Universidade do Minho

Engenharia Informática

Trabalho Prático Sistemas Operativos

Diana Teixeira (A97516)

Artur Leite (A97027)

Afonso Magalhães (A95250)

Índice

1) Introdução

Este projeto consiste na implementação de um serviço que permita aos utilizadores armazenar uma cópia dos seus ficheiros de forma segura e eficiente, poupando espaço de disco. Para tal, será necessária a implementação de funcionalidades de compressão e cifragem dos ficheiros a serem armazenados, como também deverá ser permitida a submissão de pedidos e a consulta das tarefas de processamento de ficheiros. Com isto, verificamos que o maior desafio foi encontrar uma forma de implementar uma boa comunicação entre os processos, acabando por superar este obstáculo através do uso de forks. O cliente e o servidor comunicam entre si através de dois pipes com nome, um de leitura, que envia comandos do cliente para o servidor, e outro de escrita, que envia o output dos mesmos de volta para o cliente.

2) Funcionalidades Básicas

2.1) Leitura de Argumentos- Servidor

Tal como indicado no enunciado, o programa servidor recebe dois argumentos pela linha de controlo. O primeiro trata-se de um ficheiro de input que é lido através das funções *readln e readc,* disponibilizadas nas aulas práticas. As informações obtidas vão então ser guardadas no segundo argumento que se trata de uma struct que armazena as configurações de cada transfiguração juntamente com o número máximo de instâncias que se podem executar concorrentemente. Esta struct será acedida ao longo do programa para poderem ser regulados os processos realizados.

2.2) Processamento e armazenamento de um ficheiro (proc-file)

Quando é emitido um pedido de processamento por parte do cliente, o servidor utiliza o pipe de comunicação entre servidor e cliente para ler o pedido passado como argumento. Esses pedidos serão então guardados numa fila de espera para serem efetuados.

2.3) Avisos de estado dos Processos

Após o processo descrito no ponto anterior, quando os processos são colocados na fila de espera mencionada, é emitido o estado *pending* através do pipe do cliente que será imprimido no terminal pelo próprio. Para emitir os outros dois possíveis estados, *processing* e *concluded,* o processo de emissão é semelhante, passando sempre pelo pipe de cliente.

2.4) Processamento concorrente de pedidos

Antes de iniciar a execução de um pedido, é primeiro analisado qual é o processo que se trata. Assim que essa informação é obtida, analisamos uma struct que contém duas informações cruciais: as tranfigurações possiveis e a quantidade de iterações disponiveis de cada. Caso o processo pendente utilizar recursos que não estão disponiveis de momento, este vai ter de esperar que os recursos estejam livres.

2.5) Comando status

Assim que o cliente efetua o comando status, acedemos à queue de execução e imprimimos o estado de cada um. Acedemos ainda à struct de configurações do sistema, informando ao cliente quais as tranfigurações disponiveis. Ambos conjuntos de informação englobam o estado do servidor, demonstrando assim toda a informação relevante para o usuario.

3) Funcionalidades Avançadas

3.1) Número de bytes lidos e escritos

Para esta funcionalidade usamos a função pré-definida *lseek* que conta o número de bytes que anda desde o início de um ficheiro até a uma certa posição no argumento. Ora, tendo isto em conta, ao passarmos o argumento *SEEK_END* nesta função juntamente com o ficheiro em questão, vamos obter o número de bytes presentes em cada um. Posto isto, apenas aplicamos a *lseek* no ficheiro de leitura para obter o número de bytes lido e a mesma coisa para o ficheiro onde é escrito o output. Passamos então estes valores para uma string de resposta que irá ser escrita no pipe do cliente. Eis o excerto de código:

```
Int input= open(aux->pedidos[1], O_RDONLY);
Int bytes_input= lseek(input, 0, SEEK_END);
Close(input);
Int output= open(aux->pedidos[2], O_RDONLY);
Int bytes_output= lseek(output, 0, SEEK_END);
Close(output);
Char answer[100];
N_bytes= snprintf(answer, sizeof(answer), “concluded (bytes-input: %d, bytes-output: %d\n”, bytes_input, bytes_output);
Write(client_pipe, answer, n_bytes);
Close(client_pipe);
```

3.2) Sinal SIGTERM

Para esta parte do projeto, fazemos com que, quando o programa receber um sinal SIGTERM, este execute o seguinte excerto de código:

```
Void sigterm_handler(int sig){
      Kill(id_monitor,SIGUSR1);
      Waitpid(id_monitor,NULL,0);
      Kill(getPid(),SIGKILL);
}
```

Que faz com que o programa acabe de forma graciosa, ou seja, que este acabe os processos que estão ou a executar ou pendentes, impedido a submissão de novos.

3.3) Prioridades

Este funcionalidade foi implementada de forma a que, à medida que vamos recebendo processos, estes têm uma prioridade associada, sendo estes colocados e organizados numa queue por ordem de acordo com tal. Com isto, verificamos que a queue, à medida que mais processos chegam, vai ser re-organizada, mudando assim a ordem de execução dos processos, para quando o que está a executar acabar, se passar a executar o próximo de maior prioridade. Face a isto, e para não correr o risco de haver *starvation* no programa, a cada segundo que passa na execução do programa, a prioridade dos restantes pedidos é aumentada, garantindo-se então a sua execução. O seguinte excerto do programa demonstra o processo referido para evitar *starvation*:

```
Void inc_priorities_queue(struct fila *queue){
      For(; queue != NULL; queue = queue->next){
            Queue->priority += (queue->priority < 5) ? 1 : 0;
      }
}
```

4) Ficheiro Adicional- Monitor

Com o intuito de facilitar a compreensão do programa, optamos adicionar um ficheiro que está encarregue de lidar com a execução dos pedidos, deixando os outros ficheiros encarregues de receber e organizar os pedidos que chegam ao programa. Desta forma, evitamos a sobrecarga de código nos ficheiros sdstore e sdstored.

É neste ficheiro que estão presentes as diversas structs mencionadas ao longo do relatório, nomeadamente: struct de fila de execução, struct de pedidos, struct de transfigurações, etc.

5) Conclusão

Tendo em conta que fomos capazes de implementar todas as funcionalidades pedidas no enunciado do projeto, básicas e avançadas, acreditamos que temos aqui um projeto bem conseguido. Cremos ainda que utilizamos a matéria lecionada nas aulas de SO de forma eficiente, garantindo o melhor desempenho da execução do programa. Desta forma, foi-nos proporcionada ainda uma melhor consolidação destes conceitos.