Leitores e Escritores

Sistemas Operacionais

Prof. Pedro Ramos
pramos.costar@gmail.com

Pontifícia Universidade Católica de Minas Gerais
ICEI - Departamento de Ciência da Computação

HOJE - SINCRONIZAÇÃO PARA O PROBLEMA LEITORES E ESCRITORES

Um objeto é compartilhado entre várias threads de duas classes:
- Leitores: leem os dados, nunca os modificam.
- Escritores: leem os dados e os modificam.

- Usar 1 único lock nos dados compartilhados é **muito restritivo**

⇒ *Queremos vários leitores lendo os dados (pode ser um objeto, ou um array, qualquer coisa) ao mesmo tempo*

  - Permitir apenas um escritor em qualquer momento.
  - ***Como controlar o acesso aos dados para permitir isso?***

HOJE - SINCRONIZAÇÃO PARA O PROBLEMA LEITORES E ESCRITORES

Um objeto é compartilhado entre várias threads de duas classes:
- Leitores: leem os dados, nunca os modificam.
- Escritores: leem os dados e os modificam.

- Usar 1 único lock nos dados compartilhados é **muito restritivo**

⇒ *Queremos vários leitores lendo os dados (pode ser um objeto, ou um array, qualquer coisa) ao mesmo tempo*

  - Permitir apenas um escritor em qualquer momento.
  - ***Como controlar o acesso aos dados para permitir isso?***
- **Critérios de corretude:**
  - Cada leitura ou escrita dos dados compartilhados deve ocorrer dentro de uma seção crítica.
  - Garantir exclusão mútua para os escritores.
  - Permitir que vários leitores executem na seção crítica ao mesmo tempo.

PROBLEMA LEITORES E ESCRITORES (READER/WRITER)

```
class ReadWrite {
 public:
   void Read(); // ler
   void Write(); // escrever
 private:
   int leitores; // contador de leitores
   Semaforo mutex; // controla o acesso aos leitores
   Semaforo wrt; // controla a entrada do 1º escritor ou leitor
}

// construtor
ReadWrite::ReadWrite {
   leitores = 0;
   mutex->valor = 1;
   wrt->valor = 1;
}
```

PROBLEMA LEITORES E ESCRITORES (READER/WRITER)

```
ReadWrite::Write(){
   wrt.Wait(); // há escritores ou leitores? (se < 0, bloqueia)
   <realizar escrita>
   wrt.Signal(); // permitir outros
}
```

PROBLEMA LEITORES E ESCRITORES (READER/WRITER)

```
ReadWrite::Write(){
    wrt.Wait(); // há escritores ou leitores? (se < 0, bloqueia)
    <realizar escrita>
    wrt.Signal(); // permitir outros
}
```

"WRITE()" (Escrever) É SIMPLES: USA UM SEMÁFORO GLOBAL

PROBLEMA LEITORES E ESCRITORES (READER/WRITER)

```
ReadWrite::Write(){
   wrt.Wait(); // há escritores ou leitores? (se < 0, bloqueia)
   <realizar escrita>
   wrt.Signal(); // permitir outros
}

ReadWrite::Read(){
   mutex.Wait(); // garantir exclusão mútua
   leitores += 1; // mais um leitor
   if (leitores == 1)
      wrt.Wait(); // bloquear escritores
   mutex.Signal();
   <realizar leitura>
   mutex.Wait(); // garantir exclusão mútua
   leitores -= 1; // leitor finalizado
   if (leitores == 0)
      wrt.Signal(); // permitir escritores
   mutex.Signal();
}
```

PROBLEMA LEITORES E ESCRITORES (READER/WRITER)

```
ReadWrite::Write(){
    wrt.Wait(); // há escritores ou leitores? (se < 0, bloqueia)
    <realizar escrita>
    wrt.Signal(); // permitir outros
}

ReadWrite::Read(){
    mutex.Wait(); // garantir exclusão mútua
    leitores += 1; // mais um leitor
    if (leitores == 1)
        wrt.Wait(); // bloquear escritores
    mutex.Signal();
    <realizar leitura>
    mutex.Wait(); // garantir exclusão mútua
    leitores -= 1; // leitor finalizado
    if (leitores == 0)
        wrt.Signal(); // permitir escritores
    mutex.Signal();
}
```

QUEREMOS MÚLTIPLOS LEITORES AO MESMO TEMPO!

"mutex" (UM SEMÁFORO BINÁRIO == LOCK) É USADO PARA GARANTIR SOMENTE O ACESSO ATÔMICO AO CONTADOR LEITORES

A SEÇÃO CRÍTICA PODE SER REALIZADA POR VÁRIOS LEITORES AO MESMO TEMPO.

PROBLEMA LEITORES E ESCRITORES (READER/WRITER)

```
ReadWrite::Write(){
   wrt.Wait(); // há escritores ou leitores? (se < 0, bloqueia)
   <realizar escrita>
   wrt.Signal(); // permitir outros
}

ReadWrite::Read(){
   mutex.Wait(); // garantir exclusão mútua
   leitores += 1; // mais um leitor
   if (leitores == 1)
      wrt.Wait(); // bloquear escritores
   mutex.Signal();
   <realizar leitura>
   mutex.Wait(); // garantir exclusão mútua
   leitores -= 1; // leitor finalizado
   if (leitores == 0)
      wrt.Signal(); // permitir escritores
   mutex.Signal();
}
```

POR QUE == 1 E NÃO >= 1 ?

SE APÓS ENTRAR NO IF, UM ESCRITOR (*writer*) CHAMAR wrt.Wait(), O LEITOR (*reader*) FICA BLOQUEADO.

A PARTIR DAÍ, PRÓXIMOS LEITORES (readers) QUE CHEGAREM FICARÃO BLOQUEADOS NO MUTEX ACIMA.

PROBLEMA LEITORES E ESCRITORES CENÁRIO 1

LEITOR_1
Read()

LEITOR_2

Read()

ESCRITOR_1

Write()

PROBLEMA LEITORES E ESCRITORES CENÁRIO 2

LEITOR_1

Read()

LEITOR_2

Read()

ESCRITOR_1

Write()

PROBLEMA LEITORES E ESCRITORES CENÁRIO 3

LEITOR_1	LEITOR_2	ESCRITOR_1
Read()		
	Read()	
		Write()

PROBLEMA LEITORES E ESCRITORES - DISCUSSÃO DA SOLUÇÃO

Notas de Implementação

- **O primeiro leitor bloqueia se houver um escritor ativo:** Quando o primeiro leitor tenta entrar na seção crítica e já há um escritor ativo, ele é bloqueado. Qualquer outro leitor que tentar entrar enquanto isso ficará bloqueado no mutex (que controla o acesso à contagem de leitores).

PROBLEMA LEITORES E ESCRITORES - DISCUSSÃO DA SOLUÇÃO

Notas de Implementação

- O primeiro leitor bloqueia se houver um escritor ativo; qualquer outro leitor que tente entrar é bloqueado no mutex.

- **O último leitor a sair sinaliza para o escritor que está esperando**: Quando o **último leitor termina sua leitura e sai da seção crítica**, ele **sinaliza** para um **escritor** que pode estar **esperando** para escrever, permitindo que ele entre na seção crítica.

PROBLEMA LEITORES E ESCRITORES - DISCUSSÃO DA SOLUÇÃO

Notas de Implementação

- O primeiro leitor bloqueia se houver um escritor ativo; qualquer outro leitor que tente entrar é bloqueado no mutex.
- O último leitor a sair sinaliza para o escritor que está esperando

- **Quando um escritor termina, se houver tanto leitores quanto escritores esperando, quem entra depende do escalonador**: O comportamento aqui depende do sistema operacional, que decide se um leitor ou um escritor será o próximo a acessar a seção crítica.

PROBLEMA LEITORES E ESCRITORES - DISCUSSÃO DA SOLUÇÃO

Notas de Implementação

- O primeiro leitor bloqueia se houver um escritor ativo; qualquer outro leitor que tente entrar é bloqueado no mutex.
- O último leitor a sair sinaliza para o escritor que está esperando
- Quando um escritor termina, se houver tanto leitores quanto escritores esperando, quem entra depende do escalonador

- **Se um escritor sair e um leitor for o próximo a entrar, todos os leitores que estiverem esperando também entram:** *Nesse caso, quando um escritor termina e um leitor começa a ler, todos os leitores que estavam esperando são liberados, incluindo aqueles que estavam esperando no wrt e outros que estavam bloqueados no mutex.*

PROBLEMA LEITORES E ESCRITORES - DISCUSSÃO DA SOLUÇÃO

Notas de Implementação

- O primeiro leitor bloqueia se houver um escritor ativo; qualquer outro leitor que tente entrar é bloqueado no mutex.
- O último leitor a sair sinaliza para o escritor que está esperando
- Quando um escritor termina, se houver tanto leitores quanto escritores esperando, quem entra depende do escalonador
- Se um escritor sair e um leitor for o próximo a entrar, todos os leitores que estiverem esperando também entram

Essa solução garante que todas as threads farão progresso?

PROBLEMA LEITORES E ESCRITORES - DISCUSSÃO DA SOLUÇÃO

Notas de Implementação

- O primeiro leitor bloqueia se houver um escritor ativo; qualquer outro leitor que tente entrar é bloqueado no mutex.
- O último leitor a sair sinaliza para o escritor que está esperando
- Quando um escritor termina, se houver tanto leitores quanto escritores esperando, quem entra depende do escalonador
- Se um escritor sair e um leitor for o próximo a entrar, todos os leitores que estiverem esperando também entram

Essa solução garante que todas as threads farão progresso?

Semântica alternativa: deixar o escritor ENTRAR na seção crítica o mais cedo possível.

PROBLEMA LEITORES E ESCRITORES - FAVORECENDO OS ESCRITORES

```
ReadWrite::Write() {
 mutex_escritores.Wait(); // garante a exclusão mútua para escritores
 escritores += 1; // incrementa o número de escritores pendentes
 if (escritores == 1) // se este for o primeiro escritor, bloqueia leitores
   bloco_leitores.Wait();
 mutex_escritores.Signal(); // libera o mutex para outros escritores

 bloco_escritores.Wait(); // garante exclusão mútua entre escritores
 <realiza a escrita>
 bloco_escritores.Signal(); // permite que outro escritor entre

 mutex_escritores.Wait(); // garante a exclusão mútua para atualizar contagem
 escritores -= 1; // escritor terminou, decrementa a contagem
 if (escritores == 0) // se não houver mais escritores, libera leitores
   bloco_leitores.Signal();
 mutex_escritores.Signal(); // libera o mutex para outros processos
}
```

PROBLEMA LEITORES E ESCRITORES - FAVORECENDO OS ESCRITORES

```
ReadWrite::Write() {
 mutex_escritores.Wait(); // garante a exclusão mútua para escritores
 escritores += 1; // incrementa o número de escritores pendentes
 if (escritores == 1) // se este for o primeiro escritor, bloqueia leitores
   bloco_leitores.Wait();
 mutex_escritores.Signal(); // libera o mutex para outros escritores

 bloco_escritores.Wait(); // garante exclusão mútua entre escritores
 <realiza a escrita>
 bloco_escritores.Signal(); // permite que outro escritor entre

 mutex_escritores.Wait(); // garante a exclusão mútua para atualizar contagem
 escritores -= 1; // escritor terminou, decrementa a contagem
 if (escritores == 0) // se não houver mais escritores, libera leitores
   bloco_leitores.Signal();
 mutex_escritores.Signal(); // libera o mutex para outros processos
}
```

LEITORES (readers) VÃO SE ACUMULANDO UM APÓS O OUTRO, E MESMO QUE UM ESCRITOR (writer) CHEGUE DEPOIS, ELE TERÁ PRIORIDADE

PROBLEMA LEITORES E ESCRITORES - FAVORECENDO OS ESCRITORES

```
ReadWrite::Write() {
 mutex_escritores.Wait(); // garante a exclusão mútua para escritores
 escritores += 1; // incrementa o número de escritores pendentes
 if (escritores == 1) // se este for o primeiro escritor, bloqueia leitores
   bloco_leitores.Wait();
 mutex_escritores.Signal(); // libera o mutex para outros escritores

 bloco_escritores.Wait(); // garante exclusão mútua entre escritores
 <realiza a escrita>
 bloco_escritores.Signal(); // permite que outro escritor entre

 mutex_escritores.Wait(); // garante a exclusão mútua para atualizar contagem
 escritores -= 1; // escritor terminou, decrementa a contagem
 if (escritores == 0) // se não houver mais escritores, libera leitores
   bloco_leitores.Signal();
 mutex_escritores.Signal(); // libera o mutex para outros processos
}
```

qual a diferença entre bloco_escritores e mutex_escritores ?

PROBLEMA LEITORES E ESCRITORES - FAVORECENDO OS ESCRITORES

```
ReadWrite::Read() {
    bloco_leitores.Wait();
    mutex_leitores.Wait(); // Garante exclusão mútua (evita condições de corrida)
    leitores += 1; // Incrementa o contador de leitores
    if (leitores == 1) { // Se for o primeiro leitor, bloqueia os escritores
        bloco_escritores.Wait();
    }
    mutex_leitores.Signal(); // Libera o mutex para permitir outros leitores
    bloco_leitores.Signal(); // Libera o bloqueio de leitura: permitir que leitores avancem

    <realizar leitura>     // Executa a leitura do recurso compartilhado

    mutex_leitores.Wait();     // Novamente, garante exclusão mútua
    leitores -= 1; // Leitor terminou a leitura
    if (leitores == 0) { // for o último leitor, libera o bloqueio para escritores
        bloco_escritores.Signal();
    }
    mutex_leitores.Signal();   // Libera o mutex para permitir que outros leitores
                               // ou escritores avancem
}
```

PROBLEMA LEITORES E ESCRITORES CENÁRIO 4

LEITOR_1	LEITOR_2	ESCRITOR_1	ESCRITOR_2
Read()			
	Read()		
		Write()	
			Write()

PROBLEMA LEITORES E ESCRITORES CENÁRIO 5

LEITOR_1	LEITOR_2	ESCRITOR_1	ESCRITOR_2
		Write()	
Read()			
	Read()		
			Write()

PROBLEMA LEITORES E ESCRITORES CENÁRIO 6

LEITOR_1	LEITOR_2	ESCRITOR_1	ESCRITOR_2
Read()			
		Write()	
	Read()		
			Write()

VIDA-REAL: READ/WRITE LOCKS

- Usualmente não precisamos nos preocupar com a implementação interna de READERS/WRITERS.

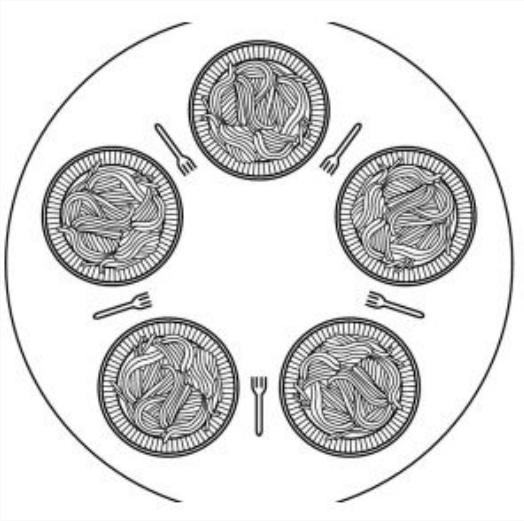
- A maioria das linguagens já suporta locks escritores e leitores.

- Uma thread pode adquirir um lock de escrita (Write Lock) ou um lock de leitura (Read Lock)
- Múltiplas threads podem ter o mesmo read lock de forma concorrente
- Apenas uma thread tem o write lock
- Java: classe ReadWriteLock (readLock() e writeLock())
- pthread:

```
pthread_rwlock_init()
pthread_rwlock_rdlock()
pthread_rwlock_wrlock()
pthread_rwlock_unlock()
```

PROBLEMA DOS FILÓSOFOS JANTANDO

- Estamos na hora do almoço no departamento de filosofia no **japão**.
- Há 5 filósofos, e cada um deles está alternadamente **comendo** ou **pensando**.
- Eles compartilham uma **mesa circular com cinco hashi's.**

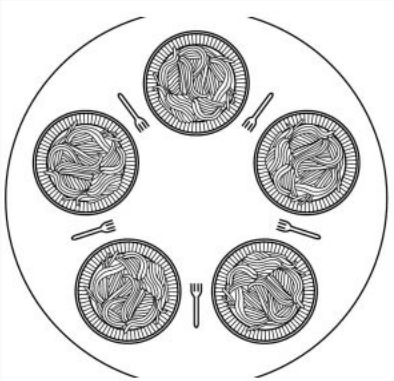


Regras:

- Pensar: **Quando um filósofo está pensando, ele não precisa de hashi** (não faz nada).
- Comer: **Para comer, um filósofo precisa pegar dois hashi** – os dois mais próximos a ele (um à esquerda e outro à direita).
  - *Ele **bloqueia (espera) se o vizinho já pegou um dos hashi** que ele precisa.*
- Depois de comer, o filósofo **devolve ambos os hashi à mesa e volta a pensar.**

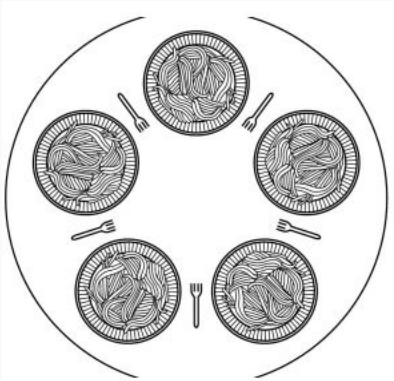
PROBLEMA DOS FILÓSOFOS JANTANDO

```
Semaforo hashi[5];
do {
  wait(hashi[i]); // hashi da esquerda
  wait(hashi[(i+1)%5]); // hashi da direita
    // comer
  signal(hashi[i]); // devolver hashi da esquerda
  signal(hashi[(i+1)%5]); // devolver hashi da direita
    // pensar
} while(TRUE);
```



PROBLEMA DOS FILÓSOFOS JANTANDO

```
Semaforo hashi[5];
do {
  wait(hashi[i]); // hashi da esquerda
  wait(hashi[(i+1)%5]); // hashi da direita
    // comer
  signal(hashi[i]); // devolver hashi da esquerda
  signal(hashi[(i+1)%5]); // devolver hashi da direita
    // pensar
} while(TRUE);
```



EM QUAL SITUAÇÃO OCORRE DEADLOCK ?

FILÓSOFOS JANTANDO - SOLUÇÃO COM SEMÁFOROS

```
#define N 5                     /* número de filósofos */
#define ESQUERDA (i+N-1)%N      /* número do vizinho à esquerda de i */
#define DIREITA (i+1)%N         /* número do vizinho à direita de i */
#define PENSANDO 0              /* filósofo está pensando */
#define FAMINTO 1               /* filósofo está tentando pegar garfos */
#define COMENDO 2               /* filósofo está comendo */

typedef int semaforo;           /* semáforos são um tipo especial de int */

int estado[N];                  /* array para acompanhar o estado de cada filósofo */
semaforo mutex = 1;             /* exclusão mútua para regiões críticas */
semaforo s[N];                  /* um semáforo por filósofo */

void filosofo(int i)            /* i: número do filósofo, de 0 a N-1 */
{
    while (TRUE) {              /* repetir para sempre */
        pensar();               /* filósofo está pensando */
        pegar_garfos(i);        /* tentar adquirir 2 garfos ou bloquear */
        comer();                /* nham-nham, espaguete */
        soltar_garfos(i);       /* devolver os dois garfos à mesa */
    }
}
```

FILÓSOFOS JANTANDO - SOLUÇÃO COM SEMÁFOROS

```
void pegar_garfos(int i)          /* i: número do filósofo, de 0 a N-1 */
{
    down(&mutex);                 /* entrar na região crítica */
    estado[i] = FAMINTO;          /* registrar que o filósofo está faminto */
    testar(i);                    /* tentar adquirir 2 garfos */
    up(&mutex);                   /* sair da região crítica */
    down(&s[i]);                  /* bloquear se os garfos não foram adquiridos */
}

void soltar_garfos(int i)         /* i: número do filósofo, de 0 a N-1 */
{
    down(&mutex);                 /* entrar na região crítica */
    estado[i] = PENSANDO;         /* filósofo terminou de comer */
    testar(ESQUERDA);             /* ver se o vizinho da esquerda pode comer agora */
    testar(DIREITA);              /* ver se o vizinho da direita pode comer agora */
    up(&mutex);                   /* sair da região crítica */
}

void testar(int i)                /* i: número do filósofo, de 0 a N-1 */
{
    if (estado[i] == FAMINTO && estado[ESQUERDA] != COMENDO && estado[DIREITA] != COMENDO) {
        estado[i] = COMENDO;      /* filósofo começa a comer */
        up(&s[i]);                /* desbloquear o filósofo que estava bloqueado */
    }
}
```

RESUMO

Problema dos leitores/escritores:
- Permitir que múltiplos leitores acessem os dados simultaneamente
- Permitir apenas um escritor por vez

Duas soluções possíveis usando semáforos:
- Priorizar leitores
- Priorizar escritores

INANIÇÃO (starvation) é possível em ambos os casos!

Filósofos jantando: acesso mutuamente exclusivo a múltiplos recursos

PERGUNTAS?

REFERÊNCIAS

- **TANENBAUM, Andrew.** Sistemas operacionais modernos.
- **SILBERSCHATZ, Abraham et al.** Fundamentos de sistemas operacionais: princípios básicos.
- **MACHADO, Francis; MAIA, Luiz Paulo.** Arquitetura de Sistemas Operacionais.
- **CARISSIMI, Alexandre et al.** Sistemas operacionais.