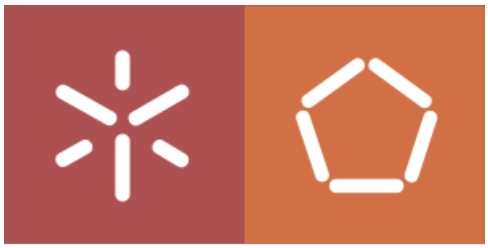


Universidade do Minho
Escola de Engenharia

Sistemas Operativos

Trabalho Prático

Conversor toml2json

Grupo 2

Hugo Reinolds A83924 Rita Lopes A8111 Sonia Pereira A81329

17 de janeiro de 2021

MIETI - 3º Ano - 1º Semestre

Conteúdo

Capítulo 1

Introdução

Este relatório inicia-se com uma breve contextualização, seguindo-se a declaração deste trabalho e em que é que este consiste. O principal objetivo deste trabalho consiste na implementação de um serviço capaz de transformar ficheiros de áudio através do uso de uma sequência de filtros. Para álem da submissão de pedidos o servidor também terá de ter a capacidade de de demonstrar o número de filtros livre e em uso, como também a consulta das tarefas em execução.

Capítulo 2

Premissas

Na criação da struct que iria ser utilizada para o Cliente comunicar com o Servidor, na qual iriam ser enviadas as informações relativas ao pedido (Request), obtivemos problemas ao enviar a struct através dos pipes com nome, já que inicialmente declaramos a struct com um tamanho variável, no entanto isso levou a termos problemas em relação à leitura, visto que o Server nunca sabia quantos bytes tinha de ler. Para ultrapassarmos este problema decidimos assumir que tanto o input-filename, o nome do ficheiro final, o número de transformações e o nome das transformações iriam ter todos um tamanho limite. Desta forma, assumimos que o nome do ficheiro (e o seu diretório) têm no máximo 255 carateres, o nome do ficheiro têm no máximo 255 carateres, e assumimos também um máximo de 10 transformações, cada uma destas tendo um nome inferior a 256 carateres. Desta forma foi possivel o envio da struct de uma forma mais linear.

Capítulo 3

Arquitetura

3.1 Estrutura de Dados

De seguida, seram apresentadas algumas das nossas estruturas de dados mais relevantes:

3.1.1 Request

```
typedef struct Request {
    int code;
    int n_transformations;
    int pid;
    char id_file[256];
    char dest_file[256];
    char transformations[10][256];
} Request;
```

Utilizamos esta estrutura para representar um pedido por parte do Cliente:

1. **code** - este é o codigo no qual dependo do do valor representa um tipo de pedido por parte do Cliente. Neste caso esta variável apenas irá variar entre dois valores 0 e 1 no qual 0 corresponde ao pedido de requisição do estado dos filtros e das tarefas a executar e 1 o pedido de requisição para executar uma tarefa.
2. **n_transformations** - Esta variável corresponde ao número de transformações de um pedido.
3. **pid** - Esta variavel corresponde ao pid associado a cada pedido.
4. **id_file** - Este é o nome correspondente o ficheiro no qual irá ser executado transformações.
5. **dest_file** - Este é o nome do ficheiro final para onde irá ser transportado logo da finalização das transformações
6. **transformations** - Esta variável corresponde a lista de transformações que um pedido irá requisitar

3.1.2 Filter

```
typedef struct Filter{
    char name[256];
    char command[256];
    int quantity;
    int in_use;
}Filter;
```

Utilizamos esta estrutura para respresentar um filtro e as suas caracteristicas:

1. **name** - Esta variável corresponde ao nome do filtro associado.
2. **command** - Esta corresponde ao comando associado a este filtro
3. **quantity** - Esta representa a quantidade total de filtros deste tipo.
4. **in_use** - Esta representa o número de filtros a serem ocupados deste tipo.

3.1.3 InProgress

```
typedef struct InProgress {
    int task_nr;
    int pid; // pid of process that requested the transformation
    int done_transformations;
    int n_transformations;
    char origin_file[256];
    char dest_file[256];
    char task_array[10][256];
    int pid_array[10];
    int pipe_matrix[10][2];
    struct InProgress* next;
} InProgress;
```

Utilizamos a estrutura InProgress para respresentar um Request no qual está executar um pedido :

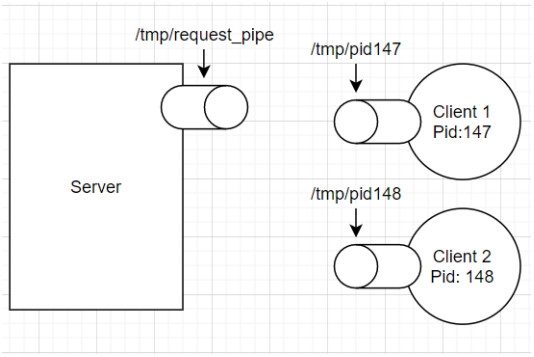
1. **task_nr** - Esta variavel corresponde so número da tarefa.
2. **pid** - Corresponde ao pid do precesso que requesitou a transformação.
3. **done_transformations** - É o número de transformações finalizadas nno pedido.
4. **n_transformations** - É o número de transformações requisitadas para este pedido.
5. **origin_file** - Este é o nome correspondente o ficheiro no qual irá ser executado transformações.
6. **dest_file** - Este é o nome do ficheiro final para onde irá ser transportado logo da finalização das transformações.

7. **task_array** - Corresponde ao array de nomes das transformações a serem executadas neste pedido.
8. **pid_array** - É o array de pids que correponde cada transformação.
9. **pipe_matrix** - É a matrix onde iremos criar os pipes.

3.2 Arquitetura

O Sistema é constituído por dois programas principais.

- Servidor
- Cliente



3.2.1 Cliente

No Cliente irão ocorrer três tipos de situações dependendo do número de argumentos que irão ser passados na linha de comando.
Se n (número de argumentos) for 0, é imprimida uma mensagem no terminal do cliente, na qual indica que não foi selecionada nenhum tipo de ação, e mostra ao cliente como utilizar o programa.
No caso de n == 1 e o argumento dado for a string "status", cria um pedido de estado, que é a estrutura *Request*, com o atributo code = 0 e o atributo n_transformations recebendo o pid do cliente, de modo a que o servidor saiba a que cliente devolver o estado. Depois de enviado o pedido, o cliente lê do pipe (bloqueia até que a resposta seja enviada pelo servidor), e depois de recebida, o conteúdo da mensagem é imprimido no terminal seguido do fecho dos pipes, e do dim da execução do cliente.

No último caso em que n >1, no qual o primeiro argumento irá corresponde à string "transform", o segundo corresponde ao ficheiro ao qual o cliente pretende aplicar transformações, e o terceiro argumento representa o ficheiro destino, o qual irá ser o resultado do ficheiro inicial depois de executadas todas as transformações. Os argumentos com o seu indíce compreendido entre 5 >= x >= 14, são correspondentes aos nomes dos filtros que o cliente pretende que sejam aplicados ao ficheiro que pretende transformar. Depois de criado o pedido com as informações acima descritas, o cliente comporta se de forma análoga ao que foi descrito no caso anterior, procedendo ao envio do pedido, e depois à

leitura do pipe, à espera de uma resposta, sendo que quando essa chega, o seu conteúdo é imprimido no ecrã, os seus pipes fechados, e o programa terminado.

3.2.2 Servidor

O Servidor é o local onde irá ocorrer a maior parte do trabalho. No Servidor existem 3 tipos de processos diferentes, sendo estes: processo pai, processo status e o processo transform. O processo pai consiste em ler continuamente o Request Pipe até receber 1 de 2 tipos de pedidos, Status ou Transfom.

Status Replier

Se for um pedido Status, o pai irá criar um filho que irá executar a função **status_replier** que consiste em demonstrar ao cliente o estado atual do Servidor, ou seja, demonstrando os pedidos que o Servidor está executando como também a quantidade de filtros que estão sendo utilizados de momento.

Transform

Se for um pedido Transform este executa uma verificação da existência dos filtros requisitados. Se aprovada é iniciada uma nova verificação agora referente à disponibilidade dos filtros. Se algum dos filtros não estiver disponivel é iniciado um ciclo while no qual espera 5 segundos, e ao fim deles, este volta a verificar a disponilidade dos filtros.

Este sono pode também ser interrompido pelo terminar da executação de um dos filhos que anteriormente tenha sido criado. Se este filho era responsável pela aplicação de algum filtro, os recursos que estavam a este associados, são então libertados. Se este filho, era também responsável pela execução do filtro que, pela sua indisponibilidade estava a impedir o início da execução do pedido que se encontra em lista de espera, este poderá então começar a sua execução. Se o filtro corresponder ao último de uma tarefa, esta é também removida da lista de tarefas que se encontram em execução.

Dispatch

Nesta função, também ela executada pelo pai, são criados os filhos responsáveis pela execução dos filtros. Primeiramente são alocados os recursos através de um aumento da variável in_use dos filtros pretendidos, depois são iterados os filtros pedidos, mais precisamente no task_array, que se encontra na estrurura InProgress, conforme visto acima. Assumamos então que, no task_array, se encontram n transformações.

A cada iteração, é primeiramente verificado se o filtro pedido, é o único a constituir o pedido, ou seja, n==1. Sendo esse o caso, esta iteração será a única a ser realizada, e o filho criado nela terá que executar todos os redirecionamentos necessários, de modo a garantir a correta execução do filtro. Esse filho terá então de, em primeiro lugar, redirecionar o conteúdo do ficheiro áudio a ser processado, para o seu stdin, e depois terá de criar o ficheiro resultante da aplicação dos filtros ao ficheiro alvo, redirecionando também o conteúdo do seu stdout para o ficheiro destino. Tendo feito estas tarefas, o filho pode então

executar o exec.

Sendo n >= 2, a execução é ligeiramente diferente, já que esta requer pipes anónimos para garantir o transporte do conteúdo resultante da aplicação de um filtro sobre um ficheiro, por parte de um filho x, para o filho x+1.

O primeiro passo numa iteração it quando n>= 2 é a verificação se it >1. Sendo este o caso, foram já criados dois pipes anónimos (pipe_matrix[0] e pipe_matrix[1]), de modo a redirecionar o conteúdo modificado pelo filho it == 0 para o filho == 1, e do filho it == 1 para o filho que será criado agora. Torna-se necesário então o fecho de tanto o writing end, como o reading end do pipe_matrix[0], caso contrário, se estes se mantivessem abertos, iria levar ao bloqueio dos processos atuais. Depois de efetuada esta verificação e ambos os ends do pipe fechados, é verificado se it == n-1. Verificando-se esta condinção, o filho que será criado na atual iteração (it), corresponde ao filho que executará a aplicação do último filtro pedido sobre o ficheiro. Sendo este o caso, não será criado um pipe, visto que o resultado da execução deste filtro deverá ser escrito no ficheiro destino. Sendo it <n-1, será criado um pipe de modo a garantir o reenvio do resultado para o filho seguinte.

Sendo estas verificações realizadas, é então criado o filho, e um de três cenários acontece:

1. **it == 0:** Sendo este o primeiro filho a ser criado, este trata de redirecionar o conteúdo do ficheiro a ser processado para o seu stdin, tratando também de fechar o reading end do pipe_matrix[0], já que este não irá ler do mesmo. Trata também de redirecionar o conteúdo do seu stdout para o reading end do pipe_matrix[0], de modo a reencaminhá-lo para o filho it == 1. Depois do reencaminhamento, é fechado o reading end do pipe, e o filho encontra-se pronto para executar o exec.

2. **0 <it <n-1:** Neste caso, o filho criado, é um "filho do meio". Os filhos do meio tratam primeiramente, de fechar os ends que não iram utilizar, nomeadamente o writing end do pipe_matrix[it-1], já que não irá enviar conteúdo para o seu irmão precedente, e o reading end do pipe_matrix[it], já que não irá ler conteúdo desse. Feito isto, trata de redirecionar o conteúdo do reading end do pipe_matrix[it-1] para o seu stdin, e redirecionar o conteúdo do stdout para writing end do pipe_matrix[it], de modo a reencaminhar o resultado da aplicação do filtro it, para o filho it+1.

3. **it==n:** Sendo este o último filho a ser criado, trata primeiro de fechar o writing end do pipe_matrix[it-1]. Seguidamente, cria o ficheiro destino e redireciona do seu stdout para o ficheiro, sendo que depois fecha tanto o reading end do pipe_matrix[it-1] como o file descriptor do ficheiro que acabou de criar, estando então pronto.

Depois de concluída a iteração dos filtros do task_array, são fechados ambos os ends do pipe_matrix[n-1], caso estes existam. Finalmente, é adicionada a tarefa à lista de tarefas em progresso.

Capítulo 4

Conclusão

Nesta secção falaremos dos sucessos e adversidades ao longo do desenvolvimento do projeto. Um dos primeiros desafios deste trabalho foi na comunicação entre Servidor e Cliente, pois tendo chegado à conclusão de que teríamos que tomar proveito de várias estruturas de dados, na transferência das mesmas não conseguíamos efetuar uma comunicação correta para isso decidimos assumir valores na construção das estruturas, como explicado no Capítulo 2 (Premissas).

Outro problema, surgiu no contexto da execução de vários tipos de teste, de modo a melhor compreendermos a capacidade de carga suportada pelo programa. Nestes testes apercebemo-nos de que ao serem enviados vários pedidos com apenas um filtro, devido à elevada cadência de processos filho a terminarem a sua execução num curto intervalo de tempo, resultavam processos defuntos. Ao investigar uma possível solução para este problema, percebemos que este acontece quando o processo pai não efetua um "wait". Tentamos algumas soluções, sendo que nenhuma surtiu efeito, no entanto, apesar de crermos que uma possível solução passa por um maior controlo sobre o fim da execução de um processo filho, no entanto, não fomos capazes de solucionar o problema.

Geralmente, fazemos um balanço positivo em relação ao trabalho realizado visto que este apresenta uma solução razoavelmente robusta para o problema descrito, sendo capaz de realizar virtualmente todos os requisitos propostos pela esquipa docente, não tendo sido implementado o fim de execução "gracioso".