# FAQ

## Trabalho Prático SO'22

1. **Ao implementar as transformações, nomeadamente, a de comprimir dados para formato gzip é suposto fazêmo-lo através do comando exec como estão nos ficheiros da blackboard ou temos de ler o ficheiro e escrevê-lo noutro com auxílio de uma biblioteca como a zlib.h ?**
   Devem usar a Makefile para compilar os diferentes ficheiros de código fonte e gerar os seus executáveis. A disponibilização do código fonte serve apenas para garantir que os executáveis são compilados de forma compatível com o hardware onde os vão correr.

   No vosso código do trabalho prático devem usar apenas esses executáveis (via chamada ao sistema exec) e nada mais. Ou seja:

   - não podem usar a zlib;
   - não podem "chamar" no vosso código os execs() que estão no código fonte destes programas (transformações);
   - não podem alterar o código fonte dos programas (transformações).

2. **Podemos usar a função printf(...)?**
   A função *printf* pode ser utilizada apenas para debugging e não para resolução do enunciado. A função *printf* resulta na escrita de dados (texto) para o *stdout*, cujo comportamento varia de acordo com o tipo de recurso (i.e., ficheiro, pipe, etc.) associado ao descritor *stdout*. A escrita para estes recursos deverá ser feita exclusivamente com recurso a chamadas ao sistema.

3. **Podemos usar as funções fopen, fwrite, etc., da biblioteca de C?**
   Não. A manipulação de ficheiros deverá ser efetuada única e exclusivamente utilizando as chamadas ao sistema para o efeito.

4. **Clarificação sobre interface do programa cliente e concorrência de pedidos**
   O cliente a ser desenvolvido no trabalho prático corresponde ao programa *sdstore*. Na interface e exemplos de utilização deste programa podem verificar que os argumentos do programa especificam o tipo de pedido a executar (p.ex., status, proc-file).

   Este programa usa o *standard output* para notificar o utilizador sobre o resultado dos pedidos executados. É possível um utilizador executar vários pedidos (chamar várias vezes o programa *sdstore*) mesmo que os anteriores não tenham terminado (p.ex., em segundo plano, ou num novo separador do terminal).

O servidor deve suportar o processamento concorrente de pedidos. Isto, tendo em conta os limites de utilização (instâncias) por transformação (conforme especificado no ficheiro de configuração).

5. **O mesmo cliente pode submeter mais do que 1 pedido? Por exemplo:**

   ***$ ./sdstore proc-file in out nop bcompress ...***
   ***./sdstore proc-file in out encrypt decrypt***
   ***./sdstore status***

   Conforme explicado na resposta à pergunta 4, o *sdstore* é um programa executável que é chamado via o terminal e recebe um conjunto de argumentos (ver enunciado). **Não é suposto o programa *sdstore* ler linhas via *standard input*!**

   Assim sendo, o utilizador pode chamar várias vezes o programa, por exemplo:

   *$ ./sdstore proc-file in out nop bcompress …*
   *$ ./sdstore proc-file in out encrypt decrypt*
   *$ ./sdstore status*

   De notar, no exemplo anterior, que cada invocação do programa só começa depois da anterior terminar (o $ indica uma nova linha do terminal). Este comportamento é idêntico a executar outros programas no terminal, por exemplo:

   *$ ls -l*
   *$ ps aux*
   *$ cat /etc/passwd*

   Notem que, para ter vários programas a executar simultaneamente, o utilizador pode chamar cada programa num separador distinto do terminal ou usar o operador *&* (**este é um operador do terminal e não um argumento do programa!**). Por exemplo, é possível ter várias instâncias do programa a executar simultaneamente com:

   *$ ./sdstore proc-file in out nop bcompress … &*
   *$ ./sdstore proc-file in out encrypt decrypt &*
   *$ ./sdstore status*

6. **Esclarecimento sobre prioridades**

   De forma a simplificar e uniformizar as soluções apresentadas para a funcionalidade avançada de atribuir diferentes prioridades à operação *proc-file*, devem usar a flag "**-p**" para definir a prioridade de uma determinada operação. Esta *flag* é opcional, ou seja, devem assumir a prioridade 0, por omissão, para operações *proc-file* que não usem a *flag*.

Assim sendo, a interface para submeter pedidos de processamento e armazenamento de ficheiros será conforme o exemplo abaixo:

*$ ./sdstore proc-file -p priority samples/file-a outputs/file-a-output bcompress nop gcompress encrypt nop*

Em que **priority** é um inteiro de 0 a 5, conforme indicado no enunciado. Pedidos sem prioridade especificada (equivalente a prioridade 0) podem ser submetidos sem a *flag*:

*$ ./sdstore proc-file samples/file-a outputs/file-a-output bcompress nop gcompress encrypt nop*

7. **O enunciado diz "*para cada operação proc-file deverá ser reportado o número de bytes que foram lidos do ficheiro a ser processado, bem como o número de bytes que foram escritos no ficheiro final após aplicar as transformações*". Ou seja, apenas interessa o número de bytes do ficheiro final, e não os bytes escritos em ficheiros resultantes de passos intermédios no processamento da totalidade do pedido, correto?**

   Certo, apenas queremos o número de bytes lidos do ficheiro inicial e número de bytes escritos no ficheiro final.

   Ainda, cuidado com a criação de ficheiros intermédios no servidor, a qual deve ser evitada conforme refere o enunciado do trabalho.

8. **É necessário que o Cliente envie o ficheiro a ser processado e receba o ficheiro após o processamento via a comunicação com o Servidor?**

   Não. Embora isso seja possível, não é requisito do trabalho. O Cliente envia o nome (caminho) do ficheiro a ser processado e o nome (caminho) do ficheiro resultado. Com essa informação o Servidor pode ler diretamente o ficheiro original e escrever o ficheiro resultado.