# Sistemas Operativos

Carlos Machado a97114, Gustavo Pereira a96867 e Vasco Manuel a96361

29 de maio de 2022

Grupo 11

# Conteúdo

# 1 Introdução

O presente documento relata o desenvolvimento de um serviço de armazenamento eficiente e seguro de ficheiros constituído por um par de programas, cliente e servidor.

Este projeto foi realizado no âmbito da unidade curricular Sistemas Operativos, integrada no 2º ano da Licenciatura em Engenharia Informática, com o propósito de solidificar os conhecimentos teórico-práticos sobre a arquitetura e funcionamento dos sistemas operativos.

# 2 Motivação

Pretendia-se implementar um serviço que permitisse aos utilizadores armazenar uma cópia dos seus ficheiros de forma segura e eficiente, poupando espaço de disco. Dessa forma, o serviço disponibilizaria funcionalidades de compressão e encriptação dos ficheiros a serem armazenados.

O serviço permitiria a submissão de pedidos para processar e armazenar novos ficheiros bem como para recuperar o conteúdo original de ficheiros guardados previamente. Era também pretendida a possibilidade de consultar as tarefas de processamento de ficheiros a serem efetuadas num dado momento e estatísticas sobre as mesmas.

Para sustentar este serviço era necessário desenvolver dois programas, um deles um cliente, responsável por fazer os pedidos, outro um servidor para onde convergem os pedidos de compressão e encriptação dos ficheiros a serem armazenados feitos pelos clientes.

# 3 Arquitetura

## 3.1 Cliente

### 3.1.1 Inicialização e interação

O cliente, programa *SDStore*, é o meio dos utilizadores fazerem os seus pedidos. Sendo inicializado através de comandos no terminal, este envia para os pedidos para o servidor, o responde, enviando mensagens relativas ao processamento dos ficheiros. É possível fazer dois tipos de pedidos, através dos seguintes comandos:

- **proc-file** usado para processar e armazenar novos ficheiros. Recebe como argumento, por ordem: a prioridade (elemento opcional), o caminho do ficheiro a ser modificado, o caminho onde o ficheiro destino será escrito no fim da transformação e as transformações que se pretendem aplicar sobre o ficheiro de entrada (designadas à frente neste relatório).

  *Ex: $sdstore proc-file -p 5 ./testin ./testout nop encrypt bcompress*

- **status** usado para consultar as tarefas de processamento de ficheiros a serem efetuadas no servidor. Este funcionamento pode ser usado através do comando:

  *$sdstore status*

### 3.1.2 Envio de Pedidos

Após validar os argumentos do comando do pedido, o programa cliente abre um FIFO identificado pelo seu PID para posteriormente receber as mensagens do servidor e cria uma *struct* onde guarda as informações relativas ao pedido para posteriormente enviar ao servidor. Consta nessas informações:

- um cabeçalho que especifica o tipo de pedido,
- a prioridade atribuída ao pedido,
- o PID do cliente.

Caso o pedido seja *proc-file*, é associado um *payload* dessa *struct* onde se encontram os caminhos dos ficheiros e as transformações requeridas.

Se o pedido foi recebido pelo servidor com sucesso, resta ao cliente aguardar as atualizações relativas ao respetivo pedido.

## 3.2 Servidor

### 3.2.1 Inicialização e configuração

O servidor, programa *SDStored*, ocupa-se de processar os pedidos que recebe dos vários clientes. A sua inicialização, feita através do terminal, implica a existência de um ficheiro de configuração e uma diretoria onde se encontram os executáveis relativos às transformações.

*Ex: $sdstored config.txt exec/*

O ficheiro de configuração corresponde a um ficheiro de texto onde se define o número máximo de instâncias para cada uma das transformações.

As configurações são aplicadas ao arranque do servidor, antes deste se encontrar apto a receber os diversos pedidos. Nessa mesma fase é criado o canal de receção dos pedidos dos clientes (matéria a ser aprofundada mais à frente), são inicializadas as filas de espera e é também definido o comportamento do programa no caso de receber um sinal de interrupção.

### 3.2.2 Receção de pedidos

O servidor recebe pedidos dos clientes através da leitura assíncrona do pipe usado para a comunicação cliente->servidor. Lê o cabeçalho do seu pedido e verifica se é um pedido de *status* ou de *proc-file*. Caso seja um pedido de *status*, abre o *pipe* do cliente e envia uma string com a informação pedida. Caso seja um pedido de *proc-file*, o servidor lê os dados do cabeçalho, cria uma estrutura *Process*, verifica se tem recursos suficientes para o executar e põe-no na fila de espera caso seja possível (*Processos* explorados mais à frente). Se o *pipe* cliente->servidor estiver vazio e se as filas não estiverem vazias, o servidor passa ao manuseamento dos processos; se estiverem vazias o servidor bloqueia à espera de mais pedidos.

### 3.2.3 Gestão de processos

**Processos** Caso o pedido do cliente seja válido e do tipo *proc-file*, esse pedido passa a ser representado na forma da estrutura *Process*, guardando as informações do pedido do cliente e da cadeia de processos necessária para fazer a transformação pedida. O servidor mantém-se a par do estado de cada processo e envia para o cliente esse estado caso ele mude, avisando então se está à espera, em execução, se foi concluído ou se falhou.

**Manuseamento de processos** Os processos estão divididos em duas filas ordenadas pela prioridade, uma para processos em espera e outra para processos em execução.

Primeiro o servidor itera ordenadamente pela fila dos processos em execução para recolher as transformações que já acabaram, e, caso encontre um processo que já acabou de aplicar todas as suas transformações, manda uma mensagem de conclusão ao cliente, junto com a informação dos bytes do ficheiro de entrada e de saída, e liberta esse processo.

Após isso, o servidor itera ordenadamente pelos processos na fila de espera para tentar pô-los a executar com as transformações disponíveis naquele momento. Caso o processo tenha sido posto a executar, ele é colocado na fila de execução.

**Execução de processos** Considera-se que um processo está a ser executado quando é formada a cadeia de transformações a que o pedido associado se refere.

### 3.2.4 Comunicação com o cliente

Para comunicar com o cliente, o servidor envia, numa *struct*, uma *string* e um *header* com o seu tamanho, através de um FIFO criado pelo cliente antes de enviar o pedido. No final do processamento, não sendo necessária mais interação com o cliente, o servidor fecha o descritor do ficheiro FIFO, que termina a execução do programa do cliente. Este método é utilizado para enviar a informação do estado do servidor e de cada processo aos respetivos clientes.

### 3.2.5 Estado

O estado do servidor é dado pelas transformações disponíveis e em utilização e pelos processos em execução num dado momento. Ao receber um pedido de *Status*, o servidor envia a informação dos processos na fila de execução e a informação dos recursos para o cliente.

## 3.3 Mecanismos de comunicação

Foi necessário o desenvolvimento de mecanismos de comunicação inter-processos de forma a que o envio de pedidos ao servidor por parte dos clientes, assim como a execução de transformações em paralelo dentro do servidor fosse possível.

Para isso recorreu-se ao uso tanto de *pipes* anónimos como *pipes* com nome (FIFOs) tendo em conta a situação para se fazer o uso correto.

### 3.3.1 Comunicação Servidor-Cliente

No caso da comunicação entre os clientes e o servidor, optou-se que esta fosse feita através de *pipes* com nome uma vez que se trata de uma comunicação entre programas distintos.

No sentido cliente -> servidor é usado um ficheiro FIFO, criado pelo servidor somente para leitura, onde todos os clientes "escrevem"os seus pedidos, sendo assim um canal único de entrada de pedidos no servidor.

Para a comunicação dada no sentido servidor -> cliente viu-se como sendo necessário a criação de ficheiros FIFO para cada cliente pois, caso contrário, i.e. um FIFO de onde todos os clientes liam as mensagens vindas do servidor, não garantiria que as mensagens chegassem ao cliente correto. O PID do cliente é usado para distinguir entre os diferentes FIFOs.

### 3.3.2 Comunicação Inter-Processos

As sequências de transformações pedidas por um dado cliente são executadas, dentro do servidor, em concorrência com a receção de pedidos de outros clientes. Sendo um dos requisitos evitar a criação de ficheiros temporários relativos aos estados intermédios de uma sequência de transformações, foi fulcral o uso de *pipes* anónimos para estabelecer pontes de comunicação entre os processos filhos do servidor, de forma a que as transformações fossem realizadas em cadeia.

# 4 Conclusão

Apesar do projeto se encontrar passível a várias otimizações, seja melhorias no uso de memória ou simplesmente clareza no código, consideramos o aspeto final do trabalho como satisfatório, estando particularmente satisfeitos pelo facto de terem sido atingidos todos os objetivos pretendidos, fazendo uso dos conhecimentos adquiridos nas aulas teóricas e práticas da presente unidade curricular, assim como de Programação Imperativa, tendo sido um meio conveniente e eficaz para solidificação do conhecimento das matérias lecionadas.