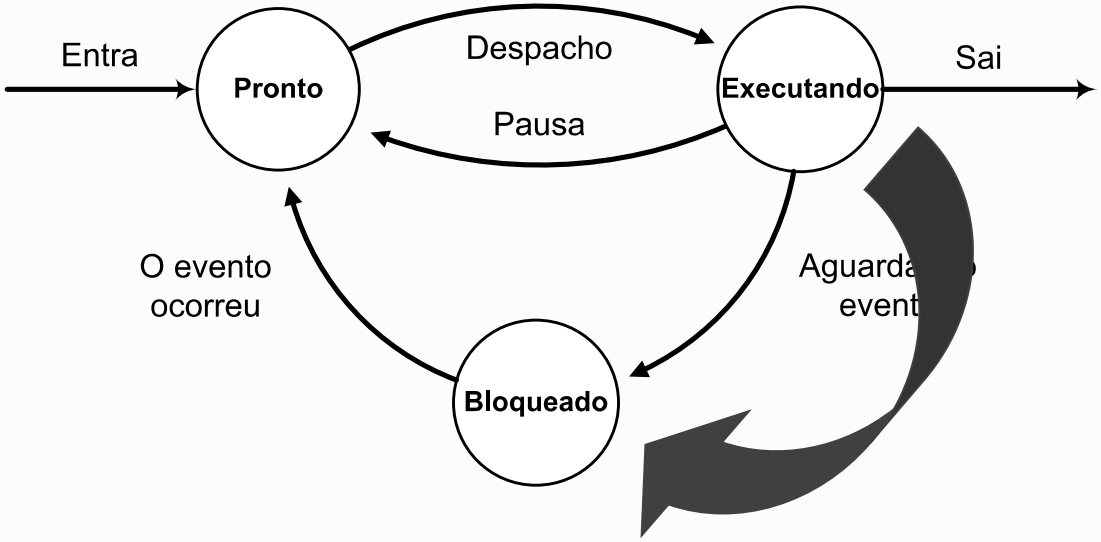
Sistemas Operacionais

Comunicação entre Processos

Sleep e Wakeup (Conceitos)

Sleep: é uma chamada de sistema que o processo/thread pode invocar;

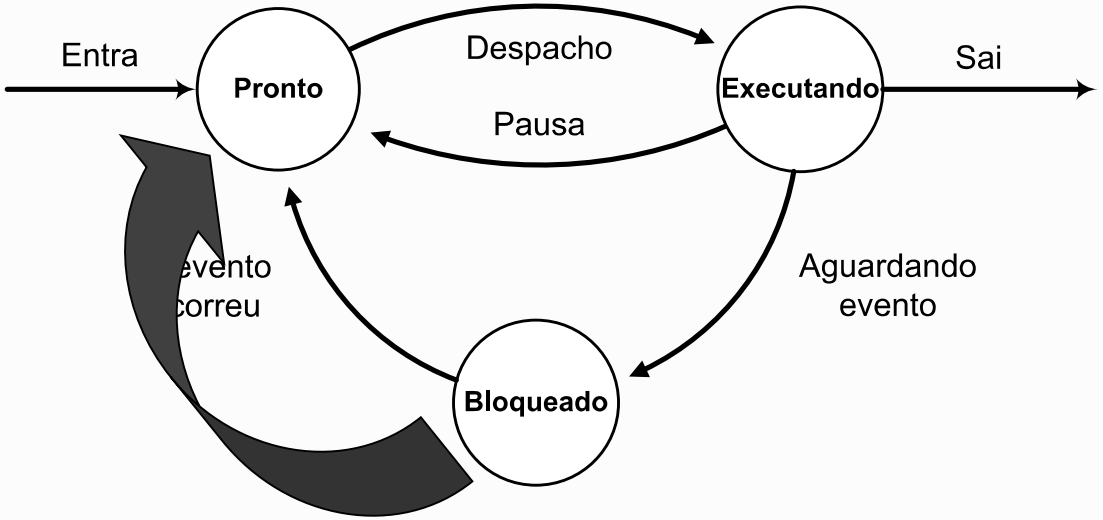
- Com esta chamada seu estado passa a ser bloqueado.
  O processo/thread que chamou o método é quem dorme.



Sleep e Wakeup (Conceitos)

Wakeup: é uma chamada de sistema que o processo/thread pode invocar;

- Um parâmetro deve ser informado: qual processo/thread deve ser acordado. Com esta chamada o processo indicado volta para a fila dos processos prontos.



Produtor-Consumidor

Dois processos compartilham um buffer comum de tamanho fixo.

- Processos "**produtor**" e "**consumidor**"
- Este problema pode ser encontrado em inúmeros casos práticos na Ciência da Computação e Engenharias.

Quando o produtor quer colocar um novo item no buffer e ele já está cheio, podemos colocar o processo para dormir (*sleep*);

Produtor-Consumidor

De forma equivalente, quando o consumidor quer retirar um item, mas o buffer está vazio, podemos colocar o processo consumidor para dormir (*sleep*).

Se o buffer está vazio, e o produtor insere um novo item no buffer, o mesmo pode disparar uma chamada de sistema na tentativa de "acordar" consumidores que estavam dormindo.

De forma análoga, quando o buffer está cheio, e o consumidor remove um item do buffer, este pode disparar uma chamada de sistema tentando acordar algum produtor que esteja dormindo.

Produtor-Consumidor

```
#define N 100                                   /* número de lugares no buffer */
int count = 0;                                  /* número de itens no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                            /* número de itens no buffer */
            item = produce_item( );             /* gera o próximo item */
            if (count == N) sleep( );           /* se o buffer estiver cheio, vá dormir */
            insert_item(item);                  /* ponha um item no buffer */
            count = count + 1;                  /* incremente o contador de itens no buffer */
            if (count == 1) wakeup(consumer);   /* o buffer estava vazio? */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                            /* repita para sempre */
            if (count == 0) sleep( );           /* se o buffer estiver vazio, vá dormir */
            item = remove_item( );              /* retire o item do buffer */
            count = count – 1;                  /* decresça de um o contador de itens no buffer */
            if (count == N – 1) wakeup(producer);  /* o buffer estava cheio? */
            consume_item(item);                 /* imprima o item */
      }
}
```

```
#define N 100                                   /* número de lugares no buffer */
int count = 0;                                  /* número de itens no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* número de itens no buffer */
          item = produce_item( );               /* gera o próximo item */
          if (count == N) sleep( );             /* se o buffer estiver cheio, vá dormir */
          insert_item(item);                    /* ponha um item no buffer */
          count = count + 1;                    /* incremente o contador de itens no buffer */
          if (count == 1) wakeup(consumer);     /* o buffer estava vazio? */
     }
}
```

```
void consumer(void)
{
     int item;

     while (TRUE) {                             /* repita para sempre */
          if (count == 0) sleep( );             /* se o buffer estiver vazio, vá dormir */
          item = remove_item( );                /* retire o item do buffer */
          count = count – 1;                    /* decresça de um o contador de itens no buffer */
          if (count == N – 1) wakeup(producer); /* o buffer estava cheio? */
          consume_item(item);                   /* imprima o item */
     }
}
```

Processo Ativo

```
#define N 100                                   /* número de lugares no buffer */
int count = 0;                                  /* número de itens no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* número de itens no buffer */
          item = produce_item( );               /* gera o próximo item */
          if (count == N) sleep( );             /* se o buffer estiver cheio, vá dormir */
          insert_item(item);                    /* ponha um item no buffe
          count = count + 1;                    /* incremente o contador
          if (count == 1) wakeup(consumer);     /* o buffer estava vazio?
     }
}
```

O consumidor tenta ler um item, mas o buffer está vazio.

```
void consumer(void)
{
     int item;

     while (TRUE) {                             /* repita para sempre */
          if (count == 0) sleep( );             /* se o buffer estiver vazio, vá dormir */
          item = remove_item( );                /* retire o item do buffer */
          count = count – 1;                    /* decresça de um o contador de itens no buffer */
          if (count == N – 1) wakeup(producer); /* o buffer estava cheio? */
          consume_item(item);                   /* imprima o item */
     }
}
```

Processo Ativo

```
#define N 100                                   /* número de lugares no buffer */
int count = 0;                                  /* número de itens no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* número de itens no buffer */
          item = produce_item( );               /* gera o próximo item */
          if (count == N) sleep( );             /* se o buffer estiver cheio, vá dormir */
          insert_item(item);                    /* ponha um item no buffe
          count = count + 1;                    /* incremente o contador
          if (count == 1) wakeup(consumer);     /* o buffer estava vazio?
     }
}
```

Antes que o processo entre no estado bloqueado, o S.O. efetua o escalonamento.

```
void consumer(void)
{
     int item;

     while (TRUE) {                             /* repita para sempre */
          if (count == 0) sleep( );             /* se o buffer estiver vazio, vá dormir */
          item = remove_item( );                /* retire o item do buffer */
          count = count – 1;                    /* decresça de um o contador de itens no buffer */
          if (count == N – 1) wakeup(producer); /* o buffer estava cheio? */
          consume_item(item);                   /* imprima o item */
     }
}
```

Processo Ativo

```
#define N 100                                   /* número de lugares no buffer */
int count = 0;                                  /* número de itens no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* número de itens no buffer */
          item = produce_item( );               /* gera o próximo item */
          if (count == N) sleep( );             /* se o buffer estiver cheio, vá dormir */
          insert_item(item);                    /* ponha um item no buffer */
          count = count + 1;                    /* incremente o contador de itens no buffer */
          if (count == 1) wakeup(consumer);     /* o buffer estava vazio? */
     }
}

void consumer(void)
{
     int item;

     while (TRUE) {                             /* repita para sempre */
          if (count == 0) sleep( );             /* se o buffer estiver vazio, vá dormir */
          item = remove_item( );                /* retire o item do buffer */
          count = count – 1;                    /* decresça de um o contador de itens no buffer */
          if (count == N – 1) wakeup(producer); /* o buffer estava cheio? */
          consume_item(item);                   /* imprima o item */
     }
}
```

Processo Ativo

```
#define N 100                                   /* número de lugares no buffer */
int count = 0;                                  /* número de itens no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* número de itens no buffer */
          item = produce_item( );               /* gera o próximo item */
          if (count == N) sleep( );             /* se o buffer estiver cheio, vá dormir */
          insert_item(item);                    /* ponha um item no buffer */
          count = count + 1;                    /* incremente o contador de itens no buffer */
          if (count == 1) wakeup(consumer);     /* o buffer estava vazio? */
     }
}

void consumer(void)
{
     int item;

     while (TRUE) {                             /* repita para sempre */
          if (count == 0) sleep( );             /* se o buffer estiver vazio, vá dormir */
          item = remove_item( );                /* retire o item do buffer */
          count = count – 1;                    /* decresça de um o contador de itens no buffer */
          if (count == N – 1) wakeup(producer); /* o buffer estava cheio? */
          consume_item(item);                   /* imprima o item */
     }
}
```

O item é produzido.

Processo Ativo

```
#define N 100                                   /* número de lugares no buffer */
int count = 0;                                  /* número de itens no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* número de itens no buffer */
          item = produce_item( );               /* gera o próximo item */
          if (count == N) sleep( );             /* se o buffer estiver cheio, vá dormir */
          insert_item(item);                    /* ponha um item no buffer */
          count = count + 1;                    /* incremente o contador de itens no buffer */
          if (count == 1) wakeup(consumer);     /* o buffer estava vazio? */
     }
}

void consumer(void)
{
     int item;

     while (TRUE) {                             /* repita para sempre */
          if (count == 0) sleep( );             /* se o buffer estiver vazio, vá dormir */
          item = remove_item( );                /* retire o item do buffer */
          count = count – 1;                    /* decresça de um o contador de itens no buffer */
          if (count == N – 1) wakeup(producer); /* o buffer estava cheio? */
          consume_item(item);                   /* imprima o item */
     }
}
```

O buffer está vazio, então a condicional falha.

Processo Ativo

```
#define N 100                                   /* número de lugares no buffer */
int count = 0;                                  /* número de itens no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* número de itens no buffer */
          item = produce_item( );               /* gera o próximo item */
          if (count == N) sleep( );             /* se o buffer estiver cheio, vá dormir */
          insert_item(item);                    /* ponha um item no buffer */
          count = count + 1;                    /* incremente o contador de itens no buffer */
          if (count == 1) wakeup(consumer);     /* o buffer estava vazio? */
     }
}

void consumer(void)
{
     int item;

     while (TRUE) {                             /* repita para sempre */
          if (count == 0) sleep( );             /* se o buffer estiver vazio, vá dormir */
          item = remove_item( );                /* retire o item do buffer */
          count = count – 1;                    /* decresça de um o contador de itens no buffer */
          if (count == N – 1) wakeup(producer); /* o buffer estava cheio? */
          consume_item(item);                   /* imprima o item */
     }
}
```

O item é inserido no buffer

Processo Ativo

```
#define N 100                                   /* número de lugares no buffer */
int count = 0;                                  /* número de itens no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* número de itens no buffer */
          item = produce_item( );               /* gera o próximo item */
          if (count == N) sleep( );             /* se o buffer estiver cheio, vá dormir */
          insert_item(item);                    /* ponha um item no buffer */
          count = count + 1;                    /* incremente o contador de itens no buffer */
          if (count == 1) wakeup(consumer);     /* o buffer estava vazio? */
     }
}

void consumer(void)
{
     int item;

     while (TRUE) {                             /* repita para sempre */
          if (count == 0) sleep( );             /* se o buffer estiver vazio, vá dormir */
          item = remove_item( );                /* retire o item do buffer */
          count = count – 1;                    /* decresça de um o contador de itens no buffer */
          if (count == N – 1) wakeup(producer); /* o buffer estava cheio? */
          consume_item(item);                   /* imprima o item */
     }
}
```

count recebe 1

Processo Ativo

```
#define N 100                                   /* número de lugares no buffer */
int count = 0;                                  /* número de itens no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* número de itens no buffer */
          item = produce_item( );               /* gera o próximo item */
          if (count == N) sleep( );             /* se o buffer estiver cheio, vá dormir */
          insert_item(item);                    /* ponha um item no buffer */
          count = count + 1;                    /* incremente o contador de itens no buffer */
          if (count == 1) wakeup(consumer);     /* o buffer estava vazio? */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                             /* repita para sempre */
          if (count == 0) sleep( );             /* se o buffer estiver vazio, vá dormir */
          item = remove_item( );                /* retire o item do buffer */
          count = count – 1;                    /* decresça de um o contador de itens no buffer */
          if (count == N – 1) wakeup(producer); /* o buffer estava cheio? */
          consume_item(item);                   /* imprima o item */
     }
}
```

Processo Ativo

Se o item foi produzido, o outro processo deve acordar. Mas repare, ele ainda não dormiu de fato.

```
#define N 100                                   /* número de lugares no buffer */
int count = 0;                                  /* número de itens no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* número de itens no buffer */
          item = produce_item( );               /* gera o próximo item */
          if (count == N) sleep( );             /* se o buffer estiver cheio, vá dormir */
          insert_item(item);                    /* ponha um item no buffer */
          count = count + 1;                    /* incremente o contador de itens no buffer */
          if (count == 1) wakeup(consumer);     /* o buffer estava vazio? */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                             /* repita para sempre */
          if (count == 0) sleep( );             /* se o buffer estiver vazio, vá dormir */
          item = remove_item( );                /* retire o item do buffer */
          count = count – 1;                    /* decresça de um o contador de itens no buffer */
          if (count == N – 1) wakeup(producer); /* o buffer estava cheio? */
          consume_item(item);                   /* imprima o item */
     }
}
```

Processo Ativo

O processo produtor tenta acordar o processo consumidor, mas o sinal é perdido, pois o outro processo ainda não dormiu.

```
#define N 100                                   /* número de lugares no buffer */
int count = 0;                                  /* número de itens no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* número de itens no buffer */
          item = produce_item( );               /* gera o próximo item */
          if (count == N) sleep( );             /* se o buffer estiver cheio, vá dormir */
          insert_item(item);                    /* ponha um item no buffer */
          count = count + 1;                    /* incremente o contador de itens no buffer */
          if (count == 1) wakeup(consumer);     /* o buffer estava vazio? */
     }
}

void consumer(void)
{
     int item;

     while (TRUE) {                             /* repita para sempre */
          if (count == 0) sleep( );             /* se o buffer estiver vazio, vá dormir */
          item = remove_item( );                /* retire o item do buffer */
          count = count – 1;                    /* decresça de um o contador de itens no buffer */
          if (count == N – 1) wakeup(producer); /* o buffer estava cheio? */
          consume_item(item);                   /* imprima o item */
     }
}
```

Em *loop*, O processo produtor vai produzir itens até que o buffer esteja cheio.

Processo Ativo

```
#define N 100                                   /* número de lugares no buffer */
int count = 0;                                  /* número de itens no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* número de itens no buffer */
          item = produce_item( );               /* gera o próximo item */
          if (count == N) sleep( );             /* se o buffer estiver cheio, vá dormir */
          insert_item(item);                    /* ponha um item no buffer */
          count = count + 1;                    /* incremente o contador de itens no buffer */
          if (count == 1) wakeup(consumer);     /* o buffer estava vazio? */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                             /* repita para sempre */
          if (count == 0) sleep( );             /* se o buffer estiver vazio, vá dormir */
          item = remove_item( );                /* retire o item do buffer */
          count = count – 1;                    /* decresça de um o contador de itens no buffer */
          if (count == N – 1) wakeup(producer); /* o buffer estava cheio? */
          consume_item(item);                   /* imprima o item */
     }
}
```

Processo Ativo

Se o buffer está cheio, o processo irá dormir (estado bloqueado)

```
#define N 100                                   /* número de lugares no buffer */
int count = 0;                                  /* número de itens no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* número de itens no buffer */
          item = produce_item( );               /* gera o próximo item */
          if (count == N) sleep( );             /* se o buffer estiver cheio, vá dormir */
          insert_item(item);                    /* ponha um item no buffer */
          count = count + 1;                    /* incremente o contador de itens no buffer */
          if (count == 1) wakeup(consumer);     /* o buffer estava vazio? */
     }
}

void consumer(void)
{
     int item;

     while (TRUE) {                             /* repita para sempre */
          if (count == 0) sleep( );             /* se o buffer estiver vazio, vá dormir */
          item = remove_item( );                /* retire o item do buffer */
          count = count – 1;                    /* decresça de um o contador de itens no buffer */
          if (count == N – 1) wakeup(producer); /* o buffer estava cheio? */
          consume_item(item);                   /* imprima o item */
     }
}
```

Processo Bloqueado

Processo Ativo

O processo consumidor assume o processador e também dorme.

```
#define N 100                                   /* número de lugares no buffer */
int count = 0;                                  /* número de itens no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* número de it                ffe
          item = produce_item( );                    era o próxi
          if (count == N) sleep( );                  uffer
          insert_item(item);
          count = count + 1;
          if (count == 1) wa
     }
}

void consumer(void)
{
     int item;

     while (TRUE) {
          if (count == 0) sleep( );                  iver va        vá dormir */
          item = remove_item( );                     tire o it      do buffer */
          count = count – 1;                         decresça de um o contador de itens no buffer */
          if (count == N – 1) wakeup(prod        er); /* o buffer estava cheio? */
          consume_item(item);                   /* imprima o item */
     }
}
```

Situação indesejável!!!
Os processos dormem eternamente
DEADLOCK!!!!

Proces Bl

ocesso Bloqueado

Semáforos

O Holandês Dijkstra alcançou em 1965 uma solução para a competição em regiões críticas, sem utilizar a espera ocupada.

Sua proposta incluiu um novo tipo de variável:

- O **semáforo** (variável inteira positiva):
- Pode indicar a quantidade de itens utilizados no buffer, por exemplo.
  - Acompanhado de duas operações básicas:
    - **down**(*Semaforo s*);
    - **up**(*Semaforo s*);

Semáforos – Operação down

```
public synchronized void down(Semaforo s) throws Exception {
   // Garante a atomicidade da execução do método.
   synchronized(this){
      // Enquanto não tem acesso ao semáforo, a thread aguarda.
      while( s.contador == 0 ){
         this.wait();
      }
      // Quando a thread tem acesso, o semáforo é decrementado em uma unidade.
      s.contador--;
   }
}
```

Esta função deve ser indivisível.
Atômica

Mas ser indivisível, não cai na necessidade de desabilitar interrupção de relógio?

Mas aqui a operação é rápida, e se a interrupção de relógio for desabilitada por um breve instante, não haverá problemas.

A nível de processo, são implementadas como chamadas de sistema (*up*/*down*);

Assim, processos de usuário não podem utilizar "*de qualquer forma*" a interrupção de *clock* diretamente.

Semáforos – Operação down

```
public synchronized void up(Semaforo s) throws Exception {
   // Garante a atomicidade da execução do método.
   synchronized(this){
      // Quando a thread libera o semáforo, ele é incrementado em uma unidade.
      // Isso possibilita que outra thread (que está dormindo) possa acessar
      // a região critica controlada pelo semáforo.
      num.contador++;
     // Acorda todas as outras threads que estão dormindo.
     // Uma delas vai obter acesso a região crítica
      this.notifyAll();
   }
}
```

Esta função deve ser indivisível.

Semáforos – Solução do Problema

A solução do problema "Produtor-Consumidor" utiliza 3 (três) semáforos:

- **full**: para contar o número de elementos do buffer que estão preenchidos;
- **empty**: para contar o número de elementos do buffer que estão vazios;
- **mutex**: para assegurar que produtor e consumidor não tenham acesso ao buffer ao mesmo tempo;

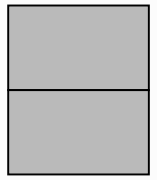
```
#define N 2                                     /* número de lugares no buffer */
typedef int semaphore;                          /* semáforos são um tipo especial de int */
semaphore mutex = 1;                            /* controla o acesso à região crítica */
semaphore empty = N;                            /* conta os lugares vazios no buffer */
semaphore full = 0 ;                            /* conta os lugares preenchidos no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* TRUE é a constante 1 */
          item = produce_item( );               /* gera algo para pôr no buffer */
          down(&empty);                         /* decresce o contador empty */
          down(&mutex);                         /* entra na região crítica */
          insert_item(item);                    /* põe novo item no buffer */
          up(&mutex);                           /* sai da região crítica */
          up(&full);                            /* incrementa o contador de lugares preenchidos */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                             /* laço infinito */
          down(&full);                          /* decresce o contador full */
          down(&mutex);                         /* entra na região crítica */
          item = remove_item( );                /* pega o item do buffer */
          up(&mutex);                           /* deixa a região crítica */
          up(&empty);                           /* incrementa o contador de lugares vazios */
          consume_item(item);                   /* faz algo com o item */
     }
}
```

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                 /* TRUE é a constante 1 */
          item = produce_item( );   /* gera algo para pôr no buffer */
          down(&empty);             /* decresce o contador empty */
          down(&mutex);             /* entra na região crítica */
          insert_item(item);        /* põe novo item no buffer */
          up(&mutex);               /* sai da região crítica */
          up(&full);                /* incrementa o contador de lugares preenchidos */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                 /* laço infinito */
          down(&full);              /* decresce o contador full */
          down(&mutex);             /* entra na região crítica */
          item = remove_item( );    /* pega o item do buffer */
          up(&mutex);               /* deixa a região crítica */
          up(&empty);               /* incrementa o contador de lugares vazios */
          consume_item(item);       /* faz algo com o item */
     }
}
```

Exemplo com N = 2

item = ???

Buffer



N=2
mutex = 1
empty = 2
full = 0

Inicialização das variáveis

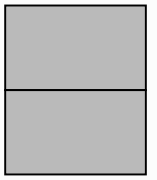
```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                 /* TRUE é a constante 1 */
          item = produce_item( );   /* gera algo para pôr no buffer */
          down(&empty);             /* decresce o contador empty */
          down(&mutex);             /* entra na região crítica */
          insert_item(item);        /* põe novo item no buffer */
          up(&mutex);               /* sai da região crítica */
          up(&full);                /* incrementa o contador de lugares preenchidos */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                 /* laço infinito */
          down(&full);              /* decresce o contador full */
          down(&mutex);             /* entra na região crítica */
          item = remove_item( );    /* pega o item do buffer */
          up(&mutex);               /* deixa a região crítica */
          up(&empty);               /* incrementa o contador de lugares vazios */
          consume_item(item);       /* faz algo com o item */
     }
}
```

item = ???

Suponha que o consumidor assume o processador. Repare que o buffer no início está vazio.

Exemplo com N = 2

Buffer



N=2
mutex = 1
empty = 2
full = 0

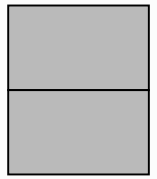
```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                 /* TRUE é a constante 1 */
          item = produce_item( );   /* gera algo para pôr no buffer */
          down(&empty);             /* decresce o contador empty */
          down(&mutex);             /* entra na região crítica */
          insert_item(item);        /* põe novo item no buffer */
          up(&mutex);               /* sai da região crítica */
          up(&full);                /* incrementa o contador de lugares preenchidos */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                 /* laço infinito */
          down(&full);              /* decresce o contador full */
          down(&mutex);             /* entra na região crítica */
          item = remove_item( );    /* pega o item do buffer */
          up(&mutex);               /* deixa a região crítica */
          up(&empty);               /* incrementa o contador de lugares vazios */
          consume_item(item);       /* faz algo com o item */
     }
}
```

item = ???

O semáforo full vale 0, e a operação de down é efetuada sobre este valor. Então, a thread do consumidor dorme!

Exemplo com N = 2

Buffer



N=2
mutex = 1
empty = 2
full = 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                 /* TRUE é a constante 1 */
          item = produce_item( );   /* gera algo para pôr no buffer */
          down(&empty);             /* decresce o contador empty */
          down(&mutex);             /* entra na região crítica */
          insert_item(item);        /* põe novo item no buffer */
          up(&mutex);               /* sai da região crítica */
          up(&full);                /* incrementa o contador de lugares preenchidos */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                 /* laço infinito */
          down(&full);              /* decresce o contador full */
          down(&mutex);             /* entra na região crítica */
          item = remove_item( );    /* pega o item do buffer */
          up(&mutex);               /* deixa a região crítica */
          up(&empty);               /* incrementa o contador de lugares vazios */
          consume_item(item);       /* faz algo com o item */
     }
}
```

Na CPU

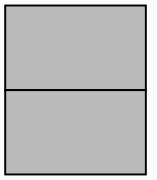
Dormindo

item = 46

O produtor produz o item que <u>será</u> armazenado na memória. Ex: 46

Exemplo com N = 2

Buffer



N=2
mutex = 1
empty = 2
full = 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

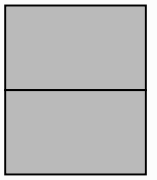
Dormindo

item = 46

Down sobre o semáforo empty

Exemplo com N = 2

Buffer



N=2
mutex = 1
empty = 2 => 1
full = 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

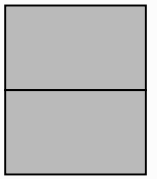
Dormindo

Down sobre o semáforo mutex.

item = 46

Exemplo com N = 2

Buffer



N=2
mutex = 1 => 0
empty = 1
full = 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}

void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU

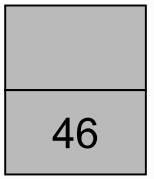
Dormindo

item = 46

O item é inserido no buffer.

Exemplo com N = 2

Buffer



N=2
mutex = 0
empty = 1
full = 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}

void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU

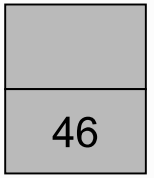
Dormindo

up no semáforo mutex

item = 46

Exemplo com N = 2

Buffer



N=2
mutex = 0 => 1
empty = 1
full = 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU

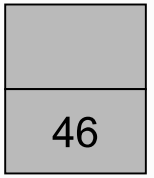
Dormindo => Fila de prontos

up no semáforo full. A thread do consumidor está aguardando a liberação de full

Exemplo com N = 2

item = 46

Buffer



N=2
mutex = 1
empty = 1
full = 0 => 1

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

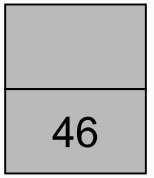
Na CPU =>
Fila de prontos

Suponha que o escalonador agora direciona a CPU para o consumidor

item = 46

Exemplo com N = 2

Buffer



N=2
mutex = 1
empty = 1
full = 1

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Fila de prontos

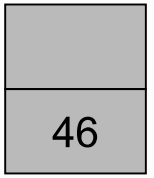
Na CPU

Agora, ele consegue passar da operação de down em full, pois sabe que existe um elemento produzido.

Exemplo com N = 2

item = 46

Buffer



N=2
mutex = 1
empty = 1
full = 1 => 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}

void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Fila de prontos

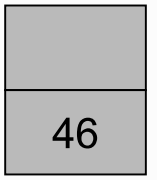
Na CPU

down sobre mutex. Passa também.

item = 46

Exemplo com N = 2

Buffer



N=2
mutex = 1 => 0
empty = 1
full = 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}

void consumer(void)
{
      int item;

      while (TRUE) {
            down(&full);
            down(&mutex);
            item = remove_item( );
            up(&mutex);
            up(&empty);
            consume_item(item);
      }
}
```

Fila de prontos => Na CPU

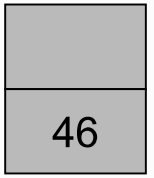
Na CPU => Fila de Prontos

Suponha que antes de consumir o item, que o escalonador dê a vez para o produtor.

item = 46

Exemplo com N = 2

Buffer



N=2
mutex = 0
empty = 1
full = 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}

void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU

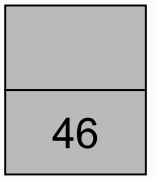
Fila de Prontos

item = 78

Produz um novo item. Ex: 78

Exemplo com N = 2

Buffer



N=2
mutex = 0
empty = 1
full = 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU

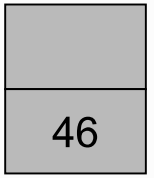
Fila de Prontos

Down sobre o semáforo empty.

item = 78

Exemplo com N = 2

Buffer



N=2
mutex = 0
empty = 1 => 0
full = 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}

void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU => Dormindo

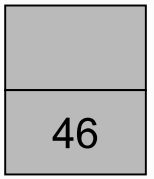
Fila de Prontos => Na CPU

item = 78

Down sobre mutex. Mas mutex vale 0. Então o produtor dorme. Proteção necessária para somente um alterar o buffer "ao mesmo tempo".

Exemplo com N = 2

Buffer



N=2
mutex = 0
empty = 0
full = 0

```
#define N   2                                   /* número de lugares no buffer */
typedef int semaphore;                          /* semáforos são um tipo especial de int */
semaphore mutex = 1;                            /* controla o acesso à região crítica */
semaphore empty = N;                            /* conta os lugares vazios no buffer */
semaphore full = 0 ;                            /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                            /* TRUE é a constante 1 */
            item = produce_item( );             /* gera algo para pôr no buffer */
            down(&empty);                       /* decresce o contador empty */
            down(&mutex);                       /* entra na região crítica */
            insert_item(item);                  /* põe novo item no buffer */
            up(&mutex);                         /* sai da região crítica */
            up(&full);                          /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                            /* laço infinito */
            down(&full);                        /* decresce o contador full */
            down(&mutex);                       /* entra na região crítica */
            item = remove_item( );              /* pega o item do buffer */
            up(&mutex);                         /* deixa a região crítica */
            up(&empty);                         /* incrementa o contador de lugares vazios */
            consume_item(item);                 /* faz algo com o item */
      }
}
```

Dormindo

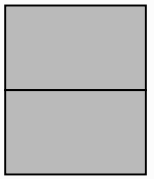
Na CPU

Agora sim o item é consumido.

Exemplo com N = 2

item = 78

Buffer



N=2
mutex = 0
empty = 0
full = 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Dormindo => Fila de Prontos

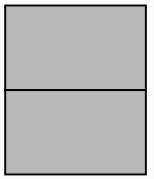
Na CPU

Up em mutex coloca o produtor na Fila de prontos.

Exemplo com N = 2

item = 78

Buffer



N=2
mutex = 0 => 1
empty = 0
full = 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Fila de Prontos

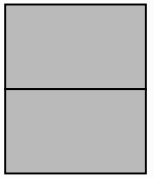
Na CPU

UP em empty.

item = 78

Exemplo com N = 2

Buffer



N=2
mutex = 1
empty = 0 => 1
full = 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

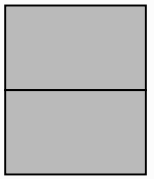
Fila de Prontos => Na CPU

Na CPU => Fila de Prontos

Exemplo com N = 2

item = 78

Buffer



N=2
mutex = 1
empty = 1
full = 0

Suponha uma troca feita pelo escalonador

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}

void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU

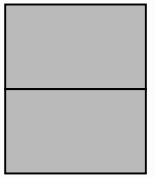
Fila de Prontos

Down sobre mutex.

item = 78

Exemplo com N = 2

Buffer



N=2
mutex = 1 => 0
empty = 1
full = 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}

void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU

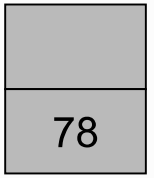
Fila de Prontos

item = 78

Agora, 78 é inserido no Buffer

Exemplo com N = 2

Buffer



N=2
mutex = 0
empty = 1
full = 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU

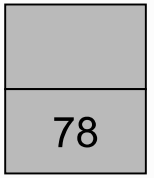
Fila de Prontos

UP mutex

item = 78

Exemplo com N = 2

Buffer



N=2
mutex = 0 => 1
empty = 1
full = 0

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU

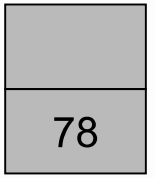
Fila de Prontos

UP full

item = 78

Exemplo com N = 2

Buffer



N=2
mutex = 1
empty = 1
full = 0 => 1

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU

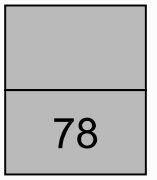
Fila de Prontos

item = 90

Produz novo item. Ex: 90

Exemplo com N = 2

Buffer



N=2
mutex = 1
empty = 1
full = 1

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU

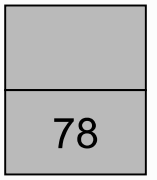
Fila de Prontos

Down empty

item = 90

Exemplo com N = 2

Buffer



N=2
mutex = 1
empty = 1 => 0
full = 1

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU

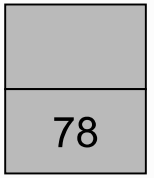
Fila de Prontos

Down mutex

item = 90

Exemplo com N = 2

Buffer



N=2
mutex = 1 => 0
empty = 0
full = 1

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}

void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU

Fila de Prontos

item = 90

Insere 90 no buffer

Exemplo com N = 2

Buffer

90
78

N=2
mutex = 0
empty = 0
full = 1

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}

void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU

Fila de Prontos

UP mutex

item = 90

Exemplo com N = 2

Buffer

90
78

N=2
mutex = 0 => 1
empty = 0
full = 1

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}

void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU

Fila de Prontos

UP full

item = 90

Exemplo com
N = 2

Buffer

90
78

N=2
mutex = 1
empty = 0
full = 1 => 2

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU

Fila de Prontos

item = 23

Produz novo item.
Ex: 23

Exemplo com
N = 2

Buffer

90
78

N=2
mutex = 1
empty = 0
full = 2

```
#define N   2                       /* número de lugares no buffer */
typedef int semaphore;              /* semáforos são um tipo especial de int */
semaphore mutex = 1;                /* controla o acesso à região crítica */
semaphore empty = N;                /* conta os lugares vazios no buffer */
semaphore full = 0 ;                /* conta os lugares preenchidos no buffer */

void producer(void)
{
      int item;

      while (TRUE) {                /* TRUE é a constante 1 */
            item = produce_item( );   /* gera algo para pôr no buffer */
            down(&empty);             /* decresce o contador empty */
            down(&mutex);             /* entra na região crítica */
            insert_item(item);        /* põe novo item no buffer */
            up(&mutex);               /* sai da região crítica */
            up(&full);                /* incrementa o contador de lugares preenchidos */
      }
}


void consumer(void)
{
      int item;

      while (TRUE) {                /* laço infinito */
            down(&full);              /* decresce o contador full */
            down(&mutex);             /* entra na região crítica */
            item = remove_item( );    /* pega o item do buffer */
            up(&mutex);               /* deixa a região crítica */
            up(&empty);               /* incrementa o contador de lugares vazios */
            consume_item(item);       /* faz algo com o item */
      }
}
```

Na CPU => Dormindo

Fila de Prontos => Na CPU

item = 23

Down empty. Então, o thread dorme, aguardando pelo menos 1 item ser consumido

Exemplo com N = 2

Buffer

90
78

N=2
mutex = 1
empty = 0
full = 2

Produtor-Consumidor

Assim, a produção e o consumo de itens segue sem que o sistema entre em *deadlock*.

É importante ressaltar que este algoritmo funciona para qualquer quantidade de produtores e consumidores.

A quantidade não precisa ser idêntica.

Justificativa para Monitores

Semáforos fornecem um mecanismo conveniente e eficaz para a sincronização de processos/threads.

Seu uso incorreto poderá resultar em erros de sincronismo difíceis de detectar.

Esses erros só acontecem se ocorrerem em determinadas sequências de execução, e essas sequência nem sempre ocorrem.

- O que dificulta validar se o algoritmo está correto ou não.

Justificativa para Monitores

A utilização de semáforos resolve TODOS os problemas de Comunicação Interprocessos.

- Não promove a espera ocupada;
- Atinge a exclusão mútua;
- A interrupção de *clock* fica desabilitada por um tempo muito pequeno (apenas dentro das funções de *down* e *up*).

A questão fundamental é: Serão simples as soluções para os diversos problemas utilizando semáforos?

- Programadores com experiência básica conseguirão descrever soluções eficazes com semáforos?

Justificativa para Monitores

Vimos as chamadas dos semáforos como:

- `meuSemaforo.down();`
- `meuSemaforo.up();`

Mas quando Dijkstra descreveu a técnica, ele utilizou as chamadas abaixo – usando sua língua nativa (holandês) como base.

- `meuSemaforo.p();` // P vem de *passeren* (passar)
- `meuSemaforo.v();` // V vem de *vrijgave* (liberar)

Justificativa para Monitores

Um problema comum é a utilização do seguinte código neste contexto:

Processo A

```
mutex.up();
alteraRegiaoCritica();
mutex.down();
```

Processo B

```
mutex.up();
alteraRegiaoCritica();
mutex.down();
```

Se os processos que concorrem a região crítica estiverem com este equivoco, a consistência de toda a área crítica é colocada xeque. O que acontece neste caso?

Todos os processos entram e saem da região critica. Sempre. A qualquer hora.

Monitores

Pensando em facilidade e corretude, Hoare (1974) e Hansen (1975) propuseram uma unidade de sincronização de alto nível nomeada **monitor**.

Monitor é a coleção de funções, variáveis, estrutura de dados, tudo isso agrupado em um módulo ou pacote. Pode ser visto como um TAD (Tipo Abstrato de Dados).

A ideia é ampliar o conceito do semáforo.

Os processos podem chamar os procedimentos em um monitor quando quiserem.

Mas não podem ter acesso direto às estruturas internas do monitor, a partir de procedimentos declarados fora do monitor.

- Conceito de encapsulamento, visto em Orientação à Objetos.

Monitores

Monitor é uma construção da linguagem de programação.

- Compiladores sabem que eles são especiais.

Por isso, tratam as chamadas aos procedimentos do monitor de forma diferente das outras chamadas de procedimento.

Quando um processo chama um procedimento do monitor, algumas das primeiras instruções verificarão se outro processo/thread está atualmente ativo dentro do monitor.

Se não existir nenhum processo concorrente no monitor, o acesso é liberado.

- Caso exista, o processo entra em estado bloqueado.

Monitores

É de total responsabilidade do compilador/interpretador implementar a exclusão mútua nas entradas do monitor.

A solução mais trivial implementada pelos compiladores é a utilização de um semáforo binário.

Como é o compilador, e não o programador que providencia a exclusão mútua, é muito menos provável que algo ocorra de errado.

Monitores

Em C e Pascal, os compiladores não implementam os monitores.

A solução então seria:

- Adicionar uma biblioteca que implementa as funções *up* e *down* (linguagem de montagem).
- Implementar um semáforo binário utilizando *up* e *down*.
- E a partir daí, controlar logicamente o monitor simulado a nível de usuário.

Monitores em Java

O uso de monitores em Java é uma variação do proposto por Hoare.

Na linguagem Java todo objeto possui um monitor associado.

Para facilitar o entendimento podemos encarar o monitor como um detentor de um "passe".

Toda *thread* pode pedir "emprestado" o passe ao monitor de um objeto antes de realizar alguma computação.

Como o monitor possui apenas um passe, apenas um *thread* pode adquirir o passe em um determinado instante.

Obviamente, o passe tem que ser devolvido para o monitor para possibilitar o empréstimo do passe a outro *thread*.

Monitores em Java

Nos resta saber como solicitar o passe ao monitor em Java:

- Isto é feito por meio da palavra chave *synchronized*.

Em java, existem duas formas de se usar a palavra chave synchronized:

- Na declaração de métodos e;
- No início de blocos.

Monitores em Java

```
class FilaCirc {
    private final int TAM = 10;
    private int vetInt[];
    private int inicio, total;

    public FilaCirc() {
        vetInt=new int[TAM];
        inicio=0;
        total =0;
    }
    public synchronized void addElement(int v) throws Exception {
        if (total == TAM) throw new Exception("Fila cheia!");
        vetInt[(inicio+total)%TAM] = v;
        total++;
    }
    public synchronized int getElement() throws Exception {
            if (total == 0 ) throw new Exception("Fila vazia!");
            int temp = vetInt[inicio];
            inicio = (++inicio)%TAM;
            total--;
            return temp;
    }
}
```

Monitores em Java

A palavra chave *synchronized* na frente dos métodos significa que o método será executado se puder adquirir o monitor do objeto a quem pertence o método.

- Caso contrário a thread que invocou o método será bloqueado até que possa adquirir o monitor.

Como a JVM identifica que ele já pode entrar no método?

- Diretamente ou Indiretamente (Sendo acordado por outro thread).

Monitores em Java

```
class FilaCirc {
    private final int TAM = 10;
    private int vetInt[];
    private int inicio, total;

    public FilaCirc() {
        vetInt=new int[TAM];
        inicio=0;
        total =0;
    }
    public void addElement(int v) throws Exception {
        synchronized(this) {
            if (total == TAM) throw new Exception("Fila cheia!");
            vetInt[(inicio+total)%TAM]=v;
            total++;
        }
    }
    public int getElement() throws Exception {
        synchronized(this) {
            if (total == 0 ) throw new Exception("Fila vazia!");
            int temp = vetInt[inicio];
            inicio = (++inicio)%TAM;
            total--;
        }
        return temp;
    }
}
```

Monitores em Java

A palavra chave synchronized limitando um bloco do código também solicita o passe daquele objeto passado como referência.

- Assim, pode-se atingir a exclusão mútua de forma semelhante ao exemplo anterior.

A vantagem/desvantagem com relação ao outro modelo é que não necessariamente as operações sobre a região crítica precisam estar encapsuladas em funções coesas.

- Você pode utilizar apenas um "pedaço" da função com o código sincronizado.

Monitores em Java

De certa forma este segundo método permite a "desorganização" do conceito de monitor.

Um método da classe A pode requisitar o passe da classe B.

- Pode ser interessante.
- Mas sai um pouco da definição purista que aproxima os monitores dos TADs.
- Deixa o sistema menos coeso, mas fornece mais possibilidades ao programador... inclusive de errar.