Departamento de Informática

Escola de Engenharia

Universidade do Minho

Orquestrador de Tarefas

Trabalho Prático

Sistemas Operativos

Licenciatura em Engenharia Informática

2023/2024

Grupo 1

Ana Cerqueira, a104188

Nuno Silva, a104089

Eduardo Faria, a104353

07/05/2024

Índice

1. Introdução

Numa área como a nossa, em constante progresso e evolução, é fundamental ter um conhecimento aprofundado sobre a forma como os computadores operam, de modo a encontrar soluções que nos permitem melhorar os nossos programas.

Com este trabalho tivemos a oportunidade de colocar em prática alguns dos principais tópicos abordados na cadeira de Sistemas Operativos. Entre eles, a coordenação de interações entre programas e a gestão de recursos e exploração de técnicas para melhorar a eficiência e rapidez dos mesmos.

Neste contexto, desenvolvemos um programa que coordena a execução e escalonamento de tarefas num computador. Os utilizadores usam o programa cliente para solicitar ao servidor a execução de uma tarefa, indicando-a juntamente com a sua duração expectável em milissegundos. O cliente envia essa tarefa para o servidor que trata do seu escalonamento e execução. O cliente tem ainda acesso à lista de tarefas em espera, em execução e terminadas.

2. Funcionalidades disponíveis no cliente

Pedido de execução de tarefas

O programa cliente pode receber as tarefas do utilizador a partir da execução do programa com os seguintes formatos:

```
./client execute time –u “prog [args]”
./client execute time –p “prog-a [args] | prog-b [args] | prog-c [args]”
```

Através destes formatos, o programa cliente cria, inicialmente, uma mensagem com os seguintes parâmetros: *pid, time, is_pipe, program e type*. Posteriormente, envia a mesma para o servidor através de um FIFO, cujo nome se encontra globalmente definido como *“fifo_main”,* e, quando finalizado o envio, abre em modo leitura o FIFO *"fifo_x"*, onde x corresponde ao *id* do processo cliente que tratou de enviar a mensagem. Desta forma, o servidor consegue enviar o número da tarefa em questão, indicando assim ao programa cliente a confirmação de que o seu processo chegou corretamente ao servidor. Finalmente, o programa cliente pode terminar a sua execução, permitindo ao utilizador realizar novos pedidos.

Execução singular ou encadeada de programas

A partir do uso da opção "-u" ou "-p" o utilizador pode escolher a forma como os programas das tarefas devem ser realizados. Para tal, "-u" permite apenas a execução de um único programa por tarefa, ao passo que "-p" já permite a execução encadeada de programas numa tarefa (pipelines), tirando partido do uso do operador "|".

Consulta de tarefas em execução

O programa cliente pode requerer do estado das tarefas no servidor executando, para tal, o programa com o seguinte formato:

```
./client status
```

Através da opção "*status",* o utilizador tem a opção de listar as tarefas em execução, as tarefas em espera para executar, bem como as tarefas terminadas. Para tal, o cliente cria uma mensagem com o tipo STATUS e envia-a para o servidor onde ela vai ser tratada de acordo com o seu tipo. Deste modo, assim que o servidor recebe a mensagem do tipo STATUS, acede ao FIFO único e temporário, criado por parte do cliente, e envia os dados em questão. Faz-se uso de um *struct* do tipo *Msg_to_print*, de forma a uniformizar a escrita, leitura e processamento de dados. Por conseguinte, enviam-se as tarefas agendadas e em execução, fazendo uso das listas de mensagens disponíveis no servidor. Por último, enviam-se todas as tarefas concluídas, que se encontram guardadas num ficheiro geral denominado "tasks_info.bin", na diretoria providenciada pelo utilizador.

Término do servidor

O programa cliente consegue terminar graciosamente a execução do servidor, executando o programa da seguinte forma:

```
./client server-stop
```

Utilizando o opção "*server-stop"*, o utilizador pede ao servidor que pare de executar todas as tarefas e seja desligado. Para tal, uma mensagem do tipo STOP é enviada para o servidor fazendo-o fechar todos os ficheiros e encerrar o FIFO.

3. Funcionalidades disponíveis no servidor

O servidor é ativado através do comando:

```
./orchestrator output_folder parallel-tasks sched-policy <'test-mode'>
                                <num-tasks>
```

Este recebe as tarefas requisitadas pelo cliente sob a forma de estruturas de dados apelidadas por mensagens, processa as mesmas e executa os programas solicitados. Por outro lado, ainda há a possibilidade de abrir o servidor em modo de teste, acrescentando aos parâmetros obrigatórios outros dois opcionais: string "*test-mode*", seguida pelo número de tarefas teste a receber pelo cliente.

Execução de tarefas do utilizador

Inicialmente, o servidor cria um FIFO responsável por receber mensagens vindas geralmente do cliente. Este abre o FIFO em modo de leitura, onde permanecerá bloqueado até existir um outro programa a abrir o mesmo FIFO em modo de escrita, neste caso, o primeiro cliente. Crucialmente, o programa faz uma nova abertura do mesmo FIFO, desta vez em modo de escrita. Isto é feito para assegurar que, caso não haja um outro programa com o FIFO aberto em modo de escrita, o servidor permaneça ativo. Assim, o servidor irá operar continuamente até receber uma mensagem do tipo STOP ou até executar exatamente 20 tarefas quando em modo de teste.

Para a organização das mensagens recebidas pelos clientes, o servidor faz uso de uma estrutura *msg_list* que permite gerir as mensagens agendadas e as mensagens em execução.

Caso a mensagem recebida seja do tipo STATUS, devolve a lista das tarefas em execução, em espera e terminadas, como explicado anteriormente. Caso seja do tipo COMPLETED, recolhe-se o processo filho responsável pela execução da tarefa em questão, e remove-se a mesma da lista de tarefas em execução. Se porventura ela não corresponder a nenhum dos campos referidos anteriormente é porque estamos perante uma mensagem recém-chegada e, por isso, o seu número de tarefa é enviado para o cliente e ela é inserida na lista de mensagens agendadas onde, em conjunto com as outras mensagens, será ordenada consoante a política de escalonamento adotada.

Ficheiro com informações das tasks

Após uma tarefa ser dada como terminada os seus dados são guardados num ficheiro "tasks_info.bin". Este contém, para cada tarefa, a informação sobre o seu PID, o(s) programa(s) executado(s) e o tempo (em milissegundos) que foi gasto para executar a tarefa.

Políticas de escalonamento

Na ativação do servidor é dada a indicação sobre a política de escalonamento escolhida no parâmetro "*sched-policy"*. Neste projeto implementamos duas políticas de escalonamento: FCFS (*First-Come, First-Served*) e SJF (*Shortest Job First*). No FCFS, as tarefas são executadas segundo a ordem de chegada e no SJF as tarefas são executas de acordo com o tempo estimado para a sua duração, onde as tarefas mais curtas têm prioridade.

Por defeito, as mensagens são inseridas na lista de mensagens agendadas pela ordem de chegada e a próxima mensagem a ser executada será sempre a que se encontra na primeira posição da mesma. Assim, caso se pretenda uma abordagem segundo o SJF, a lista será reordenada, por ordem crescente, antes de procedermos à execução das tarefas. Na eventualidade da política escolhida ser FCFS, apenas se procederá à execução, não havendo necessidade de qualquer tipo de reordenação.

4. Testes

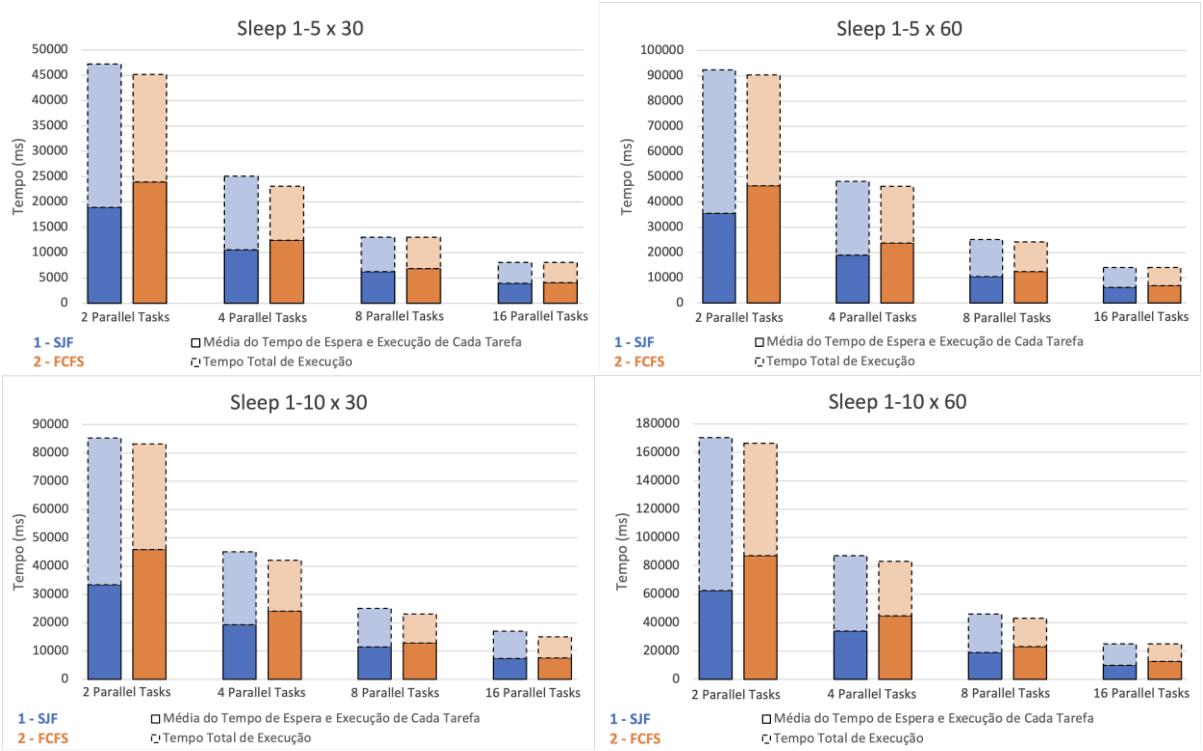
```
./run.sh num_tasks sleep_limit
```

Para perceber a eficiência das políticas de escalonamento implementadas no servidor, fizemos uso de um *script* responsável por enviar *num_tasks* tarefas diferentes, cada uma com a função *sleep* de tempo variável entre 1 e *sleep_limit*. Desta forma, conseguimos rapidamente simular diferentes pedidos por um cliente e averiguar o tempo de execução dos mesmos, isto quando o servidor está configurado para *test-mode.*

Assim, executaram-se cada um dos *scripts* "*run.sh 30 5*", "*run.sh 30 10*", "*run.sh 60 5*"e "*run.sh 60 10*" uma vez para 2, 4, 8 e 16 tarefas paralelas, para cada política de escalonamento (SJF e FCFS). Em cada teste, calculou-se o tempo total de execução das várias tarefas e a média do tempo de espera e execução de cada uma delas, estando os resultados da experiência representados abaixo:

SJF	Nr Parallel tasks	sleep 1-5 * 30	sleep 1-10 * 30	sleep 1-5 * 60	sleep 1-10 * 60
Tempo Total	2	47206	85221	92395	170350
	4	25107	45082	48170	87262
	8	13043	25056	25113	46131
	16	8072	17032	14058	25062
Tempo de Espera + Execução	2	18922,767	33421,667	35548,717	62562,65
	4	10577,933	19313,8	18913,617	34004,35
	8	6259,367	11396,167	10493,017	18826,75
	16	3963,633	7329,3	6143,383	9920,114

FCFS	Nr Parallel tasks	sleep 1-5 * 30	sleep 1-10 * 30	sleep 1-5 * 60	sleep 1-10 * 60
Tempo Total	2	45192	83177	90365	166354
	4	23106	42092	46174	83208
	8	13039	23056	24169	43113
	16	8060	15040	14047	25055
Tempo de Espera + Execução	2	23960,3	45817,467	46511,833	87125,217
	4	12450,267	24076,367	23691,833	44723,067
	8	6860,267	12834,9	12421,65	23122,767
	16	4097,733	7527,033	6854,433	12582,533



Com base nos dados dos gráficos, percebemos que, de modo geral, as duas políticas de escalonamento assemelham-se nos seus tempos totais de execução. Ainda assim, há que notar que a política de escalonamento SJF teve tempos de execução ligeiramente superiores. Isto dever-se-á, certamente, ao facto de que a política de escalonamento SJF requer a utilização de um algoritmo de ordenação da lista de mensagens agendadas, todas as vezes que uma nova tarefa é recebida pelo servidor.

Contudo, observando as médias do tempo de espera e execução de cada tarefa, concluímos que a política de escalonamento SJF sobressai-se pelo baixo tempo de espera e execução que promove, relativamente à política FCFS. Ainda assim, à medida que o número de tarefas paralelas aumenta, esta diferença vai se tornando cada vez menos percetível.

5. Decisões tomadas

De forma a simplificar o processo de variação de alguns valores impostos a diversos componentes do projeto, decidimos definir algumas constantes que, por sua vez, se encontram no ficheiro messages.h.

Numa outra vertente, após a abertura do FIFO principal no servidor, que será responsável por receber as mensagens dos clientes, escolhemos abrir esse mesmo FIFO em modo de escrita, de forma a bloquear o mesmo num estado de leitura. Isto é feito porque o servidor está constantemente à espera de mensagens vindas do cliente para processá-las e, ao abrir o FIFO em modo de escrita, asseguramos que nunca é recebido o valor 0 (EOF).

6. Conclusão

Este projeto apresenta um conceito de fácil compreensão, mas cuja execução é algo complexa. O mesmo permitiu a aplicação, a prática e a consolidação de diversos conceitos fundamentais da programação em C que foram abordados ao longo deste semestre. Entre eles destacam-se a utilização de FIFOs, utilizados na comunicação entre o cliente e servidor, a manipulação de forks, permitindo a programação paralela, e ainda a manipulação de descritores de ficheiros através de dups, como forma de centralizar a informação da execução de programas e encadeamento dos mesmos.