Universidade do Minho
Escola de Engenharia

Sistemas Operativos

Relatório Trabalho Prático

Gonçalo Semelhe Sousa Braga A97541
João Alvim A95191
Gonçalo Martins dos Santos A95354
Grupo 119

Índice

1 Introdução **2**

2 Pedidos **2**

3 Cliente - Sdstore **3**

4 Servidor - Sdstored **3**

4.1 Estrutura principal do servidor 4

4.1.1 Sinal SIGTERM 4

4.1.2 Sinal SIGUSR1 4

4.1.3 Sinal SIGCHLD 4

4.2 Filas de Espera e de Execução 4

4.3 Execução de pedidos 5

4.4 Utilização de sinais 5

5 Conclusão **5**

1 Introdução

Pretende-se implementar um serviço que permita aos utilizadores armazenar uma cópia dos seus ficheiros de forma segura e eficiente, poupando assim espaço de memória. Com isto, o servidor irá ficar responsável por comprimir e cifrar os ficheiros dados como input. O programa deve puder suportar a submissão de pedidos bem como a execução de transformações aos ficheiros. Posteriormente, deve também ser possível consultar as tarefas que estejam em execução, bem como informações sobre o número de instâncias disponíveis de cada transformação.

2 Pedidos

O utilizador quando pretende submeter um pedido tem de o fazer da seguinte forma:

1. Tipo de pedido: proc-file ou status.
2. Prioridade, apenas no tipo proc-file e opcional.
3. Ficheiros de input e output.
4. Transformações que quer executar.

```
$ ./sdstore proc-file <priority> samples/file-a outputs/file-a-output bcompress nop gcompress encrypt nop
```

Figura 1: Pedido proc-file

Como podemos ver na figura 1, o pedido do tipo proc-file é representado por este input.

Quando recebemos o pedido, o cliente trata de o enviar para o servidor e este encarrega-se de decompor o pedido para uma struct que é composta pelo número de pedido, a prioridade do pedido, o file descriptor do fifo pelo qual o servidor envia a informação para o cliente, tamanho do pedido (número strings do array de strings pedido), o pid do processo que está encarregue de processar o pedido, um array que indica as quantidades de instâncias de cada transformação a utilizar, e a linha de input que o utilizador passou como parâmetro ao cliente, exceto no nome do executável (sdstore).

```
typedef struct pedido{
    int nrPedido;
    int prioridade;
    int fifo_ouput;    //File Descriptor do fifo que envia informação para o client
    int tampedido;
    int pid;
    int transNecess[7];
    char *pedido[];
}*Pedido;
```

Figura 2: Struct Pedido

As informações que o cliente envia ao servidor são:

-Comando (linha de input menos executavel);

-Nome do fifo pelo qual o servidor vai responder ao cliente;

-Número de strings que o comando tem;

```
$ ./sdstore status
```

Figura 3: Pedido status

Como podemos ver na figura 3, o pedido do tipo status é representado por este input.

O pedido do tipo status, serve para visualizarmos quais os pedidos que se encontram em execução, bem como o número de cada tipo de transformação que se encontra a ser utilizada naquele momento.

3 Cliente - Sdstore

O cliente tem como função implementar uma interface de comunicação entre utilizador e o servidor.

O cliente ainda tem como função passar as informações descritas em cima para o servidor através de um FIFO ou named pipe.

Com isto, o cliente recebe o argumento do utilizador, e utilizando fifos, envia toda a informação para o servidor para este as puder processar. Existe duas vertentes dessa função, que realiza essa tarefa, uma apenas para pedidos proc-file e outra apenas para pedidos status. Ainda assim, a metodologia utilizada em ambas as funções é igual.

O cliente, é também responsável por receber toda a informação que o servidor envia, para ser transmitida ao utilizador.

Para que a comunicar no sentido do servidor cliente (o sevidor envia para o cliente) fosse possível, cada cliente cria um fifo privado no qual o servidor deve escrever para que o cliente receba as respostas do servidor. Ou seja, existe um fifo geral de todos os clientes para um único servidor e existem n fifos para n clientes no sentido da comunicação entre o servidor e o cliente.

4 Servidor - Sdstored

O servidor tem como função receber os ficheiros input e aplicar as transformações que o utilizador pretende aos mesmos.

O servidor está sub-dividido em 3 casos principais, que são:

- Quando recebe um novo pedido (recebe o sinal SIGUSR1) e este ainda não recebeu o sinal SIGTERM.

- Quando recebe um novo pedido (recebe o sinal SIGUSR1) e já tinha recebido o sinal SIGTERM.

- Quando recebe um sinal SIGCHLD, ou seja, um pedido terminou a sua execução.

Para facilitar o trabalho do servidor, este vai ter acesso a duas listas que servem para guardar os pedidos que estão á espera de serem executados e os pedidos que estão a ser executados.

4.1 Estrutura principal do servidor

O processo principal do servidor vai estar permanentemente num ciclo while no qual existe um pause logo no início. Este pause pode ser terminado com o recebimento de 3 sinais: SIGTERM, SIGUSR1 ou SIGCHLD.

4.1.1 Sinal SIGTERM

O sinal SIGTERM indica ou servidor que é altura de terminar e que não deve de aceitar num novo pedido e proceder ao termino dos pedidos que estão em fila de espera e os que estão a ser executados.

4.1.2 Sinal SIGUSR1

O processo principal do servidor cria um processo filho auxiliar que é responsável por ler novos pedidos dos clientes, quando este recebe um novo pedido, envia o sinal SIGUSR1 ao processo principal e seguidamente envia toda a informação dos clientes para o processo principal. Depois, o processo principal encarrega-se de determinar se deve pôr o pedido em fila de espera, se deve pôr o pedido em execução ou se deve simplesmente ignorar o pedido caso já tenho recebido o sinal SIGTERM.

4.1.3 Sinal SIGCHLD

O processo principal ainda cria um processo filho responsável por executar cada um dos pedidos. Este processo filho depois cria vários processos filhos (netos do processo principal) cada um para executar uma transformação de cada pedido. No final de todos os seus filhos executarem, o processo responsável pelo pedido termina a sua execução e o processo principal recebe o sinal SIGCHLD e, por isso, sabe que um pedido terminou a sua execução. Depois, o processo principal limita-se a limpar o pedido terminado da fila de pedidos em execução.

4.2 Filas de Espera e de Execução

No nosso trabalho, implementamos 2 listas ligadas de struct pedido, uma referente aos pedidos que estão em espera e outra aos pedidos que estão a ser executados. Decidimos usar listas ligadas porque é a estrutura de dados que permite uma remoção elementos de maneira mais fácil. Quando o servidor recebe um novo pedido este tem que decidir se o pedido deve ir para a fila de espera ou de execução. Dependendo do caso, o servidor insere o pedido na lista ligada correspondente e se for na lista de pedidos em execução o respetivo pedido é posto em execução, caso contrário simplesmente é inserido na fila de espera. Quando o servidor recebe o sinal SIGCHLD este sabe que um pedido terminou a sua execução e então o servidor prossegue a insvestigar a lista de pedidos em execuçãoe

4.3 Execução de pedidos

A execução de pedidos não é realizada pelo processo principal do servidor. Sendo assim este delega a um processo filho que averigua quantos processos filho, processos netos do processo principal necessita para executar as transformações ao ficheiro, tendo sempre em atenção ao número de transformações a executar naquele momento.

Um exemplo de execução é dado a seguir:

```
proc-file fin fout bcompress nop
```

Figura 4: Execução pedido

Olhando para um exemplo de uma linha de input (figura 4), colocada pelo utilizador, é possivel verificar que possui duas transformações a serem realizadas: bcompress e nop.

O programa recebe esta linha e cria o pedido (struct com diferentes parâmetros já abordada acima). De seguida, o servidor delega a um processo filho este pedido. Ele irá ver quantos processos necessita de criar para executar na totalidade o pedido.

Neste caso em particular necessita de dois. Visto isto um dos processos executa o bcompress no ficheiro input, manda o seu output por um pipe para o segundo processo filho e este lê do pipe executa o nop no que leu do pipe e cria o ficheiro output escrevendo o resultado final no ficheiro fout.

Com isto obtemos o resultado final deste pedido, no ficheiro fout.

4.4 Utilização de sinais

Todos os casos que o servidor possui são comandados por sinais, que quando emitidos têm uma alteração no estado do servidor. Por este motivo, é que ele cada vez que vai verificar em que situação está verifica quais os valores que as variáveis globais, que representam os sinais.

```
printf("Pause\n");
pause();
printf("Recebi um sinal!!!\n");
```

Figura 4: Receção de sinais

É pretendido que o servidor tenha uma espera passiva. Para tal acontecer, quando um dos processos a ser executado termina, envia um sinal ao processo principal a avisar que terminou, assim não é preciso que o programa principal faça uma espera ativa, diminuindo assim a carga sobre o processador.

5 Conclusão

Em suma, os resultados obtidos ao longo do trabalho foram bastante satisfatórios uma vez que conseguimos construir todos os desafios propostos no enunciado.

Concluindo, com o desenvolvimento deste trabalho podemos ver o quão importante é fazer um bom manuseamento das system call e a importância de conhecer como funcionam ficheiros e o sistema operativo. Estes são fatores bastante importantes e a ter em consideração nos próximos trabalhos ao longo do nosso percurso académico e profissional.