O Jantar da Tropa

Projecto de Sistemas Operativos – LEEC, LEA

1 Introdução

O "jantar dos filósofos" é um dos exemplos clássicos utilizados para demonstrar as dificuldades da programação concorrente. Mantendo o contexto gastronómico, pretende-se com este trabalho simular a interacção entre diversos intervenientes numa refeição em ambiente militar.

O principal objectivo deste trabalho é familiarizar os alunos com as técnicas de desenvolvimento e teste de aplicações concorrentes em ambiente Unix. A aplicação a desenvolver é constituída por diversos processos, alguns dos quais contendo múltiplas *threads*, que comunicam entre si e se sincronizam através de vários mecanismos disponibilizados pelo sistema operativo.

2 Requisitos de sincronização

Em cada turno são servidos no refeitório soldados, oficiais, e um general responsável. Devem ser respeitados os seguintes requisitos de sincronização:

1. Quando a sala está disponível os soldados entram um a um, escolhem a ementa, e aguardam em sentido a entrada do general.

2. A partir do momento em que o general entra no refeitório, nenhum soldado pode entrar ou sair até este ter terminado o jantar.

3. O general espera que todos os soldados tenham seleccionado a ementa, e dá então ordem para que o jantar comece a ser servido. Os soldados sentam-se, e cada um comerá logo que o seu o jantar (individual) for disponibilizado.

4. Os oficiais também não podem entrar depois do general, mas podem sair em qualquer altura. Admite-se que as suas escolhas são comunicadas antecipadamente, pelo que podem sentar-se e aguardar ser servidos logo que acedem ao refeitório.

5. O general também comunica a refeição antecipadamente. Quando termina levanta-se, e a partir desse momento os soldados podem sair. O general só abandonará a sala quando todos os soldados (mas não necessariamento todos os oficiais) tiverem saído.

6. Os elementos do turno seguinte só podem entrar depois de o general e os soldados do turno actual terem saído. A pertença de um elemento a um turno não é escolhida *a priori*, sendo determinada apenas pelo seu instante de chegada; Se o general ainda não entrou o elemento pode integrar-se no turno actual, senão terá de esperar pelo seguinte.

comando	argumentos	descrição
`m`		ver os menus do turno
`s`	`n`	seleccionar um dos menus (`n`)
`q`		terminar a refeição e sair (logo que possível)

Tabela 1: Comandos aceites pelos processos `soldado`

3 Arquitectura da aplicação

O sistema a desenvolver é constituído pelos processos `refeitorio`, `general`, e múltiplas instâncias de `oficial` e `soldado`.

3.1 Soldados

Após entrar no refeitório um processo `soldado` consulta as opções de menu oferecidas para esse turno. Para tal, acede a uma zona de memória partilhada criada por `refeitorio` onde essa informação está afixada, mostrando-a no terminal se tal for solicitado, e aguarda por uma selecção do utilizador. Escreve-a então, juntamente com um identificador do processo, numa "zona de pedidos" reservada para esse efeito na memória partilhada e prossegue.

A comunicação durante a refeição propriamente dita faz-se através de um *socket stream* dedicado em que `refeitorio` é o servidor. O processo é notificado por essa via quando o seu jantar for servido. Ao terminar a refeição, `soldado` usa o *socket* para anunciar esse facto, o que permitirá o cancelamento da *thread* que lhe está associada em `refeitorio`. A tabela 1 enumera os comandos aceites por um processo `soldado`.

3.2 Oficiais

Os oficiais usufruem de refeições preparadas à medida, cuja composição é especificada em argumentos da linha de comando quando cada processo `oficial` é lançado. Tal como no ficheiro de configuração de `refeitorio` descrito na secção 3.4, cada item é definido por um par `nome quantidade`. Alternativamente, poderá fornecer-se como argumento a `oficial` o nome de um ficheiro de texto contendo a composição da refeição pretendida no mesmo formato usado por `refeitorio`. Não deverão ser impostas restrições ao número, variedade e ordem dos items em causa. A comunicação com `refeitorio` durante o jantar é semelhante à dos processos `soldado`, mas faz-se através de *sockets* datagrama. Essa é também a via usada para transmitir a ementa pretendida a `refeitorio`.

Ao contrário dos soldados, os oficiais podem sair antes de o general ter terminado, presumindo-se que tal ocorre devido a solicitações urgentes de serviço. Podem igualmente permanecer no refeitório mesmo quando o turno seguinte entrar. A saída é desencadeada pelo envio ao processo `oficial` dos *signals* `SIGUSR1` ou `SIGUSR2` e, tal como no caso dos soldados, deve ser comunicada a `refeitorio`.

3.3 General

Como oficial superior, a composição da refeição de `general` é especificada na linha de comando que lança o processo tal como descrito para `oficial`. A comunicação com `refeitorio` é feita através de *mailboxes*. Os comandos aceites pelo processo são apresentados na tabela 2. Mais uma vez, `refeitorio` deve ser informado da saída de `general`. Poderá existir mais de uma instância de `general` no sistema, desde que apenas um desses processos aceda a `refeitorio` em cada turno.

comando	argumentos	descrição
`q`		terminar a refeição e sair (logo que possível)

Tabela 2: Comandos aceites pelo processo `general`

3.4 Refeitório

O processo `refeitorio` gere a distribuição das refeições, comunicando pela via apropriada o jantar[1] a cada um dos comensais logo que este esteja disponível. As refeições individuais são obtidas a partir de um stock de produtos preparados, que podem ser repostos por comandos de abastecimento (*catering*).

O processo `refeitorio` é executado a partir da linha de comandos, tendo como argumento o nome de um ficheiro de texto que descreve o stock de produtos inicial. Cada linha regista os dados de um único produto na forma `nome quantidade`. O ficheiro deve ser mapeado em memória (`mmap`) pelo processo, constituindo um inventário actualizado das existências. Após a terminação de `refeitorio` o ficheiro em disco deve conter o stock final, incluindo produtos temporariamente em falta.

Existindo múltiplas *threads* em `refeitorio`, deverão utilizar-se métodos de sincronização adequados para evitar acessos simultâneos às variáveis de stock.

Elaboração dos menus: Não é necessário respeitar uma estrutura plausível para os menus que são apresentados aos processos `soldado`. Cada um poderá consistir num subconjunto de elementos do stock (com tamanho fixo ou variável), escolhidos aleatoriamente.

Atendimento de pedidos: Ao estabelecer a comunicação com `refeitorio`, cada processo dá-lhe a conhecer a composição da refeição pretendida, quer directamente, no caso de `general` ou `oficial`, quer indirectamente, por escolha de um dos menus disponibilizados nesse turno, no caso de `soldado`. O pedido de um processo é atendido em bloco, ou seja, a refeição é confeccionada e entregue apenas quando todos os ingredientes estiverem disponíveis. Dado que todos os processos comunicam a `refeitorio` a intenção de abandonar a sala, é possível usar essa informação para actualizar a lista de pedidos pendentes.

A satisfação do maior número de pedidos a partir de um stock limitado de produtos é um problema de optimização que ultrapassa o âmbito deste trabalho. Uma estratégia simples, mas ineficiente, consiste em serializar os pedidos a `refeitorio` segundo um critério à escolha, atendendo-os sequencialmente. A rotura no stock de um produto para uma refeição pode por isso paralizar todo o sistema. Alternativamente, pode manter-se uma lista de pedidos pendentes, percorrendo-a sempre que é feito o reabastecimento de algum produto. Esta estratégia, no entanto, pode originar *starvation*.

Actualização do stock: A definição da lista de produtos inicial para `refeitorio` é deixada ao cuidado dos alunos, sendo constituída por diversos tipos de pratos preparados (sopas, pratos de carne, pratos de peixe, doces, fruta, bebidas...). Um pedido personalizado de `general` ou `oficial` poderá referenciar um prato inexistente, que deverá ser adicionado ao stock com quantidade nula.

Comandos: O programa aceita os comandos descritos da tabela 3. Deve salientar-se que a interface com o utilizador é um aspecto *acessório* deste projecto, pelo que uma

[1] Essa comunicação consiste apenas numa descrição da composição do jantar.

comando	argumentos	descrição
`a`	`produto quantidade`	incrementar o stock de `produto` de `quantidade` unidades
`p`	`<v>`	listar pedidos pendentes
`s`	`<v>`	listar stock de produtos
`q`		terminar ordeiramente a aplicação concorrente

Tabela 3: Comandos aceites pelo processo `refeitorio`

formatação básica do texto no terminal é perfeitamente aceitável. A documentação disponibilizada para este trabalho inclui código para implementação de um interpretador de comandos simples. De seguida discutem-se alguns aspectos relativos aos comandos definidos na tabela 3.

`a`: Se `produto` não constar do stock actual de `refeitorio`, deverá ser-lhe adicionado.

`p`: A listagem deverá indicar as entidades que aguardam satisfação de pedidos, respectiva categoria (general, oficial ou soldado) e identificação. Se for especificada a opção `v`, a informação de cada entidade deverá ser complementada com os detalhes do seu pedido, incluindo número de unidades pretendidas e existentes dos vários produtos.

`s`: A listagem básica deverá indicar apenas o nome e quantidade existente de cada produto. Se for especificada a opção `v`, indicar-se-á também para cada produto o número total de unidades já servidas e o número de unidades pendentes.

`q`: Este comando deve causar a terminação *ordeira* de todo os processos que tenham obtido acesso ao turno actual, ou que aguardem por um turno futuro. Para tal, `refeitorio` deve enviar um *signal* `SIGINT` a cada um desses processos, que o interpretarão da mesma forma que um comando de consola `q`. Após esta operação, `refeitorio` poderá também terminar.

4 Funcionalidades Opcionais

Melhoramento do atendimento: Ambas as estratégias para atendimento de pedidos referidas na secção 3.4 têm desvantagens. Sugere-se a concepção e implementação de uma solução que permita adiar o atendimento de um pedido que não possa ser imediatamente satisfeito, mas evitando a ocorrência de *starvation*.

Consola de gestão de processos: A interacção com múltiplas instâncias de `soldado` e `oficial` lançadas manualmente é algo incómoda, porque obriga a uma comutação frequente entre janelas. Sugere-se o desenvolvimento de uma consola, com comandos a definir, que permita a criação de processos `soldado` e `oficial`, visualização do seu *output*, e envio de comandos de forma centralizada. Poderá ser útil prever o lançamento simultâneo de vários processos. O comportamento de `soldado` e `oficial` executados manualmente não deverá ser afectado. A formatação da informação apresentada na consola não é relevante.

5 Estrutura do código

Para facilitar a colaboração entre os elementos de cada grupo e a avaliação do trabalho, o código dos processos deverá ser estruturado em módulos com funcionalidade restrita e bem definida (por exemplo, gestão do stock de produtos). É aconselhável que cada um deles disponha de uma função de inicialização (e, eventualmente, terminação/libertação) das suas variáveis. Por se tratar de um passo conceptual **importante**, esta partição deverá ser feita numa **fase inicial** do trabalho.

Em conformidade com a abordagem modular descrita acima, pretende-se também que o tratamento dos requisitos de sincronização seja efectuado em funções especializadas (por exemplo, do tipo `entrar`, `sair`, `escolher` ou `sentar`), definidas apropriadamente para os vários processos.

6 Faseamento do trabalho

Como em qualquer projecto de software, uma implementação bem sucedida depende de uma correcta planificação do trabalho e de uma estratégia faseada de desenvolvimento e teste. Sugere-se a seguinte sequência de metas:

1. Desenvolvimento de uma solução baseada em semáforos que cumpra os requisitos de sincronização da secção 2.

2. Implementação de um esqueleto de programa com múltiplas *threads* que permita confirmar experimentalmente a validade da solução proposta. As *threads* corresponderão a `general` e algumas instâncias de `oficial` e `soldado`. *Não se pretende* que estas cumpram os requisitos da secção 3, mas apenas que demonstrem os aspectos de sincronização da secção 2. Em particular, não é necessário prever a existência de `refeitorio`, usando-se *dummies* (funções triviais com retorno quase imediato) para todas as operações que devem interagir com essa entidade.

3. Definição detalhada do protocolo de comunicação (i.e., estrutura das mensagens trocadas) entre `refeitorio` e cada um dos processos `soldado`, `oficial` e `general`. Este é um requisito importante para uma colaboração eficiente em projectos de software envolvendo vários programadores.

4. Desenvolvimento de *dummies* para `refeitorio` que aceitem a ligação de *um* processo `soldado`, `oficial` ou `general` através dos mecanismos de comunicação especificados, permitindo ao operador a visualização das mensagens recebidas e o envio de texto.

5. Programação de `general` e `oficial`. Teste de interacção com `refeitorio` (*dummy*).

6. Programação de `soldado`. Teste de interacção com uma versão *dummy* de `refeitorio` que crie uma zona de memória partilhada com menus (fixos) representativos.

7. Desenvolvimento do código para manipulação do stock e leitura/escrita do ficheiro correspondente.

8. Desenvolvimento do código de `refeitorio` para comunicação com os clientes e processamento de pedidos. Teste separado para processos do tipo `soldado`, `oficial` e `general` usando uma única *thread* em `refeitorio`.

9. Criação do código para sincronização e processamento dos pedidos de `soldado` efectuados através da memória partilhada a `refeitorio`.

10. Introdução de *threads* em `refeitorio` para atendimento dedicado aos clientes.
11. Teste de integração.
12. Funcionalidades adicionais.

6.1 Demonstrações intercalares

Nas semanas de 31 de Outubro e 21 de Novembro, no horário de laboratório, deverão realizar-se demonstrações intercalares do trabalho correspondentes às metas cumulativas 5 e 8 referidas na secção 6. A correspondente avaliação qualitativa será tida em conta na atribuição da nota final.

6.2 Entrega e visualização do trabalho

Entrega: O trabalho final deve ser entregue às 10:00 do dia 5 de Dezembro, no laboratório de Sistemas Operativos. O material a entregar consiste em:

- Uma disquete contendo os programas desenvolvidos, incluindo aquele referido no ponto 2 da secção 6, e respectiva *makefile* para compilação.
- Listagens em papel do código em C.
- Um relatório conciso descrevendo:
    1. A solução encontrada para satisfação dos **requisitos de sincronização**.
    2. O **protocolo de comunicação** utilizado entre `refeitorio` e cada tipo de processo.
    3. Os **módulos** desenvolvidos para cada processo e respectiva funcionalidade.
    4. As principais **estruturas de dados** do programa.
    5. Os passos relevantes, em pseudo-código, dos **algoritmos utilizados** (por exemplo, para serialização/atendimento dos pedidos).

Todos estes elementos devem incluir a indicação do turno e grupo de laboratório, e identificação dos alunos.

Visualização: A visualização dos trabalhos com base no código entregue decorrerá na semana de 5 de Dezembro, no horário de laboratório.

Discussão: As discussões dos trabalhos serão realizadas nas semanas de 12 e 19 de Dezembro, em data e hora a afixar na página da disciplina.