Anatomia de um *Spooler* de impressão simplificado

José C. Cunha, DI-FCT/UNL

a) Operação da interface de uma impressora orientada para o *byte*

Considere que, num computador com um único processador, dispõe de uma impressora com uma interface *hardware* orientada para o *byte*. Como pode ver nos apontamentos de apoio, relativos à organização das entradas / saídas em arquitectura de computadores (disponíveis na página de FSO no CLIP, na secção Documentação de Apoio → Textos de apoio), uma tal interface tem diversos registadores hardware (chamados vulgarmente portas da interface), que são programáveis, isto é, um programa, em execução pelo CPU do computador, pode executar instruções máquina de entrada/saída, para ler o conteúdo ou escrever o conteúdo desses registadores. Desses registadores, distinguimos:

- um registador de controlo, com diversos *bits* indicando diversas operações sobre a impressora; por exemplo, para fazer *reset* da interface da impressora, o envio de um comando, pelo programa executando-se no CPU, para esse registador de controlo da interface da impressora, pode afectar um *bit* de *reset* a 1, originando a inicialização da interface, que assim fica pronta para operação;
- um registador de estado, que tem diversos *bits* indicando, por exemplo, situações de erro, ou um bit (vulgarmente dito de *Ready*) indicando que a impressora está disponível para receber um novo *byte* para impressão; este registador de estado pode ser lido por um programa em execução no CPU, quando executa uma instrução máquina de input, a qual faz uma cópia do registador de estado para um dos registadores do CPU, podendo então os seus bits serem testados, com instruções aritméticas e lógicas, executadas pelo programa no CPU;
- um registador de dados, que pode conter um byte, enviado por uma instrução máquina (instrução de *output*) executada por um programa no CPU, a qual envia um byte de um registador do CPU para esse registo; a recepção desse byte origina, geralmente, o início do processo físico de impressão do carácter correspondente ao código do *byte* recebido.

Agora considere que, nesse computador, está instalado um sistema de operação(SO) da família Unix. Como sabe (consulte os apontamentos da disciplina de FSO no CLIP), num sistema Unix, existe uma organização dos ficheiros em estruturas hierárquicas organizadas em directorias, em que se distinguem três tipos principais de ficheiros: as directorias (são ficheiros cujo conteúdo é uma lista de pares: <nome de ficheiro, número de descritor-no Unix chamado *i-node*>), os ficheiros ditos normais (que são os ficheiros que contêm sequências de bytes armazenados em disco) e os ficheiros ditos especiais (que são os ficheiros que representam, a um nível lógico, os dispositivos periféricos ligados ao computador, tal como o teclado, o écran, a impressora, etc.). A grande vantagem de modelar os periféricos também como se fossem ficheiros, é permitir realizar operações sobre eles, a um nível lógico, que permite ao programador ignorar os detalhes de muito baixo nível, que implica aceder directamente às portas das interfaces hardware, como as mencionadas acima para o caso da impressora indicada. Para além disso, o conceito de ficheiro especial permite ao SO disponibilizar apenas um pequeno e uniforme conjunto de operações para abrir/fechar canais, ler/escrever, conforme estudado nas aulas e descrito nos apontamentos (veja chamadas ao SO: open/close/read/write). Estas funções permitem ignorar as grandes diferenças que existem na operação e na natureza física de cada dispositivo periférico e assim fazer programas de controlo mais uniformes e independentes do *hardware*. Para esse efeito cada ficheiro especial, como qualquer ficheiro que se preze, tem um nome simbólico que aparece algures nalguma directoria na

estrutura do sistema de ficheiros. Em particular, tipicamente, existe uma directoria de nome absoluto ´/dev´, a qual contém os ficheiros especiais que são conhecidos de um SO. Por exemplo, à impressora acima referida pode assumir que corresponde um ficheiro especial ´dev/lp1´(lp – line printer).
Assim, torna-se possível, a um processo realizar operações tais como as seguintes:

```
fd = open (´/dev/lp1´, O_WRONLY);
write(fd, buf, BUFSIZE);
close(fd);
```

desencadeando, respectivamente as seguintes acções, efectuadas por aquelas chamadas ao sistema:
-- abrir um canal para o ficheiro especial que representa a impressora e guardar o seu número (dito *file descriptor*) na variável *fd*;
-- escrever o conteúdo do buffer *buf*, um vector de *bytes* definido como uma variável do programa daquele processo, naquele canal, com o efeito de desencadear o seu envio para a impressora;
-- fechar aquele canal.

Veja, consultando os apontamentos de FSO, de que modo são representados os canais abertos para ficheiros no Unix. Em resumo, por cada processo existe uma tabela de canais, com os canais correntemente abertos. A tabela é local a cada processo, quer dizer, cada processo tem a sua própria tabela. Esses canais são numerados a partir de 0 (a primeira entrada na tabela), até um limite de configuração do sistema. Em geral, quando um processo é criado, herda os canais do processo pai. No caso do Shell, cujos canais 0, 1 e 2, estão respectivamente ligados ao teclado, ao écran e ao écran, todos os processos criados pelo Shell, herdam aqueles canais e aquelas ligações, quando são criados.
A cada canal aberto por um processo, e registado como uma entrada na respectiva tabela de canais, existe associada uma ligação para uma outra tabela, que é uma estrutura de dados global, gerida pelos programas do SO. Note que tanto as tabelas de canais como esta tabela global (dita a tabela global de ficheiros abertos), são estruturas de dados internas e geridas exclusivamente pelo SO. Assim, por cada canal aberto, existe uma entrada na tabela global de ficheiros abertos, sendo que esta entrada tem a seguinte informação, no caso de um ficheiro normal: o valor do cursor de byte corrente (ou byte offset), que aponta o byte corrente (note que um ficheiro no Unix é visto como uma sequência lógica de N bytes, numerados de 0 até N-1); o modo em que o canal foi aberto, no exemplo acima, modo write; e uma ligação para outra tabela do SO, onde estão os descritores dos ficheiros (ditos i-nodes) onde estão descritos os atributos respectivos dos ficheiros. No exemplo acima, o canal fd aberto por open, fica assim ligado, não só à tabela global de ficheiros abertos, como também à tabela de i-nodes. Quer dizer que, de cada vez que uma chamada ao SO *write (fd,....)* for invocada, o SO sabe determinar onde encontrar a descrição do ficheiro correspondente e assim determinar de que modo há-de realizar a operação pedida.

Como no nosso caso, o ficheiro é especial ´/dev/lp1´ e representa uma impressora, isto quer dizer que o respectivo *i-node* correspondente a ´/dev/lp1´ deve lá ter indicação de que se trata de um ficheiro especial. Note que é também a partir do *i-node* de cada ficheiro que o SO determina a localização dos *buffers* em memória onde estão os *bytes* que, em sucessivas operações de *write*( ), vão sendo enviados por um processo para a impressora. E assim, é a partir desses *buffers*, que o SO vai controlar, consoante a natureza da impressora, de que forma deve ir enviando os *bytes*, neste caso um de cada vez, pois a nossa impressora é orientada para o *byte*. Em particular, quando o write() é invocado e o SO determina, pelo i-node de ´/dev/lp1´, que se trata de um ficheiro especial, vai identificar as rotinas de controlo que devem

ser responsáveis pelo envio de bytes para a interface da impressora, em particular o 1º byte, pois a impressora é orientada para o byte, no nosso exemplo. Essas rotinas, que devem efectuar operações de acesso às interfaces hardware dos dispositivos, estão integradas em módulos que se designam tipicamente por Device Drivers. Existe um Device Driver por cada classe de dispositivo periférico. Se houver várias unidades periféricas de uma mesma classe, cada interface tem um código próprio e as rotinas do Device Driver sabem assim distinguir a quem enviar os bytes ou de onde os receber. Em conclusão, vê-se que, para os ficheiros especiais, é através dos inodes, que o SO localiza as rotinas dos Device Drivers, a quem deve passar os pedidos de operações necessárias para efectuar as acções das chamadas ao SO. Neste caso o pedido de write na impressora.

Uma impressora orientada para o byte só tem capacidade para receber, no seu registador de bytes, um byte de cada vez. Assim, se assumirmos que a impressora está inicialmente livre e o programa no CPU envia um byte, através de uma instrução máquina de output, para esse registador de dados, então a interface da impressora desencadeia a impressão automática desse byte. E quando a impressão acabar, o bit Ready do registador de estado é posto a 1, indicando que a impressora está livre para receber o próximo byte. E no caso de a impressora estar programada para gerar pedidos de interrupção ao CPU, então na situação em que o bit Ready vem a 1, a interface hardware da impressora coloca um pedido de interrupção no Bus hardware do computador, através de uma linha que a liga ao CPU. Quando a lógica hardware de tratamento de interrupções, no CPU, identificar e decidir atender esse pedido, então o programa que esteja em execução no CPU, é interrompido, o seu estado (PC e Flags) são salvaguardados no topo da pilha, o modo de operação do CPU passa de modo utilizador a modo supervisor e o fluxo da execução salta para um endereço de uma rotina (dita Rotina de Serviço de Interrupções – RSI), que vai tratar o pedido de interrupção. Neste caso, essa rotina RSI vai ao buffer associado à impressora (para onde haviam sido copiados os bytes enviados quando o programa invocou write()) e obtém o próximo byte, que envia para o registador de dados da interface hardware da impressora, desencadeando a sua impressão. Assim feito, a rotina RSI acaba e executa uma instrução máquina de retorno ao programa que havia sido interrompido (o seu estado é salvaguardado a partir do topo da pilha, e o modo de operação do CPU volta a modo utilizador). (Veja o exemplo de programação de uma impressora, para fazer sair uma cadeia de caracteres, apresentado nos acetados na página do CLIP)
Como se compreende, a rotina RSI faz também parte do módulo Device Driver da impressora, sendo invocada por cada pedido de interrupção que a impressora faz. Ao passo que a outra rotina que é invocada para enviar o 1º byte, quando o programa chamou a função write() ao SO, também faz parte do módulo Device Driver da impressora. Note que a rotina RSI é invocada a partir do hardware e a rotina de write é invocada a partir do programa.

Antes de prosseguirmos: porque é que um programa em modo utilizador não pode enviar *bytes* directamente para a interface (registador de dados) da impressora, executando uma instrução máquina de *output*? Não pode, porque um programa em modo utilizador não tem privilégios para aceder às interfaces hardware pois que o controlo dos periféricos fica inteiramente sob a responsabilidade do SO. Caso contrário, qualquer programa utilizador poderia comprometer, monopolizar ou corromper a boa utilização dos periféricos. Para ter acesso às interfaces hardware dos periféricos e poder executar operação de envio/recepção dos seus registadores, um programa tem de ser executado com o CPU em modo supervisor. É esse o caso dos programas que implementam as chamadas ao sistema, tal como write(), pelo que assim já podem enviar bytes para a interface da impressora. A única coisa a que o programa, em

modo utilizador, tem direito é invocar as chamadas ao SO, como write, fazendo o pedido para o SO desencadear a escrita pedida.

b) Impressão de um ficheiro por um programa utilizador

Considere agora que uma aplicação, isto é um programa executado em modo utilizador, pretende fazer imprimir um ficheiro naquela impressora, cujo nome simbólico é ´/dev/lp1´.

Como sabe, o acesso aos ficheiros no Unix é protegido por um mecanismo de permissões e direitos de acesso. A cada (*i-node* de) ficheiro está associado um conjunto de direitos de acesso, para os diversos utilizadores considerados: dono (*owner*), grupo e outros. (veja os acetatos e apontamentos de FSO no CLIP).
Para um programa ter direito de escrever (invocar a chamada ao SO write()) na impressora, então o utilizador que invocou o programa terá de ter acesso para escrita (´´w´´) no ficheiro ´/dev/lp1´.

Imagine que todos os programas utilizadores tinham esse direito de acesso ao ficheiro especial que representa a impressora. Então poderiam fazer a sequência de acções:

```
fd = open (´/dev/lp1´, O_WRONLY);
write(fd, buf, BUFSIZE);
close(fd);
```

para irem enviando um ficheiro, como uma sucessão de *buffers* para a impressora.

Se houvesse a garantia de que cada programa utilizador executava todas as invocações de wwrite, necessárias para imprimir um dado ficheiro F1, sem que qualquer outro programa começasse uma sequência de acções idêntica, para imprimir um outro ficheiro F2, então a solução acima seria eventualmente aceitável, pois cada ficheiro (F1 ou F2) apareceria numa listagem coerente, ou seja, primeiro a listagem de F1 e só depois a de F2.

Mas tal garantia não se tem, num SO que suporta multiprogramação. Como se sabe, neste regime, múltiplos programas executam-se de forma concorrente, pois o SO vai decidindo a cada momento, qual o processo que deve obter oportunidade de execução, dependendo da sua prioridade, de não estar bloqueado à espera de dados, etc. Assim, havendo dois ou mais programas, em multiprogramação, a executar a sequências de acções acima, indicada, o resultado seria que as listagens dos ficheiros F1 e F2 iriam aparecer misturadas de forma aleatória, na impressão.

Como resolver este problema?

c) *Spooler* de impressão

A melhor solução é delegar a responsabilidade da impressão, num processo dedicado a esse fim. Esse processo, a que chamaremos Spooler de impressão, será inicializado pelo SO, e vai executar um programa, cujo dono é o superuser (ou root), ou seja o utilizador que, num sistema Unix, tem todos os

privilégios e permissões. O ficheiro especial ´/dev/lp1´ terá como permissões, apenas para esse utilizador, excluindo todos os outros utilizadores, que assim ficarão impedidos de invocar operações de open() e write() sobre ´/dev/lp1´. Apenas o programa do Spooler de impressão poderá invocar essas operações.

Tendo desta forma garantido que só o Spooler pode aceder ao ficheiro especial ´/dev/lp1´ que representa a impressora, note que o processo Spooler pode executar-se com o CPU em modo utilizador, pois que todos os acessos à impressora se farão indirectamente, via chamadas ao SO.

A outra garantia de que precisamos é a de que o Spooler processe a impressão dos ficheiros de forma estritamente sequencial, de modo a evitar entrelaçamento das listagens de ficheiros diferentes. Para isso, desenhamos o processo Spooler como um servidor que executa um ciclo de acções do seguinte tipo (omitem-se declaracoes de variáveis neste pseudo-código bem como testes de situações de erro):

```
InicioSpooler:  fd-lp = open(´/dev/lp1´, O_WRONLY);   // abre canal para a impressora
InicioCiclo:   AguardarPedidoImpressao(F);            //obtém nome de ficheiro F a imprimir
               fd = open(F, O_RDNLY);                 // abre canal para F, para ler
          while ((n = read(fd, buf, BUFSIZE)) > 0)    // lê blocos de bytes para buf
               nw = write(fd-lp, buf, n);             // pede escritas de buf na impressora
               close(fd);                             // após fim de ficheiro, fecha canal
               repete InicioCiclo;                    // volta para início, para obter próximo ficheiro
```

A primeira acção é abrir um canal para o ficheiro especial que representa a impressora pois vai ser preciso através desse canal pedir a escrita dos sucessivos buffers que irão ser lidos de cada ficheiro.
Depois é preciso, antes de iniciar cada ciclo de impressão, esperar que haja pedidos de impressão feitos pelos programas utilizadores. Admitimos que esses pedidos são colocados numa fila de pedidos, uma estrutura de dados em memória, na qual cada programa utilizador pode colocar um pedido, em ordem FIFO, através de uma função ColocarPedidoImpressão(F), sendo F o nome simbólico absoluto do ficheiro que se pretende imprimir. Admita que esta função faz uma cópia de F e coloca o nome da cópia na fila.
Assim, por cada pedido, quando este chegar à sua vez de ser atendido pelo Spooler, desencadeia-se uma impressão de forma sequencial. O processo Spooler limita-se a invocar a operação AguardarPedidoImpressao(F) que se admite devolve em F, o nome do ficheiro a imprimir e que, em caso da fila de pedidos estar vazia (como sucede no início, antes de haver quaisquer pedidos), garante que o processo Spooler passa ao estado Bloqueado, até um pedido ser colocado na fila. As formas possíveis de implementação desta operação serão estudadas mais adiante nesta disciplina.

Em conclusão, através desta estratégia, baseada em delegar no processo Spooler o tratamento sequencial dos pedidos de impressão conseguimos que uma impressora, que é fisicamente partilhada por múltiplos utilizadores, possa ser utilizada com garantia de coerência das listagens produzidas. Note que tal regime é possível mesmo que haja múltiplos utilizadores em diferentes máquinas numa rede de computadores, desde que o acesso à impressora esteja centralizado através do único processo Spooler, que controla a fila de pedidos e faz os pedidos de acesso à impressora.

Note ainda que, como se viu, o processo Spooler pode executar-se em modo utilizador, como qualquer outra aplicação, com a única diferença que tem de ser o único com permissões de acesso ao ficheiro especial que representa a impressora (´/dev/lp1´).