Sistemas Operativos

Trabalho Prático

SDStore: Armazenamento Eficiente e Seguro de Ficheiros

Grupo de Sistemas Distribuídos
Universidade do Minho

14 de Março de 2022

Informações gerais

- Cada grupo deve ser constituído por até 3 elementos;
- O trabalho deve ser entregue até às 23:59 de 14 de Maio;
- Deve ser entregue o código fonte e um relatório de até 10 páginas (A4, 11pt) no formato PDF (excluindo capas e anexos), justificando a solução, nomeadamente no que diz respeito à arquitectura de processos e da escolha e uso concreto dos mecanismos de comunicação.
- O trabalho deve ser realizado tendo por base o sistema operativo Linux como ambiente de desenvolvimento e de execução;
- O trabalho deve ser submetido num arquivo Zip com nome `grupo-xx.zip`, em que "xx" deve ser substituído pelo número do grupo de trabalho (p. ex: `grupo-01.zip`);
- A apresentação do trabalho ocorrerá em data a anunciar, previsivelmente entre os dias 2 e 4 de Junho;
- O trabalho representa 50% da classificação final.

Resumo

Pretende-se implementar um serviço que permita aos utilizadores armazenar uma cópia dos seus ficheiros de forma segura e eficiente, poupando espaço de disco. Para tal o serviço disponibilizará funcionalidades de compressão e cifragem dos ficheiros a serem armazenados.

O serviço deverá permitir a submissão de pedidos para processar e armazenar novos ficheiros bem como para recuperar o conteúdo original de ficheiros guardados previamente. Ainda, deverá ser possível consultar as tarefas de processamento de ficheiros a serem efetuadas num dado momento e estatistícas sobre as mesmas.

Programas cliente e servidor

Deverá ser desenvolvido um cliente (programa *sdstore*) que ofereça uma interface com o utilizador via linha de comando. O utilizador poderá agir sobre o servidor através dos argumentos especificados na linha de comando deste cliente. Deverá ser também desenvolvido um servidor (programa *sdstored*), mantendo em memória a informação relevante para suportar as funcionalidades descritas neste enunciado.

O *standard output* deverá ser usado pelo cliente para apresentar o estado do serviço ou o estado de processamento do pedido ("pending", "processing", "concluded"), e pelo servidor para apresentar informação de *debug* que julgue necessária.

Tanto o cliente como o servidor deverão ser escritos em C e comunicar entre si via *pipes com nome*. Na realização deste projecto não deverão ser usadas funções da biblioteca de C para operações sobre ficheiros, salvo para impressão no *standard output*. Da mesma forma não se poderá recorrer à execução de comandos directa ou indirectamente através do interpretador de comandos (p. ex.: *bash* ou `system()`).

Transformações

Existem diferentes tipos de transformações que podem ser aplicadas:

- **bcompress / bdecompress.** Comprime / descomprime dados com o formato bzip.
- **gcompress / gdecompress.** Comprime / descomprime dados com o formato gzip.
- **encrypt / decrypt.** Cifra / decifra dados.
- **nop.** Copia dados sem realizar qualquer transformação.

Para efeito do trabalho, cada uma destas transformações corresponde a um programa executável, cujo código fonte e respetiva *Makefile* estão diponibilizados na pasta `SDStore-transf` no Blackboard. **Não pode** alterar o código fonte desses programas. Note ainda que estes programas fazem uso das ferramentas de linha de comandos *gzip*, *bzip2* e *ccrypt*. Assegure-se que estas ferramentas estão instaladas[1] e que são executáveis sem necessidade de especificação do seu caminho (*path*) no sistema de ficheiros.

Todos os programas recebem o conteúdo do ficheiro através do *standard input* e emitem o conteúdo transformado para o *standard output*. Deve testar os mesmos de forma independente para garantir que estão a funcionar corretamente. Por exemplo, ao executar:

```
$ ./bcompress < file-a > file-a-comp
$ ./bdecompress < file-a-comp > file-a-original
```

O primeiro comando deve comprimir o ficheiro *file-a* e guardar o resultado em *file-a-comp*. O segundo comando deve descomprimir o ficheiro *file-a-comp* e guardar o resultado em *file-a-original*. O conteúdo dos ficheiros *file-a* e *file-a-original* devem ser idênticos. Deve conseguir realizar experiências análogas para as restantes transformações.

Funcionalidades Básicas (15 valores)

O serviço deverá suportar as seguintes funcionalidades básicas:

- O programa servidor recebe dois argumentos pela linha de comando. O primeiro corresponde ao caminho para um ficheiro de configuração que é composto por uma sequência de linhas de texto, uma por tipo de transformação, contendo: identificador da transformação (para simplificar, o mesmo pode corresponder ao nome do ficheiro executável que a implementa); número máximo de instâncias de uma certa transformação que podem executar concorrentemente num determinado período de tempo. Segue-se um exemplo do conteúdo do ficheiro:

  ```
  nop 3
  bcompress 4
  bdecompress 4
  gcompress 2
  gdecompress 2
  encrypt 2
  decrypt 2
  ```

  O segundo argumento corresponde ao caminho para a pasta onde os executáveis das transformações estão guardados.

[1] Na distribuição Ubuntu, as ferramentas *gzip* e *bzip2* já devem ser suportadas pela instalação base do sistema operativo. A ferramenta *ccrypt* pode ser instalada com o comando *sudo apt install ccrypt*.

- Cada cliente solicita o processamento e armazenamento de um ficheiro (opção `proc-file`) através da aplicação no servidor de uma sequência de transformações. Para tal, o cliente passa como argumentos de linha de comando:

  1. O caminho do ficheiro a ser processado;
  2. O caminho onde o serviço deve guardar a nova versão do ficheiro após aplicar transformações;
  3. Uma sequência de identificadores de transformações a aplicar.

- O utilizador é informado (via *standard output* do cliente) caso o pedido tenha ficado pendente, quando entra em processamento, e quando é concluído. O comando termina quando o resultado (ficheiro processado) está disponível.
- O servidor deve suportar o processamento concorrente de pedidos. Quando um pedido começa a executar, todas as transformações a serem usadas pelo mesmo devem ser identificadas como estando em utilização. O processamento de cada pedido não poderá ser iniciado enquanto, para alguma das transformações necessárias para a sua execução, o número de instâncias a correr esteja no limite máximo (conforme especificado no ficheiro de configuração).
- O cliente pode consultar (comando `status`), a qualquer instante, o estado de funcionamento do servidor. Nomeadamente, os pedidos de processamento em execução, bem como, o estado de utilização das transformações.

Funcionalidades Avançadas (5 valores)

A avalição do trabalho prático será valorizada pelas seguintes funcionalidades avançadas:

- Quando um pedido termina, deverá ser reportado ao cliente o número de bytes recebidos e o número de bytes produzidos por operação. Ou seja, 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.
- Se receber o sinal `SIGTERM`, o servidor deve terminar de forma graciosa, deixando primeiro terminar os pedidos em processamento ou pendentes, mas rejeitando a submissão de novos pedidos.
- Assuma que as operações `proc-file` podem ter diferentes prioridades. O servidor deverá dar prevalência a pedidos com maior prioridade. Para realizar esta funcionalidade, cada operação deve ser acompanhada da sua prioridade (identificadas como inteiros de 0 a 5, em que 5 atribui prioridade máxima).

Interface e Modo de Utilização

Este serviço deverá ser usado do seguinte modo:

- submeter pedidos de processamento e armazenamento de ficheiros, conforme o exemplo abaixo:

  ```
  $ ./sdstore proc-file <priority> samples/file-a outputs/file-a-output bcompress nop gcompress encrypt nop
  pending
  processing
  concluded (bytes-input: 2048, bytes-output: 1024)
  ```

  Note que:

  - o argumento `priority` bem como a notificação de estatísticas (bytes de *input* e *output*) só são necessários para as funcionalidades avançadas;
  - para recuperar o conteúdo original do ficheiro *file-a*, a operação `proc-file` deve invocar as transformações de compressão e cifra pela ordem inversa.

    ```
    $ ./sdstore proc-file <priority> outputs/file-a-output dir/file-a decrypt gdecompress nop bdecompress
    pending
    processing
    concluded (bytes-input: 1024, bytes-output: 2048)
    ```

  - a transformação `nop` é especial, uma vez que é idempotente (i.e., pode ser aplicada em qualquer ordem e entre quaisquer transformações sem modificar o resultado esperado).

- obter o estado de funcionamento do servidor, conforme o exemplo abaixo:

```
$ ./sdstore status
task #3: proc-file 0 /home/user/samples/file-c file-c-output nop bcompress
task #5: proc-file 1 samples/file-a file-a-output bcompress nop gcompress encrypt nop
task #8: proc-file 1 file-b-output path/to/dir/new-file-b decrypt gdecompress
transf nop: 3/3 (running/max)
transf bcompress: 2/4 (running/max)
transf bdecompress: 1/4 (running/max)
transf gcompress: 1/2 (running/max)
transf gdecompress: 1/2 (running/max)
transf encrypt: 1/2 (running/max)
transf decrypt: 1/2 (running/max)
```

- obter informação de utilização do programa cliente:

```
$ ./sdstore
./sdstore status
./sdstore proc-file priority input-filename output-filename transformation-id-1 transformation-id-2 ...
```

- executar o servidor:

```
$ ./sdstored config-filename transformations-folder
```

  Por exemplo,

```
$ ./sdstored etc/sdstored.conf bin/sdstore-transformations
```

  No exemplo anterior os programas executáveis que implementam as transformações devem ser colocados manualmente na pasta *bin/sdstore-transformations*.

Observações

Para a implementação dos dois conjuntos de funcionalidades (básicas e avançadas), deve considerar as seguintes observações:

- O servidor deve evitar a criação de ficheiros temporários;
- Assuma que é da responsabilidade do cliente aplicar as transformações pela ordem correta ao processar o conteúdo de ficheiros geridos pelo serviço. Por exemplo, se um dado ficheiro, numa operação `proc-file`, for comprimido e cifrado (por esta ordem), a operação `proc-file` inversa, com o objetivo de reverter o ficheiro processado para o seu conteúdo original, deverá usar as transformações decifrar e descomprimir (por esta ordem). O mesmo se aplica a usar o mesmo algoritmo para comprimir e descomprimir.
- A utilização de sinais é apenas obrigatória para a funcionalidade de terminação graciosa do servidor. Todos os restantes pontos poderão ser resolvidos sem ter que recorrer a este mecanismo.

Makefile

Tenha em conta que a Makefile que se apresenta deverá ser usada como ponto de partida mas poderá ter necessidade de a adaptar de modo a satisfazer, por exemplo, outras dependências do seu código-fonte. Em todo o caso, deverá manter sempre os objectivos (*targets*) especificados: `all`, `server`, `client` e `clean`. Não esqueça que por convenção a indentação de uma Makefile é especificada com uma tabulação (*tab*) no início da linha (nunca com espaços em branco).

```
all: server client

server: bin/sdstored

client: bin/sdstore
```

```
bin/sdstored: obj/sdstored.o
	gcc -g obj/sdstored.o -o bin/sdstored

obj/sdstored.o: src/sdstored.c
	gcc -Wall -g -c src/sdstored.c obj/sdstored.o

bin/sdstore: obj/sdstore.o
	gcc -g obj/sdstore.o -o bin/sdstore

obj/sdstore.o: src/sdstore.c
	gcc -Wall -g -c src/sdstore.c obj/sdstore.o

clean:
	rm obj/* tmp/* bin/{sdstore,sdstored}
```