	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.

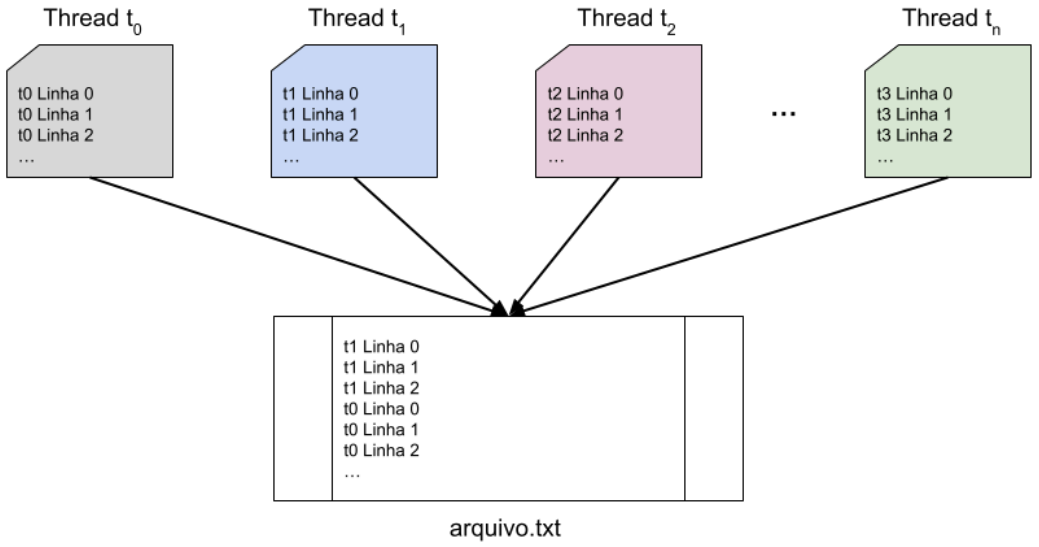
Linguagem de programação a ser utilizada: Python 3.x

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 escrever todas as linhas no arquivo, sem interrupções;
- No entanto, o “identificador” da thread não indica prioridade. Ou seja: a *thread* t_0 não precisa, necessariamente, escrever seus dados antes da thread t_1, e assim sucessivamente. Tanto que, observando a figura, pode-se notar que a *thread* t_1 realizou sua escrita antes da *thread* t_0.

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;
- Um terceira versão que atenda a seguinte condição adicional:
    - 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.

CONFIGURAÇÕES A SEREM TESTADAS

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

- $n = 2$
- $n = 4$
- $n = 8$
- $n = 16$

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	

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 sem protocolos de sincronização com *n* = 16
        - Implementação de semáforo com *n* = 2
        - Implementação de semáforo com *n* = 4
        - Implementação de semáforo com *n* = 8
        - Implementação de semáforo com *n* = 16
        - Implementação “a descobrir” com *n* = 2
        - Implementação “a descobrir” com *n* = 4

- Implementação “a descobrir” com *n* = 8
- Implementação “a descobrir” com *n* = 16

○ Uma tabela considerando cada implementação e configuração, calculando a taxa de linhas que foram escritas em ordem errada para as duas situações a seguir:

- As *threads* não possuem uma ordem específica de escrita (para esse caso, considere que uma linha foi escrita “errada” se ela não está imediatamente após a primeira ocorrência de uma linha da sua *thread*)
- As *threads* precisam seguir a ordem do seu identificador para escrever no arquivo.

Exemplo de tabela:

Configuração	**Thread sem ordem de escrita**	**Thread com ordem de escrita**
sem protocolo, n = 2		
sem protocolo, n = 4		
sem protocolo, n = 8		
sem protocolo, n = 16		
semáforo, n = 2		
semáforo, n = 4		
semáforo, n = 8		
semáforo, n = 16		
“a descobrir”, n = 2		

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	

“a descobrir”, n = 4		
“a descobrir”, n = 8		
“a descobrir”, n = 16		

Por fim, faça um gráfico proveniente da tabela, analisando protocolos x configuração nas duas situações. Realize uma breve discussão sobre os resultados obtidos, considerando principalmente como o número de threads afeta a taxa de linhas escritas em ordem errada.

Data final da entrega: 23h59min do dia 30 de Maio de 2025, exclusivamente via Google Classroom