Relatório de Desempenho do Exercício Programa 02

Artur Egidio Launikas e Cupelli 11795738
Gabriela Jie Han 11877423
Maria Eduarda Rodrigues Garcia 11796621
Renan Nakazawa 10723836

Novembro 2021

1 Resumo

A partir de um arquivo base contendo 36.242 linhas de uma palavra cada, foi preenchida uma lista de *strings*, cada elemento contendo uma das linhas deste arquivo de entrada.

Após a população da lista, foram criados 100 objetos de *thread* com os tipos de leitura ou escrita, nas proporções descritas abaixo. Para cada objeto alocado aleatoriamente no arranjo de *threads*, ele será exceutado de acordo com a seu tipo: leitores somente lendo o valor do campo; escritores registrando um novo valor na posição do campo.

A comparação de execução foi feita entre uma implementação da solução do problema de Leitores e Escritores (*Readers-Writer Lock*), e outra implementação que bloqueia qualquer acesso, ambos leitores e escritores.

2 Resultados dos testes

Na Tabela 1 temos os respectivos tempos de execução médios utilizando ambas as implementações. Note que as duas primeiras colunas indicam a proporção entre número de leitores e escritores, sendo que a soma de ambos deve sempre resultar em 100 objetos.

Tabela 1: Tempo médio gasto nas execuções

Número de Leitores	**Número de Escritores**	**Tempo médio com Readers-Writer Lock (ms)**	**Tempo médio sem Readers-Writer Lock (ms)**
0	100	114	113
1	99	112	114

Número de Leitores	**Número de Escritores**	**Tempo médio com Readers-Writer Lock (ms)**	**Tempo médio sem Readers-Writer Lock (ms)**
2	98	110	113
3	97	110	111
4	96	109	111
5	95	107	111
6	94	107	112
7	93	105	111
8	92	104	112
9	91	102	111
10	90	102	111
11	89	101	111
12	88	100	111
13	87	99	111
14	86	98	110
15	85	97	111
16	84	97	111
17	83	96	111
18	82	94	111
19	81	93	110
20	80	92	110
21	79	91	111
22	78	91	110
23	77	89	111
24	76	88	111
25	75	87	111
26	74	87	111
27	73	85	111
28	72	84	115
29	71	83	112
30	70	82	111
31	69	81	110
32	68	80	111
33	67	80	111
34	66	78	111
35	65	77	110
36	64	76	111
37	63	75	111
38	62	75	111
39	61	73	111
40	60	72	111
41	59	71	111
42	58	70	111
43	57	68	111

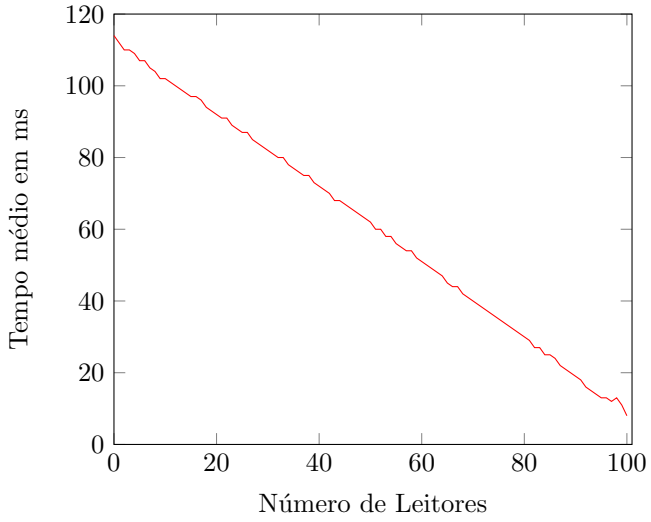
Número de Leitores	**Número de Escritores**	**Tempo médio com Readers-Writer Lock (ms)**	**Tempo médio sem Readers-Writer Lock (ms)**
44	56	68	111
45	55	67	110
46	54	66	110
47	53	65	110
48	52	64	111
49	51	63	111
50	50	62	111
51	49	60	112
52	48	60	111
53	47	58	111
54	46	58	111
55	45	56	111
56	44	55	111
57	43	54	111
58	42	54	111
59	41	52	111
60	40	51	113
61	39	50	111
62	38	49	111
63	37	48	111
64	36	47	111
65	35	45	111
66	34	44	111
67	33	44	111
68	32	42	111
69	31	41	111
70	30	40	111
71	29	39	111
72	28	38	111
73	27	37	111
74	26	36	111
75	25	35	111
76	24	34	111
77	23	33	111
78	22	32	111
79	21	31	111
80	20	30	111
81	19	29	111
82	18	27	111
83	17	27	111
84	16	25	111
85	15	25	111

Número de Leitores	Número de Escritores	Tempo médio com Readers-Writer Lock (ms)	Tempo médio sem Readers-Writer Lock (ms)
86	14	24	111
87	13	22	111
88	12	21	111
89	11	20	111
90	10	19	111
91	9	18	111
92	8	16	111
93	7	15	111
94	6	14	111
95	5	13	111
96	4	13	111
97	3	12	112
98	2	13	112
99	1	11	112
100	0	8	112

2.1 Teste 01 - Tempo médio de execução com Readers-Writer Lock

Na figura abaixo temos a representação gráfica dos resultados da implementação do *Readers-Writer Lock*.

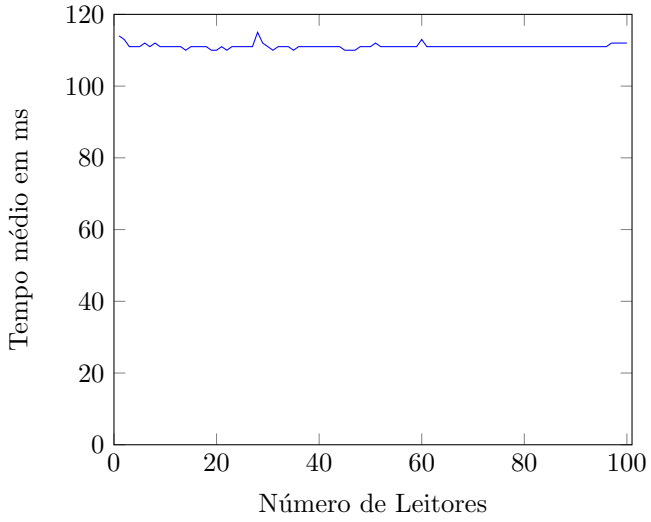
Gráfico 01 - Resultado Teste 01



2.2 Teste 02 - Tempo médio de execução sem Readers-Writer Lock

Na figura abaixo temos a representação gráfica dos resultados da implementação sem o *Readers-Writer Lock*.

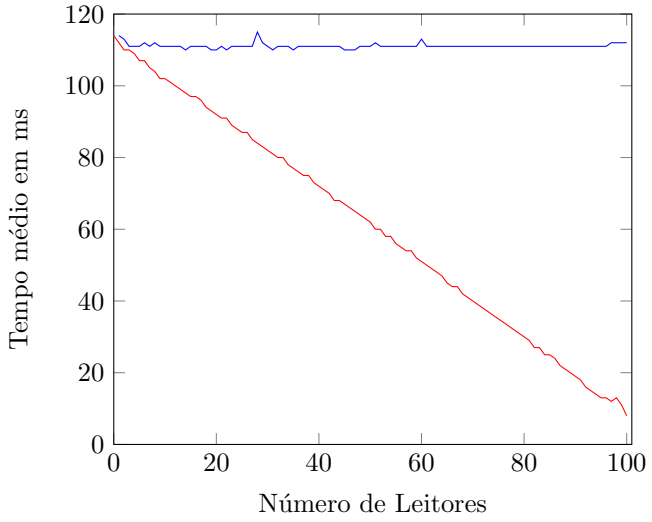
Gráfico 02 - Resultado Teste 02



2.3 Comparação dos testes

Na figura abaixo temos a comparação de ambas as implementações: com *Readers-Writer Lock* (vermelho) e sem *Readers-Writer Lock* (azul).

Gráfico 03 - Comparação entre o Teste 01 e Teste 02



3 Discussão dos Resultados

No Teste 01, podemos perceber que na implementação com o *Readers-Writer Lock*, quanto maior a proporção de leitores para escritores, menor o tempo médio gasto na execução. Podemos explicar esse comportamento já que os leitores tem acesso compartilhado à base, o que não ocorre com os escritores, que bloqueiam a entrada de novos objetos. Com mais leitores do que escritores em execução conseguimos que mais *threads* não estejam bloqueadas e consigam executar simultâneamente, aumentando a vazão de processos.

No Teste 02, notamos praticamente uma linha constante apesar da alteração na proporção de leitores para escritores. Como o bloqueio nesta implementação é feito independentemente do tipo de objeto, não há diferença se há mais escritores ou mais leitores.

A disparidade entre as duas implementações se torna bem explícita quando chegamos na proporção final de 100 leitores para 0 escritores, onde temos 104ms a mais para execução sem o *Readers-Writer Lock*. Quando não há escritores, no Teste 01, todas as *threads* serão executadas de uma vez só, por isso a execução se torna muito rápida. Já no Teste 02, ainda há o bloqueio em todas as entradas na base, gerando o mesmo atraso na execução de quando contém muitos escritores.

É importante notar que ambos os testes possuem resultados muitos semelhantes quando a proporção está com a quantidade maior de escritores. Para a proporção 100 escritores para 0 leitores, temos uma equiparação das implementações. Isso pode ser explicado já que o *Readers-Writer Lock* bloqueia o acesso quando há um escritor na base, sendo que nesta proporção analisada, todos os objetos são escritores, portanto todos irão causar o bloqueio.

No entanto, é importante ressaltar que como a proporção de leitores é muito maior em relação aos escritores e, os leitores sempre possuem prioridade, pode haver o processo de *starvation* dos escritores. Isto é, com a grande quantidade de leitores, os escritores podem acabar não tendo a chance de acessar o recurso bloqueado até que todos os leitores terminem de ler.

Uma possível solução para esse problema seria não permitir mais a entrada de leitores assim que um escritor solicitar o acesso à zona crítica até que o mesmo finalize a sua operação.

4 Conclusão

Podemos concluir com os resultados dos experimentos que a implementação do *Readers-Writer Lock* é mais eficiente comparada a outro modelo bloqueante na concorrência à zona crítica. Entretanto, o processo de *starvation* pode ocorrer na primeira implementação, na situação em que a quantidade de leitores é bem maior que a quantidade de escritores, uma vez que a prioridade está sendo dada aos leitores.

A eficiência de tempo do algoritmo *Readers-Writer Lock*, porém, é maior em casos de sistemas com uma proporção elevada de leitores em relação a escritores. No caso de muito mais escritores do que leitores, este algoritmo não mantém

seu baixo tempo de execução, em relação ao sistema sem *Readers-Writer Lock*.

Portanto, é necessário entender o comportamento dos acessos ao sistema para decisão do melhor algoritmo, verificando se é compensatório para o desempenho priorizar os leitores ou escritores. Além disso, é importante considerar se compensa deixar os escritores aguardando em situações em que há muito mais leitores.