Primos até N (com Semáfotos)

O objetivo da tarefa é estudar concorrencia de programas calculando a quantidade de primos de 1 a N, distribuindo a carga dinamicamente entre as threads utilizando a `pthread.h` a partur de uma implementação de Produtor/Consumidor com semáforos.

Geração dos Binários

O binário foi gerado com o comando:

```
gcc atividade.c -Wall -lm
```

Versão de Debug

Note que a versão verborrágica com o passo a passo pode ser gerada incluindo a macro `DEBUG` na compilação:

```
gcc atividade.c -Wall -lm -DDEBUG
```

Detalhe de Implementação

Como solicitado, foi utilizado o padrão Produtor/Consumidor. Porém, vale ressaltar, que implementei da seguinte forma:

- Quando o canal está vazio, o produtor preenche o canal inteiro de uma vez;
- Os consumidores vão consumindo o canal, e, quando está vazio, enviam um único sinal para a thread produtora encher o canal.

Note, também, que, garanti a atomicidade na implementação da thread produtora. Dessa maneira, se quiséssemos aumentar a quantidade de produtores, o código estaria preparado. Como não é o caso, entendo que gera o overhead da exclusão mútua desnecessariamente, mas tudo bem.

Exemplo de Saída

`M` É Múltiplo de `N`

Segue um exemplo de execução com os parâmetros:

- 3 threads consumidoras; e
- Verificando os primos até 25; e
- Canal de tamanho 5.

O comando utilizado, portanto, foi `./a.out 3 25 5`

```
[1:117] Thread 0: Produzi um batch de 5 (até 5).
[1:148] Thread 1: Iniciando rotina de consumismo.
[1:168] Thread 1: Verificando n = 1.
```

```
[l:168] Thread 1: Verificando n = 2.
[l:168] Thread 1: Verificando n = 3.
[l:168] Thread 1: Verificando n = 4.
[l:159] Thread 1: Enviando sinal para reabastecer o canal.
[l:168] Thread 1: Verificando n = 5.
[l:117] Thread 0: Produzi um batch de 5 (até 10).
[l:168] Thread 1: Verificando n = 6.
[l:168] Thread 1: Verificando n = 7.
[l:168] Thread 1: Verificando n = 8.
[l:168] Thread 1: Verificando n = 9.
[l:159] Thread 1: Enviando sinal para reabastecer o canal.
[l:148] Thread 2: Iniciando rotina de consumismo.
[l:168] Thread 1: Verificando n = 10.
[l:117] Thread 0: Produzi um batch de 5 (até 15).
[l:148] Thread 3: Iniciando rotina de consumismo.
[l:168] Thread 3: Verificando n = 11.
[l:168] Thread 3: Verificando n = 12.
[l:168] Thread 2: Verificando n = 13.
[l:168] Thread 3: Verificando n = 14.
[l:159] Thread 2: Enviando sinal para reabastecer o canal.
[l:168] Thread 2: Verificando n = 15.
[l:117] Thread 0: Produzi um batch de 5 (até 20).
[l:168] Thread 1: Verificando n = 16.
[l:168] Thread 3: Verificando n = 17.
[l:168] Thread 2: Verificando n = 18.
[l:168] Thread 1: Verificando n = 19.
[l:159] Thread 3: Enviando sinal para reabastecer o canal.
[l:168] Thread 3: Verificando n = 20.
[l:117] Thread 0: Produzi um batch de 5 (até 25).
[l:168] Thread 2: Verificando n = 21.
[l:168] Thread 1: Verificando n = 22.
[l:168] Thread 3: Verificando n = 23.
[l:168] Thread 2: Verificando n = 24.
[l:159] Thread 1: Enviando sinal para reabastecer o canal.
[l:168] Thread 1: Verificando n = 25.
[l:174] Thread 1: Encontrei n = 25 == N! Setando terminei = 1.
[l:196] Thread 1: Fim.
[l:117] Thread 0: Produzi um batch de 5 (até 30).
[l:181] Thread 3: Encontrei n = 26 > N! Ignorando.
[l:196] Thread 3: Fim.
[l:181] Thread 2: Encontrei n = 27 > N! Ignorando.
[l:192] Thread 2: Enviando sinal terminal para produtor.
[l:196] Thread 2: Fim.
[l:128] Thread 0: Fim.

RESUMO
```

```
------
Thread #1
  Avaliados:          14
  Primos descobertos: 5
Thread #2
  Avaliados:          5
  Primos descobertos: 1
Thread #3
  Avaliados:          6
  Primos descobertos: 3

Total Avaliados: 25
Total de Primos: 9

Vencedora: Thread #1
```

Note que para que as threads consumidoras identifiquem que a tarefa chegou ao fim (encontraram `n > N`), foi necessário produzir um canal inteiro extra. Essa foi a minha decisão de projeto consciente.

`M` Não É Múltiplo de `N`

Segue um exemplo de execução com os parâmetros:

- 3 threads consumidoras; e
- Verificando os primos até 25; e
- Canal de tamanho 6.

O comando utilizado, portanto, foi `./a.out 3 25 6`

```
[l:117] Thread 0: Produzi um batch de 6 (até 6).
[l:148] Thread 1: Iniciando rotina de consumismo.
[l:168] Thread 1: Verificando n = 1.
[l:168] Thread 1: Verificando n = 2.
[l:168] Thread 1: Verificando n = 3.
[l:168] Thread 1: Verificando n = 4.
[l:168] Thread 1: Verificando n = 5.
[l:159] Thread 1: Enviando sinal para reabastecer o canal.
[l:168] Thread 1: Verificando n = 6.
[l:148] Thread 2: Iniciando rotina de consumismo.
[l:117] Thread 0: Produzi um batch de 6 (até 12).
[l:168] Thread 1: Verificando n = 7.
[l:168] Thread 1: Verificando n = 8.
[l:168] Thread 2: Verificando n = 9.
[l:168] Thread 1: Verificando n = 10.
[l:168] Thread 2: Verificando n = 11.
[l:159] Thread 2: Enviando sinal para reabastecer o canal.
```

```
[1:148] Thread 3: Iniciando rotina de consumismo.
[1:168] Thread 2: Verificando n = 12.
[1:117] Thread 0: Produzi um batch de 6 (até 18).
[1:168] Thread 1: Verificando n = 13.
[1:168] Thread 1: Verificando n = 14.
[1:168] Thread 3: Verificando n = 15.
[1:168] Thread 2: Verificando n = 16.
[1:168] Thread 1: Verificando n = 17.
[1:159] Thread 1: Enviando sinal para reabastecer o canal.
[1:168] Thread 1: Verificando n = 18.
[1:117] Thread 0: Produzi um batch de 6 (até 24).
[1:168] Thread 2: Verificando n = 19.
[1:168] Thread 1: Verificando n = 20.
[1:168] Thread 3: Verificando n = 21.
[1:168] Thread 2: Verificando n = 22.
[1:168] Thread 1: Verificando n = 23.
[1:159] Thread 3: Enviando sinal para reabastecer o canal.
[1:168] Thread 3: Verificando n = 24.
[1:117] Thread 0: Produzi um batch de 6 (até 30).
[1:168] Thread 2: Verificando n = 25.
[1:174] Thread 2: Encontrei n = 25 == N! Setando terminei = 1.
[1:196] Thread 2: Fim.
[1:181] Thread 1: Encontrei n = 26 > N! Ignorando.
[1:196] Thread 1: Fim.
[1:181] Thread 3: Encontrei n = 27 > N! Ignorando.
[1:192] Thread 3: Enviando sinal terminal para produtor.
[1:196] Thread 3: Fim.
[1:128] Thread 0: Fim.

RESUMO
------
Thread #1
  Avaliados:          15
  Primos descobertos: 7
Thread #2
  Avaliados:          7
  Primos descobertos: 2
Thread #3
  Avaliados:          3
  Primos descobertos: 0

Total Avaliados: 25
Total de Primos: 9

Vencedora: Thread #1
```

Dessa vez, no final da execução, não foi necessário produzir mais inteiros, mas, de qualquer forma, houve overhead da thread produtora. Foram produzidos até `N = 30`, mas foram avaliados apenas `25`. Mais uma decisão de projeto consciente.