Universidade do Minho

Mestrado Integrado em Engenharia Informática

Trabalho Prático de Sistemas Operativos

Gestão de Vendas

Grupo 65

João de Macedo
A76268

Paulo Pacheco
A77808

Nelson Gonçalves
A78173

6 de Setembro de 2022

Conteúdo

1 Introdução

Este documento apresenta o projeto desenvolvido na unidade curricular de Sistemas Operativos, desenvolvido em linguagem C, descrevendo o problema dado e a forma como este foi resolvido.

Neste trabalho pretende-se construir um protótipo de um sistema de gestão de inventário e vendas. O sistema deverá ser constituído por vários programas: manutenção de artigos, servidor de vendas, cliente de vendas e agregador de dados.

Ao longo do relatório serão mostradas todas as funcionalidades básicas e avançadas que foram construídas e desenvolvidas de modo a ser possível efetuar a gestão de vendas.

2 Descrição do problema

Este trabalho será dividido em 4 programas: *manutenção de artigos, servidor de vendas, cliente de vendas e agregador*. Cada um destes programas corresponde a uma funcionalidade diferente.

Na **manutenção de artigos** será possível adicionar novos artigos, alterar o seu nome e preço e fazer agregações das vendas. Este programa recebe todo o seu input pelo seu stdin, lendo linhas de texto com o formato do exemplo seguinte:

```
$ ma
i <nome> <preço>          --> insere novo artigo, mostra o código
n <código> <novo nome>    --> altera nome do artigo
p <código> <novo preço>   --> altera preço do artigo
...
<EOF>
```

O **servidor de vendas** controla stocks, recebe pedidos do cliente de vendas, e regista as vendas efetuadas. A quantidade de stock de cada artigo será mantida num único ficheiro *stocks.txt* , para todos os artigos. Cada venda efetuada é registada, acrescentando uma entrada a um ficheiro *vendas.txt* , contendo código, quantidade e montante total da venda.

O **cliente de vendas** interage com o servidor de vendas, solicitando-lhe a execução de (uma sequência) de operações que se distinguem facilmente pelo número de parâmetros introduzidos. Caso se introduza o código do artigo, este retorna a quantidade em stock e o preço de uma unidade desse artigo. Se se introduzir o código de um artigo e de seguida uma quantidade, positiva (adicionar stock) ou negativa (venda de stock), este atualiza o stock e mostra o novo stock, como mostrado no exemplo seguinte:

```
$ cv
<código_numérico>               --> mostra no stdout stock e preço
<código_numérico> <quantidade> --> actualiza stock e mostra novo stock
...
<EOF>
```

Por fim, o **agregador**. Este programa funciona como filtro e é ativado quando o manutenção de artigos recebe como input um *"a"*.

```
$ ./ma
a
```

De seguida, criará um ficheiro cujo nome reflete o momento em que a agregação foi solicitada (e.g., 2019-03-29T14:23:56). Este ficheiro contém todas as vendas que ainda não tenham sido agregadas, de forma agregada, de maneira a resumir as vendas de cada artigo, contendo a quantidade total vendida e o montante total de cada artigo.

3 Conceção da Solução

A conceção da solução consiste na criação dos 4 programas: *Manutenção de artigos, Servidor de vendas, Cliente de vendas e Agregador.* De seguida, serão apresentados e justificados certos aspetos que achamos fundamentais relativos à solução que desenvolvemos.

3.1 Formato dos ficheiros

Com o objetivo de evitar o uso de estruturas, isto é, evitar depender da memória disponível do sistema operativo, definimos que cada linha do ficheiro tinha um formato específico. Tomando como exemplo o ficheiro ARTIGOS, este é formado da maneira seguinte.

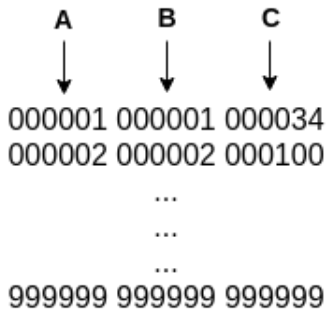

Figura 1: *Ficheiro Artigos*

Com o ficheiro neste formato, cada linha apresenta um tamanho fixo o que nos permite utilizar a função **lseek** para manipular o ficheiro, evitando assim a necessidade de criar estruturas. A letra **A** representa o id do artigo, a letra **B** a linha do ficheiro STRINGS que contém o nome do artigo e a letra **C** o preço do artigo.

Pela imagem verificamos que existem 999999 possibilidades tanto para A, como para B e C. Os restantes ficheiros mantêm uma estrutura similiar, permitindo, mais uma vez, a utilização da função lseek para a sua manipulação.

3.2 Variáveis globais

Com o objetivo de manter a correta inserção nos ficheiros ARTIGOS e VENDAS foram criadas três variáveis globais:

- **idAtualArtigos:** Contém o valor do id do próximo artigo a ser inserido;
- **idAtualString:** Contém o valor da linha onde vai estar o próximo nome a ser inserido;
- **idAtualVendas:** Contém o valor do id da próxima venda a ser inserida;

Quando os programas são executados, o primeiro passo consiste em atualizar as variáveis globais de modo a garantir um correto estado dos ficheiros.

3.3 Pipes com nome

Nesta solução existem no total quatro pipes com nome. Estes pipes permitem a comunicação entre os diferentes programas. A figura seguinte ilustra a "direção"de cada pipe, bem como os processos (programas) associados.

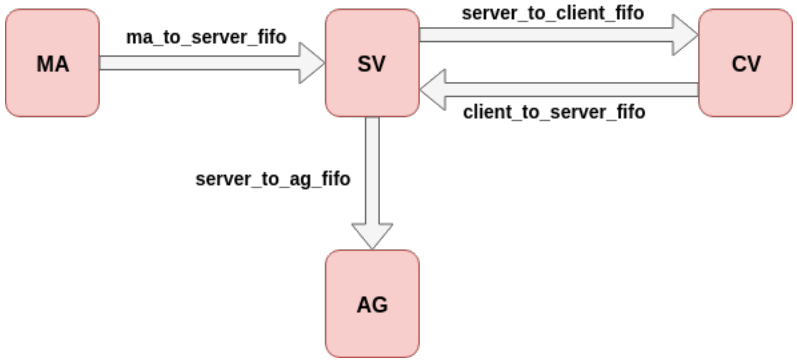

Figura 2: *Pipes com nome*

- **client_to_server_fifo:** Este pipe é utilizado na solicitação de uma operação por parte de um servidor de vendas. O cliente de vendas escreve neste pipe a instrução e o servidor lê deste mesmo pipe a instrução e executa-a.
- **server_to_client_fifo_GETPID:** Este pipe é utilizado na escrita de uma resposta a um determinado cliente de vendas. O servidor escreve a resposta de uma instrução, previamente recebida, neste pipe e o cliente de vendas lê deste mesmo pipe a resposta. Cada cliente de vendas apresenta um pipe único associado para a comunicação servidor-cliente. Isto é garantido pela utilização da função getpid().
- **ma_to_server_fifo:** Este pipe é utilizado aquando da agregação. O processo ma escreve no pipe a letra "a"e o servidor lê esta letra deste mesmo pipe. Neste momento o servidor toma conhecimento que tem de dar inicio á agregação.
- **server_to_ag_fifo:** Este pipe é utilizado aquando da agregação. Após o servidor ler do pipe anterior a letra "a", o servidor escreve neste pipe cada linha do ficheiro vendas que ainda não foi agregada.

4 Manutenção de artigos

Uma vez que este programa será responsável por permitir a adição de novos artigos, bem como a alteração do seu nome e preço, foi feito inicialmente a abertura dos ficheiros necessários, com as devidas permissões de acesso.

```
artigos_fd = open("artigos.txt",O_WRONLY);
strings_fd = open("strings.txt",O_WRONLY | O_APPEND);
stocks_fd = open("stocks.txt",O_WRONLY | O_CREAT | O_APPEND,0600);
```

```
while nBytes lidos maior que 0 do
    split por espaço;
    if primeira letra for i then
        escrever no ficheiro strings o nome do produto;
        construir a string a ser inserida no ficheiro artigos;
        escrever 000000 no ficheiro stocks;
        colocar o offset do ficheiro artigos no fim;
        escrever no ficheiro artigos a string previamente construída;
        mostrar id do artigo inserido e atualizar as variáveis globais;
    else
    end
    if primeira letra for n then
        if id inserido menor que o idAtualArtigos then
            escrever no ficheiro strings o novo nome inserido;
            avancar o offset para a linha do artigo a ser alterado;
            escrever o novo id associado ao novo nome;
        else
            Notificar que o artigo não existe;
        end
    else
    end
    if primeira letra for p then
        if id inserido menor que o idAtualArtigos then
            avancar o offset para a linha do artigo a ser alterado;
            escrever o novo preço;
        else
            Notificar que o artigo não existe;
        end
    else
    end
    if primeira letra for a then
        criar processo filho;
        if processo filho then
            abrir o pipe ma_to_server_fifo para escrita;
            escrever no pipe a letra 'a';
            executar o executável ag;
        else
            Processo pai espera;
        end
    else
    end
end
```

Algorithm 1: MA

5 Servidor de Vendas

Quando o servidor de vendas é posto a correr, os quatro pipes anteriormente descritos são criados. Para além destes é criado um processo filho responsável por ler do pipe 'ma_to_server_fifo'. Quem escreve neste pipe é o ma aquando da receção de um pedido para agregar, escrevendo a letra 'a' no pipe. Como forma de saber qual a última linha agregada criamos um ficheiro 'ultimaLinha.txt' que contém a linha a partir da qual a próxima agregação deve ter início.

Quando o processo filho lê a letra 'a' sabe que deve dar início à agregação. Este coloca o offset a apontar para o início da linha, do ficheiro vendas, apartir da qual começa a agregação e lê linha a linha até eof. Cada linha lida é escrita pelo filho no pipe 'server_to_ag_fifo'. Após serem escritas todas as linhas no pipe, o ficheiro ultimaLinha.txt é atualizado.

O processo pai é responsável por ler do pipe 'client_to_server_fifo' as operações escritas neste mesmo pipe pelo cliente de vendas. Cada tipo de operação é executada por um processo filho que trata de fazer as devidas alterações nos ficheiros e de escrever a resposta no pipe 'server_to_client_fifo', ficando o pai à espera que o filho termine a execução.

6 Agregador

Como dito anteriormente, o filho que está no servidor vai escrever cada linha do ficheiro de vendas, a agregar, para o pipe 'server_to_ag_fifo'. O agregador lê pelo stdin cada uma dessas linhas, sendo isto garantido com as seguintes instruções,

```
char *myfifo3 = "server_to_ag_fifo";
server_to_ag_fifo = open(myfifo3, O_RDONLY);
dup2(server_to_ag_fifo,0);
```

Para a agregação das linhas recebidas pelo stdin, desenvolvemos um algoritmo que recorre ao uso de dois ficheiros, sendo eles, o 'agAuxiliar.txt' e o ficheiro cujo nome corresponde ao instante ("timestamp") da agregação. Estes dois ficheiros são criados antes de serem lidas as linhas através do stdin.

- Ficheiro **"timestamp"**: Cada linha deste ficheiro representa um artigo diferente, e é formada por id do artigo, número total de vendas e montante total. Estes valores são atualizados ao longo da execução do agregador.
- Ficheiro **agAuxilar**: Este ficheiro contém, por linha, dois valores. O primeiro representa o id do artigo e o segundo representa a linha do ficheiro "timestamp"onde se encontra a informação das vendas associadas ao artigo. Inicialmente, este ficheiro tem o segundo campo de cada linha com o valor zero, significando que ainda não existem no ficheiro "timestamp"vendas com esse id de artigo.

```
Criar uma variável (linhaAtual=1) que representa o nrº da próxima linha a adicionar
  uma nova venda no ficheiro "timestamp";
Fazer open do ficheiro agAuxiliar.txt;
Inicializar o ficheiro anterior;
Fazer open do ficheiro cujo nome é o instante em que foi criado (ficheiro "timestamp");
Abrir o pipe 'server_to_ag_fifo' para leitura;
dup2(server_to_ag_fifo,0);

while nBytes lidos maior que 0 do
    split por espaço;
    obter o valor do segundo campo do ficheiro agAuxilar associado ao id do artigo
      recebido;

    if se o valor for zero then
        Atualizar o segundo campo da linha, do ficheiro agAuxilar, associada ao artigo
          com o valor da variável linhaAtual;
        Incrementar variável linhaAtual;
        Escrever a linha lida do pipe no final do ficheiro "timestamp";
    else
        Extrair do ficheiro "timestamp"o valor atual do total de vendas e do montante
          total associadas ao artigo;
        Criar uma variável que armazena a soma do total de vendas anterior com o
          novo valor associado;
        Criar uma variável que armazena a soma do montante total anterior com o
          novo valor associado;
        Atualizar a linha, do ficheiro "timestamp", associada ao artigo com estes novos
          valores;
    end
end
Ler linha a linha do ficheiro "timestamp"e mostrar o seu conteúdo para o stdout;
```

Algorithm 2: AG

7 Cliente de Vendas

Como podem haver vários clientes de vendas a realizar operações em simultâneo, ou seja, a solicitar para o mesmo servidor diversos pedidos, decidimos que cada cliente de vendas tem um pipe com nome único de comunicação entre servidor para cliente. Para a criação de pipes com nome únicos, recorremos à função getpid().

8 Conclusão

Ao longo do processo de desenvolvimento da solução, existiram certas decisões que tiveram impacto na solução final. O facto de usarmos apenas ficheiros, recorrendo ao uso da função lseek, não recorrendo portanto a estruturas, mostrou vantagens ao nível da rapidez dos programas. Para além disto, como os dados são lidos e escritos sempre sobre ficheiros, no caso do servidor ficar inativo a perda de informação é reduzida.
Apesar destes aspetos, estamos cientes que a implementação do aspetos valorizados, presentes no enunciado, iriam melhor a nossa solução.