E/S em Unix

Em C é habitual usar-se a biblioteca `stdio` para efetuar operações de E/S (e.g. `printf()`, `scanf()`, etc.)..

As funções desta biblioteca são de *alto-nível* e realizam 3 operações importantes:

- *buffering* automático. Usam um buffer interno invisível ao programador que permite a leitura de muitos bytes de cada vez.
- conversões de input e output. Por exemplo, conversão da representação de um inteiro no seu valor numérico.
- formatação automática de input e output.

E/S em Unix

Contudo, nem sempre estas operações são desejadas. Quando se faz E/S diretamente para um periférico, e.g. “tape”, é necessário ter maior controlo, nomeadamente sobre o tamanho dos buffers.

O Unix fornece uma interface *baixo-nível* para E/S, na qual um ficheiro é referenciado por um *descritor de ficheiro* que mais não é do que um inteiro.

Descritores de ficheiros

Ao criar-se um ficheiro novo ou abrir-se um ficheiro existente, o que se obtém é um descritor do ficheiro, que identifica o ficheiro em operações subsequentes.

Descritores 0, 1 e 2 são pré-definidos e correspondem a `stdin`, `stdout` e `stderr`.

Operações sobre ficheiros

→ **abertura e criação de ficheiros:**

```
int open(char *path, int flags, mode_t mode);
int creat(char *path, mode_t mode);
```

`flags`: `O_RDONLY`, `O_WRONLY`, `O_CREAT`, ...

`mode`: define permissões de acesso.

A função `creat` é equivalente à função `open` com as flags definidas por: `O_CREAT|O_WRONLY|O_TRUNC`.

→ **fecho de um descritor:**

```
int close(int fd);
```

Operações sobre ficheiros (cont.)

→ **leitura e escrita:**

```
ssize_t read(int fd,void *buffer,size_t count);
```

lê `count` bytes do ficheiro cujo descritor é `fd` para o `buffer`. Devolve o número de bytes lidos ou `-1` se ocorrer um erro.

```
ssize_t write(int fd,void *buffer,size_t count);
```

escreve para o ficheiro cujo descritor é `fd`, `count` bytes a partir de `buffer`. A função retorna o número de bytes escritos, ou `-1` se ocorrerem erros.

Operações sobre ficheiros (cont.)

→ **acesso aleatório:**

```
off_t lseek(int fd, off_t offset, int whence);
```

posiciona a "cabeça de leitura" associada ao descritor `fd` numa posição definida por `offset`.

`offset` representa o deslocamento a fazer sobre a posição corrente no ficheiro e `whence` define como interpretar o `offset`. Valores para `whence`:

`SEEK_SET` - posição corrente passa para `offset` bytes.
`SEEK_CUR` - adicionar `offset` à posição corrente.
`SEEK_END` - subtrair `offset` à posição corrente.

`offset= lseek(fd,0,SEEK_CUR)` – determina a posição corrente
`new_offset= lseek(fd,1024,SEEK_SET)` – posição corrente fica no byte 1024.

Exemplo: junção de dois ficheiros

Para simplificar não se inclui código de teste a situações de erro.

```
#define FLAGS O_WRONLY|O_CREAT|O_APPEND

main(int argc, char *argv[])
{
  int n, from, to;
  char buf[1024];

  from= open(argv[1], O_RDONLY);
  to= open(argv[2], FLAGS, 0644);

  while ((n= read(from, buf, sizeof(buf))) >0)
    write(to, buf, n);

  close(from);
  close(to);
  exit(0);
}
```