FOLHA DE PROBLEMAS Nº 2

Consola, ficheiros e directórios

---

1. – Manipulação da consola – leitura sem eco.

Escreva um programa que leia uma sequência de caracteres do teclado sem ecoar o que o utilizador escreve mas apenas um asterisco por cada carácter introduzido, como acontece normalmente ao introduzir uma *password*. Use as funções de configuração da consola do Unix e apenas chamadas directas ao sistema operativo para fazer a leitura (p1.c, fornecido na página Web da disciplina).

2. – Operações básicas sobre ficheiros. Efeitos de *buffering* e do número de chamadas ao sistema.

Escreva um programa, chamado `copy`, que copie o conteúdo de um ficheiro para outro. Os ficheiros de origem e de destino devem ser indicados como argumentos da linha de comandos (ex.: `copy source destination`). Resolva este problema usando:

a) funções da biblioteca da linguagem C para abrir, ler e escrever nos ficheiros (`fopen()`, `fread()` e `fwrite()`) (= folha1-p6a.c);

b) chamadas directas ao sistema operativo (`open()`, `read()` e `write()`) (p2b.c).

c) Determine o tempo de execução de cada uma das versões do programa anterior ao copiar um ficheiro de grande dimensão (Nota: o comando `find` *`dir_name`* `-size +1000k` pode ser usado para procurar ficheiros ocupando mais do que 1MB). Repita, usando *buffers* de tamanhos diferentes e analise os resultados.

3. – Operações básicas sobre ficheiros. Redireccionamento de entrada/saída de um processo.

a) Escreva um programa que apresente no écran o conteúdo de um ficheiro de texto cujo nome lhe é passado como argumento da linha de comando (modificar p2b.c).

b) Altere o programa da alínea anterior de modo a que, caso lhe seja passado um nome de um segundo ficheiro como argumento, a saída seja redireccionada para este ficheiro; use a chamada `dup2()` para fazer o redireccionamento da saída.

4. – Ficheiros de texto e outros.

a) Escreva um programa que leia do teclado os nomes dos alunos de uma disciplina e a respectiva classificação, gravando-os num ficheiro de texto. Na escrita do ficheiro, use apenas chamadas directas ao sistema.
Verifique, usando o comando `cat`, que os dados foram gravados correctamente.

b) Repita o problema da alínea anterior, gravando os dados num ficheiro de *struct*'s de C, em que os campos de cada *struct* são o nome (uma *string* com um comprimento máximo de 50 caracteres) e a classificação (um inteiro).
Escreva um programa que mostre o conteúdo gerado pelo programa anterior no ecrã. Explique por que é que o conteúdo não pode ser mostrado recorrendo ao comando `cat`.

5. – Manipulação do "cursor do ficheiro". Sincronização no acesso a ficheiros.

Considere os dois seguintes programas (p5a.c e p5b.c):

```
// PROGRAMA p5a.c
#include ... //a completar

int main(void)
{
  int fd;
  char *text1="AAAAA";
  char *text2="BBBBB";

  fd = open("f1.txt",O_CREAT|O_EXCL|O_TRUNC|O_WRONLY|O_SYNC,0600);
  write(fd,text1,5);
  write(fd,text2,5);
  close(fd);
  return 0;
}
```

```
// PROGRAMA p5b.c
#include ... //a completar

int main(void)
{
  int fd;
  char *text1="CCCCC";
  char *text2="DDDDD";

  fd = open("f1.txt", O_WRONLY|O_SYNC,0600);
  write(fd,text1,5);
  write(fd,text2,5);
  close(fd);
  return 0;
}
```

a) Compile-os e execute-os, lançando-os em execução em duas janelas de terminal diferentes. Interprete os resultados.

b) Introduza algumas modificações nos programas (uma de cada vez) como, por exemplo,

- fazer testes de erro nas chamadas de abertura e de escrita no ficheiro,
- controlar a ordem pela qual as operações de escrita são efectuadas, executando `getchar()`antes de cada `write()`, (deste modo o programa esperará que o utilizador "dê ordem" para prosseguir)
- acrescentar e ou retirar *flags* de controlo da abertura do ficheiro `f1.txt`, como `O_EXCL` , `O_APPEND`, `O_TRUNC` e `O_SYNC`,

e execute-os novamente. Interprete os resultados.

6. – Acesso a directórios. *Hard links* e *symbolic links*.

a) Considere o programa apresentado na página seguinte (p6a.c) que se pretendia que apresentasse no écran alguma informação sobre o conteúdo de um directório que lhe seja passado como argumento da linha de comando. Interprete o programa, execute-o (com diferentes argumentos) e corrija os erros. Sugestão: faça um teste de erro ao valor retornado por `lstat()`.

b) Altere o programa por forma a mostrar no écran o *i-node* associado a cada entrada do directório e o tamanho dos ficheiros regulares.

c) Crie um ficheiro de texto (`temp.txt`), com um conteúdo qualquer, no seu directório corrente. Crie um *hardlink* (`temp1.txt`) e um *symbolic link* (`temp2.txt`) para esse ficheiro. Execute o programa da alínea anterior e interprete o resultado no que diz respeito ao tamanho do ficheiro de texto original e dos dois *links*.

d) Modifique a chamada `lstat()` para `stat()` e volte a executar o programa. Interprete novamente os resultados.

e) Apague os ficheiros de texto, pela ordem a seguir indicada: `temp.txt`, `temp2.txt` e `temp1.txt`. Após cada apagamento, execute o comando `ls -la`, tente mostrar no ecrã o conteúdo dos ficheiros que ainda restarem (usando o comando `cat`) e interprete os resultados.

```
// PROGRAMA p6a.c ( referido na alínea a) )
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <errno.h>

int main(int argc, char *argv[])
{
 DIR *dirp;
 struct dirent *direntp;
 struct stat stat_buf;
 char *str;

 if (argc != 2)
 {
  fprintf( stderr, "Usage: %s dir_name\n", argv[0]);
  exit(1);
 }

 if ((dirp = opendir( argv[1])) == NULL)
 {
  perror(argv[1]);
  exit(2);
 }

 while ((direntp = readdir( dirp)) != NULL)
 {
  lstat(direntp->d_name, &stat_buf);
  if (S_ISREG(stat_buf.st_mode)) str = "regular";
  else if (S_ISDIR(stat_buf.st_mode)) str = "directory";
  else str = "other";
  printf("%-25s - %s\n", direntp->d_name, str);
 }

 closedir(dirp);
 exit(0);
}
```