Sistemas Operativos

Trabalho Prático

Grupo - 68

Ivo Miguel Alves Ribeiro A96726

Diogo Luís Almeida Costa A95460

A96726

A95460

Índice

Introdução

O intuito deste trabalho é implementar um serviço de monitorização dos programas executados numa máquina. Os utilizadores devem conseguir executar programas, através do cliente, e obter o seu tempo de execução.

Para além disso, o cliente deve conseguir consultar, através do servidor, todos os programas que se encontram atualmente em execução, incluindo o tempo de execução até ao instante atual pelos mesmos.

Finalmente, o servidor deve também permitir a consulta de estatísticas sobre programas já terminados.

Metodologia

De modo a facilitar a comunicação entre o servidor e os clientes via *"pipes"*, decidimos definir dois *"pipes"* globais dos quais um servirá para o envio de mensagens por parte dos clientes e a receção das mesmas por parte do servidor, enquanto o outro servirá para pedidos de consulta de programas terminados por parte dos clientes ao servidor.

Tanto o cliente como o servidor sabem como diferenciar as ações possíveis em cada um desses *"pipes"*.

De maneira geral o nosso cliente consoante aquilo que lhe é passado manda uma mensagem ao servidor por um dos "*pipes*" globais, com a indicação do início de um novo processo, após o envio da mesma o cliente inicia então a execução do programa passado se for esse o caso ou então aguarda a resposta do servidor caso uma consulta tenha sido pedida pelo mesmo. No caso de ter iniciado uma execução, apos esta ação ter terminado por completo, é então enviada uma nova mensagem ao servidor indicando esse termino, garantindo assim que o servidor sabe sempre o estado de todos os processos.

Quanto ao funcionamento do servidor este divide-se em dois processos que estão ativos em 100% do tempo onde cada um é responsável pela leitura de um dos *"pipes"* globais, sendo que é garantido que cada um destes processos apenas leia um pedido de um cliente por vez para que não haja erros.

Esta estratégia foi pensada de maneira a tornar o servidor mais rápido não afetando a execução daquilo que é pretendido.

Modelo de comunicação

Todas as comunicações feitas por parte do cliente para o servidor são identificadas, isto é, cada mensagem tem como primeiro elemento da mesma um número que indica ao servidor o tipo de mensagem a que esta se refere.

(Nota: A estratégia de identificar todas as mensagens tem um prepósito que acabou por não ser necessário, uma vez que ao interpretar o enunciado, pensávamos que o servidor é que seria responsável por executar os programas, porem após sermos alertados pelo professor reestruturamos o nosso código inicial e movemos a execução dos programas para o lado do cliente, contudo mantivemos as tags de identificação das mensagens.)

Quanto às mensagens do servidor para os clientes estas não precisam de ser identificadas pois são escritas em *"pipes"* abertos pelo cliente em questão, não proporcionando erros nesse aspeto.

Modelo para pedidos de execução:

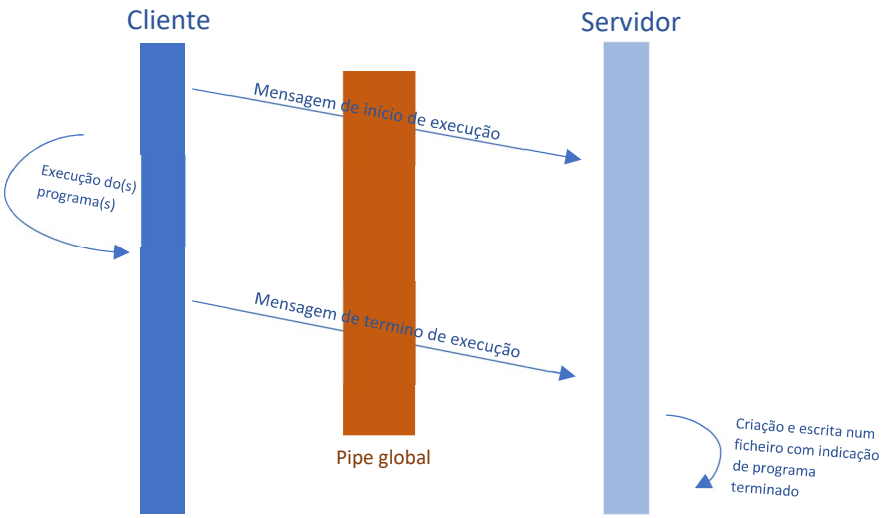

Figura 1 – Representação de comunicação entre servidor e clientes

A mensagem de início de execução é definida por:

- Uma "tag" de identificação do tipo de mensagem, neste caso em específico o número "1" caracteriza um pedido de execução de um só programa, já o número "2" caracteriza um pedido de execução de múltiplos programas;

- Um inteiro que se refere ao tamanho do conteúdo da mensagem em questão, uma vez que este campo tem tamanho varável;

- Conteúdo da mensagem contendo:

  - Pid do cliente;

  - Duas variáveis que se referem ao instante temporal do início da execução do programa em questão;

  - Informação relativa ao(s) programa(s) a executar, assim como o nome do mesmo e os seus argumentos;

Após o envio desta mensagem, o cliente começa então a execução do(s) programa(s) em questão.

De maneira idêntica á mensagem de indicação de início de execução, quando todos os programas terminarem a sua execução é enviada uma nova mensagem ao servidor, definida por:

- Uma "tag", neste caso com o número 9;

- O tamanho do conteúdo dessa mensagem;

- O conteúdo dessa mensagem que é composta por:

  - Pid do cliente;

  - Duas variáveis referentes ao instante em que terminou a execução.

O servidor assim que recebe esta informação faz o cálculo do tempo de execução, com a informação presente em memoria, e cria um ficheiro no destino desejado com o pid do cliente, o nome e os argumentos do programa executado e ainda o tempo de execução, libertando essa informação da sua memoria;

Modelo para consulta de programas em execução:

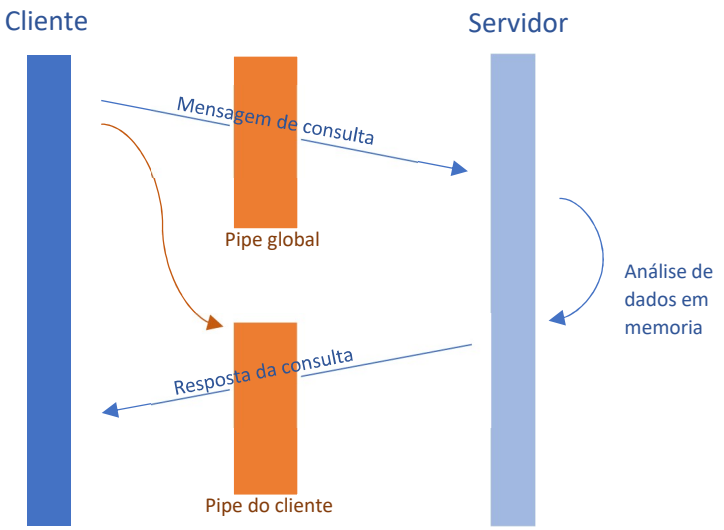

Figura 2 – Representação de uma consulta de programas em execução entre servidor e clientes

A mensagem de consulta é composta por:

- Uma “tag” que para este caso corresponde ao número 3;

- O pid do cliente;

O servidor identificando esta “tag”, abre um pipe para escrita com base no pid do cliente, uma vez que este já o criou anteriormente e tem o mesmo aberto para leitura.

Depois percorre a hashtable responsável por guardar a informação sobre os processos em ainda em execução, e a cada processo encontrado é escrita uma mensagem no pipe criado pelo cliente composta por:

- Pid do cliente que tem determinado processo em execução;

- O programa em execução e os seus argumentos

- O tempo em milissegundos desde o início da sua execução até ao momento atual;

Aquando do encerramento do pipe para a parte de escrita pelo servidor o cliente deixa de ler desse mesmo pipe e termina esse processo.

Modelo para consulta de programas terminados:

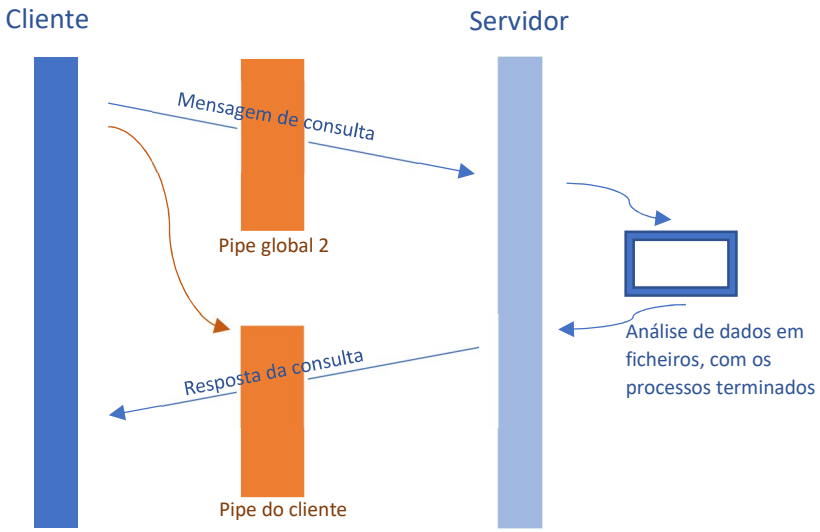

Figura 3 – Representação de uma consulta de programas terminados entre servidor e clientes

A mensagem de consulta de programas finalizados é composta por:

-Uma “tag”, neste caso o valor “4” corresponde a mensagens do tipo stats-time, já o valor “5” corresponde ao comando stas-command e por fim o valor “6” corresponde ao comando stats-uniq;

- O tamanho do conteúdo da mensagem;

- O conteúdo da mensagem:

    - Pid do cliente que pediu a consulta;

    - Pids dos processos que queremos consultar, concatenados numa string separados por vírgula, no caso do comando stats-command temos ainda a programa como primeira palavra desta string;

Posteriormente o servidor verifica a existência de cada um dos processos através do seu pid, e caso esse processo exista aplicamos a função necessária para fornecer ao cliente o resulta à procura que pretende.

A resposta é escrita num pipe já criado e aberto pelo cliente com base no seu pid, e uma vez que o pid do cliente foi passado ao servidor este consegue aceder a esse pipe escrever então a resposta.

Execução de programas

Depois de recebida a mensagem proveniente do cliente, o servidor cria um processo filho responsável por dividi-la através da função sscanf obtendo o pid do clinete, programa a executar é o instante inicial deste processo. Após obter esta informação é a fornecida a informação deste novo processo a uma hashtable previamente criada para esse efeito no arranque do servidor, essa informação será guardada na hashtable até o servidor receber uma mensagem por parte do cliente a dizer que a execução do programa terminou.

Enquanto isso, do lado do cliente, a execução do programa é iniciada logo apos enviar a mensagem ao servidor. Para executar o programa passado como argumento no formato de string concatenado com os seus argumentos, começamos por separar tanto o programa como os seus argumentos num array de strings. Esta estratégia deve-se ao facto de utilizarmos a função “execvp” para executar o programa consoante todos os seus argumentos.

Para a execução propriamente dita, decidimos criar um processo filho responsável por executar a função “execvp” uma vez que após a chamada da mesma o processo que a chamou morre e nos queríamos que o nosso processo não terminasse. Contudo o processo pai certificasse de esperar a morte do próprio filho para poder avançar, garantindo que o quando chamarmos a função que devolve o instante após o programa ter terminado não é chamada antecipadamente.

Como foi referido acima, após a execução é enviada uma nova mensagem para o servidor com a informação do instante em que o programa terminou.

Estruturação da consulta de programas em execução

Para a consulta de programas em execução é enviada uma mensagem de tag “3” para o servidor e o pid do cliente que pediu a consulta.

Após receber este pedido o servidor chama uma função que percorre a hashtable e escreve no pipe aberto pelo cliente identificado pelo seu pid, todos os pids, os programas e o tempo de execução até ao momento, dos processos em execução no servidor.

A hahstable que descrevemos em cima é composta por estruturas do tipo *“pid_stats”,* contendo um identificador para o pid, o(s) programa(s) em execução por parte desse cliente, e duas variáveis que identificam o instante de início de execução, como mostra a figura ao lado.

```
struct pid_stat{
    char* pid;
    char* msg;
    long int start_sec;
    long int start_milisec;
};
```

Figura 4 – Estrutura para guardar processos em execução

Uma vez que a cada nova mensagem de execução recebida no servidor é adicionado um processo a esta hashtable e este só é eliminado quando é recebida também no servidor a mensagem de que o processo terminou, garantimos que percorrendo a hashtable temos a informação de todos os processos ainda em execução.

Para o cálculo do tempo de execução até ao momento, chamamos a função “gettimeofday” que nos retorna o instante presente, e com base nos valores retornados por esta função juntamente com os valores presentes na estrutura em causa, relativos ao instante no início da execução desse processo, calculamos a diferença entre os dois e obtemos o tempo de execução até ao momento.

Apos percorrer toda a hashtable é fechado o pipe do lado do servidor, informando assim o cliente que pode deixar de ler deste pipe encerrando assim a execução deste processo.

Execução encadeada de programas

Na execução encadeada de programas assim como na execução simples é enviada uma mensagem para o servidor com o pid do cliente, os programas que vão ser executados, e o instante inicial. Após esse envio, criamos o mesmo número de pipes anónimos que o número de programas a ser executados, e depois separamos a string recebida como argumento assim que encontramos o caráter “|” e criamos um processo filho responsável por substituir os descritores de base para os pipes anónimos na ordem correta, e após fecharmos todos esses descritores chamamos a função “execute” usada também anteriormente para a execução de processos únicos. Esta função é responsável por separar a string correspondente a um programa e seus argumentos em um array e executar o mesmo através da função “execvp”, assim como anteriormente criamos um processo filho para executar enquanto o processo pai aguarda a morte desse mesmo filho.

Como criamos um processo filho para cada programa apos a chamar a função que o executa temos de nos certificar que acabamos com esse processo.

Por fim fechamos os descritores no processo pai e esperamos a morte de todos os filhos por ele criados, e enviamos a mensagem com a instante final para o servidor.

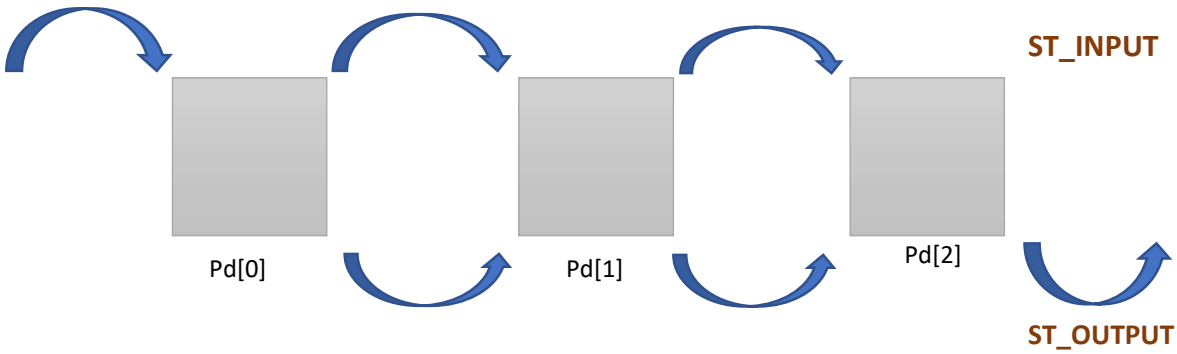

Figura 5 – Exemplo da troca dos descritores para um programa como: "Prog 1 | Prog 2 | Prog 3 | Prog 4"

No primeiro processo mudamos o descritor de escrita Standard para o descritor do pipe[0], depois ate ao ultimo processo mudamos o descritor de leitura do standard para o do pipe[posição anterior] e o de escrita para o pipe[posição em questão] e no ultimo processo substituímos apenas o descritor de leitura do Standard para o pipe[posição anterior], executando os programas apos essa mudança, garantimos que o pipelining se dá de forma correta.

Armazenamento e consulta de tarefas terminadas

Os processos terminados são armazenados em um ficheiro, um por processo, contendo cada um o seu pid o programa que foi executado e seus argumentos e ainda o tempo de execução total desse processo.

Estes ficheiros são todos armazenados num local passado como argumento na inicializarão do servidor.

Para efetuar a consulta dos mesmos, como recebemos os pids dos quais queremos procurar, tentamos abrir os ficheiros que a eles correspondem. Caso não exista esse ficheiro apenas avançamos e fazemos a consulta do próximo processo, caso exista fazemos a consulta conforme o que for pretendido pelo cliente.

O cliente recebe a resposta ao seu pedido através de um pipe criado por ele mesmo, com base no seu pid, onde o servidor como recebeu este pid na mensagem abre o mesmo pipe só que desta vez para escrita enviando para o cliente toda a informação que encontra. Após percorrer todos os pids passados na mensagem é encerrado o pipe por parte do servidor indicando ao cliente que pode deixar de ler do mesmo e encerrar este processo.

Distribuição de tarefas

	Estruturação do modelo de comunicação	*Estruturação da execução de programas*	*Estruturação da execução encadeada de programas*	*Estruturação da consulta de programas em execução*	*Estruturação da consulta de programas terminados*	*Relatório*
Ivo	***50%***	***60%***	***75%***	***75%***	***40%***	***50%***
Diogo	***50%***	***40%***	***25%***	***25%***	***60%***	***50%***

Conclusão

Para concluir achámos este projeto de grande relevância visto que abordou praticamente todos os conceitos lecionados ao longo deste semestre, onde os aplicamos num desafio de maior escala.

Em suma achamos ter cumprido todos os objetivos pedidos para este projeto, pois conseguimos cumprir todas as funcionalidades básicas e ainda praticamente todas as funcionalidades avançadas.