

Universidade do Minho

UNIVERSIDADE DO MINHO

LICENCIATURA EM CIÊNCIAS DA COMPUTAÇÃO

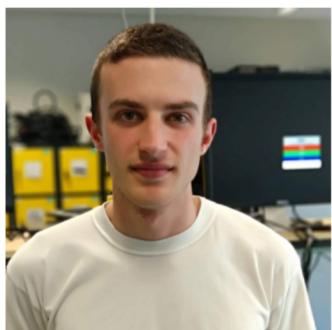
Sistemas Operativos - Trabalho Prático
Grupo nº16

David José de Sousa Machado
(A91665)

Miguel Ângelo Alves de Freitas
(A91635)

Tomás Vaz de Carvalho Campinho
(A91668)

29 de maio de 2022



Conteúdo

1	Introdução e principais desafios	3
2	Funcionalidades disponíveis	4
2.1	Cliente	4
2.1.1	Status	4
2.1.2	proc-file	5
2.2	Cliente	5
3	Arquitetura	6
3.1	Cliente-to-server communication	6
3.2	Parsing	7
3.3	Priority line	7
3.4	Processo	7
3.5	Queue	7
3.6	Processamento	8
3.6.1	Executar apenas 1 transformação	8
3.6.2	Executar processo de várias transformações	8
4	Conclusão	10

Capítulo 1

Introdução e principais desafios

O objetivo deste relatório é explicar as nossas escolhas/soluções para os problemas apresentados neste trabalho, no âmbito deste curso. Para esta tarefa tivemos que implementar uma Arquitetura Cliente/Servidor capaz de lidar com várias solicitações simultâneas para 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.

Ainda, deverá ser possível consultar as tarefas de processamento de ficheiros a serem efetuadas num dado momento.

Esta implementação é constituída por um servidor e por um cliente que deverão comunicar por pipes com nome, um envia comandos para o servidor, e o outro envia o output dos comandos do servidor para o cliente.

Capítulo 2

Funcionalidades disponíveis

2.1 Cliente

O cliente é o programa que aceita a entrada do usuário e a envia para o servidor a ser processado.

O cliente tem duas funcionalidades: status e proc-file.

No lado do servidor, garantimos cuidadosamente que qualquer entrada incorreta - que não seguiu o formato desejado - não vai para o servidor, ao invés disso nós analisamos de que forma está com defeito e enviamos ao cliente um erro apropriado mensagem. Todos os casos em que considerar:

- Um comando não é uma *proc – file* nem um *status*.
- Um comando de *status* tem argumentos extras.
- Um comando de *proc – file* não tem entrada, saída e pelo menos uma transformação.

Na ausência de argumentos o cliente recebe uma mensagem a explicar a interface da aplicação.

2.1.1 Status

Esta tarefa é executada usando

```
1 $ ./sdstored status
```

e mostra o estado atual do servidor (quais tarefas estão a ser executadas e a atual alocação de recursos):

```
1 $ ./sdstored status
2 transf nop: 0/3 (running/max)
3 transf bcompress: 0/4 (running/max)
4 transf bdecompress: 0/4 (running/max)
5 transf gcompress: 0/2 (running/max)
6 transf gdecompress: 0/2 (running/max)
7 transf encrypt: 0/2 (running/max)
8 transf decrypt: 0/2 (running/max)
```

2.1.2 proc-file

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.

Esta tarefa é executada usando

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

e gera o resultado da execução do arquivo de entrada por meio de um ou várias transformações. Ele também envia o estado da tarefa para a *bash*:

```
1 $ ./sdstore proc-file <priority> ../samples/file-a ../samples/file-a-output  
      bcompress nop gcompress encrypt nop  
2 pending  
3 processing  
4 concluded (bytes-input: 2048, bytes-output: 1024)
```

2.2 Cliente

O Servidor deve ser executado, antes de qualquer cliente, e é capaz de receber solicitações de processamento, processá-las e enviar notificação ao usuário. É executado com:

```
1 $ ./sdstored .. /etc/sdstored.conf .. /bin/sdstore-transformations
```

Funciona até ser fechado manualmente. Termina graciosamente ao receber um SIGINT (principalmente por meio de um CTRL+C).

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 correntemente num determinado período de tempo

Segue-se um exemplo do conteúdo do ficheiro:

```
1 nop 3  
2 bcompress 4  
3 bdecompress 4  
4 gcompress 2  
5 gdecompress 2  
6 encrypt 2  
7 decrypt 2
```

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

Capítulo 3

Arquitetura

3.1 Cliente-to-server communication

Quando executado, o servidor cria um pipe nomeado para comunicação cliente-servidor. Por causa do protocolo de solicitação, um único pipe pode suportar comunicações para todos os clientes.

Para habilitar a comunicação servidor-cliente, o cliente cria um pipe nomeado nomeado após seu próprio PID.

O cliente então envia ao servidor uma estrutura de dados contendo seu PID (portanto mais tarde o servidor pode procurar o dono da requisição e enviar a mensagem apropriada), um array com os argumentos que foi dado na execução e o número desses argumentos também. Em seguida, ele fecha a extremidade do pipe e começa a ler (bloquear) do pipe servidor-cliente que tinha anteriormente.

No caso de um pedido de transformação, após a leitura da estrutura do pedido enviado pelo cliente, o servidor abre o pipe com o nome do PID do cliente e escreve a mensagem 'pendente', que o cliente lê e imprime em seu stdout, retorna ao seu estado de leitura de bloqueio anterior, isso inicia a fase de processamento de um pedido de aplicação de transformações que explicaremos detalhadamente mais adiante.

3.2 Parsing

Após receber a solicitação e informar o cliente sobre sua situação de pendência, o servidor começa a trabalhar no pedido.

Ele extrai os nomes das transformações solicitadas. Em seguida, verifica para certificar-se de que são válidos (de acordo com o arquivo de configuração do servidor): se não estiverem, ele envia uma mensagem de volta para o cliente e volta a ler de para o pipe cliente-servidor, como explicamos anteriormente.

No entanto, se forem válidos, procede-se à sua fusão; por outras palavras, cria uma nova estrutura que contém tuplos de um nome de uma transformação e o número de vezes que é usado. Essa estrutura é crucial para processar os pedidos em uma fação abstrata, pois podemos nos por em múltiplas verificações e alocação e desalocação de recursos para solicitações de aplicar transformações únicas e múltiplas.

Assim, de acordo com a menção logística acima, esta lista de tuplos é então usado para verificar se é possível processar a solicitação: se numa solicitação uma transformação é usado mais vezes do que a configuração do servidor permite, essa solicitação é considerado impossível, descartado e o cliente notificado (como mencionamos acima, também).

3.3 Priority line

Para a funcionalidade de prioridades nós fizemos com que o programa ao solicitar um novo processo(de acordo com o arquivo de configuração do servidor) insere de forma ordenada de acordo com a prioridade, e para isso criamos uma lista ligada para conseguir fazer com que os processos sejam inseridos de forma ordenada.

3.4 Processo

Se a solicitação passar em todas as verificações anteriores, ela será representada em um processo de estrutura. Essa estrutura contém todos os dados sobre uma solicitação que será processado: o PID do cliente (usado para procurar o nome do qual a comunicação servidor-cliente acontecerá), o número de transformações que será necessário (útil na hora de verificar a disponibilidade do servidor para um processo e alocar e desalocar recursos rapidamente, sempre de acordo com o arquivo de configuração), os arquivos de entrada e saída e a lista de tuplos. Essa estrutura é então colocado em uma fila e aguarda o processamento.

3.5 Queue

A fila é uma matriz multidimensional. Pensamos nisso como uma matriz com 16 linhas e 1024 colunas. Um processo é armazenado nele de acordo com quantas transformações que usa. Um processo com n transformações será anexado à linha $n-1$; se tiver 16 transformações ou mais, ele será armazenado na linha 15. A razão pela qual este método foi escolhido será discutido na próxima secção.

3.6 Processamento

Após analisar o pedido e colocar sua representação de processo em uma fila, o servidor inicia o processamento da referida fila.

O servidor possui um contador que armazena o número de slots de transformações disponíveis, ou seja, os slots de transformações não são reservados por outro processo.

Como ele conhece o número máximo de slots de transformações que pode alocar naquele momento, digamos, n , ele precisa apenas processar a fila das linhas 0 a $n-1$. Dessa forma, não precisamos percorrer a restante da fila e validar esses processos, pois sabemos desde o início que nunca haverá slots de transformações suficientes disponíveis.

Tendo estabelecido o intervalo pelo qual percorrer a fila, o servidor começa a percorrer a linha 0 e valida e processa cada processo. Ele faz isso até que a fila esteja vazia ou não haja mais transformações disponíveis.

Quando a fila está vazia, o servidor aguarda uma leitura de bloqueio (do pipe cliente-servidor) até que outro cliente envie outra solicitação.

1. verifica a disponibilidade da transformação: o servidor verifica se todas as transformações que o processo irá necessitar estão disponíveis (se não estiverem, o processo permanece na fila)
2. são reservados as transformações que o processo necessita;
3. uma mensagem é enviada ao cliente informando que o processo será iniciado em processamento;
4. a execução do processo é iniciada

3.6.1 Executar apenas 1 transformação

A logística para esta execução foi bastante simples. Como só tivemos a necessidade de realizar uma aplicação da transformação, apenas tivemos que abrir tanto a entrada e saída, duplique o $STDIN$ e o $STDOUT$ para apontar para seus descritores de arquivo, respetivamente, feche-os e execute a transformação usando a concatenação da pasta fornecida na execução do servidor e o binário da transformação fornecido pelo arquivo de configuração e obtido de seu nome que foi usado pelo usuário para invocá-lo.

3.6.2 Executar processo de várias transformações

Esta parte era um pouco mais complicadas. Para fazer a aplicação de várias transformações precisávamos executar cada execução em seu próprio processo e comunicar entre as transformações usando pipes anónimos para criar um pipeline de streaming para realizar essa tarefa de maneira ideal. Digamos que precisamos realizar N transformações, isso significa que temos que usar $N-1$ pipes, onde precisamos codificar apenas 3 situações diferentes:

1. **a primeira transformação** lê a partir do arquivo de entrada (usando abertura e duplicação do descritor de arquivo como acima) e grava na extremidade de entrada do primeiro pipe (usando abertura e duplicação do descritor de arquivo como acima)

2. **a ultima transformação** lê a partir da extremidade de gravação do último pipe e grava no arquivo de saída (usando abertura e duplicação do descritor de arquivo como acima)
3. **todos os outros** no meio, se necessário (3 ou mais transformações), lê o pipe anterior e escreve para o próximo

Isso foi feito com uma matriz de pipes usando um loop for de 0 a $N - 1$ transformações e lembre-se de que as extremidades de pipes não usadas em cada etapa precisavam ser fechadas antes de qualquer execução da transformação (usando execl da mesma forma).

Capítulo 4

Conclusão

Como é possível constatar, o nosso trabalho possui todas as funcionalidades pedidas a funcionar perfeitamente, para além da funcionalidade avançada. A nível geral, e tendo em conta o que foi explicado nos capítulos anteriores, como grupo achamos que todos os objetivos foram cumpridos e apesar das dificuldades que fomos encontrando o grupo conseguiu superar, sempre com um olhar crítico e a pensar no próximo passo. Acreditamos que respondemos de forma correta ao problema apresentado pela equipa docente da disciplina.

Este trabalho está longe de ser perfeito, ignoramos o uso de memória dinâmica nas nossas estruturas ao longo do projeto. Isso significa que haverá sempre algum limite rígido para o que o servidor e os clientes podem fazer, mas achamos que vale a pena neste caso, pois reduz significativamente a complexidade e o tamanho do código e nos permite focar nos problemas essenciais que sentimos que eram mais importantes para este curso. Há também, com certeza, muitas outras otimizações que podem ser feitas em termos de memória e desempenho, como implementar algumas das nossas estruturas como mapas ou tabelas de hash, árvores binárias, min-heaps (para escolher processos de forma eficiente em vários filtros logísticos).