Relatório do Trabalho Prático de Sistemas Operativos

André Gonçalves (a80368) Francisco Reinolds (a82982)
José Gomes (a82418)

13 de Maio de 2019

Resumo

Este relatório pretende identificar a arquitetura da solução desenvolvida bem como indicar algumas decisões tomadas, terminando com um conjunto de conclusões sobre o trabalho realizado.

Conteúdo

1 Introdução

Para este trabalho prático, foi sugerida a criação de um sistema de gestão de inventário e vendas, que permitisse o acesso concorrente de vários utilizadores.A arquitetura da solução é baseada no modelo cliente-servidor.

2 Arquitetura

O sistema deverá ser constítuido por vários programas:

- Manutenção de artigos
- Servidor de vendas
- Cliente de vendas
- Agregador de dados

Para cada um destes está destinada uma subsecção deste capítulo, juntamente com as estruturas utilizadas.

2.1 Estruturas de Dados

2.1.1 Article

```
typedef struct article {
 int refI;
 int refF;
 int price;
 int accesses;
}article;
```

Utilizamos esta estrutura para escrita e leitura de artigos do ficheiro ARTIGOS.

1. refI. Esta é a referência inicial da string do nome do artigo em questão, indica o primeiro byte correspondente ao seu nome.
2. refF. Esta é a referência final da string do nome do artigo em questão, indica o último byte da string do nome do artigo. Em conjunção com a referência inicial, permite-nos saber o alcance de leitura pretendido caso queiramos saber o nome do artigo.
3. price. Este é o preço do artigo.
4. accesses. Este é o número de acessos do artigo, utilizamos este inteiro de modo a podermos controlar o que deve ficar na cache, ou seja, os artigos mais acessados ficarão na cache.

2.1.2 Query

Utilizamos esta estrutura para enviar pedidos de CVs / MAs (iremos nos referir a eles como Utilizadores) para o Server.

```
typedef struct query {
    int pid;
 int type;
 int operation;
 int code;
 int value;
 char name[128];
} query;
```

1. pid. Este é o process id correspondente ao utilizador que enviou a query para o Servidor.
2. type. Este é o type do utilizador que enviou a query sendo 0 MA e 1 CV.
3. operation. Esta variável indica-nos que tipo de operação pretendemos realizar. As operações que realizamos e os seus respetivos valores serão explicados posteriormente.
4. code. Esta variável indica o código do artigo sobre o qual irá ser realizada uma certa operação, é utilizada nas operações 3, 4 e 5.
5. value. Esta variável indica-nos o valor que devemos aplicar a uma certa operação, é utilizada nas operações 3, 4 e 5.
6. name[128]. Esta variável permite-nos enviar uma string até 128 bytes, é utilizada no caso 0.

As operações e os seus códigos são:

1. Código 0 - Sincronização. Utilizamos esta operação para armazenar dados sobre o Utilizador em questão numa estrutura apropriada: a estrutura User, que será explicada mais à frente. Implementamos um controlo de concorrência quanto aos MA visto que estes poderiam potencialmente estar a alterar as mesmas áreas no ficheiro ARTIGOS. Escolhemos implementar uma queue FIFO de MA. Caso não haja nenhum MA ativo de momento, é permitida a execução imediata, através de um sinal. Caso contrário é adicionado à queue, e quando chegar à sua vez, ser-lhe-á sinalizado que pode começar a executar.
2. Código 1 - Aviso de Compactação do ficheiro STRINGS. O MA utiliza esta operação após executar uma compactação, avisando assim o server que deve de abrir de novo o ficheiro. Isto é apenas necessário uma vez que quando o SV encerra, este corre o ficheiro de ARTIGOS, STOCKS e STRINGS de modo a imprimir a informação final sobre o que lá se encontra no fim da sessão e estes têm de estar atualizados.

3. Código 2 - Mudança de Preço de um Artigo. O MA utiliza esta operação para sinalizar o Servidor de que este deve executar uma mudança de preço a um certo produto. Preferimos que seja o Servidor a realizar esta operação no produto ao invés do MA dado que como utilizamos o número de acessos de um produto como critério de decisão em relação a que produtos ficam na cache, o Servidor precisará de atualizar estes valores de modo a que possa, caso seja necessário, atualizar os produtos que se encontram na sua cache. Dado este volume adicional de processamento que iria necessitar de ser feito no Servidor, preferimos adicionar a isso a operação de escrita no ficheiro do preço atualizado.

4. Código 3 - Agregação. O MA utiliza esta operação para sinalizar o Servidor de que este deve de agregar os dados do ficheiro de VENDAS.

5. Código 4 - Verificação de Stock e Preço. O CV utiliza esta operação para sinalizar o Servidor que pretende saber o preço e o nível de stock atual de um certo produto.

6. Código 5 - Movimentação de Stock. O CV utiliza esta operação para sinalizar o Servidor de que pretente efetuar uma movimentação de stock de um produto.

7. Código 6 - Término de Ligação. Como fizemos com o código 0, no qual efetuamos a Sincronização do Utilizador com o Servidor, efetuamos também o término da ligação. Aproveitamos esta oportunidade também para, caso seja um MA a efetuar o término, sinalizar o próximo MA de que pode começar a efetuar as suas operações.

2.1.3 Reply

Utilizamos esta estrutura de modo a uniformizar as respostas do Server.

```
typedef struct reply {
    int code;
    int amount;
    int price;
}reply;
```

1. O int code corresponde ao código do artigo.

2. O int amount corresponde à quantidade de artigo movimentada.

3. O int price corresponde ao preço do artigo.

2.1.4 User

Utilizamos esta estrutura de dados de modo a podermos armazenar informações sobre os Utilizadores (CV e MA) em estruturas. Para armazenarmos os utilizadores, temos 2 Listas Ligadas uma para o CV e outra para o MA.

```
typedef struct user {
 int pid;
 char namedPipe[128];
 int type;
 int fd;
}user;
```

1. O int pid correspondente ao utilizador que enviou a query para o Servidor.
2. namedPipe[128]. Esta variável permite-nos saber qual o namedPipe utilizado para o cliente receber respostas do server.
3. type. Este é o type do utilizador que enviou a query sendo 0 MA e 1 CV.
4. fd. Corresponde ao file descriptor do pipe individual de cada Utilizador, através do qual serão enviadas as respostas aos seus pedidos.

2.1.5 Cached

Utilizamos a estrutura cached para armazenar informação dos artigos na cache. A nossa cache é um array de cached, de tamanho variado ajustável por uma variável global. Para decidir quais artigos são mais "populares"decidimos implementar um número de acessos. Consideramos um acesso a um artigo: mudanças de preço, verificações de stocks e movimentações de stocks.

```
typedef struct cached{
 int code;
 int price;
 int stock;
 int accesses;
}cached;
```

1. code. O int code identifica o código do produto.
2. price. O int price identifica o preço do produto.
3. stock. O int stock identifica o stock atual do produto.
4. accesses. O int accesses identifica o número atual de acessos do produto.

2.1.6 Sale

Utilizamos a estrutura sale de modo a representar a informação associada a uma venda.

```
typedef struct sale {
    int code;
 int quantity;
 int paidAmount;
}sale;
```

1. O int code corresponde ao código do artigo.
2. O int quantity corresponde à quantidade vendida.
3. O int paidAmount corresponde ao preço total pago.

2.2 Manutenção de artigos

Este programa permite fazer a inserção de novos artigos no sistema, alterar o nome de um artigo, atualizar o preço de um artigo e sinalizar o Servidor para efetuar uma agregação das vendas. Este programa utiliza três ficheiros: ARTIGOS, STOCKS e STRINGS. No ficheiro ARTIGOS são guardados os artigos (representados pela struct article) existentes, no ficheiro STOCKS o valor de Stock atual de cada artigo enquanto que o ficheiro STRINGS apenas contém a string correspondente ao nome de cada artigo.

2.2.1 Inserção de artigos

A operação de inserção de um novo artigo é efetuada da seguinte forma: primeiro é inicializado o preço do artigo, o seu número de acessos colocado a zero e atribuímos a sua referência inicial, ou seja, o próximo byte a ser escrito no ficheiro STRINGS.

De seguida, é escrito o nome do artigo, sendo possível agora saber a sua referência final, ou seja, o byte anterior ao seguinte a ser escrito no ficheiro STRINGS. Após isso, o artigo é escrito no ficheiro ARTIGOS e o seu stock, 0, é escrito no ficheiro STOCKS no local apropriado.

Sendo que o código do artigo é facilmente encontrado uma vez que a struct escrita para o ficheiro ARTIGOS: "article"têm dimensão fixa: code*sizeof(struct article) ou no caso do ficheiro STOCKS : code*sizeof(int).

2.2.2 Alterar nome de artigo

A operação de alterar o nome de um artigo é efetuada da seguinte forma: primeiro lemos as referências do nome que irá ser substituído, calculamos o tamanho do nome que irá se tornar desperdício e adicionamos isso à nossa variável global, waste, que monitoriza a quantidade de desperdício presente no ficheiro STRINGS. De seguida escrevemos no fim do ficheiro o novo nome, atualizando as referências do artigo para as mais recentes. Após isso, verificamos se o desperdício atual ultrapassa os 20% e caso isso aconteça, executamos a função fileCompressor. Explicaremos o funcionamento da mesma na secção da Compactação do ficheiro Strings.

2.2.3 Alterar preço de artigo

A operação de alterar o preço de um produto é efetuada da seguinte forma: o MA envia uma query com código de operação 3 ao Server, indicando que este deverá atualizar o preço de um certo produto.

Ao receber este pedido, primeiro é verificado se o artigo com aquele código existe no nosso sistema, ou seja, verificamos se o código do produto vezes o sizeof(article) existe dentro do tamanho atual do ficheiro. Caso exista, prosseguimos com a operação, alterando o preço do artigo no ficheiro e seguidamente verificando se este se encontra na cache. Se existir na cache atualizamos lá tanto o preço como os acessos, já que este foi incrementado. Se não existir na cache, atualizamos o valor dos acessos do produto no ficheiro, lemos o seu valor de stock atual e alimentamos essa informação à nossa função updateCache.

É dada a esta função as informações de um produto e efetua uma verificação na nossa cache de modo a mantê-la atualizada.

2.3 Servidor de vendas

O servidor de vendas é o responsável por realizar todas as operações relativas ao cliente de vendas e à alteração de preço de um artigo. Depois de as executar, este envia uma struct reply como resposta para o cliente de vendas ou para a manutenção de artigos:

O servidor mantêm também uma lista com todos os clientes ligados e outra para os "ma".Em cada uma destas listas, a estrutura guardada é a seguinte:

2.4 Cliente de vendas

O cliente de vendas deve interagir com o servidor de vendas. Essa interação é conseguida através de pipes. Uma vez que o sistema deverá permitir a execução concorrente de vários clientes de vendas, os diferentes clientes comunicam com o servidor através do mesmo named pipe: pipe. Mas para o servidor comunicar com cada cliente, cada um dos clientes possui um pipe apenas para esse efeito. Para além disso realizamos um fork para dividir as operações no cliente de vendas: o pai trata do envio das operações para o servidor enquanto que o filho trata das respostas que o servidor enviou.

2.4.1 Mostrar stock e preço de um artigo

Esta operação é efetuada da seguinte forma: inicialmente verificamos através do código do artigo se este se encontra na cache.Caso isso aconteça, o servidor recolhe as informações relevantes, o preço e o stock atual, cria uma estrutura reply com esses dados e envia para o cliente com as informações requisitadas. Caso contrário, o servidor acede ao ficheiro ARTIGOS para saber o preço e ao ficheiro STOCKS para saber o stock e depois envia a reply para o cliente. Em ambos os casos, aumentamos a variável acesses: (na cache ou no ficheiro ARTIGOS).

2.4.2 Movimentação de stock

Aqui consideramos que caso a quantidade seja negativa é o cliente a aumentar o stock, caso contrário trata-se de uma venda.

Primeiramente, verificamos a cache de modo a ver se o produto se encontra lá. Caso se encontre lá, recolhemos todas as nossas informações de modo a que a estrutura reply que enviaremos para o cliente se encontre correta. Finalmente, caso a movimentação do stock seja do Servidor para o Cliente, registamos a venda no ficheiro VENDAS.

Caso o artigo não se encontre na cache, verificamos os ficheiros para recolher e atualizar os dados necessários, ou seja, o stock e o preço, atualizando também a variável de acessos do artigo. Após estas atualizações, enviamos a reply para o Cliente e utilizamos a nossa função updateCache caso seja necessário alterar algum dos artigos que se encontram em cache.

2.5 Agregador de Dados Concorrente

O algoritmo utilizado para a agregação de dados envolve, de forma sucinta, o seguinte: Mudamos o nome do ficheiro "VENDAS"para "VENDASAG"e criamos um novo ficheiro com o mesmo nome ("VENDAS"), de modo a permitir o funcionamento normal do servidor. A informação em "VENDASAG"é alimentada para o *stdin* do agregador através de um pipe. Os dados são analisados e processados por N processos-filhos que os compilam em ficheiros próprios. O numéro de processos-filho oscila com a dimensão do ficheiro. Após o processamento de "VENDASAG"todos os ficheiros criados pelos processos-filho e o último ficheiro resultante de uma agregação de dados, caso exista, são compilados num só ficheiro. E por fim, o ficheiro "VENDASAG"é eliminado e o ficheiro resultante da agregação colocado na diretoria *AgFiles*.

De acordo com o enunciado, o ficheiro resultante da agregação foi guardado em dois formatos, um binário para futuro processamento e outro em *Plain Text*.

2.6 Caching de preços

Para implementar esta funcionalidade, foi necessário utilizar a variável "acesses"anteriormente explicada uma vez que os artigos com maior número de acessos são aqueles que naturalmente estarão guardados também em cache no servidor de vendas. Por essa razão, sempre que for alterado o preço de um artigo, tal deve ser comunicado ao servidor de vendas, para este poder usar os preços atualizados. Esta comunicação é feita através do named pipe: "pipe".

Graças ao Caching de Preços podemos denotar uma melhoria de desempenho em geral visto que os artigos utilizados são predominantemente os mesmos. Por outro lado, caso a utilização de artigos seja completamente aleatória verifica-se uma perda de desempenho inerente ao constante acesso à memória na atualização do ficheiro e da cache.

2.7 Compactação do ficheiro STRINGS

A implementação desta funcionalidade foi talvez a mais interessante de ser realizada pois existem várias maneiras de o fazer, contudo, acreditamos que o algoritmo da nossa solução seja o mais simples.

Para a compactação do ficheiro, utilizamos como foi mencionado previamente a função fileCompressor. Esta função cria um novo ficheiro, newStrings, depois corre todos os artigos no ficheiro ARTIGOS de modo a ir buscar as referências atuais mais recentes e escrevendo os seus nomes mais recentes nesse novo ficheiro, atualizando as referências para refletir as do novo ficheiro. Após ter percorrido todo o ficheiro, a função elimina o ficheiro STRINGS, troca o nome do newStrings para STRINGS e informa o server de que deve fechar o file descriptor para o STRINGS e voltar a reabri-lo visto que este aponta para um ficheiro que não existe.

3 Conclusões

Nesta secção falaremos dos nossos sucessos e adversidades ao longo do projeto. Um dos primeiros desafios deste trabalho foi a forma de comunicação entre processos não relacionados, ou seja, pai e filho. Rapidamente percebemos que teríamos de utilizar Named Pipes de modo a que os processos soubessem de antemão por onde comunicar. Após termos efetuado essa implementação, chegámos à conclusão de que teríamos de tomar proveito de várias estruturas neste programa de modo a efetuar a comunicação entre Servidor / Cliente e fazer a permanência de dados em ficheiro.

Subsequentemente, o trabalho foi bastante "straight-forward"graças à estrutura regular que se verificava nos ficheiros, exceto no STRINGS, o que contribuiu para um mais fácil desenvolvimento de soluções para os problemas em mão.

Quanto a desenvolvimento futuro e melhorias que poderíamos vir a implementar, este poderá passar por permitir ter várias programas MA a executar em simultâneo, algo que seria possível de implementar, por exemplo, com sincronização dos pedidos através do Servidor.

No geral consideramos o projeto bastante interessante e com algumas dificuldades ao longo do mesmo. Estamos felizes com o programa que desenvolvemos visto que realiza todas as tarefas requisitadas pela equipa docente, de maneira eficiente e flexível.