Universidade do Minho
Mestrado Integrado em Engenharia Informática
Licenciatura em Ciências da Computação

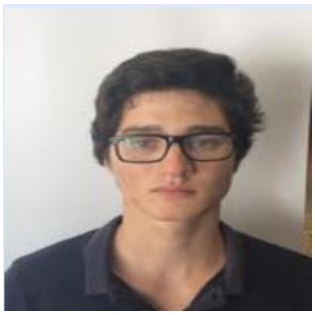
Sistemas Operativos

Ano Letivo de 2019/2020

Client/Server

Gonçalo Pinto Nogueira
A88617

João Nuno Costa Neves
A81366



Carlos Manuel Marques Afonso
A82529

Resumo

O trabalho proposto no âmbito da disciplina de Sistemas Operativos, tendo como objetivo implementar e consolidar os conceitos adquiridos no decorrer do semestre.

O projeto apresentado de seguida é composto essencialmente por duas partes, o cliente, onde são enviadas tarefas, e o servidor, que é responsável por receber, analisar, executar e também guardar em memoria o registo das tarefas enviadas pelo cliente.

Por isto, o presente relatório tem como objetivo explicar de uma forma sucinta todos os passos e todas as decisões, que em conjunto, foram tomadas de forma a fazer o melhor projeto possível.

Índice

Conteúdo

Índice de Ilustrações

Cliente/Servidor

O presente trabalho elabora uma estrutura de Cliente <-> Servidor, cuja implementação é feita com base em “pipes” com nome, que são responsáveis pela comunicação entre estes dois, sendo que decidimos implementar dois “pipes” com nome, um encarregado de enviar a informação do cliente para o servidor, e o outro encarregado de enviar a informação do servidor de volta para o cliente.

Do lado do servidor, sempre que recebe um pedido, é iniciado um processo de forma a tratar desse pedido. No entanto, o processo pai não espera que o processo filho seja terminado, pois assim não seria obtida a concorrência. Numa primeira fase, pensamos que não haveria forma de eliminar os processos zombies que seriam criados, mas eventualmente, conseguimos dar a volta a este problema. A solução passa pelo seguinte. Cada vez que o processo filho termina, é feito um sinal SIGCHILD, que permite ao processo pai terminar esse processo, não tendo assim de esperar que este termine, garantindo a concorrência, e a boa terminação dos processos filhos.

Do lado do cliente, caso o cliente seja chamado sem argumentos, é criado um processo filho que apenas estará a ouvir as respostas do servidor. O processo pai fica então encarregue de enviar os pedidos que o utilizado deseja, garantindo assim que o cliente pode enviar vários pedidos, sem ter de esperar pela resposta. Caso o cliente seja chamado com argumentos, o pedido é feito ao servidor, e o cliente fica em espera até obter a resposta.

Definir o tempo Máximo de inatividade num “”pipe” anónimo

De forma a que o servidor saiba qual é o tempo de inatividade entre “pipes” anónimos, foi criada uma variável global, de forma a todos os processos terem acesso a ela. Quando o servidor é iniciado, tem associado um tempo de execução inicial definido.

Com o objetivo de responder a este ponto, numa tarefa com “pipes” anónimos, é adicionado um novo “pipe” entre cada comando da tarefa, para verificar o fluxo entre as respostas de cada comando. Em cada processo associado à verificação de fluxo, é feito um alarme, de forma a que, caso não haja fluxo, esse alarme é disparado, e o tratamento desse sinal termina a tarefa. Caso realmente haja resposta, ao ler o fluxo num loop até não haver mais nada para ler, o alarme é sempre reiniciado, garantindo assim, que o tempo de inatividade não é ultrapassado.

Decidimos fazer esta implementação, pois achamos que era uma boa forma de verificar se existe fluxo entre os pipes, e também se mostrou uma uma implementação simples.

Definir o tempo máximo de execução de uma tarefa

Para este ponto, tal como no tempo de inatividade, foi definido uma variável global, para que todos os processos tenham acesso a ela.

Também esta inicia com um valor default, e, caso assim deseje, o cliente poderá alterá-la.

Quando uma tarefa é iniciada, no processo onde se vai tratar de executar essa tarefa, é iniciado um alarme, de forma a que, caso a tarefa ultrapasse o tempo de execução, é enviado um sinal SIGALARM, semelhante ao caso de tempo de inatividade, cujo handler tratará de terminar o processo.

Executar uma tarefa

Para executar uma tarefa, antes de tudo, é verificado se a tarefa faz uso de “pipes” anónimos. Caso não use, é feito um processo, onde é redirecionado o “stdout” para o extremo de escrita do “pipe” com nome encarregado de enviar informação para o cliente, e de seguida, é executado a “System Call” execvp.

Caso realmente faça uso de “pipes”, são criados “pipes” anónimos. De seguida são criados processos, um para cada comando da tarefa, e um para cada verificação de fluxo entre “pipes”. Os descritores são alterados, de forma a que cada resposta de cada comando seja entregue ao processo correspondente, e por fim, para o ultimo comando, o stdout é redirecionado para o extremo de escrita do pipe com nome, de forma a resposta ser entregue ao cliente.

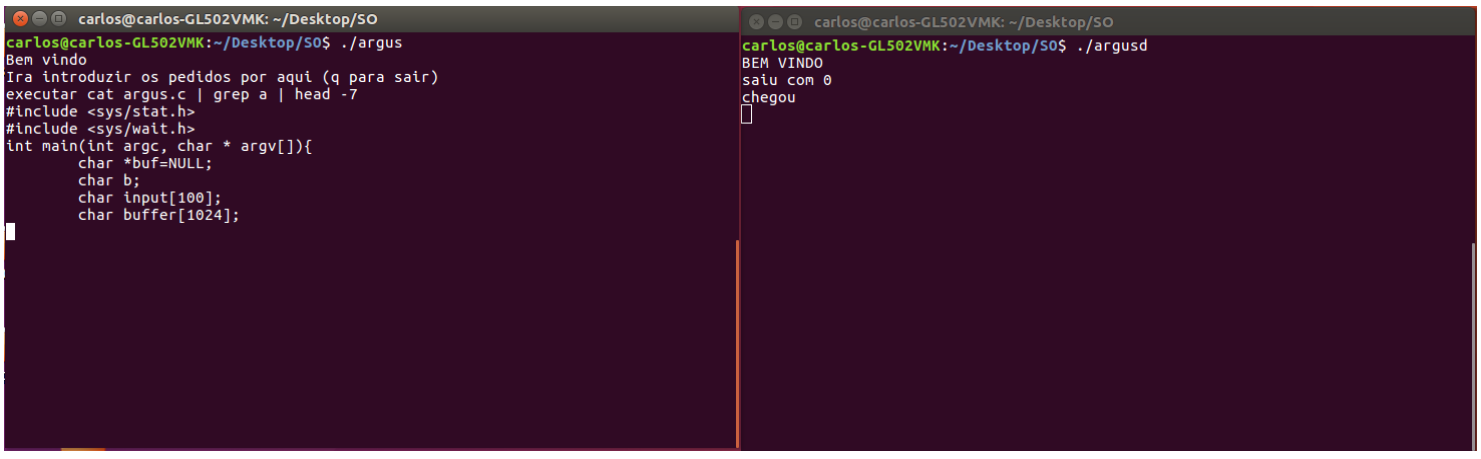


Figura 1 Executar Tarefa

Listar tarefas em execução

Neste ponto, o grupo discutiu bastante, de forma a tentar chegarmos à melhor implementação possível. Inicialmente, foi pensado em escrever as tarefas em execução em ficheiro, e, à medida que estas eram terminadas, iriam ser removidas do ficheiro. Desta forma, o cliente, quando pedisse a lista das tarefas em execução, o servidor simplesmente iria ler tudo o que existia no ficheiro. No entanto, isto mostrou-se ser demasiado complicado, e também achamos que não era uma boa alternativa.

Após mais um pouco de discussão, decidimos então, guardar em memoria, pois iria ser muita mais simples a implementação. Foi decido, então, que, cada vez que uma tarefa é iniciada, esta é guardada, em conjunto com o seu PID, num “char**”. À medida que uma tarefa é iniciada, o servidor aumenta o índice de tarefas a executar e coloca no índice seguinte da estrutura. Por fim, quando a tarefa é terminada, antes do _exit é enviado um sinal ao servidor, que tem como resposta a eliminação do processo correspondente ao PID que enviou o sinal, daí também termos decidido guardar o PID na estrutura, e o decremento do índice de tarefas em execução.

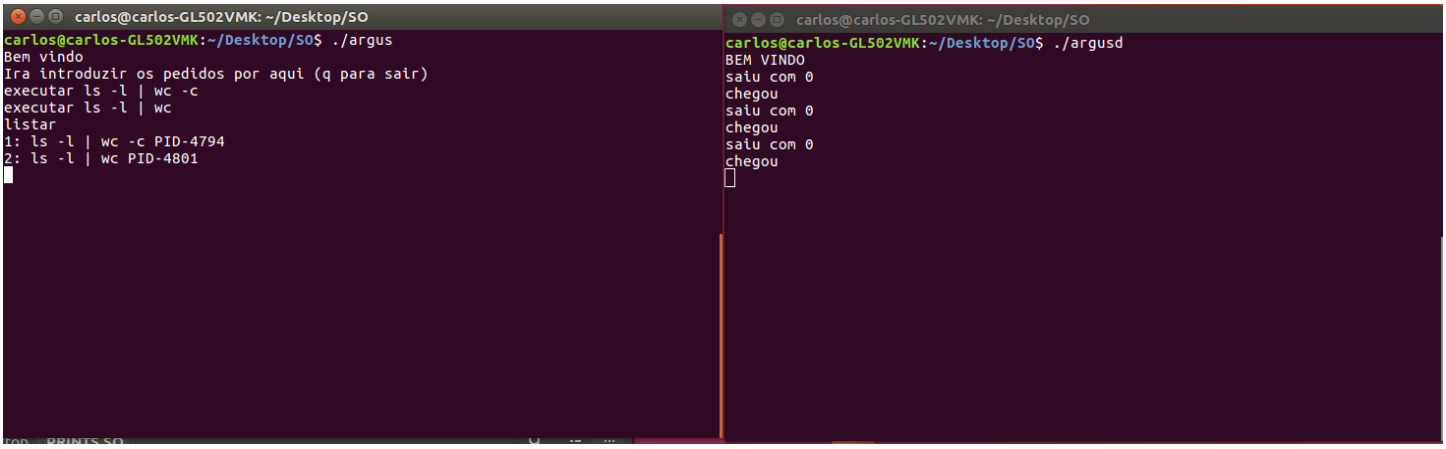


Figura 2 Lista de Tarefas em execução

Terminar uma tarefa em execução

Infelizmente, este ponto não foi implementado com sucesso. A abordagem que tentamos implementar passa por enviar um sinal ao pid que o cliente deseja terminar, sendo que o handler deste sinal, o mesmo de terminar uma tarefa por falta de tempo de execução. No entanto, isto não se mostrou eficaz, uma vez que, apesar do processo desejado ser terminado, não é feito com sucesso a eliminação desse processo da lista de tarefas em execução, e os outros processos ficam “presos”. Posto isto, não consideramos que este ponto tenha sido efetuado com sucesso.

Listar registo histórico de tarefas terminadas

Sempre que o servidor inicia, cria o ficheiro “logs.txt” que vai guardar todas as tarefas terminadas, distinguidas pela forma como foi terminada.

Quando uma tarefa é terminada, o processo pai do processo que executou a tarefa obtém o “WEXITSTATUS” do pid filho, e, consoante o valor que obtém, escreve a forma como terminou, ou seja, se terminou porque o tempo de execução acabou, se o tempo de inatividade foi atingido ou se foi concluído com sucesso. Quanto ao registo de terminar uma tarefa a pedido do cliente, este não é registado, uma vez que a implementação desse ponto não foi bem-sucedida.

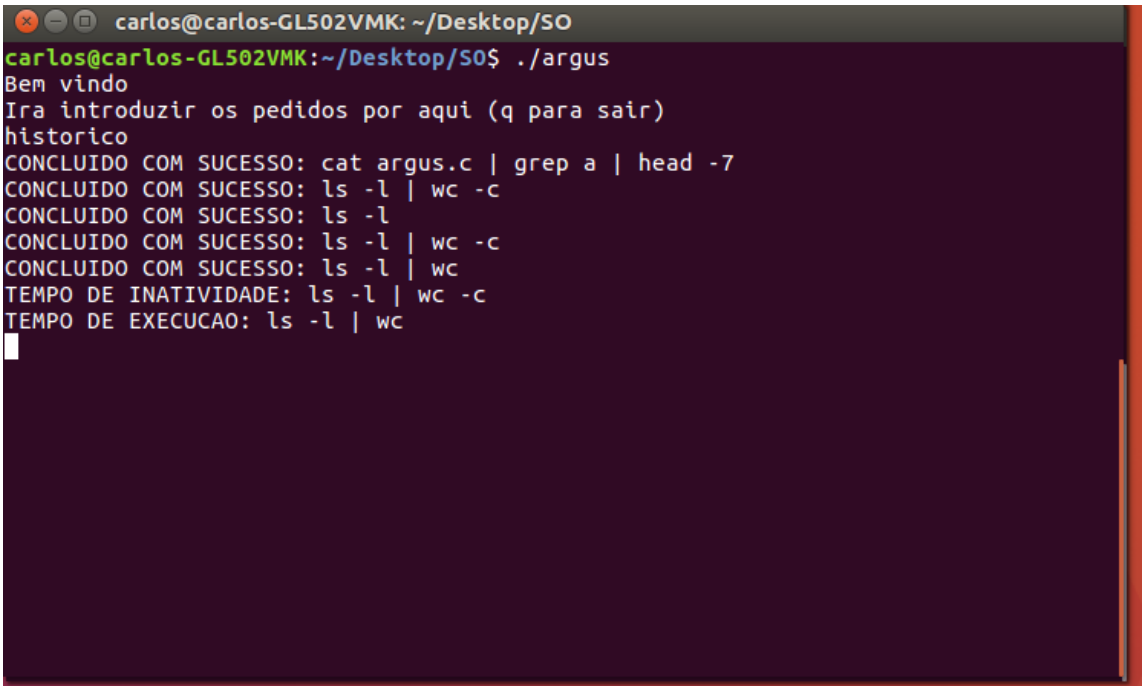


Figura 3 histórico de tarefas terminadas

Apresentar ajuda à sua utilização

Este ponto foi, sem dúvida, o mais simples de implementar, uma vez que apenas foi necessário escrever para o “stdout” os comandos que podem ser introduzidos.

4-Conclusão

Para concluir é de salientar o desafio que foi este projeto mas também a importância que tem para a compreensão daquilo que foi lecionado ao longo do semestre na cadeira de Sistemas Operativos.

Com este projeto evoluímos não só como programadores, mas também a nível de raciocínio e ganhando também experiência na linguagem C, nomeadamente na manipulação de System calls.

Durante a realização do projeto fomo-nos apercebendo de que algumas das soluções que tínhamos encontrado não eram as mais corretas e fomos assim alterando código e adaptando-nos sempre à situação.

Sofremos um pouco devido ao facto de termos inciado a implementação de certos pontos, que, deveriam ter sido implementados por ultimo, pois foram pontos que levaram a nossa implementação inicial mostrar-se errada.

Em suma, podemos dizer que a nossa prestação foi satisfatória, tendo alcançado a maior parte dos objetivos propostos e implementadas as funcionalidades solicitadas.