Sistemas Operativos

Ano Letivo 22/23

Escola de Engenharia

Licenciatura em Engenharia Informática

Grupo 48

Inês Castro	**A95458**
Inês Ferreira	**A97040**
José Ferreira	**A97642**

Conteúdo

Introdução

O objetivo deste trabalho é a obtenção de um serviço de comunicação entre programas clientes, utilizadores e um servidor. Os utilizadores devem ser capazes de pedir a execução de programas através de um cliente e obter o seu tempo de execução. Além disso, um administrador de sistema (utilizadores, que invocam o "status") deve ser capaz de visualizar todos os programas em execução e o tempo gasto em cada um deles através de um servidor. O servidor também deve permitir a consulta de estatísticas sobre programas concluídos.

Para isto criar-se-á um programa cliente, "tracer", que oferece uma interface de linha de comando ao utilizador, e um programa servidor, "monitor", com o qual o cliente irá interagir. Ambos os programas devem ser escritos em C e comunicar através de pipes com nome. O stdout deve ser usada pelos clientes para exibir as respostas necessárias aos utilizadores e pelo servidor para fins de depuração.

Comunicação do lado do Cliente

O cliente serve como executor e como intermediário entre o utilizador e o Servidor, isto é, sempre que um utilizador solicita a execução de comandos este deve para além de os executar, informar o servidor do estado do mesmo e quando termina a execução deve também passar essa informação de modo que o servidor registe informações relativamente aos pedidos como exemplo o tempo total de duração. Por outro lado, um utilizador/administrador pode pedir o status dos pedidos ao servidor e para tal envia esse pedido pelo cliente que, nestas circunstâncias é o intermediário entre os dois.

O cliente ao receber um comando (argumentos na main) na forma **./tracer execute -u "command arg0 … argN"** ou **./tracer close** ou **./tracer status** por parte de um utilizador avalia se o segundo argumento é um **status**, um **close** ou então um **execute**.

Caso seja um **status** envia por um pipe nomeado **canal** (canal de comunicação no sentido Cliente -> Servidor) uma mensagem de status e lê de um pipe nomeado com o seu valor de **pid** (canal de comunicação no sentido Servidor -> Cliente/administrador) a resposta com o estado dos programas em execução no momento dado pelo servidor, retornando assim essa informação para o cliente/administrador.

Se o argumento for antes, um close envia pelo pipe **canal** uma mensagem a pedir o encerramento do servidor. Quando recebe a mensagem pelo **canal_status** (canal de comunicação no sentido Servidor -> Cliente) a indicar o encerramento, este fecha o **canal**.

Caso contrário, trata-se de um pedido de **execute** e desse modo, cria um filho através de um fork().

Comunicação do lado do Cliente

No filho a variável pid recebe o pid do filho e, antes de executar, usa a função gettimeofday() para extrair o tempo antes da execução, pelo pipe anónimo criado no início da main denominado **heranca** envia esse tempo para que depois possa ser lido pelo pai. Entretanto, envia ao utilizador uma mensagem ao utilizador (**message_runningCliente**) na forma **Running comando_e_respetivos_args valor_pid time_de_inicio** e ao servidor uma semelhante em que apenas não é enviado o campo entre o Running e o pid. Por fim, faz o exec pretendido.

Do lado do pai este lê o valor do tempo inicial que recebe através do pipe **heranca** e usa novamente a função gettimeofday() para calcular o tempo entre o início e fim da execução do programa desejado, preparando à semelhança do que o filho fez mensagens para informar o servidor e o utilizador da finalização da execução do programa, neste caso muda o último argumento que passa a ser o tempo de duração total do programa que é enviado para o utilizador que solicitou o pedido.

Comunicação do lado do Servidor

O servidor tem como principais objetivos guardar os pedidos ainda em execução em memória e registar as informações relativas a estes em ficheiro após a indicação de término da sua execução por parte do programa cliente responsável.

Inicialmente, cria uma lista para os pedidos e, posteriormente, cria e abre em modo de leitura o pipe com nome denominado **canal** por onde recebe dados enviados pelos clientes.

Ao ler do **canal** faz o parsing das mensagens e verifica se, de acordo com o primeiro argumento das mesmas, é um **Running**, um **Ended**, um **status** ou um **close**.

No caso de receber o **Running**, passa à função **adicionar_pedido** a lista dos pedidos, o pid do processo do cliente responsável pela mensagem, a string com os comandos a serem executados, o tempo de início de execução, e o número de comandos que contém.

A função **adicionar_pedido** está presente no ficheiro de código do servidor/monitor e é responsável por criar uma struct Pedido e a alocar na memória enquanto o pedido associado está a ser executado, esta struct pedido é criada usando como valores os parâmetros passados na função, depois de criada é alocada à cabeça da lista de pedidos.

Comunicação do lado do Servidor

Caso receba um **Ended** como primeiro argumento da mensagem este abre um ficheiro de texto com o nome do pid responsável pela mensagem onde guarda as informações relativas à execução feita por este processo que lhe enviou a notificação de término da execução. Para calcular a duração do pedido ao remover da memória o pedido através da função **remover_pedido** recebe desta o tempo de início da execução e consegue então calcular a diferença entre o momento de agora e esse momento inicial. Por fim, escreve isto no ficheiro ficando assim registadas informações de mais um pedido realizado e fecha o ficheiro.

A função **remover_pedido** à semelhança da função **adicionar_pedido**, está no ficheiro de código do monitor e a sua função é remover da lista de pedidos que recebe como parâmetro o pedido com o pid recebido também nos parâmetros. Para utilizar código de modo eficiente a função retorna a string com o tempo inicial de execução do pedido.

Se o primeiro argumento for então um **status** é aberto um **canal_status** que serve de comunicação entre o servidor e o processo que solicitou o status através da mensagem, daí o envio do pid do processo escritor na mensagem, para abertura do **canal_status**. Depois, percorre toda a lista de pedidos na memória e calcula a diferença entre o tempo atual e o gravado no campo timeStamp do pedido que corresponde ao tempo de início da execução. Assim para cada pedido escreve para o **canal_status** toda a informação dos pedidos ainda em execução que são contidos na lista.

Se por outro lado, o argumento é um **close** resta ao servidor mudar a variável **running,** que lhe permite correr continuamente enquanto é verdadeira, para falso enquanto indica que encerrará no stdout.

Diretoria fifos

Esta diretoria é usada para alocar todos os canais e os ficheiros criados em tempo de compilação pelos clientes e servidor de forma a melhorar a organização do projeto.

Conclusão

Por problemas relacionados com a capacidade de garantir o ambiente Unix em todas as máquinas do grupo, tornou-se complicado o processo de elaboração de todo o programa, dessa forma o grupo não conseguiu completar as tarefas avançadas com mais alguns dias teríamos conseguido fazê-lo, certamente. No entanto, parece ao grupo que as funções básicas cumprem os requisitos exigidos.