

# Sincronização

## Parte II – Programação Concorrente

---

Sistemas Operativos

2011 / 2012

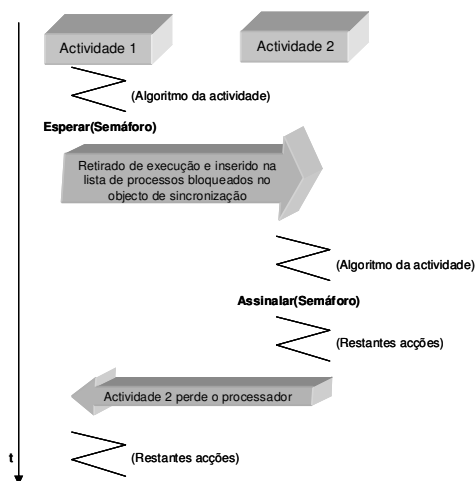
## Cooperação entre Processos

- Vários processos executam em conjunto uma ou mais tarefas, nas quais
  - Competem por recursos
  - Indicam uns aos outros a:
    - Ausência/existência de recursos
    - Ocorrência de acontecimentos

## Modelo inicial de sincronização

- Para sincronizar tarefas, pretendemos um mecanismo que:
  - independentemente do estado da tarefa e da sua velocidade de execução, permita assinalar que um acontecimento já ocorreu
  - e.g.: a Actividade 2 quer ser informada quando a Actividade 1 terminar a sua unidade de trabalho para poder prosseguir.
- Precisamos, portanto, de um mecanismo independente da velocidade de execução das acções que permita:
  - à Actividade 2 bloquear-se até que lhe seja assinalado que a Actividade 1 já concluiu a sua unidade de trabalho;
  - à Actividade 1 assinalar a conclusão, desbloqueando a Actividade 2.

## Situação Mais Simples de Cooperação



## Cooperação entre Dois Processos com Semáforos

```
Proci
semEvent = CriarSemaforo(0);
void EsperarAcontecimento() {
    Esperar (semEvent);
}

Procj
void AssinalarAcontecimento() {
    Assinalar (semEvent);
}
```

O semáforo é  
inicializado a  
zero

## Gestão de Recursos

- Um processo requisita um recurso
  - Executa Esperar (SemRecursos)
- Um processo liberta um recurso
  - Executa Assinalar (SemRecurso)
- O semáforo que controla o algoritmo é inicializado com o número de recursos disponíveis
  - SemRecurso = CriarSemaforo (NUM\_RECURSOS)

## Semáforos (exemplo)

```
#define MAX_PILHA 100  
char* pilha[MAX_PILHA];  
int topo = MAX_PILHA-1;  
semáforo_t SemMem;  
semáforo_t mutex;
```

```
char* PedeMem() {  
    → Esperar (SemMem);  
    Esperar(mutex);  
    ptr = pilha[topo];  
    topo--;  
    → Assinalar(mutex);  
    return ptr;  
}
```

```
void DevolveMem(char*  
ptr) {  
    → Esperar(mutex);  
    topo++;  
    pilha[topo]= ptr;  
    Assinalar(mutex);  
    → Assinalar (SemMem);  
}
```

```
main(){  
    /*...*/  
    semExMut = CriarSemaforo(1);  
    semMem = CriarSemaforo(MAX_PILHA);  
}
```

**Alocador de  
memória com  
o semáforo  
SemMem a  
controlar a  
existência de  
memória livre**

**O semáforo é  
inicializado com o valor  
dos recursos  
disponíveis**

## Problemas Típicos de Sincronização

- Algoritmo do Barbeiro
  - uma ou mais tarefas servidoras de tarefas clientes
- Algoritmo dos Produtores/Consumidores
  - tarefas que produzem informação para um buffer e tarefas que lêem a informação do buffer
- Algoritmo dos Leitores/Escretores
  - tarefas que pretendem ler uma estrutura de dados e tarefas que actualizam (escrevem) a mesma estrutura de dados

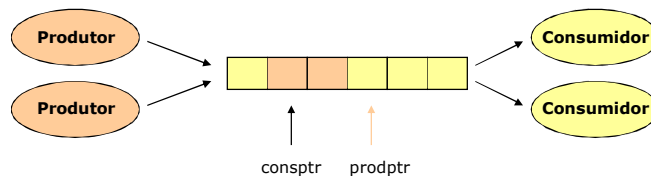
## Exercício - Barbeiro

- Numa barbearia existe uma cadeira onde:
  - o barbeiro corta cabelo e
  - N cadeiras para os clientes que estão à espera.
- Se não existem clientes:
  - o barbeiro senta-se na cadeira e adormece.
- Quando um cliente chega:
  - ele tem que acordar o barbeiro dorminhoco para lhe cortar o cabelo.
- Se entretanto chegarem mais clientes enquanto o barbeiro estiver a cortar o cabelo ao primeiro:
  - ou esperam numa cadeira livre ou
  - vão-se embora se já não houver mais cadeiras livres.



Será estudado na aula prática

## Exemplo de Cooperação entre Processos: Produtor - Consumidor



```

/* ProdutorConsumidor */
int buf[N];
int prodptr=0, consptr=0;

produtor()
while(TRUE) {
    int item = produz();
    buf[prodptr] = item;
    prodptr = (prodptr+1) % N;
}

```

Que acontece se o buffer estiver cheio ?

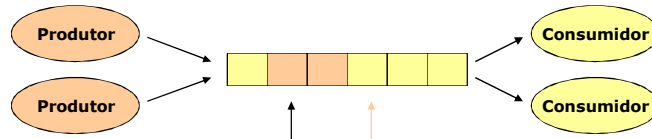
```

consumidor(){
while(TRUE) {
    int item;
    item = buf[consptr];
    consptr = (consptr+1) % N;
    consome(item);
}
}

```

Que acontece se não houver itens no buffer ?

## Exemplo de Cooperação entre Processos: Produtor - Consumidor



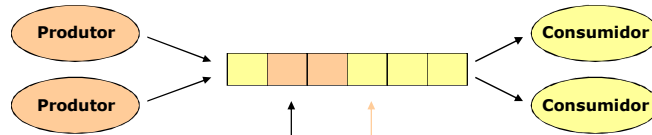
```
int buf[N];
int prodptr=0, consptr=0;
trinco_t trinco;
semaforo_t pode_prod = criar_semaforo(N),
           pode_cons = criar_semaforo(0);
```

Cada semáforo representa um recurso:  
pode\_produzir: espaços livres, inicia a N  
pode\_consumir: itens no buffer, inicia a 0

```
produtor() {
    while(TRUE) {
        int item = produz();
        esperar (pode_prod);
        fechar (trinco);
        buf[prodptr] = item;
        prodptr = (prodptr+1) % N;
        abrir (trinco);
        assinalar (pode_cons);
    }
}
```

```
consumidor() {
    while(TRUE) {
        int item;
        esperar (pode_cons);
        fechar (trinco);
        item = buf[consptr];
        consptr = (consptr+1) % N;
        abrir (trinco);
        assinalar (pode_prod);
        consome (item);
    }
}
```

## Exemplo de Cooperação entre Processos: Produtor - Consumidor



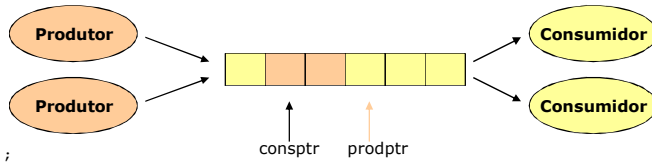
```
int buf[N];
int prodptr=0, consptr=0;
trinco_t trinco;
semaforo_t pode_prod = criar_semaforo(N),
           pode_cons = criar_semaforo(0);
```

```
produtor() {
    while(TRUE) {
        int item = produz();
        esperar (pode_prod);
        fechar (trinco);
        buf[prodptr] = item;
        prodptr = (prodptr+1) % N;
        abrir (trinco);
        assinalar (pode_cons);
    }
}
```

```
consumidor() {
    while(TRUE) {
        int item;
        esperar (pode_cons);
        fechar (trinco);
        item = buf[consptr];
        consptr = (consptr+1) % N;
        abrir (trinco);
        assinalar (pode_prod);
        consome (item);
    }
}
```

**Problema?**

### Exemplo de Cooperação entre Processos: Produtor - Consumidor



```

int buf[N];
int prodptr=0, consptr=0;
trinco_t trinco;
semaforo_t pode_prod = criar_semaforo(N),
           pode_cons = criar_semaforo(0);

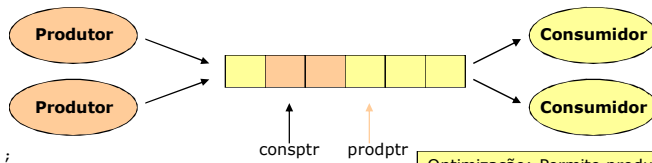
produtor() {
    while(TRUE) {
        int item = produz();
        esperar (pode_prod);
        fechar (trinco);
        buf[prodptr] = item;
        prodptr = (prodptr+1) % N;
        abrir (trinco);
        assinalar (pode_cons);
    }
}

consumidor() {
    while(TRUE) {
        int item;
        esperar (pode_cons);
        fechar (trinco);
        item = buf[consptr];
        consptr = (consptr+1) % N;
        abrir (trinco);
        assinalar (pode_prod);
        consome (item);
    }
}

```

**Problema?**

### Exemplo de Cooperação entre Processos: Produtor - Consumidor



```

int buf[N];
int prodptr=0, consptr=0;
trinco_t trinco_p, trinco_c;
semaforo_t pode_prod = criar_semaforo(N),
           pode_cons = criar_semaforo(0);

produtor() {
    while(TRUE) {
        int item = produz();
        esperar (pode_prod);
        fechar (trinco_p);
        buf[prodptr] = item;
        prodptr = (prodptr+1) % N;
        abrir (trinco_p);
        assinalar (pode_cons);
    }
}

consumidor() {
    while(TRUE) {
        int item;
        esperar (pode_cons);
        fechar (trinco_c);
        item = buf[consptr];
        consptr = (consptr+1) % N;
        abrir (trinco_c);
        assinalar (pode_prod);
        consome (item);
    }
}

```



## Problema dos Leitores - Escritores

- Pretende-se gerir o acesso a uma estrutura de dados partilhada em que existem duas classes de processos:
  - Leitores – apenas lêem a estrutura de dados
  - Escritores – lêem e modificam a estrutura de dados
- Condições
  - Os escritores só podem aceder em exclusão mútua
  - Os leitores podem aceder simultaneamente com outros leitores mas em exclusão mútua com os escritores
  - Nenhuma das classes de processos deve ficar à mingua



## Problema dos Leitores - Escritores

```
leitor() {  
    while (TRUE) {  
        inicia_leitura();  
        leitura();  
        acaba_leitura();  
    }  
}  
  
escritor() {  
    while (TRUE) {  
        inicia_escrita();  
        escrita();  
        acaba_escrita();  
    }  
}
```



## Problema dos Leitores – Escritores: hipótese 1

```
leitor() {  
  while (TRUE) {  
    fechar(mutex);  
    leitura();  
    abrir(mutex);  
  }  
}  
  
escritor() {  
  while (TRUE) {  
    fechar(mutex);  
    escrita();  
    abrir(mutex);  
  }  
}
```

**Demasiado forte!**  
**É possível permitir mais paralelismo!**

## Leitores – Escritores: Dificuldades

- Condições de bloqueio mais complexas:
  - escritor bloqueia se houver um leitor ou um escritor em simultâneo
- Com quem deve ser feita a sincronização?
  - quando termina uma escrita, deve ser assinalado o leitor seguinte (se houver) ou o escritor seguinte (se houver).
  - e se não estiver ninguém à espera?
- Solução:
  - ler variáveis antes de efectuar esperar/assinalar

## Leitores-Escritores: esboço da solução

```

int nleitores=0;
boolean_t em_escrita=FALSE;
int leitores_espera=0, escritores_espera=0;
-----
inicia_leitura()
{
    if (em_escrita || escritores_espera > 0) {
        leitores_espera++;
        Bloquear até não haver ninguém a escrever
        leitores_espera--;
    }
    nleitores++;
}
-----
acaba_leitura()
{
    nleitores--;
    Desbloquear quem esteja à espera para escrever
}
-----
inicia_escrita()
{
    if (em_escrita || nleitores > 0) {
        escritores_espera++;
        Bloquear até não haver ninguém a escrever ou a ler
        escritores_espera--;
    }
    em_escrita = TRUE;
}
-----
acaba_escrita()
{
    em_escrita = FALSE;
    Desbloquear quem esteja à espera para ler ou para escrever
}

```

## Leitores-Escritores: esboço da solução

```

int nleitores=0;
boolean_t em_escrita=FALSE;
int leitores_espera=0, escritores_espera=0;
-----
inicia_leitura()
{
    if (em_escrita || escritores_espera > 0) {
        esperar (leitores);
        Bloquear até não haver ninguém a escrever
        leitores_espera--;
    }
    nleitores++;
}
-----
acaba_leitura()
{
    nleitores--;
    Desbloquear quem esteja à espera para escrever
    assinalar (escritores);
}
-----
semaforo_t leitores=0, escritores=0;
-----
inicia_escrita()
{
    if (em_escrita || nleitores > 0) {
        escritores_espera++;
        esperar (escritores);
        Bloquear até não haver ninguém a escrever ou a ler
        escritores_espera--;
    }
    em_escrita = TRUE;
}
-----
acaba_escrita()
{
    em_escrita = FALSE;
    Desbloquear quem esteja à espera para ler ou para escrever
    if (leitores_espera > 0)
        for (i=0; i<leitores_espera; i++)
            assinalar (leitores);
    else if (escritores_espera > 0)
        assinalar (escritores);
}

```

**Não existem secções críticas???**

## Leitores-Escritores: esboço da solução

```

int nleitores=0;
boolean_t em_escrita=FALSE;
int leitores_espera=0, escritores_espera=0;
-----
inicia_leitura()
{
  fechar(m);
  if (em_escrita || escritores_espera > 0) {
    leitores_espera++;

    esperar(leitores);
    // Bloquear até não haver ninguém a escrever

    leitores_espera--;
  }
  nleitores++;
  abrir(m);
}
-----
acaba_leitura()
{
  fechar(m);
  nleitores--;
  if (nleitores == 0 && escritores_espera > 0)
    assinalar(escritores);
  // Desbloquear quem esteja à espera para escrever

  abrir(m);
}

semaforo_t leitores=0, escritores=0;
trinco_t m;
-----
inicia_escrita()
{
  fechar(m);
  if (em_escrita || nleitores > 0) {
    escritores_espera++;

    esperar(escritores);
    // Bloquear até não haver ninguém a escrever ou a ler

    escritores_espera--;
  }
  em_escrita = TRUE;
  abrir(m);
}
-----
acaba_escrita()
{
  fechar(m);
  em_escrita = FALSE;
  if (leitores_espera > 0)
    for (i=0; i<leitores_espera; i++)
      assinalar(leitores);
  // Desbloquear quem esteja à espera para ler ou para escrever

  else if (escritores_espera > 0)
    assinalar(escritores);

  abrir(m);
}

```

## Leitores-Escritores: esboço da solução

```

int nleitores=0;
boolean_t em_escrita=FALSE;
int leitores_espera=0, escritores_espera=0;
-----
inicia_leitura()
{
  fechar(m);
  if (em_escrita || escritores_espera > 0) {
    leitores_espera++;
    abrir(m);
    // Bloquear até não haver ninguém a escrever

    esperar(leitores);
    fechar(m);
    leitores_espera--;
  }
  nleitores++;
  abrir(m);
}
-----
acaba_leitura()
{
  fechar(m);
  nleitores--;
  if (nleitores == 0 && escritores_espera > 0)
    assinalar(escritores);
  // Desbloquear quem esteja à espera para escrever

  abrir(m);
}

semaforo_t leitores=0, escritores=0;
trinco_t m;
-----
inicia_escrita()
{
  fechar(m);
  if (em_escrita || nleitores > 0) {
    escritores_espera++;
    abrir(m);
    // Bloquear até não haver ninguém a escrever ou a ler

    esperar(escritores);
    fechar(m);
    escritores_espera--;
  }
  em_escrita = TRUE;
  abrir(m);
}
-----
acaba_escrita()
{
  fechar(m);
  em_escrita = FALSE;
  if (leitores_espera > 0)
    for (i=0; i<leitores_espera; i++)
      assinalar(leitores);
  // Desbloquear quem esteja à espera para ler ou para escrever

  else if (escritores_espera > 0)
    assinalar(escritores);

  abrir(m);
}

```

**Leitores-Escritores: esboço da solução**

```

int nleitores=0;
boolean_t em_escrita=FALSE;
int leitores_espera=0, escritores_espera=0;

-----
inicia_leitura()
{
  fechar(m);
  if (em_escrita || escritores_espera > 0) {
    leitores_espera++;
    abrir(m);
    esperar(leitores);
    fechar(m);
    leitores_espera--;
  }
  nleitores++;
  abrir(m);
}
-----
acaba_leitura()
{
  fechar(m);
  nleitores--;
  if (nleitores == 0 && escritores_espera > 0)
    assinalar(escritores);
  abrir(m);
}

```

```

semaforo_t leitores=0, escritores=0;
trinco_t m;

-----
inicia_escrita()
{
  fechar(m);
  if (em_escrita || nleitores > 0) {
    escritores_espera++;
    abrir(m);
    esperar(escritores);
    fechar(m);
    escritores_espera--;
  }
  em_escrita = TRUE;
  abrir(m);
}
-----
acaba_escrita()
{
  fechar(m);
  em_escrita = FALSE;
  if (leitores_espera > 0)
    for (i=0; i<leitores_espera; i++)
      assinalar(leitores);
  else if (escritores_espera > 0)
    assinalar(escritores);
  abrir(m);
}

```

Bloquear até não haver ninguém a escrever

Bloquear até não haver ninguém a escrever ou a ler

Desbloquear quem esteja à espera para escrever

Desbloquear quem esteja à espera para ler ou para escrever

Problema: e se uma nova tarefa obtém acesso antes das tarefas assinaladas?

**Leitores-Escritores: esboço da solução**

```

semaforo_t leitores=0, escritores=0;
trinco_t m;

int nleitores=0;
boolean_t em_escrita=FALSE;
int leitores_espera=0, escritores_espera=0;

-----
inicia_leitura()
{
  fechar(m);
  if (em_escrita || escritores_espera > 0) {
    leitores_espera++;
    abrir(m);
    esperar(leitores);
    fechar(m);
    leitores_espera--;
  }
  else nleitores++;
  abrir(m);
}
-----
acaba_leitura()
{
  fechar(m);
  nleitores--;
  if (nleitores == 0 && escritores_espera > 0)
    assinalar(escritores);
  em_escrita=TRUE;
  escritores_espera--;
  abrir(m);
}

```

```

-----
inicia_escrita()
{
  fechar(m);
  if (em_escrita || nleitores > 0) {
    escritores_espera++;
    abrir(m);
    esperar(escritores);
    fechar(m);
    escritores_espera--;
  }
  em_escrita = TRUE;
  abrir(m);
}
-----
acaba_escrita()
{
  fechar(m);
  em_escrita = FALSE;
  if (leitores_espera > 0)
    for (i=0; i<leitores_espera; i++) {
      assinalar(leitores);
      nleitores++;
      leitores_espera--;
    }
  else if (escritores_espera > 0) {
    assinalar(escritores);
    em_escrita=TRUE;
    escritores_espera--;
  }
  abrir(m);
}

```

Bloquear até não haver ninguém a escrever ou a ler

Desbloquear quem esteja à espera para escrever

**Leitores-Escritores: esboço da solução**

**semaforo\_t leitores=0, escritores=0;**  
**trinco\_t m;**

```

int nleitores=0;
boolean_t em_escrita=FALSE;
int leitores_espera=0, escritores_espera=0;
-----
inicia_leitura()
{
  fechar(m);
  if (em_escrita || escritores_espera > 0) {
    leitores_espera++;
    abrir(m);
    esperar(leitores);
    fechar(m);
    leitores_espera--;
  }
  else nleitores++;
  abrir(m);
}
-----
acaba_leitura()
{
  fechar(m);
  nleitores--;
  if (nleitores == 0 && escritores_espera > 0)
    assinalar(escritores);
  em_escrita=TRUE;
  escritores_espera--;
}
-----
abrir(m);
}

```

**inicia\_escrita()**

```

{
  fechar(m);
  if (em_escrita || nleitores > 0) {
    escritores_espera++;
    abrir(m);
    esperar(escritores);
    fechar(m);
    escritores_espera--;
  }
  em_escrita = TRUE;
  abrir(m);
}
-----
acaba_escrita()
{
  fechar(m);
  em_escrita = FALSE;
  if (leitores_espera > 0)
    for (i=0; i<leitores_espera; i++) {
      assinalar(leitores);
      nleitores++;
      leitores_espera--;
    }
  else if (escritores_espera > 0) {
    assinalar(escritores);
    em_escrita=TRUE;
    escritores_espera--;
  }
}
-----
abrir(m);
}

```

**Bloquear até não haver ninguém a escrever ou a ler**

**Bloquear até não haver ninguém a escrever**

**Desbloquear quem esteja à espera para escrever**

**Desbloquear quem esteja à espera para ler ou para escrever**

**Problema?**

**Leitores-Escritores: esboço da solução**

**semaforo\_t leitores=0, escritores=0;**  
**trinco\_t m;**

```

int nleitores=0;
boolean_t em_escrita=FALSE;
int leitores_espera=0, escritores_espera=0;
-----
inicia_leitura()
{
  fechar(m);
  if (em_escrita || escritores_espera > 0) {
    leitores_espera++;
    abrir(m);
    esperar(leitores);
    fechar(m);
    leitores_espera--;
  }
  else nleitores++;
  abrir(m);
}
-----
acaba_leitura()
{
  fechar(m);
  nleitores--;
  if (nleitores == 0 && escritores_espera > 0)
    assinalar(escritores);
  em_escrita=TRUE;
  escritores_espera--;
}
-----
abrir(m);
}

```

**inicia\_escrita()**

```

{
  fechar(m);
  if (em_escrita || nleitores > 0) {
    escritores_espera++;
    abrir(m);
    esperar(escritores);
    fechar(m);
    escritores_espera--;
  }
  em_escrita = TRUE;
  abrir(m);
}
-----
acaba_escrita()
{
  fechar(m);
  em_escrita = FALSE;
  if (leitores_espera > 0)
    for (i=0; i<leitores_espera; i++) {
      assinalar(leitores);
      nleitores++;
      leitores_espera--;
    }
  else if (escritores_espera > 0) {
    assinalar(escritores);
    em_escrita=TRUE;
    escritores_espera--;
  }
}
-----
abrir(m);
}

```

**Bloquear até não haver ninguém a escrever ou a ler**

**Bloquear até não haver ninguém a escrever**

**Desbloquear quem esteja à espera para escrever**

**Desbloquear quem esteja à espera para ler ou para escrever**

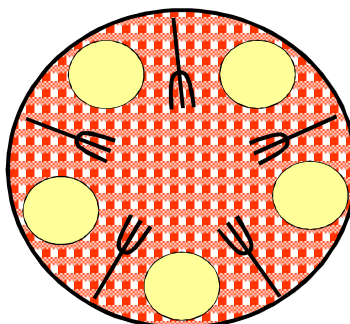
**Problema?**

**Eficiência: liberta leitor mas este fica bloqueado no trinco m (inicia\_leitura)**

## Jantar dos Filósofos

- Cinco Filósofos estão reunidos para filosofar e jantar spaghetti:
  - Para comer precisam de dois garfos, mas a mesa apenas tem um garfo por pessoa.
- Condições:
  - Os filósofos podem estar em um de três estados : Pensar; Decidir comer ; Comer.
  - O lugar de cada filósofo é fixo.
  - Um filósofo apenas pode utilizar os garfos imediatamente à sua esquerda e direita.

## Jantar dos Filósofos



## Jantar dos Filósofos com Semáforos, versão #1

```
semaforo_t garfo[5] = {1, 1, 1, 1, 1};
```

```
filosofo(int id)
{
    while (TRUE) {
        pensar();
        esperar(garfo[id]);
        esperar(garfo[(id+1)%5]);
        comer();
        assinalar(garfo[id]);
        assinalar(garfo[(id+1)%5]);
    }
}
```

- Problema?

## Jantar dos Filósofos com Semáforos, versão #2

```
semaforo_t garfo[5] = {1, 1, 1, 1, 1};
```

```
filosofo(int id)
{
    while (TRUE) {
        pensar();
        if (id == 4) {
            esperar(garfo[(id+1)%5]);
            esperar(garfo[id]);
        } else {
            esperar(garfo[id]);
            esperar(garfo[(id+1)%5]);
        }
        comer();
        assinalar(garfo[id]);
        assinalar(garfo[(id+1)%5]);
    }
}
```

- Adquirir os semáforos **sempre pela mesma ordem** (ordem crescente de número de semáforo)
- Solução preventiva genérica para interbloqueamento
- Outras soluções:
  - Requisitar recursos no início da execução com chamada não bloqueante “trylock”, libertar se falha
  - Detecção de interbloqueamento, libertação forçada por eliminação do processo

## Jantar dos Filósofos com Semáforos, versão #3



## Jantar dos Filósofos com Semáforos, versão #3

```

#define PENSAR 0
#define FOME 1
#define COMER 2
#define N 5
int estado[N] = {0, 0, 0, 0, 0};
semaforo_t semfilo[N] = {0, 0, 0, 0, 0};
trinco mutex;

Testa(int k){
    if (estado[k] == FOME &&
        estado[(k+1)%N] != COMER &&
        estado[(k-1)%N] != COMER){
        estado[k] = COMER;
        assinalar(semfilo[k]);
    }
}

filosofo(int id)
{
    while (TRUE) {
        pensar();
        fechar(mutex);
        estado[id] = FOME;
        Testa(id);
        abrir(mutex);
        esperar(semfilo[id]);
        comer();
        fechar(mutex);
        estado[id] = PENSAR;
        Testa((id-1+N)%N);
        Testa((id+1)%N);
        abrir(mutex);
    }
}

```

- Usar um semáforo para representar a condição de bloqueio (e não o estado dos garfos/filósofos)
- Representar o estado dos garfos/filósofos com variáveis acedidas numa secção crítica



## Jantar dos Filósofos com Semáforos, versão #4

```
semaforo_t garfo[5] = {1, 1, 1, 1, 1};  
semaforo_t sala = 4;
```

```
filosofo(int id)  
{  
    while (TRUE) {  
        pensar();  
        esperar(sala);  
        esperar(garfo[id]);  
        esperar(garfo[(id+1)%5]);  
        comer();  
        assinalar(garfo[id]);  
        assinalar(garfo[(id+1)%5]);  
        assinalar(sala);  
    }  
}
```

- Limitar o acesso à “sala” a N-1 filósofos (fica sempre pelo menos um garfo livre)

## Monitores

# Monitores

- Objectivo
  - Mecanismos de sincronização para linguagens de programação que resolvesse a maioria dos problemas de partilha de estruturas de dados:
    - Garantir implicitamente a exclusão mútua
    - Mecanismos para efectuar a sincronização explícita dos processos em algoritmos de cooperação ou de gestão de recursos

## Monitores, tão simples como isto:

```
synchronized void meuMetodo () {  
    //Aqui estou na secção crítica  
    while (condicaoNecessaria == false) wait();  
    //Aqui estou na secção crítica e a condição verifica-se  
  
    // Quando retornar, liberto a secção crítica  
}
```

Caso precise esperar  
por condição,  
liberto a secção crítica  
e bloqueio-me

## Monitores

- Declarado como um tipo abstracto, classe ou módulo:
  - Estrutura de dados interna
  - Interface Funcional:
    - Procedimentos acedidos em exclusão mútua
    - Procedimentos que não modificam o estado e que podem ser invocados sem ser em exclusão mútua.

## Monitores - Sincronização

- Exclusão mútua:
  - implícita na entrada no monitor.
- Tarefa que entre no monitor ganha acesso à secção crítica.
- Tarefa que sai do monitor liberta a secção crítica.

## Monitores - Sincronização

- Variáveis condição
  - declaradas na estrutura de dados
- Wait
  - liberta a secção crítica.
  - tarefa é colocada numa fila associada à condição do wait.
- Signal
  - assinala a condição.
  - se existirem tarefas na fila da condição, desbloqueia a primeira.

## Semântica do signal

- Semântica habitual:
  - Signal desbloqueia uma tarefa da fila associada à condição.
  - Mas não liberta a secção crítica.
    - Wait só retorna após a tarefa que se desbloqueou voltar a conseguir entrar na secção crítica
    - Acontece quando?
  - Se não existirem tarefas na fila, o efeito perde-se (ao contrário dos semáforos as condições não memorizam os acontecimentos).
- Existem outras semânticas

## Comparação: Semáforos e Monitores

	Semáforos	Monitores
Exclusão Mútua	Mecanismo Básico de Sincronização: <i>Mutexes</i> ou Semáforo inicializado a 1	Implícita
Cooperação	Semáforos Inicializados a zero (sem. privados)	Variáveis condição

## Problemas na Definição dos Monitores

- Chamadas de um monitor a outros monitores.
  - Possibilidade de interbloqueagem
- Semântica do Signal
  - Quando é executado um signal o processo que executa e o que é acordado ficam simultaneamente activos no interior do monitor o que invalida a exclusão mútua.

## Problema do Signal

- Soluções com signal
  - o processo que executa o signal abandona o monitor
  - o signal é a última instrução do procedimento
- Notify
  - Uma primitiva alternativa ao signal que apenas indica a possibilidade que a condição que levou ao bloqueio já não se verifique.
  - Neste caso o processo desbloqueado deve testar novamente a condição

## Monitores: leitores/escritores (em Java)

```
class escritoresLeitores
    int leitores = 0; int escritores = 0;
    int leitoresEmEspera = 0; int escritoresEmEspera = 0;
```

```
synchronized void iniciaLeitura () {
    leitoresEmEspera++;
    while (escritoresEmEspera > 0 || escritores > 0) wait();
    leitoresEmEspera--; leitores++;
}
```

```
synchronized void acabaLeitura () {
    leitores--;
    notifyAll();
}
```

```
synchronized void iniciaEscrita () {
    escritoresEmEspera++;
    while (leitores > 0 || escritores > 0) wait();
    escritoresEmEspera --;
    escritores ++;
}
```

```
synchronized void acabaEscrita () {
    escritores--;
    notifyAll();
}
```

**Esta solução não evita mingua**



## Monitores: caixa de correio (em Java) (produtores-consumidores)

```
class caixaCorreio {
    int MAX = 10; int[] tampao = new int[MAX];
    int contador = 0; int indPor = 0; int indTirar = 0;

    synchronized void enviar () {
        while (contador == MAX) wait();
        tampao[indPor] = mensagem;
        indPor++; if (indPor == MAX) indPor = 0; contador ++;
        notifyAll();
    }

    synchronized void receber () {
        while (contador == 0) wait();
        mensagem = tampao[indTirar] ;
        indTirar++; if (indTirar == MAX) indTirar = 0; contador --;
        notifyAll();
    }
}
```



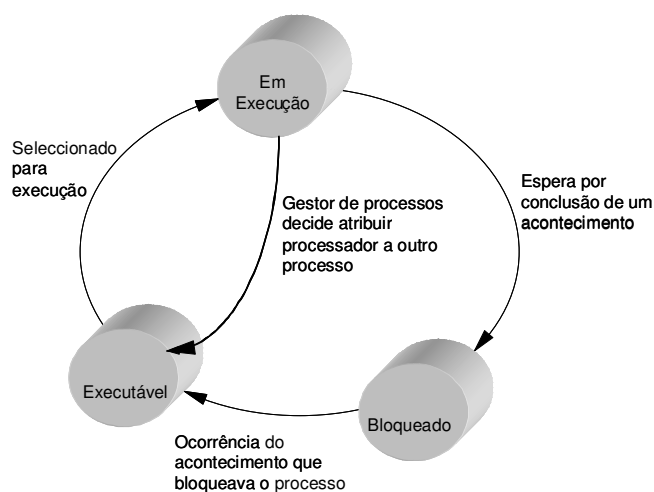
## Mecanismos Directos de Sincronização

- Objectivo
  - Suspender temporariamente a execução de subprocessos
- Limitações:
  - A sincronização directa implica o conhecimento do identificador do processo sobre o qual se pretende actuar.
  - Não se pode dar aos programas dos utilizadores a possibilidade de interferirem com outros utilizadores
  - A restrição habitual é apenas permitir o uso de sincronização directa entre processos do mesmo utilizador

## Mecanismos Directos de Sincronização

- Funções que actuam directamente sobre o estado dos processos:
  - `Suspend` (`IdProcesso`)
  - `Acordar` (`IdProcesso`)
- A função de suspensão é também frequentemente utilizada para implementar mecanismos de atraso temporizado que funcionam como uma auto-suspensão:
  - `Adormecer` (`Período`)

## Diagrama de Estado dos Processos





# Diagrama de Estado dos Processos

