

Universidade do Minho

Licenciatura em Engenharia Informática

Sistemas Operativos  
Trabalho Prático  
Grupo 115

Duarte Parente (A95844)  
Gonçalo Pereira (A96849)  
Diogo Silva (A96277)

Ano Letivo 2021/2022

# 1 Introdução

Este projeto consistiu na implementação de um serviço, com um formato servidor-cliente, que permitisse aos utilizadores (clientes) armazenar uma cópia dos seus ficheiros de forma segura e eficiente, através de funcionalidades de compressão e cifragem dos mesmos. Toda a interação é feita através de pedidos do cliente para o servidor, com a particularidade de poder ainda obter informação acerca do estado do servidor nesse mesmo instante. Era ainda essencial que o projeto explorasse a concorrência de pedidos de forma a corresponder com um dos principais objetivos da UC, assim como do próprio trabalho prático.

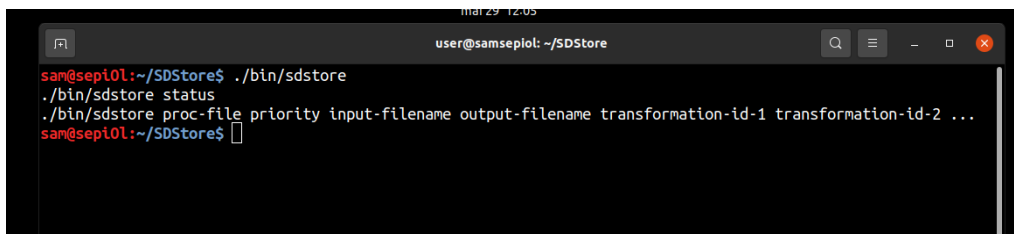
Para além de existir a restrição do uso de funções de bibliotecas para qualquer tipo de operação sobre ficheiros, o cliente e o servidor teriam de comunicar entre si usando pipes com nome.

## 2 Funcionalidades

Existem 3 formas distintas de interagir com o programa.

### 2.1 Obtenção de informação

De forma a pedir informação acerca da utilização do programa cliente pode ser efetuado o seguinte comando: `./bin/sdstore`.

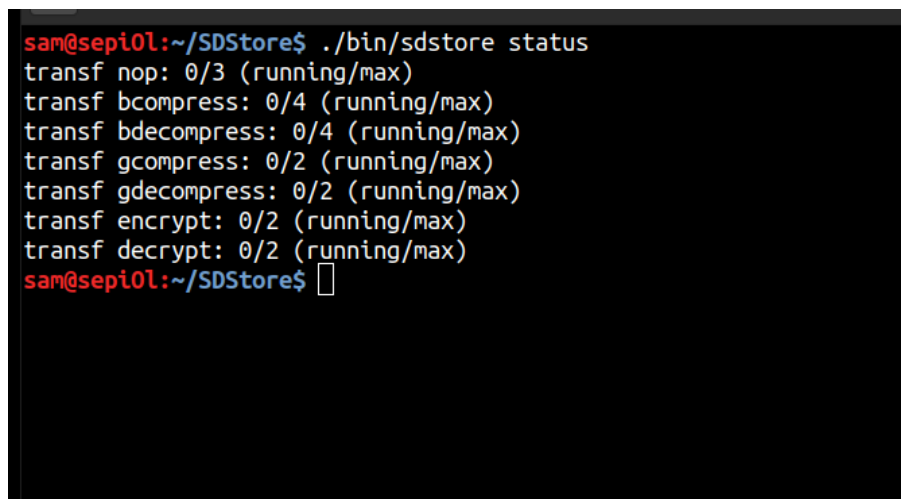
A terminal window titled 'user@samsepiol: ~/SDstore' with a search icon, menu icon, and window control buttons. The terminal shows the command `./bin/sdstore` being executed, which outputs `./bin/sdstore status` and `./bin/sdstore proc-file priority input-filename output-filename transformation-id-1 transformation-id-2 ...`. The prompt `sam@sepi01:~/SDstore$` is visible at the end of the output.

```
user@samsepiol: ~/SDstore
sam@sepi01:~/SDstore$ ./bin/sdstore
./bin/sdstore status
./bin/sdstore proc-file priority input-filename output-filename transformation-id-1 transformation-id-2 ...
sam@sepi01:~/SDstore$
```

Fig. 1: Output do comando de obtenção de informação.

### 2.2 Obtenção do estado do servidor

Em qualquer momento da execução do programa poderá ser consultada a informação acerca do estado do servidor, isto é, os pedidos de processamento em execução, bem como, o estado de utilização das transformações. O comando a usar é `./bin/sdstore status`.

A terminal window showing the command `./bin/sdstore status` being executed. The output lists various transformation types and their counts: `transf nop: 0/3 (running/max)`, `transf bcompress: 0/4 (running/max)`, `transf bdecompress: 0/4 (running/max)`, `transf gcompress: 0/2 (running/max)`, `transf gdecompress: 0/2 (running/max)`, `transf encrypt: 0/2 (running/max)`, and `transf decrypt: 0/2 (running/max)`. The prompt `sam@sepi01:~/SDstore$` is visible at the end of the output.

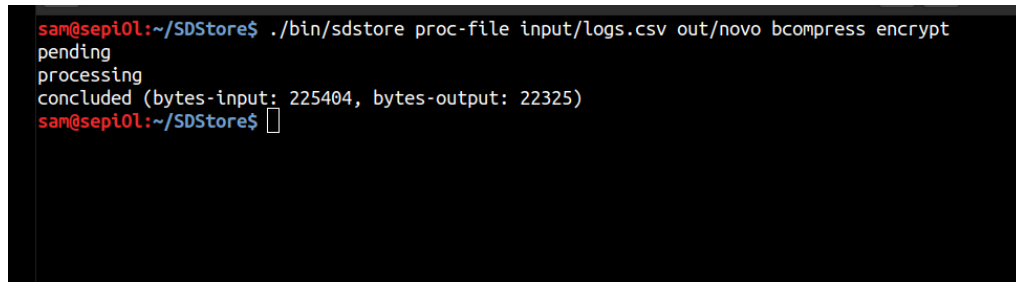
```
sam@sepi01:~/SDstore$ ./bin/sdstore status
transf nop: 0/3 (running/max)
transf bcompress: 0/4 (running/max)
transf bdecompress: 0/4 (running/max)
transf gcompress: 0/2 (running/max)
transf gdecompress: 0/2 (running/max)
transf encrypt: 0/2 (running/max)
transf decrypt: 0/2 (running/max)
sam@sepi01:~/SDstore$
```

Fig. 2: Output do comando de obtenção do estado do servidor.

De notar que neste exemplo não há qualquer pedido pendente dentro do servidor, e por isso todas as transformações se encontram a 0.

## 2.3 Efetuar pedidos de transformações

O pedido para a transformação de um ficheiro é através da opção *proc-file*, onde é passado o caminho para os ficheiros de input e output e o conjunto de todas as transformações a aplicar aos ficheiros. De notar que a execução destes pedidos será feita concorrentemente, tendo em atenção os limites para o número de execuções concorrentes de cada transformação.

A terminal window with a black background and red and white text. The prompt is 'sam@sepi01:~/SDStore\$'. The command entered is './bin/sdstore proc-file input/logs.csv out/novo bcompress encrypt'. The output shows the status of the transformation: 'pending', 'processing', and 'concluded (bytes-input: 225404, bytes-output: 22325)'. The prompt returns to 'sam@sepi01:~/SDStore\$' with a cursor.

```
sam@sepi01:~/SDStore$ ./bin/sdstore proc-file input/logs.csv out/novo bcompress encrypt
pending
processing
concluded (bytes-input: 225404, bytes-output: 22325)
sam@sepi01:~/SDStore$
```

Fig. 3: Exemplo de um pedido efetuado.

## 3 Estrutura de implementação

### 3.1 Cliente

O cliente comunica com o servidor através do pipe principal com o nome *notebook*. Este pipe é o responsável por receber os pedidos de todos os clientes, assim como o nome do pipe responsável pela comunicação servidor-cliente. Dada a necessidade de enviar informação de retorno para cada cliente em específico, foi necessário arranjar uma forma de criar canais específicos para cada um.

Para responder a esta necessidade optamos por utilizar um identificador único associado a cada cliente. O mais óbvio foi escolher o próprio pid do processo, ficando o pipe com o nome no seguinte formato **"task-pid"**.

Sendo assim a única preocupação do cliente é enviar o pedido ao servidor, e ficar à espera das mensagens de retorno por parte do mesmo, nomeadamente a informação sobre o estado de execução do pedido.

### 3.2 Servidor

O servidor é responsável pela criação do pipe principal que faz a comunicação clientes-servidor. Tal como referido anteriormente, este pipe é responsável por receber os pedidos efetuados pelos diversos clientes, assim como os pipes associados a cada um.

Os pedidos são guardados numa estrutura *Task\_Controller* que representa um array de pedidos. Por sua vez, um pedido é também representado por uma estrutura *Task*. Esta estrutura contém toda a informação necessária para a execução das funcionalidades propostas.

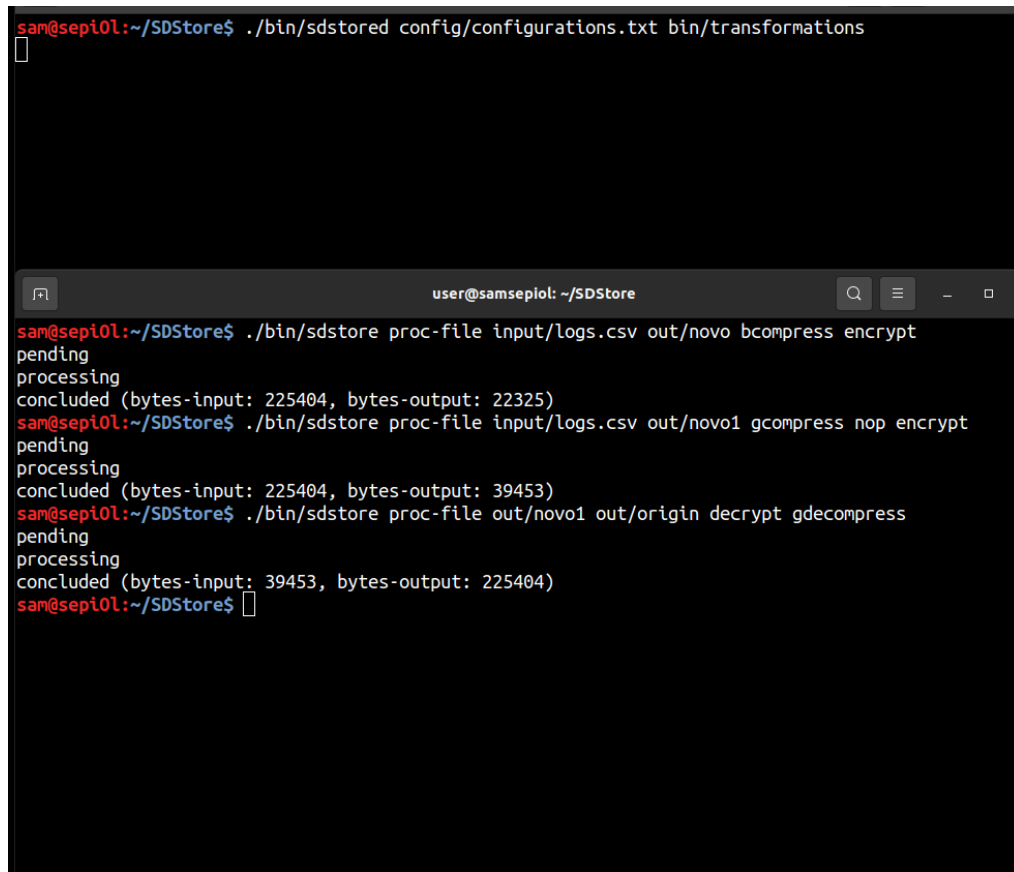
```
typedef struct {
    char *fifo;
    char *arguments;
    int task_id;
    int priority;
    char *input;
    char *output;
    char **transformations;
    int transformations_counter;
    int state;
} *Task;
```

**Fig. 4:** Estrutura que representa um pedido.

A estrutura de controlo dos pedidos funciona como uma fila de espera, e esse controlo é feito através do campo "state". O estado a 1 representa o processo bloqueado, quando não é possível a sua execução devido ao limite do número de execuções concorrentes. O estado a 2 representa um pedido pronto a ser executado, enquanto que o estado a 3 dita que o pedido já foi executado.

O servidor mantém-se sempre em execução até lhe ser ordenada a sua paragem, através de sinais, SIGINT ou SIGTERM. Nenhum cliente tem possibilidade de ordenar o fecho do servidor a partir de um possível comando.

## 4 Programa em execução



```
sam@sepi01:~/SDStore$ ./bin/sdstored config/configurations.txt bin/transformations
[ ]

user@samsepi01: ~/SDStore
sam@sepi01:~/SDStore$ ./bin/sdstore proc-file input/logs.csv out/novo bcompress encrypt
pending
processing
concluded (bytes-input: 225404, bytes-output: 22325)
sam@sepi01:~/SDStore$ ./bin/sdstore proc-file input/logs.csv out/novo1 gcompress nop encrypt
pending
processing
concluded (bytes-input: 225404, bytes-output: 39453)
sam@sepi01:~/SDStore$ ./bin/sdstore proc-file out/novo1 out/origin decrypt gdecompress
pending
processing
concluded (bytes-input: 39453, bytes-output: 225404)
sam@sepi01:~/SDStore$ [ ]
```

**Fig. 5:** Programa em execução.

Neste exemplo o primeiro terminal é usado para suportar o servidor que terá de se manter sempre em execução até ao término do programa.

## 5 Conclusão

De forma geral este projeto revelou-se um bom desafio para a aplicação dos conceitos abordados nas aulas, nomeadamente, concorrência, criação/eliminação de processos, duplicação de descritores, criação de pipes anónimos e com nome, aplicação de sinais, redirecionamento e utilização de system calls. A resolução dos guiões práticos revelou-se um bom ponto de partida para conseguir corresponder corretamente às funcionalidades propostas.