MC504 - Sistemas Operacionais

Produtores e Consumidores

Islene Calciolari Garcia

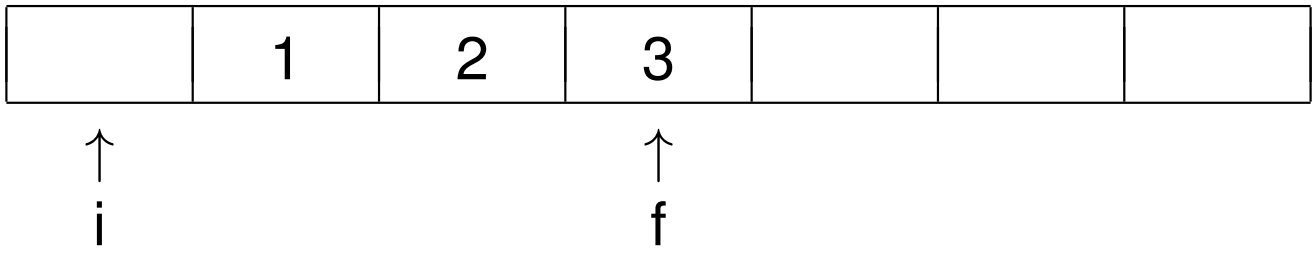
Primeiro Semestre de 2017

Sumário

Problema do Produtor-Consumidor

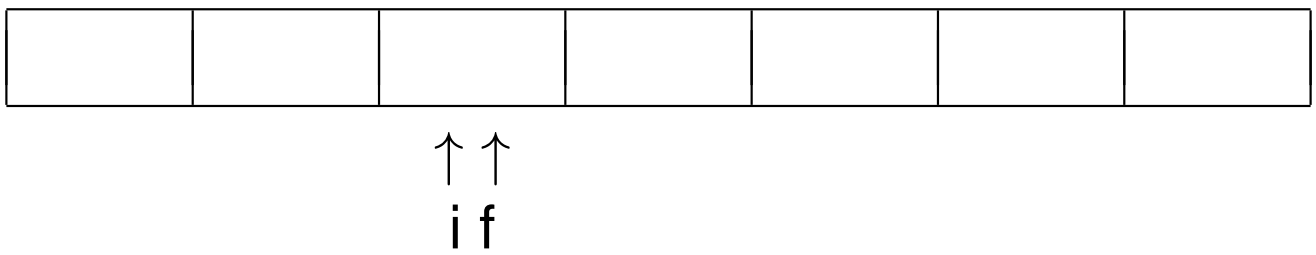
- Dois processos compartilham um *buffer* de tamanho fixo
- O produtor insere informação no *buffer*
- O consumidor remove informação do *buffer*

Controle do buffer



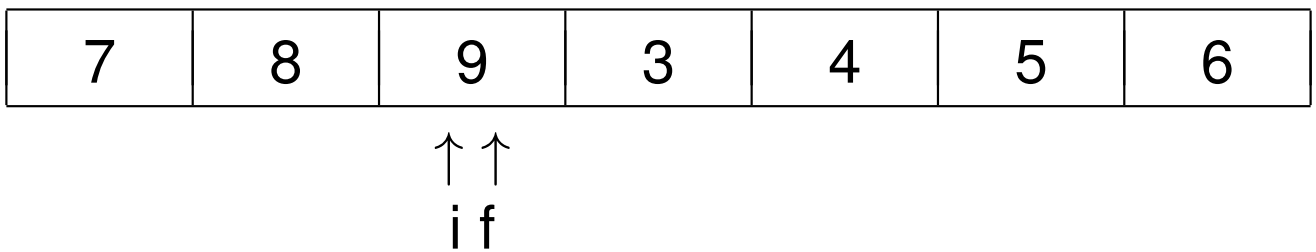
- **i:** aponta para a posição anterior ao primeiro elemento
- **f:** aponta para o último elemento
- **c:** indica o número de elementos presentes
- **N:** indica o número máximo de elementos

Buffer vazio



- **i == f**
- **c == 0**

Buffer cheio



- **i == f**
- **c == N**

Saída esperada

```
Produtor, item = 0.
Consumidor, item = 0.
Produtor, item = 1.
Produtor, item = 2.
Consumidor, item = 1.
Produtor, item = 3.
Consumidor, item = 2.
Consumidor, item = 3.
Produtor, item = 4.
Consumidor, item = 4.
Produtor, item = 5.
Consumidor, item = 5.
```

Implementação sem sincronização

```
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-sem-sinc.c`

Bug

Consumidor tenta consumir antes da produção

```
Produtor, item = 0.
Consumidor, item = 0.
Consumidor, item = -1.
Erro:  item foi consumido antes de ser produzido.
Produtor, item = 1.
Consumidor, item = -1.
Erro:  item foi consumido antes de ser produzido.
Produtor, item = 2.
Consumidor, item = -1.
Erro:  item foi consumido antes de ser produzido.
Produtor, item = 3.
Produtor, item = 4.
```

Bug

Produtor sobrescreve posição que *ainda* não foi consumida

```
Produtor, item = 0.
Produtor, item = 1.
Produtor, item = 2.
Produtor, item = 3.
Produtor, item = 4.
Produtor, item = 5.  (Suponha buffer de 5 posições)
Consumidor, item = 5.
Erro:  item foi produzido antes de a posição estar livre.
```

Algoritmo com espera ocupada

```
int buffer[N];
int c = 0, 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-busy-wait.c`

Possibilidade de inconsistência

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 a animação: processsync
- Veja o código: `prod-cons-race.c`

Operações atômicas

- Veja info gcc - C extensions - Atomic builtins
- Veja o código `prod-cons-atomic-inc.c`
- Exercícios
  - incremento atômico com máximo (questão 2, prova 1 2s2016)
  - incremento atômico módulo N
  - decremento atômico se positivo

Possibilidade de Lost Wake-Up

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

Produtor

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

Consumidor

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

Possibilidade de Lost Wake-Up

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

Produtor

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

Consumidor

```
while (true)
  if (c == 0) sleep();
  i = (i+1)%N;
  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

Outra possibilidade de inconsistência

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

Produtor

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

Consumidor

```
while (true)
  if (c == 0) sleep();
  i = (i+1)%N;
  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?

Implementação com futexes

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

Produtor

```
while (true)
  while (c == N)
    futex_wait(&c, N);
  f = (f + 1)%N;
  buffer[f]= produz();
  atomic_inc(c);
  if (c == 1)
    futex_wake(&c,1);
```

Consumidor

```
while (true)
  while (c == 0)
    futex_wait(&c, 0);
  i = (i+1)%N;
  consome(buffer [i]);
  atomic_dec(c);
  if (c == N - 1)
    futex_wake(&c,1);
```

- Veja o código prod-cons-futex.c

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

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 o código mult-prod-cons-sem.c

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);
```