MESTRADO INTEGRADO EM ENGENHARIA BIOMÉDICA

Sistemas Operativos

Trabalho prático[1]

Ano lectivo de 2008-2009 Duração: Até às férias de Natal

Gestão de Stocks de Medicamentos

Este trabalho consiste na análise e desenvolvimento de uma aplicação informática para gestão de stocks. Deve admitir-se que se trata de um conjunto muito grande de medicamentos e que há as tradicionais operações de consulta e actualização da quantidade de medicamentos existentes em stock.

A grande dimensão da "base de dados" acarreta desde logo a necessidade de olhar com cuidado para as questões de eficiência e desempenho (espaço ocupado, tempo de execução, espera passiva, etc.). Outro dos requisitos desta aplicação é a capacidade de lidar correctamente com concorrência, uma vez que se prevê que existam vários utilizadores simultâneos. A segurança também não deverá ser esquecida, sendo necessário controlar o acesso aos dados.

Sendo um trabalho de Sistemas Operativos, pretende-se uma solução *de baixo nível*: se se considerar um sistema constituído por várias camadas, onde no topo estão os utilizadores e na base está o hardware, baixo-nível significa que se pretende trabalhar tanto quanto possível próximo da base. Assim, soluções que envolvam a utilização de bases de dados, linguagens de interrogação de alto-nível, linguagens orientadas a objectos, etc., estão excluídas à partida por esconderem grande parte do detalhe que se pretende explorar aqui. De acordo com o objectivo primário da unidade curricular de Sistemas Operativos, o de ajudar a perceber como funcionam os sistemas informáticos, vamos "abrir o capôt do automóvel" e tentar perceber a mecânica do software de sistemas. Podemos contar apenas com a cultura dos Sistemas Operativos clássicos — linguagem C, *system calls* do Unix (Linux, Mac OSX...), ficheiros, interacção entre processos, controle de concorrência, etc. Assim, da próxima vez que alguém disser que o sistema está lento, que está a gastar muito espaço ou que há problemas de privacidade de informação, ter-se-á uma ideia muito mais precisa do que se passa lá dentro e da forma de ultrapassar esses problemas.

O enunciado é razoavelmente vago para fomentar alguma criatividade nos grupos de trabalho e permitir que estes se concentrem nas áreas que acham mais interessantes (por oposição à típica situação de *trabalho prático* = *xarope* que se começa a tomar uns dias antes de acabar o prazo de entrega). Pela mesma razão, outro dos aspectos pouco habituais do trabalho é a forma incremental como deverá ser realizado: ao longo de várias semanas, são sugeridos durante as aulas pequenos "trabalhos de casa" que não são mais do que componentes do trabalho final. Assim, no final das aulas o trabalho deve estar praticamente concluído sendo apenas necessário escrever o relatório final.

[1] Cotação — 40% nota final

Conforme anteriormente referido, pretende-se desenvolver a capacidade de analisar e especificar problemas, e partir daí para a sua solução por via informática, e ainda a capacidade de lidar com algoritmos concorrentes. Para atingir estes objectivos deverá começar-se por especificar o problema, definindo os tipos de dados (i.e., responder à pergunta "o que é um medicamento" em termos informáticos?), as operações sobre eles e a forma como os dados são armazenados de forma persistente (ou seja, responder a perguntas como "o que é uma *tabela*? Está indexada? Ordenada?").

Ainda dentro espírito do baixo-nível e de fazer as coisas *à moda antiga* para ver as vantagens e desvantagens que a subida no nível de abstracção pode trazer, vamos recuar algumas décadas e perceber como seriam os sistemas de gestão de stocks dessa altura. Não havia computadores portáteis nem internet, tipicamente existiria apenas um grande sistema em regime de *time-sharing*: um computador central ao qual estariam ligadas dezenas ou centenas de terminais *dumb*[2]. É esse o cenário deste trabalho, se bem que ele seja agora simulado em portáteis com internet. Vai precisar de vários portáteis, um deles a fazer de computador central e a aceitar ligações telnet/ssh de outros computadores, que se limitam a simular os antigos terminais VT-100 ou similar[3].

No sistema central, a aplicação a desenvolver será constituída por um programa escrito em C e activado a partir da *shell*: o utilizador faz login (remoto) e a aplicação arranca automaticamente. Isto significa que tem de haver alguma configuração (lembra-se do início das aulas, scripts da bash? É altura de refrescar esses conhecimentos...). A interface com o utilizador deverá ser tão simples quanto possível: lembre-se que apenas dispõe de uma janela com um terminal remoto, não tem interface gráfica. Idealmente esta camada de diálogo com o utilizador deveria ser separada da camada de "negócio" para permitir, por exemplo, que o programa cliente semelhante aos que estará a desenvolver em java na UCE de Sistemas Distribuídos possa ser adaptado para cliente do gestor de stocks. Ou seja, a interface e as escolhas feitas nessa camada não devem influenciar as decisões e tecnologias usadas no resto da aplicação. De igual forma, a camada dos dados deveria ser isolada para permitir, por exemplo, a sus substituição por uma genuína base de dados com acesso por SQL.

Alguns dos trabalhos de casa úteis para a realização deste trabalho incluem:

- Ordenação de vectores
- Representação interna de *tabelas*
- Criação e comunicação de processos
- Leitura e escrita de ficheiros e pipes
- Sincronização de processos
- Partilha de memória, semáforos

O trabalho será portanto um percurso e não apenas um programa final: consiste numa série de programas comprovativos do domínio de técnicas de programação sequencial e concorrente, bem com de diferentes padrões de soluções concorrentes. Em relação a estas, deve começar-se pela aplicação sequencial com um só utilizador, passando então para a execução concorrente dessa mesma aplicação. A execução concorrente provavelmente conduzirá a resultados errados, ter-se-ão de identificar as causas e aplicar

[2]Isto significa que a camada da interface com o utilizador será muito pouco apelativa quando comparada com as actuais interfaces gráficas; terá no entanto a grande vantagem de ser "leve" e funcionar com a largura de banda de antigamente.

[3]Poderá ainda criar máquinas virtuais no seu PC, uma para o servidor e pelo menos outra para um terminal "remoto", mas perde o não-determinismo de pedidos simultâneos emitidos por diferentes utilizadores.

então as respectivas "receitas" de controle de concorrência (e.g. locks, semáforos). É nesta altura que, ainda dentro do modelo *self-scheduling*, se poderá substituir a leitura/escrita de ficheiros de dados pelo mapeamento da base de dados em memória.

Uma vez percebidos os perigos da partilha simultânea de recursos, partir-se-á então para uma arquitectura alternativa, já muito próxima do modelo cliente-servidor, e onde os clientes se limitam a solicitar operações a um gestor da base de dados. Nesta altura já deverá ter bem definida, codificadas e testadas as camadas da interface com o utilizador (que não se modifica seja *self-scheduling* seja *cliente/servidor*), e também a camada de acesso aos dados/ficheiros. Apenas se altera a entidade que tem acesso físico aos dados — deixa de ser a aplicação como um todo, com os privilégios do utilizador que a invocou, para ser apenas o ou os processos servidores. Ter um ou mais processos servidores será sobretudo uma decisão que tem a ver com desempenho (performance), ou seja, um pormenor de configuração.

Por analogia com as *milestones* de um projecto de software podem identificar-se as etapas seguintes:

1. Programação sequencial:
   (a) ordenação de vectores de inteiros
   (b) "tabelas" da base de dados de medicamentos, *structs*, ordenação de tabelas
   (c) dimensão fixa versus dimensão variável? inserção/remoção? arrays versus listas...
2. Programação concorrente:
   (a) sort/merge com um só processo
   (b) sort/merge concorrente (pai+filho), com diversos mecanismos de sincronização (WEXITSTATUS, kill/signal manual, eventualmente uso de semáforos (ver `fsmipc.c`))
   (c) sort/merge com named e unnamed pipes; comparação com as versões anteriores com sincronização explícita
3. Gestão de medicamentos *self-scheduling*:
   (a) partilha dos ficheiros de dados
   (b) mapeamento do(s) ficheiro(s) para memória virtual
4. Gestão de medicamentos *cliente/servidor*:
   (a) comunicação através de pipelines
   (b) comunicação através de mensagens (`msgops`, ver `fsmipc.c`)
   (c) comunicação através de sockets

Como será evidente, este conjunto de soluções presta-se a uma grande dose de *reutilização* de código, cujo desenvolvimento e teste poderá ser repartido pelos diversos elementos dos grupos de trabalho. Isso obriga à especificação prévia das APIs para um um elemento possa estar a trabalhar, por exemplo, na garantia de exclusão mútua no acesso a ficheiros de inteiros, enquanto outro lida com a manipulação sequencial de tabelas de medicamentos. Uma vez testada cada parte, deverá ser fácil combinar as duas para obter um programa (operacional!) de acesso concorrente à base de dados de medicamentos. Pretende-se desta forma cobrir também algumas das boas práticas de projecto de software desenvolvido em equipa e estimular o trabalho em grupo e a integração dos seus elementos, ao invés da mera divisão antecipada do trabalho pelos elementos do grupo, onde cada elemento fica apenas especialista do seu fragmento.