Sétima aula de FSO

José A. Cardoso e Cunha
DI-FCT/UNL

Este texto resume o conteúdo da aula teórica.

1 Objectivo

O objectivo da aula foi a continuação do estudo do sistema de ficheiros, através do exemplo do sistema Unix.

2 Criação de ficheiros

A criação de um ficheiro normal em disco, no Unix, faz-se pela operação *int create(char *name, int mode)*, que define o nome do ficheiro e os direitos de acesso que lhe ficam associados, para verificar posteriores acessos ao ficheiro.

Um utilizador, num sistema interactivo como o Unix, tem associado um *nome de utilizador*, previamente registado em ficheiros geridos pelos administradores do SO, e que é verificado na entrada em cada sessão interactiva ao terminal, juntamente com a senha de entrada (*password*). O administrador do SO tem, habitualmente, associado o nome de utilizador *root*.

O nome de utilizador identifica as actividades do utilizador durante a sua sessão de trabalho, relativamente a todos os comandos executados e todos os ficheiros criados ou acedidos.

O controlo dos acessos aos ficheiros baseia-se em três categorias de utilizador, relativamente às operações sobre cada ficheiro:

- o *dono* do ficheiro, ou seja, quem criou o ficheiro;
- o *grupo do dono*: um conjunto de utilizadores, que foi definido pelo administrador do SO;
- os *outros* utilizadores.

O acesso a cada ficheiro pode ser discriminado de forma selectiva, em função de cada uma das categorias na qual se situa o utilizador pedindo o acesso.

No exemplo seguinte, ilustra-se um caso de um ficheiro para o qual se permite: acesso para ler, escrever ou executar, pelo seu dono; ler e executar, pelos utilizadores do mesmo grupo; apenas ler, pelos outros utilizadores.

```
dono      grupo     outros

rwx       r-x       r--
111       101       100
 7         5         4
```

Mostram-se as mnemónicas **r***ead*, **w***rite*, *e***x***ecute*, a sua representação binária e octal (usada unicamente por razões históricas, ligadas à génese do Unix, nos computadores PDP, da Digital Equipment Corporation, nos quais se usava a notação octal, em vez da hexadecimal).

O admnistrador do SO, de nome *root*, tem acesso a todos os ficheiros, sem quaisquer restrições.

A chamada de *fd = create('/usr/f1', 0754)* criaria um novo ficheiro de nome simbólico '/usr/f1', e com os direitos de acesso indicados no exemplo. Se um ficheiro com este nome já existir, esta operação iria truncá-lo, ou seja, o ficheiro ficaria coercivamente vazio.

A operação *create* devolve, em fd, o número inteiro que identifica o canal virtual, que pode ser seguidamente usado para escrever no ficheiro (se quisermos ler, teremos de usar uma outra chamada ao SO, posteriormente, para abrir um canal virtual de leitura). Devolve o valor *-1*, em caso de erro na criação do ficheiro, e.g. falta de espaço em disco.

3 Abertura de canais para ficheiros no Unix

A operação *int open(char *name, int flags, int perms)* devolve o identificador inteiro de um canal virtual, aberto para o ficheiro de nome indicado, o que permite ler o ficheiro (se flags for O_RDONLY), escrevê-lo (se flags for O_WRONLY), ou ler e escrever (se for O_RDWR).

Outras opções do parâmetro *flags*, tais como O_CREAT, para criar um ficheiro (indicando os seus direitos de acesso em *perms*), ou O_EXCL, para abrir o ficheiro em modo exclusivo, podem ser consultadas no manual do Unix (comando *man*).

Exemplo: abrir um ficheiro só para leitura

fd = open(name, O_RDONLY, 0);

4 Leitura e escrita de ficheiros abertos

As operações:

*int read(int fd, char *buf, int n);*

*int write(int fd, char *buf, int n);*

permitem ler ou escrever num canal fd previamente aberto para um ficheiro. Os parâmetros buf e n indicam um vector de caracteres e o seu tamanho, para conter, respectivamente, os dados a receber ou os dados a enviar, numa leitura ou numa escrita.

Na figura 1 ilustra-se o caso da leitura de um ficheiro normal de disco, num SO como o Unix.

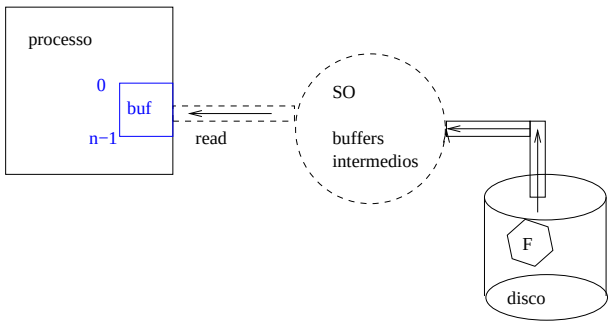

Figura 1: Leitura de um ficheiro

O valor n define um número qualquer de caracteres que se pretende ler, independentemente do tamanho do bloco de bytes que o SO usa internamente nos seus buffers ou na transferência de disco. A operação read nem sempre desencadeia uma leitura física de disco: como o SO usa buffers internos, para cada ficheiro aberto, acontece frequentemente que os dados pedidos pelo programa já se encontram em memória, naqueles buffers, por terem sido trazidos de disco, em leituras de blocos pedidos anteriormente. Note-se, contudo, que o buffer indicado como argumento de read define uma zona de memória do espaço de endereços do processo utilizador, pelo que há sempre uma cópia de bytes para este buffer.

Há casos em que o número de caracteres efectivamente lidos pela operação read e que é devolvido como resultado da chamada ao SO, é inferior ao número n de caracteres pedidos pelo programa:

- ao ler de um ficheiro normal em disco, se se atinge o fim do ficheiro antes de ter lido n bytes; o fim de ficheiro é indicadom quando, ao ter pedido um read, este devolve 0 bytes lidos;
- ao ler de um teclado, normalmente as rotinas de controlo lêem só uma linha de cada vez;
- ao ler de uma interface de rede, lê conforme o tamanho de buffers usados;

Em caso de erro, qualquer das operações devolve -1.

Exemplo: um processo que copia bytes da sua entrada 0 para a sua saída 1

```
main()

{ char buf[BUFSIZE];
  int n;

  ...

  while ((n = read(0,buf,BUFSIZE)) > 0)
          write(1,buf,n);

  ...

}
```

Este programa funciona bem, desde que ao ser criado o processo que o irá executar, os canais 0 (standard input) e 1 (standard output), tenham sido abertos para os ficheiros relativamente aos quais se pretendem copiar caracteres.

5 Acesso sequencial e acesso diecto a ficheiros normais Unix

As operações read e write, assumem, por omissão, o acesso sequencial, baseado na posição corrente do cursor (*byte offset*) associado ao canal que se abriu para um dado ficheiro (figura 2). Por exemplo, ao abrir um canal de leitura

para um ficheiro, o cursor deste canal fica inicializado a 0 e, por cada operação read ou write, vai avançando o número de posições correspondentes ao número de caracteres lidos ou escritos

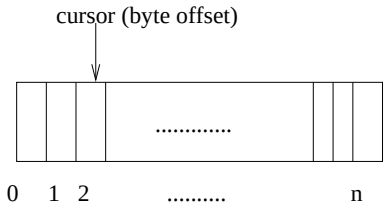

Figura 2: Cursor de byte num canal associado a um ficheiro

A operação *long lseek(int fd, long offset, int origin);* permite mover o cursor de byte do canal fd para uma qualquer posição, relativamente a uma posição de referência, indicada por origin, e devolvendo, como resultado, o valor da nova posição do cursor:

- origin = 0 indica que a referência é o início do ficheiro;
- origin = 1 indica que a referência é a posição corrente do cursor;
- origin = 2 indica que a referência é o fim do ficheiro.

Exemplos:
lseek(fd,0,2) põe o cursor no fim e devolve o número de bytes do ficheiro.
lseek(fd,0,0) põe o cursor no início e devolve 0.

É possível avançar com o cursor de um canal virtual, mesmo para além do fim real do ficheiro, por exemplo, veja a figura 3. No início o cursor tinha a posição c1, depois de movido para c2, pode-se escrever e obtém-se o resultado indicado.

Note-se que o conceito de cursor de byte é associado ao canal aberto para o ficheiro normal em disco e não ao próprio ficheiro. Isto significa que um mesmo processo pode abrir múltiplos canais para um mesmo ficheiro, uns para leitura, outros para escrita, e, para cada canal aberto, haverá um cursor de byte associado, que regista os avanços e recuos das operações de acesso ao ficheiro, invocadas através desse canal. Diferentes processos podem também ter canais abertos para um mesmo ficheiro, e logicamente com valores diferentes dos cursores, mas neste caso há que tomar precauções especiais para controlar os efeitos de interferência devidos à execução concorrente de operações desses processos sobre um mesmo ficheiro.

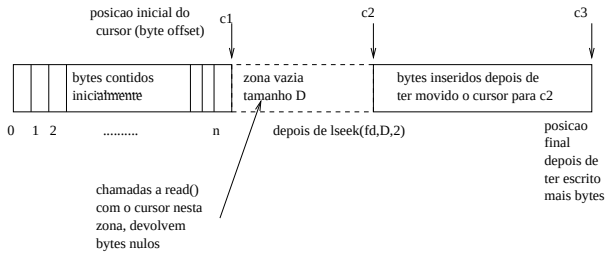

Figura 3: Cursor de byte num canal, para além do fim de um ficheiro

Como veremos adiante, o SO regista o valor do cursor de bye para cada canal, numa tabela global, gerida internamente pelo SO, e que tem uma nova entrada por cada canal aberto para um ficheiro. Todas estas entradas irão apontar para uma única entrada, noutra tabela globaldo SO, na qual, aí sim, estará a informação de localização do ficheiro em disco, bem como dos seus buffers intermédios em memória central. Todos os canais abertos para um ficheiro partilham o acesso a esses buffers, ainda que cada um possa estar localizado num cursor em posição diferente do ficheiro.

6 Fechar um canal aberto para um ficheiro

A operação *int close(int fd);* fecha a ligação entre o canal fd e o ficheiro respectivo (figura 4).

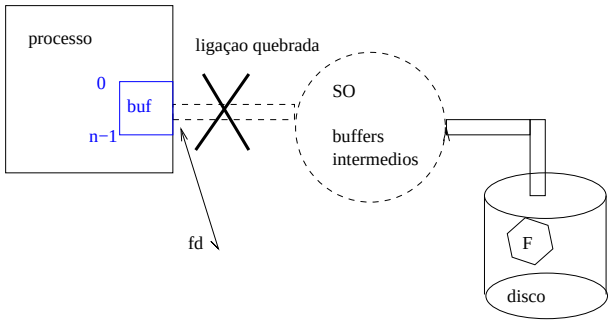

Figura 4: Fechar um canal, aberto para um ficheiro

Note-se que, na figura, se quebra a ligação do canal fd, aberto pelo processo indicado, às estruturas intermédias do SO. Note que o SO poderá

ainda ter de manter a ligação das suas estruturas intermédias ao ficheiro em disco, pois pode haver múltiplos canais abertos para um mesmo ficheiro: esta ligação só será quebrada quando náo houver mais quaisquer canais abertos para um dado ficheiro.

Após fechado um canal fd, o processo náo poderá mais invocar operações de read, write, lseek sobre esse canal. O ficheiro, contudo, permanece acessível, desde que o processo abra outros canais para lhe aceder.

Como o número máximo de canais aberto, por processo, tem um limite, imposto pelo tamanho máximo das entradas do SO, que registam as estruturas intermédias que representam os canais, é uma boa prática, a de fechar os canais logo que deixam de ser necessários no programa. Independentemente disso, o SO encarrega-se de fechar automaticamente todos os canais abertos por um processo, quando este termina.

7 Destruir um ficheiro

Na figura 5 ilustra-se a acção de destruição de um ficheiro, baseada na remoção da referência ao nome do ficheiro, na estrutura de directorias, que lista os nomes dos ficheiros existentes.

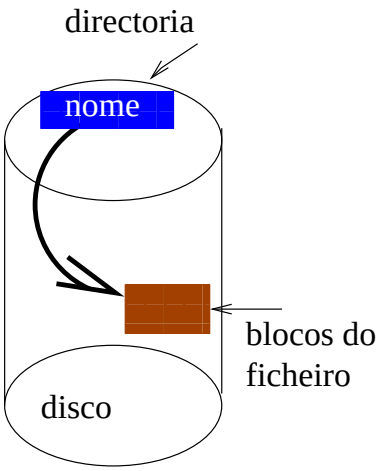

Figura 5: Remover o nome de um ficheiro de uma directoria

A destruição do ficheiro não corresponde à eliminação física dos seus blocos em disco, mas apenas à remoção do nome do ficheiro da estrutura de directorias do sistema. Deixando de ter nome, o ficheiro deixa de ser

acessível aos programas, que não podem abrir mais canais para o ficheiro.

A operação *unlink(char *name)* remove, do SO, o nome indicado do ficheiro. Esta operação tem esta designação por oposição ao da operação *link()* que, como veremos mais adiante, atribui um novo nome, isto é, um sinónimo (chamado *hard link* no Unix) a um ficheiro existente.

8 Directorias

Para manter o registo dos nomes simbólicos dos ficheiros, o SO mantém lista desses nomes, em estruturas chamadas directorias. Na verdade, cada entrada numa directoria dá acesso a informação sobre os diversos atributos do ficheiro, para além do nome:

- tipo de ficheiro: normal em disco, directoria, especial
- nome do dono
- direitos de acesso
- datas de criação, acesso, modificação de atributos
- blocos de dados, se for ficheiro normal em disco
- outras informações

A informação sobre os atributos do ficheiro, excluindo o nome, no caso do Unix, está reunida numa estrutura interna, designada por *i-node*. Um *i-node*, identificado por um número inteiro, identifica um ficheiro de forma únivoca, dentro de um cada 'sistema de ficheiros' (aqui usa-se o termo 'sistema de ficheiros' para denotar um determinado conjunto de ficheiros, por exemplo, contido num determinado espaço em disco).

No Unix, uma directoria é simplesmente uma lista de entradas, sendo cada entrada um par *nome,i-node*. Para efectuar as operações de acesso ao ficheiro, o SO precisa de localizar o seu i-node. Para isso, tem de pesquisar o seu nome nas directorias.

Nos primeiros SO, de um utilizador e de pequena dimensão, a lista dos nomes de todos os ficheiros poderia estar reunida numa única directoria. Num SO com múltiplos utilizadores, surgiu uma motivação para ter, pelo menos, uma directoria por cada utilizador. A organização mais geral e, portanto, mais flexível, permite ter uma hierarquia de directorias, conforme se ilustra na figura 6.

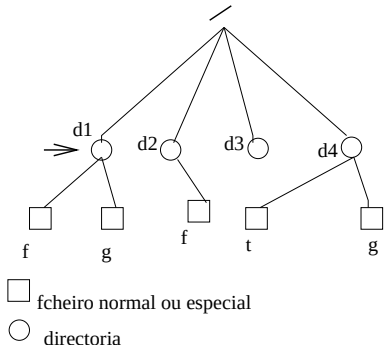

Figura 6: Hierarquia de directorias e ficheiros no Unix

Os nomes de ficheiros podem ser denotados através do seu nome composto absoluto, que identifica o seu caminho a partir da raiz da hierarquia, denotada por '/'. A raiz da hierarquia corresponde a uma directoria cujo i-node é o primeiro do sistema de ficheiros por ela encimado.

Um ficheiro, por exemplo, 'f' na directoria 'd1', é denotado por um nome absoluto (*pathname*) '/d1/f'. É possível ter dois ficheiros diferentes, mas com os mesmos nomes, localemente a duas directorias diferentes: por exemplo, o ficheiro 'f' na directoria 'd1' e o ficheiro 'f' na directoria 'd2', têm nomes absolutos diferentes, respectivamente, '/d1/f' e '/d2/f'.

Um ficheiro, por exemplo, '/d1/f', pode ser denotado por um nome relativo, desde que tenha uma referência em relação à qual esse nome se define. Existe, associado a cada processo no Unix, o conceito de *directoria corrente*, que permite usar, durante a execução de programas, nomes mais pequenos para os ficheiros, e relativos ao ambiente corrente da execução. Veja na figura 6, o apontador para a directoria '/d1': se esta for a directoria corrente de um processo, este pode aceder ao ficheiro '/d1/f' apenas pelo nome 'f'.

O conceito de directoria corrente (*working directory*) surge também associado ao ambiente de execução de comandos de um utilizador interactivo Unix, cuja interface é definida pelo programa interpretador de comandos de linha, designado por *shell*. Este interpretador, no arranque de uma sessão interactiva de trabalho, logo após a entrada (*login*) bem sucedida, define o valor de uma variável global do ambiente chamada *HOMEDIR* (directoria base), que é, à partida, a directoria corrente do processo *shell*. O utilizador pode ir alterando esta directoria corrente, através do comando do *shell* *cd*(*change directory*). Um processo pode alterar a sua directoria corrente através da chamada da

função *chdir*) e pode consultar o seu valor através da função *getcwd*) (*get current working directory*) (veja o manual do Unix).

9 Descritores de recursos no SO

Um SO precisa de manter estruturas de dados internas, que lhe permitem controlar o acesso aos recursos, quando os processos utilizadores invocam chamadas ao SO.

Dois exemplos de recursos são os *processos* e os *ficheiros*.

Para os processos, o SO mantém uma tabela interna de descritores de processos, cujo conteúdo descreve o estado da execução dos múltiplos processos que existem no sistema, num dado momento. A entrada correspondente a cada processo nessa tabela, é identificada através de um número interno, o identificador do processo (*process identifier, pid*).

Para os ficheiros, o SO mantém directorias, as quais, dado um nome simbólico de um ficheiro, permitem localizar o seu i-node interno, isto é, o descritor que contém os atributos do ficheiro.