Universidade do Minho

Braga, Portugal

TRABALHO PRÁTICO - RELATÓRIO

Orquestrador de Tarefas

Sistemas Operativos

Departamento de Informática

Engenharia Informática 2023/24

Equipa de Trabalho:

A104537 - Afonso Pedreira

A104344 - Dário Guimarães

A104365 - Fábio Magalhães

1. Comandos

Texto

1.1. Cliente

1.1.1. Command Sender

Texto

1.2. Servidor

1.2.1. Command Interpreter

Texto

1.2.2. Command Runner

Texto

2. Pipes

Texto

2.1. Cliente

Texto

2.2. Servidor

Texto

3. Escalonamento de Tarefas

O escalonador é responsável por armazenar e escolher tarefas para o servidor executar. Esta seleção tem em conta a política de escalonamento previamente definida, e tenta obter o menor tempo possível de espera na fila. A nossa implementação do escalonador conta com duas políticas de escalonamento diferentes: `FCFS` e `SJF`.

3.1. Implementação

O escalonador, ou `scheduler`, está implementado de forma encapsulada e modular, sendo necessário inicialmente instancia-lo, e podendo depois ser adicionadas ou removidas tarefas, ou pedido um estado da fila com os respetivos métodos. Para exemplificar, a utilização do escalonador é feita da seguinte forma:

```
// Instanciação
Scheduler scheduler = create_scheduler(FCFS);

// Adição de uma tarefa
enqueue_process(scheduler, "ls /etc/ -la", 10);

// Estado do escalonador - lista de todas as tarefas na fila
Process* status = scheduler_status(scheduler)
```

```
// Escolha de uma tarefa para ser executada
Process p = dequeue_process(scheduler);

// Libertação de toda a memória alocada, bem como das tarefas pendentes
destroy_scheduler(scheduler);
```

Internamente o escalonador aproveita-se da modularidade e da interface bem definida para as políticas de escalonamento para facilmente ser expandido. Para cada método do escalonador deve existir um método equivalente na implementação da política, que será armazenado e usado com recurso a funções de ponteiro. De seguida podemos ver a estrutura do escalonador, com a assinatura dos seus métodos:

```
typedef struct queue *Queue;
typedef int (*EnqueueFunction)(Queue, Process);
typedef Process (*DequeueFunction)(Queue);
typedef Process* (*StatusFunction)(Queue);
typedef void (*DestroyFunction)();

typedef struct scheduler
{
  SchedulePolicy policy;

  Queue queue;

  EnqueueFunction enqueue_fun;
  DequeueFunction dequeue_fun;

  StatusFunction status_fun;

  DestroyFunction destroy_fun;

} *Scheduler;
```

3.2. Políticas de Escalonamento

3.2.1. First Come First Serve

A política `FCFS` consiste em escolher a tarefa que chegou primeiro à fila. Esta política é implementada com recurso a uma `circular queue`, que permite a adição e remoção de tarefas de forma fácil e eficiente. Assim, sempre que é adicionada ou removida uma tarefa não há necessidade de se mover os elementos ou alocar mais espaço para lidar com as alterações. Quando a capacidade realmente é atingida, a fila é redimensionada para o dobro e todos os elementos são copiados para o início da nova fila.

3.2.2. Shortest Job First

A segunda política disponível é a `SJF`, que escolhe a próxima tarefa com base no tempo de execução estimado. Para tal, é usada uma `min-heap` para se atingir a menor complexidade nas operações de adição e remoção de tarefas.

3.3. Performance

Para perceber que política se adequa melhor a cada cenário realizamos vários testes, onde nos focamos em variáveis como o númeor de tarefas em requesitadas, e o tempo médio que estas demoram.

Para automatizar estes testes desenvolvemos um script em `bash` que requesita um número variado de tarefas, com tempos de execução dados por ordem crescente, decrescente ou aleatório (mas com soma total do tempo igual, independetemente da ordem). Este script executa, intercaladamente, os programas fornecidos pelos professores no BlackBoard - `hello` e `void`.

```
$ bin/runner.sh <número de tarefas> <asc | desc | random> <tempo mínimo>
<tempo máximo>
```

3.3.1. Cenários

Os cenários que escolhemos tentam abrangir uma variadade de situações, em que podem existir várias tarefas curtas, longas, ou uma mistura de ambas.

Os resultados, apresentados, em segundos, nas seguintes tabelas, foram recolhidos do ficheiro `log.txt`, que lista o tempo que uma tarefa leva desde a entrada na fila até ao fim da sua execução, e depois tratados no ficheiro `statistics.ods`. Para a nossa análise é nos relevante apenas o tempo de espera, que pode ser calculado ao subtrair o tempo estimado de execução ao tempo total, uma vez que essa estimativa é bastante próxima da realidade nos programas usados.

3.3.1.1. 1º Cenário

Para o primeiro cenário, foram executadas 100 tarefas, com tempos de execução crescentes, descrescentes e aleatórios entre 1 e 100 segundos e com um máximo de 3 tarefas em simultâneo.

Soma dos tempos de execução	5050

	FCFS			SJF		
	Cresc.	Decresc.	Aleatório	Cresc.	Decresc.	Aleatório
Média	555.966022	1126.75302	804.277562	555.947148	604.905078	580.635962
Máximo	1665.3687	1731.0704	1694.9604	1665.1746	1668.2747	1663.6163

3.3.1.2. 2º Cenário

No segundo cenário, foram executadas 3 000 tarefas, com tempos de execução aleatórios entre 1 e 5 segundos e um máximo de 6 tarefas em simultâneo.

Soma dos tempos de execução	9000

	FCFS	SJF
Média	766.4419126	565.2882803

Máximo	1539.6931	1539.6927

3.3.1.3. 3º Cenário

No terceiro cenário, foram executadas 50 tarefas, com tempos de execução aleatórios entre 500 e 550 segundos e um máximo de 3 tarefas em simultâneo.

Soma dos tempos de execução	26275

	FCFS	SJF
Média	4265.13465	4195.026936
Máximo	8629.6755	8622.8741

3.3.1.4. 4º Cenário

No quarto e último cenário, foram executadas 500 tarefas, com tempos de execução aleatório, em que 250 tarefas têm tempos de execução entre 1 e 10 segundos, e as restantes entre 100 e 350 segundos e novamente um máximo de 3 tarefas em simultâneo.

Soma dos tempos de execução	57750

	FCFS	SJF
Média	9724.804148	4233.552515
Máximo	19674.6588	19520.8571

3.3.2. Conclusões sobre as políticas

Ao analisarmos as tabelas anteriores podemos tirar algumas conclusões sobre a performance de cada uma das políticas.

Comecemos por notar que no cenário 1, a FCFS tem uma performance péssima com tempos de espera médios, quando os tempos de execução vêem de forma decrescente. Este já era um comportamento esperado uma vez que estes processos maiores, que chegam primeiro, monopolizam totalmente os recursos. Ao analisarmos os tempos crescentes percebemos que estes são os melhores em performance. Logo, juntando estas duas análises, a ordem de chegada de primeiro os mais curtos e depois os mais demorados parece ser vantajosa em termos de tempo de espera. Isto significa então que o SJF é uma boa estratégia, uma vez que o seu algoritmo é baseado justamente nesse fator? A verdade é que a ordem aleatória - que seria a ordem de tempos mais realista - também nos confirma que o SJF consegue ser mais eficiente, pelo menos neste cenário 1.

No segundo cenário tentamos executar o máximo de tarefas, todas com tempos relativamente curtos. O resultado mostrou-se uma vez mais favorável ao SJF, no entanto sem grandes alterações à ordem de grandeza da difereneça entre as duas políticas, e a nosso ver, ainda não suficiente para tirar conclusões. De notar que embora a difereneça aqui seja menor (200 segundos), foram executadas o dobro das tarefas em paralelo.

A verdade é que o terceiro cenário também não nos traz a certeza que procuravamos. Este cenário, que tenta simular um ambiente com tarefas mais longas, e que já traz uma grande diferença na soma dos tempos de execução, revelou uma das menores diferenças entre as duas políticas. A nosso ver, isto pode ser explicado por uma grande parte do tempo ser passado em execução. Como foram executadas menos tarefas, a diferença de tempo de espera não foi tão significativa.

Apesar de não termos uma conclusão clara até agora, o último cenário parece-nos ser o mais revelador. Aqui, a diferença entre as duas políticas mostrou-se completamente dispersa dos últimos testes, mas acreditamos que por um bom motivo - este teste, que também pode ser visto como o mais realista - tenta juntar todos os testes anteriores, com várias tarefas longas, várias tarefas curtas, e uma soma de tempos de execução muito maior que os anteriores. Estes fatores permitiram ao SJF mostrar que consegue lidar tanto com processos curtos como longos, desde que lhe seja dado um número suficiente de tarefas para executar.

Assim, a nossa conclusão é que o SJF mostra-se sempre mais eficiente que o FCFS. Mesmo que inicialmente a diferença fosse mínima, o último teste deu-nos a certeza da decisão, uma vez que é também o teste mais completo.

3.4. Servidor

Texto