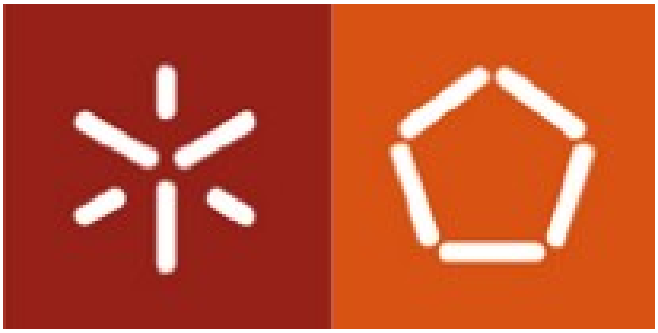

Universidade do Minho

Licenciatura em Engenharia Informática

Sistemas Operativos

Trabalho Prático I

Ano Letivo 2022/2023

Filipe Santos Gonçalves ,A100696

João Manuel Machado Lopes ,A100594

Tiago Nuno Magalhães Teixeira ,A100665

Introdução e objetivos

No âmbito desta unidade curricular foi-nos proposto um serviço que seja capaz de rastrear e monitorizar a execução de programas numa máquina.

Para isso foram desenvolvidos dois programas, um “tracer”, que executa um programa ou programas em pipeline passado como argumento e envia informações sobre o processo associado à sua execução antes e depois deste terminar, e um programa “monitor”, que cria um “fifo” com o intuito de permitir a comunicação entre este mesmo processo e o “tracer” anteriormente referido, bem como recebe as mensagens enviadas sobre os processos executados pelo “tracer” e armazena as mesmas em estruturas de dados que foram criadas com o intuito de serem eficientes e permitirem a rápida consulta de processos a decorrer e terminados.

Funcionalidades Básicas

Execução de programas do utilizador

Tendo como objetivo o Tracer (cliente) executar um programa de um utilizador e comunicar ao Monitor (servidor) o estado dessa execução sendo que este último também a armazena para a disponibilizar se necessário, criamos uma struct que guarda toda a informação necessária (nome do programa a ser executado, o PID de processo, o timestamp da mensagem e o seu tipo).

Quando a opção “execute -u” é chamada é utilizado um pipe anónimo, para comunicar entre o filho e o pai do processo Tracer. Desta forma, podemos criar um timestamp imediatamente antes da execução do programa passado como argumento para o filho, garantindo assim a fidedignidade do tempo de execução posteriormente calculado.

De forma a estabelecer contacto com o servidor, o Tracer abre um “fifo” que é criado pelo servidor quando este é iniciado, permitindo assim a troca unidirecional de informação entre os vários programas Tracer e o servidor. Importante realçar que, com o intuito deste fifo não se fechar aquando de um programa terminar, abrimos, no servidor, o fifo com as opções de leitura e escrita, nunca usando as de escrita.

Da perspetiva do Tracer, este envia mensagens antes e depois de executar os programas com informações sobre o PID do programa em execução, o nome do programa, o timestamp e o seu tipo, como mencionado anteriormente. Por sua vez, o servidor desconhece se estas correspondem a mensagens antes ou depois da execução e desconhece também o processo que as enviou. De forma a contrariar este problema, primeiramente, o servidor verifica que se trata da execução de um programa e não de algum tipo de status (através do campo “tipo das mensagens) e depois faz um lookup à hashTable responsável por guardar os programas que ainda estão a ser executados. Caso seja uma mensagem correspondente ao início da execução, este vai adicioná-la à HashTable e, caso corresponda a uma mensagem de que o programa terminou, este vai remover a mensagem de inicialização do programa guardada na hashTable e vai criar um ficheiro (em que o nome corresponde ao PID do processo que realizou o pedido do cliente) onde guardará o nome do programa que terminou de executar e o tempo que demorou. Importante referir que a informação guardada no ficheiro é antes guardada numa struct “Store” e só posteriormente escrita para o ficheiro, de forma a facilitar a leitura do ficheiro caso necessário.

Em relação à opção “status”, o Tracer envia uma mensagem com o tipo 2 e com o seu PID de forma a notificar o servidor o que pretende executar. A partir desta informação, o servidor abre o fifo com o nome do PID do processo que pediu o “status”, criado pelo mesmo antes de enviar a mensagem, e percorre a hashTable, que guarda os programas que estão a ser executados, enviando um a um para o fifo. Posteriormente, o Tracer, apenas lê do fifo as mensagens e imprimi-as uma a uma no STDOUT_FILENO.

Funcionalidades Avançadas

Execução encadeada de programas

De maneira a um cliente suportar a execução de vários programas do utilizador em pipeline, o Tracer envia inicialmente uma mensagem com os dados do programa a executar, e depois cria (N_processos –1) pipes anónimos e N processos filhos. Os N processos filhos são criados de forma a ser possível a execução de todos os processos, juntamente com os seus argumentos.

Quanto aos pipes anónimos, estes servem para encadear os processos, isto é, para permitir que cada processo opere sobre o "resultado" do processo anterior. Para que tal aconteça, todos os processos exceto o primeiro devem ter o seu standard_input substituído pela parte de leitura do pipe_anónimo do processo anterior e todos os processos exceto o último devem substituir o seu standard_output pela extremidade de escrita do seu pipe, o que alcançamos com o uso da função "dup2" pela parte do filho. Cada filho depois fecha todos os pipes e executa o programa que lhe compete. O pai, que foi responsável por criar os pipes anónimos, espera pelos filhos e também ele fecha todos os pipes.

Semelhante a quando o Tracer executa o programa "execute -u", antes da execução da pipeline e depois, o Tracer envia mensagens a notificar o servidor.

Armazenamento de informação sobre programas terminados

Aquando do fim da execução de cada programa, é criado um ficheiro cujo nome é o PID do processo que terminou, onde são guardados o nome e o tempo de execução do programa.

Consulta de programas terminados

Em primeiro lugar, relativamente ao "stats-time", o Tracer cria e o servidor estabelecem dois fifos entre si, ambos unidirecionais, um no sentido do tracer para o monitor e outro em sentido inverso. Desta forma, o Tracer envia os PIDs dos dos programas terminados, e o servidor devolve os tempos de execução dos mesmos através do outro pipe, conferindo sempre se o PID enviado pelo Tracer corresponde a um programa terminado. Posteriormente, o Tracer, apenas lê do fifo, que recebe pacotes do servidor, os tempos dos programas que solicitou e soma-os, terminando por imprimir o resultado no STDOUT_FILENO.

Relativamente ao "stats-command" este procede de forma semelhante, criando na mesma dois fifos e enviando os PIDs dos programas para o servidor. De forma a dar a conhecer ao servidor o programa que pretende conhecer o número de execuções, o Tracer passa esta informação aquando do envio da mensagem que notifica o servidor que está a ser solicitado um "stats-command". Por fim, o servidor apenas envia, o número 1 quando encontra uma execução do programa e, depois, o Tracer simplesmente conta o número de uns recebidos.

Por último, relativamente ao "stats-uniq", a única diferença é que de forma ao servidor evitar enviar o nome do mesmo programa múltiplas vezes para o Tracer, este cria um array dinâmico de forma a ir conferindo se um nome já foi enviado ou não.

Dificuldades enfrentadas

As principais dificuldades sentidas pelos membros do grupo aquando da criação do projeto foram as questões relativas aos bloqueios dos processos associados às leituras e escritas em pipes, bem como a formulação de um raciocínio eficiente para o envio e receção de diferentes informações sobre diferentes processos sem problemas de conflitos.

Conclusão

Ao implementar um serviço de monitorização de programas executados numa máquina cremos que apresentamos uma solução eficiente e sólida para atender aos requisitos pedidos. A arquitetura cliente-servidor juntamente com o uso de pipes/FIFOs, permitiu uma interação fluida entre os mesmos.

Durante a realização deste projeto, foi possível compreender a importância que os processos têm na criação de um sistema de monitorização de programas, assim como o papel que os pipes/FIFOs apresentam na comunicação/ sincronização de processos distintos e na implementação de pipelines.

Em suma, foram aprimoradas as competências relativas à unidade curricular de sistemas operativos, nomeadamente através da criação de funções que simulam as system calls, o que nos permitiu “estar mais próximos” das nossas máquinas e perceber melhor como elas funcionam.