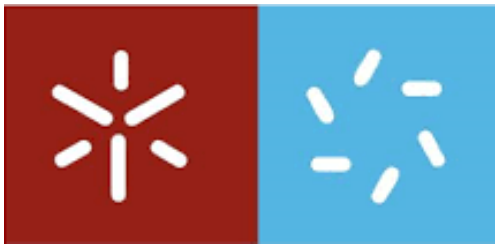

Universidade do Minho

Licenciatura em Ciências da Computação

Trabalho Prático - Sistemas Operativos

Grupo 6

Diogo Aires (a91685) João Silva (a91638) Eduardo Pereira (a70619)

13 de maio de 2023

Resumo

O propósito deste relatório é dar um resumo do funcionamento do trabalho prático proposto na Unidade Curricular de Sistemas Operativos.

Este trabalho prático consiste na implementação de um serviço de monitorização dos programas executados numa máquina. Para tal, o utilizador deve conseguir executar determinados programas, através de pedidos.

Os pedidos feitos ao sistema são realizados pelo utilizador, a partir da interface Cliente (*Tracer*), via linha de comandos. As respostas a estes pedidos devem também ser recebidas pela interface Cliente, que é responsável por apresentar essa informação ao utilizador.

O Servidor (*Monitor*) é responsável pelo processamento do sistema, tal como armazenamento de informação e resposta aos pedidos realizados a partir do *Tracer*.

A partir de pedidos feitos do Cliente para o Servidor, será também possível consultar alguns dados estatísticos sobre os programas em execução num determinado instante e tempo total de execução de programas.

Índice

Capítulo 1

Introdução

Este relatório descreve o desenvolvimento de um serviço de monitorização de programas executados numa máquina, conforme especificado no enunciado. O objetivo do serviço é permitir que os utilizadores executem programas através de um Cliente e obtenham informações sobre a sua resposta e o tempo de execução desses programas. Além disso, um administrador de sistemas pode consultar os programas atualmente em execução e obter estatísticas sobre programas já terminados.

O serviço foi implementado na linguagem C, utilizando comunicação por pipes com nome *FIFO's* entre o Cliente e o Servidor. O cliente, chamado "Tracer", fornece uma interface de linha de comando para os utilizadores interagirem com o serviço. O servidor, chamado "Monitor", é responsável por armazenar as informações relevantes sobre a execução dos programas e responder aos pedidos dos clientes.

Ao longo deste relatório, detalhamos as decisões de implementação tomadas, os desafios encontrados e as soluções adotadas. Também apresentamos uma análise crítica do serviço desenvolvido, destacando os seus pontos fortes e possíveis melhorias.

Capítulo 2

Funcionalidades

2.1 Pedidos

O cliente é responsável por executar programas dos utilizadores através da opção **Execute** e comunicar ao servidor o estado dessa execução. O servidor é responsável por armazenar essa informação de execução e disponibilizá-la para consulta.

Assim, para executar um programa, o utilizador passa ao cliente os seguintes argumentos de linha de comando:

1. A opção execute -u;
2. O nome do programa a executar;
3. Os argumentos do programa, caso existam.

2.2 Respostas

Como resposta ao pedido do utilizador, este recebe, via *output* - *prompt*, o PID respetivo ao programa executado, o output do programa executado e o seu tempo total de execução.

2.3 Dados Estatisticos

É possível fazer consulta da informação de cada programa (como PID, o nome do programa em execução e o tempo de execução até ao momento), a pedido do administrador, com o comando *status*.

Capítulo 3

Estrutura de Implementação

3.1 Tracer

3.1.1 Execução de Programas

A função **runCommand** é utilizada para executar um programa individualmente. Esta cria um processo-filho através da System Call fork() que, de seguida, utiliza execvp() para executar o programa fornecido como argumento dentro do processo filho, de forma a não impactar o resto da execução de código. Antes de executar o programa, o cliente envia informações relevantes para o servidor, através de uma única string da forma "UI%05d%d %s %llu", em que os parâmetros são:

- U: Informa que se trata de um execute -u
- I: Informa que é o início da execução
- %05d: Nº de bytes da informação que vem a seguir na string
- %d: PID
- %s: Programa
- %llu: Tempo de início de um novo programa

O cliente indica também ao utilizador o PID do processo que vai executar o programa pedido. Quando a execução termina, o processo-filho é terminado e o processo-pai espera que este termine. O processo-pai, após verificar que o seu processo-filho terminou, informa o Servidor do fim da execução e transmite-lhe o tempo de finalização, através de uma string da forma "UF%05d%d %llu", em que os seus parâmetros são:

- U: Informa que se trata de um execute -u
- F: Informa que a execução terminou
- %05d: Nº de bytes da informação que vem a seguir na string
- %d: PID
- %llu: Tempo de finalização do programa relativo a este PID

Após a conclusão do programa, o utilizador recebe o tempo de execução (exibido no stdout após o output do programa executado).

3.1.2 Consulta de *status*

A função **runStatus** é responsável por consultar o estado de execução de programas. Esta recebe o PID do programa que faz o pedido como argumento, de forma a criar um FIFO, sendo que o seu nome é o PID recebido. Assim, existe uma linha de comunicação com o Servidor exclusiva a cada cliente que consulta o estado de execução.

A função envia uma requisição para o serviço de monitorização (monitor) por um FIFO através de uma string na forma "S%s"em que os seus parâmetros são:

- S: Informa que se trata de um status
- %s: PID do processo que fez o pedido

O Cliente aguarda, então, a resposta do Servidor. Quando a recebe, lê a resposta a partir do FIFO e disponibiliza essa informação para o Utilizador, na interface (via prompt).

3.2 Monitor

3.2.1 Manipulação de Sinais

A função **signal_handler_C** é responsável por lidar com o sinal SIGINT (gerado pelo Ctrl+C) para encerrar o servidor. O servidor desvincula os FIFOs e imprime uma mensagem a informar que vai terminar antes de sair.

3.2.2 Armazenamento de Informações

Os dados sobre os pedidos de todos os clientes são armazenados em memória e em ficheiro. Para guardar em memória foi criada uma lista ligada:

```
typedef struct process
{
    char *pid;
    char *program;
    char *time_s;
    char *time_f;
    struct process *next;
} *Process;
```

Esta lista ligada foi designada de forma a guardar as informações de execução de programas: PID, nome do programa e tempo inicial. O tempo final do programa apenas é guardado aquando da terminação da execução, sendo que até esse momento, a string time_f é guardada apenas com ***NULL***.

As funções **addProcessoExecuted**, **runExecuteI** e **runExecuteF** são usadas para armazenar informações sobre os processos.

- **addProcessoExecuted** adiciona um novo processo à lista.
- **runExecuteI** é chamada quando um programa é iniciado (execute -U I) e adiciona o processo à lista.
- **runExecuteF** é chamada quando um programa é encerrado (execute -U F) e atualiza o tempo de término do processo correspondente.

3.2.3 Status

A função **runStatus** é usada para enviar informações de status para o cliente (tracer). Abre o FIFO fornecido pelo cliente em modo de escrita. Percorre a lista de processos em execução e, para cada processo não finalizado, calcula o tempo de execução (até ao dado instante) e envia os detalhes para o cliente através do FIFO específico desse cliente.

3.3 Funcionamento

- O cliente (tracer) envia comandos para o servidor (monitor) por um FIFO chamado *args_fifo*. Os comandos podem ser do tipo "execute -U I"para iniciar um programa, "execute -U F"para encerrar um programa ou "status"para solicitar informações de status.
- Quando o cliente (tracer) envia um comando de início de programa ("execute -U I"), ele fornece o PID, o nome do programa e o tempo de início. O servidor (monitor) recebe esse comando, cria uma nova estrutura de processo e adiciona-a à cabeça da lista de processos.
- Quando o cliente (tracer) envia um comando de término de programa ("execute -U F"), este fornece o PID e o tempo de término. O servidor (monitor) recebe esse comando, procura o processo correspondente na lista de processos em execução e atualiza o tempo de término.
- Quando o cliente (tracer) envia um pedido de estado ("status"), este fornece o seu PID, de forma ao servidor saber para qual cliente enviar informação. O servidor (monitor) recebe esse comando, percorre a lista de processos em execução e calcula o tempo de execução de cada processo não finalizado. De seguida, envia as informações de status para o cliente através de um FIFO.
- O servidor (monitor) regista um manipulador de sinal para capturar o sinal SIGINT (gerado pelo Ctrl+C) e permitir que o programa seja encerrado de forma adequada. Quando o sinal SIGINT é recebido, o servidor (monitor) desvincula os FIFOs e finaliza a execução.

Em resumo, o cliente (tracer) envia comandos de controlo para o servidor (monitor), que mantém uma lista de processos em execução e fornece informações de status quando solicitado. O programa permite a consulta e monitorização de programas em execução.

Capítulo 4

Conclusão

Neste trabalho prático, desenvolvemos um serviço de monitorização de programas executados numa máquina. O serviço permite ao utilizador executar programas, consultar programas em execução e obter estatísticas sobre programas terminados.

Implementamos com sucesso as funcionalidades básicas, como a execução de programas e a comunicação entre o cliente e o servidor através de FIFO's.

Durante o projeto, adquirimos conhecimentos sobre C, comunicação entre processos e manipulação de arquivos. Também enfrentamos desafios relacionados ao processamento concorrente e à gestão eficiente dos recursos do sistema.

Embora o serviço esteja funcional e atenda às necessidades básicas dos usuários, há oportunidades de melhoria. Dois dos obstáculos que encontramos durante a realização deste projeto foi a nossa falha em conseguir implementar uma Pipeline funcional e armazenar informação em ficheiro sobre processos terminados.

Em resumo, o desenvolvimento deste serviço de monitorização de programas foi um desafio interessante e permitiu-nos complementar e aplicar os conceitos aprendidos durante as aulas de Sistemas Operativos, nomeadamente system calls e comunicação entre diferentes processos.

Foi também um bom exercício para dar um pouco mais de transparência àquilo que acontece em *background* quando se executa um determinado programa na linha de comandos.