Quinta 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 o estudo dos ambientes de multiprogramação.

2 Resumo de vantagens da multiprogramação

A primeira vantagem é a de suportar a execução de múltiplos programas independentes como ilustrado na figura 1.

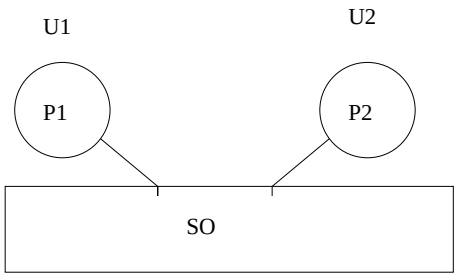

Figura 1: Utilizadores concorrentes

O SO comuta alternadamente a execução entre os processos P1 e P2, associados a cada um dos utilizadores, para garantir que ambos são atendidos de forma equilibrada, ou seja, com um tempo de resposta aceitável. Para um computador cujo relógio interno tenha uma frequência da ordem de GigaHertz, o SO consegue suportar dezenas de utilizadores interactivos, mesmo que haja apenas um CPU. Cada utilizador tem tempos de resposta da ordem dos segundos, pelo que não se apercebe dos tempos em que o seu programa não está em execução.

A segunda vantagem da multiprogramação é tornar mais rentável a utilização do CPU. Quando um programa invoca READ e tem de aguardar pelos dados, que podem demorar um tempo indeterminado ou tempos muito elevados (ex, da ordem dos milisegundos, se estiverem a ser lidos de um disco), o SO põe outro programa em execução.

A mesma vantagem ocorre no caso da saída de dados, ou seja, quando um processo não consegue mais escrever num buffer cheio, ficando então bloqueado e dando oportunidade a outros processos.

A terceira vantagem é a de permitir a cooperação entre processos concorrentes numa mesma aplicação, como se ilustra na figura 2.

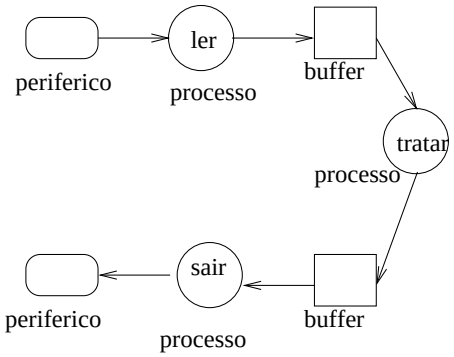

Figura 2: Processos concorrentes

Na perspectiva do SO a multiprogramação tem a vantagem de melhorar o rendimento de utilização do CPU e dos periféricos. Na perspectiva do utilizador tem a vantagem de lhe permitir explorar tarefas concorrentes, por exemplo, editar e compilar, concorrentemente com a impressão de um ficheiro. Outra vantagem é a de permitir múliplos utilizadores concorrentes que, partilhando o tempo do CPU, obtêm assim uma ilusão de que cada um dispõe de um computador dedicado, quando, afinal o que se passa é o controlo, pelo SO, de quais os momentos oportunos para comutar de um processo para outro.

3 Sistemas interactivos

A figura 3 ilustra um SO de multiprogramação que possibilita a execução concorrente de múltiplos processos, cada um dos quais executa um programa interpretador de comandos, dedicado à interacção com um utikizador através

de um terminal (teclado e écran).

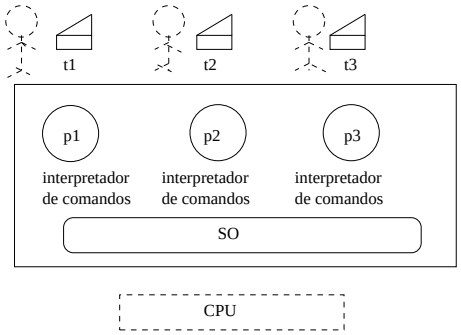

Figura 3: Sistema *time-sharing* com utilizadores interactivos

A figura 4 esquematiza a interacção de cada utilizador com um processo interpretador da linha de comandos inseridos pelo utilizador ao terminal *shell*.

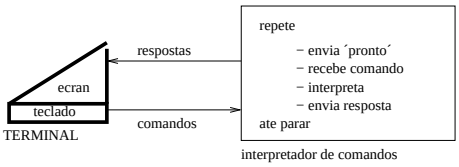

Figura 4: Sistema *time-sharing* com utilizadores interactivos

4 Máquina virtual suportada pelo SO

Na perspectiva do utilizador ao terminal, um sistema de multiprogramação aparece da forma esquematizada na figura 5

Um utilizador ao terminal, após um controlo de acesso, que verifica se o seu nome e senha estão correctamente registados, passa a ter acesso ao interpretador de comandos de linha do terminal, habitualmente designado por *shell* no SO Unix. Através deste programa, o utilizador tem acesso aos *recursos* registados no sistema, que são em geral acedidos sob a forma de ficheiros: de código (ditos executáveis), de texto, de imagem, etc. Através de programas de aplicação, eles próprios armazenados sob a forma de ficheiros

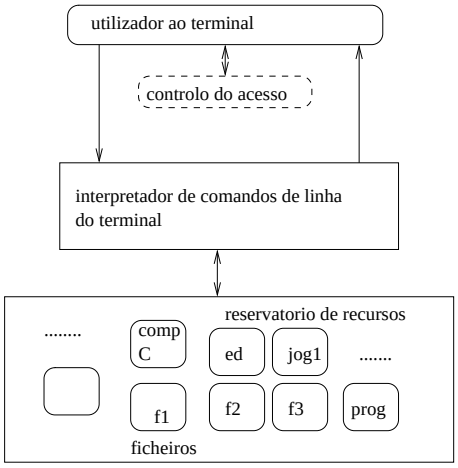

Figura 5: Máquina virtual oferecida ao utilizador

executáveis, pode-se ter acesso aos ficheiros de dados, e fazer a sua interpretação mais adequada destes.

De forma a ser o mais genérico possível, o SO apenas precisa de suportar um conjunto de mecanismos básicos, sobre os quais se podem implementar os programas de aplicação:

- Sistema de ficheiros: são os programas do SO que gerem a forma como os ficheiros são acedidos pelo utilizador, a forma como são armazenados, em disco ou noutros periféricos, e a forma como estão organizados em directorias.

- Controlo das operações de entrada e saída: inclui não só o acesso aos ficheiros, mas também o controlo da entrada e saída de dados, e o tratamento dos pedidos de interrupção gerados por cada periférico.

- Gestão de memória: a gestão do carregamento dos programas em memória central e o modo como as regiões de código, dados e pilha dos processos são gerids em memória.

- Gestão de processos: o controlo da execução de múltiplos processos concorrentes, através dos mecanismos de multiprogramação.

Estes mecanismos suportam o ambiente de execução dos programas. Alguns desses mecanismos são garantidos automaticamente pelo SO, isto

é, sem ser necessário que os programas utilizadores os invoquem através de chamadas ao SO explícitas: é, por exemplo, o caso da gestão das interrupções e da gestão da multiprogramação. Outros, pelo contrário, têm de ser invocados explicitamente pelos programas: é o caso das operações sobre ficheiros ou da activação de outros programas (envolvendo o seu carregamento em memória e a iniciação da sua execução).

Um programa em execução num computador tem acesso a:

- instruções da máquina hardware,
- chamadas a rotinas das bibliotecas de suporte a cada linguagem (ex, para processar cadeiras de caracteres em C),
- chamadas a rotinas do SO

Para além disso, o programa em execução tem garantido um ambiente protegido, a partir do qual pode comunicar com o exterior (ler e escrever dos periféricos).

A figura 6 esquematiza a máquina virtual que caracteriza o ambiente de execução de um programa num SO. A máquina tem três componentes principais que são versões abstractas dos componentes principais de um computador.

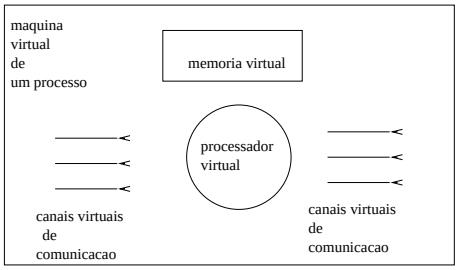

Figura 6: Máquina virtual de um processo num SO

4.1 Processadores Virtuais

As figuras seguintes ilustram 4 cenários de arquitecturas de computadores, nos quais os componentes abstractos da máquina virtual podem ser implementados.

A figura 7 representa um monoprocessador. Aí cada processo tem de esperar a sua vez para execução, visto só haver um CPU no computador. Um

SO de multiprogramação pode gerir múltiplos programas concorrentes, pelo que convém que estes possam estar simultaneamente em memória central (RAM), evidentemente em zonas diferentes de memória.

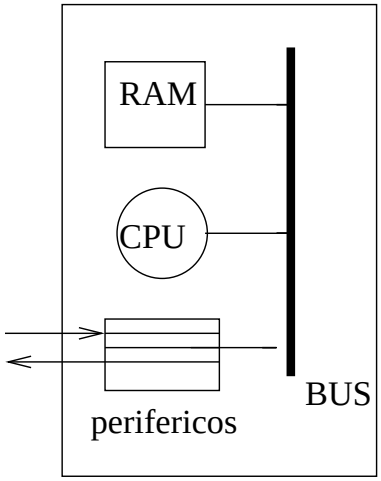

Figura 7: Monoprocessador: um só CPU

A figura 8 representa um multiprocessador, isto é, um computador com mais do que um CPU. Todos os CPU têm acesso à mesma memória central, através de um mesmos BUS (com linhas de dados, endereços e controlo). A memória é partilhada por múltiplos programas, que podem estar em execução *simlutânea*, em CPU diferentes.

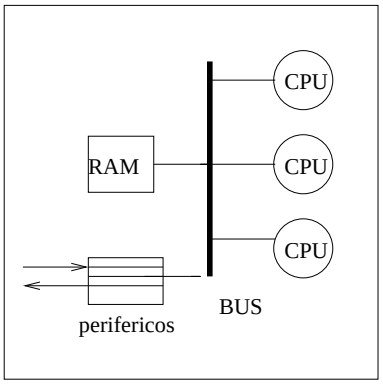

Figura 8: Multiprocessador com memória partilhada

A figura 9 ilustra um multiprocessador como a figura 8, mas em que cada CPU tem agora uma memória privada ou local, a que é o único a poder aceder. Esta memória privada pode ser apenas uma *Cache* ou ser uma RAM local. Continua a haver uma memória partilhada, mas, havendo memórias locais a cada CPU, é de esperar que a frequência de conflitos no acesso ao BUS comum seja inferior ao caso da figura 8. Neste caso, em cada memória local de um dado CPU podem estar o código, parte dos dados e a região de pilha dos processos que devem ser executados por esse CPU, ficando, na memória partilhada, apenas os dados que devem poder ser lidos ou escritos por qualquer dos processos, seja qual for o seu CPU. Os dados partilhados podem servir, por exemplo, para suportar uma região de comunicação entre processos.

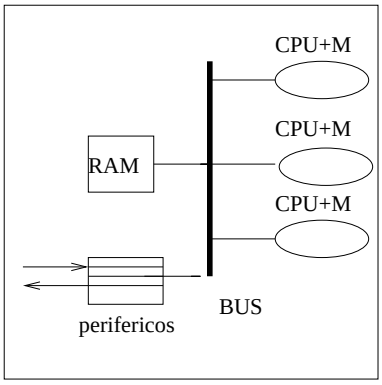

Figura 9: Multiprocessador de memória partilhada, e com *Caches*

A figura 10 ilustra um esquema muito simplificado do que acontece numa rede de computadores. Múltiplos computadores, cada um dos quais pode ser de tipos diferentes (um monoprocessador, um multiprocessador, sem ou com memórias locais), estão ligados por uma estrutura de interligação, baseada geralmente num esquema de comunicação de *bits* em série. Neste caso, em cada computador poderão ser executados múltiplos processos, seja em regime de multiprogramação, seja em regime de processamento paralelo, se esse computador tiver múltiplos CPU. As regiões de memória de cada processo residirão naturalmente na memória do computador respectivo. Em geral, um processo executar-se-á numa mesma máquina, do princípio ao fim.

Contudo, um processo pode, por exemplo, bloquear-se quando estava em execução num computador, esperando, por exemplo, para ler dados, e se estes dados chegam a outro computador, temos duas possibilidades: transferir os

dados para o computador onde possam ser lidos pelo processo que estava bloqueado, ou 'transferir o processo' para o computador onde estejam os dados. Este último caso (com a designação de *computação móvel*) exigirá mais trabalho, pois há que transferir todas regiões do mapa de memória do processo (código, dados e pilha) para o outro computador, mas há situações onde isto tem de ser feito. Imagine, por exemplo, que um computador não pode executar, por falta de memória local, mais do que um processo de cada vez e não tem capacidade de disco, nem para guardar os ficheiros executáveis todos de que necessita: nesse caso poderia, quando necessário, ir buscar o ficheiro executável a outro computador para executar esse programa no próprio computador.

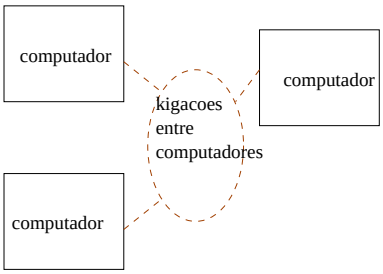

Figura 10: Rede de computadores

4.2 Memória Virtual

A memória virtual baseia-se num espaço de endereços, dito *virtual*, que é acessível idealmente a um programa em linguagem máquina, conforme a definição dos campos de endereços e dos modos de endereçamento das instruções de referência de memória de um dado processador. A figura 11 ilustra a relação entre os mapas de memória virtual de diversos processos e a sua localização nas unidades de memória física do computador: memória central (RAM) e memória secundária (disco).

A memória virtual é um conceito que garante uma memória dedicada à execução de cada programa, contendo, portanto, regiões lógicas de código, dados e pilha, que são geridas pelo SO, de forma transparente ao processo utilizador. Isto significa que o espaço de endereços virtuais de cada processo tem um tamanho máximo independente da capacidade de memória real disponível num dado computador. Significa também que o programa pode ignorar a localização física das suas regiões lógica de código, dados e pilha,

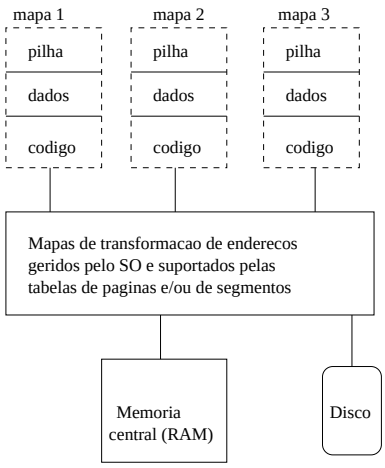

Figura 11: Memória virtual

pois o SO e o hardware de transformação de endereços encarregam-se de, a cada referência gerada pelo programa, localizar a correspondente célula, em memória central ou em disco. Por outro lado, este mecanismo garante a separação lógica e física – protecção – entre os espaços de endereçamento dos diferentes processos concorrentes.

4.3 Canais virtuais de entrada e de saída

Para que um programa possa comunicar com o exterior, durante a sua execução, é necessário que um processo possa abrir *canais de entrada e de saída* e possa ligá-los aos diversos tipos de dispositivos periféricos (teclado, écran, impressora, disco) ou ligá-los aos ficheiros em disco.

Quanto à comunicação de um processo com o seu ambiente exterior, distinguimos três principais objectivos de um SO:

- uniformizar a interacção com a maior diversidade possível de dispositivos, por mais diferentes que sejam as suas características físicas; isto é possível se definirmos um conjunto de comandos abstractos, para ler/escrever/controlar operações de entrada e de saída, e garantirmos que tais comandos se podem aplicar a qualquer tipo de dispositivo, com as necessárias adaptações;

- suportar a ilusão de que um processo dispõe de periféricos virtuais, isto é, que lhe sejam dedicados, ainda que estes sejam, na realidade,

suportados por dispositivos físicos partilhados por múltiplos utilizadores ou processos concorrentes; um exemplo é dado por uma impressora, partilhada por múltiplos utilizadores, mas controlada por um processo SPOOL de saída, que garante que dois pedidos de listagens não ficam entrelaçadas; outro exemplo é dado por um disco físico, partilhado pelos múltiplos utilizadores de um computador, para guardarem os seus ficheiros, ainda que na realidade todos utilizem um mesmo suporte físico;

- esconder, ao programa utilizador, a natureza física e os pormenores de operação dos dispositivos periféricos (e.g. a ocorrência de interrupções é escondida a um processo em execução).

  O primeiro objectivo exige que o SO possa designar os diversos dispositivos por nomes simbólicos, lógicos, que sejam independentes do seu suporte físico, isto é, que não se refiram explicitamente aos endereços das portas das interfaces hardware de i/o.

  Uma forma que o SO Unix encontrou para satisfazer esta exigência foi a de reduzir todo e qualquer dispositivo periférico ao conceito de *ficheiro*, e dotar estes de uma forma de nomeação única e independente do suporte físico. No Unix, um teclado, um écran, uma impressora, um disco, ou melhor dizendo, as suas interfaces de i/o, são nomeadas como se fossem ficheiros, através de uma mesma notação, que os identifca de forma única, como nós de uma estrutura hierárquica. E são acedidos por um mesmo conjunto de primitivas (e.g. *open, read, write, close*), que são as mesmas para manipular os ficheiros, ditos normais, em disco.

  Esta matéria - o estudo dos Sistemas de Ficheiros – é o objecto da próxima aula.