Trabalho prático de Sistemas Operativos
Gestão de Vendas

4 de Maio de 2019

Indicações adicionais

Esta secção destina-se a esclarecer um conjunto de dúvidas que têm vindo a surgir após a publicação do enunciado do trabalho. Aproveita-se ainda para recordar aspectos fundamentais deste trabalho de SO, numa tentativa de se evitarem surpresas durante a avaliação.

- *Para o trabalho prático podemos usar a biblioteca do GLib? Ela seria usada, como por exemplo, para implementar hashes ou arrays?*

  Antes de responder, vamos a umas perguntas...

  Qual é o problema que a Glib (neste caso) resolve? Que problema está a querer evitar? A questão que se podia coloca, de facto, é: se posso usar a Glib, no <módulo/parte do enunciado> para <em vez de fazer assim> usar uma hash table / array <desta forma> ? Isso revela conhecimentos e provavelmente até sugere logo a resposta, desde que se consiga responder às inevitáveis questões de "que alternativas existem?" e "porque não as quero usar?".

  A resposta que dei a esta questão foi mais ou menos assim:

  Não pode, ou seja, não deve (investir o seu tempo para) usar hash tables, arrays dinâmicos ou no geral qualquer receita que tenha aprendido para aplicações "in-core". Este trabalho é sobre ficheiros, é sobre dados "out-of-core". Não pode trazer para memória o conteúdo total dos ficheiros. Não "cabe". Os dados estão em disco, com algumas partes, poucas, em memória. Pense em grande, pense em terabytes de dados persistentes, com programas que correm numa *memória central volátil e bastante mais pequena do que o disco*.

  A essência deste trabalho de SO, é o **"Out-of-core"**. Não é um gestor de stocks, não é para se fazer em mysql ou tecnologia similar, é para se fazer com ficheiros, supostamente porque nesses cabe "tudo" e em memória central ficaria limitado a "poucos" artigos. Aqui não cabe, nem queremos só um processo (porquê?) e não queremos que o utilizador do ponto de vendas possa destruir a nossa "base de dados". Felizmente o enunciado já apresenta uma estrutura de software (código e dados) que facilita muito chegar-se a um bom trabalho, presumivelmente bem melhor/rápido/seguro do que um gestor de stocks à chegada à UMinho[1].

  Portanto, basta usar arrays (os do C, int a[200]...). Não é proibido desde que saiba responder às questões acima, mas também normalmente não será valorizado. Por exemplo, a cache de preços pode ser um array capaz de manter em memória um determinado número, pequeno, de preços de artigos vendidos. Não importa se usa pesquisa sequencial ou uma hash table para procurar o preço do produto que está ser vendido. A cache será sempre *minúscula* em relação ao número de

[1] Outras UCs virão com gestores bem melhores do que este... ainda só estamos no 2º ano.

artigos que estão em ficheiro. O objectivo é comparar os tempos com e sem cache para ver se para um determinado tipo de carga, a estratégia de gestão da cache consegue despachar as vendas em menos tempo. Se optar por usar código que não é seu, vai ter de responder às perguntas acima mais algumas: e um simples array não serve? Consegue quantificar o benefício? Poupa tempo? Quanto? Poupa espaço? Quanto? E não podia economizar isso noutra parte do programa? Desculpem, isto é Sistemas Operativos, espera-se uma visão "sistémica"(hardware + SO + meu software).

Conclusão: não se recomenda que use algo fora do mundo de SO, sobretudo se tiver sido muito bom em aplicações single-process, single-user, in-core, nada corre mal... Aqui tem precisamente o contrário.

- *Então não posso usar funções?*

  A que funções se refere? Hash e seus amigos já vimos acima. Funções definidas no seu código para melhorar a estrutura do seu programa, pode e deve! É proibido apresentar lençóis com um main() de 6 páginas. Pode usar printf no output para os utilizadores finais, i.e. pessoas, pode usar atoi para facilitar a leitura de números, pode e deve usar uma função que lhe permita saber a data+hora e gerar facilmente o nome do ficheiro da agregação. São exemplos de funções que não retiram conhecimento, apenas facilitam o trabalho.

  Readln? Bem, se acha que tem mesmo vantagens então se for o readln que fez no seu Guião 1, pode. Mas cuidado, se se aparecerem dúzias de readln idênticos, passa a ser do domínio publico e dilui-se a valorização. E se o readln não for a melhor solução?

  Fgets e similares? Porquê, qual é o problema que o fgets resolve? Tem a certeza? Será assim tão complicado reconhecer dois inteiros numa linha de texto????

- *Tenho dúvidas no formato de input; aqueles sinais de < e > tem de aparecer sempre? E são redirecionamentos para onde?*

  A ideia de ter uma interface com a aplicação apenas por stdin e stdout foi simplificar o esforço de quem tem de tratar do diálogo com os utilizadores/pessoas mas sobretudo simplificar o processo de teste dos vários programas a desenvolver. Não só a avaliação preliminar no dia da discussão pode ser muito automatizada como os próprios grupos podem/devem evitar testes manuais no teclado, que serão obviamente muito limitados. Escreva num ficheiro uma linha que insere um artigo, repita essa linha 500000 vezes e voilá, facilmente terá um script que lhe carrega meio milhão de artigos para a sua base de dados enquanto vai tomar um café. Até deve colocar esse script no relatório.

  O formato é o mais simples que pode ser, por exemplo

  ```
  letra espaço nome espaço inteiro
  ```

  Na manutenção de artigos podemos ter algo como (note que aqui só apresento o input, a parte que estaria no script de teste; o resultado de cada comando aparece no stdout, que poderá querer redirecionar para um ficheiro de log para mais tarde validar)

  ```
  $ ./ma
  i YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY 100
  n 1 A1234
  p 1 200
  ```

  Ou seja, temos uma letra minúscula no início da linha, um espaço, preços sempre inteiros, não há meta caracteres para ter nomes com espaços, há uma só palavra para cada nome, de tamanho razoável (não superior a 80 caracteres).

1	resultado seria 0 200, muito simples: zero stock, preço 200
1 10	resultado já seria 10, acrescenta ao stock 10 unidades do artigo A1234
1	resultado agora seria 10 200
1 -4	resultado passa a 6, vendeu 4 produtos com número 1

De igual modo, no cliente de vendas (`cv`) podemos ter como input

Se não houver stock para a encomenda total, por exemplo se agora tentar vender 10 artigos com número 1, não fica stock negativo nem deve estourar o programa. O enunciado permitia não vender nenhum artigo ou vender só até ao stock existente. No entanto, na discussão do trabalho os scripts só vão testar se não se perdem entradas ou saídas de stock devido a eventuais bugs ou race conditions. Acrescentam-se 10000 unidades ao stock, fazem-se 10000 vendas concorrentes, no fim o stock desse artigo está a zero, óptimo.

- *Existe o executáve ag, mas como se despoleta a agregação?*

  Pode ser feita arrancando manualmente os comandos ag... Essa iria ser uma pergunta na avaliação! Fazer uma experiência de agregação, guardar o resultado, pedir a um elemento do grupo que repetisse a mesma agregação agora através do servidor de vendas e comparar os resultados.

  No entanto, para uniformizar a avaliação (e também porque estão a surgir opções pouco razoáveis) e porque a agregação será algo que interessa mais à administração do que à pessoa que está no cliente de vendas, sugere-se que seja mais uma opção na manutenção de artigos:

```
$ ./ma
a
```

  A opção de um "a" sozinho numa linha significa agregue AGORA. Isto pede ao servidor de vendas que gere o ficheiro correspondente a esta hora. O mecanismo para ser feito esse pedido entre o `ma` e o `sv` permanece ao critério de cada grupo.

  Note que agregar é criar uma sucessão de ficheiros correspondentes aos totais de vendas até ao instante em que foi solicitada a agregação. O seu programa tem de ser capaz de agregar pelo menos uma vez. Recomenda-se que realize um número de vendas de vários artigos, agregue e verifique se os valores estão correctos. Note ainda que o número de artigos vendidos poder ser muito elevado, no limite todos os artigos teriam vendas... é por isso que faz sentido a agregação concorrente. E, como deve calcular, dá jeito que o resultado da agregação seja legível, ie. seja "impresso" em formato de texto ASCII.

- *Este ano não há sinais?*

  Não são obrigatórios (mas veja a questão seguinte), no entanto, tenha-os usado ou não no seu trabalho, vai sempre ter de estar preparado para responder porquê... Como treino para o teste/exame ou mesmo na própria apresentação, sugere-se (muito) que tenha um caso de comunicação por sinais com o servidor de vendas. São sinais enviados por programas, os seus, não por pessoas.

- *Posso re-arrancar tudo ao fim de cada teste?*

  Claro que não deve fazer isso. Um servidor presta serviços desde que é iniciado por algum/a administrador/a da aplicação até ser voluntariamente encerrado. Vai respondendo aos pedidos, nem sabemos a que carga vai ser sujeito. E se for desligado, ao recomeçar não pode ter perdido nada, tem de voltar rapidamente ao estado em que estava quando foi desligado (para efeito de teste

usar-se-á o comando kill com SIGTERM). Também não faz sentido limpar os ficheiros de dados. Só no final da apresentação terá de os limpar se a avaliação tiver sido feita no computador de um docente, assim como eliminar quaisquer processos que tenha criado.

- *Gostava de ter 20 no trabalho. O que me aconselha a fazer?*

  Há vários conselhos. O principal é que mostre o que aprendeu em SO, não noutras UCs.

  Outro será perguntar a si próprio os porquês das ideias que surgem, garantir que discutiram e perceberam os prós e contras de cada uma. Em particular, assegure-se que percebeu verdadeiramente que está no mundo "out-of-core", o dos dados que não cabem todos em memória central. Pense em concorrência, desempenho, segurança/robustez, simplicidade...

  Outro será pedir que pense na carga com que vão testar cada aspecto do trabalho e automatize/documente esse teste num ou mais scripts de comandos.

  Por último, não se entusiasmem demasiado. O trabalho só vale 1/3 da nota final e não há notas superiores a 20. Mas há 20s!