Universidade do Minho

Licenciatura em Ciências da Computação

Trabalho Prático de SO
(Sistemas Operativos)
Grupo nº27
29/05/2022

André Neves da Costa - a95869 (1)
Bruno Miguel Fernandes Araújo - a97509 (2)
Francisco José Pereira Teófilo - a93741 (3)

(1)

(2)

(3)

Índice:

Introdução:

Este projeto consiste na criação de um serviço que permite aos utilizadores **armazenar** uma cópia dos seus ficheiros de forma **segura e eficiente**, poupando espaço de disco.

Para isto, criamos um **servidor** que se encarrega de resolver pedidos de transformações sobre um ficheiro, enviados por um ou mais **clientes**.

Além disso, o cliente é capaz de **pedir o estado** do servidor.

Acabamos por ter um trabalho com praticamente todos as funcionalidades pedidas, a única que não desenvolvemos foi as que envolviam prioridades, de resto temos um projeto que satisfaz o que previamente mencionamos e duas funcionalidades avançadas (Indicação do número de bytes do ficheiro input/output e encerramento gracioso do servidor com SIGTERM).

Cliente:

Funcionalidades:

Na inicialização do cliente, este verifica primeiro se a sua chamada está **correta** e se o servidor com que vai comunicar se encontra-se **operacional**.

Para uma chamada de **"proc-file"** este verifica se os seus argumentos estão corretos, se o **ficheiro input** existe, se o **ficheiro output** não existe e se as **transformações** colocadas são válidas (nop, bdecompress, bcompress, gcompress, gdecompress, encrypt, decrypt).

De seguida, independentemente da sua chamada (proc-file ou status), este cria um **pipe com nome** usando o seu **pid** e envia uma mensagem customizada para o servidor ,onde inclui o **path** para o pipe, dando acesso a este ao pipe com nome.

Para executar o cliente existem dois comandos:
./execs/cliente proc-file ./input/teste.txt ./output/(ficheiro-output).txt (transformações)
./execs/cliente status

Servidor:

Funcionalidades:

Na inicialização do servidor, este verifica se a sua **chamada** é válida e se o **ficheiro de configs** (ficheiro que contém o máximo de pedidos para cada transformação) existe.

Este processa o ficheiro das transformações e armazena os **número máximo de ocorrências** de cada transformação.

Após a leitura dos parâmetros, o servidor cria um **pipe com nome** pelo qual vai receber os **pedidos dos clientes** e entra num ciclo de ler pedidos que apenas termina ao receber o sinal **SIGTERM** e caso a lista das tarefas a executar se encontre **vazia**. Ao receber **SIGTERM** o servidor para de receber pedidos dos clientes e termina as tarefas que ainda tem na lista das tarefas a executar antes de terminar o ciclo.

Dentro do ciclo, o servidor primeiro **trata** do comando que recebeu através de um processo filho (**TaskManager**) para o servidor não perder tempo enquanto outros clientes lhe tentam enviar pedidos.

No **TaskManager**, caso o pedido do cliente seja um **status**, este abre o FIFO de resposta ao cliente e envia-lhe o **estado atual** do servidor. Caso o pedido seja um **proc-file**, o **TaskManager** verifica se há recursos para satisfazer o pedido e caso haja é criado um novo processo filho (**ExecsManager**) e adiciona este pedido à **lista de tarefas**. Além disso, se o pedido **proc-file** enviado pelo cliente ultrapassar logo o máximo este nem processa o pedido e responde ao cliente que não tem recursos, por exemplo temos o encrypt com máximo 2, se um cliente enviar um pedido com 3 encrypt's, o servidor recusa o seu pedido e faz o que foi previamente mencionado.

No **ExecsManager**, caso não haja transformações suficientes para receber o pedido, este entra em estado de espera e coloca no **STOUT** do cliente "*Pending*..." dizendo-lhe que o seu pedido está em espera. Quando o pedido sai do **estado de espera**, é colocado no **STOUT** do cliente "*Processing*..." informando-o que o seu pedido está a ser processado e é iniciada a execução do pedido que acontece através de processos filho a correrem concorrentemente entre si e que comunicam através de pipes anônimos, cada processo executa uma das transformações necessárias.

Quando a execução do pedido termina é terminado o **ExecsManager**, voltamos ao **TaskManager**, que remove o pedido da lista de tarefas e termina logo depois.

A lista de tarefas mencionada é uma **linked-list** criada para guardar os pedidos ainda por executar, em cada nodo esta tem o **pid** do **ExecsManager** que o executa e um **string** com o pedido em forma de texto.

Observações:

Tanto para o Servidor como para o Cliente é possível receber o formato do comando que é suposto inserir ,colocando na linha do terminal apenas o argumento de execução , ou seja da seguinte forma:
Para o servidor, "./execs/servidor".
Para o cliente, "./execs/cliente".

Tendo em conta que os pedidos são resolvidos rapidamente no servidor, obter um caso de Pending e um status que indique as tasks que estão a ser processadas é bastante difícil, então para simular testar estes casos fizemos o seguinte:
Para um caso de Pending colocamos em comentário o seguinte for

```
for(int i = 0; i < 7; i++){
    *transusados[i] -= transneeded[i];
}
```

Assim, os execs que foram realizados nunca são registados, isto então faz com que o transusados ultrapasse o transmax(obtido na leitura dos máximos do ficheiro config) provocando um Pending eterno.
Para o caso de indicar as tasks que estão a ser processadas colocamos em comentário o seguinte

```
sprintf(novaTask, "%d,remove%c",pidmon,'\0');
//pede ao servidor para tirar esta task
write(fd2, novaTask, strlen(novaTask)+1);
```

Assim,o servidor nunca recebe uma mensagem de **remove** e nunca vai remover a task com aquele pid, fazendo com que na execução de um status apareça essa task ainda a ser processada.

Na execução do comando status, ocorrem uma certa peculiaridade, quando existem muitas tasks , estas parecem que não estão ligadas corretamente, no sentido em que se fizermos status duas vezes aparecem diferentes tasks em cada ocasião, e nunca aparecem todas as que estão a ser processadas (É uma coisa que não entendemos como acontece, porque o código de add's está correto e quando é recebido um pedido de status este percorre a lista ligada dos pedidos toda).
Ex: Imaginemos que temos 5 tasks a serem processadas, no primeiro status aparece as tasks 0,2, o segundo aparece as tasks 1,4 e num terceiro status apareceria as tasks 0,3.
(Noutra situação as tasks poderiam ser apresentadas de outra forma, é algo completamente aleatório).

Conclusão:

Este projeto ajudou-nos a consolidar os conhecimentos práticos de Sistemas Operativos e tornou-nos capazes de os aplicar em cenários reais.

Foi um projeto bastante desafiante, tivemos muitas dificuldades para verificar se o código estava a funcionar corretamente, printf's e write's para o Stdout por vezes simplesmente não executavam e dificultou bastante o progresso.

O que se provou mais difícil foi implementar a comunicação entre processos a partir de pipes anônimos quando se executavam as transformações.

Finalmente, conseguimos aplicar todas as funcionalidades básicas e duas das funcionalidades avançadas de forma organizada e eficiente e, por isso, estamos satisfeitos com o nosso trabalho.