Universidade do Minho

Licenciatura em Engenharia Informática

Sistemas Operativos
Relatório Trabalho Prático - LEI 2021/22

Pedro Ferreira (A93282) Diogo Casal Novo (A88276)
Ana Gonçalves (A93259)

Maio 2022

Conteúdo

Capítulo 1

Introdução

No âmbito da Unidade Curricular de Sistemas Operativos, foi proposto ao grupo a implementação de um serviço que permita aos utilizadores armazenar uma cópia dos seus ficheiros de forma segura e eficiente.

Neste documento o grupo vai explicar a estratégia usada tanto no *servidor* como no *cliente* para conseguir executar todas as funcionalidades pedidas no enunciado deste trabalho. Sendo no fim apresentada uma conclusão sobre o desenvolvimento deste projeto.

Capítulo 2

Sdstore

O cliente tem como objetivo oferecer uma interface com o utilizador através da linha de comandos, onde este poderá interagir com o servidor através de argumentos especificados na linha de comandos do cliente. Ele ainda escreve no *standard output* informações sobre o estado do serviço ou do processamento do pedido, como mensagens *pending ,processing e concluded*.

Para ser possível o correr correto de um cliente terá o formato terá de ser um dos seguintes:

- ./sdstore proc-file <priority><ficheiro input><ficheiro output>de operações>
- ./store status

No código do ficheiro do cliente começamos por criar um *fifo* com o nome igual ao pid do cliente que vai receber as mensagens com origem no servidor. Em seguida é utilizada a função *sendRequest*, que tem o objetivo de criar a mensagem que vai ser enviada pelo *"fifo FifoMain"* com o formato necessário para ser corretamente interpretado pelo servidor.

Esta mensagem poderá ter dois formatos:

- caso o comando seja um pedido para processamento do ficheiro:
  proc-file <priority><pid ><número de comandos ><inputFilePath><outputFilePath><comandos>
- caso o comando seja para receber o status:
  status<pid>

Para o primeiro caso, este deverá ficar à espera de resposta do servidor no *fifo* com nome igual ao seu pid. A primeira resposta vai ser para uma simples notificação que o pedido está a ser processado de momento, e a segunda resposta deverá conter o valor de bytes do ficheiro criado e do ficheiro original as quais vão ser apresentadas ao utilizador juntamente com a mensagem *"Concluded!"*.

Por fim, o cliente ainda envia uma mensagem **finish<pid>** para que o servidor possa utilizar as operações que este pediod utilizou noutros pedidos.

Quanto ao segundo caso, apenas são recebidas as informações do estado do servidor, pelo *fifo* com nome igual ao seu pid, e são diretamente impressas para o *standard output*, que incluem as informações de todos os pedidos a serem atualmente processados, juntamente com as o número de cada operação a ser utilizada como também o seu limite.

Capítulo 3

Sdstored

O servidor tem o objetivo de manter em memória todas as informações importantes de modo a suportar as funcionalidades pedidas no enunciado.

Este começa por receber e interpretar dois ficheiros: o primeiro corresponde ao caminho para um ficheiro de configuração, o segundo argumento corresponde ao caminho para a pasta onde os executáveis das transformações estão guardados.

Em seguida é feita a inicialização das listas *queue* e *working*. Estas listas têm um tamanho máximo representado pela variável *QUEUESIZE*, e cada valor representa a estrutura *Request*.

Estas listas são usadas para representar as listas de pedidos em espera e a serem processados, respetivamente.

Esta estrutura é constituída por:

- **int** nrCmds: que representa o número de comandos a executar
- **char*** pid: que representa o pid do servidor
- **char*** intput: corresponde ao caminho para o ficheiro de configuração
- **char*** output: corresponde ao caminho para a pasta onde os executáveis das transformações estão guardados
- **char**** cmds: que representa a lista de comandos a ser executados
- int priority: que representa a prioridade da operação, retirada dos comandos dados pelo utilizador

3.1 proc-file

A primeira coisa a fazer neste caso é criar um *request*, que será preenchido com a informação retirada do *FifoMain* através da função *fillRequest*. Esta função separa o conteúdo retirado do *fifo* por espaços, e em seguida coloca os valor nas variáveis do *request* que lhes correspondem.

De seguida, adiciona esse pedido na lista *queue* para que este fique à espera da sua vez.

Por fim, chama a função *checkQueue*. Esta função retira da lista *queue* o pedido mais prioritário e possível de ser iniciado. Um pedido é mais prioritário se tiver um maior nível de prioridade, ou, em caso de empate, se for o primeiro a chegar.

A função *requestInQueueCanProceed* verifica se o pedido pode ou não ser iniciado. 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.

Para o caso do pedido que vai ser iniciado, começa por acrescentá-lo à lista *working* e retirá-lo da lista *queue*, atualizar a lista *transformationsReps* com os comandos usados nesse pedido, e para finalizar é criado um processo filho para processar este pedido.

3.2 finish

Para o caso do *finish*, vai ser retirado da lista *working* o pedido associado ao *pid* recebido e, de seguida vai ser atualizada a lista *transformationsReps* removendo os comandos usados nesse mesmo pedido. Finalmente, vai ser chamada a função *checkQueue* (previamente referida na secção *proc-file*) para que novos pedidos possam ser iniciados.

3.3 status

Quando um pedido do tipo *status* é recebido, ele é imediatamente respondido por um processo filho com a função *showState*. Esta mostra uma imagem do servidor no momento em que o novo processo foi criado.

Começa por percorrer a lista *working* para obter informações relativas aos ficheiros que estão a ser processados do momento e, seguidamente, as listas *transformationsFile*, *transformationsReps* e *repsFile* para obter as informações relativas ao número instâncias de uma certa transformação a correr concorrentemente.

3.4 terminação graciosa

Implementámos um sistema de terminação graciosa em que o sevidor ao receber o sinal *sigint* ele coloca a variável global *end* a um, esta variável sendo positiva faz o server não receber mais pedidos, e quando temina os que estão em *queue* ele faz um *sigkill* para matar o servidor.

Capítulo 4

Conclusão

Finalizando assim a apresentação do nosso projeto, o grupo conclui que este trabalho ajudou bastante a aprofundar e a consolidar conhecimento adquiridos ao longo da cadeira, tais como, a utilização de *pipes* com nome para estabelecer comunicações entre servidor-cliente e cliente-servidor, *pipes* anónimos para permitir a processamento das diversas transformações, criação e manuseamento de processos pais e filhos, e utilização de sinais. Foi também uma boa oportunidade para desenvolver as nossas capacidades a nível da programação em linguagem C.