# Modelos para Computação Concorrente ou Sistemas Operacionais

## Memória Compartilhada – Semáforos – Modelo Produtor-Consumidor

(com slides de Ben-Ari)

Fernando Luís Dotti

PUCRS | ESCOLA POLITÉCNICA

# Bibliografia Base

[disponível na biblioteca]

**M. Ben-Ari**

**Principles of Concurrent and Distributed Programming**

**Second Edition**

**Addison-Wesley, 2006**

# Modelo Produtor/Consumidor

- Produção e consumo
  - situação pervasiva:  protocolos de comunicação, sistemas operacionais, processos colaborativos diversos, etc.
  - buffer infinito (construção teórica)
  - buffer finito

## Algorithm 6.6: Producer-consumer (infinite buffer)

```
infinite queue of dataType buffer ← empty queue
semaphore notEmpty ← $(0, \emptyset)$
```

| **producer** | **consumer** |
|---|---|
| dataType d | dataType d |
| loop forever | loop forever |
| p1: d ← produce | q1: wait(notEmpty) |
| p2: append(d, buffer) | q2: d ← take(buffer) |
| p3: signal(notEmpty) | q3: consume(d) |

### Algorithm 6.8: Producer-consumer (finite buffer, semaphores)

```
finite queue of dataType buffer ← empty queue
semaphore notEmpty ← $(0, \emptyset)$
semaphore notFull ← $(N, \emptyset)$
```

| | producer | | consumer |
|---|---|---|---|
| | dataType d | | dataType d |
| | loop forever | | loop forever |
| p1: | d ← produce | q1: | wait(notEmpty) |
| p2: | wait(notFull) | q2: | d ← take(buffer) |
| p3: | append(d, buffer) | q3: | signal(notFull) |
| p4: | signal(notEmpty) | q4: | consume(d) |

finite queue of dataType buffer ← empty queue
semaphore notEmpty ← $(0, \emptyset)$
semaphore notFull ← $(N, \emptyset)$

| | **producer** | | **consumer** |
|---|---|---|---|
| | dataType d | | dataType d |
| | loop forever | | loop forever |
| p1: | d ← produce | q1: | wait(notEmpty) |
| p2: | wait(notFull) | q2: | d ← take(buffer) |
| p3: | append(d, buffer) | q3: | signal(notFull) |
| p4: | signal(notEmpty) | q4: | consume(d) |

| | | | | | |
|---|---|---|---|---|---|
| | | estado inicial, N=3 | | | buffer [_,_,_] |
| p1: | ... | | | | buffer [_,_,_] |
| p2: | wait(notFull) | | | | buffer [_,_,_] |
| | | notFull=(2,{}) | | | buffer [_,_,_] |
| ... | | | q1: | wait(notEmpty) | buffer [_,_,_] |
| | | notEmpty=(0,{q}) | | | buffer [_,_,_] |
| p3: | append (...) | | q: | blocked | buffer [I1,_,_] |
| p4: | signal(notEmpty) | | | | buffer [I1,_,_] |
| | | notEmpty=(0,{}) | q: | unblocked - wait completes | buffer [I1,_,_] |
| p1: | ... | | q2: | ... take | buffer [_,_,_] |
| | | | q3: | signal(notFull) | buffer [_,_,_] |
| | | notFull=(3,{}) | | | |
| | | | q4: | consume | |

```
finite queue of dataType buffer ← empty queue
semaphore notEmpty ← $(0, \emptyset)$
semaphore notFull ← $(N, \emptyset)$
```

| | **producer** | | **consumer** |
|---|---|---|---|
| | dataType d | | dataType d |
| | loop forever | | loop forever |
| p1: |    d ← produce | q1: |    wait(notEmpty) |
| p2: |    wait(notFull) | q2: |    d ← take(buffer) |
| p3: |    append(d, buffer) | q3: |    signal(notFull) |
| p4: |    signal(notEmpty) | q4: |    consume(d) |

| | | | |
|---|---|---|---|
| | estado inicial, N=3 | | buffer [_,_,_] |
| p1: ... | | | buffer [_,_,_] |
| p2: wait(notFull) | | | buffer [_,_,_] |
| | notFull=(2,{}) | | buffer [_,_,_] |
| p3: append (...) | | | buffer [l1,_,_] |
| p4: signal(notEmpty) | | | buffer [l1,_,_] |
| | notEmpty=(1,{}) | | buffer [l1,_,_] |
| p1: ... | | | buffer [l1,_,_] |
| p2: wait(notFull) | | | buffer [l1,_,_] |
| | notFull=(1,{}) | | buffer [l1,_,_] |
| p3: append (...) | | | buffer [l1,l2,_] |
| p4: signal(notEmpty) | | | buffer [l1,l2,_] |
| | notEmpty=(2,{}) | | buffer [l1,l2,_] |
| p1: ... | | | buffer [l1,l2,_] |
| p2: wait(notFull) | | | buffer [l1,l2,_] |
| | notFull=(0,{}) | | buffer [l1,l2,_] |
| p3: append (...) | | | buffer [l1,l2,l3] |
| p4: signal(notEmpty) | | | buffer [l1,l2,l3] |
| | notEmpty=(3,{}) | | buffer [l1,l2,l3] |
| p1: ... | | | buffer [l1,l2,l3] |
| p2: wait(notFull) | | | buffer [l1,l2,l3] |
| | notFull=(0,{ p }) | | buffer [l1,l2,l3] |
| p: blocked | | | |
| | | q1: wait(notEmpty) | buffer [l1,l2,l3] |

# Prod / Cons – Exemplo em Java

```
/*
   Exemplo de produtor consumidor com buffer finito.
   PUCRS - Escola Politecnica
   Prof: Fernando Dotti
*/
import java.util.concurrent.Semaphore;

class FiniteBuffer {

  private int size;
  private int in = 0;
  private int out = 0;
  private int[] buffer;

  private Semaphore naoCheio;
  private Semaphore naoVazio;
  private Semaphore mutex;

  private void incrIn() {  in = (in+1)%size; }
  private void incrOut() { out = (out+1)%size; }

  public FiniteBuffer(int _size){
    size = _size;
    buffer = new int[size];          // armazena os itens

    mutex = new Semaphore(1);        // para exclusao mutua  (sc)
    naoCheio = new Semaphore(size);  // controle de espaco disponivel
    naoVazio = new Semaphore(0);     // controle de itens
  }

  public void insert(int v){
    try { naoCheio.acquire(); //   espera ter espaco
          mutex.acquire();    //   entra sc
    } catch (InterruptedException ie) {}
        buffer[in]=v;         //   sc: insere elemento
        incrIn();             //   sc: insere elemento
    mutex.release();          //   sai sc
    naoVazio.release();       //   avisa que nao esta vazio
  }

  public int delete(){
    int val;
    try {  naoVazio.acquire(); //   espera nao estar vazio
           mutex.acquire();    //   entra na sc
    }  catch (InterruptedException ie) {}
        val = buffer[out];     //   sc: retira elemento
        incrOut();             //   sc: retira elemento
    mutex.release();           //   sai da sc
    naoCheio.release();        //   avisa que tem espaco
    return val;
  }
}
```

```
class ProducerThread extends Thread {
  private int id;
  private int limit;
  private FiniteBuffer fb;

    public ProducerThread(int _id, FiniteBuffer _fb, int _limit){
           id = _id;      fb = _fb;     limit = _limit;
    }

    public void run() {
       for (int i = 0; i < limit; i++) {
         fb.insert(i);
         System.out.println("Prod "+id+"  val "+i);
       }
    }
}

class ConsumerThread extends Thread {
  private int id;
  private int limit;
  private FiniteBuffer fb;

    public ConsumerThread(int _id, FiniteBuffer _fb, int _limit){
           id = _id;      fb = _fb;   limit = _limit;
    }

    public void run() {
       int v;
       for (int i = 0; i < limit; i++) {
         v = fb.delete();
         System.out.println(”Cons "+id+"  val "+v);
       }
    }
}

class TesteProdCons {
    public static void main(String[] args) {

      FiniteBuffer fb = new FiniteBuffer(10);

      ProducerThread p = new ProducerThread(1,fb,10);
      ProducerThread q = new ProducerThread(2,fb,10);
      ProducerThread r = new ProducerThread(3,fb,10);

      ConsumerThread s = new ConsumerThread(3,fb,15);
      ConsumerThread t = new ConsumerThread(4,fb,15);

      p.start();  q.start();  r.start();  s.start();  t.start();
      try { p.join(); q.join(); r.join(); s.join(); t.join(); }
      catch (InterruptedException e) { }
      System.out.println("Fim ");
    }
}
```