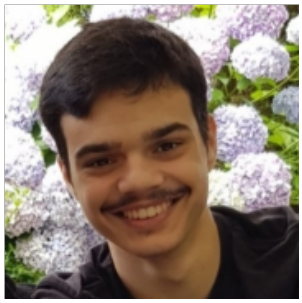
Universidade do Minho
Escola de Engenharia

Sistemas Operativos

Relatório Trabalho Prático

Daniel José Silva Furtado A97327
João Alvim A95191
Ricardo Silva Machado Araújo A96394
Grupo 79



Índice

1 Introdução

Pretende-se implementar um serviço que permita monitorizar programas que são executados por um ou mais clientes. Cada cliente pode executar programas e pedir para informação sobre os programas, que estão a ser executados e que já foram executados.

O cliente tem como função comunicar com o servidor e fazer-lhe pedidos. Existem os seguintes tipos de pedidos possíveis para serem realizados:

- execute - u
- execute - p
- status
- stats-time
- stats-command
- stats-uniq

2 Comunicação Tracer - Monitor

A primeira etapa entre a comunicação do *tracer* e *monitor* é a criação de um FIFO para que os dois possam mandar informação sem interferência de outros clientes. Por isso existe um FIFO principal chamado *FIFO* que está sempre aberto para receber pedidos de conexão. Para fazer um pedido de conexão, cada cliente envia para esse FIFO o *PID* do seu processo que é usado como nome do novo FIFO por onde irá comunicar com o servidor. Assim cada cliente tem um canal que só o o servidor conhece.

Aberto o canal entre o *tracer* e *monitor*, o *tracer* envia um código que indica qual o tipo de pedido que quer realizar, no nosso caso temos os seguintes códigos:

- 11. quando se faz: execute -u (é iniciada a execução)
- 12. quando se faz: execute -u/-p (é terminada a execução)
- 13. parar o servidor graciosamente
- 14. para status
- 15. para stats-time
- 16. para stats-command
- 17. para stats-uniq

Assim se por exemplo um cliente quiser executar uma pipeline de comandos:

1. Envia o seu pid para o servidor e passa a comunicar com o servidor principal através de um novo fifo com o nome a ser o pid.
2. Envia o código 11, que corresponde a ação a executar.
3. Envia a informação do programa para o servidor, neste caso envia a *struct* **Process**.
4. São executados os comandos ao longo da pipeline, em processos filhos.
5. O processo pai aguarda pela conclusão dos processos filhos.
6. Processo pai envia o código 12 para dizer que terminou o comando.
7. Envia a informação final do programa para o servidor.

Para enviar a informação sobre um programa para o servidor utilizamos uma estrutura, *Process*,que contém os dados de:

- *PID* do processo
- tempo de execução em milissegundos
- comando

2.1 Estrutura principal do servidor

O processo principal do servidor vai estar permanentemente num ciclo *while* que recebe pedidos de conexão. Assim que receba os pedidos e a informação de um processo o servidor, para evitar espera ativa, fecha a conexão com esse cliente. Que depois, só volta a ser estabelecida no fim do programa do cliente terminar. Para a gestão dos **Processos**, quando o cliente envia a *struct*, não são criados processos filhos no lado do servidor. Adicionar ou remover à *Hashtable* a *struct* é algo insignificante e não afeta significativamente a concorrência do nosso programa. Também caso alterássemos a *Hashtable* em processos filhos não era mantida a concorrência de dados, visto que, na verdade, a *Hashtable* não iria ser alterada. Já na execução dos **Status** é necessário a criação de processos filhos para manter a concorrência entre processos.

2.2 Execução de pedidos

2.2.1 Execute

Para a execução de um comando passado como argumento é necessário fazer a separação do comando pelos parâmetros e guardar num array. Após realizar todo o processo de conexão com o servidor, é criado um pipe e um processo filho, o pipe para manter a conexão entre o processo filho e o pai, e o processo filho para executar o comando. O processo filho vai esperar pelo pai mandar a informação toda ao servidor, ficando bloqueado até que o pai lhe envie um sinal que já enviou a informação. Quando é desbloqueado, o processo filho executa o comando. O processo pai verifica se o filho terminou normalmente, e após o filho terminar o pai envia para o server a informação que o programa já foi executado.

2.2.2 Pipeline

Fizemos uma pipeline muito semelhante ao exercício 6 do guião 6. Antes de tudo, tal como acontece quando é executado apenas um comando, é enviado o código 11, para informar que a execução dos comandos inciou. Depois percorremos a string do comando para sabermos quantos comandos a executar são e depois guardamos-los num array de strings. Para os executar encadeamos a execução destes com pipes, à medida que percorremos a pipeline vamos fechando as extremidades de leitura e escrita que não são utilizadas. Após o término do último comando da pipeline é informado o servidor que terminou a execução, tal como acontece quando é apenas um único comando.

3 Consultas

3.1 Status

Sobre o comando Status que é passado como argumento para o cliente, é criado um FIFO com o número do *PID* e é enviado para o servidor o *PID* para criar a ligação com o cliente. Depois é enviado o código único, 14, para informar o servidor. Também é enviado o novo número, neste caso será o número do *PID* mais "0057"para criar novo FIFO e criar uma ligação com o servidor, para que o servidor envie a resposta. No lado do servidor, cada vez que recebe o código único para executar o status é sempre criado um processo filho para executar o status e manter assim a concorrência e o paralelismo. Dentro do processo filho, lê do FIFO o número que irá ser atribuído para o FIFO de escrita no servidor. Depois, é consultada a *HashTable* onde estão guardados todos os processos em execução e por fim são enviados todos os processos em execução ao cliente.

3.2 Stats-Time

A estratégia para conseguir obter o tempo total utilizado por um dado conjunto de programas identificados por uma lista de *PID*s passada como argumento foi similar à forma de comunicação do *Status*.

Após o cliente enviar o código pré-definido para o servidor e também o *PID* para comunicar com o servidor por um FIFO independente são enviados todos os *PID*s para o *monitor* dados como argumento e no fim o *tracer* envia um código de terminação, neste caso -1, para informar o servidor que os *PID*s foram enviados.

À medida que o *monitor* recebe os *PID*s abre o ficheiro correspondente, percorre o ficheiro, lê todas as *structs* **Process** e armazena o valor do tempo de execução de cada processo numa variável.

No final, o valor é somado a uma variável global e depois esse valor é enviado pelo FIFO de volta para o cliente.

3.3 Stats-Command

Tal como no Stats-Time, após todo o processo de criar FIFOs independentes para comunicar com o servidor e mandar o código único, o cliente envia

todos os *PID*s passados como argumento após, neste caso, enviar o nome do programa também passado como argumento, e para o FIFO não bloquear uma decisão tomada foi enviar o tamanho da *string* do programa primeiro.

O *monitor* lê o comprimento da string do comando e armazena o comando numa variável. Ao longo que recebe os *PID*s, verifica se o ficheiro correspondente possui o comando que foi armazenado ao percorrer o ficheiro e comparar o comando passado como parâmetro com o que é extraido da *struct* Process. Caso seja igual é incrementada uma variável e no fim retorna o valor de vezes que foi executado o comando nesse ficheiro.

Caso a *struct* Process possua um comando proveniente de uma pipeline, é feito o *parse* do comando e é verificado para cada comando da pipeline.

Após obter o valor total do número de vezes que o comando foi executado para um dado conjunto de *PID*s é devolvido ao cliente esse valor.

3.4 Stats-Uniq

O Stats-Uniq também segue a mesma regra, envia o código correspondente e também todos os *PID*s.

Já no servidor, à medida que lê os *PID*s, itera o ficheiro e armazena todos os comandos dos processos que estavam no ficheiro num array. Depois itera o array e verifica se cada valor já existe num array principal. Caso não exista, é adicionado o comando ao array principal e depois é enviado pelo FIFO o comando. Uma pequena diferença, é quando existem comandos de pipeline, que é necessário dividir os comandos primeiro.

4 Conclusão

Em suma, os resultados obtidos ao longo do trabalho foram bastante satisfatórios uma vez que conseguimos construir todos os desafios propostos no enunciado.

Com o desenvolvimento deste trabalho podemos ver o quão importante é fazer um bom manuseamento das *system call* e a importância de conhecer como funcionam ficheiros e o sistema operativo. Estes são fatores bastante importantes e a ter em consideração nos próximos trabalhos ao longo do nosso percurso académico e profissional.