UNIVERSIDADE DO MINHO
MIEI

SISTEMAS OPERATIVOS

Gestão de Vendas

Um protótipo de um sistema de gestão de inventário e vendas

Grupo 99:
João Nunes - a82300
Luís Braga - a82088
Luís Martins - a82298

Braga, Portugal
13 de Maio de 2019

Conteúdo

1 Introdução

O presente relatório documenta o projecto proposto na unidade curricular de **Sistemas Operativos**, projecto este com o tema **Gestão de Vendas** e com o objectivo de se implementar um sistema de gestão de inventários e vendas.

Tendo em conta o objectivo deste projecto elaboraram-se quatro programas distintos: manutenção de artigos, servidor de vendas, cliente de vendas, e agregador de dados.

O programa de manutenção de artigos permite inserir novos artigos.

O servidor de vendas controla os stocks, recebe pedidos de clientes e regista as vendas efectuadas.

O cliente de vendas interage com o servidor de vendas e solicita-lhe a execução de operações como verificar a quantidade se stock e o preço de um artigo, efectuar vendas ou entradas de stock com os parâmetros correctos. Podendo-se executar concorrentemente vários clientes de vendas.

Por último temos o agregador, que funciona como um filtro e percorre entradas no formato do ficheiro de vendas e produz os dados agregados de cada artigo com vendas efectuadas, o código do mesmo, a quantidade total e o montante total de vendas do artigo (mantendo o formato do ficheiro de vendas).

2 Descrição do Projecto

Uma vez que o projeto engloba uma totalidade de quatro programas, o grupo de trabalho debruçou-se primeiramente pela manutenção de artigos.

2.1 Manutenção de artigos

Na manutenção de artigos, são manipulados principalmente os ficheiros referentes aos artigos e as strings, sendo que quando porventura é adicionado um artigo, é também criado o ficheiro stocks sendo este inicializado a zero. Dentro do ficheiro strings, encontra-se apenas os nomes dos artigos sendo que a linha do nome de cada produto corresponde à referência encontrada no ficheiro artigos. Dentro do ficheiro artigos, encontra-se primeiramente a referência do artigo no ficheiro Strings, ou seja, a linha onde se poderá encontrar o nome referente ao artigo que se está a tratar neste ficheiro, de seguida encontra-se o preço do artigo.

```
ARTIGOS
Linha 1 : 2 10
Linha 2 : ...
STRINGS
Linha 1 : ...
Linha 2 : iogurte pêssego
Linha 3 : ...
```

Onde, neste caso, por exemplo, o artigo que se localiza na linha 1, possui a referência para a linha 2 no ficheiro strings, sendo esse artigo o iogurte de pêssego e possui o preço de 10 unidades monetárias.

Portanto, tendo explicado o funcionamento base deste ficheiro, segue-se a explicação da implementação do programa responsável pela manipulação deste tipo de ficheiro.

Em primeiro plano, foram criadas uma série de funções com o intuito de auxiliar na implementação final, começando pela **insertString** cujo intuito é adicionar um nome de um produto, passando como argumento o caminho do ficheiro, retornado o número da linha, sendo esta função aplicada ao ficheiro strings. Para tal, é primeiro necessário abrir o ficheiro, de seguida é utilizada a função read no *file descriptor* do ficheiro dentro de um ciclo while, sendo que de cada vez que encontra um $"\backslash n"$, aumenta o contador das linhas, ou seja o referência, no final do ciclo é escrito o nome do artigo, e um $"\backslash n"$ no final, sendo retornado a referência do nome do artigo escrito no ficheiro strings.

Também foi elaborada uma função para a inserção de um artigo o **insertArtigo**, onde foram elaborados dois forks encadeados um no outro, onde no fork mais posterior, no processo filho do fork orginal, e no processo filho do fork posterior, junta-se o path da pasta que contem os ficheiros sendo feito um *strcat* com o nome do ficheiro stocks É utilizado o *lseek* para escrever

sempre no fim, inicializando o ficheiro a zero. No processo pai do fork mais posterior, é escrito o nome do produto no ficheiro strings, sendo retornado na função o número da linha onde escreveu, onde se manda um sinal com a referência do artigo adicionado no ficheiro strings. É de referir, que uma vez que se utilizam forks, ambos os processos filho e o processo pai são executados em simultâneo. No processo pai do fork original, é aberto o ficheiro artigos, sendo percorrido o ficheiro todo, incrementado uma variável referente ao número de linhas. De seguida é recebido o sinal mandado do fork anterior com a referência, onde é junto numa string a referência do artigo recebida anteriormente com o preço do artigo passado como argumento na função. São de seguida reservados espaços tanto para a referência do nome do artigo como para o preço de artigo, de modo a salvaguardar a integridade do ficheiro. Tendo feito todos estes preparativos anteriores, é colocado um "$\backslash n$"e é escrito no ficheiro artigos a referência do artigo do ficheiro strings, juntamente com o preço do artigo, sendo retornada a linha do ficheiro artigos onde se adicionou o tal artigo.

A seguinte figura, exemplifica o funcionamento desta função:

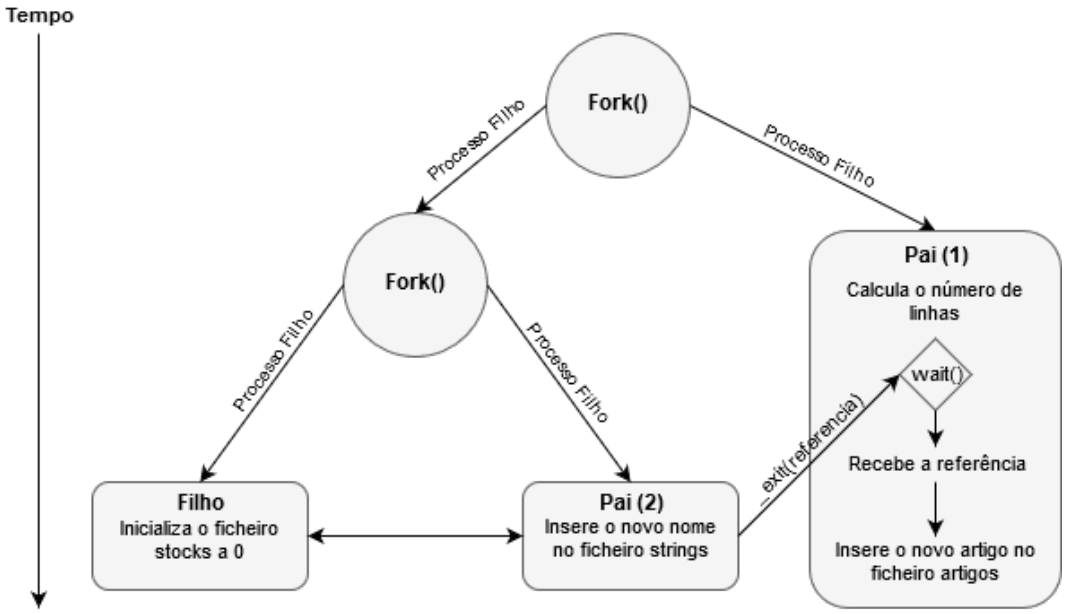

Figura 2.1: Diagrama exemplificativo da função **insertArtigo**.

Na função responsável por alterar o nome do artigo, são novamente utilizados os forks para explorar o paralelismo, no processo filho é utilizada a função previamente explicada o insertString de modo a inserir no final do ficheiro strings o novo nome do artigo sendo para tal aberto o ficheiro strings previamente, onde é passado como argumento nesta função **alteraNomeArtigo**, é também enviado um sinal com a referência da linha do novo nome inserido no ficheiro strings. No processo pai, é primeiramente aberto o ficheiro artigos, sendo ele percorrido até ser encontrado o identificador correspondente no ficheiro artigos. De seguida espera pelo processo filho, onde de seguida, é contemplado o caso onde o id passado como referência não exista no ficheiro artigos, e caso contrário, é recebido o sinal com a nova referência, onde este é passado para um char de modo a alterar no ficheiro, sendo efetuado um lseek, uma vez que já estamos na linha pretendida, contudo é necessário passar para trás uma linha de modo a rescrever a linha pretendida. Sendo de seguida, reescrita a linha com a nova referência e com o mesmo preço.

Também foi elaborada uma função para alterar o preço dos artigos, sendo que para tal é

apenas necessário percorrer o ficheiro dos artigos, onde tal como no anterior é calculado o número de linhas, comparando o número de linhas com o id da linha passado como argumento nesta função **alteraPrecoArtigo**, caso seja diferente o artigo que se pretende alterar o preço não existe, caso contrário existe e novamente é feito um lseek para a linha anterior de modo a reescrever a linha em causa, sendo utilizado um strsplit de modo a dividir os espaços referentes à referência e ao preço, sendo de seguida elaborados os strcat de modo a reconstruir a linha do artigo com o preço atualizado, sendo de seguida escrita a nova linha no ficheiro.

A próxima função que se segue é a **showArtigo**, que irá apresentar o nome e o preço do artigo, portanto é necessário aceder aos dois ficheiros referentes aos artigos e as strings, onde no ficheiro artigos novamente é necessário percorrer as linhas todas, contando o número de linhas, sendo efetuado o mesmo teste que a função anterior, onde caso o número de linhas não corresponda com a referência do artigo, o artigo que se pretende mostrar não existe, caso contrário ele existe, onde novamente é o utilizado o strsplit e o atoi de modo a buscar a referência do artigo e para passar essa referência para um int. De seguida é elaborado um ciclo while, onde é percorrido o ficheiro strings, comparando a referência retirada anteriormente, com o número da linha do ficheiro strings, sendo que quando encontra, ou seja a referência corresponde com a linha do ficheiro strings, é efetuado um readln dessa linha, sendo retirado o nome portanto. No final, é efetuado um conjunto de strcats de modo a conjugar o resultado final com o nome e preço do artigo com a dada referência passada como argumento da função. O resultado depois é escrito para o terminal, via o STDOUT.

Por fim, segue-se a **main**, onde é necessário ler o input efetuado na linha de comandos, onde se possui os vários casos contemplados na leitura do input, ou seja, onde são chamadas as várias funções explicadas anteriormente consoante o input na linha de comandos. Para além das funcionalidades supracitadas, na manutenção de artigos também deverá ser possível correr a agregação, pelo que na main também está contemplada essa operação, que será explicada posteriormente.

2.2 Servidor de Vendas

O servidor de vendas tem como intuito controlar stock, receber e registar as vendas efetuadas, e também terá que providenciar os mecanismos necessários para correr o agregador. Portanto, é fácil de inferir que o servidor terá uma tremenda importância no panorama geral do trabalho proposto.

Começando pela primeira função definida para o servidor de vendas temos o **verPrecoArtigo**, onde é passado o path para o ficheiro artigos e o id do artigo que se pretende consultar o preço, como tal é necessário percorrer as linhas até ser encontrado o id correto, com o auxílio do readln que lê sucessivamente até ao "$\backslash n$", e enquanto não se encontrar o id correto do artigo, incrementa-se o número de linhas. De seguida, e assumindo que encontrou um artigo, é utilizado o *strsplit* de modo a separar o preço do resto da linha do ficheiro artigos. De seguida é passado como um char para um int.

De seguida, o servidor necessita de ser capaz de registar vendas efetuadas num ficheiro específico chamado vendas. Como tal, criou-se a função **regista_Venda**, como esse intuito específico. Portanto, de modo a tirar proveito do paralelismo, criou-se um fork. Como tal, o processo filho apenas necessitará de abrir o ficheiro do artigos, e consultar o preço e consultar o artigo pretendido, com o id passado como argumento na função, e retirar o seu preço, sendo enviado um sinal para o processo pai com esse preço. No processo pai, é necessário tratar do ficheiro vendas, sendo primeiramente aberto o ficheiro, e irá procurar no ficheiro, utilizando

o *lseek* no final do ficheiro. De seguida espera-se que o processo filho acabe e coloca-se o preço retirado da chamada do verPrecoArtigo, feita no processo filho, no status. Prontamente, é retirado o preço do artigo, usando para tal o WEXITSTATUS, sendo que o montante total é calculado multiplicando o preço do artigo pelo stock, que também é passado como argumento nesta função. O preço é passado para uma string, o mesmo acontece para o stock que é passado como argumento na função, o mesmo se aplica ao id. Portanto, no resultado final, é necessário fazer o *strcat* de todos estes elementos, escrevendo essa string no ficheiro vendas.

O ficheiro vendas, portanto possuirá o seguinte formato:

```
VENDAS
Linha 1: 2,10,50
Linha 2: ...
```

Onde, na linha 1 o 2 é a referência do artigo do ficheiro artigos, o 10 é o número de quantidade de artigo vendido, e o 50 o montante total resultante da venda do artigo.

Continuando, foi elaborada também uma função com o intuito de mostrar o stock e o preço de um dado artigo cujo identificador é passado como argumento, a função **mostra_SP**, onde é necessário manipular mais um tipo de ficheiro, o ficheiro stocks. O ficheiro primeiramente é aberto, as linhas de seguida são percorridas, utilizando para tal o *readln*, onde dentro desse ciclo é incrementada uma variável linha que indica as linhas percorridas até ao dado momento. São também elaborados alguns testes, caso não seja encontrada a linha que corresponde ao id passado como argumento, ou seja, uma linha não existente atualmente, é retornado um NULL, ou caso o *readln* não tenha conseguido ler nenhum carater, é também retornado outro NULL. Posto estes mecanismos de segurança, é aberto o próximo ficheiro, o ficheiro artigos de modo a retornar o preço do artigo com o dado identificador passado como argumento nesta função, usando para tal a função supracitada (*verPrecoArtigo*). De modo a construir o resultado final da função, é necessário alocar o espaço necessário para construir o resultado, usando o *malloc*, de seguida, é apenas necessário retirar o stock da linha encontrada e juntar com o preço retirado através do mecanismo explicado anteriormente, sendo retornado como o resultado final um char*, que contém o stock e o preço do artigo com o id passado como argumento.

O servidor também necessitará de atualizar o stock, onde é necessário passar como argumento a referência, ou id, do artigo e um dado stock com o intuito de adicionar ou remover stock do dado artigo. Para tal, foi elaborada a função **atualiza_Stock**, onde novamente, é aberto o ficheiro stocks, sendo este percorrido e incrementado uma variável representativa do número de linhas percorrida até, por ventura, ser encontrada a linha correspondente à linha, representada por um id passado como argumento, mecanismo este análogo ao explicado anteriormente na função *mostra_SP* no que toca na maneira em que percorre o ficheiro stocks. Contudo, é possível ter percorrido todas as listas e mesmo assim o id ser sempre diferente do número de linhas percorridas, portanto é fechado o *file descriptor* associado ao ficheiro stocks, sendo retornado um código de erro. Caso contrário, encontrou o stock correspondente ao id passado como argumento, portanto, é realizado um *readln* de modo a assegurar que conseguiu ler os carateres da linha, e como tal é apenas necessário retirar o stock dessa linha, e como é um char este é passado para um int através do *atoi*. Continuando, a este int, que corresponde ao stock atual em mão, é adicionado o stock passado como argumento nesta função, onde é também testado o caso em que este resultado é menor que zero (realizou uma venda), sendo retornado -1. É necessário também ter em conta a natureza do stock que se pretende adicionar, uma vez que, é possível que esta stock tanto possa ser positivo, ou seja, em título exemplificativo, poderá ter recebido uma encomenda de stock, ou, este stock poderá também ser negativo,

onde neste caso trata-se de uma venda, pelo que é necessário registar essa venda, usando a função anteriormente explicada a função *regista_Venda*, que passa o mesmo identificador da linha e o mesmo stock passado como argumento nesta função. Na reta final da função, é necessário reescrever no ficheiro stocks a atualização do stock, pelo que é necessário utilizar o *lseek* cujo *offset* aponta para a linha anterior à linha atual, sendo de seguida rescrita a linha através do charstock que possui o resultado de somar o stock atual com o stock passado como argumento, usando para tal o sprintf de modo a escrever o int num array de char, onde de modo a reescrever usa-se o *write*. A função retorna o stock atual no ficheiro stocks após a operação aritmética explicada previamente.

O seguinte diagrama, mostra o funcionamento geral desta função:

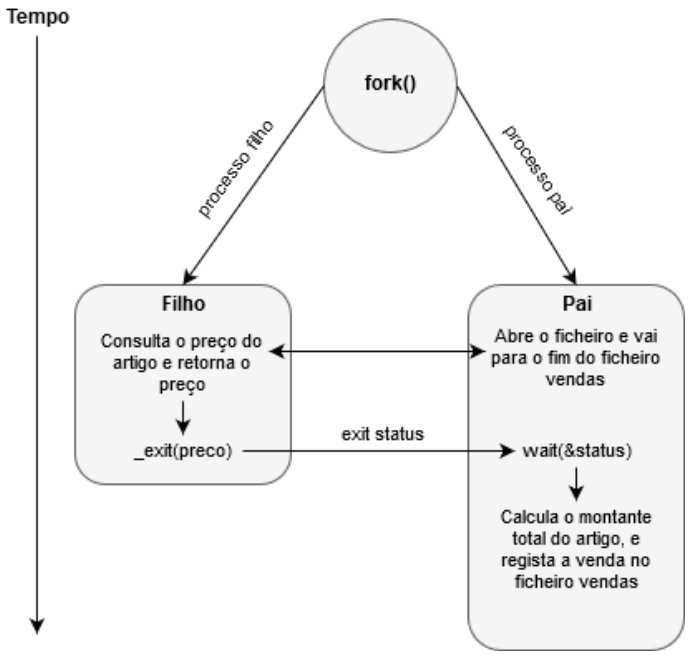

Figura 2.2: Diagrama exemplificativo da função **atualiza_Stock**.

O culminar de todas estas funções é a main, onde é aqui que se simula o funcionamento pseudo-real de um servidor. Para tal recorreu-se aos pipes com nome (named pipe), uma vez que estes permitem que a comunicação seja efetuada entre quais queres processos (*IPC*), sendo que estes criam um género de ficheiro que se distingue pela particularidade de não serem escritos para o disco, ao contrário dos ficheiros "normais", pelo que se revelam como sendo muito menos custosos em termos de performance comparado com uma implementação usando ficheiros *standard*. Portanto, de ínicio é gerado um pipe anónimo referente ao input do cliente no servidor, sendo este pipe um FIFO (First In First Out), que assegura um buffer em memória com sincronização gerida pelo kernel entre quem escreve, ou seja os *writers* e entre quem lê, os *readers*. Uma vez que um servidor deverá estar sempre a correr, a não ser que seja encerrado voluntariamente por alguém, portanto como este deverá estar sempre pronto a receber e tratar de pedidos é colocado num ciclo infinito através de um *while(1)*. O FIFO é aberto para leitura de seguida por parte do servidor, o FIFO é escrito pelo cliente de vendas, e possuirá os pedidos realizados pela parte dos clientes. Caso se consiga ler os carateres no FIFO, é processado o input do cliente através do *readln*, onde é efetuado um *strsplit* dos espaços de modo a averiguar o número de argumentos enviados pelo cliente de vendas ao servidor, sendo que o cliente apenas poderá requerir duas operações ao servidor, sendo essas operações representadas pela

função *mostra_SP* e a *atualiza_Stock*. Caso o resultado deste *strsplit* seja dois, então é porque o mecanismo processado corresponde à função cujo intuito é mostrar o stock e o preço de um determinado artigo, o *mostra_SP*. Caso o resultado seja 3 então é necessário atualizar o stock, através do *atualiza_Stock*, onde se verifica a integridade do resultado desta função, caso o resultado do processo do *atualiza_Stock* seja maior que zero, então é necessário imprimir o resultado para o output, caso contrário ou o artigo que se pretende atualizar o stock não existe ou não há stock suficiente para atualizar o artigo. Para além disso existe um processo filho responsável por receber os pedidos de agregação da manutenção de artigos, executando o *execv* de modo a executar o agregador, passando o resultado da agregação para o lado da manutenção de artigos.

2.3 Cliente de vendas

O cliente de vendas interage principalmente com o servidor de vendas, onde as operações que realiza sobre o servidor se distinguem pelo número de argumentos, tal como foi explicado na secção referente ao servidor de vendas. Portanto, no cliente de vendas, também se tira partido dos pipes anónimos, sendo criado um FIFO com o output dos comandos realizados pelo cliente, de seguida enquanto que é possível ler os argumentos realizados no *STDIN* (Standard Input), o comando é processado utilizando o *strsplit* que parte o input com o critério do espaço onde coloca num endereço o número de argumentos partidos da operação solicitada pelo cliente. Uma vez que o cliente de vendas apenas poderá requerer duas operações, sendo que a primeira possuirá apenas um argumento e a segunda dois, é testado se o número de argumentos varia entre esses dois delimitadores, de seguida é necessário processar tanto a primeira como a segunda operação. Caso o número de argumentos seja dois, então corresponde ao comando responsável por atualizar o stock, onde é aberto o FIFO do input do cliente ,gerado pelo servidor, sendo escrita essa operação nesse FIFO, de seguida, como já não é necessário escrever mais nesse FIFO é fechado o input do lado do cliente nesse FIFO. Continuando nesta operação, é aberto o FIFO representativo do output, que possuirá os resultados retornados, onde é utilizada a função *readln* conjugada com o *strcat* de modo a formatar o ficheiro e escrever no resultado o resultado da operação requerida. Caso o número de argumentos seja um, corresponde a mostrar apenas no STDOUT (Standard Output) o preço e o stock de um dado artigo. Como tal, é necessário abrir novamente o FIFO gerado pelo servidor, o inputCliente, sendo escrito para o inputCliente a operação pedida aqui pelo cliente vendas, que se encontrará a espera de ser processada pelo servidor de vendas. Tal como no anterior, é necessário retornado os resultados do comando no STDOUT, sendo que é aberto um ficheiro output, onde é combinado novamente o *readln* com o *strcat* de modo a se preparar para ser escrito o resultado através do *write*.

Portanto, tendo explicado o cliente de vendas, e o servidor, é agora possível desenhar o seguinte diagrama, que, de uma maneira simplificada, mostra o funcionamento geral de ambas estas componentes:

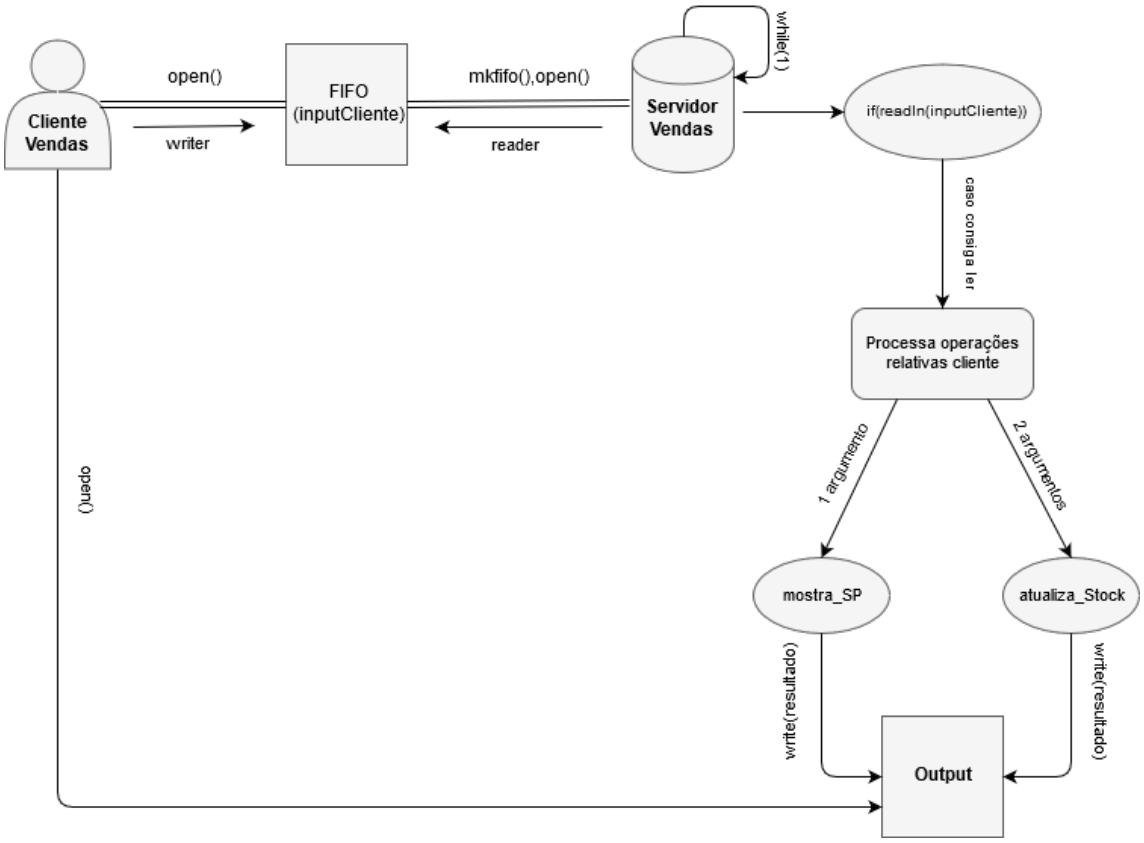

Figura 2.3: Diagrama exemplificativo da interação entre o cliente vendas com o servidor.

2.4 Agregador

O agregador funciona como um filtro, onde irá receber as entradas do ficheiro vendas, e irá compactar cada entrada, como se pode verificar no seguinte exemplo:

```
VENDAS
Linha 1: 1,10,50
Linha 2: 1,30,150
Linha 3: 2,40,4000
Linha 4: ...
RESULTADO AGREGAÇÃO
Linha 1: 1,40,200
Linha 2: 2,40,4000
Linha 3: ...
```

Portanto, o agregador terá que percorrer o ficheiro vendas, e juntar todos os artigos com o mesmo referência num só, somando as quantidades e o preço, portanto, segue-se a seguinte implementação do agregador.

O grupo definiu inicialmente a seguinte estrutura de dados, que é utilizada de modo a guardar a informação relativa a cada venda de artigos do ficheiro vendas, tal como se segue:

```
typedef struct venda{
    int id;
    int quantidade;
    int valor;
}*Venda;
```

Portanto, o grupo optou por criar uma estrutura de dados, ao invés de utilizar ficheiros adicionais, uma vez que esta solução revela-se como sendo menos "custosa".

A primeira função estabelecida foi a função **quantas** que indica quantas vendas existem no ficheiro vendas, para tal é necessário abrir o ficheiro vendas, e percorrer o ficheiro efetuando o *read* até *EOF* (End Of File), incrementando a variável que conta o número de linhas sempre que encontra uma nova mudança de linha ("$\backslash n$").

De seguida, foi também criada a função **cria_espaco** com o intuito de, por exemplo, tendo o seguinte exemplo:

```
VENDAS
Linha 1: 2,30,300
Linha 2: 2,50,500
Linha 3: 6,10,10
Linha 4: 1,9,90
Linha 5: ...
RESULTADO AGREGAÇÃO
Linha 1: 1,9,90
Linha 2: 2,70,800
Linha 3: 3,0,0
Linha 4: 4,0,0
Linha 5: 5,0,0
Linha 6: 6,10,10
Linha 7: ...
```

Portanto, a função supracitada é utilizada, como se pode ver no exemplo anterior, quando existem *"gaps"* entre as referências no ficheiro vendas, de tal forma que, pegando no exemplo anterior, e admitindo que não houveram mais referências entre o 2 e o 6, irá colocar as referências intermédias entre esses valores e irá inicializar a quantidade e o preço total a zero. A função portanto, irá ter como argumento o *linhascriadas* que indica quantas linhas foram criadas no ficheiro resultante da agregação e o *novalinhas* que indica quantas linhas serão criadas. Inicialmente é declarada a variável *linha* que será igual a *linhascriadas*, de seguida é aberto o ficheiro resultante da agregação, e é alocado o tamanho a estrutura vendas. De seguida é utilizado o *lseek* de modo a procurar dentro do ficheiro resultante da agregação onde irá apontar para o fim do ficheiro. De seguida, é elaborado um ciclo while, onde é testada a condição em que o número de linhas criadas tem de ser menor que o número de linhas que se pretende criar. De seguida, a estrutura definida pelo grupo é manipulada, de tal forma que o id da venda será dado pelo número de linhas criadas mais um, enquanto que o resto dos parâmetros da estrutura da venda será inicializado a zero, sendo de seguida escrito para o ficheiro resultante da agregação onde o número de linhas criadas é incrementado.

Continuando, a função **quantasA** é utilizada no ficheiro resultante da agregação de modo a indicar a quantidade de linhas é que foram agregadas, ou seja, quantas linhas existem no ficheiro. Portanto, primeiramente é aberto o ficheiro resultante da agregação, e é criada uma

variável indicativa do número de linhas. De seguida, é declarada a estrutura da venda juntamente com uma alocação da memória, sendo que de seguida é utilizada a função *read* de modo a ler linha a linha é incrementar a variável indicativa do número de linhas, sendo esta retornada no final da função.

De seguida, foi criada a função **agregaArtigo**, onde é necessário passar como argumento a esta função um caminho, que será o ficheiro resultante da agregação, a referência do artigo do ficheiro vendas, a quantidade e o valor retirado, sendo esta função chamada tantas vezes quantas entradas o ficheiro vendas possui. O objetivo desta função é, para vendas que possuem a mesma referência, agrega esse artigo utilizando a mesma referência, somando a quantidade e o total faturado. Portanto, inicialmente é necessário abrir o ficheiro da agregação resultante, de seguida é necessário utilizar a função *lseek* de modo a apontar para a linha cuja qual é pretendido que seja lida. Aloca-se a memória a estrutura vendas, e é lido o ficheiro resultante da agregação, as quantidades e os valores são somados de modo a agregar os artigos com o mesmo id, aponta para o fim do ficheiro e escreve no final do ficheiro resultante da agregação.

Segue-se a função imperativa da funcionalidade da agregação, a função **agregar**, onde inicialmente irá preparar para a criação do ficheiro relativo ao histórico da agregação, que irá possuir o seguinte formato:

```
HISTÓRICO AGREGAÇÃO
2019-5-13T0:28:26,16
2019-5-13T0:30:13,16
2019-5-13T0:31:29,18
...
```

O ficheiro responsável pelo histórico da agregação irá indicar a data, a hora, os minutos e o segundos juntamente com o número de linhas agregado. Esta construção é feita de seguida, onde é elaborado o *localtime* que irá indicar essa mesma data, a hora e os minutos, sendo elaborado um *sprintf* para mandar a string contendo estes dados formatados para o nome do ficheiro pretendido, que irá ser o resultado da agregação a cada determinado momento, admitindo que de facto foi elaborada alguma agregação do ficheiro vendas. De seguida é elaborado um teste de modo a testar se não existe nenhum ficheiro histórico criado, caso não exista, é aberto o ficheiro histórico com uma flag para o criar, sendo de seguida executado o *touch* dentro da função *excelp* de modo a criar um novo resultado a agregação segundo o nome explicado anteriormente, ou seja, com a data, hora, minutos e segundos. Caso já possua um histórico de agregação, é necessário percorrer o ficheiro do histórico da agregação, guardando o número de carateres lidos, de seguida, é utilizado o *lseek* de modo a apontar para a última linha escrita do ficheiro do histórico da agregação. De seguida, é elaborado um *strsplit*, de modo a partir o que foi lido pelas vírgulas que existem dentro do ficheiro do histórico da agregação. De seguida é elaborado um *atoi* de modo a se perceber a linha inicial do ficheiro do histórico da agregação, sendo que dentro de mais um fork, é utilizado o comando para copiar ou seja o *cp*, utilizando o *execlp* de modo copiar o ficheiro antigo para um novo ficheiro com um novo nome. No fim de todos estes testes, é agora possível correr a agregação, onde é necessário abrir o ficheiro vendas, inicializar o ficheiro do histórico da agregação com apenas uma linha, com a data atual e com uma vírgula, sendo que de seguida, escrito para o ficheiro o número de linhas do ficheiro vendas, através da função *quantas* explicado anteriormente, juntamente com um "$\backslash n$" de modo a mudar de linha. É instanciada ainda neste momento, uma variável responsável por contar o número de linhas percorridas, de tal maneira que, enquanto que a linha em que está a percorrer é diferente da linha inicial e enquanto que é possível ler do ficheiro vendas, incrementa o número de linhas. De seguida, é também efetuado o cálculo do número de linhas agregadas

no ficheiro resultante da agregação. Continuando com a leitura do ficheiro vendas, através do *readln*, é elaborado mais um *strsplit* de cada entrada do ficheiro vendas pelas vírgulas, com o seguinte funcionamento:

```
VENDAS
Linha 1: 2,10,50
Linha 2: ...
APLICADO COM O STRSPLIT
2 10 50 ...
```

Portanto, a primeira entrada da lista resultante corresponde à referência do artigo, sendo que enquanto que a referência do artigo é maior que o número de linhas do ficheiro resultante da agregação, será necessário criar os espaços pelo meio da referência da agregação, tal como foi explicado na função *cria_Espacos*. Sendo atualizada de seguida a linha do ficheiro da agregação resultante com a referência do artigo. Uma vez tratada desta exceção, é possível agora utilizar a função do *agregaArtigo* de modo a possibilitar a agregação de cada artigo, sendo passado como argumento o ficheiro da agregação criado, com a referência do artigo retirada anteriormente, juntamente com a quantidade e o preço total, também retirados através do *strsplit*. No final disto tudo, é agora possível fechar o ficheiro vendas.

De modo a esquematizar toda esta linha de pensamento, foi criado o seguinte diagrama;

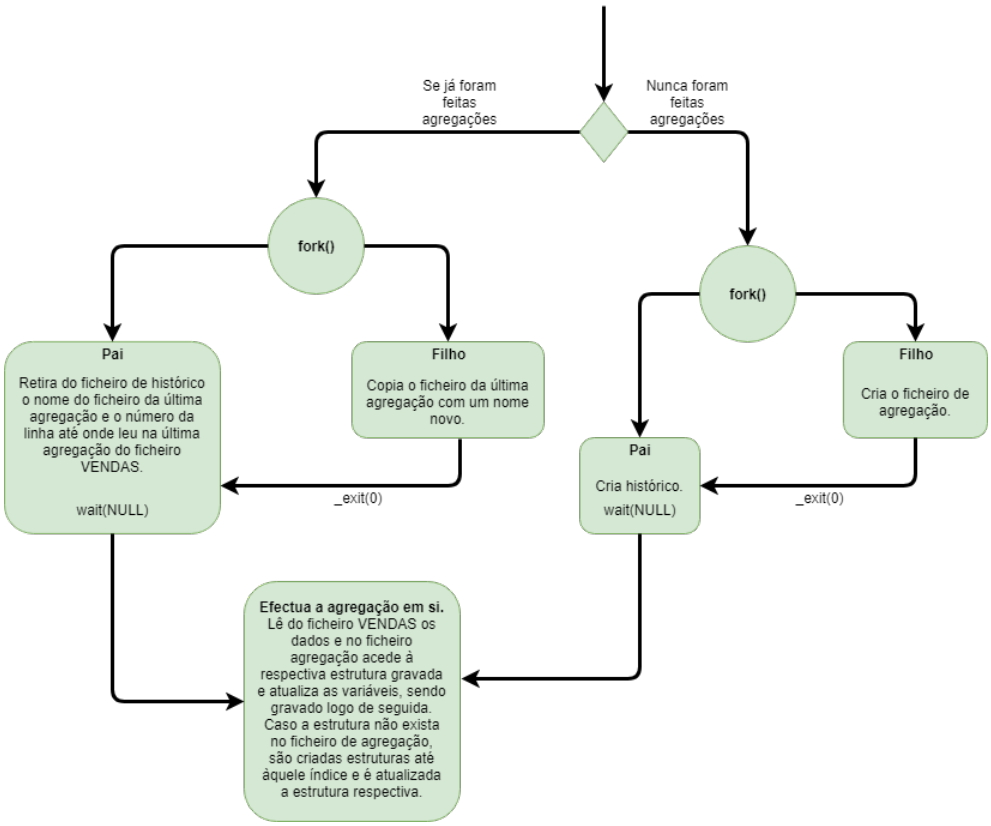

Figura 2.4: Diagrama exemplificativo do funcionamento da função **agregar**.

Tendo explicado todos estes mecanismos, é agora apenas necessário chamar nesta a função do agregar, que é onde se localiza todos os métodos supracitados.

3 Conclusão

Em suma, o grupo de trabalho considerou que o trabalho proposto possuía uma dificuldade elevada, dificuldade essa que em vários momentos levou o grupo a parar e a reconsiderar a abordagem realizada de modo a enfrentar o problema.

Dentro das dificuldades encontradas, a maior parte envolve a comunicação entre o servidor de vendas e o cliente de vendas, uma vez que a utilização de pipes com nome envolve um todo conjunto de conhecimentos que aborda basicamente toda a matéria dada na unidade curricular. Uma vez que a data de entrega se aproximava, rapidamente foi necessário que o grupo consolidasse a matéria, de modo entregar uma versão do trabalho funcional e robusta.

Contudo, o grupo reconhece que o trabalho possui falhas, por exemplo no que toca à gestão de concorrência no servidor de vendas, este poderia ter sido melhorado caso possuíssemos mais tempo, o mesmo se aplica para o agregador, mais concretamente o agregador concorrente.

Mesmo assim, o grupo considera que elaborou um trabalho razoável dentro do espaço de tempo que teve para o fazer, sendo este trabalho sem dúvida um trabalho enriquecedor no que toca à utilização de funções dentro do mundo de *SO* permitiu com que o grupo conhecesse melhor o sistema operativo (Linux neste caso), e o seu funcionamento geral.