Universidade do Minho

Mestrado Integrado em Engenharia Biomédica

Plataformas de Software

Trabalho realizado por:

Fábio França 61785

Filipe Fernandes 61754

Docente:

Professor Francisco Coelho Soares Moura

Novembro de 2013

Resumo

Este trabalho encontra-se englobado no âmbito da Unidade Curricular Plataformas de Software. Assim sendo pretende-se desenvolver uma aplicação para gestão concorrente de stock de medicamentos. Esta, deverá ser desenvolvida em linguagem C e executável em ambiente Unix, desenvolvendo-se uma solução de baixo nível.
Além de operações de consulta e manipulação de stock, procura-se aprofundar questões de desempenho e eficiência de aplicação numa perspectiva de realização de escalabilidade da mesma.

Conteúdo

1 Introdução

O trabalho prático em questão consiste na análise e desenvolvimento de uma aplicação informática para a gestão do stock de medicamentos. Assim sendo o principal objectivo enquadra-se na utilização de primitivas de programação concorrente em ambiente Unix/Linux, system calls, linguagem C, de forma a efectuar não só as tradicionais operações de consulta e actualização do stock de medicamentos, bem como a percepção de desempenho e eficiência da aplicacão.
Para o desenvolvimento do mesmo deve admitir-se que a base de dados de medicamentos é de tal forma grande que não é possível coloca-la em memória central, ou seja esta tem de ser armazenada em disco. Avaliar o desempenho da aplicação realizada é assim um dos grandes objectivos do trabalho, pelo que foi necessária a inserção de diferentes montantes de medicamentos na base de dados medindo o tempo de execução de cada operação, se possível com processos concorrentes,de modo a garantir a fiabilidade e a persistência dos dados.
Deste modo e numa perspetiva de teste foram desenvolvidas duas versões da aplicação. A primeira designada por "Vanilla File IO"no qual não são desenvolvidas especificações de optimização da mesma. Na segunda versão, implementou-se *HashTables* como solução de optimização.
Importa ainda realçar que o conjunto de funções implementadas têm em vista uma solução de *baixo nível* na ordem de temática de Sistemas Operativos.

2 Preliminares

2.1 Linguagem de Programação C

Na realização do trabalho prático em causa, foi desenvolvida uma aplicação em linguagem de programação C. De seguida pretende-se distinguir, genericamente, as linguagens de programação em dois grandes grupos: linguagens de baixo nível e alto nível.

- Linguagens de baixo nível: São linguagens voltadas para a máquina, isto é, são escritas usando as instruções do microprocessador do computador. São genericamente chamadas de linguagens Assembly.
  - *Vantagens*: Programas são executados com maior velocidade de processamento. Os programas ocupam menos espaço na memória.
  - *Desvantagens:* Em geral, programas em Assembly tem pouca portabilidade, isto é, um código gerado para um tipo de processador não serve para outro. Códigos Assembly não são estruturados, tornando a programação mais difícil.
- Linguagens de alto nível: São linguagens voltadas para o ser humano. Em geral utilizam sintaxe estruturada tornando seu código mais legível. Necessitam de compiladores ou interpretadores para gerar instruções do microprocessador.Compiladores fazem a tradução de todas as instruções do programa fonte gerando um programa executável
  - *Vantagens:* Por serem compiladas ou interpretadas, tem maior portabilidade podendo ser executados em varias plataformas com pouquíssimas modificações. Em geral, a programação torna-se mais fácil por causa do maior ou menor grau de estruturação de suas linguagens.
  - Desvantagens: Em geral, as rotinas geradas (em linguagem de maquina) são mais genéricas e portanto mais complexas ocupando por conseguinte mais memória e tornando-se mais lenta.

A linguagem de programação C é uma linguagem de alto nível genérica, desenvolvida por programadores para programadores apresentando características de flexibilidade e portabilidade. Esta é uma linguagem simples acente num paradigma algorítmico e procedimental de acesso direto à memória. Desta forma consegue-se "trabalhar próximo da base".
Entre as principais características do C pode-se citar:

- O C é uma linguagem de alto nível com uma sintaxe bastante estruturada e flexível tornando sua programação bastante simplificada.
- Programas em C são compilados, gerando programas executáveis.

- O C compartilha recursos tanto de alto quanto de baixo nível, pois permite acesso e programação direta do microprocessador. Com isto, rotinas cuja dependência do tempo é crítica, podem ser facilmente implementadas usando instruções em Assembly. Por esta razão o C é a linguagem preferida dos programadores de aplicações.

- Embora estruturalmente simples (poucas funções intrínsecas) o C não perde funcionalidade pois permite a inclusão de uma farta quantidade de rotinas do usuário. Os fabricantes de compiladores fornecem uma ampla variedade de rotinas pré-compiladas em bibliotecas.

- O compilador C gera códigos mais "limpos"e velozes do que muitas outras linguagens.

2.2 Base de Dados em Memória

Uma *In-memory database*(IMDB ou *Main Memory Database System* é um sistema de gerenciamento de base de dados acente na memória principal para armazenamento de dados computacionais. Contrariamente com os sistemas de gestão de base de dados que empregam um mecanismo de armazenamento em disco, uma IMDB é mais "rápida", uma vez que a optimização interna dos algoritmos é simples e executa poucas instruções de CPU. O acesso a dados em memória elimina o *seek time* na consulta de dados, o que proporciona um desempenho mais rápido e previsível do que em disco. Em aplicações cuja responta em tempo é crítica, tal como redes de telecomunicações, a utilização de IMDB é eficazmente implementada.

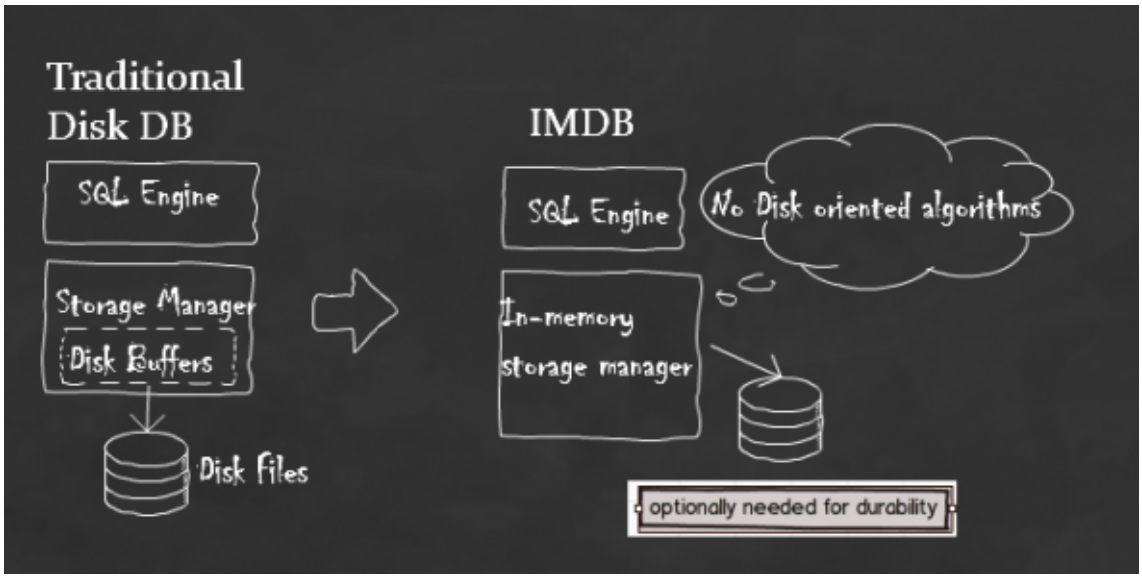

Figura 1: Bases de dados em Disco VS Memória

2.3 Hashtables

Como anteriormente referido desenvolveram-se duas versões da aplicação. Além da versão Vanilla, cada grupo ficou incunbido de implementar uma das seguintes soluções:

1. Pesquisa binária em ficheiros ordenados
2. Ficheiros indexados
3. Hash Table
4. Memory mapped files
5. Cache em memória central dos medicamentos mais usados

No caso específico deste grupo foram utilizadas Hash tables. Uma Hash table ou tabela de espalhamento é uma estrutura de dados especial, que associa chaves de pesquisa a valores. O seu objetivo primário é realizar uma pesquisa rápida através de uma chave simples por forma a obter o valor desejado. as tabelas de hash são fundamentalmente utilizadas para indexação de grandes volumes de informação. A implementação típica procura uma função de dispersão que seja de complexidade O(1), não importando o número de registos na tabela, desconsiderando as colisões. Na prática as funções de hash perfeitas ou quase perfeitas são encontradas apenas onde a colisão é intolerável(por exemplo em funções criptográficas), ou quando o conteúdo da tabela é previamente conhecido. Por causa das colisões muitas tabelas são implementadas conjuntamente com alguma estrutura de dados tais como, uma lista ligada(estrutura de dados linear e dinâmica, composta por células que apontam para o próximo elemento da lista) , ou árvore de busca binária auto-balanceada. Noutras situações a colisão é solucionada dentro da própria tabela.

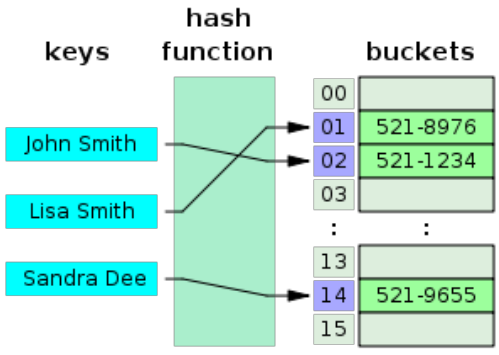

Figura 2: Hash Table

2.3.1 Colisões/Problemáticas

A função de hash pode calcular o mesmo índice para duas chaves diferentes, situação designada por colisão. Nesse sentido, deverá ser projetada para evitar o máximo de ocorrência de colisões. Por mais que a função seja projetada, invariavelmente ocorrerão colisões. Os mecanismos mais comuns para o tratamento de colisões são o endereçamento aberto e estruturas ligadas.

Endereçamento Aberto A informação é armazenada na própria tabela de dispersão. Possui três estratégias: tentativa linear (incremental), tentativa quadrática e dispersão dupla.

- Para a estratégia linear, é utilizada uma segunda função matemática para calcular a posição em que deve ser feita a próxima prova, a função de redispersão. Desta forma elimina-se o agrupamento primário porém gera-se outro fenómeno conhecido como agrupamento secundário. Ocorre que as chaves de valores diferentes e mesmo valor de dispersão possuem a mesma sequência de provas.

- Para a estratégia quadrática, por forma a reduzir o agrupamento primário, procura-se por um lugar livre através da fórmula: h+n2, em que h representa o índice da colisão e n o sequencial de busca pela nova posição.

- No método de dispersão dupla, para evitar a concentração de posições ocupadas na tabela, esta estratégia produz uma variação na forma de procura de posição livre a fim de armazenar o elemento que colidiu. Assim utiliza-se uma segunda função de dispersão h'.

Encadeamento A informação é armazenada em estruturas encadeadas fora da tabela de dispersão. Neste sentidoencontra-se uma posição disponível tabela, indicando de seguida que essa posição deve ser procurada a seguir. Os métodos mais conhecidos são:

- O encadeamento separado é a solução mais simples, em que normalmente um registo aponta para uma lista ligada em que são armazenados os registos em conflito. A inserção na tabela requer uma busca e inserção dentro da lista ligada;a remoção requer atualizar os índices dentro da lista, como se faria normalmente.

- Já no método de encadeamento aberto os registos em conflito são armazenados dentro da própria tabela. A resolução das colisões é realizada através de procuras padronizadas dentro da própria tabela. A forma mais simples de fazer a pesquisa é procurar linearmente na tabela até encontrar um registo vazio ou o registo procurado.

2.4 Programação Concorrente

Uma das problemáticas inseridas na realização do trabalho prático é o controlo de concorrência. Se bem utilizado, o paralelismo resulta num melhor desempenho de programas. No entanto o acesso concorrente a dados e recursos pode gerar problemas, uma vez que estes dados se podem tornar inconsistentes ao serem acessados concorrentemente.

Assim, para garantir a coerência dos dados é necessário que os processos cooperem e acedam ordenadamente aos recursos partilhados. Cabe ao Sistema Operativo fornecer os mecanismos necessários para que os processos se sincronizem e controlem a ordem de acesso aos recursos partilhados. Alguns mecanismos de Controle de Concorrência disponíveis são:

- Monitor - protege trechos de código/métodos que manipulam dados/recursos compartilhados, impedindo o acesso concorrente;
- Lock (ou Mutex) - cria uma fila de acesso a um dado compartilhado, impedindo o acesso concorrente;
- Semáforo - limita o número de utilizadores que acessam simultaneamente um recurso, criando filas de acesso se este número for excedido.

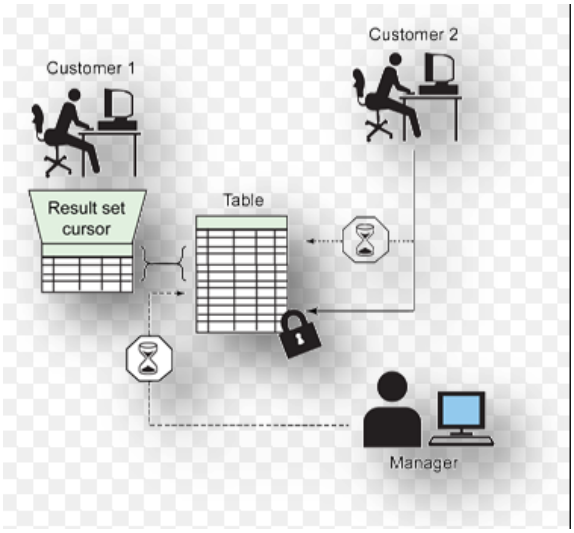

Figura 3: Controlo de Concorrência

Concretamente, cada processo tem definidas as regiões do seu código que acedem a recursos partilhados, isto é quais as suas regiões/secções críticas. Uma região crítica protege uma estrutura de dados partilhada, garantindo a exclusão mútua no acesso a esses dados (mesmo perante múltiplos processadores), isto é, assegura-se que não estão dois processos a executar as suas regiões críticas (relativas a um recurso partilhado X) ao mesmo tempo. Assim, cada processo deve pedir autorização para entrar na sua região crítica. Por outro lado, é também

necessário garantir que qualquer processo que o pretenda aceder deverá inevitavelmente poder executar a sua região crítica (espera limitada). A implementação eficiente de regiões críticas e´ conseguida utilizando primitivas de comunicação entre processos. O IPC (Interprocess communication) é um mecanismo através do qual dois ou mais processos comunicam entre si para executar certas tarefas. A comunicaça˜o entre processos é suportada por todos os sistemas UNIX. Contudo, diferentes sistemas UNIX implementam diferentes me´todos de IPC. Para processos que executam na mesma ma´quina, o UNIX suporta: Mensagens, Sema´foros e Memo´ria partilhada. E para processos que correm em ma´quinas diferentes suporta: Sockets, TLI – Transport Layer Interface e XTI – X/Open Transporte Interface (baseado em TLI). Assim sendo os semáforos fornecem um conjunto de variáveis disponíveis em todo o sistema que podem ser modificadas e usadas por processos a correr na mesma ma´quina. A memória partilhada permite que vários processos a correr na mesma máquina partilhem uma região de memória comum, de tal modo que os dados escritos nessa região podem ser lidos/modificados por esses processos.

3 Descrição do Trabalho e Análise de Resultados

Em função dos pressupostos analisados nas seções anteriores e tendo em conta os objetivos do trabalho prático, procurou deliniar-se as directrizes para o mesmo. Inicialmente foi necessário trabalho de pesquisa por parte dos elementos do grupo, de forma a familiarizar-se com as problemáticas a enfrentar e aprofundar conhecimentos da Linguagem de programação em C. Após esta fase inicial, foi necessário decidir o tipo de armazenamento da base de dados, no qual se optou por um ficheiro binário. Este permite um fácil acesso à base de dados bem como a sua manipulação.
De seguida, iniciou-se a elaboração de vários programas em C por forma a simular as diversas acções realizadas na base de dados. Tendo em conta que todos os utilizadores podem realizar as mesmas operações, parte-se do princípio que todos os processos utilzam o mesmo código. Assim sendo, e como todos os utilizadores possuem acesso à mesma base de dados, este trata-se de um recurso partilhado por todos os processos.
Para solucionar os problemas de concorrência utilizou-se mecanismos de sincronização de forma a que quando determinado utilizador estiver a "escrever"no ficheiro, nenhum outro utilizador terá acesso ao mesmo, uma vez que este se trata de um recurso partilhado.
Como necessário foram desenvolvidos programas em C para as duas versões. Incialmente desenvolveu-se a versão Vanilla e posteriormente elaborou-se a versão final e optimizada contendo estruturação em Hashtables. Após a elaboração de ambas as versões foi necessário comparar o desempenho e eficiência de ambas. Tal como esperado, a segunda versão apresenta valores optimizados garantindo uma solução escalável para o stock de 20 milhões de medicamentos. No seguimento deste relatório será apresentado uma comparação entre ambas as versões.
Para a versão da informação armazenada na HashTable, foi decidido usar a *Uthash* como base do nosso trabalho . Esta decsão foi tomada após alguma pesquisa sobre bibliotecas relacionadas com o uso das HashTables. No entanto, a *Uthash* não é uma biblioteca , mas sim um ficheiro *header* o qual é apenas necessário adicionar na diretoria do projecto e fazer a inclusão deste no código C.
A razão que nos levou a escolha da *Uthash* foi o facto de esta estar otimizada para a manipulação de um grande numero de estruturas. Para além disso, traz consigo um conjunto de funções que no contexto do nosso trabalho se revelaram bastante fáceis de entender e usar. Aliando o seu desempenho com a facilidade de manipulação decidimos que que esta HashTable teria condições mais do que suficientes para satisfazer os requisitos deste trabalho.
Para lidar com as colisões esta utiliza o método de encadeamento, ou seja , na adição de um medicamento , o programa realiza a função de Hash na sua chave (nome) . Isto é feito para atribuir um "espaço"na HashTable para este registo, o qual é capaz de receber mais de que um medicamento caso a função de Hash

atribua dois valores iguais. Ao pesquisar este medicamento a *Uthash* compara as chaves para assegurar que o medicamento desejado é retornado.
Foi tido em consideração a fiabilidade da *Uthash*, e após alguma pesquisa pôde-se determinar que esta já foi incorporada tanto em usos comerciais como académicos pelo mundo.
O programa desenvolvido consiste em três fases importantes :

- A leitura - No momento em que ligamos a aplicação esta traz para a memória toda a informção armazenada no ficheiro binário na forma de HashTable;
- Manipulação dos dados - Após a leitura destes pode-se manipulá-los com bastante facilidade e rapidez , através de uma interface simples de se utilizar;
- Persistência de Dados - Depois de de todas as alterações feitas no programa , no momento de saída podemos optar por gravar os dados de modo que a próxima vez que se aceder à aplicação esta apresente uma base de dados atualizada.

4 Resultados

Depois da realização das duas versões , Vanilla e HashTables, pode-se comparar o desempenho médio de ambas, em diversos contextos. Os resultados são apresentados na tabela a seguir apresentada:

Função	Versão Vanilla / Tempo(s)	Versão HashTable / Tempo(s)
Ler	-	15.14911
Inserir	0.005	0.0033
Procurar	8.502	0.000004
Atualizar	1.2451	0.000013
Remover	53.887	0.000028
Ordenação(50)	-	7.895923
Ordenação(50000)	-	7.970544
Ordenação(5000000)	-	11.631502
Ordenação(19000000)	-	19.972137

Figura 4: Comparação de Desempenhos

Constata-se que a HashTable apresenta um desempenho superior na maior parte das operações. A razão pela qual o Vanilla não contém um tempo associado à leitura dos dados deve-se ao facto de esta não necessitar de o fazer. Relativamente às operações de inserção, procura , actualização e remoção de um registo a HashTable teve um desempenho muito superior à versão Vanilla. No que consta à ordenação da informação o Vanilla apresentou tempos demasiado elevados para serem tomados em conta enquanto que a HashTable apresentou desempenhos bastante razoáveis. Nesta última categoria é necessário ter em conta que a maior parcela do tempo não é gasta na ordenação da HashTable mas sim na impressão do resultado em ficheiro.

5 Conclusões

A elaboração do trabalho prático em questão permitiu aprofundar conhecimentos na utilização da linguagem de programação C e adjacente desenvolvimento de mecanismos de raciocínio para resolução de problemas, assim como a resolução de problemas de sincronização e concorrência.
Tendo em conta a abordagem desenvolvida pode-se considerar favorável a implementação de ambas as versões. É de notar a optmização observada com a introdução de hashtables. Este tipo de estrutura apresenta como principal vantagem de utilização o seu desempenho. Enquanto a procura linear apresenta complexidade temporal O(N) e a procura binária complexidade O(log N), o tempo de procura na tabela de Hash é praticamente independente do número de chaves armazenadas na tabela, ou seja, tem complexidade temporal O(1). A desvantagem do uso da HashTable é o facto de esta requerer a leitura e persistência de dados num ficheiro que leva a alguns segundos a realizar.
Pode-se assim concluir, pela comparação de desempenhos, que a tabela hash tem tamanho adequado ao número de chaves que irá armazenar, a função hash utilizada apresenta boa qualidade e a estratégia de manipulação por hashing é bastante eficiente.

6 Bibliografia

1. https://www.sqlite.org/inmemorydb.html
2. https://code.google.com/p/leveldb/
3. Ferreira, L.L.; Batista, B.; Viamonte, M.J.; Malheiro, N.; Programaça˜o Concorrente em Linux: Memória Partilhada e Semáforos; Instituto Superior de Engenharia do Porto
4. Sistemas Operativos, Alves Marques et al, FCA Editora Informática, 2009
5. Litwin, Witold (1980). "Linear hashing: A new tool for file and table addressing". Proc. 6th Conference on Very Large Databases. pp. 212–223.

7 Apendices