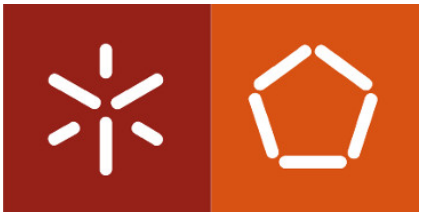

Universidade do Minho
Licenciatura em Engenharia Informática
2ºano - 2º Semestre

Sistemas Operativos

Trabalho Prático - SDStore: Armazenamento Eficiente e Seguro de Ficheiros

A93311 – Francisco Maria Sousa Gonçalves Paiva
A93197 – Miguel Oliveira Dias
A90234 – Ana Filipa da Cunha Rebelo

29 de maio de 2022
Grupo 110

Conteúdo

1 Introdução

Para este trabalho foi criado um programa que permite a comunicação entre um servidor e vários clientes, nomeadamente a efetuação de pedidos de transformação de um ficheiro através compressão e/ou encriptação dos dados do ficheiro e a criação de um novo ficheiro. Para a comunicação entre o servidor e o cliente foram usados pipes com nome.

2 Pedidos

De modo a guardar informação pertinente a um pedido foi criada uma struct auxiliar:

```
1
2 typedef struct task{
3     int id;
4     char *t;
5     char *in;
6     char *out;
7     int status;
8 } *Task;
```

Esta struct guarda o id do pedido, as transformações efetuadas, o ficheiro lido e criado e o estado do pedido. Estes pedidos são criados e guardados num array de pedidos.

3 Limites de transformações

Cada transformação tem um limite guardado num ficheiro de texto. Para garantir que se cumpria com esse limite foi criada uma struct auxiliar para guardar a capacidade e nome de transformação sendo que após identificados eram definidos os limites numa variável associada a cada transformação.

```
1 typedef struct {
2     char *type;
3     int cap;
4 }Transform;
```

Foram criadas também funções auxiliares para verificar se o número de instancias a correr está no limite maximo e para atualizar o valor das transformações.

4 Arquitectura de processos

O programa servidor é executado e fica à espera de pedidos dos clientes. Cada cliente que efetua um pedido é atendido quando o servidor se encontrar disponível. Dependendo do pedido efetuado o servidor envia a resposta adequada.

5 Mecanismos de comunicação

Para a comunicação são usados pipes com nome, nomeadamente "server_to_client"e "client_to_server", sendo o primeiro usado pelo servidor para enviar dados ao cliente e este lê dados enviados. Infelizmente o programa não consegue responder a dois pedidos concorrentes.

6 Pedidos

Existem três diferentes tipos de pedidos ao qual o servidor pode responder.

6.1 Exemplo de execução do programa

O mais básico é quando o programa cliente é executado sem argumentos, neste caso o cliente envia uma mensagem vazia através do pipe "client_to_server"e o servidor devolve a seguinte string que é enviada pelo pipe "server_to_client":

```
1 "./sdstore status\n./sdstore proc-file priority input-
    filename output-filename transformation-id-1
    transformation-id-2 ...\n"
```

6.2 Status

Como foi mencionado no enunciado do trabalho o cliente pode enviar um pedido do tipo "Status"de modo a ver o estado de funcionamento do servidor. Como função auxiliar foi criada a função **char* status ()** que guarda numa string os pedidos a serem executados, se existirem, e as transformações a serem efetuadas juntamente com as capacidades de cada uma. Infelizmente o servidor não responde a um pedido status ao mesmo tempo que está a processar um ficheiro.

6.3 Proc-file

O pedido do tipo **Proc-file** que processa um ficheiro e efetua transformações sobre ele recorre ao uso da função**int runPipedCommands(int commands, char *tokens[],char*input,char*output)** que lê de um array de chars as transformações a serem efetuadas e cria os pipes anônimos necessários e o dup para enviar o output de uma transformação para o input de outra.

7 Conclusão

Através da realização deste trabalho foi possível adquirir conhecimento técnico de comunicação entre processos. Por fim concluimos que o trabalho prático está minimamente bem concebido registrando uma evolução em termos de conhecimentos relativos a sistemas operativos.