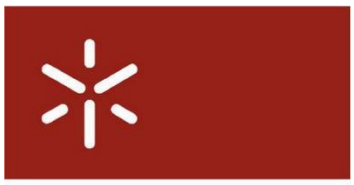

Universidade do Minho

Universidade do Minho

Relatório do Trabalho Prático SO

Mestrado Integrado em Engenharia Informática

Grupo 38

André Morais	A83899
Luís Ribeiro	A85954
Pedro Rodrigues	A84783

Introdução

O trabalho realizado foi desenvolvido em linguagem C e teve como principal objetivo a gestão dos artigos e das vendas dos mesmos. Para a concretização do objetivo, tivemos de definir 4 programas no sistema nomeadamente, Manutenção de Artigos, Servidor de Vendas, Cliente de Vendas e Agregador de Dados.

Inicialmente, começamos por criar os programas manutenção de artigos e o cliente de vendas, separadamente. Mais tarde, definimos o servidor de vendas e estabelecemos a ligação servidor/cliente através de um *pipe* criado e criamos o agregador. Ao longo do tempo fomos melhorando os programas, especialmente o agregador.

O objetivo deste relatório é justificar as nossas escolhas e também explicar as estratégias usadas para o desenvolvimento dos vários programas.

Manutenção de Artigos

Numa primeira fase, começamos por definir uma *struct* Artigos com o preço do artigo inserido, o tamanho da *string* e o número da posição onde esta começa, para depois ser mais fácil de removê-la se fosse preciso.

Seguidamente, começamos por fazer o 'insere' artigos. Criamos dois ficheiros, onde num deles ("strings.txt") escrevemos apenas o nome do artigo e no outro ("artigos.txt") íamos escrevendo os artigos inseridos, mas em forma de *struct*, sendo a primeira linha do ficheiro o *pid* do servidor.

Para alterar o nome do artigo acrescentamos o novo nome no final do ficheiro "strings.txt" e atualizamos a posição e tamanho da string do devido artigo.

Para alterar o preço procuramos o artigo no ficheiro "artigos.txt" e alteramos-lhe o preço. No final de alterar o preço enviamos um sinal para o servidor para este saber que foram alterados preços.

Servidor de Vendas

No servidor de vendas, começamos por definir a *struct* Venda, que é definida com um código de produto, a quantidade e a faturação relativa ao seu preço. A *struct* Artigo utilizada é a mesma usada na Manutenção de Artigos.

Inicialmente, carregamos os artigos mais vendidos para a memória, sendo guardados num array, funcionando assim como um sistema de cache para aumentar o desempenho nas funções de procura de preço.

Criamos um *pipe* para a comunicação cliente/servidor e criamos funções de controlo de Stock. A função getStock, se o código pretendido for maior que o número de artigos retorno um erro, caso contrário vai procurar o stock ao ficheiro. A função atualizaStock é idêntica ao getStock. Caso o artigo existir, dá *update* ao *stock* com a quantidade lida do terminal. Se a quantidade for menor que 0, quer dizer que o artigo foi vendido e registámos essa venda.

O servidor lê todas as instruções a partir do *pipe* "instructions_cv" em que todos os clientes escrevem lá as instruções pretendidas. Mal acabam de correr a instrução, devolvem a resposta através de outro *pipe* "response_(pid do cliente)", assim temos a certeza que estamos a enviar a resposta para o cliente certo.

Assim teremos tantos *pipes* de resposta como número de clientes ativos.

Cliente de Vendas

Este programa interage com o servidor de vendas, executando dois tipos de operações: ou atualiza o número do *stock* do produto, ou mostra a quantidade de stock do produto que queremos ver.

O código em si, é uma coisa simples. Recebe informação do *stdin* e escreve no *pipe* de instruções. Depois espera pela execução do servidor e lê do respetivo *pipe* de resposta e depois imprime no *stdout.* A nossa ideia era tornar o cliente concorrente, em que um processo se encarregava de enviar as instruções e o outro de receber as respostas, assim seria possível enviar inúmeras instruções sem ter que esperar pela resposta. Mas não foi possível realizar o desejado.

Agregador

Inicialmente tínhamos um agregador não concorrente. Tentamos começar por uma coisa mais básica, simples e que funcionasse. Mais tarde conseguimos implementar concorrência no agregador.

Como recebemos o ficheiro com *strings* no formato de vendas, começamos por converter esse ficheiro, num com *structs* Venda para que seja mais fácil dividir os trabalhos entre processos devido ao tamanho fixo dessa estrutura.

Calculamos o número de *forks* que podemos dar, de modo a que seja possível agregar todas as vendas e seja tirado partido da concorrência, e também o número de vendas que cada *filho* será responsável para agregar.

Cada *filho* cria um ficheiro em que para cada artigo escreve uma *struct* Venda, inicializada a zeros (para que o acesso a estas seja direto com *lseek*) e começa a agregar as vendas, adicionando no respetivo artigo, as quantidades e faturação. No final junta-se todos os ficheiros.

Temos uma variável *mud* que calcula as mudanças que houve entre o ficheiro anterior e o agregado. O programa acaba quando não houver mudanças, ou seja, já tudo foi agregado, mandando um sinal ao servidor para ele carregar os artigos mais vendidos.

Extras

Começamos por fazer a compactação de *strings* no M.A. Se o tamanho de *strings* úteis a dividir por o tamanho total for menor que 80%, escreve todas as strings úteis num ficheiro novo e elimina o ficheiro antigo.

Mais tarde, começamos a fazer o caching de preços no agregador. Enquanto este corre guarda num *array* os produtos mais vendidos e depois escreve-os num ficheiro auxiliar que depois será lido pelo servidor.

E por fim conseguimos mudar para um agregador concorrente, como tinha sido descrito anteriormente.

Conclusão

Concluído o trabalho, entendemos que houve uma aprendizagem notória relativamente aos conhecimentos adquiridos nesta Unidade Curricular.

Utilizamos todos os elementos que nos foram lecionados nas aulas práticas, desde a leitura de ficheiros até aos sinais. No decorrer do projeto, foi notória a importância da utilização do que aprendemos na UC.

Foi constantemente necessário chamar a função *lseek*, seja para calcular tamanhos de ficheiros, como para procurar elementos num ficheiro. Foram utilizados alguns *forks*, nomeadamente no agregador para garantir a concorrência como no servidor para garantir que poderiam ser feitas instruções enquanto executava o agregador. Como o agregador lê do *stdin* e imprime no *stdout*, foi necessária a utilização de *dups* e *exec* e os *pipes* são o meio fundamental de comunicação entre servidor/cliente.

Apesar de não ser obrigatório a utilização de sinais, usufruímos deles para conseguir comunicar com o servidor de outra forma se não os *pipes*.

No geral faz-se uma avaliação muito positiva das diversas métricas consideradas e entende-se que a abordagem ao problema foi concisa e direta.