta

- Verificar o valor devolvido pela função `fgets` para verificar se se conseguiu ler a linha ou não (e.g., porque o ficheiro acabou)
- O valor devolvido caso não se consiga ler será `NULL`
- Verificar o valor devolvivo pela função `sscanf` para verificar se se conseguiram ler todos os argumentos
- A função `sscanf` devolve o nº de argumentos lido ou `EOF` caso não consiga ler nenhum valor

Exemplo de interpretador de comandos

```
/** Interpreta um comando passado numa linha */
int interpretar(char *linha) {
    char comando[MAX_SIZE];
    char arg1[MAX_SIZE];
    char arg2[MAX_SIZE];
    int nargs;
    nargs = sscanf(linha, "%s %s %s",
        comando, arg1, arg2);

    if(strcmp(comando, "seq") == 0 &&
       nargs == 2)
        return cmd_seq(comando, arg1);
    else if(strcmp(comando, "sair") == 0) {
        return 0;
    } else {
        return -1; /* Erro comando nao existe*/
    }
}
```

Exemplo de interpretador de comandos

```
/**
Interpretador de comandos
*/
void interpretador() {
     int resultado = 0;
     char buffer[MAX_SIZE];
     int ciclo = 1;
     while(ciclo &&
       fgets(buffer, MAX_SIZE, stdin) != NULL){
         resultado = interpretar(buffer);
         if(resultado == 0)
             ciclo = 0;
     }
}
```