Universidade Do Minho

Mestrado Integrado Em Engenharia Biomédica

Plataformas de Software

TRABALHO PRÁTICO

Sistema de Gestão de Stock de Medicamentos

Braga, Janeiro 2013

Universidade Do Minho

Mestrado Integrado Em Engenharia Biomédica

Plataformas de Software

TRABALHO PRÁTICO

Sistema de Gestão de Stock de Medicamentos

Docente: Francisco Soares Moura

Grupo 4: Hugo Gomes (58536)
Marta Moreira (54459)
Telma Veloso (58521)

Resumo

Pretendia-se desenvolver uma solução que representasse eficazmente um sistema de gestão de *stocks* de medicamentos, recorrendo a princípios de programação concorrente, segurança de dados e desempenho, e utilizando a linguagem de programação `C` e o ambiente `Unix`.

A construção do sistema envolveu um conjunto de decisões que se crê terem contribuído para uma optimização da eficiência do mesmo, sobretudo no que diz respeito à abordagem escolhida para o armazenamento dos dados: em vez de ter apenas um ficheiro para armazenar todas as instâncias da *struct* `medicamento`, optou-se por considerar vários ficheiros distintos, o que acabou por diminuir consideravelmente o tempo de espera entre *locks* e a fracção de medicamentos bloqueados. Pese embora se reconheçam soluções alternativas potencialmente mais eficientes, crê-se que o sistema desenvolvido se apresenta, acima de tudo, equilibrado, no que diz respeito à quantidade e tamanho dos ficheiros envolvidos

Conteúdo

1 Objectivos

O presente relatório surge no âmbito da unidade curricular de Plataformas de Software e pretende descrever a abordagem considerada no contexto do trabalho prático desenvolvido, através do qual se procurou desenvolver um sistema de gestão de *stocks* de medicamentos. Os objectivos que presidiram à elaboração do trabalho passaram por:

1. Explorar a linguagem `C` e o ambiente `Unix` no sentido de desenvolver um sistema de gestão de *stocks* de medicamentos;
2. Delinear e construir um conjunto de métodos para a comunicação entre processos;
3. Associar os métodos anteriormente referidos a opções ao serviço do utilizador, por intermédio de uma interface gráfica no terminal, que veiculassem a possibilidade de manipular o sistema;
4. Aplicar princípios da programação concorrente, segurança de dados e desempenho ao pressuposto da existência de vários utilizadores em simultâneo.

2 Estruturação dos Dados

Numa primeira fase, construiu-se uma *struct* denominada de `Medicamento`, na qual se definem as variáveis relativas ao nome do medicamento (`nome`) e à respectiva quantidade de unidades presentes em *stock* (`stock`). A utilização deste tipo de estruturas garante que se estabelece uma relação efectiva entre ambas as variáveis, isto é, que existe uma correspondência entre o nome de um dado medicamento e o respectivo *stock* que lhe está associado.

```
#define TAMANHO 28
...
struct Med{
    char nome [TAMANHO];
    int stock;
};
```

Dado que a linguagem de programação `C` é de baixo nível, o conceito de *string* é necessariamente abordado de forma abstracta, correspondendo a um *array* de caracteres, de tamanho `TAMANHO`. Interessava, neste ponto, garantir que cada estrutura pudesse ser lida recorrendo a uma única invocação de leitura ao disco, de forma a maximizar a eficiência do programa, pelo que, partindo do conhecimento de que a leitura de um bloco em disco corresponde a 64 *bytes* e que um inteiro ocupa 4 *bytes*, inferiu-se que o `TAMANHO` deveria ser igual a 28, de forma a perfazer 32 *bytes* [1]. Pese embora se pudesse definir para o `nome` um `TAMANHO` de 60 *bytes*, considerou-se que 28 caracteres seriam suficientes para descrever o nome de um medicamento.

3 Armazenamento dos Dados

O armazenamento das *structs* foi realizado através da escrita das mesmas em ficheiro recorrendo à função `fwrite`. Optou-se, neste ponto, por criar vários ficheiros separados, ao invés de um único ficheiro contendo todos os medicamentos, no sentido de, quando se efectuasse uma operação sobre um medicamento, impedir que todos os ficheiros ficassem bloqueados. De acordo com as duas primeiras letras do nome do medicamento, este é gravado no ficheiro correspondente. Por exemplo, um medicamento cujo nome seja aspirina será gravado no ficheiro `medicamento_as.dat`. Na totalidade, existem 677 ficheiros que abarcam a referida combinação de aa até zz e ainda um ficheiro `defaultFile` para nomes inferiores a dois caracteres ou começados por caracteres que não letras do alfabeto.

Os medicamentos vendidos, por sua vez, são armazenados num único ficheiro.

4 Interface com o Utilizador

4.1 Login

De forma a imputar ao programa um determinado nível de segurança, ainda que básico, criou-se um sistema de *login* que desencadeia comportamentos distintos consoante as acções a operar sobre o sistema. Neste sentido, definiram-se duas *passwords*: `passUser` e `passAdmin`; uma variável `login` — inicializada a zero — e uma outra variável `read` que assume uma sequência de, no máximo, 10 caracteres que será introduzida pelo utilizador. O raciocínio por detrás do método `login()` assenta sobre as seguintes premissas:

1. Enquanto o valor de `login` for igual a zero, é impressa na consola uma mensagem que informa o utilizador acerca do ficheiro `ReadMe` que deve consultar *à priori* sobre a forma como deve proceder, indicando que, no caso de este pretender criar os ficheiros, a *password* a inserir deverá ser a de administrador; em caso contrário, isto é, se os ficheiros já existirem, este deve fazer *login* com a *password* de utilizador.

2. Após a inserção da *password*, caso esta unifique com a `passAdmin`, é desencadeado o método `genmeds()` e, na consola, é impressa a mensagem `'Login successful! Os ficheiros foram criados.'` e o utilizador tem acesso ao menu da aplicação. Se, por outro lado, a *password* coincidir com a `passUser`, para além do *output* da mensagem de *successful login*, a função `menu()` é chamada a executar e o utilizador adquire acesso ao menu principal do programa. No caso específico da *password* introduzida não corresponder a nenhum dos dois conjuntos de caracteres especificados, o utilizador é alertado de que esta não é válida e incitado a tentar novamente.

4.2 Menu

A parte interactiva do programa foi organizada num menu que, para além do cabeçalho, possui uma lista de todas as acções que o utilizador pode efectuar. Cada acção está associada a um número e, através da implementação de um *switch*, é possível desencadear a acção pretendida inserindo o número correspondente. Definiram-se, neste contexto específico, os seguintes métodos:

1. `insere_med()`;
2. `remove_med()`;
3. `actualiza_med()`;
4. `listar_med()`;
5. `consultaStock_med()`;
6. `listaVendas()`;
7. `listaTopVendas()`;

4.2.1 Inserção de Medicamentos

A função `insere_med()` gera, inicialmente, um pedido ao utilizador para que este insira o nome do medicamento a adicionar à base e o respectivo valor de *stock*, percorrendo, posteriormente, a totalidade do ficheiro e lendo, um a um e para a variável `infile`, as instâncias de medicamento nele presentes. Caso encontre algum medicamento com nome igual ao que se pretende adicionar, então a *flag*`(variável de controlo)` — que permite concluir se uma acção foi ou não executada — passa a `SIM` e o utilizador é avisado de que o medicamento já existe. Ao mesmo tempo que se procura o medicamento possivelmente repetido, também se procura a primeira posição cujo nome tem o valor `apagado`, evitando assim percorrer o ficheiro duas vezes. Porém, se não for encontrado nenhum medicamento com o mesmo nome, existem duas possibilidades para a concretização da inserção do medicamento: numa posição que foi apagada — reaproveitando-se, assim, os espaços; ou no fim do documento. Se a *flag* `apagado` não é nula, então significa que existe uma posição que pode ser substituída pelo medicamento a inserir. Recorrendo à função `fseek`, o cursor é colocado no local pretendido do ficheiro, isto é, na posição onde começa o `apagado` e o nome do

medicamento é escrito recorrendo à função `fwrite`, que reescreve o que ali se encontrava. Na possibilidade de não existir nenhum `apagado`, o cursor já se encontra no final do ficheiro (já que este foi percorrido na sua totalidade pelo `while`), bastando apenas invocar a função `fwrite` e poupando-se, desta forma, a chamada à função `fseek`, que colocaria o cursor no local pretendido.

É boa prática bloquear o acesso ao ficheiro enquanto se efectuam alterações de escrita, como é o caso, pelo que se recorre à função `flock` com o modo `LOCK_EX` (exclusivo), no sentido de garantir que mais ninguém obtém acesso ao mesmo, evitando-se, assim, a corrupção de dados. No final da execução do método, o ficheiro é libertado, através da função `flock` no modo `LOCK_UN`. Relativamente a outros pormenores, utiliza-se, aquando da abertura do ficheiro, o atributo `r+` pois este permite tanto a leitura como a escrita, e verifica-se se ocorreu algum erro no decorrer deste processo.

4.2.2 Remoção de Medicamentos

No que diz respeito à função `remove_med()`, é pedido ao utilizador que insera o nome do medicamento que deseja apagar e, de seguida, é aberto o ficheiro correspondente. Mais uma vez, percorre-se o ficheiro, lendo um a um para a variável `infile`, sendo que, quando o medicamento é encontrado — com o nome semelhante àquele introduzido pelo utilizador — utiliza-se a função `strcpy` para atribuir `apagado` ao nome desse medicamento, enquanto que ao *stock* é lhe atribuido o valor de 0 unidades. A *flag* passa a `SIM` e emprega-se a função `fseek` para posicionar o cursor no local correcto, procedendo-se à (re)escrita do medicamento. Este raciocínio evita a fragmentação do ficheiro e, conforme explicitado anteriormente, os espaços assim gerados podem ser reaproveitados. Se o medicamento não for encontrado — o que se infere pelo facto de a *flag* permanecer nula —, o utilizador é notificado com uma mensagem. Mais uma vez, liberta-se o *lock*, fechando-se depois o ficheiro com a função `fclose`.

4.2.3 Actualização do Stock

Considerando a actualização do *stock*, definiu-se o método `actualiza_med()`, que permite levar a cabo duas acções distintas: adicionar ou remover um dado *stock*, através dos métodos `add_stock()` e `rm_stock`, respectivamente. De forma semelhante àquela descrita anteriormente, a escolha do utilizador é implementada através de um *switch*. Quer o nome do mendicamento, quer a quantidade a alterar, são pedidos ao utilizador no mesmo momento, tendo-se tomado esta decisão por uma questão de eficiência, dado que a interacção com o utilizador pode revelar-se algo morosa e reflectir-se num período em que o sistema se encontra à espera do utilizador, por assim dizer.

Relativamente à adição, se o valor do *stock* não for positivo, o programa emite um aviso e repete o pedido de inserção de uma quantidade de unidades. Pelo contrário, se o valor introduzido for positivo, o ficheiro será aberto em modo `LOCK_EX` — já que serão efectuadas alterações —, sendo depois percorrido e o seu conteúdo lido elemento a elemento até que o nome seja correspondente àquele inserido. Nessa altura, o *stock* é actualizado por intermédio da soma do valor introduzido ao valor actual constante da base. Novamente, dá-se a sequência de posicionamento do cursor, escrita, remoção do *lock* e fecho do ficheiro.

Por sua vez, para a remoção considerou-se que, quando o valor de *stock* de um determinado medicamento sofre um decréscimo, considera-se que este foi comercializado. No seguimento deste raciocínio, aquando da actualização de um ficheiro de medicamentos, o ficheiro de vendas deve também ser actualizado, sendo incrementado do valor subtraído ao *stock* do medicamento em questão, motivo pelo qual, inicialmente, ambos os ficheiros são abertos. Numa primeira fase, procede-se à actualização no ficheiro relativo aos medicamentos, efectuando-se um *lock* exclusivo e o processo iterativo de percorrimento do ficheiro já explicitado em instâncias

anteriores, sendo que, quando se verifica uma correspondência entre nomes, é feita uma verificação com o objectivo de apurar se o valor que se pretende subtrair é superior ao *stock* existente, situação que, a confirmar-se, faz com que o valor da *flag* de operação ilegal (variável de controlo `opIlegal` passe a `SIM` e induza o *break*, forçando a saída do ciclo `while` que percorre o ficheiro). Caso esta situação não se confirme, o valor introduzido é subtraído ao valor do *stock* actual, a variável é actualizada com o valor resultante da operação e a *flag* `removido` assume o valor `SIM`. Se tanto a *flag* de operação ilegal como a de `removido` forem nulas, pode inferir-se que o medicamento não existe, acontecimento marcado pela impressão de uma mensagem na consola. À semelhança dos casos anteriores, este acontecimento marca o retirar do *lock* e o fecho do ficheiro de medicamentos, que não será, assim, mais manipulado. De seguida, torna-se necessário actualizar o ficheiro de vendas, mas apenas para os casos em que se tem `removido` = `SIM`, pelo que se recorre uma vez mais ao processo iterativo de *lock*, percorrimento, leitura e comparação. Aquando de uma coincidência, o valor é actualizado com uma soma, dá-se o posicionamento, a escrita e a alteração do valor da *flag* para `SIM`. Neste caso específico, caso o valor de `vendido` seja nulo, significa que o medicamento não foi encontrado no ficheiro de vendas, embora se saiba que existe e que foi actualizado no ficheiro de medicamentos. Assim, para que um medicamenro conste do ficheiro de vendas, é necessário que tenha sido vendido anteriormente ou este terá de ser adicionado.

4.2.4 Listagem de Medicamentos

Conforme explicitado na Secção 3, os medicamentos encontram-se distribuídos por vários ficheiros, pelo que se decidiu dividir a acção de listagem em duas componentes. A primeira (`listar_med_aux`) lista todos os medicamentos de um determinado ficheiro que lhe é passado como argumento, enquanto que a segunda (`listar_med`) invoca a primeira para todos os nomes possíveis de ficheiros. Atente-se que apenas são impressos os medicamentos não marcados como `apagado`. A função `listar_med_aux` abre o referido ficheiro que lhe é passado como argumento com um *lock* partilhado — permitindo que este seja acedido por vários utilizadores, embora nenhum possa fazer-lhe alterações, apenas lê-lo. Por seu lado, a função `listar_med()` vai gerar todos os nomes possíveis de ficheiros (desde `medicamento_aa.dat` até `medicamento_zz.dat`) e invocar a função `listar_med_aux`, passando cada um desses ficheiros como argumentos. Para esta acção, é necessário que todos os ficheiros estejam previamente criados.

4.2.5 Consulta de Stocks

Dado o nome de um medicamento introduzido pelo utilizador, o ficheiro que lhe corresponde é aberto, embora apenas em modo de leitura — `(fopen=ficheiro(me),"r")` —, pois não serão necessárias quaisquer modificações. Mais uma vez, não se justificava, aqui, a utilização do *lock* exclusivo, tendo-se optado igualmente por empregar um *lock* partilhado. Dado um medicamento introduzido pelo utilizador, quando encontrado o seu semelhante no ficheiro, é impressa uma mensagem com o valor do *stock*.

4.2.6 Listagem de Vendas

De forma semelhante, o ficheiro de vendas é aberto em modo de leitura com um *lock*, na medida em que não lhe serão aplicadas quaisquer alterações, sendo depois percorrido e cada medicamento que não se encontre `apagado` é impresso na consola. Por fim, retira-se o *lock* e fecha-se o ficheiro.

Uma das desvantagens associadas a este método — e comum ao `listar_med`, sob condições semelhantes — prende-se com o facto de, dado um cenário com um número de entradas muito elevado, este se tornar

desprovido de sentido e acabar por tornar-se obsoleto. No caso específico do trabalho prático desenvolvido, inseriram-se cerca de 20000 medicamentos, embora apenas 15 tenham sido efectivamente vendidos, pelo que se pode afirmar que esta função pode desempenhar um papel de utilidade considerável, dependendo das condicionantes de teste. De relevar que o código relativo a esta função foi implementado com um erro resultante de uma falha de atenção, para o qual se pretende alertar o utilizador: o cabeçalho correcto da lista impressa é `printf("Nome medicamento : Quantidade vendida \n\n");` e não aquele que actualmente se encontra codificado.

4.2.7 Top de Vendas

O valor do *top* — isto é, o número de elementos que o povoam (x) — é variável e depende inteiramente da vontade do utilizador, podendo assumir qualquer valor, contando que este não seja inferior a 1, por razões lógicas. Após a leitura do valor inserido, é criado um *array* de *struct* Medicamento, para o qual são alocadas x posições de memória, sendo que a função `calloc` retorna um apontador para a memória alocada. O *array* foi inicializado a (-1), com o objectivo de, comparativamente, os valores do *stock* de medicamentos serem sempre superiores. A título de exemplo, de forma a obter o *Top 10* de vendas, são alocadas 10 posições em memória, cada uma com o tamanho de um medicamento. É boa prática libertar a memória alocada no caso de erro ao abrir o ficheiro, pois esta não será necessária.

Enquanto o ficheiro de vendas não acabar — ou enquanto a variável `porAdicionar` (igual ao valor do *top*) não for nula, para todos os medicamentos cujo *stock* seja superior ao daqueles existentes no *array*, este é povoado, de forma ordenada, por esses medicamentos. A ordenação encontra-se fundamentada no algoritmo `Bubble Sort`: se o valor que vai entrar vai para o último lugar, este susbstitui o último; caso contrário, é necessário recorrer a uma variável temporária e desviar uma posição atrás. Sempre que um medicamento entra para o *array* ordenado, a variável `porAdicionar` é decrementada; caso esta variável atinja o valor zero antes que o ficheiro seja todo percorrido, ainda existem valores de *stock* (que, nesta situação, correspondem à quantidade vendida) que ainda não sofreram comparação no sentido de aferir se são ou não elegíveis para entrarem no *array*, pelo que se repete o mesmo raciocínio. Por exemplo, supondo que existem 25 medicamentos e que o valor introduzido para o *top* é 10; assim sendo, os 10 primeiros valores entram para o *array* ordenado, mas ficariam 15 medicamentos por comparar.

Caso seja requisitado um *top* superior à cardinalidade de entradas, isto é, se o valor do *top* for 25 quando o ficheiro de vendas possui apenas 10 medicamentos, estes 10 são impressos de forma ordenada, isto é, o *array* é impresso do maior para o menor valor, isto é, da última para a primeira posição. Após a impressão do *top*, é necessario libertar a memória alocada.

5 Elementos Auxiliares

Com o objectivo de apoiar a elaboração do trabalho prático, foram definidos alguns elementos considerados auxiliares.

5.1 initFileTable

Inicia o *array* bidimensional (tabela) com todos os nomes de ficheiros possíveis, graças a dois ciclos `for`. Revela-se necessário alocar 19 posições de memória, cada uma com o tamanho de um caracter, de forma a possibilitar o armazenamento do nome do ficheiro, cujo valor da `string` é alojado na variável `lower`.

Finalmente, a tabela é povoada com as posições correspondentes ao ficheiro. Por exemplo, se o ficheiro em questão for o `medicamento_be.dat`, este fica armazenada na posição [1][4], porque $b - a = 1$ e $e - a = 4$.

5.2 Função Hash

Recebe um medicamento e, consoante as duas primeiras letras do respectivo nome, retorna o ficheiro onde este medicamento em específico deve ser guardado. Caso o nome seja composto por menos de duas letras ou comece por caracteres não alfabéticos, então o medicamento é enviado para o ficheiro *defaultFile*.

5.3 Método Random

Gera strings aleatórias de, pelo menos, 5 caracteres, utilizadas para originar medicamentos fictícios, que foram criados propositadamente, com o intuito de gerar volume de dados para teste.

5.4 Método Gerar Medicamentos

Cria todos os 676 ficheiros, o `defaultFile` e ainda os ficheiros de vendas, que devem ser criados quando da primeira utilização do sistema. A variável i representa o número de medicamentos a criar, enquanto i for não nulo, gera uma `string` aleatória para o nome do medicamento e um inteiro aleatório para o *stock*. É aberto o ficheiro correspondente ao medicamento e este é inserido caso ainda não exista.

6 Conclusão

Um sistema de gestão de medicamentos funcional foi criado com sucesso, sendo que se crê ter atingido uma construção relativamente eficiente, principalmente devido à decisão de ter vários ficheiros para armazenar os dados relativos aos medicamentos, ao invés de ter apenas um de maiores dimensões. Esta abordagem potencia a eficiência do sistema na medida em que, ao efectuar uma operação sobre um medicamento, não é necessário bloquear todos os restantes, diminuindo ainda o tempo de espera entre *locks* — consequentemente, aumentando a rapidez das acções de inserção, procura e remoção. Assim, a fracção de medicamentos bloqueados é significativamente menor (e, paralelamente, a de medicamentos livres é significativamente maior), veiculando a capacidade de efectuar várias acções em simultâneo, desde que em ficheiros distintos.

De entre os vários testes aos quais se submeteu o programa, destaca-se um em específico, que envolveu a introdução de 2 milhões de medicamentos gerados aleatoriamente. O tempo de carregamento registado foi de, aproximadamente, 4 minutos e 30 segundos, um valor relativamente elevado que se crê, porém, ser compensado pela rapidez associada às acções sobre o sistema: ao efectuar-se uma inserção, o nome do medicamento veiculado apenas tem de ser comparado com, aproximadamente, 3000 outros, ou seja, apenas 0.15% do ficheiro total!

No que concerne sugestões para melhorias futuras, considera-se que a função do tipo *hash* — utilizada para subdividir os medicamentos em ficheiros de acordo com as duas primeiras letras do respectivo nome — poderia ter sido optimizada de forma a atender às três (ou mais) primeiras letras, de forma a agilizar o referido processo. Porém, há que ter em consideração que esta abordagem envolveria a criação de um maior número de ficheiros, pelo que se acredita que a missão de encontrar um ponto de equilíbrio entre o número de ficheiros e o tamanho dos mesmos deve ficar a cargo do parecer do programador e das condicionantes dos programas a desenvolver. Este raciocínio poder-se-ia ainda aplicar ao ficheiro relativo às vendas.

Referências

[1] Apple Inc. 64-bit transition guide — major 64-bit changes. Mac Developer Library. https://developer.apple.com/library/mac/#documentation/Darwin/Conceptual/64bitPorting/transition/transition.html.