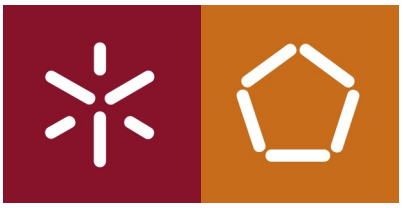
Universidade do Minho

Mestrado Integrado em Engenharia Informática



Sistemas Operativos
Trabalho Prático - Grupo 11

Sistema Operativos



Armando Silva
a87949

Gonçalo Soares
a84441

Pedro Novais
a78211

Junho 2021

Conteúdo

1 Introdução

Este projeto consistiu na criação de um serviço capaz de transformar ficheiros de áudio por aplicação de uma sequência de filtros permitindo também consultar o seu estado.

Inicialmente, o maior desafio que encontramos neste trabalho é tornar o servidor concorrente, ou seja, capaz de albergar vários clientes.

Este serviço é constituído por um servidor e múltiplos clientes que comunicam por pipes com nome, sendo que o cliente envia comandos para o servidor e o servidor o output para o cliente.

2 Servidor

2.1 Estrutura

O servidor é constituído por duas estruturas essenciais. A primeira é uma estrutura responsável por armazenar o informação com a configuração do servidor sendo útil para saber o nome e número dos filtros e os executáveis correspondentes.

A outra estrutura referenciada tem como funcionalidade tratar cada um dos processos que chegam ao servidor. Armazena então o pid do processo, o texto enviado pelo cliente, os filtros que estão associados ao pedido do cliente e por último e o mais importante, o pid associado ao cliente que vai permitir a criação do pipe com nome responsável pela comunicação entre o seridor e aquele servidor em específico tal como a imagem seguinte mostra :

```
//struct aurrasconfig
struct aurrasdconfig {
    char* name;
    char* exe;
    int max;
} *AurrasdConfig;

//struct processos
struct process{
    int pid;
    char* message;
    char** filtros;
    int pidFifo;
} *Process;
```

Estruturas

De seguida com as estruturas criadas há a necessidade de ter várias, sendo que assumimos que o servidor no máximo aguenta com 2048 processos, sendo por isso o número mais presente no tamanho dos arrays. Em suma, os arrays são todos responsáveis por aplicar as estruturas anteriores aos 2048 processos. Por fim, há dois inteiros responsáveis por armazenar o número de processos já ocorridos e o número de clientes conectados.

2.2 Funcionalidades

O servidor tem duas funcionalidades essenciais, executar filtros sobre o ficheiro input, gerando o ficheiro tal como o cliente deseja e informar o cliente do seu estado.

Para isso ser possível, tudo começa com o parsing da mensagem do cliente sendo que a primeira distinção começa se a mensagem contém um *transform* ou um

```c
struct process processes[2048];

struct aurrasdconfig aurray[5];

char* filtros_array[2048];
int exitStatus[2048];
int exitMatrix[2048][3];
int pids[2048][3];
int lastProcess = 0;

char* fifos[10];
int lastFifo = 0;
```

Arrays

status. Caso seja para processar o áudio, a informação é adicionada aos arrays em cima referidos para termos controlo quer do que o pedido deseja tal como a sua informação para saber para que pipe com nome será enviado a resposta. Há duas funções a destacar sendo a primeira responsável pelo parsing do pedido e trabalhar com estruturas e arrays :

```c
if(strncmp(buffer,"transform",9) == 0)
{
    processes[lastProcess].message = strdup(buffer);

    array[io] = strtok(buffer," ");

    while(array[io] != NULL){
        array[++io] = strtok(NULL," ");
    }
    io--;
    char** message_filters = (char**) malloc(3 * sizeof(char*));

    for (int j = 0; j < io-3; j++)
        message_filters[j] = strdup(array[j+3]);

    processes[lastProcess].filtros = malloc(sizeof(char*) * (io-3));

    for (int j = 0; j < io-3;j++)
        processes[lastProcess].filtros[j] = strdup(message_filters[j]);


    if(checkConfig(processes[lastProcess].filtros) == 0){
        processes[lastProcess].pidFifo = pidFifo;
        exitStatus[lastProcess] = WAITING;
        alarm(1);
    }
    else
        exitStatus[lastProcess] = EXECUTING;
}
```

Parsing Pedido

A outra função a destacar é uma função que recebe os filtros enviados pelo cliente para executar e executa consoante o desejado :

```c
void exec_filtros(char* filtro){

    char echo[100];
    sprintf(echo,"%s/aurrasd-echo",aurrasd_filters);

    char doubl[100];
    sprintf(doubl,"%s/aurrasd-gain-double",aurrasd_filters);


    char half[100];
    sprintf(half,"%s/aurrasd-gain-half",aurrasd_filters);

    char temp_double[100];
    sprintf(temp_double,"%s/aurrasd-tempo-double",aurrasd_filters);


    char temp_half[50];
    sprintf(temp_half,"%s/aurrasd-tempo-half",aurrasd_filters);

    if (strcmp(filtro,"echo") == 0)
        execl(echo,"aurrasd-echo", NULL);
    if (strcmp(filtro,"alto") == 0)
        execl(doubl,"aurrasd-gain-double", NULL);
    if (strcmp(filtro,"baixo") == 0)
        execl(half,"aurrasd-gain-half", NULL);
    if (strcmp(filtro,"rapido") == 0)
        execl(temp_double,"aurrasd-tempo-double", NULL);
    if (strcmp(filtro,"lento") == 0)
        execl(temp_double,"aurrasd-tempo-half", NULL);
}
```

Parsing Pedido

3 Cliente

3.1 Estrutura

A estrutura do Cliente é constituída por strings que enviam e pedem informação para o Servidor. O pedido de filtragem de áudios e pedidos sobre o estado da execução da filtragem de áudio. Também é constituído por um pipe com nome *cliente-servidor* único, que é usado para qualquer cliente comunicar com o servidor e vários *named pipes* servidor-cliente para o servidor distinguir com qual cliente ele está a comunicar, sendo o nome de cada pipe o seguinte: *server_client_fifo_pid*, onde pid é o process id do cliente.

```c
char csf[30];
char scf[30];
```

Strings com nomes dos pipes a abrir

```c
sprintf(scf,"%s%d","server_client_fifo_", pid_client);
```

Named pipe com pid

3.2 Funcionalidades

O Cliente apenas tem o papel de pedir ao Servidor informação (Status) ou mandar executar filtragens (Transform) através dos named pipes criados.

3.3 Decisões tomadas

Decidimos que o nosso servidor suporta a execução de 2048 processos. Podiamos ter usado alocação dinâmica para suportar a execução de mais processos, no entanto, achamos que 2048 é um número substancial para o projeto. Cada mensagem enviada ao servidor pode conter, no máximo, 4096.

Os Named Pipes são criados tanto pelo servidor como pelo Cliente. De forma a que haja comunicação concorrente.

Usamos sinais como SIGCHILD e SIGALARM de maneira a manipular o estado dos processos e a espera dos mesmos quando não há recursos suficientes fornecidos pelo Servidor, respetivamente.

Os pipes com nome responsáveis pela primeira interação com o cliente são criados no servidor e apenas irão servir para a primeira interação sendo que a partir daí a comunicação irá-se dar por um pipe com nome criado no cliente, tendo como nome o seu pid.

4 Conclusão

Este trabalho de Sistemas Operativos permitiu-nos consolidar tudo o que aprendemos nesta UC. A utilização de pipes e sinais ajudou-nos a entender melhor o grande papel que estas apresentam.

Sentimos que a maior dificuldade foi a aplicação da execução concorrente de pipes com nomes, e ainda apresenta alguns bugs onde não encontramos tempo para resolver.

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