

UNIVERSIDADE FEDERAL DE PELOTAS



**Trabalho 2 - Escalonamento
Sistemas Operacionais**

Nidal Martins Ali - 16201904

Heliks Mota Mersenburg - 16200769

Pelotas, 15 de dezembro de 2020.

Introdução

Shortest Job First, é uma política de escalonamento que seleciona para ser executado o processo com o menor tempo de execução. SJF é um algoritmo não preemptivo bastante vantajoso por ser simples e também porque minimiza o tempo médio que cada processo leva desde quando ele é criado até o fim de sua execução. No entanto, essa estratégia pode levar a inanição de processos com longos tempos de execução caso processos curtos sejam continuamente adicionados ao escalonador, como mencionado em [1].

Longest Job First, é o inverso do escalonamento SJF, trata primeiro os processos mais longos, tendo assim os problemas similares ao SJF, caso um processo seja grande demais, pode ocorrer de tarefas pequenas demorem muito para serem executadas.

Desenvolvimento:

1. Implementação:

Para o desenvolvimento do trabalho nos reunimos em chamada no aplicativo Discord e discutimos sobre a proposta, assim como, que modo seria feita a implementação. Inicialmente implementamos em chamada, com tela compartilhada e utilizando o github como repositório. Logo, partimos para resolução de bugs e execução de testes.

O algoritmo foi feito da seguinte forma: Inicialmente os dados do arquivo tarefas são inseridos em um vetor que armazena seu tempo e nome do processo, com isto, utilizamos o algoritmo bubble sort para ordenar os dados de forma crescente ou decrescente, para assim utilizar nos escalonadores. As tarefas são divididas para seus devidos processadores, estes processadores são constituídos por filas que enquanto as tarefas chegam é anotado o momento de sua entrada e também contabilizado o momento de entrada de uma nova tarefa, assim como sua saída. Analisamos qual processador irá acabar o primeiro processo, e então adicionamos o processo a ele... Após anotar todos os processos, as filas são destruídas e os dados armazenados em arquivos.

Utilizamos a linguagem C para implementação do escalonador.

2. Execução e compilação:

Para compilar o programa digite no terminal:

```
gcc -o nome_prog.o nome_prog.c
```

Exemplo

```
gcc -o teste.o Escalonador.c
```

Para execução do programa digite:

./nome_programa nome_arquivo qnt_processadores

Exemplo:

```
(base) heliks@heliks-Inspiron-3583:~/Área de Trabalho/Escalonador$ ./teste.o tarefas_2.txt 2
MENOR_PRIMEIRO.TXT CRIADO
MAIOR_PRIMEIRO.TXT CRIADO
PROGRAMA EXECUTADO COM SUCESSO!
```

Estudo de casos:

1. Arquivo de entrada: tarefas_1.txt

```
a1 7
a2 3
a3 5
a4 10
b1 6
b2 1
b3 6
b4 10
c1 15
c2 3
c3 4
c4 10
```

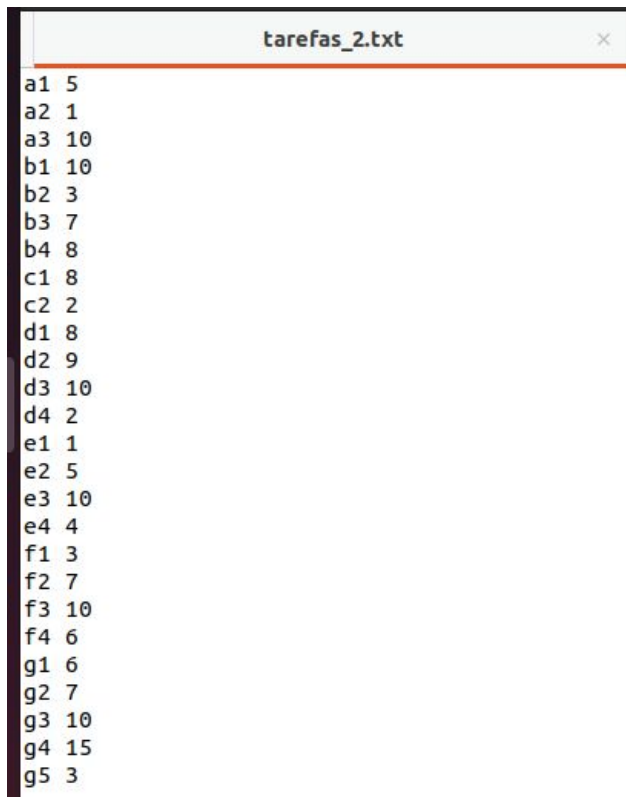
Quantidade de processadores utilizados: 2

Execução ./Escalonador.o tarefas_1.txt 2

Arquivos de saída:

menor_primeiro.txt	maior_primeiro.txt
Processador_1 b2;0;1 c2;1;4 a3;4;9 b3;9;15 a4;15;25 c4;25;35	Processador_1 c1;0;15 c4;15;25 b1;25;31 a3;31;36 a2;36;39 b2;39;40
Processador_2 a2;0;3 c3;3;7 b1;7;13 a1;13;20 b4;20;30 c1;30;45	Processador_2 a4;0;10 b4;10;20 a1;20;27 b3;27;33 c3;33;37 c2;37;40

2. Arquivo de entrada: tarefas_2.txt



The image shows a screenshot of a text editor window titled "tarefas_2.txt". The window contains a list of tasks, each consisting of a label followed by a space and a numerical value. The tasks are listed in the following order: a1 5, a2 1, a3 10, b1 10, b2 3, b3 7, b4 8, c1 8, c2 2, d1 8, d2 9, d3 10, d4 2, e1 1, e2 5, e3 10, e4 4, f1 3, f2 7, f3 10, f4 6, g1 6, g2 7, g3 10, g4 15, and g5 3.

```
a1 5
a2 1
a3 10
b1 10
b2 3
b3 7
b4 8
c1 8
c2 2
d1 8
d2 9
d3 10
d4 2
e1 1
e2 5
e3 10
e4 4
f1 3
f2 7
f3 10
f4 6
g1 6
g2 7
g3 10
g4 15
g5 3
```

Quantidade de processadores utilizados: 4
Execução ./Escalonador.o tarefas_2.txt 4

Arquivos de Saida:

```
menor_primeiro.txt
Processador_1
a2;0;1
b2;1;4
a1;4;9
b3;9;16
c1;16;24
b1;24;34
g3;34;44

Processador_2
e1;0;1
f1;1;4
e2;4;9
f2;9;16
d1;16;24
d3;24;34
g4;34;49

Processador_3
c2;0;2
g5;2;5
f4;5;11
g2;11;18
d2;18;27
e3;27;37

Processador_4
d4;0;2
e4;2;6
g1;6;12
b4;12;20
a3;20;30
f3;30;40
```

```
maior_primeiro.txt
Processador_1
g4;0;15
d2;15;24
b3;24;31
g1;31;37
b2;37;40
c2;40;42
a2;42;43

Processador_2
a3;0;10
e3;10;20
b4;20;28
f2;28;35
e2;35;40
d4;40;42
e1;42;43

Processador_3
b1;0;10
f3;10;20
c1;20;28
g2;28;35
e4;35;39
f1;39;42

Processador_4
d3;0;10
g3;10;20
d1;20;28
f4;28;34
a1;34;39
g5;39;42
```

Dificuldades Encontradas

A implementação dos escalonadores foi de certa forma fácil. Um problema encontrado foi como escalonar em diversos processadores, mas foi algo rapidamente contornado com pesquisas na internet e consulta do material disponibilizado.

Conclusão

Embora os dois algoritmos sejam bastante similares, o SJF aparenta ser mais eficaz. O problema de ambos é que dependendo do tamanho dos processos pode ocorrer de demorar muito ou até mesmo nem executar processos. Por exemplo, se houver muitos processos chegando, pode ser que os processos extremamente grandes não sejam executados. Embora estes algoritmos sejam de fácil implementação e entendimento, é cabível buscar uma solução mais robusta para a desenvoltura de problemas. No mundo real a implementação de algoritmos deste porte podem trazer problemas, cabe ao desenvolvedor buscar uma solução mais eficaz.

Trabalho Futuro

Trabalhos deste tipo são bastante interessantes, ainda mais com uma boa descrição como a que se teve. “Colocar a mão na massa” é muito bom, sair da teoria é ótimo para fixar o entendimento de diversos assuntos. No meu ponto de vista provas não mostram o quanto o aluno entende sobre o assunto, pois muitas questões fazem o uso de “receita de bolo”, onde o aluno precisa decorar em vez de entender. Trabalhos práticos sempre são bem vindos, ainda quando podemos utilizá-los para elaboração do portfólio.

Bibliografia

[1] Arpaci-Dusseau, Remzi H., and Andrea C. Arpaci-Dusseau. Operating systems: Three easy pieces. Arpaci-Dusseau Books, LLC, 2018.