Departamento de Computação

SISTEMAS OPERACIONAIS 1
21270 A

Departamento de Computação
Prof. Kelen Cristiane Teixeira Vivaldini

Apresentação baseada nos slides
do Prof. Dr. Xxxxxxx
MC514–Sistemas Operacionais: Teoria e Prática

Problema do Produtor/Consumidor

Problema do Produtor/Consumidor

- Dois processos compartilham um *buffer* de tamanho fixo. O processo produtor coloca dados no *buffer*
- O processo consumidor retira dados do *buffer*;
- Problemas:
  - Produtor deseja colocar dados quando o *buffer* ainda está cheio;
  - Consumidor deseja retirar dados quando o *buffer* está vazio;
  - Solução: colocar os processos para "dormir", até que eles possam ser executados;

Comunicação entre Processos
Sincronização Produtor-Consumidor

Para os casos extremos de ocupação do buffer (cheio/vazio), deverão funcionar as seguintes regras de sincronização:

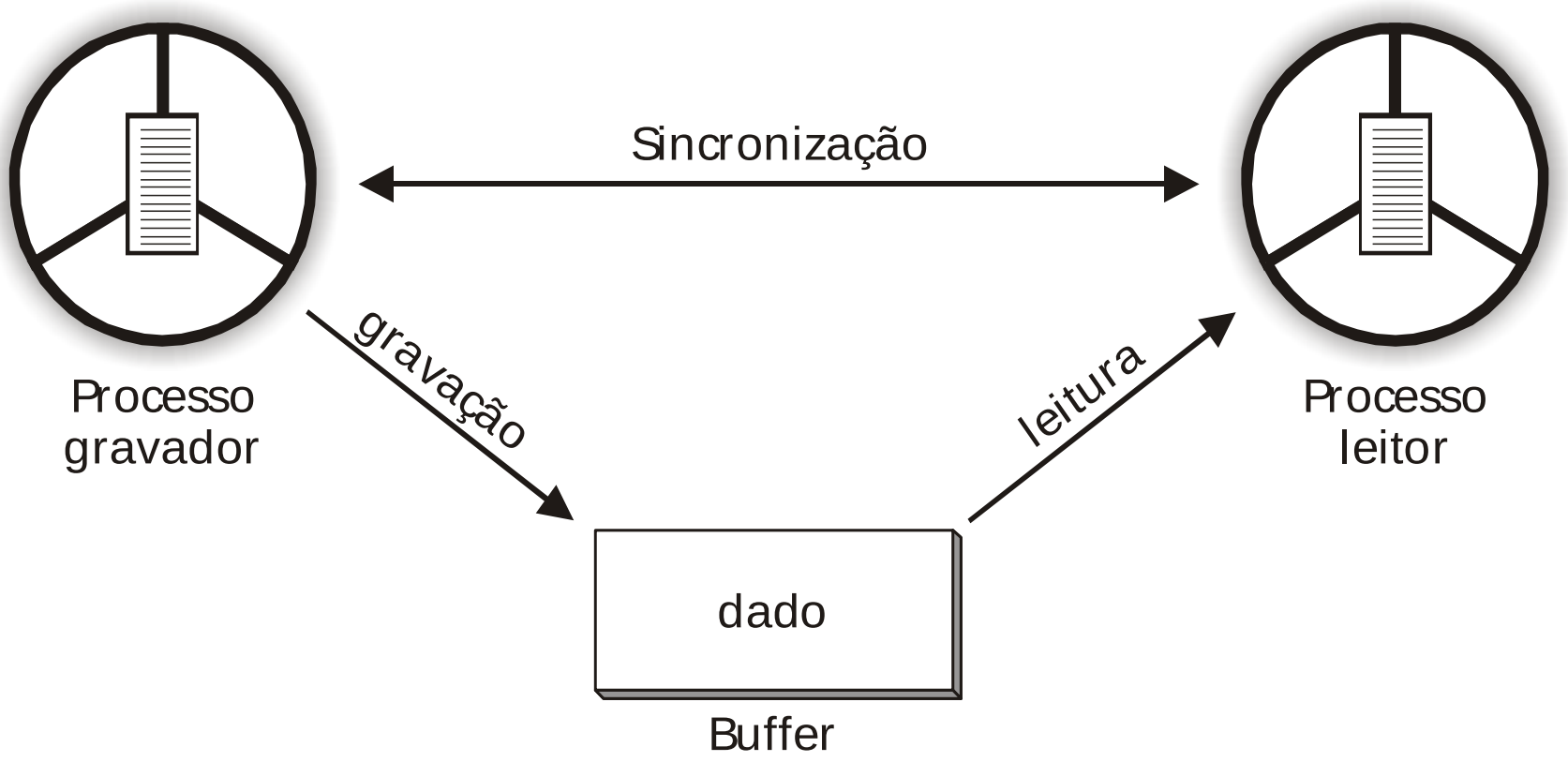
- se o produtor tentar depositar uma mensagem no buffer cheio, ele será suspenso até que o consumidor retire pelo menos uma mensagem do buffer;
- se o consumidor tenta retirar uma mensagem do buffer vazio, ele será suspenso até que o produtor deposite pelo menos uma mensagem no buffer.

Departamento de Computação

Problema do Produtor/Consumidor

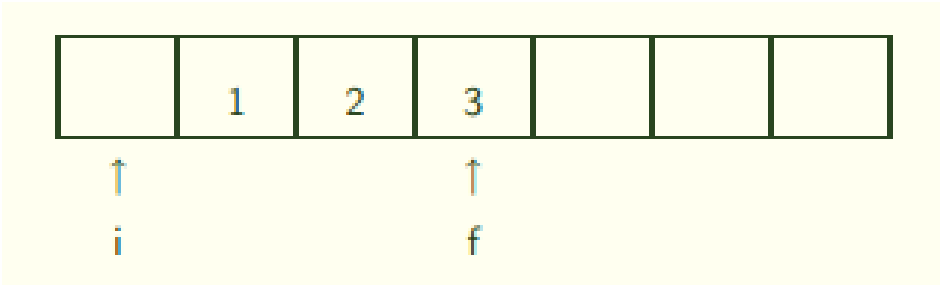
- *Buffer*: uma variável `count` controla a quantidade de dados presente no *buffer*.
- Produtor: Antes de colocar dados no *buffer*, o processo produtor checa o valor dessa variável. Se a variável está com valor máximo, o processo produtor é colocado para dormir. Caso contrário, o produtor coloca dados no *buffer* e o incrementa.
- Consumidor: Antes de retirar dados no *buffer*, o processo consumidor checa o valor da variável `count` para saber se ela está com 0 (zero). Se está, o processo vai “dormir”, senão ele retira os dados do *buffer* e decrementa a variável;

Problema do Produtor/Consumidor



Problema do Produtor/Consumidor

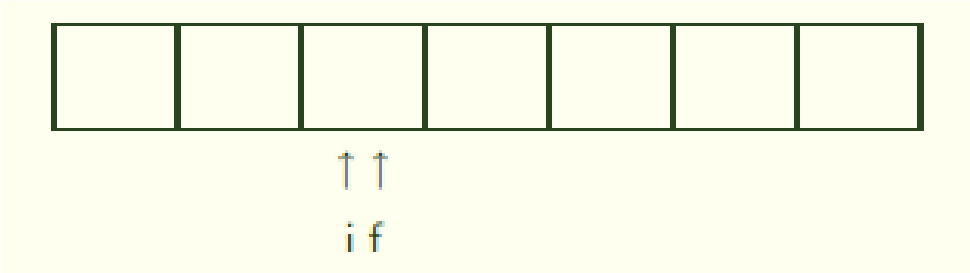
Controle do Buffer



- i: aponta para a posição anterior ao primeiro elemento
- f: aponta para o último elemento
- c: indica o número de elementos presentes
- N: indica o número máximo de elementos

Problema do Produtor/Consumidor

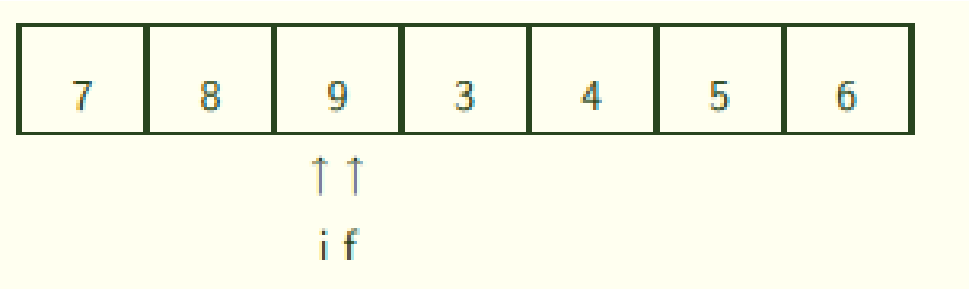
Buffer vazio



- i == f
- c == 0

Problema do Produtor/Consumidor

Buffer cheio



- i == f
- c == N

Departamento de Computação

Problema do Produtor/Consumidor

Comportamento básico

```
int buffer[N];
int c = 0;
int i = 0, f = 0;
```

Produtor

```
while (true)
f = (f+1)%N;
buffer[f]= produz();
c++;
```

Consumidor

```
while (true)
i = (i+1)%N;
consome(buffer [i]);
c--;
```

Veja código: prod-cons-basico.c

Problema do Produtor/Consumidor

Problemas

1. produtor insere em posição que ainda não foi consumida
2. consumidor remove de posição já foi consumida

Problema do Produtor/Consumidor

Algoritmo com espera ocupada

```
int buffer[N];
int c = 0;
int i = 0, f = 0;
```

Produtor

```
while (true)
        while (c == N);
        f = (f+1)%N;
        buffer[f]= produz();
        c++;
```

Consumidor

```
while (true)
        while (c == 0);
        i = (i+1)%N;
        consome(buffer[i]);
        c--;
```

Veja código: prod-cons-basico-busy-wait.c

Link : http://cs.uttyler.edu/Faculty/Rainwater/COSC3355/Animations/processsync.htm

Departamento de Computação

Problema do Produtor/Consumidor

Condição de disputa

Produtor	Consumidor
c++;	c--;
mov rp,c	mov rc,c
inc rp	dec rc
mov c,rp	mov c,rc

- Decremento/incremento não são atômicos
- Veja código: prod-cons-basico-race.c

Problema do Produtor/Consumidor

Operações atômicas

- Veja info gcc - C extensions - Atomic builtins
- Veja o código prod-cons-basico-atomic-inc.c

Problema do Produtor/Consumidor

Possibilidade de Lost Wake-Up

```
int buffer[N];
int c = 0;
int i = 0, f = 0;
```

Produtor

```
while (true)
   if (c == N) sleep();
    f = (f + 1);
    buffer[f]= produz();
    atomic inc(c);
    if (c == 1)
            wakeup consumidor();
```

Consumidor

```
        while (true)
                if (c == 0) sleep();
                i = (i+1);
                consome(buffer [i]);
                atomic dec(c);
                if (c == N - 1)
                wakeup produtor();
```

- Produtor envia sinal antes de o consumidor ir dormir
- Consumidor envia sinal antes de o produtor ir dormir

Problema do Produtor/Consumidor

Possibilidade de Lost Wake-Up

```
int buffer[N];
int c = 0;
int i = 0, f = 0;
```

Produtor

```
while (true)
   if (c == N) sleep();
    f = (f + 1);
    buffer[f]= produz();
    atomic inc(c);
    if (c == 1)
            wakeup consumidor();
```

Consumidor

```
        while (true)
                if (c == 0) sleep();
                i = (i+1);
                consome(buffer [i]);
                atomic dec(c);
                if (c == N - 1)
                wakeup produtor();
```

- Consumidor consome único item e vai dormir antes do produtor ter enviado sinal
- Cenário simétrico para o produtor?

Problema do Produtor/Consumidor

Futex e operações atômicas

- Veja o código prod-cons-basico-futex.c
- O algoritmo não é tão simples para vários produtores e vários consumidores

Problema do Produtor/Consumidor

Semáforos

- Semáforos são contadores especiais para recursos compartilhados.
- Proposto por Dijkstra (1965)
- Operações básicas (atômicas):
  - decremento (*down*, *wait* ou P)

    bloqueia se o contador for nulo
  - incremento (*up*, *signal* (*post*) ou V)

    nunca bloqueia

Problema do Produtor/Consumidor

Semáforos - Comportamento básico

- sem init(s, 5)
- wait(s)

```
        if (s == 0)
                bloqueia_processo();
        else s--;
```

- signal(s)

```
if (s == 0 && existe processo bloqueado)
        acorda_processo();
else s++;
```

Problema do Produtor/Consumidor

Produtor-Consumidor com Semáforos

```
semaforo cheio = 0;
semaforo vazio = N;
```

Produtor:

```
while (true)
        wait(vazio);
        f = (f+1)%N;
        buffer[f] = produz();
        signal(cheio);
```

Consumidor:

```
while (true)
        wait(cheio);
        i = (i+1)%N;
        consome(buffer[i]);
        signal(vazio);
```

Veja código: prod-cons-sem.c

Problema do Produtor/Consumidor

Vários produtores e consumidores

semaforo cheio = 0, vazio = N;
semaforo lock_prod = 1, lock_cons = 1;

Produtor:

```
while (true)
        wait(vazio);
        wait(lock_prod);
        f = (f+1)%N;
        buffer[f] = produz();
        signal(lock_prod);
        signal(cheio);
```

Consumidor:

```
while (true)
        wait(cheio);
wait(lock_cons);
        i = (i+1)%N;
        consome(buffer[i]);
        signal(lock_cons);
        signal(vazio);
```

Veja código: mult-prod-cons-sem .c

Problema do Produtor/Consumidor

Vários produtores e consumidores

```
semaforo cheio = 0, vazio = N;
semaforo lock_prod = 1, lock_cons = 1;
```

Produtor:

```
while (true)
        item = produz();
        wait(vazio);
        wait(lock_prod);
        f = (f + 1) % N;
        buffer[f] = item;
        signal(lock_prod);
        signal(cheio);
```

Consumidor:

```
while (true)

        wait(cheio);
        wait(lock_cons);
         i = (i + 1) % N;
         item = buffer[i];
        signal(lock_cons);
        signal(vazio);
        consome(item);
```

Problema do Produtor/Consumidor

Semáforos

- Exclusão mútua
- Sincronização

Problema do Produtor/Consumidor

Mutex locks

⇒ Exclusão mútua

- pthread mutex lock
- pthread mutex unlock

Problema do Produtor/Consumidor

Variáveis de condição

- ⇒ Sincronização
- pthread cond wait
- pthread cond signal
- pthread cond broadcast
- precisam ser utilizadas em conjunto com mutex locks

Problema do Produtor/Consumidor

Thread 0 acorda Thread 1

```
int s;
```

Thread 1:

```
mutex_lock(&mutex);
if (preciso_esperar(s))
        cond_wait(&cond, &mutex);
mutex_unlock(&mutex);
```

Thread 0:

```
mutex_lock(&mutex);
if (devo_acordar_thread_1(s))
        cond_signal(&cond);
mutex_unlock(&mutex);
```

Veja código: cond_signal.c

Problema do Produtor/Consumidor

Produtor-Consumidor

```
int c = 0; /* Contador de posições ocupadas */
mutex_t lock_c; /* lock para o contador */


cond_t pos_vazia; /* Para o produtor esperar */
cond_t pos_ocupada; /* Para o consumidor esperar */
```

Problema do Produtor/Consumidor

Produtor-Consumidor

```
int f = 0;
Produtor:
        mutex_lock(&lock_c);
        if (c == N)
                cond_wait(&pos_vazia, &lock_c);
        f = (f+1)%N;
        buffer[f] = produz();
        c++;
        if (c == 1)
                cond_signal(&pos_ocupada);
        mutex_unlock(&lock_c)
```

Problema do Produtor/Consumidor

Produtor-Consumidor

int i = 0;

Consumidor:

```
mutex_lock(&lock_c);
if (c == 0)
        cond_wait(&pos_ocupada, &lock_c);
f = (i+1)%N;
consome(buffer[i]);
if (c == N-1)
        cond_signal(&pos_vazia);
c--;
mutex_unlock(&lock_c);
```

Problema do Produtor/Consumidor

Produtor-Consumidor

Consumidor:

```
mutex_lock(&lock_o);
if (no == 0) cond_wait(&pos_ocupada, &lock_o);
no--;
mutex_unlock(&lock_o);
i = (i+1)%N;
consome(buffer[i]);
mutex_lock(&lock_v);
nv++;
cond_signal(&pos_vazia);
mutex_unlock(&lock_v);
```

Problema do Produtor/Consumidor

Thread 0 acorda alguma thread

```
int s;
```

Thread i:

```
mutex_lock(&mutex);
if (preciso_esperar(s))
        cond_wait(&cond, &mutex);
mutex_unlock(&mutex);
```

Thread 0:

```
mutex_lock(&mutex);
if (devo_acordar_alguma_thread(s))
        cond_signal(&mutex);
mutex_unlock(&mutex);
```

Veja código: cond_signal_n.c

Departamento de Computação

Problema do Produtor/Consumidor

The pthread cond signal() function shall unblock at least one of the threads that are blocked on the specified condition variable cond (if any threads are blocked on cond).

Multiple Awakenings by Condition Signal

On a multi-processor, it may be impossible for an implementation of pthread cond signal() to avoid the unblocking of more than one thread blocked on a condition variable. For example, consider the following partial implementation of pthread cond wait() and pthread cond signal(), executed by two threads in the order given. One thread is trying to wait on the condition variable, another is concurrently executing pthread cond signal(), while a third thread is already waiting.

Problema do Produtor/Consumidor

```
pthread_cond_wait(mutex, cond):
  value = cond->value; /* 1 */
  pthread_mutex_unlock(mutex); /* 2 */
  pthread_mutex_lock(cond->mutex); /* 10 */
  if (value == cond->value) { /* 11 */
    me->next_cond = cond->waiter;
    cond->waiter = me;
    pthread_mutex_unlock(cond->mutex);
    unable_to_run(me);
  } else
     pthread_mutex_unlock(cond->mutex); /* 12 */
  pthread_mutex_lock(mutex); /* 13 */

pthread_cond_signal(cond):
  pthread_mutex_lock(cond->mutex); /* 3 */
  cond->value++; /* 4 */
  if (cond->waiter) { /* 5 */
    sleeper = cond->waiter; /* 6 */
    cond->waiter = sleeper->next_cond; /* 7 */
    able_to_run(sleeper); /* 8 */
  }
  pthread_mutex_unlock(cond->mutex); /* 9 */
```

Problema do Produtor/Consumidor

```
pthread_cond_wait(mutex, cond):
  pthread_mutex_lock(cond->mutex); /* <=== Pega este lock primeiro */
  value = cond->value;
  pthread_mutex_unlock(mutex);
  if (value == cond->value) {
    me->next_cond = cond->waiter;
    cond->waiter = me;
    pthread_mutex_unlock(cond->mutex);
    unable_to_run(me);
  } else
     pthread_mutex_unlock(cond->mutex);
  pthread_mutex_lock(mutex);

pthread_cond_signal(cond):
  pthread_mutex_lock(cond->mutex);
  cond->value++;
  if (cond->waiter) {
    sleeper = cond->waiter;
    cond->waiter = sleeper->next_cond;
    able_to_run(sleeper);
  }
  pthread_mutex_unlock(cond->mutex);
```

Problema do Produtor/Consumidor

Produtores-Consumidores

Será que funciona?

```
cond_t pos_vazia, pos_ocupada;
mutex_t lock_v, lock_o;
int nv = N, no = 0;

mutex_t lock_i, lock_f;
int i = 0, f = 0;
```

Departamento de Computação

Problema do Produtor/Consumidor

Produtor-Consumidor

```
cond_t pos_vazia, pos_ocupada; mutex_t lock_v, lock_o;
int i = 0, f = 0, nv = N, no = 0;
```

Produtor:

```
        mutex_lock(&lock_v);
        if (nv == 0) cond_wait(&pos_vazia, &lock_v);
        nv--;
        mutex_unlock(&lock_v);
        f = (f+1)%N;
        buffer[f] = produz();
        mutex_lock(&lock_o);
        no++;
        cond_signal(&pos_ocupada);
        mutex_unlock(&lock_o);
```

Problema do Produtor/Consumidor

Produtor:

```
item = produz();
mutex_lock(&lock_v);
if (nv == 0) cond_wait(&pos_vazia, &lock_v);
nv--;
mutex_unlock(&lock_v);
mutex_lock(&lock_f);
f = (f+1)%N;
buffer[f] = item;
mutex_unlock(&lock_f);
mutex_lock(&lock_o);
no++;
cond_signal(&pos_ocupada);
mutex_unlock(&lock_o);
```

Problema do Produtor/Consumidor

Consumidor:

```
mutex_lock(&lock_o);
if (no == 0) cond_wait(&pos_ocupada, &lock_o);
no--;
mutex_unlock(&lock_o);
mutex_lock(&lock_i);
i = (i+1)%N;
item = buffer[i];
mutex_unlock(&lock_i);
mutex_lock(&lock_v);
nv++;
cond_signal(&pos_vazia);
mutex_unlock(&lock_v);
consome(item);
```

Problema do Produtor/Consumidor

Resumo

- Cooperação usando variáveis compartilhadas.
- Necessita operações atômicas por causa de condições de corrida;
- Implementação através de exclusão mútua:
- Comportamento difícil de prever;
- Tem que ser usado com critério;
- Tem que ser correto, justo, eficiente;