

**INE 5410**  
**PROGRAMAÇÃO**  
**CONCORRENTE**

**Prof. Vitório Bruno**  
**Mazzola**  
**mazzola@inf.ufsc.br**



## Capítulo 2

# SEMÁFORO S



# Sincronização

- **Sequenciamento de instruções**

- Num programa comum, seqüência entre instruções é ditada pela ordenação estabelecida no código
- Uma instrução A é executada antes de uma instrução B se A aparece antes de B no código do programa
- Ocorrência de eventos associados às instruções é determinística

# Sincronização

- **Não determinismo**

- **Situação em que não se pode prever a ordem em que eventos irão ocorrer...**
  - Num computador com diversos processadores
  - Num computador com um único processador capaz de executar diversos fluxos de instrução (*threads*)
- **Nestes casos, pelos mecanismos tradicionais, o programador não tem controle sobre quando cada thread ou programa irão ser executados**

# Sincronização

- **Impondo o seqüenciamento**

- **Dois amigos, Pedro e Paulo, moram em cidades diferentes**

- **Acordo entre os dois que Paulo só irá almoçar depois que Pedro almoçou**

- **Solução utilizando relógios não garante o seqüenciamento**

- **A troca de mensagens pode ser uma solução interessante**



# Sincronização

- **Impondo o seqüenciamento**

Pedro

a1: tomar café da manhã  
a2: estudar  
a3: almoçar  
a4: ligar para Paulo

Paulo

b1: tomar café da manhã  
b2: aguardar ligação  
b3: almoçar

- **Nos programas**

- **Pedro:  $a1 < a2 < a3 < a4$**
- **Paulo:  $b1 < b2 < b3$**

# Sincronização

- Impondo o seqüenciamento

*Pedr*

<sup>0</sup>  
a1: tomar café da manhã  
a2: estudar  
a3: almoçar  
a4: ligar para Paulo

*Paulo*

b1: tomar café da manhã  
b2: aguardar ligação  
b3: almoçar

- Entre os programas

- $a3 < b3$

# Sincronização

- **Impondo o seqüenciamento**

- **Escrita concorrente**

a1:  $x = 5$   
a2: escrever  $x$

b1:  $x = 7$

- **Questões:**

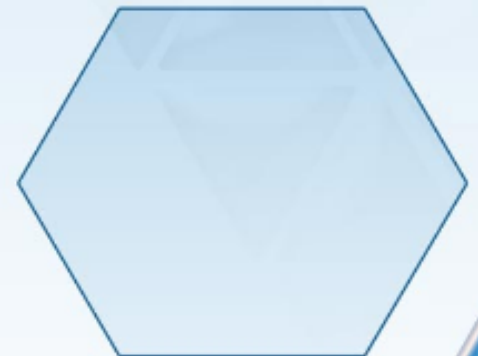
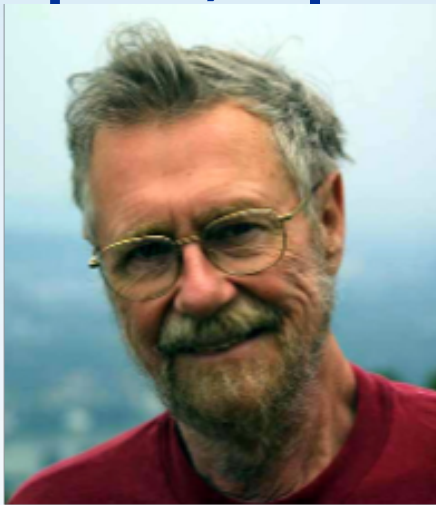
- Qual valor será impresso?
    - Qual será o valor final de  $x$ ?



# Semáforo

- **Primórdios**

- Foram criados por Edsger Dijkstra, com base nos semáforos do dia a dia, considerando que, de certa forma, estes servem à sinalização de controle de um recurso compartilhado que é o espaço de circulação para veículos e pedestres



# Semáforo

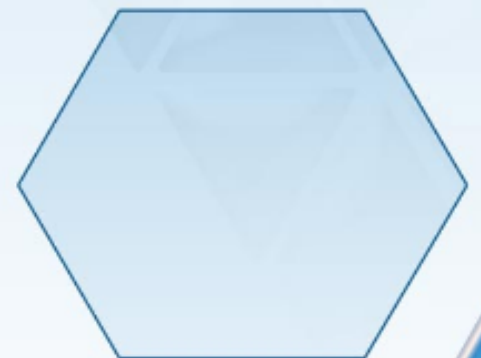
- **Definição**

- **Semáforo é um mecanismo importante para o controle da concorrência que é definido como um inteiro, com algumas diferenças:**
  - **Ao ser criado, pode ser inicializado a qualquer valor inteiro e as únicas operações possíveis são incremento e decremento de uma unidade**
  - **Quando uma thread decrementa o valor de um semáforo e este assume o valor negativo, ela fica bloqueada até que uma outra thread incremente este semáforo**

# Semáforo

- **Definição**

- **Semáforo é um mecanismo importante para o controle da concorrência que é definido como um inteiro, com algumas diferenças:**
  - Quando uma thread incrementa um semáforo, se existirem outras threads bloqueadas em relação a ele, uma delas é desbloqueada, continuando sua execução e as demais permanecerão bloqueadas



# Semáforo

- Sintaxe Genérica

```
mutex := semáforo(1);
```

```
mutex.decrementar;  
mutex.incrementar;
```

```
mutex.wait();;  
mutex.signal();
```

```
mutex.V();;  
mutex.P();
```

# Semáforo

- **Semântica das operações**

- **Decremento – wait() ou V()**

- Quando uma thread executa a operação *decremento*, ela fica bloqueada se o semáforo assume um valor negativo... Caso contrário, ela continua sua execução, entrando na parte de código que sucede o comando de decremento do semáforo
    - Isto corresponderia ao bloqueio da utilização de um ou mais recursos por outras threads que concorram a estes



# Semáforo

- **Semântica das operações**

- **Incremento – signal() ou P()**

- Quando uma thread executa a operação de incremento, ela ao mesmo tempo está sinalizando threads bloqueadas, podendo, eventualmente, permitir a continuidade da execução destas
- Corresponderia ao aviso de liberação de um ou mais recursos, para uso por threads bloqueadas que compartilhem o uso do mesmo.



# Semáforo

- **Cenários de uso**

- **Sinalização de threads**

- Semáforos podem ser utilizados por uma thread para sinalizar uma ou várias outras que algum evento acabou de ocorrer
- Situação típica de quando se quer garantir que um trecho de código de uma thread ocorra antes da execução de um trecho de código de outra
- Solução do problema de *serialização* entre *threads*

# Semáforo

- Cenários de uso

*thread A*

```
instrução a1;  
instrução a2;  
...  
instrução an;  
sem.signal();
```

*thread B*

```
sem.wait();  
instrução b1;  
instrução b2;  
...  
instrução bn;
```

- Neste caso, o semáforo deve ser inicializado a 0.

# Semáforo

- **Cenários de uso**

- **Sincronização (rendez-vous) de threads**

- A situação, neste caso, consiste em garantir que certos trechos de códigos em diferentes threads executem de forma ordenada
- Neste caso, diversos semáforos podem vir a ser utilizados
- A sincronização é feita utilizando-se corretamente as operações sobre os diversos semáforos entre as diversas threads

# Semáforo

- Cenários de uso

*thread A*

instrução a1;  
instrução a2;

*thread B*

instrução b1;  
instrução b2;

- Objetivo é assegurar que *a1* execute antes de *b2* e que *b1* execute antes de *a2*.

# Semáforo

- Cenários de uso

*thread A*

```
instrução a1;  
SA.signal();  
SB.wait();  
instrução a2;
```

*thread B*

```
instrução b1;  
SB.signal();  
SA.wait();  
instrução b2;
```

- Semáforos SA e SB devem ser inicializados a 0.

# Semáforo

- Cenários de uso

*thread A*

```
instrução a1;  
SB.wait();  
SA.signal();  
instrução a2;
```

*thread B*

```
instrução b1;  
SB.signal();  
SA.wait();  
instrução b2;
```

- Solução que funciona porém exige um chaveamento adicional desnecessário.



# Semáforo

- Cenários de uso

*thread A*

```
instrução a1;  
SB.wait();  
SA.signal();  
instrução a2;
```

*thread B*

```
instrução b1;  
SA.wait();  
SB.signal();  
instrução b2;
```

- Solução que vai gerar bloqueio entre as threads (*deadlock*).

# Semáforo

- **Cenários de uso**

- **Exclusão mútua**

- Uma situação bastante comum onde os semáforos são utilizados é o caso da exclusão mútua
    - Gerenciamento de recursos compartilhados entre threads
    - Objetivo é garantir que apenas uma thread assuma o controle exclusivo do recurso a cada momento, liberando-o quando as operações sobre este estiverem finalizadas

# Semáforo

- Cenários de uso

*thread A*

`conta := conta + 1;`

*thread B*

`conta := conta + 1;`

- As duas threads não podem executar o código simultaneamente.

# Semáforo

- Cenários de uso

*thread A*

```
mutex.wait();  
conta := conta + 1;  
mutex.signal();
```

*thread B*

```
mutex.wait();  
conta := conta + 1;  
mutex.signal();
```

- Semáforo *mutex* é inicializado a 1.

# Semáforo

- **Cenários de uso**

- **Multiplex**

- **Cenário que se configura quando um dado recurso pode ser acessado simultaneamente por diversas threads**
    - **Gerenciamento de recurso com acesso limitado em  $n$**
    - **Um caso prático seria um servidor configurado para atender a um número limitado de acessos da parte de clientes;**

# Semáforo

- **Cenários de uso**

- **Multiplex**

- Neste caso, é criado um semáforo associado ao recurso
    - Este semáforo é inicializado a  $n$ , onde  $n$  é o limite de acesso ao recurso considerado;
    - Quando o número de acessos atinge o limite, as próximas threads que tentem acessá-lo terão de ficar em espera pela liberação do mesmo;



# Semáforo

- Cenários de uso

*thread A*

`conta := conta + 1;`

*thread B*

`conta := conta + 1;`

- As duas threads não podem executar o código simultaneamente.

# Semáforo

- Cenários de uso

*thread*

```
sem.wait();  
código da seção crítica;  
sem.signal();
```

- Semáforo *sem* é inicializado a *n*.

# Semáforo

- Implementação de semáforo no Pascal-FC

- Declarações de Semáforos

- São declarados na parte sessão VAR de programas Pascal-FC
- Podem ser declarados como escalares ou vetores (ARRAY);

```
...  
VAR  
mutex : semaphore;  
BEGIN  
...
```

```
...  
VAR  
s : ARRAY[1..15] of semaphore;  
BEGIN  
...
```

# Semáforo

- Implementação de semáforo no Pascal-FC

- Inicializando um semáforo

- A inicialização de um semáforo é realizada utilizando um procedimento denominado *initial*;
- Este procedimento apresenta como argumentos o nome da variável semáforo e o valor com o qual o semáforo será inicializado;

```
...  
BEGIN  
...  
initial(mutex,1);  
...  
END;
```

# Semáforo

- Implementação de semáforo no Pascal-FC

- Operação WAIT

- Utilizada pelos processos Pascal-FC para ter acesso à região crítica controlada pelo semáforo
- Se o semáforo possui valor igual a 0, o processo fica bloqueado... Se possui valor maior que 0, o processo entra em sessão crítica e decrementa o valor da variável;

```
...  
BEGIN  
...  
wait(mutex);  
...  
END;
```

# Semáforo

- Implementação de semáforo no Pascal-FC

- Operação SIGNAL

- Utilizada pelos processos Pascal-FC para liberar o acesso à região crítica controlada pelo semáforo
- Após a liberação de um semáforo através da procedure SIGNAL, um ou mais processos que aguardavam a liberação podem entrar em sua sessão crítica (procedure WAIT);

```
...  
BEGIN  
...  
signal(s);  
...  
END;
```



# Semáforo

## ● Exemplo 1

```
PROGRAM Concorre;  
VAR mutex : semaphore;  
PROCESS Type TP(n : integer);  
BEGIN  
  WHILE (TRUE) DO  
  BEGIN  
    sleep(random(10));  
    wait(mutex);  
    writeln('Processo ',n,' executando!!!');  
    signal(mutex);  
  END;  
END;  
  
VAR  
Proc : ARRAY[1..2] OF TP;  
I : integer;  
BEGIN  
  initial(mutex,1);  
  COBEGIN  
    FOR I := 1 TO 2  
    DO Proc[I](I);  
  COEND;  
END.
```

# Semáforo

## ● Exemplo2

```
PROGRAM atualiza;  
VAR  
  conta : integer;  
  mutex : semaphore;  
PROCESS P1;  
VAR I : integer;  
BEGIN  
  FOR I := 1 TO 20 DO  
  BEGIN  
    wait(mutex);  
    conta := conta + 1;  
    signal(mutex);  
  END;  
END; (* P1 *)
```

# Semáforo

## ● Exemplo 2

```
PROCESS P2;  
VAR  
I : integer;  
BEGIN  
FOR I := 1 TO 20 DO  
BEGIN  
wait(mutex);  
conta := conta + 1;  
signal(mutex);  
END;  
END; (* P2 *)
```

# Semáforo

- Exemplo 2

```
BEGIN  
conta := 0;  
initial(mutex,1);  
COBEGIN  
P1;  
P2  
COEND;  
Writeln('Contagem total: ',conta)  
END.
```

# Semáforo

## ● Exemplo 3

```
PROGRAM ordena;  
VAR  
  vetor : ARRAY[1..10] of integer;  
  mutex : semaphore;  
  
PROCESS Type OrdProc(pid : INTEGER);  
VAR  
  I, BUFFER : integer;  
  ORD : BOOLEAN;  
BEGIN  
  REPEAT  
    SLEEP(5);  
    wait(mutex);  
    ORD := FALSE;
```

# Semáforo

## ● Exemplo 3

```
FOR I := 1 TO 9 DO  
  IF vetor[i] > vetor[i+1]  
  THEN  
    BEGIN  
      ORD := TRUE;  
      BUFFER := vetor[i];  
      vetor[i] := vetor[i+1];  
      vetor[i+1] := buffer  
    END;  
    signal(mutex);  
  UNTIL NOT ORD;  
END;
```



# Semáforo

## ● Exemplo 3

```
VAR
J : INTEGER;
P : ARRAY[1..15] OF OrdProc;
BEGIN
  initial(mutex,1);
  FOR J:= 1 TO 10
  DO
  BEGIN
    vetor[J] := random(15);
    write(vetor[J], ' ');
  END;
  COBEGIN
  FOR J := 1 TO 15
  DO P[J](J);
  COEND;
  writeln;
  FOR J:= 1 TO 10
  DO write(vetor[J], ' ')
  END.
```

# Semáforo

- **Resolvendo problemas**

- **Folha de pagamento**

- **Fazer uso de semáforo para desenvolver um sistema de folha de pagamento. O sistema deve ser composto de processos leitores (3), calculadores (3) e escritores (2), que acessam o cadastro dos funcionários para manipular os dados e imprimir a folha de pagamento;**
    - **Processos leitores apenas acessam a ficha dos funcionários e transmitem os dados para um buffer de entrada do sistema, o qual será lido por um dos processos calculadores;**

# Semáforo

- **Resolvendo problemas**

- **Folha de pagamento**


- **Processos calculadores realizam a leitura do cadastro do funcionário, calculam o salário bruto do funcionário com base no número de horas trabalhadas e o salário líquido com base no desconto a aplicar... Finalmente, armazenam o resultado (cadastro completo) num buffer de saída;**
- **Processos escritores nada mais fazem do que imprimir a folha de pagamento a partir das informações do cadastro do funcionário, armazenadas num buffer de saída.**

# Semáforo

- **Resolvendo problemas**

- **Folha de pagamento**

- **O sistema vai fazer uso de 5 variáveis semáforo:**

- ***SvagaIN* e *SdadosIN* para controlar, respectivamente, o acesso em escrita e leitura do buffer de entrada;**
        - ***SvagaOUT* e *SdadosOUT* para controlar, respectivamente, o acesso em escrita e leitura do buffer de saída;**
        - **Tela para controlar o acesso à impressão na tela pelos diversos processos**
- 

# Semáforo

- **Resolvendo problemas**

- **Folha de pagamento**

- **Três tipos de processos serão definidos:**

- *TPleitor, TPescritor e TPcalculador;*

- **Estrutura que representa o cadastro do funcionário será um registro (RECORD) do Pascal;**

- **Acesso em leitura e escrita nos buffers será implementado por procedimentos Pascal que serão ativados pelos processos.**

# Semáforo

## ● Exemplo Folha de Pagamento

```
program FolhaPag;  
type tpFunc = record  
  nFunc : integer;  
  VHora : real;  
  horasT : integer;  
  SalBruto : real;  
  Desc : real;  
  SalLiq : real;  
  NLeitor : integer;  
  NCalc : integer;  
end;  
  
var  
  bufferIN, bufferOUT : tpFunc;  
  SvagaIN, SdadosIN, SvagaOUT, SdadosOUT, tela : semaphore;  
  
procedure INcoloca (var dados1 : tpFunc);  
begin  
  wait(SvagaIN); bufferIN := dados1; signal(SdadosIN);  
end;
```



# Semáforo

## ● Exemplo Folha de Pagamento

```
procedure INbusca (var dados2 : tpFunc);  
begin  
wait (SdadosIN); dados2 := bufferIN; signal(SvagaIN);  
end;  
  
procedure OUTcoloca (var dados3 : tpFunc);  
begin  
wait(SvagaOUT); bufferOUT := dados3; signal(SdadosOUT);  
end;  
  
procedure OUTbusca (var dados4 : tpFunc);  
begin  
wait (SdadosOUT); dados4 := bufferOUT; signal(SvagaOUT);  
end;
```

# Semáforo

- **Exemplo Folha de Pagamento**

```
process type tpLeitor ( pid: integer );  
Var data : tpFunc;  
begin  
  repeat  
    sleep ( random ( 10 ) + 5 );  
    data.NLeitor := pid;  
    data.NFunc := random (900);  
    data.VHora := random ( 90 )+ 30.0;  
    data.horasT := random ( 90 ) + 19;  
    INcoloca ( data );  
  forever;  
end;
```

# Semáforo

## ● Exemplo Folha de Pagamento

```
process type tpCalculador ( pid: integer );  
var data1 : tpFunc;  
begin  
  repeat  
    sleep ( random ( 10 ) + 5 );  
    INbusca ( data1 );  
    data1.NCalc := pid;  
    data1.SalBruto := data1.VHora*data1.horasT;  
    data1.Desc := data1.SalBruto/10;  
    data1.SalLiq := data1.SalBruto - data1.Desc;  
    OUTcoloca ( data1 );  
  forever;  
end;
```

# Semáforo

## ● Exemplo Folha de Pagamento

```
process type tpEscrivor ( pid: integer );  
var  
data2 : tpFunc;  
begin  
repeat  
OUTbusca ( data2 );  
sleep ( random ( 10 ) + 2 );  
wait ( tela );  
write(data2.NLeitor,data2.NCalc,pid, ' - ');  
write(data2.NFunc:3, ' - R$/h: ',data2.Vhora:3:2,' - Horas: ', data2.horasT:3);  
writeln(' - SB: ',data2.SalBruto:6:2,' - D: ', data2.Desc:6:2, ' - SL: ', data2.SalLiq:6:2);  
signal ( tela );  
  
forever;  
end;
```

# Semáforo

## ● Exemplo Folha de Pagamento

```
var
leitor1, leitor2, leitor3 : tpLeitor;
calc1, calc2, calc3 : tpCalculador;
escr1, escr2 : tpEscritor;

(*=====*)
begin
initial ( tela , 1 );
initial ( SdadosIN , 0 );
initial ( SvagaIN, 1 );
initial ( SdadosOUT , 0 );
initial ( SvagaOUT, 1 );
cobegin
leitor1( 1 ); leitor2( 2 ); leitor3( 3 );
calc1( 1 ); calc2( 2 ); calc3( 3 );
escr1(1); escr2(2);
coend;
end.
```

# Semáforo

- **Resolvendo problemas**

- **Pombos Correio**

- Desenvolvimento de um programa que represente a situação em que diversos pombos correio transportar cartas de uma caixa postal e são acomodados em uma gaiola após terem realizado o transporte das cartas
    - Cada pombo só realiza a viagem quando existem 5 cartas na caixa postal;
    - Cada pombo é liberado da gaiola quando ela contém três pombos... O tratador é responsável pela liberação.



# Semáforo

- Resolvendo problemas

- Pombos Correio

- 5 semáforos:

- spombo (liberação do pombo para a viagem);
      - vpombo (controle de acesso à variável *npombos*);
      - mutex (controle de acesso à variável *cxpostal*)
      - gaiola (controle de acesso à gaiola);
      - tela (controle de acesso à impressão na tela).

# Semáforo

## ● Exemplo Folha de Pagamento

```
program pombos;  
var  
  cxpostal,npombos: integer;  
  mutex, spombo,vpombo, gaiola , tela: semaphore;  
  process type tppessoa ( pid : integer );  
begin  
  repeat  
    sleep ( random ( 10 ) + 3 );  
    wait ( mutex );  
    cxpostal := cxpostal + 1;  
    if ( ( cxpostal mod 5 ) = 0 ) then  
      begin  
        wait ( tela );  
        writeln ( pid:3, ' Acorda Pombo, saldo: ', cxpostal:3 );  
        signal ( tela );  
        signal ( spombo );  
      end;  
    signal ( mutex );  
  forever  
end;
```

# Semáforo

## ● Exemplo Folha de Pagamento

```
process type tppombo ( pid: integer );  
begin  
repeat  
wait ( spombo );  
wait ( mutex );  
cxpostal := cxpostal - 5;  
wait ( tela );  
writeln ( 'POMBO: ',pid:3, ' A -----> B saldo: ', cxpostal:3 );  
signal ( tela );  
signal ( mutex );  
sleep ( random ( 9 ) + 5 );  
wait(vpombo);  
npombos := npombos + 1;  
signal(vpombo);  
wait ( tela );  
writeln ( 'POMBO: ',pid:3, 'ENTROU NA GAIOLA - POMBOS NA GAIOLA: ',npombos);  
signal(tela);  
wait(gaiola);  
wait ( tela );  
writeln ( 'POMBO: ',pid:3, ' B -----> A');  
signal ( tela );  
sleep ( random ( 14 ) + 5 );  
forever;  
end;
```

# Semáforo

## ● Exemplo Folha de Pagamento

```
process type tptratador ( pid: integer );  
var  
i : integer;  
begin  
repeat  
sleep(15);  
wait(vpombo);  
if npombos >= 3  
then  
begin  
for i:= 1 to npombos  
do signal(gaiola);  
wait(tela);  
write('Processo Tratador liberados ',npombos,'!!!');  
writeln;  
signal(tela);  
npombos := 0;  
end;  
signal(vpombo);  
forever;  
end;
```

# Semáforo

## ● Exemplo Folha de Pagamento

```
var
p1, p2,p3,p4,p5 : tppombo;
tratador : tptratador;
pessoa : array [ 1..12 ] of tppessoa;
i: integer;
begin
cxpostal := 0;
npombos := 0;
initial ( mutex , 1 );
initial ( tela, 1 );
initial ( spombo, 0 );
initial (gaiola ,0);
initial (vpombo, 1);
cobegin
p1( 1 ); p2 ( 2 ); p3(3); p4(4); p5(5);
tratador(1);
for i := 1 to 12 do pessoa [ i ] ( i+10 );
coend;
end.
```