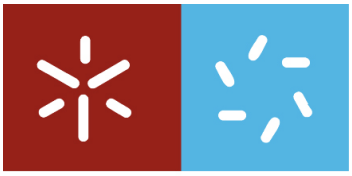

Universidade do Minho

Universidade do Minho

Licenciatura em Ciências da Computação

SO - Trabalho Prático

Grupo nº37

Simão Pedro Batista Caridade Quintela
(A97444)

Pedro Alexandre Silva Gomes
(A91647)

Hugo Filipe de Sá Rocha
(A96463)

29 de maio de 2022


Conteúdo

Capítulo 1

Introdução

Este relatório descreve o desenvolvimento do projeto prático da Unidade Currícular de Sistemas Operativos, inserida no 2ºano da Licenciatura em Ciências da Computação da Universidade do Minho.

Este trabalho consiste no desenvolver de um serviço que permite aos utilizadores armazenar uma cópia dos seus ficheiros de forma segura e eficiente, tendo como objetivo também a poupança de espaço de disco. O serviço possui funcionalidades de compressão e cifragem dos ficheiros a serem armazenados. Permite também a submissão de pedidos para processar e armazenar novos ficheiros, bem como para recuperar o conteúdo original de ficheiros guardados previamente. É possível consultar as tarefas de processamento de ficheiros a serem efetuadas num dado momento e as estatistícas sobre as mesmas. Para que a realização do referido fosse possível, utilizamos os conhecimentos da linguagem *C*, e todos os lecionados e adquiridos ao longo do semestre nesta UC. O trabalho foi realizado em ambiente **Linux**, tal como pedido.

Capítulo 2

Estrututa do projeto

Desenvolvemos um cliente (programa *sdstore*) que dispõe de uma interface com o utilizador via linha de comando. O utilizador age sobre o servidor através dos argumentos especificados na linha de comando deste cliente. Desenvolvemos também um servidor (programa *sdstored*), mantendo em memória a informação relevante para suportar as funcionalidades pedidas. O *standard output* é usado pelo cliente para apresentar o estado do serviço ou o estado de processamento do pedido **(“pending”, “processing”, "concluded")**, e pelo servidor para apresentar a informação de *debug* necessária. Tanto o cliente como o servidor foram escritos em *C* e comunicam entre si via *pipes com nome*. Na realização deste projecto não usamos funções da biblioteca de *C* para operações sobre ficheiros, como pedido, salvo para impressão no *standard output*. Não executamos, em nenhum caso, comandos directa ou indirectamente através do interpretador de comandos .

2.1 Estruturas de dados utilizadas

Neste projeto sentimos a necessidade de utilizar estruturas de dados para guardar a informação dos diferentes pedidos feitos por clientes, bem como para guardar informação acerca da configuração do servidor. Tais estruturas são:

```
typedef struct config_file {
        char* transformation;
        int current_num_transf;
        int max_exec_transf;
} config_file;

config_file conf_file[7];
```

Esta estrutura de dados foi criada para guardar informação acerca da configuração do servidor, nomeadamente da informação presente no **ficheiro de configuração**. Conseguimos ver na estrutura que temos o campo **transformation** que representa uma transformação, o campo **current_num_transformation** que representa o número de transformações que estão a correr no servidor do tipo **transformation** e, por fim, o campo **max_exec_transf** que representa o número máximo de transformações que podem estar a correr, do respetivo tipo. Por fim, como se pode ver abaixo da estrutura, a inicialização da mesma é feita na forma de um array de 7 posições(dado que temos 7 transformações) em que cada elemento do array é do tipo **config_file**.

```
typedef struct tasks_running {
        int task_num;
        char pid[10];
        int pid_fork;
        char line[128];
        char* line_splitted[20];
        int in_process;
        int fixed_args;
        int num_args;
        int priority;
        struct tasks_running *prox_task;
} *tasks_running;
tasks_running tasks = NULL;
```

A nossa segunda, e última, estrutura de dados utilizada foi pensada para resolver o problema de armazenamento de tarefas vindas de diferentes clientes. A solução encontrada foi representar todas as tarefas numa lista ligada, em que cada bloco tem informação acerca de **número da tarefa**, **pid do cliente que enviou a task**, **pid do processo que manda executar a task**, **prioridade** da mesma, **estado de processamento**, etc..

2.2 Transformações

Neste projeto foram apresentadas 7 transformações a serem utilizadas:

- **bcompress** - Comprime dados com o formato bzip.
- **bdecompress** - Descomprime dados com o formato bzip.
- **gcompress** - Comprime dados com o formato gzip.
- **gdecompress** - Descomprime dados com o formato bzip.

- **encrypt** - Cifra dados
- **decrypt** - Decifra dados
- **nop** - Copia dados sem realizar qualquer transformação

Capítulo 3

Arquitetura do Programa

3.1 Comunicação Cliente-Servidor

A comunicação entre cliente e servidor é feita através de pipes com nome em que são usados, por cliente, 2 pipes. O primeiro é comum a todos os clientes e tem o nome de **main_pipe**. Este serve para os clientes enviarem a tarefa que querem ver realizada. O segundo pipe tem o nome diferente para todo o cliente, no entanto, apresenta o mesmo propósito, receção de informação aquando do fim da execução da tarefa. O nome associado a estes pipes é o **pid do processo** do respetivo cliente.

O cliente quando submete o seu pedido escreve no **main_pipe** a linha que quer ser executada e o pid do seu processo, para posteriormente, o servidor dar parse a essa linha recebida através do pipe e guardar toda a informação na estrutura de tasks apresentada previamente.

O servidor, após receber e guardar a tarefa, escreve no pipe com o nome do pid do cliente **Pending...**, quando começa a execução da task escreve, no mesmo pipe, **Processing...** e, por fim, quando termina a execução da mesma escreve **Concluded**, acrescentando ainda informação acerca do tamanho inicial e final do ficheiro de input e output.

3.2 Pipeline

Para executar os diferentes tipos de transformações, tivemos a necessidade de criar uma *pipeline* para conseguirmos redirecionar inputs e outputs. Este processo foi realizado através de *pipes anónimos*, *dups* e *execs*. Os *pipes anónimos* foram úteis na medida em que, quando temos uma sequência de transformações, através destes conseguimos enviar

informação para a transformação seguinte a executar. Os *dups* foram úteis na cópia de descritores de ficheiros para podermos, por exemplo, colocar o *stdin* a apontar para um ficheiro de texto. Para finalizar, os *execs* serviram para podermos mandar executar as diferentes transformações solicitadas em qualquer task.

3.3 Signals

Usamos também *signals*, nomeadamente o *SIGCHLD*. Este *signal* é utilizado sempre que um processo filho termina, mais especificamente quando o processo que criamos para executar uma determinada task termina. Quando isso acontece, a função por nós definida **child_handler** trata de dar update à estrutura que contém o número de transformações que está a correr no momento, bem como da remoção da task da lista de tasks.

Capítulo 4

Funcionalidades

4.1 Funcionalidades Básicas

No que toca a funcionalidades básicas implementamos tudo o que foi pedido.

4.1.1 Ficheiro de Configuração

A leitura do ficheiro de configuração é feita utilizando a função **fill_struct_conf_file** que coloca na estrutura destinada ao ficheiro de configuração a informação presente no mesmo.

4.1.2 Processamento e armazenamento de um ficheiro

Como já foi descrito anteriormente, esta funcionalidade foi implementada usando *pipes com nome* e uma estrutura de listas ligadas.

4.1.3 Concorrência

Esta funcionalidade foi implementada através da criação de processos filhos, que tratam de executar tasks, enquanto que o processo pai continua a ler informação vinda de clientes.

4.1.4 Status

Para finalizar as funcionalidades básicas, o **status** foi implementado usando comunicação cliente-servidor via *pipes com nome*.

4.2 Funcionalidades avançadas

4.2.1 Prioridades

As **prioridades** foram implementadas considerando as duas formas aceites para o input da mesma tais como:

```
$ ./sdstore proc-file -p <priority> samples/file-a outputs/file-a-output
    transf1 ...

$ ./sdstore proc-file samples/file-a outputs/file-a-output transf1 ...
```

sendo que, no 1º caso a prioridade será a que está no argumento <priority> e, no 2º caso, a prioridade, por *default*, será 0. A implementação das **prioridades** foi feita usando a estrutura de armazenamento de **tasks**, visto que a inserção de novas tasks está a ser feita tomando por ordem a prioridade da tarefa a ser inserida, da maior para a menor.

4.2.2 Tamanho do ficheiro inicial e final

Para resolver o problema de calcular o tamanho de um ficheiro implementamos uma função que recorre à função **lseek**, posicionando o apontador de escrita/leitura no fim do ficheiro. Como a função **lseek** devolve o número de bytes que percorreu, tornou-se trivial a implementação desta funcionalidade.

Capítulo 5

Conclusão

Em conclusão, a nível geral, e tendo em conta o panorama apresentado nos capítulos anteriores e os objetivos pedidos para este trabalho, como grupo, achamos que todos os objetivos foram cumpridos, conseguindo superar com sucesso, todas as dificuldades que fomos encontrando, sempre com um olhar crítico e criterioso. Temos a certeza que aprendemos os conteúdos e objetivos desta Unidade Curricular conseguindo, por isso, realizar com sucesso este projeto.