Aula 18 de FSO

José A. Cardoso e Cunha
DI-FCT/UNL

Este texto resume o conteúdo da aula teórica.

1 Objectivo

Objectivo da aula: Exemplo de servidores sequenciais e concorrentes, com um número fixo ou com um número variável de processos trabalhadores.

2 Servidor sequencial

Um servidor sequencial tem o seguinte esquema típico (figura 1).

```
WHILE true DO
BEGIN
  receber(cliente, mensagem);
  extrair (mensagem, servico, parametros):
  CASE servico OF
    ....
    id: BEGIN
          executar−servico[id](parametros, resultados);
          enviar(cliente, resultados)
        END
    ....
  END
```

Figura 1: Servidor sequencial.

Conforme o tipo de pedido, o cliente pode limitar-se a fazer o pedido e prosseguir a sua execução (por exemplo, quando se pede a impressão de um ficheiro ao servidor de SPOOL), ou pode ter de aguardar uma resposta (por exemplo, quando se consulta um servidor que dá acesso ao sistema de ficheiros ou a uma base de dados).

Em qualquer dos casos, a interacção entre clientes e servidores pode ser programada recorrendo quer a um modelo de comunicação baseada em memória partilhada ou baseada em mensagens. Em qualquer dos casos, o esquema da figura 1 aplica-se, com as diferenças de implementação das operações de recepção de pedidos e de envio das respostas.

Havendo múltiplos clientes concorrentes, o tratamento sequencial dos pedidos pode originar apreciáveis tempos de espera, a não ser que o tratamento de cada pedido seja praticamente 'imediato', por exemplo, envolva apenas a simples consulta de uma estrutura em memória.

Em geral, contudo, há pedidos que envolvem por exemplo acessos a disco, pelo que demoram um tempo apreciável, e há outros pedidos cujo tempo de tratamento é indeterminado. Um exemplo do segundo tipo de pedidos corresponde ao caso em que um cliente faz um pedido a um servidor S1 que, por sua vez, precisa de pedir a um outro servidor S2 que realize uma parte do tratamento pedido e o servidor S2 pode demorar um tempo imprevisível a responder a S1, por exemplo, devido a sobrecarga de processos no sistema, num dado momento. Nesse caso, se o servidor S1 é sequencial, todos os pedidos de outros clientes de S1 ficam pendentes, aguardando pelo tratamento do pedido corrente, que está bloqueado aguardando a resposta de S2. A figura 2 ilustra este esquema de dependências, que torna o modelo de servidor sequencial inaceitável nestes casos.

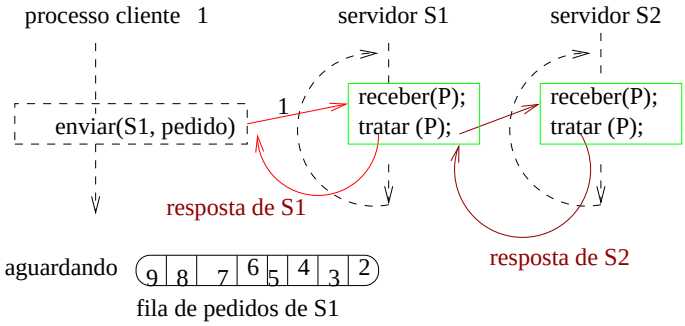

Figura 2: Pedido encadeado, com um servidor sequencial.

O modelo sequencial, nestes casos, é tanto mais inaceitável quanto possa haver outros pedidos pendentes na fila de S1, que pudessem ser tratados de forma quase imediata e que, assim, se vêem impedidos de serem atendidos, quando o próprio servidor S1 está inactivo, aguardando a resposta de S2.

3 Servidor concorrente

O problema anterior justifica que se organize um servidor como um conjunto de processos concorrentes, que cooperam, distribuindo entre si o tratamento dos pedidos feitos por múltiplos clientes. A organização clássica corresponde a ter um processo *vigilante* que está atento à chegada de novos pedidos, encarregando-se também da sua distribuição pelos processos *trabalhadores* disponíveis. Estes processos podem ser todos criados antecipadamente, constituindo um reservatório (*pool*) de capacidade fixa, ou podem ser criados dinamicamente, isto é, pelo processo vigilante, logo que um novo pedido chega. A vantagem do modelo dinâmico, em apenas gerar trabalhadores quando estes são mesmo necessários, é, contudo, contrabalançada pela sobrecarga (*overhead*), em tempo de execução e em ocupação de espaço de memória, devida à criação de novos processos (no Unix, através da chamada ao SO *fork()*...).

Evidentemente, pode haver variantes correpondentes a casos intermédios, em que se começa, por exemplo, com um certo número de trabalhadores e, conforme o número de pedidos que afluem ao servidor esteja num crescendo ou num diminuendo, assim, se vão criando novos trabalhadores ou se vão deixando destruindo alguns. Este ajuste dinâmico do número de trabalhadores, em função da afluência dos pedidos, pode ser feito pelo vigilante, desde que este disponha de um mecanismo de monitoração que o informe da carga de pedidos em certos intervalos de tempo.

As figuras 3 e 4 ilustram os dois esquemas, o estático e o dinâmico.

Mesmo nas variantes que combinam os dois métodos seria vantajoso podermos dispor de um mecanismo de criação de processos o mais eficiente possível (isto é, rápido e sem exigir muito espaço de memória). É esta uma das motivações do conceito de *processo leve* (*lightweight*), mais habitualmente designado por *thread*. Embora possamos traduzir *thread* para o termo *fio de execução*, no que se segue usaremos a designação *processo* para ndicar o conceito clássico de processo Unix e a designação de *thread* para indicar este novo conceito de processo leve, que explicaremos em pormenor numa das aulas seguintes. O leitor fica, entretanto, avisado de que estas designações podem assumir diferentes significados noutros contextos (e noutros textos...).

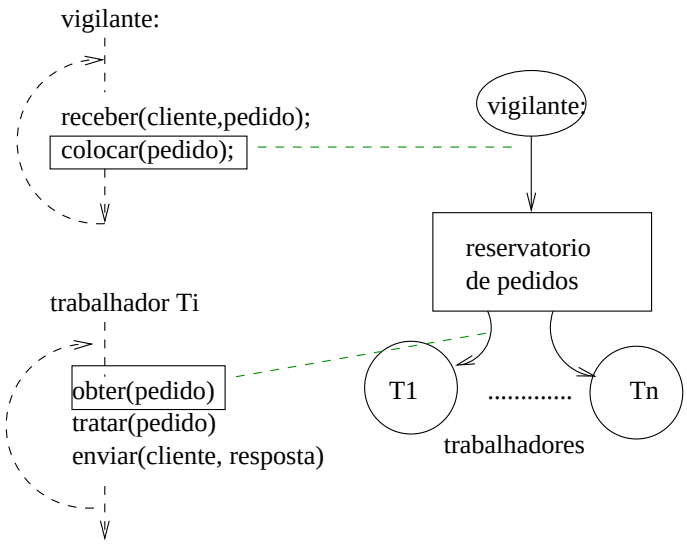

Figura 3: Servidor concorrente - estático.

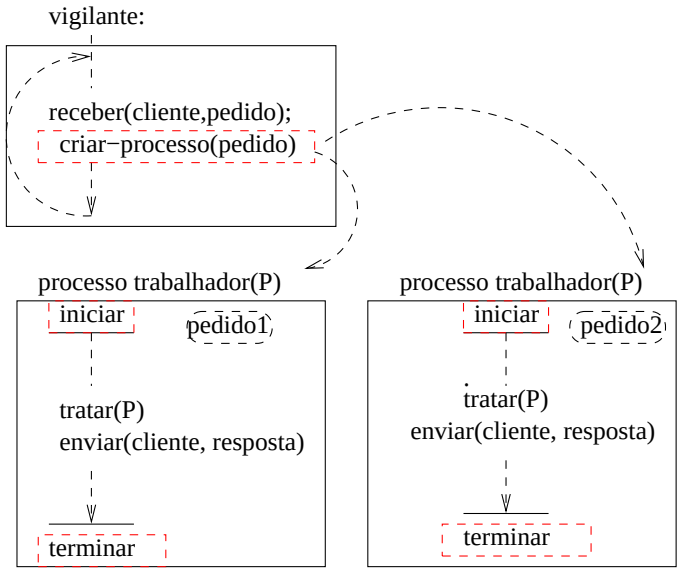

Figura 4: Servidor concorrente - dinâmico.