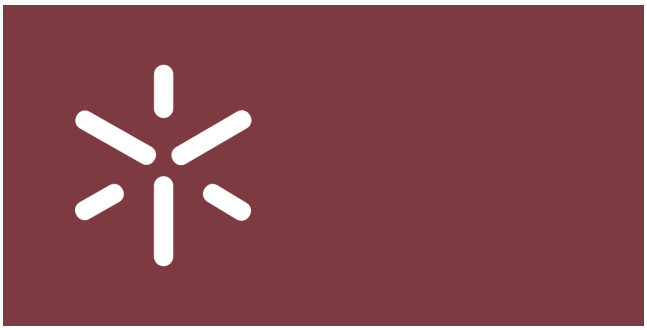

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Sistemas Operativos

Gestão de Vendas

1

Miguel Ribeiro
A67717

Frederico Pinto
A73639

Eduardo Semanas
A75536

13 de Maio de 2019

Conteúdo

Capítulo 1

Introdução

Este trabalho prático foi desenvolvido no âmbito da Unidade Curricular de Sistemas Operativos, e foi-nos proposto pela equipa docente a realização de um sistema de gestão de stocks e vendas. Para isso, possuimos um componente que faz a manutenção de artigos, capaz de inserir novos artigos, alterar o seu nome ou o seu preço. Precisamos de um Servidor capaz de fazer a gestão do stock, capaz de receber pedidos do Cliente de Vendas e também registar vendas que foram efetuadas. Este Cliente de Vendas é o responsável por fazer pedidos ao Servidor. Para além disso, temos também o Agregador que tem como principal função agregar vendas que lhe serão entregues por um processo.

Capítulo 2

Makefile

```
1  compile:
2          gcc -o sv sv.c
3          gcc -o ma ma.c
4          gcc -o cv cv.c
5          gcc -o ag ag.c
6
7  server:
8          ./sv
9
10 client:
11         chmod +x ../scripts/client.sh
12         ./../scripts/client.sh
13
14 mait:
15         chmod +x ../scripts/maitanance.sh
16         ./../scripts/maitanance.sh
17
18 clean:
19         rm ag cv sv ma
20
21 install:
22         mkdir -p ../agregationFiles ../communicationFiles
23
24 uninstall:
25         rm ag cv sv ma
26         rm -r ../agregationFiles
27         rm -r ../communicationFiles
```

Esta *MakeFile* por nós criada permite-nos tratar de uma maneira mais clara toda a aplicação, compilando, executando entre outras coisas.

- **compile** - Compila os programas;
- **server** - Corre o servidor;
- **client** - Corre um cliente;
- **mait** - Corre a manutenção de artigos;
- **clean** - remove todos os executáveis;

- **install** - Instala toda a aplicação;
- **unistall** - Desinstala a aplicação;

Capítulo 3

Contextualização

Após a análise cuidada do enunciado e sua interpretação, vamos agora falar sobre a estrutura de cada ficheiro que o nossa aplicação vai utilizar e de seguida sobre as estruturas que representam os objetos que vão ser tratados e que estão presentes nos ficheiros.

3.1 Ficheiros

Dos quatro ficheiros que o nosso software vai utilizar três deles vão ser escritos em *bytes* e apenas um em *strings*.

O ficheiro *ARTIGOS* vai ser escritos em *bytes* e vai possuir todos os artigos, contendo por cada entrada um inteiro que representa o código, outro que representa a referência para o nome e um *double* que representa o preço.

Tal como o ficheiro *ARTIGOS*, o ficheiro de *STOCKS* vai ser escrito também em binário e por cada entrada tem um inteiro para representar o código do artigo e outro para representar a quantidade de stock disponível. Já o ficheiro *VENDAS* também escrito em bytes, tem por entrada um inteiro que representa o código do artigo cuja venda foi efetuada, outro que corresponde à quantidade e o preço total, num *double*.

Por outro lado o ficheiro *STRINGS* será escrito com caracteres, sendo cada entrada uma *string*, contendo esta o código do nome e o nome.

3.2 Estruturas

Após seguirmos a informação presente no enunciado sobre os ficheiros e termos tomado certas decisões referidas acima, optamos numa primeira instância, por criar uma estrutura para representar um artigo. Cada artigo é composto por um código, referência para o nome e preço.

Listing 3.1: Definição da estrutura Artigo.

```
1 typedef struct article {
2     int code;
3     int ref;
4     double price;
5 } *Article;
```

De seguida, implementamos uma estrutura para representar uma venda. Cada venda possui um código, que representa o código do artigo, uma quantidade e um preço.

Listing 3.2: Definição da estrutura Venda.

```
1 typedef struct sell {
2     int code;
3     int quantity;
4     double price;
5 } *Sell;
```

Como cada inteiro ocupa 4 *bytes* e um *double* ocupa 8 *bytes*, sabemos que ambas as estruturas acima referidas possuem 16 *bytes*, sabendo essa informação, o nosso trabalho com a leitura e escrita nos ficheiros é bastante simplificado.

Por fim, decidimos implementar uma estrutura que nos ajude a tratar da parte do enunciado relativo ao *caching* dos preços. Sendo assim, foi implementada uma estrutura que possui um *array*, um inteiro que representa o número de elementos que o *array* possui e outro que representa a posição de inserção no *array*.

Listing 3.3: Definição da estrutura Cache.

```
1 typedef struct cache {
2     Article* array;
3     int posCache;
4     int nCache;
5 } *Cache;
```

Capítulo 4

Implementação

Como proposto no enunciado, este trabalho prático possui quatro programas independentes e neste capítulo iremos falar sobre a implementação de cada um, referindo todas as estratégias por nós utilizadas.

4.1 Servidor

O servidor de vendas é o responsável por grande parte das operações.

Para a implementação deste programa tivemos que ter em conta na criação da arquitetura a concorrência existente entre os processos que o contactam, sendo esses processos múltiplos clientes de venda e a manutenção de artigos.

Para resolver este problema de concorrência procedemos à implementação de um *FIFO* que servirá de caixa de correio para todos os pedidos que este receberá, tratando um de cada vez.

Caso no *FIFO* receba uma mensagem do cliente executa o que foi pedido, consultando a cache implementada e os ficheiros a que tem acesso, para fazer procura do preço dos artigo e atualizar o stock, caso o pedido seja de alteração ao stock. Se uma compra for pedida pelo cliente, também será adicionada um nova entrada ao ficheiro de vendas.

Por outro lado, se receber alguma mensagem por parte da manutenção de artigos, faz a atualização da cache ou altera o número de artigos que o servidor tem conhecimento, a manutenção também permite ativar o agregador, sendo o servidor o responsável pela sua execução.

Quando este último caso é pedido, o servidor de vendas lê o ficheiro de vendas e divide o número de entradas que o ficheiro possui por um certo número de agregadores, que varia com o número de entradas que estão por agregar, permitindo assim executar agregação concorrente.

4.2 Manutenção de Artigos

Na manutenção do artigo é permitido ao utilizador através do *standard input* pedir ao programa a execução de vários comandos, tais como adicionar um artigo, alterar um nome ou o preço. Nestes comandos, o programa escreve e lê nos ficheiros artigos e nomes.

Quando o nome do artigo é inserido ou alterado, primeiro é efetuado uma procura para verificar se o nome já existe no ficheiro de nomes, caso não exista apenas é adicionado ao ficheiro.

No inicio do programa é aberto um *file descriptor* relativo ao *FIFO* que é a caixa de correio do servidor, isto permite comunicar ao servidor mudanças no preço de um artigo, a inserção de um novo artigo e também executar o agregador.

É também na manutenção de artigos que a compactação do ficheiro de nomes é executada. Isto é, quando o desperdicio de espaço no ficheiro nomes passa os 20%, o ficheiro é tratado de maneira a remover o lixo presente. Mais à frente neste relatório iremos desenvolver mais sobre esta compactação.

4.3 Cliente de Vendas

Este programa é o responsável pela comunicação entre utilizador e servidor. Aqui o utilizador pode pedir o *stock* e preço de um determinado artigo, ou fazer uma alteração de stock de um artigo, podendo adicionar ou retirar stock, efetuando neste último caso uma venda.

No desenvolvimento deste programa, surgiu o problema de fazer com que o servidor consiga enviar a resposta ao processo responsável pelo cliente de servidor, tendo ser possível possuir vários clientes ligados ao servidor ao mesmo tempo. Para resolver este problema, decidiu-se criar um *FIFO* com o nome que é o *PID* do processo, e enviá-lo na mensagem para o *FIFO* do servidor. O servidor, depois apenas se terá que ligar ao *FIFO* com esse nome e responder.

Sendo assim, no programa existe um processo filho que fica à espera de receber respostas do servidor apartir do seu *FIFO* e devolve para o *standard output* a resposta. O processo pai trata de escutar do *standard input* e enviar ao servidor a mensagem que representa o comando que o utilizar desejou e o *PID* do processo para a estratégia de comunicação.

Quando o utilizador sai é enviado um *signal* para matar o processo filho que estava à escuta e um *exec* para remover o ficheiro resultante de criar o *FIFO* para comunicação por parte do cliente.

4.4 Agregador de Vendas

Para fazer a agregação o utilizador comunica com o processo da manutenção que por sua vez, envia uma mensagem ao servidor para assim executar a agragação. Para isso, no servidor, é criado um processo filho e estabelecida uma comunicação com *pipes* que é seguida da execução do programa *ag* por parte do processo filho.

O programa *ag*, agregador, recebe entrada a entrada do ficheiro de vendas que é lido pelo servidor.

Para resolver o problema do somatório, é criado um ficheiro auxiliar e sempre que uma entrada é recebida, é verificado se alguma venda com o mesmo código está no ficheiro auxiliar, caso esteja, são efetuadas as leituras da quantidade e preço, seguido das somas com os valores da entrada e escritas de volta no ficheiro. Caso o código não esteja presente no ficheiro, é inserido no final.

Quando todas as entradas forem recebidas e tratadas, o ficheiro auxiliar é lido e o resultado enviado para o servidor que o vai guardar num ficheiro com o nome da data, escrito em *strings*.

Por fim o ficheiro auxiliar criado para fazer os somatórios é removido através de um *exec*.

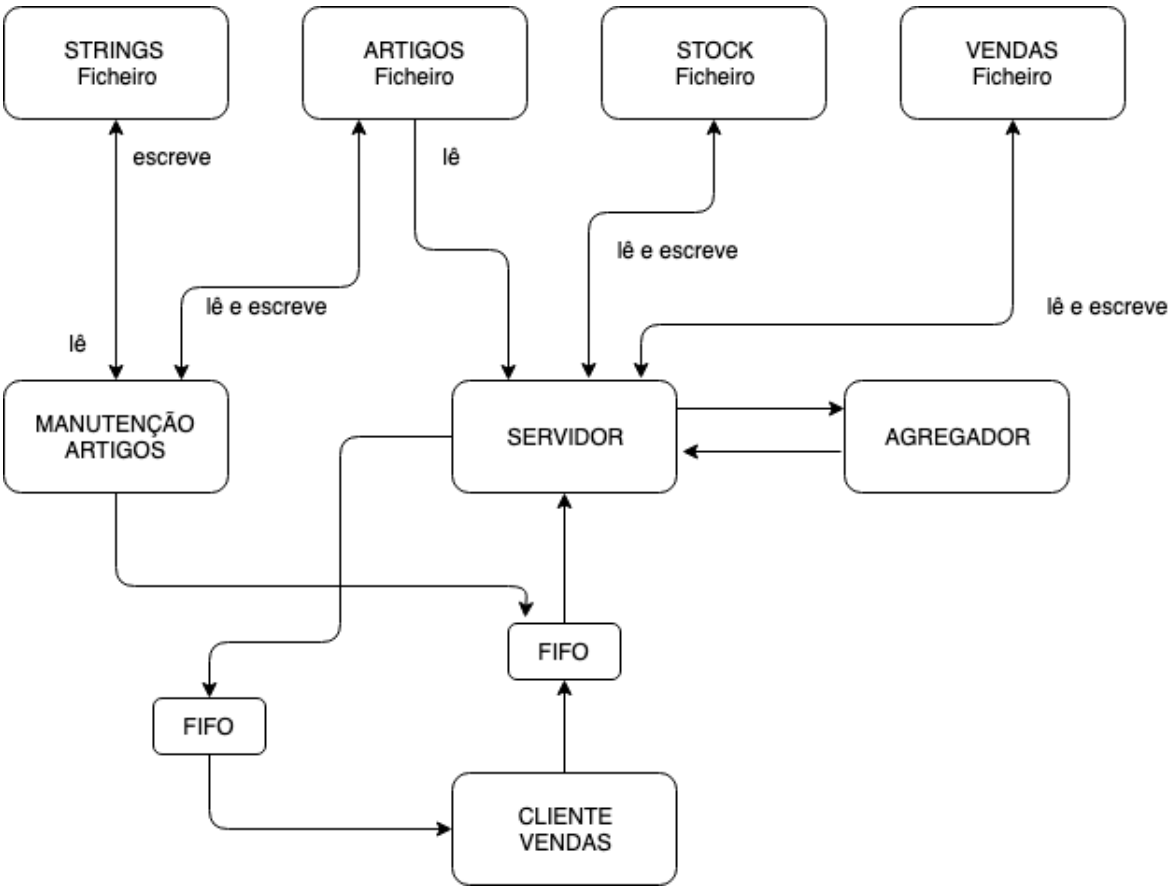

Figura 4.1: Descrição da Arquitetura Implementada.

Capítulo 5

Definições Avançadas

Neste capítulo, falamos sobre as definições avançadas propostas no enunciado. Iremos descrever toda a nossa implementação relativamente a cada operação.

5.1 Caching de Preços

Esta operação foi imposta com o intuito de reduzir o número de acessos ao ficheiro de artigos contendo em *cache* os n últimos artigos pedidos, no processo do servidor.

Para resolver este problema utilizamos a estrutura de *cache* apresentada no capítulo da contextualização. Criando um *array* com apontadores para estruturas que representam artigos. Quando o utilizador através do programa cliente de vendas faz um pedido, o servidor procura o código nesse *array* de *cache*. Caso esteja, é devolvido o preço que está na estrutura, caso contrário é inserido na *cache*.

É de salientar que os preços que estão em cache estão sujeitos a alterações por parte do programa manutenção de artigos. Para resolver este problema, obrigamos a manutenção de artigos a comunicar a alteração ao servidor. O servidor recebe e verifica se contêm esse artigo em cache, caso contenha, altera o valor do preço.

5.2 Agregação Concorrente

A agregação concorrente tem como intuito para paralelizar a operação de agregação, tornando-a mais rápida. Então para implementar essa paralelização, verificamos quantas entradas terão que ser agregadas e dividimos pelo tamanho máximo de entradas que cada agregador pode agregar. Após descobrir quantos agregadores são necessários são criados dois *pipes* por agregador pois cada um será um processo filho, como já referido acima.

O processo pai, que é o servidor, cria os processos filhos, lê o ficheiro de vendas e envia o as entradas que cada agregador irá receber, ficando o último com as restantes entradas presente no ficheiro. Por fim, o servidor recebe pelo *pipe* o resultado de cada processo filho.

Quando tudo foi recolhido, é executado um novo agregador que irá receber toda a informação e agregar uma última vez. O resultado desse agregador é guardado num ficheiro pelo servidor em formato de texto.

5.3 Compactação do ficheiro de Nomes

A compactação do ficheiro de nomes é bastante importante pelo facto de eliminar lixo em memória de informação não utilizada.

Esta funcionalidade está presente na manutenção de artigos e sempre que 20% dos nomes não sejam utilizados, automaticamente, irá entrar em ação.

Nesta implementação é criado um ficheiro auxiliar para irmos armazenando os nomes utilizados em artigos. Após isso, é efetuada uma leitura ao ficheiro de artigos de maneira a descobrir qual o seu nome que está presente no ficheiro

de nomes.

De seguida, é verificado o seu nome está presente no ficheiro auxiliar criado, caso se encontre, é apenas é atualizada a referência para o nome no artigo, caso contrário o nome é inserido no ficheiro de nomes e a referência atualizada.

No final, é efetuado um *dup2* para atualizar a tabela de *file descriptors* inserindo o ficheiro auxiliar no sitio do ficheiro válido, tornando-o inválido. De seguida a remoção do ficheiro inválido é efetuada e o nome do ficheiro auxiliar é alterado através do uso de um *exec*.

Capítulo 6

Conclusão

A realização deste trabalho prático gerou algumas dificuldades, pois apesar de exigir a matéria lecionada nas aulas, foi também preciso efetuar umas pesquisas extra de modo a ultrapassar as barreiras e dificuldades que fomos sentido ao longo do projeto.

A implementação das definições avançadas foi aquilo que mais complicações nos trouxe, principalmente a agregação concorrente devido ao número dinâmico de processos filho, que trouxe uma complexidade mais alta.

Em suma, consideramos deste modo, que foi feito um bom trabalho. Cumprimos todos os requisitos propostos pela equipa docente e isso é algo pelo qual nos orgulhamos. Sentimos que foi um trabalho que nos enriqueceu bastante no que diz respeito a programação concorrente e também a conhecimentos do ambiente LINUX.