Sistemas Operativos
Acesso a Ficheiros

Grupo de Sistemas Distribuídos
Universidade do Minho

1 Objectivos

Familiarizar-se e utilizar as chamadas ao sistema essenciais para operação sobre ficheiros.

2 Chamadas ao sistema

```
#include <unistd.h>   /* chamadas ao sistema: defs e decls essenciais */
#include <fcntl.h>    /* O_RDONLY, O_WRONLY, O_CREAT, O_* */

int     open(const char *path, int oflag [, mode]);
ssize_t read(int fildes, void *buf, size_t nbyte);
ssize_t write(int fildes, const void *buf, size_t nbyte);
int     close(int fildes);
```

3 Exercícios propostos

1. Implemente um programa que crie um ficheiro com 10MB de tamanho cujo nome é passado como argumento. O conteúdo do ficheiro propriamente dito é irrelevante: por exemplo, pode ser composto exclusivamente pelo caractere 'a'.

   ```
   $ ./10mb 10mb.dat
   ```

2. Implemente em C um programa `mycat` com funcionalidade similar ao comando `cat`. Comece por suportar apenas o funcionamento como filtro, ou seja, deverá operar num ciclo de leitura de dados a partir do standard input, processamento dos dados lidos, e escrita do resultado do processamento no standard output. Na primeira versão leia e escreva caractere a caractere (recorde que a indicação de fim de ficheiro (*end of file*) no standard input resulta da combinação `Control+D` no início de uma linha).

   ```
   $ ./mycat-v1
   linha
   linha
   outra
   outra
   ```

3. Reimplemente o comando `mycat` de modo a que, em vez de ler o standard input caractere a caractere, o faça agora em blocos de N bytes, valor passado como seu argumento.

4. Compare o tempo de execução desta versão do comando `mycat` para diferentes valores de N (sugestão: utilize potências de 2). Para o efeito utilize o comando `time` como se ilustra no exemplo abaixo. Em ambos os casos, repare nas parcelas de tempo real, modo utilizador e sistema. Como justifica a diferença observada? Nota: no exemplo abaixo o redireccionamento da saída justifica-se para minimizar o atraso adicional na execução do programa causado pela escrita no terminal.

```
$ time mycat-v2 1 < 10mb.dat > /tmp/lixo
real    0m10.865s
user    0m1.860s
sys     0m8.997s
$ time mycat-v2 1024 < 10mb.dat > /tmp/lixo
real    0m0.020s
user    0m0.003s
sys     0m0.015s
```

5. Implemente a leitura de uma linha numa função `readln`, cujo protótipo é compatível com a chamada ao sistema `read`. Nesta versão, leia um caractere de cada vez.

```
ssize_t readln(int fildes, void *buf, size_t nbyte);
```

6. Implemente, utilizando a função `readln`, um programa com funcionalidade similar ao comando `nl`, o qual repete linha a linha o standard input ou o conteúdo de um (só) ficheiro especificado na sua linha de comando. Cada linha repetida é numerada sequencialmente.

```
$ ./nl-v1
um
     1  um
dois
     2  dois
$ ./nl-v1 a.txt
     1  linha 1
     2  linha 2
```

7. Reimplemente a função `readln` de modo a reduzir o número de invocações da chamada ao sistema `read`. Para esse efeito deverá procurar ler blocos de, por exemplo, 1024 caracteres de cada vez. No caso de ter sido lida mais do que uma linha de texto, a função deve preservar informação de modo a poder retornar (parte) do texto remanescente numa invocação subsequente. A nova versão deve seguir a API proposta abaixo. A função `create_buffer` permite dimensionar um buffer associado a um descritor de ficheiro.

```
int create_buffer(int filedes, struct buffer_t *buffer, size_t size);
int destroy_buffer(struct buffer_t *buffer);
ssize_t readln(struct buffer_t *bufer, void *buf, size_t nbyte);
```

8. Reimplemente o programa `nl` utilizando a função `readln` do exercício anterior.

4 Exercícios Adicionais

1. Reimplemente o programa `mycat` acrescentando à sua operação de filtro a capacidade de repetir o conteúdo de ficheiros especificados como argumento da sua linha de comando.

```
$ ./mycat-v2 a.txt
linha 1
linha 2
$ ./mycat-v2 b.txt
linha 3
$ ./mycat-v2 a.txt b.txt
linha 1
linha 2
linha 3
```

2. Implemente uma API inspirada na última versão da função `readln` de forma a agrupar escritas, reduzindo desta forma o número de invoações da chamada ao sistema `write`. O conteúdo deve ser escrito para disco quando o buffer está cheio ou a função `flush` é invocada. Pode considerar a seguinte API.

```
ssize_t writebatch(struct buffer_t *buffer, void* data, ssize_t size)
ssize_t flush(struct buffer_t *buffer);
```

3. Reimplemente a função `readln` de modo a seguir a API proposta abaixo. A função `readln` devolve em `buf` o endereço para onde foi lida a linha que está a ser retornada.

```
ssize_t readln(struct buffer_t *bufer, void **buf);
```

4. Implemente um programa com funcionalidade similar ao comando `head` que funcione como filtro ou que opere sobre ficheiros especificados na sua de comando.

```
$ ./myhead -1
linha
$ ./myhead -1 a.txt b.txt
==> a.txt <==
linha 1
==> b.txt <==
linha 3
```

5. Implemente um programa similar ao comando `grep` que funcione como filtro ou que opere sobre ficheiros especificados na sua linha de comando.

```
$ ./mygrep 1 a.txt b.txt
a.txt:linha 1
```

6. Implemente um programa similar ao comando `wc` que funcione como filtro ou sobre ficheiros da sua linha de comando.

```
$ ./mywc a.txt b.txt
       2       4      16 a.txt
       1       2       8 b.txt
       3       6      24 total
```

7. Implemente um programa similar ao comando `cmp` que verifica se dois ficheiros são iguais (byte a byte).

```
$ ./mycmp a.txt b.txt
a.txt b.txt differ: char 7, line 1
```