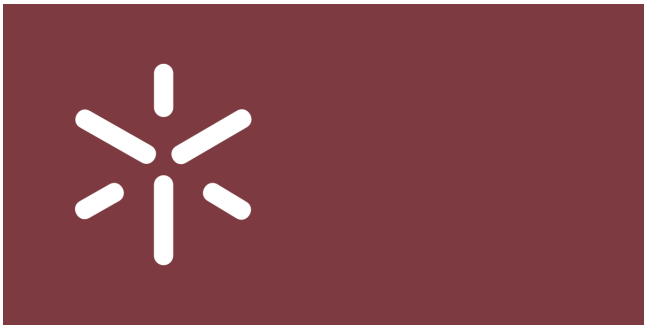

Universidade do Minho

Mestrado Integrado de Engenharia Informática

Sistemas Operativos

Gestão de Vendas

Bernardo Viseu
A74618

Fernando Pereira
A75496

Ricardo Leal
A75411

Conteúdo

1 Introdução **2**

2 Arquitetura da Solução Proposta **2**
2.1 Estrutura Artigo (artigo.c) 2
2.2 *Layout* do Projeto 2

3 Makefile **3**
3.1 Comandos disponíveis 3
3.2 Código Makefile 3

4 Servidor e Cliente **4**

5 Agregador **4**
5.1 Método de agregação 4

6 Manutenção de Artigos **5**

7 Testes e resultados **6**
7.1 scriptMA 6
7.2 scriptCV 6

8 Conclusão **6**

1 Introdução

Este é o relatório do trabalho prático proposto na cadeira Sistemas Operativos em que o objetivo passa por colocar em prática alguns dos conhecimentos teóricos aprendidos ao longo do semestre que assentam sobretudo na gestão de processos, memória, ficheiros e periféricos.

O objetivo do trabalho consiste em desenvolver um sistema de gestão de inventário e vendas. É ainda exigido que a constituição do sistema tenha por base vários programas: manutenção de artigos, servidor de vendas, cliente de vendas, e agregador de dados.

2 Arquitetura da Solução Proposta

Após termos feito uma análise ao problema que nos foi proposto decidimos que, para uma melhor realização das funções que modificam o que está guardado nos vários ficheiros, iríamos utilizar uma estrutura auxiliar, denominada Artigo, que nos facilita muito o acesso a artigos no ficheiro, e a modificação dos parâmetros do Artigo.

2.1 Estrutura Artigo (artigo.c)

O Artigo é então uma `struct` com apenas três parâmetros: `int nome`, `int preco` e `int codigo`.

O `nome` é um inteiro que representa a localização do nome de um certo Artigo, no ficheiro STRINGS. O `preco` é o preço do Artigo, e o `codigo` é o código único do Artigo.

As principais funções definidas aqui são:

- `void save_artigo(Artigo a)` recebe um Artigo, guarda a informação do mesmo no ficheiro ARTIGOS (na devida linha do ficheiro, que é dada pelo código do Artigo), e inicializa a stock do Artigo a 0, no ficheiro STOCKS.
- `Artigo seek_artigo(int code)` recebe um código, e retorna o Artigo correspondente a esse código. Isto é feito lendo a linha do ficheiro ARTIGOS a qual esse código corresponde.
- `int save_name(char* name)` recebe uma string, guarda-a no ficheiro STRINGS, e devolve um `int` que representa um apontador para onde esta string está escrita, este apontador é o que se guarda no ficheiro ARTIGOS.
- *getters* e *setters* que nos facilitam o acesso e alteração do nome, preço, código, stock e apontador do nome dos Artigos.

Todas estas funções foram essenciais no desenvolvimento deste trabalho, e tendo em conta que o objetivo deste projeto é o uso de ficheiros como forma de armazenamento de dados, certificamos-nos que qualquer Artigo inicializado seja libertado.

2.2 *Layout* do Projeto

- `"./"` Diretoria inicial, aqui encontra se a `Makefile`, `Relatorio.pdf`, e as diretorias mencionadas abaixo. É aqui que se geram os executáveis todos do projeto.
- `"database/"` Diretoria onde se guardam os ficheiros ARTIGOS, STOCKS, STRINGS e VENDAS, assim como os FIFOs gerados, e os ficheiros gerados pelo agregador.
- `"include/"` Header files.
- `"src/"` Ficheiros `.c` dos vários programas do trabalho. Guardamos o ficheiro `artigo.c` em `"src/lib/"` .
- `"test_scripts/"` Aqui está guardado o código dos dois scripts utilizados neste trabalho. Também é aqui que ficam guardados os ficheiros de input gerados pelos scripts.

3 Makefile

Para este projeto utilizamos uma Makefile simples permite compilar todos os ficheiros `.c` necessários, limpar os executáveis, e caso necessário limpar toda os Artigos, Stocks, Vendas, Strings e ficheiros de Agregação gerados.

3.1 Comandos disponíveis

- "`make`" compila todos os programas disponíveis (ma, sv, cv, ag, e scripts).
- "`make <nome programa>`" compila apenas o programa pretendido. Opções disponíveis são: `manutencaoArtigos`, `servidorVendas`, `clienteVendas`, `agregador` e `scripts`.
- "`make clean`" apaga todos os executáveis, ficheiros FIFO, e ficheiros gerados pelos scripts.
- "`make resetDB`" limpa todos os ficheiros ARTIGOS, VENDAS, STOCKS e STRINGS, e também ficheiros gerados pelo agregador.
- "`make automate`" corre todos os scripts para criar vários artigos e vendas para os mesmos.

3.2 Código Makefile

Makefile

```
CC = gcc
CFLAGS=-Wall

all: manutencaoArtigos servidorVendas clienteVendas agregador scripts

manutencaoArtigos:
    $(CC) $(CFLAGS) -o ma src/ma.c src/lib/artigo.c

servidorVendas:
    $(CC) $(CFLAGS) -o sv src/sv.c src/lib/artigo.c

clienteVendas:
    $(CC) $(CFLAGS) -o cv src/cv.c src/lib/artigo.c

agregador:
    $(CC) $(CFLAGS) -o ag src/ag.c src/lib/artigo.c

scripts:
    $(CC) $(CFLAGS) -o scriptMA test_scripts/script1.c
    $(CC) $(CFLAGS) -o scriptCV test_scripts/script2.c

automate:
    ./scriptMA
    ./ma < test_scripts/insertScript
    ./scriptCV

clean:
    rm -f ma cv sv ag scriptMA scriptCV
    rm -f database/serverFIFO database/clienteFIFO*
    rm -f test_scripts/insertScript test_scripts/vendasScript

resetDB:
    rm -f database/*
    touch database/ARTIGOS database/STOCKS database/STRINGS database/VENDAS

```

Figura 1: Makefile utilizada.

4 Servidor e Cliente

A comunicação entre múltiplos Processos Clientes e Processo Servidor reflete o paradigma a comunicação inter-processos. Temos multiplos Cliente a enviar pedidos a um servidor que, processa os dados e envia as resposta. Para tal efeito, implementou-se uma arquitetura fazendo uso de pipes com nome (FIFOs), um tipo de ficheiro especial usado para ler e escrever dados. A arquitetura final, são vários cliente a escrever bytes para um único FIFO (Server FIFO), e receber uma resposta pelo seu FIFO (Cliente FIFO) identificado com o seu pid

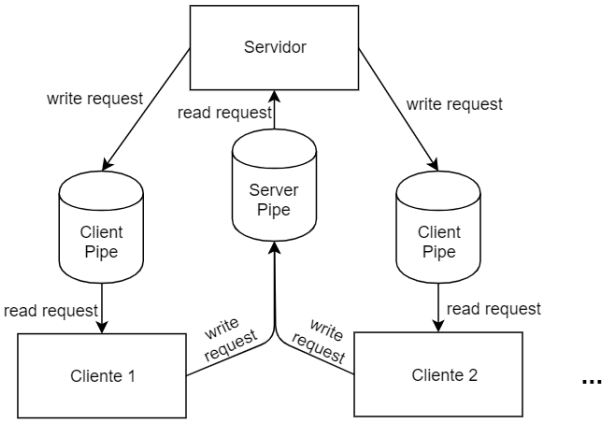

Figura 2: Desenho do funcionamento da estrutura.

O modulo Cliente, é um processo/conjunto de processos que enviam e recebem bytes através de pipes com nome(FIFOs). O cliente lê uma linha do STDIN, ou no caso dos testes lê de um ficheiro, comandos a serem executados. Os comandos são computados por funções de validação, se se provarem válidos são enviados para o Servidor de Vendas juntamente com o pid do respetivo processo Cliente. Os bytes são escritos para um FIFO conhecido por todos os clientes, Server FIFO, que se encontra a ler do FIFO . O Servidor processa os bytes lidos, executa o comando, e escreve o resultado no FIFO do Cliente. Devido a situações muitos especificas de chamadas*bloqueantes*, foi necessário ter muito cuidado com as leituras e escritas que FIFOS com extremos de escrita fechados.

5 Agregador

Para a realização do Agregador decidimos utilizar uma abordagem que consiste em vários processos a correr em paralelo, de forma a aumentar a velocidade deste programa. Isto porque é possível existirem milhares de vendas registadas, e por isso esta otimização é necessária.

5.1 Método de agregação

A primeira coisa que o agregador faz é calcular a data atual. Para isto utilizamos funções contidas em `<time.h>` que se provaram essenciais para esta parte do agregador. Com a data atual já conhecida, guardamos-la data numa string auxiliar, com o formato que queremos (`YYYY-MM-DD'T'HH:MM:SS`).

Esta string vai ser o nome do ficheiro em que o agregado vai estar escrito.

```c
time_t currtime;
char strTime[21] = "";
char file[64] = "";
currtime = time(NULL);
struct tm *localTime;

localTime = localtime(&currtime);
strftime (strTime, 21, "%Y-%m-%dT%H:%M:%S", localTime);
snprintf(file, 64, "database/%s", strTime);

int fdFinal = open(file, O_WRONLY | O_CREAT | O_APPEND, 0666);
```

Figura 3: Excerto do código com o método de guardar a data atual, e criar um ficheiro com esse nome.

Com o ficheiro criado, podemos então seguir para a agregação das vendas. Vamos então, em primeiro lugar, calcular o número de Artigos total que temos.

Para isso abrimos o ficheiro ARTIGOS e utilizamos `lseek()` para calcular a "posição" final do ficheiro, e, sabendo que cada entrada de um Artigo tem tamanho fixo, calculamos o número da linha final, que corresponde ao número de Artigos totais.

Fazemos isto para depois, utilizando um loop `for`, criamos N processos filhos que correm em paralelo (em que N é o numero de Artigos), e cada filho vai calcular o agregado de vendas para apenas de um Artigo. Cada agregado é escrito no ficheiro criado anteriormente, e assim fica concluído o agregado de todos os Artigos.

```c
int fdArtigos = open("database/ARTIGOS", O_RDONLY);
int posFinal = lseek(fdArtigos, 0, SEEK_END);
int codeFinal = (posFinal / 65);
int i = 0, status = 0;
pid_t pid;
for(i = 1; i <= codeFinal; i++){
    if ((pid = fork()) == 0) {
        char* res = agregate(i);
        write(fdFinal, res, strlen(res));
        _exit(0);
    }
}

for(i = 1; i <= codeFinal; i++){
    wait(&status);
}
```

Figura 4: Excerto do código com o método de criar os processos filhos. A função agregar() retorna o agregado de apenas um Artigo.

O agregador pode ser executado diretamente a partir do seu executável, ou a partir do comando `"a"`, no programa `./ma` .

6 Manutenção de Artigos

O módulo da Manutenção de Artigos pode ser visto como a parte administrativa do Sistema, juntamente com as funções que trabalham com a estrutura Artigo, são criadas as bases para fazer leitura do STDIN, funções de manuseamento dos ficheiros ARTIGOS, STRINGS, STOCKS. Representem funções de base para o trabalho. O programa de Manutenção de Artigos (`ma`) tem as seguintes funcionalidades:

- `"i <nome> <preço>"` cria um novo Artigo com as informações dadas, guarda o no ficheiro ARTIGOS (e o nome dele em STRINGS) e mostra ao user o código do Artigo criado.
- `"n <codigo> <novo nome>"` altera o nome de um Artigo, para isto adiciona-se o novo nome ao ficheiro STRINGS, e muda se o apontador de nome do Artigo para essa posiçao no ficheiro.
- `"p <codigo> <novo preço>"` altera o preço de um Artigo, para isto basta mudar o valor de preço que está guardado para esse Artigo no ficheiro ARTIGOS.
- `"c <codigo>"` mostra ao user os vários dados de certo Artigo (como nome, preço, etc...).

- `"l"` lista todos os artigos atualmente guardados.
- `"m"` mostra o menu de comandos do `ma`.
- `"a"` corre o agregador.

Este programa utiliza muitas das funções definidas em `artigos.c`, para a criação de alteração dos Artigos. Só tivemos de ter o cuidado de libertar a memória do programa que estava a ser utilizada pelos Artigos.

7 Testes e resultados

Para melhor testarmos este projeto decidimos implementar dois *scripts*, o `scriptMA` e o `scriptCV`.

7.1 scriptMA

Este *script* é utilizado para gerar um ficheiro com comandos de inserção de artigos. O programa pede ao *user* a quantidade de Artigos que se quer gerar, e para cada artigo, escreve se no ficheiro `test_scripts/insertScript` o seguinte:

`"i A<CODE> <CODE>"`

Assim, se quisermos gerar 100 artigos, o ficheiro vai desde o `"i A1 1"` até `"i A100 100"`. Este ficheiro tem de ser dado como input a `./ma` da seguinte maneira: `"./ma < test_scripts/insertScript"` de maneira a guardar todos os artigos nos devidos ficheiros.

7.2 scriptCV

Depois de guardados os Artigos, este *script* vai, utilizando um ficheiro de input para o `./cv`, simular várias vendas para cada artigo, da seguinte forma:

Primeiro, para cada artigo guardado, vai gerar um comando que fornece 100 de stock, por exemplo, para o artigo 53 seria escrito `"53 100"`. É necessário fazer isto porque todos os artigos criados no `ma` são inicializados com 0 stock.

Depois disto, para cada artigo é feita uma venda de 10 artigos, ou seja `"53 -10"` (para o artigo 53), e este ciclo é repetido 10 vezes, fazendo então com que a stock de todos os artigos volte para 0, e tenham sido feitas 10 vendas para cada artigo.

Para se aplicarem estes comandos, é necessário iniciar o servidor de vendas (`./sv`), e depois, um terminal separado, executar um cliente (ou mais), com o ficheiro `test_scripts/vendasScript` como input: `"./cv < test_scripts/vendasScript"`.

Para ser mais simples a execução de testes, o comando `"make automate"` executa automaticamente todos os scripts de maneira a só ser necessário executar o ultimo passo, com o servidor e o cliente.

8 Conclusão

Este trabalho mostrou se ser desafiante mas foi essencial para o melhor entendimento de muitos temas da cadeira de *Sistemas Operativos*. Pondo em prática tudo o que aprendemos nas aulas num projeto final foi uma experiência muito gratificante.

Ficamos satisfeitos com o resultado final, e, apesar de sabermos que podíamos ter implementado mais *features* a este projeto, decidimos que era melhor aperfeiçoarmos o que era essencial para o funcionamento correto do projeto, e minimizar o número de *bugs* presente no mesmo.