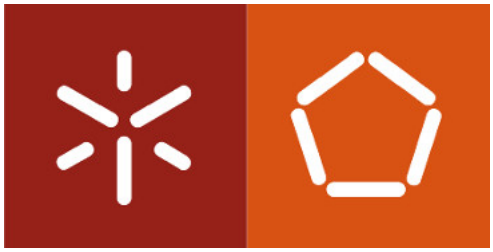

Universidade do Minho
Licenciatura em Engenharia Informática

Sistemas Operativos

Trabalho Prático - Grupo 42

Armando Silva - A87949 Gonçalo Soares - A84441

Maio 2022

Conteúdo

1 Introdução

Este projeto consiste na criação de um serviço capaz de armazenar cópias de ficheiros de forma segura e eficente, implementando funcionalidades de compressão e cifragem dos mesmos.

Inicialmente, o maior desafio que encontramos foi tornar o servidor concorrente, isto é, capaz de albergar vários clientes.

Este serviço é constituído por um servidor e múltiplos clientes que comunicam entre si através de pipes com nome, sendo que o cliente envia informações de quais funcionalidades pretende aplicar e o servidor aplica-as.

2 Servidor

2.1 Estrutura

O servidor é constituído por duas estruturas. A primeira é uma estrutura responsável pela manutenção dos processos-filhos, isto é, guardamos informações importantes dos processos filhos. Trata-se de uma lista ligada onde em cada nodo guarda o pid, a lista de transformações e um inteiro que representa o tipo de mensagem (*status* - 1 ou *proc-file* - 0).

A outra estrutura referenciada tem como funcionalidade armazenar a informação presente na configuração do servidor, sendo útil saber o nome e número máximo de transformações disponíveis.

```
//struct processos
Gonçalo Soares, 4 minutes ago | 1 author (Gonçalo Soares)
struct process
{
    int pid;
    //1 -> Proc-File 0 -> Status
    int message;
    //0 -> pending 1 -> executing
    int status;

    char* request;
    char** transformations;
    struct process *next;
};
struct process *start=NULL;

Gonçalo Soares, 2 days ago | 2 authors (Gonçalo Soares and others)
struct transformation
{
    char* name;
    int running;
    int max;
}* Transformation;
```

Figura 1. Estruturas do Servidor

Este servidor é capaz de suportar um número indefinido de clientes.

2.2 Funcionalidades

O servidor tem duas funcionalidades essenciais, executar as transformações de compressão e cifragem sobre o input dado, gerando o ficheiro tal como o cliente deseja e informar o cliente do seu estado.

Para isso ser possível, tudo começa com uma mensagem do cliente para o servidor através de um pipe com nome com os argumentos dados no cliente. De seguida, o servidor, através de parsing, popula a estrutura global *process* com as informações do processo-filho que vai tratar do pedido do cliente e o pedido em si.

No caso do pedido ser *proc-file*, o processo-filho verifica se é possível começar a executar, caso contrário, fica à "espera"até haver disponibilidade de todas as transformações referidas no pedido, esta "espera"será melhor explicada mais abaixo. Estas transformações são executadas através de pipes de processos-netos que comunicam entre si.

No caso do pedido ser *status*, é enviado toda a informação presente da variável global *transformations*.

```c
void procfile(char* buffer, int server_client_fifo)
{
    char* args = buffer + 10; // removes proc-file and space

    /*
    * PARSING ARGS
    */

    char* token = strtok(args," ");

    char* input_files = malloc(sizeof(char) * 50);
    char* output_files = malloc(sizeof(char) * 50);

    strcpy(input_files, token);
    token = strtok(NULL, " ");
    strcpy(output_files, token);
    token = strtok(NULL, " \0");
    char* transformations[MAX_TRANSF];
    int i = 0;

    while(token != NULL)
    {

        transformations[i] = malloc(sizeof(char) * strlen(token));
        strcpy(transformations[i], token);
        token = strtok(NULL, " \0");
        i++;
    }
    transformations[i] = '\0';
    free(token);

    /*
    * APPLY TRANSFORMATIONS
    */

    transform(input_files, output_files, transformations);

    for(int j = 0; j < i; j++){
        free(transformations[j]);
    }

    write(server_client_fifo,"concluded\n", 10);

}
```

Figura 2. Tratamento do pedido proc-file

```c
void getStatus(int server_client_fifo)
{
    char* buff = malloc(sizeof(char)*50*MAX_TRANSF);
    for (int i = 0;i < MAX_TRANSF; ++i)
    {
        char temp[30];
        sprintf(temp,"transf %s: %d/%d (running/max)\n",transf[i].name,transf[i].running,transf[i].max);
        strcat(buff,temp);
    }

    write(server_client_fifo,buff,strlen(buff));
}
```

Figura 3. Tratamento do pedido status

A "espera"anteriormente abordada pode ser explicada da seguinte forma:

Ao iniciar o servidor é criado um processo-filho que lida com a ordem dos pedidos, isto é, tem o mesmo comportamento que um FIFO, a primeiro pedido à espera de recursos, será o primeiro a ser executado.

3 Cliente

3.1 Estrutura

A estrutura do Cliente é constituída por strings que enviam e pedem informação para o Servidor, o pedido de compressão e filtragem de ficheiros e pedidos sobre o estado da execução destes. Também é constituído por um pipe com nome *connection_fifo* onde é enviado o seu pid e o pedido.

3.2 Funcionalidades

O Cliente apenas tem o papel de pedir ao Servidor informação (*proc-file*) ou pedido de compressão e cifragem de um ficheiro (*proc-file*) através do pipe com nome criado.

4 Decisões tomadas

Decidimos que o cliente apenas cria um pipe com nome para enviar toda a informação pertinente para o servidor e de seguida, conecta-se a outro pipe com nome criado pelo servidor onde recebe informações do estado do seu pedido.

A comunicação entre o processo pai e processo-filho é feita através do sinal SIGCHILD que nos ajuda a atualizar o número de transformações que estão a ser feitas e também na espera de recursos necessários para o pedido ser executado.

5 Conclusão

Este trabalho de Sistemas Operaticos permitiu-nos consolidar tudo o que aprendemos nesta UC. A utilização de pipes e sinais ajudou-nos a entender melhor a importância destes.

Sentimos que a maior dificuldade foi a procura de uma estratégia de comunicação entre processos e a implementação de sinais. Infelizmente não foi possível implementar funcionalidades avançadas devido ao tempo elevado gasto na implementação das funcionalidades básicas.

Em suma, sentimos que este trabalho reflete parte da nossa aprendizagem em relação a esta UC.