Filas de Prioridades e Escalonador de Processos

Proj4

Informações

- Trabalho em *duplas*;
- Entrega pelo Moodle em <http://trab.dc.unifil.br/moodle/>;
- Peso desse trabalho no bimestre: **20%**.

1 Introdução

Neste trabalho, implementaremos um programa de linha de comando que simulará os diversos algoritmos de escalonamento estudados em sala de aula. O programa recebe parâmetros de linha de comando e um arquivo com perfil de execução de processos, realiza a simulação e exibe informações de mudanças de estado a cada troca de contexto.

O objetivo deste trabalho é simular o funcionamento de escalonamento de processos em um sistema operacional. Para tal, será necessário desenvolver um programa que implemente processos, filas e os algoritmos de escolha dos processos de uma fila. Antes de mais nada, o programa deve definir quais processos existirão, a carga de processamento, o tempo de chegada e a prioridade de cada um deles. Isso será feito lendo as definições em um arquivo texto no formato CSV, que representa planilhas e tabelas.

Considere a seguinte tabela:

pid	carga	chegada	prioridade
1	8	0	10
2	4	0	10
3	2	0	10
4	3	4	10

Sua representação no formato CSV proposto seria um arquivo texto com o seguinte conteúdo:

```
1;8;0;10
2;4;0;10
3;2;0;10
4;3;4;10
```

Com isso em mente e com base em seus estudos e nos livros de apoio da disciplina, construa seu programa simulador de processos seguindo o roteiro de atividades da página seguinte. Bom trabalho!

2 Roteiro de Atividades

Para fazer este trabalho, parta do projeto `EscalonamentoProcessos`, incluso no pacote deste roteiro.

1. Leia as especificações das interfaces `Fila` e `FilaPrioritaria`, e faça o que se pede.
   (a) (25 pontos) Programe a classe `EscalonadorProcessos` que implementa `FilaPrioritaria`, cujas operações são:

   $enfileirar(e)$**:** Insere a entrada e na fila de prioridades, com prioridade maior que a do elemento com maior prioridade atualmente na fila.

   $enfileirar(e,k)$**:** Insere a entrada e na fila de prioridades, com prioridade k.

   $proximo()$**:** Acessa o elemento mínimo da fila de prioridades.

   $desenfileirar()$**:** Acessa e remove o elemento mínimo da fila de prioridades.

   $alterarPrioridade(e,k)$**:** Altera a prioridade k do elemento e, reorganizando a fila para que priorize corretamente o elemento recém modificado.

   (b) (5 pontos) Apresente a análise de complexidade assintótica para todas as operações de `FilaPrioritaria`.

2. (5 pontos) Crie arquivo texto de nome `processos.csv` no seu computador para representar a tabela de processos abaixo no formato CSV proposto:

pid	carga	chegada	prioridade
1001	18	0	30
1102	4	0	50
1003	22	0	50
1014	3	3	30
1061	8	3	50
1002	7	15	50
1203	2	15	70
1304	13	15	30

3. (5 pontos) Crie o programa de nome *simproc*, e implemente nele a funcionalidade de receber o arquivo CSV como argumento de linha de comando, e imprima todos os processos com suas propriedades lidas, como no exemplo abaixo (dica: defina uma classe para representar processos):

```
$ ./java -jar simproc.jar ex3 processos.csv
pid 1001, com carga 18, chegada em 0 e prioridade 30
pid 1102, com carga 4, chegada em 0 e prioridade 50
...
pid 1304, com carga 13, chegada em 15 e prioridade 30
```

```
Execução completada.
```

4. (5 pontos) Implemente, em *simproc*, a funcionalidade de imprimir a ordem da fila de prontos do sistema, baseada nos processos lidos do arquivo CSV. Utilize uma classe para representar a fila de prontos, que contém objetos da classe de processos. Exemplo:

```
$ ./java -jar simproc.jar ex4 processos.csv
1001:1102:1003:1014:1061:1002:1203:1304
Execução completada.
```

5. (5 pontos) Implemente, em *simproc*, o algoritmo *First-Come First-Served*. O programa deverá exibir, para cada momento de escalonamento, o processo que estava em execução, o processo retirado da fila, e a própria fila de execução. Exemplo:

```
$ ./java -jar simproc.jar ex5 processos.csv
Execução: nenhum Escolhido: 1001 Prontos: 1102:1003:1014:1061:
Execução: 1001 Escolhido: 1102 Prontos: 1003:1014:1061:1002:
Execução: 1102 Escolhido: 1003 Prontos: 1014:1061:1002:1203:
...
Execução: 1304 Escolhido: nenhum Prontos: vazia
Execução completada.
```

6. (5 pontos) Após completar a simulação de *First-Come First-Served*, *simproc* deverá imprimir as seguintes métricas de desempenho: tempo total de execução, taxa de utilização de CPU, vazão e tempo de retorno médio. Além disso, *simproc* deverá receber um novo parâmetro de linha de comando, que é o tempo de *kernel* do sistema em trocas de contexto. Exemplo:

```
$ ./java -jar simproc.jar ex5 processos.csv 1
Execução: nenhum Escolhido: 1001 Prontos: 1102:1003:1014:106
...
Execução: 1304 Escolhido: nenhum Prontos: vazia
Total: 84
CPU: 91,7%
Vazão: 0,0952
Retorno médio: ...
Execução completada.
```

7. (10 pontos) Implemente, em *simproc*, o algoritmo *Shortest Job First*, com exibição dos resultados de maneira similar à atividade 6.

8. (10 pontos) Implemente, em *simproc*, o algoritmo *Round-Robin*, com exibição dos resultados de maneira similar à da atividade 6, porém exibindo as seguintes métricas de desempenho: tempo total, utilização de CPU, tempo de resposta. Além disso, *simproc* deverá receber mais um parâmetro de linha de comando, após o overhead, para definir o quantum do sistema.

9. (25 pontos) Implemente, em *simproc*, o algoritmo *Priority Scheduling*, com exibição dos resultados de maneira similar à da atividade 8, porém exibindo as prioridades de cada processo entre parênteses, após o *pid*. Exemplo:

```
$ ./java -jar simproc.jar ex9 processos.csv 1
Execução: nenhum Escolhido: 1001(70) Prontos: 1102(50):1003(
...
Execução completada.
```

Referências

[1] CORMEN, Thomas H. et al. Algoritmos: teoria e prática. Rio de Janeiro: Campus, 2012. 926 p. ISBN 978-85-352-3699-6.

[2] GOODRICH, Michael T.; TAMASSIA, Roberto. Estruturas de dados e algoritmos em Java. 4. ed. Porto Alegre: Bookman, 2007. 600 p. ISBN 9788560031504.

[3] SILBERSCHATZ, Abraham; GALVIN, Peter Baer; GAGNE, Greg. Sistemas operacionais com java. 7. ed. rev. e atual. Rio de Janeiro: Elsevier, 2008. 673 p. ISBN 978-85-352-2406-1.