EXECUTANDO NORMALMENTE COM O CÓDIGO DE 'semaforos.py' DO REPOSITÓRIO:

```
[Produtor 1] Produziu P1-0 | Pos: 0 | Buffer: ['P1-0', None, None, None, None]
[Produtor 2] Produziu P2-0 | Pos: 1 | Buffer: ['P1-0', 'P2-0', None, None, None]
             [Consumidor 1] Consumiu P1-0 | Pos: 0 | Buffer: [None, 'P2-0', None, None, None]
             [Consumidor 2] Consumiu P2-0 | Pos: 1 | Buffer: [None, None, None, None, None]
[Produtor 2] Produziu P2-1 | Pos: 2 | Buffer: [None, None, 'P2-1', None, None]
[Produtor 1] Produziu P1-1 | Pos: 3 | Buffer: [None, None, 'P2-1', 'P1-1', None]
             [Consumidor 1] Consumiu P2-1 | Pos: 2 | Buffer: [None, None, None, 'P1-1', None]
             [Consumidor 2] Consumiu P1-1 | Pos: 3 | Buffer: [None, None, None, None, None]
[Produtor 1] Produziu P1-2 | Pos: 4 | Buffer: [None, None, None, None, 'P1-2']
[Produtor 2] Produziu P2-2 | Pos: 0 | Buffer: ['P2-2', None, None, None, 'P1-2']
             [Consumidor 2] Consumiu P1-2 | Pos: 4 | Buffer: ['P2-2', None, None, None, None]
[Produtor 1] Produziu P1-3 | Pos: 1 | Buffer: ['P2-2', 'P1-3', None, None, None]
[Produtor 2] Produziu P2-3 | Pos: 2 | Buffer: ['P2-2', 'P1-3', 'P2-3', None, None]
             [Consumidor 1] Consumiu P2-2 | Pos: 0 | Buffer: [None, 'P1-3', 'P2-3', None, None]
[Produtor 2] Produziu P2-4 | Pos: 3 | Buffer: [None, 'P1-3', 'P2-3', 'P2-4', None]
[Produtor 1] Produziu P1-4 | Pos: 4 | Buffer: [None, 'P1-3', 'P2-3', 'P2-4', 'P1-4']
             [Consumidor 1] Consumiu P1-3 | Pos: 1 | Buffer: [None, None, 'P2-3', 'P2-4', 'P1-4']
             [Consumidor 2] Consumiu P2-3 | Pos: 2 | Buffer: [None, None, None, 'P2-4', 'P1-4']
             [Consumidor 1] Consumiu P2-4 | Pos: 3 | Buffer: [None, None, None, None, 'P1-4']
             [Consumidor 2] Consumiu P1-4 | Pos: 4 | Buffer: [None, None, None, None, None]
```

PERGUNTAS DOS SLIDES

1. **Por que precisamos do mutex?**

Para essa pergunta, comentei todas as linhas que fazem uso do mutex, ou seja, todos os mutex.acquire(), mutex.release() e o próprio mutex = threading.Semaphore(1). Em seguida, executei o código, resultando em:

```
[Produtor 1] Produziu P1-0 | Pos: 0 | Buffer: ['P1-0', None, None, None, None]
[Produtor 2] Produziu P2-0 | Pos: 1 | Buffer: ['P1-0', 'P2-0', None, None, None]
             [Consumidor 1] Consumiu P1-0 | Pos: 0 | Buffer: [None, 'P2-0', None, None, None]
             [Consumidor 2] Consumiu P2-0 | Pos: 1 | Buffer: [None, None, None, None, None]
[Produtor 2] Produziu P2-1 | Pos: 2 | Buffer: [None, None, 'P2-1', None, None]
[Produtor 1] Produziu P1-1 | Pos: 2 | Buffer: [None, None, 'P1-1', None, None]
             [Consumidor 1] Consumiu P1-1 | Pos: 2 | Buffer: [None, None, None, None, None]
             [Consumidor 2] Consumiu None | Pos: 2 | Buffer: [None, None, None, None, None]
[Produtor 2] Produziu P2-2 | Pos: 4 | Buffer: [None, None, None, None, 'P2-2']
[Produtor 1] Produziu P1-2 | Pos: 0 | Buffer: ['P1-2', None, None, None, 'P2-2']
             [Consumidor 2] Consumiu P2-2 | Pos: 4 | Buffer: ['P1-2', None, None, None, None]
             [Consumidor 1] Consumiu None | Pos: 4 | Buffer: ['P1-2', None, None, None, None]
[Produtor 1] Produziu P1-3 | Pos: 1 | Buffer: ['P1-2', 'P1-3', None, None, None]
[Produtor 2] Produziu P2-3 | Pos: 1 | Buffer: ['P1-2', 'P2-3', None, None, None]
[Produtor 1] Produziu P1-4 | Pos: 3 | Buffer: ['P1-2', 'P2-3', None, 'P1-4', None]
[Produtor 2] Produziu P2-4 | Pos: 3 | Buffer: ['P1-2', 'P2-3', None, 'P2-4', None]
             [Consumidor 2] Consumiu P2-3 | Pos: 1 | Buffer: ['P1-2', None, None, 'P2-4', None]
             [Consumidor 1] Consumiu None | Pos: 1 | Buffer: ['P1-2', None, None, 'P2-4', None]
             [Consumidor 1] Consumiu P2-4 | Pos: 3 | Buffer: ['P1-2', None, None, None, None]
             [Consumidor 2] Consumiu None | Pos: 3 | Buffer: ['P1-2', None, None, None, None]
```

Como é possível perceber comparando-o a execução normal, há inconsistência nos dados como resultado de um acesso simultâneo ao buffer. Ocorre interferência entre as threads (dois produtores conseguem escrever na mesma posição e um consumidor pode consumir de uma posição já vazia) e as posições do buffer são corrompidas. Então, a presença do mutex é essencial para evitar race condition, garantindo que apenas uma thread por vez possa acessar e modificar o buffer (a região crítica). Sem

ele, múltiplas threads acessam simultaneamente, causando sobrescrita de dados, e resultados incorretos ou imprevisíveis.

2. **Onde estão as regiões críticas?**

A região crítica são os trechos que acessam dados compartilhados, no caso, o buffer. Logo, a região crítica estão entre os mutex.acquire e os mutex.release():

```
mutex.acquire()
buffer[in_index] = item
print(
    f"[Produtor {pid}] Produziu {item} | Pos: {in_index} | Buffer: {buffer}")
in_index = (in_index + 1) % BUFFER_SIZE
mutex.release()
```

```
mutex.acquire()
item = buffer[out_index]
buffer[out_index] = None
print(
    f"              [Consumidor {cid}] Consumiu {item} | Pos: {out_index} | Buffer: {buffer}")
out_index = (out_index + 1) % BUFFER_SIZE
mutex.release()
```

3. **Podemos ter mais de N threads bloqueadas em wait(empty)?**

Para essa pergunta, aumentei a quantidade de 2 produtores para 10 e adicionei a linha `print(f"[Produtor {pid}] Tentando entrar em empty")` antes de empty.acquire() do produtor, assim como a linha `print(f"[Produtor {pid}] Passou pelo empty")` depois do mesmo empty.acquire(). O resultado foi esse:

```
[Produtor 1] Tentando entrar em empty
[Produtor 1] Passou pelo empty
[Produtor 2] Tentando entrar em empty
[Produtor 2] Passou pelo empty
[Produtor 1] Produziu P1-0 | Pos: 0 | Buffer: ['P1-0', None, None, None, None]
[Produtor 3] Tentando entrar em empty
[Produtor 2] Produziu P2-0 | Pos: 1 | Buffer: ['P1-0', 'P2-0', None, None, None]
[Produtor 5] Tentando entrar em empty
[Produtor 4] Tentando entrar em empty
[Produtor 3] Passou pelo empty
[Produtor 5] Passou pelo empty
[Produtor 4] Passou pelo empty
[Produtor 8] Tentando entrar em empty
[Produtor 6] Tentando entrar em empty
[Produtor 7] Tentando entrar em empty
[Produtor 3] Produziu P3-0 | Pos: 2 | Buffer: ['P1-0', 'P2-0', 'P3-0', None, None]
[Produtor 10] Tentando entrar em empty
[Produtor 9] Tentando entrar em empty
[Produtor 5] Produziu P5-0 | Pos: 3 | Buffer: ['P1-0', 'P2-0', 'P3-0', 'P5-0', None]
[Produtor 4] Produziu P4-0 | Pos: 4 | Buffer: ['P1-0', 'P2-0', 'P3-0', 'P5-0', 'P4-0']
              [Consumidor 1] Consumiu P1-0 | Pos: 0 | Buffer: [None, 'P2-0', 'P3-0', 'P5-0', 'P4-0']
              [Consumidor 2] Consumiu P2-0 | Pos: 1 | Buffer: [None, None, 'P3-0', 'P5-0', 'P4-0']
[Produtor 8] Passou pelo empty
[Produtor 6] Passou pelo empty
[Produtor 8] Produziu P8-0 | Pos: 0 | Buffer: ['P8-0', None, 'P3-0', 'P5-0', 'P4-0']
[Produtor 6] Produziu P6-0 | Pos: 1 | Buffer: ['P8-0', 'P6-0', 'P3-0', 'P5-0', 'P4-0']
[Produtor 1] Tentando entrar em empty
[Produtor 5] Tentando entrar em empty
[Produtor 4] Tentando entrar em empty
[Produtor 8] Tentando entrar em empty
[Produtor 6] Tentando entrar em empty
              [Consumidor 1] Consumiu P3-0 | Pos: 2 | Buffer: ['P8-0', 'P6-0', None, 'P5-0', 'P4-0']
[Produtor 7] Passou pelo empty
[Produtor 7] Produziu P7-0 | Pos: 2 | Buffer: ['P8-0', 'P6-0', 'P7-0', 'P5-0', 'P4-0']
              [Consumidor 2] Consumiu P5-0 | Pos: 3 | Buffer: ['P8-0', 'P6-0', 'P7-0', None, 'P4-0']
[Produtor 10] Passou pelo empty
[Produtor 10] Produziu P10-0 | Pos: 3 | Buffer: ['P8-0', 'P6-0', 'P7-0', 'P10-0', 'P4-0']
[Produtor 7] Tentando entrar em empty
[Produtor 10] Tentando entrar em empty
              [Consumidor 1] Consumiu P4-0 | Pos: 4 | Buffer: ['P8-0', 'P6-0', 'P7-0', 'P10-0', None]
[Produtor 9] Passou pelo empty
[Produtor 9] Produziu P9-0 | Pos: 4 | Buffer: ['P8-0', 'P6-0', 'P7-0', 'P10-0', 'P9-0']
              [Consumidor 2] Consumiu P8-0 | Pos: 0 | Buffer: [None, 'P6-0', 'P7-0', 'P10-0', 'P9-0']
[Produtor 1] Passou pelo empty
[Produtor 1] Produziu P1-1 | Pos: 0 | Buffer: ['P1-1', 'P6-0', 'P7-0', 'P10-0', 'P9-0']
[Produtor 9] Tentando entrar em empty
[Produtor 1] Tentando entrar em empty
              [Consumidor 1] Consumiu P6-0 | Pos: 1 | Buffer: ['P1-1', None, 'P7-0', 'P10-0', 'P9-0']
[Produtor 2] Passou pelo empty
[Produtor 2] Produziu P2-1 | Pos: 1 | Buffer: ['P1-1', 'P2-1', 'P7-0', 'P10-0', 'P9-0']
              [Consumidor 2] Consumiu P7-0 | Pos: 2 | Buffer: ['P1-1', 'P2-1', None, 'P10-0', 'P9-0']
[Produtor 3] Passou pelo empty
[Produtor 3] Produziu P3-1 | Pos: 2 | Buffer: ['P1-1', 'P2-1', 'P3-1', 'P10-0', 'P9-0']
[Produtor 2] Tentando entrar em empty
```

Ou seja, sim, podemos ter mais que N threads bloqueadas em wait(empty).

4. **Quanto vale empty + full (valor dos contadores)?**

Para essa pergunta, adicionei a linha `print(f"[DEBUG] Semáforos -> empty={empty._value}, full={full._value}, soma={empty._value + full._value}")` após o full.release() do produtor e, da mesma forma, adicionei a linha `print(f"[DEBUG] Semáforos -> empty={empty._value}, full={full._value}, soma={empty._value + full._value}")` após o empty.release() do consumidor. Segue o resultado:

```
[Produtor 1] Produziu P1-0 | Pos: 0 | Buffer: ['P1-0', None, None, None, None]
[DEBUG] Semáforos -> empty=4, full=1, soma=5
[Produtor 2] Produziu P2-0 | Pos: 1 | Buffer: ['P1-0', 'P2-0', None, None, None]
[DEBUG] Semáforos -> empty=3, full=2, soma=5
              [Consumidor 1] Consumiu P1-0 | Pos: 0 | Buffer: [None, 'P2-0', None, None, None]
[DEBUG] Semáforos -> empty=4, full=0, soma=4
              [Consumidor 2] Consumiu P2-0 | Pos: 1 | Buffer: [None, None, None, None, None]
[DEBUG] Semáforos -> empty=5, full=0, soma=5
[Produtor 2] Produziu P2-1 | Pos: 2 | Buffer: [None, None, 'P2-1', None, None]
[DEBUG] Semáforos -> empty=3, full=1, soma=4
[Produtor 1] Produziu P1-1 | Pos: 3 | Buffer: [None, None, 'P2-1', 'P1-1', None]
[DEBUG] Semáforos -> empty=3, full=2, soma=5
              [Consumidor 1] Consumiu P2-1 | Pos: 2 | Buffer: [None, None, None, 'P1-1', None]
[DEBUG] Semáforos -> empty=4, full=0, soma=4
              [Consumidor 2] Consumiu P1-1 | Pos: 3 | Buffer: [None, None, None, None, None]
[DEBUG] Semáforos -> empty=5, full=0, soma=5
[Produtor 2] Produziu P2-2 | Pos: 4 | Buffer: [None, None, None, None, 'P2-2']
[DEBUG] Semáforos -> empty=3, full=1, soma=4
[Produtor 1] Produziu P1-2 | Pos: 0 | Buffer: ['P1-2', None, None, None, 'P2-2']
[DEBUG] Semáforos -> empty=3, full=2, soma=5
              [Consumidor 1] Consumiu P2-2 | Pos: 4 | Buffer: ['P1-2', None, None, None, None]
[DEBUG] Semáforos -> empty=4, full=1, soma=5
              [Consumidor 2] Consumiu P1-2 | Pos: 0 | Buffer: [None, None, None, None, None]
[DEBUG] Semáforos -> empty=5, full=0, soma=5
[Produtor 2] Produziu P2-3 | Pos: 1 | Buffer: [None, 'P2-3', None, None, None]
[DEBUG] Semáforos -> empty=3, full=1, soma=4
[Produtor 1] Produziu P1-3 | Pos: 2 | Buffer: [None, 'P2-3', 'P1-3', None, None]
[DEBUG] Semáforos -> empty=3, full=2, soma=5
[Produtor 2] Produziu P2-4 | Pos: 3 | Buffer: [None, 'P2-3', 'P1-3', 'P2-4', None]
[DEBUG] Semáforos -> empty=1, full=3, soma=4
[Produtor 1] Produziu P1-4 | Pos: 4 | Buffer: [None, 'P2-3', 'P1-3', 'P2-4', 'P1-4']
[DEBUG] Semáforos -> empty=1, full=4, soma=5
              [Consumidor 2] Consumiu P2-3 | Pos: 1 | Buffer: [None, None, 'P1-3', 'P2-4', 'P1-4']
[DEBUG] Semáforos -> empty=2, full=2, soma=4
              [Consumidor 1] Consumiu P1-3 | Pos: 2 | Buffer: [None, None, None, 'P2-4', 'P1-4']
[DEBUG] Semáforos -> empty=3, full=2, soma=5
              [Consumidor 2] Consumiu P2-4 | Pos: 3 | Buffer: [None, None, None, None, 'P1-4']
[DEBUG] Semáforos -> empty=4, full=1, soma=5
              [Consumidor 1] Consumiu P1-4 | Pos: 4 | Buffer: [None, None, None, None, None]
[DEBUG] Semáforos -> empty=5, full=0, soma=5
```

Esse resultado prova que a soma de empty e full é sempre constante e igual ao tamanho do buffer, que no meu caso é BUFFER_SIZE = 5. Afinal, ao produzir o empty ↓ e o full ↑. E, ao consumir, o empty ↑ e o full ↓.

5. **Podemos trocar a ordem das chamadas de wait dos semáforos mutex e empty/full ?**

Para responder essa pergunta, alterei o que antes era uma linha de empty.acquire() seguida por mutex.acquire() no produtor pelo seguinte:

```
print(f"[DEBUG] Produtor {pid} tentando pegar mutex")
mutex.acquire()
print(f"[DEBUG] Produtor {pid} pegou mutex")
print(f"[DEBUG] Produtor {pid} tentando pegar empty")
empty.acquire()
```

E o resultado foi:

```
[DEBUG] Produtor 1 pegou mutex
[DEBUG] Produtor 1 tentando pegar empty
[DEBUG] Produtor 2 tentando pegar mutex
[Produtor 1] Produziu P1-0 | Pos: 0 | Buffer: ['P1-0', None, None, None, None]
[DEBUG] Produtor 2 pegou mutex
[DEBUG] Produtor 2 tentando pegar empty
[Produtor 2] Produziu P2-0 | Pos: 1 | Buffer: ['P1-0', 'P2-0', None, None, None]
              [Consumidor 1] Consumiu P1-0 | Pos: 0 | Buffer: [None, 'P2-0', None, None, None]
              [Consumidor 2] Consumiu P2-0 | Pos: 1 | Buffer: [None, None, None, None, None]
[DEBUG] Produtor 1 tentando pegar mutex
[DEBUG] Produtor 1 pegou mutex
[DEBUG] Produtor 1 tentando pegar empty
[Produtor 1] Produziu P1-1 | Pos: 2 | Buffer: [None, None, 'P1-1', None, None]
[DEBUG] Produtor 2 tentando pegar mutex
[DEBUG] Produtor 2 pegou mutex
[DEBUG] Produtor 2 tentando pegar empty
[Produtor 2] Produziu P2-1 | Pos: 3 | Buffer: [None, None, 'P1-1', 'P2-1', None]
              [Consumidor 1] Consumiu P1-1 | Pos: 2 | Buffer: [None, None, None, 'P2-1', None]
              [Consumidor 2] Consumiu P2-1 | Pos: 3 | Buffer: [None, None, None, None, None]
[DEBUG] Produtor 1 tentando pegar mutex
[DEBUG] Produtor 1 pegou mutex
[DEBUG] Produtor 1 tentando pegar empty
[Produtor 1] Produziu P1-2 | Pos: 4 | Buffer: [None, None, None, None, 'P1-2']
[DEBUG] Produtor 2 tentando pegar mutex
[DEBUG] Produtor 2 pegou mutex
[DEBUG] Produtor 2 tentando pegar empty
[Produtor 2] Produziu P2-2 | Pos: 0 | Buffer: ['P2-2', None, None, None, 'P1-2']
              [Consumidor 1] Consumiu P1-2 | Pos: 4 | Buffer: ['P2-2', None, None, None, None]
[DEBUG] Produtor 2 tentando pegar mutex
[DEBUG] Produtor 1 tentando pegar mutex
              [Consumidor 2] Consumiu P2-2 | Pos: 0 | Buffer: [None, None, None, None, None]
[DEBUG] Produtor 2 pegou mutex
[DEBUG] Produtor 2 tentando pegar empty
[Produtor 2] Produziu P2-3 | Pos: 1 | Buffer: [None, 'P2-3', None, None, None]
[DEBUG] Produtor 1 pegou mutex
[DEBUG] Produtor 1 tentando pegar empty
[Produtor 1] Produziu P1-3 | Pos: 2 | Buffer: [None, 'P2-3', 'P1-3', None, None]
[DEBUG] Produtor 2 tentando pegar mutex
[DEBUG] Produtor 2 pegou mutex
[DEBUG] Produtor 2 tentando pegar empty
[Produtor 2] Produziu P2-4 | Pos: 3 | Buffer: [None, 'P2-3', 'P1-3', 'P2-4', None]
[DEBUG] Produtor 1 tentando pegar mutex
[DEBUG] Produtor 1 pegou mutex
[DEBUG] Produtor 1 tentando pegar empty
[Produtor 1] Produziu P1-4 | Pos: 4 | Buffer: [None, 'P2-3', 'P1-3', 'P2-4', 'P1-4']
              [Consumidor 1] Consumiu P2-3 | Pos: 1 | Buffer: [None, None, 'P1-3', 'P2-4', 'P1-4']
              [Consumidor 2] Consumiu P1-3 | Pos: 2 | Buffer: [None, None, None, 'P2-4', 'P1-4']
              [Consumidor 1] Consumiu P2-4 | Pos: 3 | Buffer: [None, None, None, None, 'P1-4']
              [Consumidor 2] Consumiu P1-4 | Pos: 4 | Buffer: [None, None, None, None, None]
```

Como é possível observar, trocar a ordem das chamadas de wait() dos semáforos mutex e empty não causou problemas aparentes no meu experimento, mas não é recomendado. A ordem padrão (primeiro empty, depois mutex) é importante para evitar deadlocks e manter a região crítica mínima.