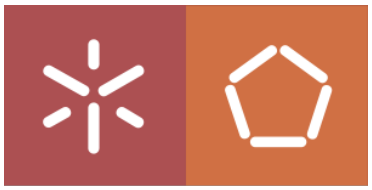

Universidade do Minho
Escola de Engenharia

Universidade do Minho
Mestrado Integrado em Engenharia Informática

Sistemas Operativos
Grupo 26

Controlo e Monotorização de Processos e Comunicação

14 Junho 2020


Ana Margarida Campos
(A85166)


Ana Catarina Gil
(A85266)


Tânia Rocha
(A85176)

1 Resumo

O presente relatório descreve o trabalho prático realizado no âmbito da unidade curricular de Sistemas Operativos ao longo do segundo semestre, do segundo ano, do Mestrado Integrado em Engenharia Informática da Universidade do Minho.

O objetivo do projeto é implementar um serviço de monitorização de execução e de comunicação entre processos. O serviço deverá permitir a um utilizador a submissão de sucessivas tarefas bem como a sua execução.

Neste documento descrevemos sucintamente o sistema desenvolvido, as principais funções utilizadas, tal como as decisões tomadas durante o projeto.

Conteúdo

2 Introdução

2.1 Contextualização

O serviço desenvolvido permite a um utilizador a submissão de sucessivas tarefas, cada uma delas sendo uma sequência de comandos encadeados por pipes anônimos. Assim sendo,o serviço é capaz de identificar as tarefas em execução, bem como a conclusão da mesma. A interface com o utilizador contempla duas possibilidades: uma através de linha de comando, indicando opções apropriadas e a outra uma *shell*.

2.2 Conceção da Solução

Para o desenvolvimento deste projeto foram elaborados dois programas referentes ao Cliente e ao Servidor. A comunicação entre ambos é feita através da criação de pipes com nomes, nomeadamente um pipe no servidor responsável por ler os pedidos do Cliente e um pipe no Cliente responsável por receber as respostas do Servidor, sendo este último diferente para cada cliente.

A interpretação dos comandos é feita no lado do Servidor para o qual é criado um *fork* para cada um dos mesmos.

De maneira a guardar informação relativa a cada um dos comandos, é criada uma estrutura *Comandos* com o objetivo de guardar dados relativos a cada um. No ficheiro *historico.txt* vão ser armazenadas todas as structs referentes a cada comando. Um comando pode estar no estado *Running* se ainda estiver em execução, *concluida* se terminou a sua execução normalmente, *max execução* no caso de ter ultrapassado um tempo de execução limite (que pode ser definido através de um comando) e *terminado* se o processo tiver sido terminado à força por um cliente.

3 *Struct*

A estrutura de dados **Comandos** armazena os dados referentes a um determinado comando, entre os quais: comando que representa a linha a ser executada, o status que é o estado atual do processo responsável pela execução da tarefa, o id, o pid do processo e os inteiros início e fim utilizados para a funcionalidade adicional.

```
typedef struct Comandos
{
    char comando[200];
    char status[200];
    int id;
    int pid;
    int inicio; //extra
    int fim;   //extra
} Comandos;
```

Figura 1: Estrutura de Dados

4 Cliente

A comunicação entre os diversos Clientes e o Argus(Servidor) é efetuada através de um **fork** em que o processo filho lê do terminal e envia a respetiva tarefa para o Servidor interpretar e executar. No caso do processo pai, este tem a função de receber a resposta à tarefa enviada e escrevê-la no seu terminal.

De modo a conseguir a multiplexagem de Clientes no mesmo Servidor, para cada tarefa enviada é identificado o *pid* referente ao processo de um determinado Cliente permitindo ao Servidor responder ao Cliente correto.

De forma a permitir a execução do programa tanto em *shell* como em linha de comando, é criada uma condição que verifica o número de argumentos especificados, ou seja, quando este é igual a 1 é porque estamos a executar numa *shell*, caso contrário a tarefa foi enviada da linha de comando.

5 Argus

O nosso Servidor nomeado de **Argus**, é responsável por receber e interpretar todas as tarefas que lhe são enviadas. De cada vez que este recebe uma tarefa este cria um **fork**, sendo esta a funcionalidade que permite a concorrência entre vários clientes.

- **Tempo de execução**: De forma a limitar o tempo de execução de uma determinada tarefa, foi criada uma variável global que pode ser alterada através do comando "tempo-execucao t"onde t representa esse mesmo tempo.
- **Tempo de inatividade**: De forma a limitar o tempo que um processo responsável por um determinado comando está parado, foi também criada uma variável global que pode ser alterada através do comando "tempo-inatividade t"onde t representa esse mesmo tempo.
- **Executar**: Com o objetivo de executar uma tarefa, o comando recebido é divido e executado através de pipes anónimos. Imediatamente antes do programa executar o comando, é guardado no ficheiro *historico.txt* esse mesmo comando, especificando o seu estado (*Running*). Após ele terminar o seu estado é atualizado de acordo com a causa da sua morte.
- **Listar**: Recorrendo ao ficheiro *historico.txt* são filtrados todos as estruturas comandos cujo estado é *Running*.
- **Histórico**: Recorrendo ao ficheiro *historico.txt* são filtrados todos as estruturas comandos cujo estado não é *Running*.
- **Ajuda**: É apenas construído um *char** com todas as opções de tarefas disponíveis no programa.

Sempre que o Servidor é terminado todos os ficheiros e pipes utilizados na execução do programa são eliminados.

6 Funcionalidade Adicional

Com o objetivo de realizar a funcionalidade adicional, nomeadamente a consulta do *standard output* de um comando já executado, está presente na nossa struct os inteiros início e fim. Estes representam os limites do outputo no ficheiro *log.txt*. O **inicio** é obtido através do valor do descritor após ser aberto e o **fim** é obtido através do descritor após ser escrito todo o output. Após adquiridos ambos os valores é chamada uma função que atualiza na respetiva struct estes valores. Aquando a requesição do output de um determinado comando especificado pelo seu id, este é procurado no ficheiro *historico.txt* e retirando os valores de início e fim é recolhida no ficheiro *log.txt* o conteúdo posicionado entre estes dois limites e enviado para o Cliente.

7 Conclusão

Face ao problema apresentado e analisando criticamente a solução proposta concluímos que cumprimos a maior parte das tarefas, excetuando apenas o tempo de inatividade. No entanto a funcionalidade adicional foi desenvolvida com sucesso. Deste modo, foi então construído um serviço de monotorização de execução e de comunicação entre processos funcional.