Projeto Prático

Sistemas Operativos

João Faria A100062
Mário Rodrigues A100109
Pedro Santos A100110

Projeto desenvolvido para a Licenciatura em Ciências da Computação

Departamento de Informática
Universidade do Minho
maio 2023

1 Introdução

O presente relatório refere-se ao trabalho prático proposto na unidade curricular de Sistemas Operativos na Licenciatura em Ciências da Computação da Escola de Ciências da Universidade do Minho no ano de 2023.

O objetivo do trabalho consiste no desenvolvimento de uma aplicação cliente/servidor cujo objetivo consiste em processar comandos de bash e consultar os programas em execução, o servidor é responsável por armazenar as informações e o estado de programas terminados.

A seguir, são expostas as decisões tomadas pelo grupo, bem como o porquê das mesmas e as suas vantagens e eventuais desvantagens. Também é explicada a implementação de forma breve, aprofundamento apenas as partes que o grupo considera as mais relevantes para a compreensão do funcionamento do *software* desenvolvido. Finalmente, é analisado o desempenho da solução em diferentes cenários.

2 Arquitetura

Como mencionado anteriormente, a aplicação segue o modelo cliente/-servidor, pelo que existem dois programas a desenhar: o servidor, responsável por armazenar informações e o estado de programas; e o cliente, é responsável por executar comandos de bash e consultar os programas em execução.

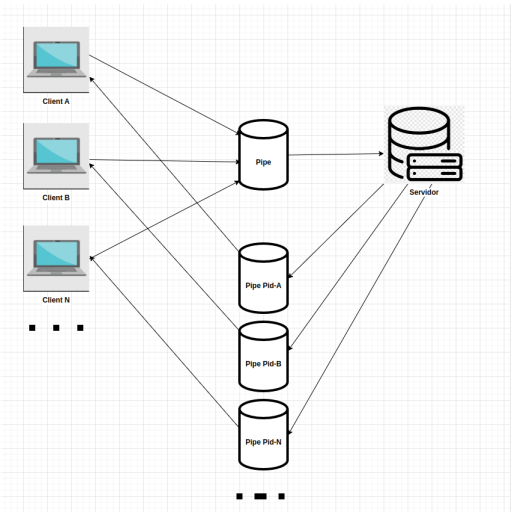

Figura 1: Arquitetura

2.1 Servidor

O servidor é constituído essencialmente por três componentes, cada uma correspondendo a um processo filho do processo principal do servidor.

De seguida explica-se em detalhe, de forma agnóstica relativamente à sua implementação, como cada um destes componentes funciona.

2.1.1 Gestor de Pedidos

Apesar de não ter sido implementado, nós acreditamos que com o uso de threads ou processos o servidor seria capaz de lidar com múltiplos processos ao mesmo tempo. No caso de threads, o servidor criaria uma nova thread para lidar com cada novo pedido de cliente recebido. No caso de usar processos, o servidor criaria um novo processo filho para lidar com cada novo pedido de cliente. Para garantir que os dados estejam protegidos, poderia ser usado técnicas de sincronização, como semáforos ou mutexes, para garantir que cada thread ou processo tenha acesso exclusivo aos dados relevantes em um determinado momento.

2.2 Cliente

O cliente é um processo independente do servidor - aliás é possível vários clientes estarem ligados ao servidor simultaneamente. O cliente deve receber o seu pedido através dos seus argumentos, processá-los para o formato utilizado pelo servidor, criar um canal de comunicação (*FIFO*) para o qual o servidor lê desse mesmo canal até o pedido ser terminado, imprimindo a resposta do cliente e do servidor para o *standard output*.

3 Implementação

Com a arquitetura da aplicação detalhada, são apresentados de seguida os detalhes de implementação considerados relevantes para a compreensão do funcionamento do projeto.

A filosofia da implementação foi tornar o programa o mais genérico possível, isto é, tentar que este não esteja limitado apenas às restrições do enunciado (nomeadamente no número e nome dos programas que é possível executar), ou a outras limitações, tais como o tamanho dos caminhos e nomes de ficheiros. Além disso, procurou desenvolver-se uma aplicação com um grau de escalonamento relativamente alto, ou seja, que consiga suportar um elevado número de pedidos simultaneamente.

Isto levou a desafios adicionais de desenvolvimento, nomeadamente a garantia de que o programa não tem fugas de memória, o que foi verificado através da ferramenta Valgrind. Atualmente, o programa não apresenta qualquer fuga de memória.

3.1 Pedidos

Para uniformizar a implementação de pedidos entre o servidor e o cliente, e de forma a facilitar o processamento do pedido no lado do servidor, foi implementada a estrutura Program que armazena a informação de um pedido. A definição da estrutura é a seguinte:

```
typedef struct {
    int running; // 0 se o programa estiver a correr, 1 se já estiver terminado
    int status; // 0 para execução de programas, 1 para status
    pid_t pid; // PID
    char* program; // Programa a ser executado
    long sec; // sec
    long ms; // ms sem sec
} program;
```

O ms é preenchido pelo servidor à chegada do pedido, correspondendo à ordem de chegada ao servidor.

3.2 Comunicação entre processos

De forma a otimizar as operações de leitura e escrita em pipes, foram criados pipes que permitem a utilização de buffers de forma transparente às suas chamadas de funções. Estes tipos suportam a leitura e escrita, respetivamente, de um número fixo de bytes (inteiros, por exemplo) ou de strings terminadas por um caracter nulo.

É importante ressaltar que a comunicação entre processos pode ser assíncrona, o que significa que um processo pode enviar uma mensagem para outro processo sem precisar esperar uma resposta imediata. Além disso, a comunicação entre processos pode ser bloqueante ou não-bloqueante, o que significa que um processo pode esperar por uma resposta do outro processo.

3.3 Configuração do Servidor

Seguindo esta lógica, podemos criar um índice de transformações. Mais concretamente, o índice de uma transformação é a posição nos arrays de

nome e número máximo de instâncias dessa mesma transformação. Assim sendo, é possível referir-se a operações usando apenas um inteiro, o que diminui a quantidade de dados a transmitir.

3.4 Pedidos de *Status*

O pedido de status permite ao utilizador do cliente solicitar informações sobre os programas que estão em execução no momento. Para realizar esta interrogação, o cliente envia um pedido ao servidor através de um fifo. O servidor recebe este pedido e começa a processá-lo.

Para obter as informações necessárias sobre os programas em execução, o servidor verifica o estado de cada processo em execução no sistema operativo. Para isso, o servidor utiliza funções do sistema operativo, como a função gettimeofday(), que permite obter informações de tempo precisas sobre os processos em execução.

Com base nestas informações, o servidor monta uma lista com os dados de cada processo em execução, incluindo o seu PID (identificador único), nome e tempo de execução até ao momento (em milisegundos).

Após ter reunido as informações necessárias, o servidor envia a resposta ao cliente através da conexão de pipe com nome. O cliente recebe esta resposta e apresenta-a ao utilizador através da linha de comandos, mostrando a lista de programas em execução no momento, um por linha, com as informações de PID, nome e tempo de execução.

3.5 Cliente

A implementação do cliente é relativamente simples: o programa lê os seus argumentos, converte-os para um Program e envia-o para o servidor. Quando esse *FIFO* fechar, significa que o pedido terminou de executar e o cliente termina normalmente.

4 Conclusão

Deste modo, e para concluir, o grupo considera ter cumprido todos os requisitos impostos, implementando uma aplicação cliente/servidor através de comunicação entre processos não só fiável como também com um bom desempenho, permitindo aplicar transformações a ficheiros de forma concorrente.

Como em qualquer projeto, existem aspetos que podiam ter sido melhorados. Um desses aspetos seria a implementação, utilização de threads pode

melhorar o desempenho do servidor ao permitir que várias conexões sejam processadas em paralelo. Em vez de esperar por uma conexão ser processada antes de aceitar outra.

Assim sendo, o grupo considera o seu projeto como tendo sido bem sucedido.