

## Atividade - Resolução de Race Condition

Integrantes:

Júlio Cesar Braga Parro - 16879560

Alberto Galhego Neto - 17019141

### Revisão do código anterior

O código original utiliza um contador global sendo incrementado por duas Threads (A e B) independentes, de mesma prioridade e tempo de espera, usando, também, uma função auxiliar que simula o tempo de execução.

Tal código resulta em uma contagem duplicada no contador, sendo: A: 1, B: 1, A: 2, B: 2, A: 3, B: 3..., no caso base gerado pela IA. O erro ocorre em toda a execução do código, desde o começo.

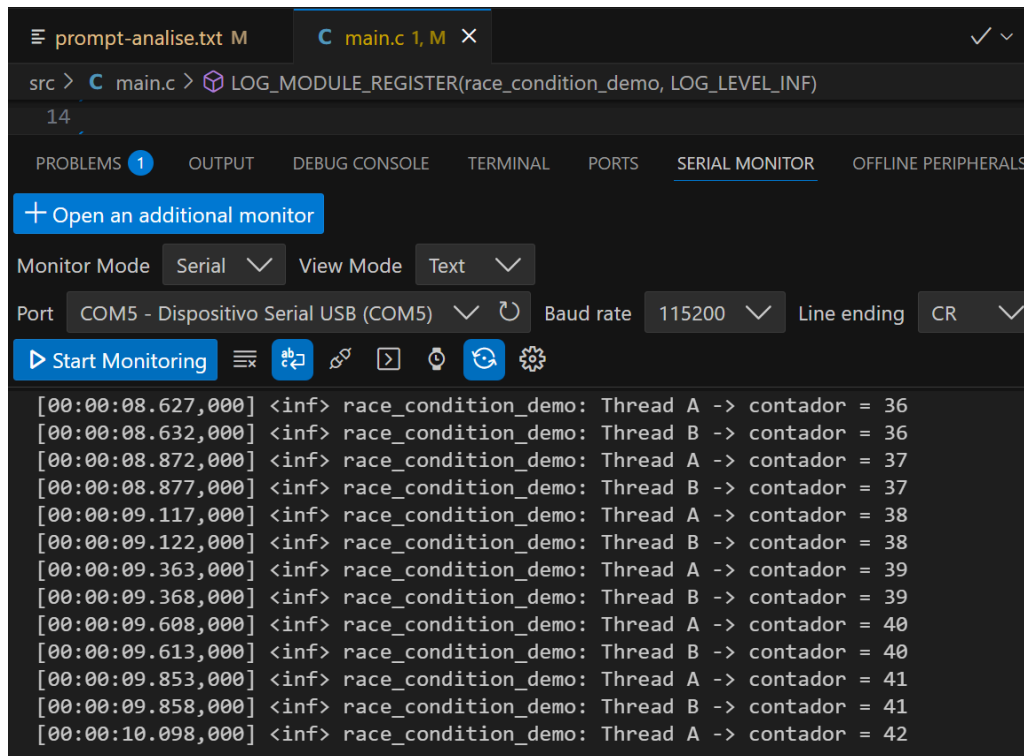
### Planejamento de testes

Os casos de teste criados e executados são:

Caso de Teste	Pré-condição	Etapas de Teste	Pós-condição Observada	Conclusão
<b>1 – Mesmas prioridades e tempos iguais (Caso base)</b>	THREAD_A_PRIO = 5, THREAD_B_PRIO = 5 k_msleep(200) em ambas.	Compilar e executar o código original. Observar logs no terminal.	Contador cresce em sequência <b>duplicada</b> (1,1,2,2,3,3...). Nenhum salto, mas valores se repetem.	Threads acessam o contador simultaneamente. O comportamento é previsível, porém incorreto.
<b>2 – Diferença de temporização</b>	THREAD_A_PRIO = 5, THREAD_B_PRIO = 5 k_msleep(A)=130 ms, k_msleep(B)=100 ms.	Executar o código e observar logs.	Contador <b>se dessincroniza</b> : valores se repetem e/ou saltam (4,4,5,6,6,7...). LEDs piscam em ritmos diferentes.	O problema se manifesta claramente — cada thread sobrescreve o valor da outra.
<b>3 – Alteração de prioridade (tentativa de mitigação)</b>	THREAD_A_PRIO = 4, THREAD_B_PRIO = 5 (A tem prioridade mais alta).	Executar e observar o comportamento .	Contador aparece correto (1,2,3,4...), sem repetições.	A race condition foi <b>mascarada</b> : o escalonador impede a preempção, mas o problema lógico ainda existe.

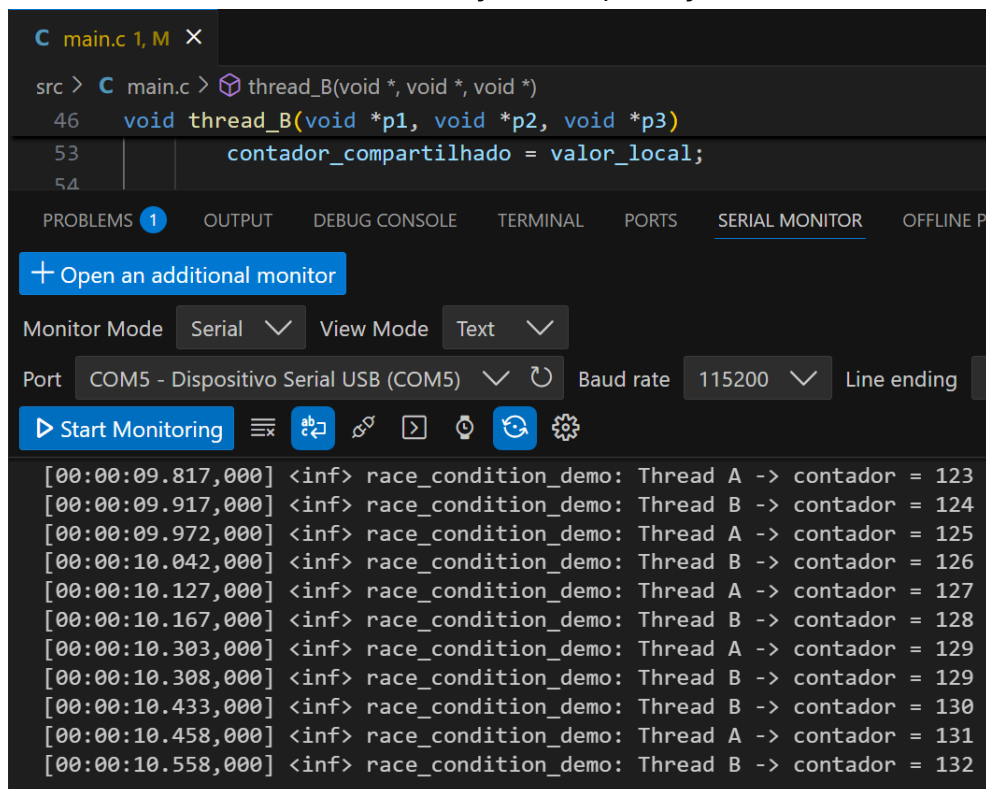
Os testes realizados nas diferentes condições estão registrados abaixo em formato de printscreen dos Logs obtidos em cada caso:

### Caso 1: Mesmas prioridades e tempos iguais



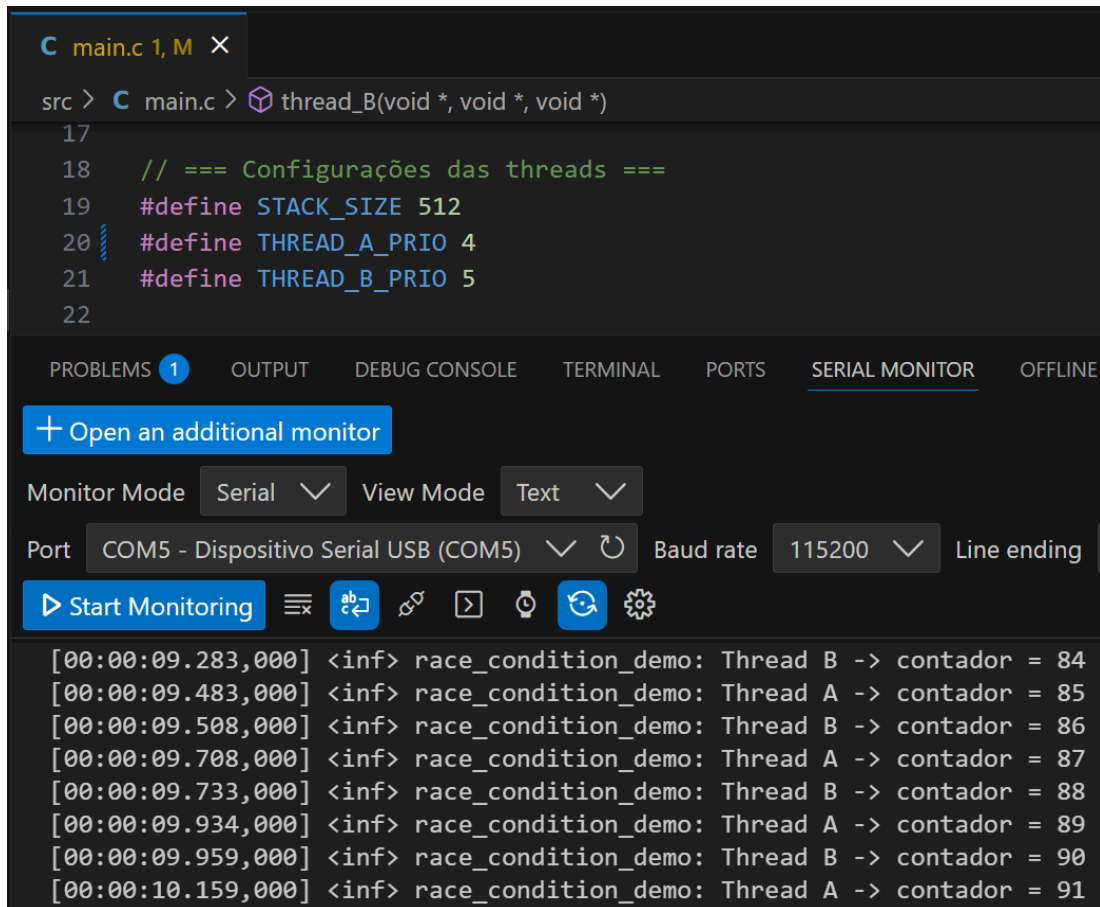
```
src > C main.c > LOG_MODULE_REGISTER(race_condition_demo, LOG_LEVEL_INF)
14
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS SERIAL MONITOR OFFLINE PERIPHERALS
+ Open an additional monitor
Monitor Mode Serial View Mode Text
Port COM5 - Dispositivo Serial USB (COM5) Baud rate 115200 Line ending CR
Start Monitoring
[00:00:08.627,000] <inf> race_condition_demo: Thread A -> contador = 36
[00:00:08.632,000] <inf> race_condition_demo: Thread B -> contador = 36
[00:00:08.872,000] <inf> race_condition_demo: Thread A -> contador = 37
[00:00:08.877,000] <inf> race_condition_demo: Thread B -> contador = 37
[00:00:09.117,000] <inf> race_condition_demo: Thread A -> contador = 38
[00:00:09.122,000] <inf> race_condition_demo: Thread B -> contador = 38
[00:00:09.363,000] <inf> race_condition_demo: Thread A -> contador = 39
[00:00:09.368,000] <inf> race_condition_demo: Thread B -> contador = 39
[00:00:09.608,000] <inf> race_condition_demo: Thread A -> contador = 40
[00:00:09.613,000] <inf> race_condition_demo: Thread B -> contador = 40
[00:00:09.853,000] <inf> race_condition_demo: Thread A -> contador = 41
[00:00:09.858,000] <inf> race_condition_demo: Thread B -> contador = 41
[00:00:10.098,000] <inf> race_condition_demo: Thread A -> contador = 42
```

### Caso 2: Diferença de temporização



```
C main.c 1, M X
src > C main.c > thread_B(void *, void *, void *)
46 void thread_B(void *p1, void *p2, void *p3)
53     contador_compartilhado = valor_local;
54
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS SERIAL MONITOR OFFLINE PERIPHERALS
+ Open an additional monitor
Monitor Mode Serial View Mode Text
Port COM5 - Dispositivo Serial USB (COM5) Baud rate 115200 Line ending CR
Start Monitoring
[00:00:09.817,000] <inf> race_condition_demo: Thread A -> contador = 123
[00:00:09.917,000] <inf> race_condition_demo: Thread B -> contador = 124
[00:00:09.972,000] <inf> race_condition_demo: Thread A -> contador = 125
[00:00:10.042,000] <inf> race_condition_demo: Thread B -> contador = 126
[00:00:10.127,000] <inf> race_condition_demo: Thread A -> contador = 127
[00:00:10.167,000] <inf> race_condition_demo: Thread B -> contador = 128
[00:00:10.303,000] <inf> race_condition_demo: Thread A -> contador = 129
[00:00:10.308,000] <inf> race_condition_demo: Thread B -> contador = 129
[00:00:10.433,000] <inf> race_condition_demo: Thread B -> contador = 130
[00:00:10.458,000] <inf> race_condition_demo: Thread A -> contador = 131
[00:00:10.558,000] <inf> race_condition_demo: Thread B -> contador = 132
```

### Caso 3: Prioridades diferentes



The screenshot shows an IDE with a C program named `main.c` and its serial monitor output. The code defines two threads, `Thread A` and `Thread B`, with different priorities. The serial monitor shows the output of the program, which is a sequence of log messages indicating the execution of the threads and the value of a shared counter.

```
src > C main.c > thread_B(void *, void *, void *)
17
18 // === Configurações das threads ===
19 #define STACK_SIZE 512
20 #define THREAD_A_PRIO 4
21 #define THREAD_B_PRIO 5
22
```

Serial Monitor Output:

```
[00:00:09.283,000] <inf> race_condition_demo: Thread B -> contador = 84
[00:00:09.483,000] <inf> race_condition_demo: Thread A -> contador = 85
[00:00:09.508,000] <inf> race_condition_demo: Thread B -> contador = 86
[00:00:09.708,000] <inf> race_condition_demo: Thread A -> contador = 87
[00:00:09.733,000] <inf> race_condition_demo: Thread B -> contador = 88
[00:00:09.934,000] <inf> race_condition_demo: Thread A -> contador = 89
[00:00:09.959,000] <inf> race_condition_demo: Thread B -> contador = 90
[00:00:10.159,000] <inf> race_condition_demo: Thread A -> contador = 91
```

### Correção e Reteste

Em seguida, foram feitas as devidas correções no código, adicionando um mutex para controlar o acesso ao recurso compartilhado (contador global), de modo a evitar a Race condition.

Dessa forma, o contador agora é incrementado corretamente, apresentando uma sequência como: A: 1, B: 2, A: 3, B: 4, A: 5, B: 6...

Ou seja, cada Thread incrementa o contador em 1 alternadamente, sem haver competição pelo recurso e sem causar comportamento indesejado ou imprevisível.

A tabela com os retestes realizados e seus resultados e conclusões está registrada abaixo:

<b>Caso de Teste</b>	<b>Mudança Aplicada</b>	<b>Etapas de Reteste</b>	<b>Pós-condição Esperada (resultado após correção)</b>
<b>1 – Mesmo cenário original (200 ms / 200 ms)</b>	Inserção de k_mutex_lock() e k_mutex_unlock() envolvendo a manipulação de contador_compartilhado.	Executar novamente.	Contador cresce corretamente (1,2,3,4...). Nenhuma repetição, mesmo com mesma prioridade.
<b>2 – Temporização diferente (130 ms / 100 ms)</b>	Mesma proteção por mutex.	Executar novamente com tempos diferentes.	Contador consistente (1,2,3,4...). LEDs piscam fora de fase, mas contagem é linear.
<b>3 – Alteração de prioridade (4 / 5)</b>	Teste de robustez com prioridades diferentes.	Executar e observar.	Contagem correta e previsível em qualquer ordem de execução.

Os retestes realizados com o código atualizado estão registrados abaixo, por meio de printscreen dos Logs apresentados no Serial Monitor:

## Caso 1: Mesma prioridade e tempos iguais

```
src > C main.c > thread_B(void *, void *, void *)

15 // === Variável compartilhada e mutex ===
16 volatile int contador_compartilhado = 0;
17 K_MUTEX_DEFINE(contador_mutex); // protege o acesso ao contador
18
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS SERIAL MONITOR OFFLINE PERIPHERALS

+ Open an additional monitor

Monitor Mode Serial View Mode Text

Port COM5 - Dispositivo Serial USB (COM5) Baud rate 115200 Line ending CR

Start Monitoring

```
[00:00:09.258,000] <inf> race_condition_demo_fixed: Thread A -> contador = 83
[00:00:09.283,000] <inf> race_condition_demo_fixed: Thread B -> contador = 84
[00:00:09.483,000] <inf> race_condition_demo_fixed: Thread A -> contador = 85
[00:00:09.508,000] <inf> race_condition_demo_fixed: Thread B -> contador = 86
[00:00:09.708,000] <inf> race_condition_demo_fixed: Thread A -> contador = 87
[00:00:09.734,000] <inf> race_condition_demo_fixed: Thread B -> contador = 88
[00:00:09.934,000] <inf> race_condition_demo_fixed: Thread A -> contador = 89
[00:00:09.959,000] <inf> race_condition_demo_fixed: Thread B -> contador = 90
[00:00:10.159,000] <inf> race_condition_demo_fixed: Thread A -> contador = 91
[00:00:10.184,000] <inf> race_condition_demo_fixed: Thread B -> contador = 92
```

## Caso 2: Diferença de temporização

```
C main.c 1, M x
src > C main.c > ...

14
15 // === Variável compartilhada e mutex ===
16 volatile int contador_compartilhado = 0;
17 K_MUTEX_DEFINE(contador_mutex); // protege o acesso ao contador
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS SERIAL MONITOR OFFLINE PERIPHERALS

+ Open an additional monitor

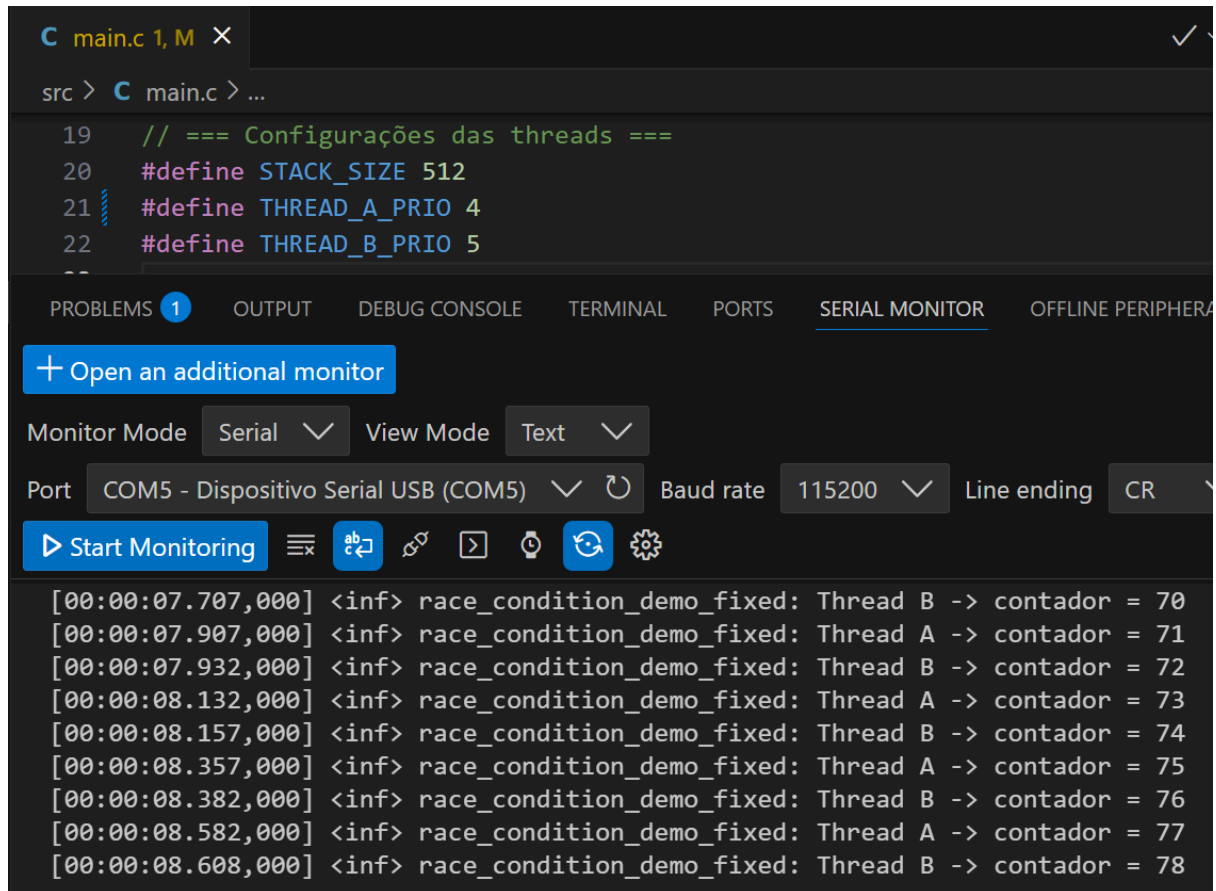
Monitor Mode Serial View Mode Text

Port COM5 - Dispositivo Serial USB (COM5) Baud rate 115200 Line ending CR

Start Monitoring

```
[00:00:09.402,000] <inf> race_condition_demo_fixed: Thread B -> contador = 133
[00:00:09.488,000] <inf> race_condition_demo_fixed: Thread A -> contador = 134
[00:00:09.528,000] <inf> race_condition_demo_fixed: Thread B -> contador = 135
[00:00:09.643,000] <inf> race_condition_demo_fixed: Thread A -> contador = 136
[00:00:09.668,000] <inf> race_condition_demo_fixed: Thread B -> contador = 137
[00:00:09.793,000] <inf> race_condition_demo_fixed: Thread B -> contador = 138
[00:00:09.818,000] <inf> race_condition_demo_fixed: Thread A -> contador = 139
[00:00:09.918,000] <inf> race_condition_demo_fixed: Thread B -> contador = 140
[00:00:09.974,000] <inf> race_condition_demo_fixed: Thread A -> contador = 141
[00:00:10.044,000] <inf> race_condition_demo_fixed: Thread B -> contador = 142
```

### Caso 3: Diferentes prioridades



The screenshot shows an IDE with a C program in `main.c` and its serial monitor output. The code defines two threads, A and B, with different priorities. The serial monitor shows the execution of these threads, with Thread B running first and then Thread A, resulting in a final counter value of 78.

```
19 // === Configurações das threads ===
20 #define STACK_SIZE 512
21 #define THREAD_A_PRIO 4
22 #define THREAD_B_PRIO 5
--
```

Serial Monitor Output:

```
[00:00:07.707,000] <inf> race_condition_demo_fixed: Thread B -> contador = 70
[00:00:07.907,000] <inf> race_condition_demo_fixed: Thread A -> contador = 71
[00:00:07.932,000] <inf> race_condition_demo_fixed: Thread B -> contador = 72
[00:00:08.132,000] <inf> race_condition_demo_fixed: Thread A -> contador = 73
[00:00:08.157,000] <inf> race_condition_demo_fixed: Thread B -> contador = 74
[00:00:08.357,000] <inf> race_condition_demo_fixed: Thread A -> contador = 75
[00:00:08.382,000] <inf> race_condition_demo_fixed: Thread B -> contador = 76
[00:00:08.582,000] <inf> race_condition_demo_fixed: Thread A -> contador = 77
[00:00:08.608,000] <inf> race_condition_demo_fixed: Thread B -> contador = 78
```

### Avaliação Cruzada do Código do Alberto

Contexto:

- Duas threads independentes que acessam o mesmo contador global.
- O código já faz um auto-diagnóstico de funcionamento, mostrando automaticamente no monitor serial o valor final do contador e o valor final esperado, de modo a evidenciar o problema.

O que estava errado antes:

- Nos cenários propostos 1 e 2, o resultado final do contador (1000) é diferente do valor final esperado após a execução do código (2000), devido à ocorrência de race condition.
- Já no cenário 3, que apresenta o resultado final igual ao esperado, a ocorrência desse problema é mascarada pela atribuição de uma prioridade maior a Thread A e pela definição de seu delay como 0. Nessas condições, não ocorreu o uso simultâneo do recurso compartilhado, pois B inicia sua tarefa somente após A ter concluído todos os seus ciclos, justificando a ausência de race condition e o resultado correto encontrado.

O que mudou com a correção:

- Tanto na versão corrigida utilizando Lock manual, quanto na versão com Mutex, o resultado final do contador global é igual ao valor final esperado (2000), em todos os cenários. Este valor é estável, sendo cada um dos cenários verificados 3 vezes em ambas as implementações.