Sistemas Operacionais – Ciência da Computação

Gerência de Processos

Comunicação Interprocessos
(*Sleep*, *Weakup* e Semáforos)

Prof. Humberto Brandão
humberto@dcc.ufmg.br

aula disponível no site:
http://www.dcc.ufmg.br/~humberto/unifal/

Universidade Federal de Alfenas
Departamento de Ciências Exatas
versão da aula: 0.3

Conceitos vistos na aula anterior...

- Condição de disputa;
- Região crítica;
- Exclusão mútua;

Soluções vistas na aula anterior

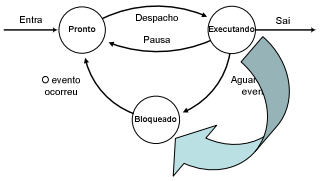
- Desabilitando a interrupção de *clock*;
- Variáveis de impedimento (de bloqueio):
  - Não garante todos os casos.
- Solução de Peterson:
  - Funciona, mas gasta tempo do processador com um loop que verifica se o processo "bloqueado" pode entrar na região crítica;
  - Ou seja, o processador perde tempo efetuando esperas desnecessárias.

Dormir e Acordar
"Sleep and Wakeup"

A partir de agora veremos soluções sem a espera ociosa...

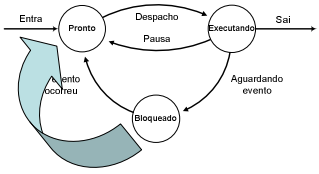
Sleep and wakeup (Conceitos)

- *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 and 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.



Problema Clássico

Produtor Consumidor

Um Problema Clássico:
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*);
- De forma equivalente, quando o consumidor quer retirar um item, mas o buffer está vazio, podemos colocar o processo consumidor para dormir (*sleep*);

Um Problema Clássico:
Produtor-Consumidor

- 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 ítem do buffer, este pode disparar uma chamada de sistema tentando acordar algum produtor que esteja dormindo.

Um Problema Clássico:
Produtor-Consumidor utilizando *sleep/wakeup*

```
#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 */
     }
}
```

Um Problema Clássico:
Produtor-Consumidor utilizando *sleep/wakeup*

```
#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

Um Problema Clássico:
Produtor-Consumidor utilizando *sleep/wakeup*

```
#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? */
     }
}
```

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

Um Problema Clássico:
Produtor-Consumidor utilizando *sleep/wakeup*

```
#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 buffe
     }
}

void consumer(void)
{
     int item;

     while (TRUE) {                             /* repita para sempre */
          if (c          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 */
     }
}
```

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

Processo Ativo

Um Problema Clássico:
Produtor-Consumidor utilizando *sleep/wakeup*

```
#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 (c          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

Um Problema Clássico:
Produtor-Consumidor utilizando *sleep/wakeup*

```
#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 (c          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

Um Problema Clássico:
Produtor-Consumidor utilizando *sleep/wakeup*

```
#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 (c          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

Um Problema Clássico:
Produtor-Consumidor utilizando *sleep/wakeup*

```
#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 (c          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

Um Problema Clássico:
Produtor-Consumidor utilizando *sleep/wakeup*

```
#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 (c          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

Um Problema Clássico:
Produtor-Consumidor utilizando *sleep/wakeup*

```
#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 (c          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

Um Problema Clássico:
Produtor-Consumidor utilizando *sleep/wakeup*

```
#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...

Um Problema Clássico:
Produtor-Consumidor utilizando *sleep/wakeup*

```
#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...

Um Problema Clássico:
Produtor-Consumidor utilizando *sleep/wakeup*

```
#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

Um Problema Clássico:
Produtor-Consumidor utilizando *sleep/wakeup*

```
#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)

Um Problema Clássico:
Produtor-Consumidor utilizando *sleep/wakeup*

```
#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...

Um Problema Clássico:
Produtor-Consumidor utilizando *sleep/wakeup*

```
#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 Bloqueado

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

Semáforos

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 nao tem acesso ao semáforo... a thread aguarda...
      while( s.contador == 0 ){
         this.wait();
      }
      //quando a thread tem acesso,
      //o semaforo eh decrementado em uma unidade...
      s.contador--;
   }
}
```

Semáforos
Operação *down*

```
public synchronized void down( Semaforo s ) throws Exception{
   //garante a atomicidade da execução do método...
   synchronized( this ){
      //enquanto nao tem acesso ao semáforo... a thread aguarda...
      while( s.contador == 0 ){
         this.wait();
      }
      //quando a thread tem acesso,
      //o semaforo eh decrementado em uma unidade...
      s.contador--;
   }
}
```

Esta função deve ser indivisível.
Atômica

Semáforos
Operação *down*

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

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

Semáforos
Operação *down*

```
public synchronized void down(
   //garante a atomicidade
   synchronized( this ){
      //enquanto não tem acesso ao
      while( s.contador == 0
         this.wait();
      }
      //quando a thread tem ace
      //o semáforo é decrementa
      s.contador--;
   }
}
```

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 *up*

Esta função deve ser indivisível

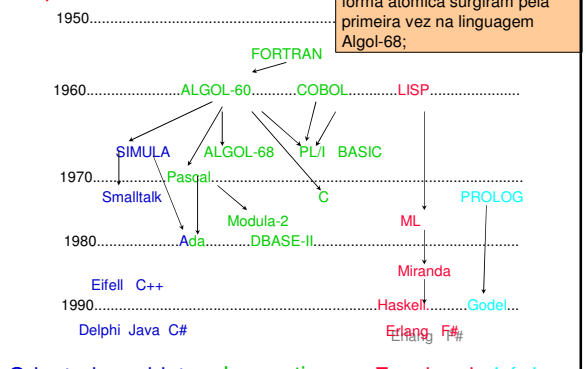
```
public synchronized void up( Semaforo num ){
    //garante a atomicidade da execução do método...
    synchronized(this){
        //quando a thread libera o semáforo,
        //ele é incrementado em uma unidade...
        //isso possibilita que outra thread
        //(que está dormindo) possa acessar
        //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();
    }
}
```

Criando...

- Um acesso a um arquivo compartilhado por dois processos...
- Vamos pensar...

Up and Down

Estes mecanismos de atualização de variáveis de forma atômica surgiram pela primeira vez na linguagem Algol-68;



Orientado a objetos Imperativo Funcional Lógico

Solução do Problema
Produtor-Consumidor Utilizando Semáforos

- A solução utiliza 3 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;

Solução do Problema
Produtor-Consumidor Utilizando Semáforos

```
#define N 100                       /* 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 */
    }
}
```

Referencia

- Sistemas Operacionais Modernos. Tanenbaum, A. S. 2ª edição. 2003