Laboratório 7
Problemas clássicos de concorrência usando locks e variáveis de condição (produtor/consumidor)

Computação Concorrente (MAB-117)
Prof. Silvana Rossetto

[1]DCC/IM/UFRJ — 3 de outubro de 2019

Introdução

O objetivo deste Laboratório é implementar problemas clássicos de concorrência usando locks e variáveis de condição. Começaremos com o problema dos **produtores/consumidores**.

Atividade 1

Objetivo: Projetar e implementar uma aplicação produtor/consumidor.

Roteiro: Implemente uma aplicação em C com duas threads: uma que gera e deposita números inteiros em um **buffer** ("produtor") e outra que consome esses elementos ("consumidor").

1. Defina um buffer de **5** elementos.
2. A thread produtora gera os **25** primeiros elementos da série de Fibonacci: $F_n = F_{n-1} + F_{n-2}$, sendo $F(1) = 1$ e $F(2) = 1$
3. A thread consumidora retira um número e verifica a sua primalidade (veja dica de função abaixo).
4. Os elementos devem ser consumidos na mesma ordem em que são inseridos no buffer e nenhum elemento deve ser perdido (sobreescrito) no buffer.
5. A thread produtora deverá ser bloqueada sempre que tentar inserir um elemento e encontrar o buffer cheio.
6. A thread consumidora deverá ser bloqueada sempre que tentar retirar um elemento e encontrar o buffer vazio.
7. Execute o programa **várias vezes** e verifique se a solução está correta.

```
//funcao para determinar se um numero é primo
int ehPrimo(long unsigned int n) {
   int i;
      if(n<=1) return 0;
      if(n==2) return 1;
      if(n%2==0) return 0;
      for(i=3; i< sqrt(n)+1; i+=2) {
         if(n%i==0) return 0;
      }
      return 1;
}
```

Obs.: Os 25 primeiros números da série de Fibonacci são:
1, 1, 2 (primo), 3 (primo), 5 (primo), 8, 13 (primo), 21, 34, 55, 89 (primo), 144, 233 (primo), 377, 610, 987, 1597 (primo), 2584, 4181, 6765, 10946, 17711, 28657 (primo), 46368, 75025

Atividade 2

Roteiro:

1. Altere a aplicação anterior (Atividade 1) para que ela possa executar com **um produtor** e **vários consumidores**.
2. Execute o programa **várias vezes**, alterando o número de consumidores, e verifique se a solução está correta.

Atividade 3

Roteiro:

1. Altere a lógica da aplicação anterior (Atividade 2) fazendo o **consumidor** processar o buffer inteiro a cada operação de remoção (ao invés de remover e processar apenas um elemento de cada vez). Para isso, o consumidor deverá aguardar o buffer ficar completamente cheio.
2. Mantenha a lógica do produtor inalterada. O programa poderá criar mais de uma thread consumidora.
3. Execute o programa **várias vezes**, alterando o número de consumidores, e verifique se a solução está correta.