Sistemas Concorrentes e Distribuídos

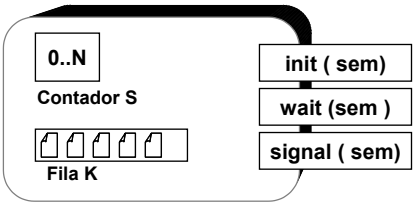
Semáforos

Sincronização - Soluções por Software

Semáforos

- mecanismo proposto em 1967 por Dijkstra
- recurso acessado somente por funções específicas: wait e signal
- wait e signal são atômicos
- o semáforo deve ser compartilhados por todos os processos que vão utilizá-lo como mecanismo para exclusão mútua ou sincronização

Sincronização - Soluções por Software



- **wait (sem) : s:= s - 1**
  - **se sem >= 0 então processo prossegue**
  - **se sem < 0 então processo é bloqueado na fila de espera K**
- **signal (sem) : s:= s + 1**
  - **se houver processo bloqueado na fila de espera, *um deles é escolhido* para se desbloquear e prosseguir**

- **operações também chamadas de P e V por motivos históricos**

Implementação de semáforos:

- **Definir um semáforo como um "record":**

```
type semaphore = record
                 value: integer;
                 L: list of process;
                 end;
```

- **Oferecer duas operações simples:**
  - **"block" suspende o processo que a invocou.**
  - **"wakeup(*P*)" reinicia a execução de um processo "P" bloqueado.**

Implementação (Cont.)

- As operações dos semáforos agora serão definidas como:

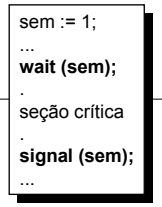
```
wait(S):   S.value := S.value – 1;
           if S.value < 0
             then begin
                    adiciona esse processo à S.L;
                    block;
                  end;
signal(S): S.value := S.value + 1;
           if S.value ≤ 0
             then begin
                    remove o processo P de S.L;
                    wakeup(P);
                  end;
```


Dois tipos de semáforos

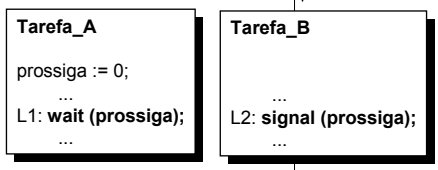
- *Counting* semaphore (semáforo contador) – valor inteiro que pode variar por uma faixa de valores sem restrição.
- *Binary* semaphore (semáforo binário) – valor inteiro que varia somente entre 0 e 1; pode ser simples de se implementar.
- Podemos implementar um semáforo contador como um semáforo binário.

Exemplos

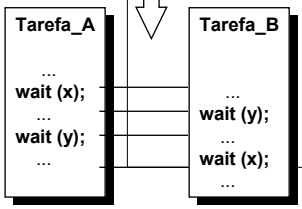
- **exclusão mútua**



- **sincronização**
  - **tarefa A espera em L1: até que tarefa B atinja ponto L2:**



- **situação de deadlock**



Problema I: Produtores/Consumidores

- **vários processos:**
  - **produtores**
  - **consumidores**
- **cada processo tem seu ritmo (nada é assumido sobre o tempo de execução**
- **produtor**
  - **produz item e coloca em um buffer**
- **consumidor**
  - **pega item no buffer e consome**
- **buffer**
  - **infinitos**
  - **finitos (circular)**

Produtor/Consumidor Buffer Ilimitado

- **proposta da turma**
- **dica**
  - **consumidor só é liberado para pegar item depois que o produtor colocar o item no buffer**
  - **um semáforo**
  - **um ponteiro para a posição do produtor na fila**
  - **um ponteiro para a posição do consumidor na fila**

Exemplos

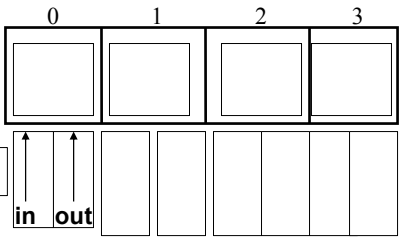
Bounded Buffer
Solução "Ad Hoc"

```
Module Example ( );
type item : ...;
var buffer: array [ 0..3 ] of item;
      in, out : 0..3;
      nextp, nextc: item;
```

```
task producer: loop
  ...
  produz um item em netxp
  ...
  while (in + 1) mod 4 = out do nothing;
  ...
  buffer [in] := nextp;
  in := (in + 1) mod 4;
endloop;
```

```
task consumer: loop
  ...
  while in = out do nothing;
  nextc := buffer [out];
  out := (out + 1) mod 4;
  ...
  consome o item em netxp
  ...
endloop;
```

```
Begin
   in := 0;
   out := 0;
endModule;
```



Solução com semáforo de contagem

```
program BufferLimitado;
const
      TamFila = ...;
var
      Fila : array [1..TamFila] of TipDado;
      Item : semaphore;
      Vaga: semaphore;
```

```
procedure Consumidor;
var
      IniFila : (1..TamFila);
      W : tipDado;
begin
      IniFila := 1;
      repeat
            wait (Item);
            W := Fila [ IniFila ] ;
            IniFila := (IniFila mod TamFila) + 1;
            signal (Vaga);
            Consumir (W);
      until Fim;
end;
```

```
procedure Produtor;
var
      FimFila : (1..TamFila);
      V : tipDado;
begin
      FimFila := 1;
      repeat
            Produzir (V);
            wait (Vaga);
            Fila [ FimFila ] := V;
            FimFila := (FimFila mod TamFila) + 1;
            signal (Item);
      until Fim;
end;
```

```
Begin
      InitSem (Item, 0);
      InitSem (Vaga, TamFila);
      cobegin
            Produtor; Consumidor;
      coend;
End.
```


Exemplos
Bounded Buffer
Solução com Semáforos

```
Module Example;
type item : ...;
var buffer: array [0.. n-1] of item;
      full, empty, mutex: semaphore;
      nextp, nextc: item;
      pontP, pontC: ponteiros;
```

```
task producer: loop
   produz um item em netxp
   ...
   wait (empty);
   wait (mutex);
   ...
   adiciona nextp ao buffer;
   ajusta PontP;
   ...
    signal (mutex);
    signal (full);
endloop;
```

```
task consumer: loop
   ...
   wait (full);
   wait (mutex);
    ...
    remove um item do buffer
    ajusta pontC
    ...
    signal (mutex);
    signal (empty);
   ...
   consome o item em netxp
endloop;
```

```
Begin
   full := 0;       pontC = 0;
   empty := n;   pontP = 0;
   mutex := 1;
endModule;
```

funciona para qualquer estrutura de dados para o buffer

- mutex: exclusão mútua
- full > 0 existe algum item produzido
- empty > 0 existe espaço vazio no buffer