Relatório Trabalho Prático SO - Grupo 30

Duarte Afonso Freitas Ribeiro (A100764)
Francisco Macedo Ferreira (A100660)
Júlio José Medeiros Pereira Pinto (A100742)

13 de maio de 2023

Conteúdo

1 Introdução

Este relatório tem como intuito abordar o projeto da UC Sistemas Operativos do ano letivo 2022/2023.

Este relatório irá abranger as decisões tomadas pelo grupo tal como o método de raciocínio para desenvolvimento do mesmo. O objetivo do projeto é implementar um serviço de monitorização dos programas executados numa máquina.

2 Funcionalidades

De maneira a cumprir o trabalho na totalidade, decidimos implementar todas as funcionalidades: básicas e avançadas.

- **Execução de programas utilizador**
- **Consulta de programas em execução**
- **Servidor**
- **Execução encadeada de programas**
- **Armazenamento de informação sobre programas terminados**
- **Consulta de programas terminados**

3 Arquitetura

O objetivo da aplicação é seguir o modelo cliente/servidor. Temos o cliente que envia os pedidos ao servidor e o servidor que é responsável pelo processamento dos mesmos.

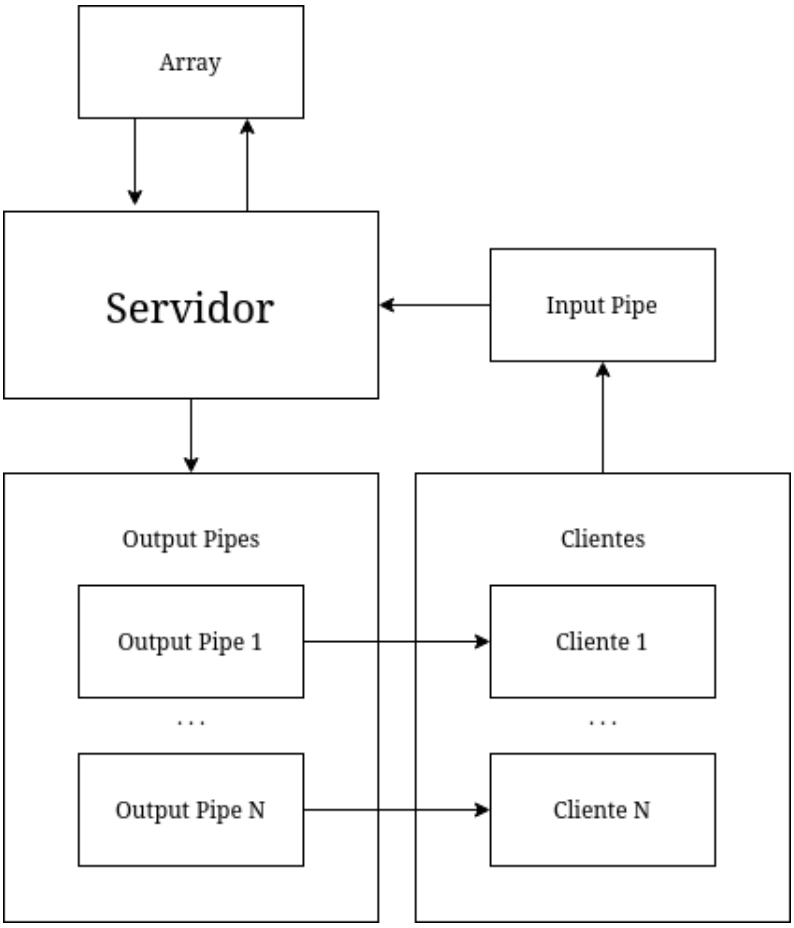

Figura 1: Diagrama de arquitetura

3.1 Servidor

O nosso servidor funciona da seguinte maneira: Para começar criamos o *input_pipe*, este é aberto e o servidor fica atento a *"ouvir"* se recebe algum *request*.

Quando este recebe um *request* o servidor vai processá-lo. Se for uma notificação de execução de um programa, o *request* é adicionado a um *array* onde os processos que estão a acontecer são guardados.

Após o processamento do *request*, quando aplicável, o servidor envia os resultados do *request* para o cliente, através de um *output_pipe* especifíco a este cliente. Este *output_pipe* é criado pelo cliente no início do *request* mas apenas é aberto quando é necessário.

3.2 Cliente

Todos os *requests* criados pelo cliente são enviados através do *input_pipe* do servidor e o cliente vai receber todas as informações geradas pelo servidor através do *output_pipe* criado pelo cliente no início do programa.

Levando isso em conta, o cliente consegue realizar diferentes tipos de *requests*. Este pode pedir um *execute*, este *execute* pode ser um único comando ou então comandos encadeados (flags *-u* e *-p* respetivamente), uma *pipeline*, simulando a funcionalidade de *piping* no Bash. Quando o *request* do *execute* é feito, este notifica o servidor do início do programa/programas e quando este acaba o servidor volta a ser notificado do término e o *output_pipe* do cliente é aberto de maneira a que o servidor possa enviar as informações geradas. Ainda mais, o cliente também pode pedir dados estatísticos ao servidor, estes dados são calculados na altura pelo servidor, desta maneira ao contrário dos *executes* não existe um início e um fim, assim sendo o cliente não tem de notificar o servidor, basta realizar o *request*. Tal como nos *executes* os dados são enviados para o cliente através do *output_pipe*.

4 Implementação

Ao longo do projeto ponderamos diversas maneiras de implementação, porém acreditamos que a maneira que temos será a melhor solução que conseguimos realizar. Para isto aplicamos conceitos de outras cadeiras, juntamente com os conceitos aprendidos na unidade curricular de Sistemas Operativos.

O nosso serviço funciona da seguinte forma. Primeiramente temos que iniciar o servidor e fornecemos como argumento o nome de uma pasta, a pasta em questão sendo para onde pretendemos enviar os ficheiros que guardam os *outputs* de programas terminados. Após isto qualquer cliente pode enviar um *request* ao servidor. O cliente abre um *output_pipe* e de seguida um *request* é enviado pelo *input_pipe* para o servidor. Caso este seja um *execute*, caso seja programa e não um *request* de estatísticas, este *request* é guardado num *array* de maneira permitir que este esteja a ser processado em algum lado. Depois de o *request* ser processado o *output_pipe* do cliente é aberto e por este é enviado os dados respetivos ao *request* para o cliente.

4.1 Requests

Para o cliente conseguir facilmente comunicar com o servidor utilizamos um *request*. Os nossos *requests* tem como base a seguinte estrutura:

```
typedef enum {
      UNKNOWN,
      FINISHED_EXEC,
      SINGLE_EXEC,
      PIPELINE_EXEC,
      STATUS,
      STATS_TIME,
      STATS_COMMAND
 } request_type;
```

```
typedef struct {
     request_type type;
     pid_t requesting_pid;
     pid_t child_pid;
     char program_name[NAME_MAX];
     struct timeval time;
} Request;
```

Ao longo do projeto pensamos em diversas maneiras de guardar os nossos *requests* e chegamos a esta *estrutura*. Deste modo temos o *request_type* que serve para identificar o tipo de *request* que está a ser feito ao servidor. Guardamos o nome do programa, o PID e o PID do filho. Por último

guardamos uma variável que indica os momentos de início e fim da execução de um programa nas requests do tipo *execute*. Este último só é populado e acessado no caso do *request* ser do tipo de execução. O *program_name* é utilizado enviar informação dos PIDs no caso dos comandos de estatísticas.

4.2 Gestor de Requests

De maneira a conseguirmos gerir os nossos *requests* da melhor maneira possível, face a maneira como estruturamos os nossos *requests*, temos uma função *handle_requests*. Esta função tem como objetivo, através do *request_type mapear* a maneira como os *requests* vão ser processados pelo servidor. Este é o que permite as diferentes formas de processamento dos programas para as estatísticas (dos *executes* para as *stats*). Através do *handle_requests* controlamos os dados que o servidor envia para o *output_pipe* e a maneira como estes são enviados.

4.3 Array

Quando um *request* é enviado ao servidor confirmamos o tipo do mesmo. Caso este seja um *execute* o servidor necessita guardar os dados dos mesmos em algum lado. Para isso criamos um *array*, este *array* é utilizado de maneira a que consigamos saber que quando um programa é executado este está a ser processado em algum lado. Para isso utilizamos a seguinte *struct*:

```
typedef struct {
    Request *array;
    int current_size;
    int max_size;
} Running_Programs;
```

Desta forma, quando o servidor é notificado do início do *execute* o *request* respetivo é adicionado ao *array*. Quando for notificado do fim do *execute* este é removido do *array*.

5 Outros

5.1 Makefile

De maneira a conseguirmos realiza o trabalho o melhor possível baseamos-nos bastante na Makefile fornecida pelos professores. Porém de maneira a nos permitir maior liberdade e um melhor desempenho fizemos algumas alterações a mesma, desde de adicionar modos de *debugging* a permitir que a mesma nos notificasse de forma mais clara o que está a acontecer.

5.2 Testes Unitários

Originalmente idealizamos realizar algum tipo de testes unitários. Porém ao desenvolver do projeto achamos que não seriam necessários, isto pois a grande parte do trabalho era fácil de testar e ao comparar alguns valores de desempenho apercebemo-nos que os valores tendem sempre a ser similares, deste modo achamos mais essencial focar no desenvolvimento do trabalho do que desenvolver testes.

6 Conclusão

Acreditamos que existem vertentes do projeto que poderiam ser melhoradas, desde de modularidade a outros conceitos como *clean code*. Porém acreditamos que conseguimos desenvolver o pretendido do projeto focando nos conceitos novos que aprendemos ao longo da unidade curricular.

Mesmo assim acreditamos que com o desenvolvimento deste trabalho, conseguimos consolidar o nosso conhecimento de C com conhecimentos da unidade curricular sistemas operativos. Agora estamos mais familiarizados com estes conhecimentos e acreditamos que estes serão muito úteis no desenvolvimento de projetos futuros nesta área.