	UNIVERSIDADE ESTADUAL DE MARINGÁ **CENTRO DE TECNOLOGIA** **DEPARTAMENTO DE INFORMÁTICA**
Disciplina 12035 - Sistemas Operacionais (Ano letivo de 2025, 1º Semestre)	
Bacharelado em Ciência da Computação	
Professor Dr. Alisson Renan Svaigen	
Atividade Prática 02: Sincronização	

Objetivo da aula prática: aplicar o conhecimento obtido no Módulo 05 (Sincronização) de maneira prática, por meio da implementação de aplicações que geram condições de corrida , necessitando de protocolos que minimizem o problema da região crítica.

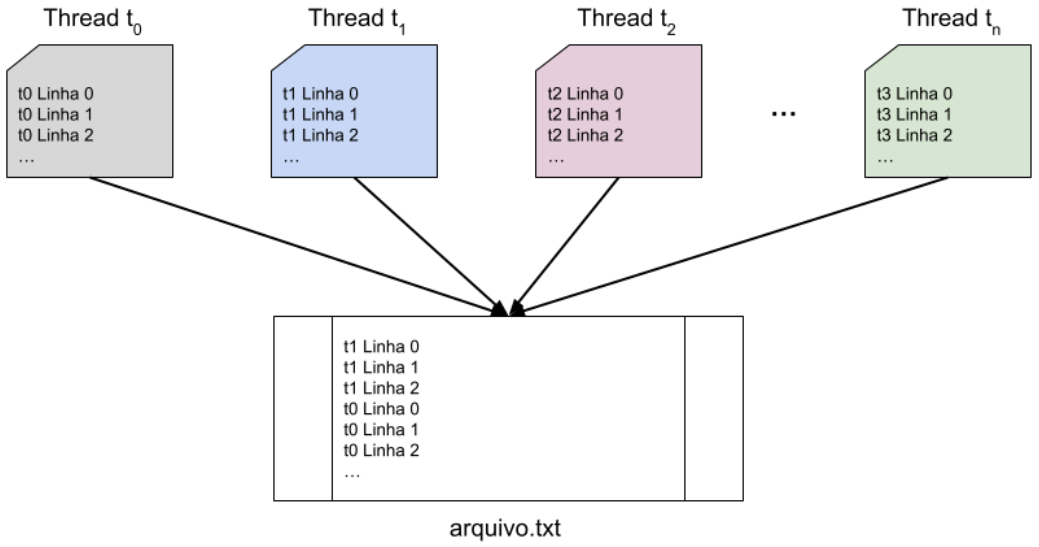
Linguagem de programação a ser utilizada: Python 3.x

SO a ser utilizado: qualquer distribuição com um ambiente Python instalado.

DIFERENTES THREADS ESCREVENDO EM UM ARQUIVO

Em aplicações reais, é comum que múltiplas threads trabalhem de forma cooperativa para processar dados e gerar resultados. Um exemplo clássico ocorre quando diferentes threads precisam escrever simultaneamente em um mesmo arquivo, seja para registrar logs, armazenar resultados parciais ou construir um relatório final. Conforme discutido em sala de aula, essa situação pode gerar **condições de corrida**, gerando arquivos corrompidos, linhas sobrepostas ou perda de dados.

A figura a seguir exibe visualmente o cenário que será considerado nesta atividade prática.



UNIVERSIDADE ESTADUAL DE MARINGÁ **CENTRO DE TECNOLOGIA** **DEPARTAMENTO DE INFORMÁTICA**	
Disciplina 12035 - Sistemas Operacionais (Ano letivo de 2025, 1º Semestre)	
Bacharelado em Ciência da Computação	
Professor Dr. Alisson Renan Svaigen	
Atividade Prática 02: Sincronização	

De modo geral, o cenário consiste na presença de *n* threads que acessam um determinado arquivo (de caminho relativo `arquivo.txt`), utilizando-o para escrita por inclusão (popularmente sendo a opção de abertura “a”).

O cenário trabalhado possui as seguintes características:

- Uma vez que uma thread inicia a sua escrita, ela deve requerer acesso ao arquivo e escrever nele caractere a caractere. Do ponto de vista de uma Linguagem de Programação, é como se a requisição para abertura do arquivo e para “fechamento” do arquivo devessem acontecer antes e depois da escrita de cada caractere, respectivamente;
- Cada thread possui uma quantidade específica de linhas a serem escritas no arquivo.
- É esperado que, embora cada *thread* tenha acesso ao arquivo, ela escreva todo o seu conteúdo de maneira sequencial, de modo a manter a escrita coesa.

Nessa aula prática, vamos desenvolver a implementação de três versões diferentes que atendam a especificação desse cenário:

- Uma versão que não utiliza nenhum protocolo de sincronização;
- Uma versão que implementa um semáforo para sincronização das threads a cada linha de escrita (ou seja, a seção crítica é “liberada” cada vez que a thread finaliza a escrita de uma linha;
- Uma versão que implementa um semáforo para sincronização das threads considerando o momento em que a *thread* demonstrar a intenção de iniciar a sua escrita (a partir da primeira linha), “liberando” a sua seção crítica apenas quando a escrita completa finalizar.

CONFIGURAÇÕES A SEREM TESTADAS

Três diferentes configurações de *n*, tal que *n* é o número de threads:

- $n = 2$

- $n = 4$
- $n = 8$

Para cada configuração, considere sempre que cada thread deve escrever 5 linhas, seguindo o seguinte formato:

```
[thread i] linha j
```

Na qual `i` é o identificador da *thread* e `j` é o número da linha escrita (que varia de 0 a 4).

ENTREGAS NO GOOGLE CLASSROOM

- Três arquivos .py com o código fonte da implementação de cada versão desenvolvida;
- Um arquivo .pdf contendo:
    - O log do arquivo para cada execução, considerando as diferentes configurações e as diferentes versões de implementação:
        - Implementação sem protocolos de sincronização com $n = 2$
        - Implementação sem protocolos de sincronização com $n = 4$
        - Implementação sem protocolos de sincronização com $n = 8$
        - Implementação de semáforo controlando escrita de linhas com $n = 2$
        - Implementação de semáforo controlando escrita de linhas com $n = 4$
        - Implementação de semáforo controlando escrita de linhas com $n = 8$
        - Implementação de semáforo controlando escrita completa com $n = 2$
        - Implementação de semáforo controlando escrita completa com $n = 4$

- Implementação de semáforo controlando escrita completa com *n* = 8

- Uma tabela que calcula a taxa de caracteres escritos de maneira incorreta para cada configuração.

Exemplo de tabela:

Configuração	**Taxa de caracteres escritos de maneira incorreta para n = 2**	**Taxa de caracteres escritos de maneira incorreta para n = 4**	**Taxa de caracteres escritos de maneira incorreta para n = 8**
sem protocolo			
semáforo para linhas			
semáforo para operação completa			

Para realizar o cálculo dessa taxa, considere que os caracteres estariam escritos corretamente se seguissem os seguintes gabaritos (considere que ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’ são índices quaisquer de uma thread realizando a escrita):

- Para n = 2:

```
[thread a] linha 0
[thread a] linha 1
[thread a] linha 2
[thread a] linha 3
[thread a] linha 4
[thread b] linha 0
[thread b] linha 1
[thread b] linha 2
[thread b] linha 3
[thread b] linha 4
```

- Para n = 4:

```
[thread a] linha 0
[thread a] linha 1
```

```
[thread a] linha 2
[thread a] linha 3
[thread a] linha 4
[thread b] linha 0
[thread b] linha 1
[thread b] linha 2
[thread b] linha 3
[thread b] linha 4
[thread c] linha 0
[thread c] linha 1
[thread c] linha 2
[thread c] linha 3
[thread c] linha 4
[thread d] linha 0
[thread d] linha 1
[thread d] linha 2
[thread d] linha 3
[thread d] linha 4
```

- Para n = 8:

```
[thread a] linha 0
[thread a] linha 1
[thread a] linha 2
[thread a] linha 3
[thread a] linha 4
[thread b] linha 0
[thread b] linha 1
[thread b] linha 2
[thread b] linha 3
[thread b] linha 4
[thread c] linha 0
[thread c] linha 1
[thread c] linha 2
[thread c] linha 3
[thread c] linha 4
[thread d] linha 0
[thread d] linha 1
[thread d] linha 2
[thread d] linha 3
[thread d] linha 4
[thread e] linha 0
[thread e] linha 1
[thread e] linha 2
[thread e] linha 3
[thread e] linha 4
```

```
[thread f] linha 0
[thread f] linha 1
[thread f] linha 2
[thread f] linha 3
[thread f] linha 4
[thread g] linha 0
[thread g] linha 1
[thread g] linha 2
[thread g] linha 3
[thread g] linha 4
[thread h] linha 0
[thread h] linha 1
[thread h] linha 2
[thread h] linha 3
[thread h] linha 4
```

- Por fim, responda às seguintes questões:
    - Possivelmente, existiram uma série de caracteres que foram considerados como “escritos corretamente” na implementação da primeira e segunda versão dos algoritmos (sem protocolos de sincronização e com protocolo de sincronização para a escrita de linha). No entanto, é possível garantir com 100% de certeza que esses caracteres realmente correspondem aos caracteres provenientes da thread correta? Justifique sua resposta.
    - O protocolo de semáforos é a escolha mais eficiente para o problema tratado? Justifique sua resposta.
    - Suponha que o cenário apresentado adicionasse a seguinte restrição:
        - “*O identificador das threads indica prioridade de escrita no arquivo. Ou seja: primeiro quem deve escrever os dados é a thread t_0, depois a t_1, depois a t_2, e assim sucessivamente.*
        - Nesse caso, qual protocolo poderia ser utilizado para atender essa restrição? Quais os “efeitos colaterais” que esse protocolo poderia causar?

Data final da entrega: 23h59min do dia 03 de Junho de 2025, exclusivamente via Google Classroom