UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Curso de Bacharelado em Ciência de Computação
Curso de Bacharelado em Engenharia de Computação

Sistemas Operacionais

Relatório de Trabalho Prático

Darlei Matheus Schmegel

Dauan Ghisleni Zolinger

Gianlucca de Mendonça Buzo

Heitor Felipe Matozo dos Santos

Letícia Pegoraro Garcez

Implementação com Pthreads

Relatório de Trabalho Prático apresentado aos professores Rafael Burlamaqui Amaral e Gerson Geraldo Homrich Cavalheiro como avaliação parcial da disciplina de Sistemas Operacionais.

<Apresentação[1]>

<Implementação[2]>

Pelotas, Maio de 2021.

[1] Clique para acessar o vídeo da apresentação.
[2] Clique para acessar a implementação do trabalho.

1. Testes com Fibonacci

Realizando testes com diferentes números para cálculo do Fibonacci, foi possível extrair o tempo decorrido em cada execução para posterior análise e comparação de desempenho. Para isso foi usada a biblioteca *time.h*, uma mesma máquina (computador) e foi variada a política de escalonamento assim como o número de processadores virtuais em cada um dos testes. Alguns resultados (com o tempo em segundos), podem ser conferidos abaixo.

Tabela 1.1: Fibonacci (7) = 13				
Escalonamento	Quantidade de processadores virtuais			
	1	2	3	4
FIFO / fila	0,031050	0,030591	0,026565	0,020340
LIFO / pilha	0,032827	0,052404	0,060184	0,085736

Tabela 1.2: Fibonacci (16) = 987				
Escalonamento	Quantidade de processadores virtuais			
	1	2	3	4
FIFO	1,262367	4,976524	8,539019	6,357336
LIFO	2,525376	5,958358	11,527181	18,418340

Tabela 1.3: Fibonacci (23) = 28657				
Escalonamento	Quantidade de processadores virtuais			
	1	2	3	4
FIFO	50,507752	104,911077	157,991396	265,087099
LIFO	223,978957	545,031739	1.012,104071	1.634,706602

Analisando os resultados obtidos nos testes com o Fibonacci (tabelas 1.1, 1.2 e 1.3), percebemos que o critério de escalonamento *First In First Out* (FIFO) obteve resultados mais satisfatórios se comparado a *Last In First Out* (LIFO).

Pensando no fluxo de execução do Fibonacci como uma árvore, notamos que a política FIFO a percorre em níveis (horizontalmente, maior largura), enquanto a LIFO tende priorizar a execução em profundidade. Também, tendo em mente um contexto multithreading, é possível observar que há mais espaço para exploração

de threads no escalonamento de fila (FIFO), onde é possível dividir/paralelizar a execução de mais ramos se comparado ao escalonamento de pilha (LIFO).

Apesar de contraintuitivo, podemos observar também que, conforme o aumento do número de processadores virtuais, em algumas vezes o tempo de execução não apresentou melhora em nenhuma das políticas de escalonamento implementadas.

Entre os possíveis motivos para isso estão o tempo necessário para trocas de contexto (que podem ocorrer na função sync, por exemplo) e principalmente o tempo despendido à espera de mutexes, já que é provável que para uma quantidade maior de processadores virtuais mais tempo seja gasto esperando a liberação de recursos.

Além disso, o custo de criação das threads também é uma das possíveis causas, entretanto, como a contagem do tempo de execução avaliado começa após a criação das threads (para permitir uma comparação mais justa entre os tempos de execução) esta possibilidade acabou sendo desconsiderada.

2. Testes com Merge Sort

Da mesma forma, foram realizados testes com o algoritmo de ordenação implementado Merge Sort e anotado o tempo de cada execução (em segundos) para observação de desempenho. Procurou-se não usar vetores com tamanhos muito pequenos para fazer sentido o uso da ferramenta. Também, testou-se para diferentes estados de ordenação\desordenação. Alguns dos resultados podem ser conferidos abaixo.

Tabela 2.1: Merge Sort [24] com 8 desordens

[**17**,2,3,4,5,**19**,7,8,9,**14**,**21**,12,13,**10**,15,16,**1**,18,**6**,20,**11**,22,23,24]

Escalonamento	Quantidade de processadores virtuais			
	1	2	3	4
FIFO	0,000688	0,014803	0,001875	0,00814
LIFO	0,001659	0,03394	0,0027	0,000609

Tabela 2.2: Merge Sort [24] com 16 desordens				
[**17**,2,**16**,**7**,5,**19**,**4**,8,9,**14**,**21**,12,**23**,**10**,15,**3**,**1**,18,**6**,**24**,**11**,22,**13**,**20**]				
Escalonamento	Quantidade de processadores virtuais			
	1	2	3	4
FIFO	0,009905	0,0385106	0,003513	0,008008
LIFO	0,001371	0,018829	0,015023	0,010507

Tabela 2.3: Merge Sort [24] com 24 desordens				
[**17**,**9**,**16**,**7**,**15**,**19**,**4**,**22**,**2**,**14**,**21**,**18**,**23**,**10**,**5**,**3**,**1**,**12**,**6**,**24**,**11**,**8**,**13**,**20**]				
Escalonamento	Quantidade de processadores virtuais			
	1	2	3	4
FIFO	0,01135	0,013932	0,00072	0,000524
LIFO	0,043445	0,000374	0,000484	0,017007

Tabela 2.4: Merge Sort [1000] com valores gerados aleatoriamente				
Escalonamento	Quantidade de processadores virtuais			
	1	2	3	4
FIFO	0,09471	0,069892	0,162732	0,800689
LIFO	0,244181	0,132651	0,042818	0,020037

Observando os resultados obtidos nos testes do Merge Sort com um vetor de 24 posições (tabelas 2.1, 2.2 e 3.3), por se tratar de um vetor muito pequeno, ocorreram alguns resultados inconsistentes, o que não os tornaram muito esclarecedores.

Porém, ao fazer a ordenação de um vetor com 1000 posições com valores gerados aleatoriamente (tabela 2.4), já se pode visualizar algumas tendências. Nota-se que ao aumentar o número de processadores virtuais, existe uma diminuição no tempo de execução da função. Um dos motivos disso acontecer se deve pelo fato de o Merge Sort explorar melhor o paralelismo do que um algoritmo que é sequencial.