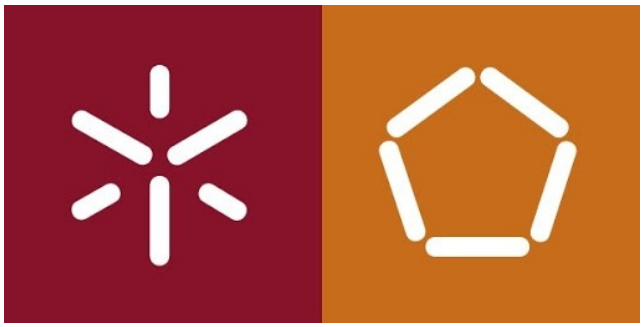

Controlo e Monitorização de Processos e Comunicação

Sistemas Operativos 2019/2020

(Grupo 105)

Alexandre Ferreira Gomes A89549

Marco Avelino Teixeira Pereira A89556

Manuel Jorge Mimoso Carvalho A69856

Introdução

Para o desenvolvimento deste trabalho o nosso grupo queria não só explorar os conteúdos aprendidos nas aulas práticas de forma mais profunda do que a feita nos guiões mas também de forma mais autónoma, sem o professor connosco para garantir que vai correr tudo bem.
Para isso deparámo-nos com alguns problemas, estes foram resolvidos após algum tempo gasto a formular uma solução funcional mentalmente transpondo depois para código, e também com a ajuda dos docentes em algumas questões pontuais.
Uma das maiores preocupações na realização deste trabalho foi, não ter o nosso servidor em espera ativa até que um cliente fizesse um pedido e também manter a concorrência de vários comandos a executar ao mesmo tempo, a nosa solução será explorada com maior detalhe nas secções dedicadas à estruturação e às funcionalidades.

Estruturação

A nossa primeira preocupação foi como seria de esperar a estruturação do projeto, isto é, as variáveis que seriam globais e que iriam ter os registos necessários de modo a possuir sempre o controlo total daquilo que se encontrava a acontecer no momento.

Para isto tivemos de tomar logo no início uma grande decisão, guardar todos estes valores em heap podendo expandir o tamanho deles, dando uma garantia que a única forma de ficar sem espaço era quando não houvesse alternativa, ou guardar tudo na stack, de forma a garantir um acesso muito rápido à informação mas perdendo o aspeto de durabilidade do servidor, podendo este causar o tão conhecido segmentation fault assim que o número de tarefas a executar ao mesmo tempo fosse maior do que o tamanho designado.
Decidimos assim confirmar com um docente se existia a necessidade de usar a heap ou se se poderia desenvolver o projeto apenas na stack, tomando as devidas precauções, após uma conversa com o docente decidimos optar por usar a stack.

Tendo isso decidido tinhamos agora que decidir as variáveis globais a criar, as mais óbvias foram simples, tempo_inatividade e tempo_execução, que iriam guardar respetivamente o tempo máximo de inatividade e execução, foram criados também o ntarefas que irá guardar o número de tarefas atual para permitir a etiquetação, falta-nos ainda uma matriz de chars que seria no fundo um array de strings para poder guardar o comando que o user deu de forma a guardar este mais tarde no histórico.
Posto isto já só nos falta definir o local onde guardar os vários pids das tarefas a executar e etc, para isto criamos uma matriz de nome pid em que o indice 0 corresponde ao pid do pai da tarefa enviada, o indice 1 corresponde ao número da tarefa, o 2 corresponde ao número de filhos que estão em execução dentro do pai da tarefa, o 3 corresponde ao número de comandos e por fim todos os outros correspondem aos pids dos vários filhos da tarefa.

Funcionalidades

Funcionalidades mínimas

tempo-inatividade ou -i:

A realização desta foi bastante simples sendo apenas necessário alterar o valor da variável global tempo_inatividade pelo valor que o cliente fornecer.

tempo-execução ou -m:

Esta funcionalidade foi também bastante simples de implementar seguindo apenas o conceito da anterior.

executar ou -e:

Esta foi certamente a funcionalidade mais complicada de implementar pois foi necessário garantir que todos os comandos da tarefa fornecida pelo utilizador iriam funcionar ao mesmo tempo e que punhamos em prática as 2 funcionalidades faladas anteriormente.

Nesse sentido o raciocínio por nós utilizado foi, após todo o parsing dos comandos, criar um filho que designaremos ao longo do trabalho como pai da tarefa, o pid deste será como anteriormente falamos guardado no índice 0 da linha correspondente a este da matriz pid, a primeira função deste é lançar um alarm com o valor do tempo de execução, depois disso o seu código fica responsável por criar todos os filhos, tal que cada um destes esteja responsável por, executar um dos comandos ou verificar a inatividade do pipe de um destes, o código utilizado para executar os comandos é igual ao utilizado no guião prático não valendo a pena referir, por outro lado o código de verificação de inatividade apesar de seguir o mesmo esquema, procede ao mapeamento, dentro do filho responsável, do sinal SIGALRM para uma função que o identifique como uma tarefa que irá terminar por ter ultrapassado o tempo de inatividade, após isto apenas terá de ler do pipe onde o exec escrever para um outro reativando o alarme sempre que este consiga ler.

Desta forma garantimos que não só os comandos de cada tarefa executam concorrentemente mas também que as várias tarefas que o cliente passe executam ao mesmo tempo, cumprindo assim um dos maiores objetivos deste trabalho.

listar ou -l:

Esta funcionalidade usufrui da matriz pid, já referida anteriormente, percorrendo todas as linhas desta e verificando se o valor em pid[linha][0] é maior do que 0, se isto se verificar significa que temos uma tarefa em execução, após isto vai ao índice 1 dessa linha para conseguir obter a tarefa atribuida a esse pid enviando assim uma mensagem para o cliente para este saber que a tarefa está em execução.

terminar ou -t:

Esta é no fundo uma segunda parte da funcionalidade executar, utilizando o mesmo esquema de quando uma tarefa é terminada por tempo de execução/inatividade, nesse sentido esta envia um sinal, diferente dos de inatividade e execução, ao pai da tarefa que o utilizador pediu para terminar e todos estes irão estar mapeados para a mesma função, esta função é terminate. A função terminate irá em primeiro lugar enviar um sinal de paragem a todos os intervenientes da tarefa e depois escrever no ficheiro temporário com o nome do pid do pai da tarefa, as informações relevantes a esta que são a forma como morreu na primeira linha e o resto das linhas irá conter o que esta tarefa escreveu até ser terminada, por fim esta irá enviar o sinal SIGKILL a todos os filhos do pai da tarefa e também ao pai da tarefa, dando um reset à linha por estes utilizada na matriz pid.

A parte final da terminação de uma tarefa passa por um signal definido no servidor quando este é iniciado esse signal irá mapear a morte de um filho para uma função que irá escrever no ficheiro de logs e de

histórico onde se irá encontrar a informação de todas as tarefas, lendo de um ficheiro com o pid da tarefa que terminou e eliminando no fim esse ficheiro temporário.

histórico ou -r:

Esta funcionalidade é bastante simples de perceber e implementar devido à forma como guardamos o histórico, sendo ele guardado num ficheiro a única preocupação que temos é ler deste e enviar para o cliente.

ajuda ou -h:

A funcionalidade de ajuda é apenas uma mensagem enviada ao cliente que possui a infornação dos vários comandos que este pode utilizar.

Funcionalidade adicional

output ou -o:

Esta funcionalidade foi sendo explicada ao longo das outras, temos um ficheiro índice preenchido com 3 inteiros por linha, o primeiro é o número da tarefa para a poder identificar, o segundo é o primeiro byte onde o output da tarefa começa e o terceiro é o último byte escrito pelo output da tarefa. Sabendo isso basta fazer um lseek dentro do ficheiro de logs para encontrar a posição de onde começar a ler e ler (fim - inicio - 1) bytes, enviando no fim o que leu ao cliente.

Outras funcionalidades implementadas

clean ou -c:

Esta funcionalidade foi implementada para facilitar os testes feitos, pois como o servidor implementado irá guardar a informação de modo a poder retornar ao estado onde estava anteriormente.

A forma de funcionamento dela é terminar todas as tarefas que estão a correr e limpar os ficheiros onde estão salvos os dados. Provocando assim um fresh start do servidor.

quit:

Como o nome diz esta funcionalidade serve apenas para fechar o cliente podendo facilmente testar se o servidor se encontra em espera ativa ou passiva sem ter de recorrer ao uso de ^C ou ^Z.

backup ou -b:

Esta funcionalidade serve para criar um backup do estado atual do servidor, criando dentro da pasta backups uma nova pasta com o nome da data do momento em que o utilizador pediu o backup e criando dentro ficheiros com os dados que estão carregados no servidor atualmente.

fill ou -f:

Como seria de esperar após a criação de uma funcionalidade de backup procedemos à criação de uma funcionalidade que permita ao cliente carregar, para o estado atual, dados guardados anteriormente através da funcionalidade backup anteriormente referida.

Testes Realizados

De modo a facilitar o teste do nosso programa decidimos criar um script que corria diversos comandos e com diferentes formatos, a imagem abaixo apresentada tem o objetivo de cada comando documentado após este.

```
make argus
gcc -o p1 p1.c
gcc -o p2 p2.c
sleep 0.5
./argus -c #cleaning all data for fresh start
echo -e "\n"
./argus -r #shows history (should be empty)
echo -e "\n"
./argus -l #shows list (should be empty)
echo -e "\n"
./argus -e 'ls | sleep 1000' #task that takes a lot of time
echo -e "\n"
sleep 1
./argus -t 1 #terminates task that takes a lot
echo -e "\n"
./argus -e 'sleep 3 | ./p2' #task that writes and isnt first
echo -e "\n"
./argus -e 'cut -f7 -d: /etc/passwd | uniq | wc -l' #executes 2 piped task
echo -e "\n"
./argus -e 'ls > ali' #sends output to file
echo -e "\n"
sleep 1
./argus -e 'wc < ali > aqui' #reads and send to file
echo -e "\n"
./argus -l #lists running proceses (should be only task 2 running)
echo -e "\n"
./argus -i 3 #sets inactivity to 3
echo -e "\n"
sleep 1
./argus -e './p1' #runs program that prints every 2 seconds (wont end)
echo -e "\n"
./argus -e './p2' #runs program that prints every 4 seconds (will end without any prints)
echo -e "\n"
./argus -m 1 #execution time to 1
echo -e "\n"
sleep 1
./argus -e 'sleep 1000' #runs command that takes a lot
echo -e "\n"
./argus -l #lists tasks (should be only 2 and/or 6 and/or 7 running and/or 8 running)
echo -e "\n"
./argus -o 3 #shows output of task 3
sleep 5 #sleeps to make sure that tasks 7 and 8 end
echo -e "\n"
./argus -t 6 #terminates task 6 because it would never end
echo -e "\n"
sleep 1
./argus -o 6 #shows output of task 6 (all the prints she made until she was killed)
echo -e "\n"
./argus -r #shows history (all tasks should appear here except task 2)
echo -e "\n"
./argus -h #shows help
echo -e "\n"
./argus -b #creates backup
echo -e "\n"
./argus -t 2 #ends task 2 that was writing every 4 seconds since the beginning
echo -e "\n"
./argus -l #não devem aparecer tarefas
echo -e "\n"
./argus -e 'cat file | wc' #ficheiro que contem um script de um filme
echo -e "\n"
sleep 1
./argus -o 9 #should show wc aplied to the script
make clean #cleans all created files
```

Figure 1: Script de testes

Reflexões críticas

Embora estejamos satisfeitos com o resultado obtido existem pontos que desgostamos mais no trabalho como a organização deste.

Durante a resolução deste deparámo-nos com várias barreiras, as quais tivemos de ultrapassar sozinhos e que puseram à prova o aprendido nas aulas práticas, a maior dificuldade foi embora depois de feito seja fácil de entender, a espera passiva no servidor de modo que este não fosse percorrer um while (1) infinitamente até que um cliente o tentasse utilizar abrindo o descritor de escrita para o fifo de input, passamos por várias versões, umas mais prudentes do que outras mas atingimos aquela que na nossa opinião é a real espera passiva entre cliente e servidor, tanto no input como output. Um outro desafio mas certamente menos desafiador foi um que ainda não conseguimos resolver completamente mas que iremos posteriormente tentar, que é abilitar o uso de comandos como cat sem argumentos tal que o input do cliente vá sempre para este e que o servidor se mantenha parado à espera que este comando seja temrinado pelo utilizador, a ideia atual seria duplicar o descritor de leitura e enviar o sinal SIGSTOP ao servidor mas isto iria fazer com que tivesse de haver uma diferenciação de quando o comando fornecido fosse cat ou wc ou outros que não precisem de input e fossem causar uma pausa no servidor.

A maioria dos problemas de menor escala que nos foram aparecendo eram facilmente solucionados com o uso de outras funções as quais não fomos introduzidos na Unidade Curricular, exemplo destas seriam sigaction e sigqueue que nos iriam permitir em conjunto com o “trigger” de um sinal, enviar informação relevante a este, como a forma de morte ou a tarefa, entre outros.

Conclusão

Na nossa opinião o resultado final do trabalho foi satisfatório dando agora a possibilidade de implementar vários clientes e cada um ter os seus dados, e.t.c..

Os principais objetivos deste foram cumpridos tendo ainda implementado algumas funcionalidades extra que dão algum jeito quando nos encontramos a testar o programa, além destas conseguimos garantir que mesmo que o servidor fosse morto a qualquer momento este iria, quando reiniciasse, carregar para o estado atual a informação contida nos ficheiros, estes ficheiros estão sempre a ser atualizados para garantir que se perde o mínimo de informação possível quando o servidor morre.

Como ponto final gostaríamos de agradecer aos docentes pela disponibilidade para sessões de esclarecimento de dúvidas que tivemos ao longo da realização deste.