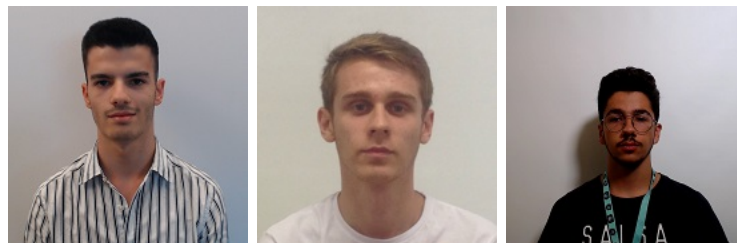


Universidade do Minho

Aurras: Processamento de Ficheiros de Audio

Trabalho Prático de Sistemas Operativos

Mestrado Integrado em Engenharia Informática



Duarte Moreira A93321

João Amorim A74806

Lucas Carvalho A93176

02 de Maio de 2021

1 Introdução

No âmbito da Unidade Curricular de Sistemas Operativos, foi-nos proposta a criação de um serviço com a capacidade de transformar ficheiros de áudio através da aplicação de um ou mais filtros em sequência.

Este serviço deverá ser baseado numa arquitetura Cliente-Servidor, sendo que o Servidor tem que suportar o envio e processamento de um pedido por parte de um ou mais Clientes.

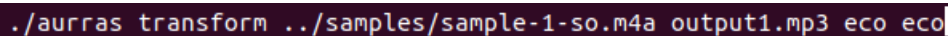
Este relatório tem como objetivo descrever toda a aplicação, fazendo uma apresentação inicial das funcionalidades a implementar, tanto do Cliente como do Servidor, apresentar o caminho e as ideias seguidas para as implementações que dão resposta a essas mesmas funcionalidades e finalmente apresentar os resultados finais.

2 Descrição do Projecto

Tal como já foi mencionado, neste trabalho é necessário a criação de um Servidor e de um ou mais Clientes. De notar, que a comunicação feita entre Cliente e Servidor deverá ser feito com o recurso a Named Pipes.

2.1 Cliente

No Cliente, devem ser suportadas as funcionalidades de envio de pedidos para o Servidor, sendo que estes pedidos possuem um de dois tipos, "status" ou "transform". O pedido de "transform", deve enviar como argumentos o ficheiro de input, ou seja, o ficheiro a transformar por aplicação dos filtros, o ficheiro output que corresponderá ao ficheiro final após transformação, seguido dos nomes dos vários filtros.



```
./aurras transform ../samples/sample-1-so.m4a output1.mp3 eco eco
```

Figure 1: Exemplo de comando "transform"

Como é possível observar na figura, neste pedido do Cliente, irão ser aplicados os filtros "eco" e "eco" ao ficheiro "sample-1-so.m4a." No final, após aplicação dos filtros, deverá ser devolvido o ficheiro com o nome "output1.mp3". É importante salientar que neste projeto existe uma limitação do número de instâncias para cada filtro, ou seja, se pedido A e pedido B necessitarem ambos da aplicação de um filtro "eco" e o limite de filtros "eco" a correr ao mesmo tempo for de 1, então se o pedido A já estiver em processamento, o pedido B terá que aguardar que o pedido A acabe a aplicação do filtro "eco" e liberte os recursos, ou seja, que indique ao Servidor que o filtro "eco" já tem uma instância livre.

Por outro lado, o Cliente deve ainda ser capaz de enviar pedidos do tipo "status" para o Servidor. Assim que o Servidor receba esta pedido, deverá enviar de volta, para o Cliente que efetuou o pedido, uma resposta contendo todas as tarefas em processamento, assim como o número de filtros em uso, para cada tipo de filtro.

2.2 Servidor

Em relação ao Servidor, este deverá ser capaz de estar constantemente à escuta de novos pedidos vindos dos vários Clientes, pedidos estes que correspondem aos descritos na secção anterior.

É então aqui que serão recebidos e processados os pedidos de transformação dos ficheiros de áudio. Inicialmente, o Servidor deverá ser executado com o nome do ficheiro de configuração e com o "path" para a pasta dos filtros:

```
./aurrasd ../etc/aurrasd.conf aurrasd-filters
```

Figure 2: Execução do Servidor

Em relação ao ficheiro de configuração, é a partir deste que iremos obter a informação sobre o número de instâncias máximas que existirão para cada filtro.

3 Arquitetura da Aplicação

Nesta secção, irão ser apresentadas as soluções obtidas, tanto para o Cliente como para o Servidor, para dar resposta às várias funcionalidades, assim como as justificações para as mesmas.

3.1 Cliente - Aurras

Para o Cliente, teríamos que dar resposta a 3 tipos de comandos, um "transform", um "status" e outro apenas com o executável.

Para isto, é aberto o Named Pipe de escrita para o Servidor, por onde serão mandados os comandos do Cliente. De notar que para cada Cliente, é também criado um Named Pipe com o nome "fifo[pid]" onde [pid] corresponde ao pid do processo do Cliente obtido com o getpid(). Isto é necessário uma vez que o Servidor terá que enviar respostas para um determinado Cliente e desta maneira, saberá sempre qual o Named Pipe que terá que utilizar para enviar essa resposta.

```
pid_t pid = getpid();
char pipe_name[17];
sprintf(pipe_name, "../tmp/fifo%d", pid);

if(mkfifo(pipe_name, 0666) < 0){
    perror("mkfifo");
}
```

Figure 3: Criação do Named Pipe do Cliente de acordo com o Pid

Em relação aos comandos, no caso do "transform", era necessário remover o primeiro argumento, correspondente ao executável e fazer o tratamento dos restantes argumentos. Quanto aos restantes, o segundo corresponde ao comando em si, neste caso "transform", o terceiro e o quarto aos ficheiros de input e de output, respetivamente, e todos os restantes que correspondem aos nomes dos filtros.

```
}else if(argc >= 2 && strcmp(argv[1],"transform")==0){  
    for (int i=1; i<argc; i++) {  
        strsize += strlen(argv[i]);  
        if (argc > i+1)  
            strsize++;  
    }  
    char *cmdstring;  
    cmdstring = malloc(strsize+1);  
    cmdstring[0] = '\0';  
  
    for (int i=1; i<argc; i++) {  
        strcat(cmdstring, argv[i]);  
        if (argc > i+1)  
            strcat(cmdstring, " ");  
    }  
  
    cmdstring[strsize]='\n';  
  
    write(client_to_server,cmdstring,strsize+1);  
}
```

Figure 4: Código do Cliente para o comando transform

No que diz respeito ao comando "status", o tratamento feito é semelhante, com a diferença que desta vez é enviado o pid do Cliente na mensagem para que do lado do Servidor este abra o Named Pipe correspondente ao Cliente que envia a mensagem, para que o Servidor envie então a resposta ao "status".

Finalmente, o comando do Cliente sem argumentos, imprime apenas no ecrã informações de execução para o Cliente.

```

}else if(argc >= 2 && strcmp(argv[1],"transform")==0){
    for (int i=1; i<argc; i++) {
        strsize += strlen(argv[i]);
        if (argc > i+1)
            strsize++;
    }
    char *cmdstring;
    cmdstring = malloc(strsize+1);
    cmdstring[0] = '\0';

    for (int i=1; i<argc; i++) {
        strcat(cmdstring, argv[i]);
        if (argc > i+1)
            strcat(cmdstring, " ");
    }

    cmdstring[strsize]='\n';

    write(client_to_server,cmdstring,strsize+1);
}

```

Figure 5: Código do Cliente para o comando status

3.2 Servidor - Aurrasd

3.2.1 Pedidos do Cliente

O Servidor é então o responsável por processar os pedidos que vêm dos vários Clientes. Tal como já mencionado, estes pedidos correspondem ao "status" e ao "transform".

Em ambos o caso, é feito o parsing do pedido, na main, e guardado num array de strings. No "status", é utilizado o valor na segunda posição do array, correspondente ao PID do Cliente que enviou o pedido, para a abertura do Named Pipe para onde se vai escrever a resposta para o Cliente. Assim é garantido que a resposta vai para o Cliente em questão.

Para o "transform", depois de efetuado o parsing, é chamada a função processTask() que recebe como argumento o array de strings resultante do parsing.

3.2.2 Processamento dos Pedidos

Antes de mais e para dar resposta aos problemas apresentados, foram criadas as seguintes estruturas:

```
typedef struct running_filter{
    char name[7];
    pid_t pid;
} RunningFilter;

typedef struct filter{
    char name[7];
    int max_running;
    int running;
} Filter;
```

Figure 6: Estruturas

A primeira, contém o nome do filtro e o pid que lhe é atribuído no momento da execução do processo filho, responsável por tratar desse mesmo filtro. A segunda é utilizada para que no momento de leitura do ficheiro de configuração, se criem os filtros existentes, com o nome dos mesmos e o número de instâncias máximas que estes podem correr. Aqui também será mantido o número de filtros que estão a correr no momento, desse tipo de filtro.

Adicionalmente, são criados 2 arrays, em que o primeiro contém todos os filtros lidos do ficheiro de configuração e o segundo contém todos os pids dos processos em execução:

```
Filter allFilters[MAX_FILTERS];
```

```
RunningFilter runningFilters[MAX_RUNNING_FILTERS];
```

Para o processamento de pedidos, era necessário ter em conta certos aspetos, nomeadamente a execução de pedidos com aplicação de vários filtros, a execução e processamento de pedidos concorrentes e os limites de instâncias dos filtros do mesmo tipo que podem estar a correr ao mesmo tempo.

Em relação ao primeiro e segundo caso, recorreremos ao uso de pipes anónimos, sendo criado um processo filho para a execução de cada filtro. Recorrendo ao redirecionamento dos descritores de ficheiros, o primeiro processo recebe como input o ficheiro a transformar e o último escreve o resultado final da transformação no ficheiro output. De notar que ambos estes ficheiros input e output, são os mesmos que vêm no pedido de "transform" do Cliente. Uma vez que a execução dos filtros é feita por processos filhos, não existe qualquer problema de concorrência, sendo que podemos executar

quantos pedidos quisermos, desde que o pedido não necessite de mais filtros do que aqueles que se encontram disponíveis.

Em relação ao número máximo de instâncias de filtros a correr, este foi sem dúvida a parte mais trabalhosa do trabalho. Aquilo que o grupo decidiu fazer foi uma função chamada `reapRunningTasks()` que é chamada em pontos estratégicos, nomeadamente assim que recebe um comando do Cliente e no momento de execução da função `processTask()`.

O objetivo desta função, é percorrer o array `runningFilters` e verificar se existem processos que terminaram. Caso tenham terminado, é decrementada a variável `running` do filtro em específico de `allFilters`, que contém o número de instâncias a correr desse filtro, e o `pid` da posição de `runningFilters` que obtemos passa novamente a -1. Desta maneira, quando na função `processTasks()` for feita a verificação ao filtro num ciclo, correspondente ao guardado em `allFilters()`, para ver se existem filtros suficientes para responder ao pedido do Cliente, iremos garantir que existe um mecanismo que vai estar constantemente a fazer o Reap dos processos que terminaram e desta maneira abrir espaço para que os pedidos que estão à espera da libertação de filtros podem ser processados normalmente.

Na primeira imagem, podemos ver a função `reapRunningTasks()`, enquanto que na segunda podemos ver o ciclo na função `processTask()` onde é feita a espera do pedido que necessita de mais filtros livres para poder ser processado:

```
void reapRunningTasks(){
    for(int i=0;i<MAX_RUNNING_FILTERS;i++){
        if (runningFilters[i].pid!=-1){

            pid_t pid = waitpid(runningFilters[i].pid, NULL, WNOHANG);

            if (pid>0){
                int f = findFilter(runningFilters[i].name);
                printf("Reap: %d %s\n", i, runningFilters[i].name);
                allFilters[f].running--;
                runningFilters[i].pid=-1;
            }
        }
    }
}
```

Figure 7: Função `reapRunningTasks()`


```

for(int i=0;i<numFilters;i++){//percorrer todos os filtros necessarios
    int f = findFilter(filters[i]);
    int maxWait=MAX_WAIT_TIME_FOR_FILTER;
    while(maxWait>0 && allFilters[f].running==allFilters[f].max_running){
        printf("WAITING for %s %d\n",filters[i],maxWait);
        reapRunningTasks();
        sleep(1);
        maxWait--;
    }
    if (maxWait==0){
        abort=1;
        //nao se conseguiu o filtro em tempo util
        //libertar todos e cancelar a tarefa

        for(int j=0;j<i;j++){
            allFilters[findFilter(filters[j])].running--;
        }
        break;
    }

    allFilters[f].running++;
}

```

Figure 8: Ciclo da função processTask()

4 Resultados Finais

De seguida apresentam-se algumas imagens de testes feitos, sendo que a primeira mostra a execução do comando status e consequentemente o número máximo de instâncias de cada filtro:

```
joao@joao-VirtualBox:~/Desktop/grupo-104/bin$ ./aurras status
filter alto: 0/2 (running/max)
filter baixo: 0/2 (running/max)
filter eco: 0/3 (running/max)
filter rapido: 0/2 (running/max)
filter lento: 0/3 (running/max)
Pid do Cliente: 5291█
```

Figure 9: Execução do comando status

```
Cmd: transform ../samples/sample-1-so.m4a output2.mp3 rapido alto eco
Cmd: transform ../samples/sample-1-so.m4a output1.mp3 eco rapido baixo
Reap: 0 rapido
Reap: 3 eco
Reap: 1 alto
Reap: 4 rapido
Reap: 2 eco
Reap: 5 baixo
```

Figure 10: Execução de dois pedidos concorrentemente, sem espera por filtros

```

Cmd: transform ../samples/sample-1-so.m4a output1.mp3 eco rapido baixo eco
Cmd: transform ../samples/sample-1-so.m4a output2.mp3 rapido alto eco eco eco
WAITING for eco 15
WAITING for eco 14
WAITING for eco 13
Reap: 0 eco
WAITING for eco 15
WAITING for eco 14
WAITING for eco 13
WAITING for eco 12
Reap: 1 rapido
WAITING for eco 11
WAITING for eco 10
WAITING for eco 9
WAITING for eco 8
Reap: 2 baixo
WAITING for eco 7
WAITING for eco 6
WAITING for eco 5
WAITING for eco 4
Reap: 3 eco
Reap: 0 rapido
Reap: 1 alto
Reap: 2 eco
Reap: 3 eco
Reap: 4 eco

```

Figure 11: Execução de dois pedidos concorrentemente, com espera por filtros

Como podemos observar na primeira imagem, o número máximo de instâncias para o filtro "eco" é de 3. Como o primeiro pedido precisa de processar 2 filtros "eco", o segundo pedido que necessita de 3 filtros "eco" livres vai ter que esperar, tal como acontece na figura.

Para finalizar, é importante mencionar que ao ser feito um pedido de "transform" ao servidor, no final do processamento do pedido, o Cliente não fecha, o que faz com que o comando "make test" não funcione, uma vez que este precisaria que cada Cliente encerrasse. No entanto, o comando "make" cria todos os ficheiros que deve criar sem qualquer problema. Desta maneira, todo o projeto foi feito de maneira a que os executáveis fossem executados a partir da pasta bin, o que significa que a maneira como os paths são introduzidos, terá que ser de maneira diferente. No caso de um pedido de "transform" o Cliente deverá efetuar um pedido da seguinte maneira:

```

joao@joao-VirtualBox:~/Desktop/grupo-104/bin$ ./aurras transform ../samples/sample-1-so.m4a ../tmp/output1.mp3 rapido eco alto

```

Figure 12: Execução do comando transform pelo Cliente

Para o caso do Servidor, o comando para o iniciar deverá ser:

```
~/Desktop/grupo-104/bin$ ./aurrasd ../etc/aurrasd.conf aurrasd-filters
```

Figure 13: Execução do comando para iniciar o Servidor

5 Conclusão

Neste trabalho, abordamos os vários temas lecionados ao longo do semestre na disciplina de Sistemas Operativos, sendo cada um deles crucial para a correta implementação de toda a aplicação, uma vez que sem conhecimentos em algum deles, o resultado final não seria funcional.

Sem dúvida que a funcionalidade que nos deu mais trabalho foi a da implementação de uma limitação ao número de instâncias a correr para cada filtro, sendo que de resto, achamos o projeto dentro do esperado.

Uma vez finalizado o trabalho, fazemos um balanço positivo da solução desenvolvida, visto que cumprimos com os requisitos mais importantes que nos foram apresentados no enunciado e vemos como trabalho futuro a evolução do Cliente, de dar resposta a todos os pontos visados no enunciado.