Programação Concorrente(3)

José C. Cunha, DI/FCT/UNL

- Semáforos de sincronização (E.W.Dijkstra)
- As realizações de semáforos e o problema do escalonamento de processos pelo Sistema de Operação

Semáforos (E.W.Dijkstra)

- **Um tipo de dados abstracto caracterizado por:**

-- um valor inteiro não negativo (S)

-- três operações indivisíveis:

- **Inicializar(S):**
- **P(S): o processo invocador bloqueia-se até que S>0 e, então, decrementa S de uma unidade**

P = ´´proberen´´: ´´testar´´, em holandês.

- **V(S): o processo invocador incrementa S de uma unidade e prossegue sem se bloquear**

V = ´´verhogen´´: ´´incrementar´´

Na definição de semáforos

-- Nada é dito sobre a realização de qual o processo se deve escolher, quando se faz V(), nem sobre qual é a ordem de execução dos processos.

-- Assim: as únicas hipóteses a assumir devem apenas basear-se na definição abstracta de Dijkstra.

Realização de Semáforos

- **Cada Semáforo é representado por:**

-- uma variável inteira (valor S)

-- uma fila de espera de processos bloqueados f(S)

- **Quando há processos concorrentes e 1 CPU:**

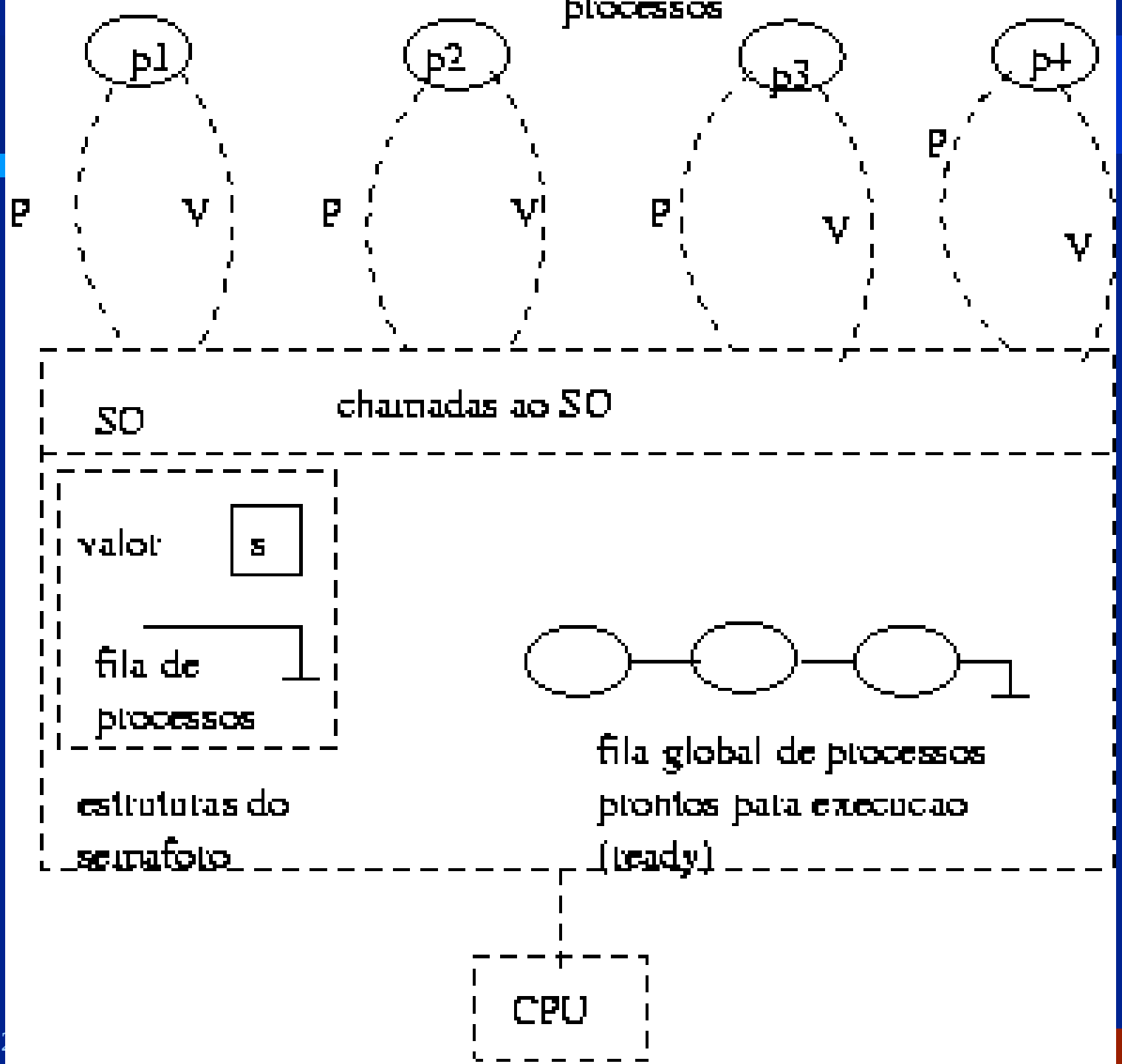
-- um no estado EXECUTANDO;

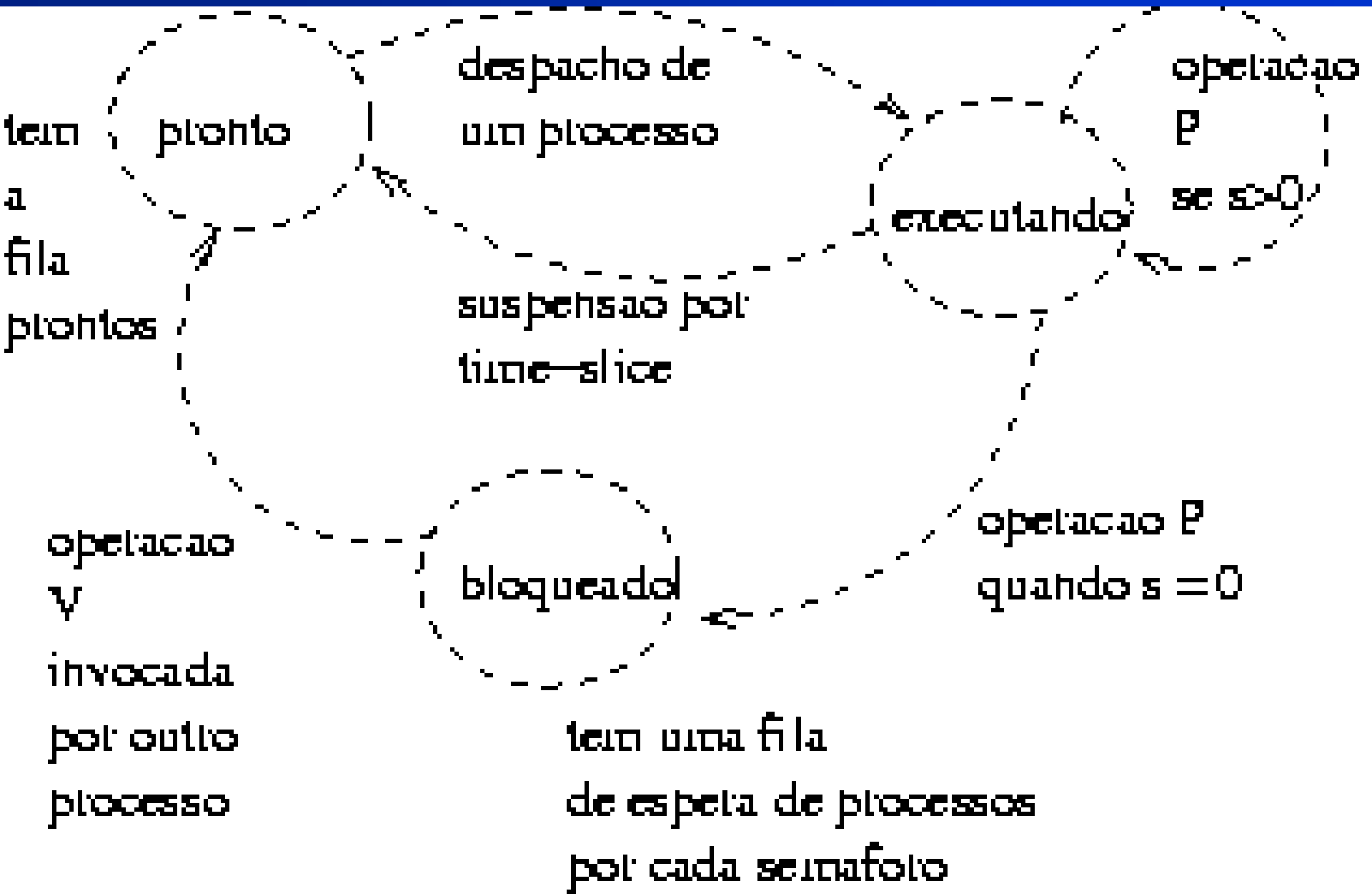
-- outros: PRONTOS, mas aguardam a sua vez;

estão numa fila de processos prontos (READY);

-- se um P(S) bloqueia: o processo passa a BLOQUEADO e fica na FILA f(S)

-- se um processo faz V(S): pode permitir que um outro processo seja DESBLOQUEADO e saia da fila f(S)


processos
p1
p2
p3
p4
P
V
P
V
P
V
P
V
SO
chamadas ao SO
valor
s
fila de
processos
estrututas do
semafoto
fila global de processos
prontos para execucao
(ready)
CPU


tem
a
fila
prontos
pronto
despacho de
um processo
executando
operacao
P
se s>0
suspensao por
time-slice
operacao
V
invocada
por outro
processo
bloqueado
operacao P
quando s =0
tem uma fila
de espera de processos
por cada semaforo

Uma realização - 1

- **Há várias realizações possíveis:**

-- diferem no modo como as operações P e V são realizadas

- **A seguir vêem-se diversas soluções.**
- **Todas as realizações devem cumprir a definição abstracta, devida a Dijkstra:**

V. Inicial + Nº ops V >= Nº ops P completadas

```
P(S): (*operação indivisível*)
 if S>0 then S = S-1
 else
    begin (*bloqueia este processo*)
      salvaguarda estado/regs da máquina;
      estado(processo) = BLOQUEADO;
      inserir(processo, fila f(S));
      rotina-despacho-SO escolhe outro processo
      da fila PRONTOS, para execução
    end
```

```
V(S): (*operação indivisível*)
   if (fila f(S) VAZIA) then S = S+1
   else
       begin (*desbloqueia um processo*)
          remover(fila f(S), Proc);
          inserir(Proc, fila PRONTOS);
          estado(Proc) = PRONTO
       end
```

Comentário

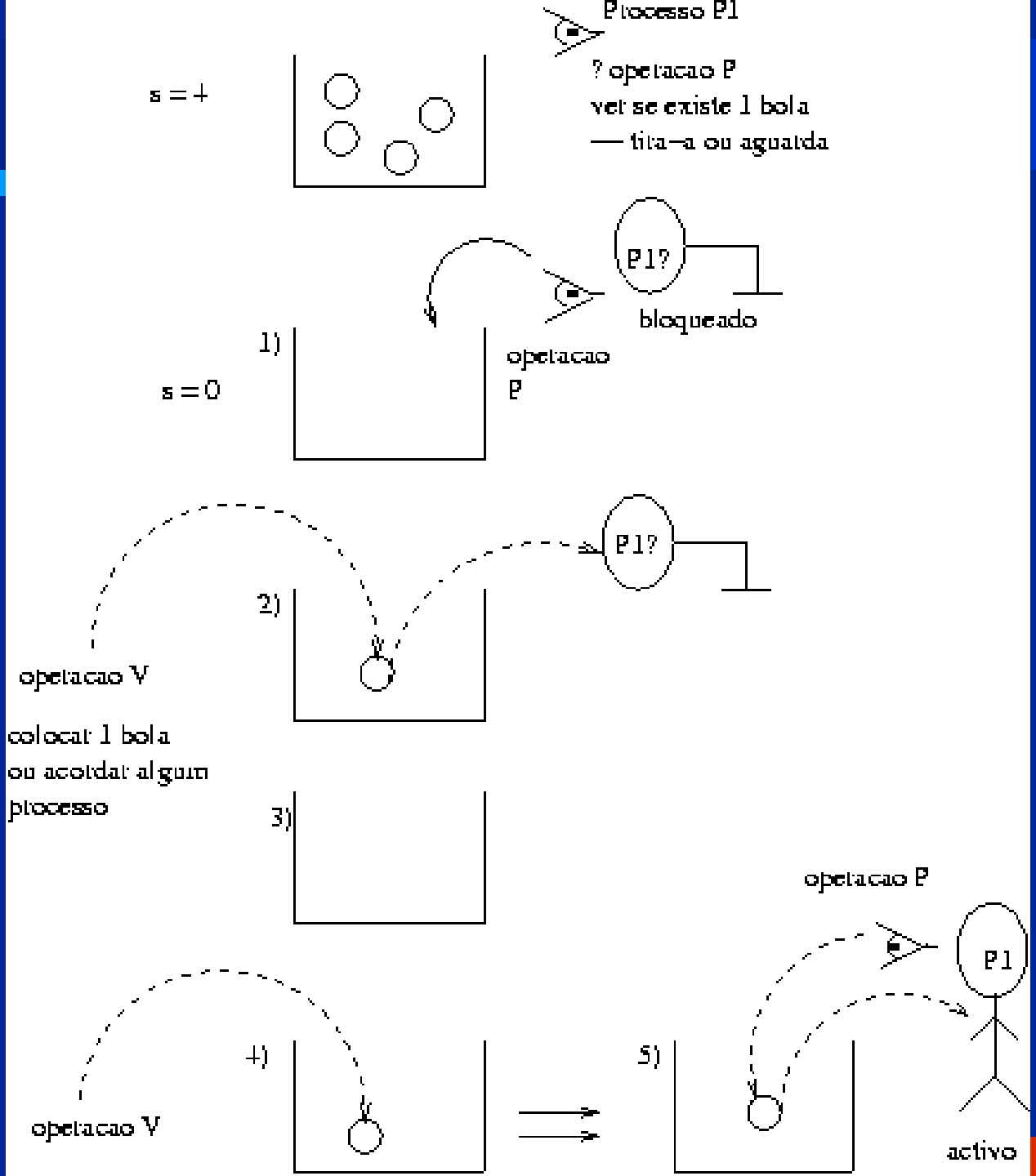
- **Na realização apresentada, um processo, uma vez bloqueado em P(), não voltará a testar o valor do semáforo, quando for desbloqueado. É como se o valor do PC (Program counter) salvaguardado qando se bloqueou, fosse o endereço de retorno ao programa que invocou P() como uma subrotina.**
- **De forma consistente com isto, o processo que fizer V() só incrementa o valor do semáforo, no caso de a fila de processos estar vazia; caso contrário, apenas desbloqueia um dos processos que esteja bloqueado.**

Comentário

- **Esta realização tem uma característica "interessante":**

-- havendo um ou mais processos bloqueados num semáforo, num dado momento, quando um outro processo fizer V(), um daqueles processos será desbloqueado e poderá prosseguir a execução, logo que o SO assim o seleccionar para execução, e outros processos que façam P() logo a seguir ao V(), não poderão completar as suas operações P antes do processo desbloqueado

- **Em geral as realizações não garantem esta propriedade.**


Processo P1
? operacao P
vet se existe 1 bola
— tira-a ou aguarda
s = +
P1?
bloqueado
1)
s = 0
operacao
P
P1?
2)
operacao V
colocar 1 bola
ou acordar algum
processo
3)
operacao P
P1
4)
operacao V
5)
activo

Regiões Críticas – Exclusão mútua

Basta inicializar um Semáforo exmut
com valor igual a 1
no início a fila f(exmut) está VAZIA.
e cada processo deve fazer:

P(exmut);

Região Crítica;

V(exmut)

P1 (inicio: exmut=1) P2

```
WHILE true DO                WHILE true DO
{ P(exmut);                  { P(exmut);
  Região Crítica;              Região Crítica;
  V(exmut);                    V(exmut)
}                            }
```

- Um cenário possível (entre outros…):

P1:P()-> exmut=0; P1:RegiãoCrítica;

P1: *Time-slice*; → ... SO comuta para processo P2

P2:P()->bloqueia:P2; …?…; P1: RegiãoCrítica; →

P1:V()→desbloqueia:P2, mas SO continua P1; →

Se P1:P() → bloqueia P1; → P2:RegiãoCrítica; …..

- E se tiver mais do que 2 processos?
- Em geral não posso afirmar nada sobre a ordem de execução dos processos:

-- o SO faz a multiprogramação de forma imprevisível

-- a operação V() escolhe **um qualquer** dos processos bloqueados na fila (conforme a sua realização)

- Mas, no exemplo: tenho a garantia de exclusão mútua, seja qual for a ordem de execução
- **Conclusão: a Correcção dos nossos programas NÃO PODE DEPENDER de hipóteses sobre a ORDEM de EXECUÇÃO.**

Outra realização - 2

- **Mantém a mesma ideia da anterior mas difere na forma como a realiza.**

Outra realização - 2

- **Mantém a mesma ideia da anterior mas difere na forma como a realiza.**

- **NOTA IMPORTANTE: esta implementação vai gerir valores negativos do semáforos, mas só internamente: a definição de semáforo, para o programador, mantém-se inalterada.**

```
P(S): (*operação indivisível*)
  S = S-1;  (*decrementa SEMPRE!!!*)
  if S< 0 then
    (*mas só prossegue se não deu valor negativo*)
     begin (*bloqueia este processo*)
       salvaguarda estado/regs da máquina;
       estado(processo) = BLOQUEADO;
       inserir(processo, fila f(S));
       rotina-despacho-SO escolhe outro processo
       da fila PRONTOS, para execução
     end
```

```
V(S): (*operação indivisível*)
   S = S+1;  (* incrementa SEMPRE*)
  if S <= 0 then
    (* se o valor, após incrementado, não é positivo
     então é porque há lá processos bloqueados!*)
      begin (*desbloqueia um processo*)
         remover(fila f(S), Proc);
         inserir(Proc, fila PRONTOS);
         estado(Proc) = PRONTO
      end
```

Exemplo1 da realização 2

1. Inicializar semáforo exmut a 1
2. P1 tenta P(): fica exmut = 0 e P1 continua
3. P2 tenta P(): fica exmut = -1 e P2 bloqueia
4. P3 tenta P(): fica exmut = -2 e P3 bloqueia
5. P1 faz V(): fica exmut == -1
   e um dos processos P2 ou P3 é desbloqueado

Conclusão: o valor absoluto do semáforo, se for negativo, indica quantos processos estão lá bloqueados

Exemplo2 da realização 2

1. **Inicializar semáforo exmut a 1**
2. **P1 tenta P(): fica exmut = 0 e P1 continua**
3. **P2 tenta P(): fica exmut = -1 e P2 bloqueia**
4. **P1 faz V(): fica exmut=0 e P2 é desbloqueado, mas P1 continua**
5. **P1 tenta P() de novo: fica exmut = -1 e P1 bloqueia**
6. **Quando P2 for escolhido pelo SO, P2 continua a execução**
7. **P2 faz V(): fica exmut=0 e P1 é desbloqueado.**

Outra realização - 3

P(S): (*operação indivisível: ou consegue executar todas as acções de uma só vez ou então não executa nenhuma e fica a aguardar*)

```
if S > 0 then S = S-1
else
  begin
   (* bloqueia este processo  e só quando puder
       é que decrementa *)
aguardar (bloqueado) até que S>0 e então S  = S-1
   (*entretanto, o SO escolhe outro processo*)
  end
```

- Nota: o processo só completa o P() e prossegue a execução depois de, tendo encontrado o valor de S>0, conseguir decrementá-lo.

```
V(S): (*operação indivisível*)
  S = S+1;  (* incrementa SEMPRE*)
  if (fila f(S) NÃO VAZIA) then
       begin (*desbloqueia um processo*)
          remover(fila f(S), Proc);
          inserir(Proc, fila PRONTOS);
          estado(Proc) = PRONTO
       end
```

Exemplo da realização 3

1. Inicializar semáforo exmut a 1
2. P1 tenta P(): fica exmut = 0 e P1 continua
3. P2 tenta P(): exmut = 0 e P2 bloqueia
4. P3 tenta P(): exmut = 0 e P3 bloqueia
5. P1 faz V(): fica exmut=1 e um dos processos P2 ou P3 é desbloqueado mas apenas um deles consegue decrementar o exmut:

   por exemplo: se P2 for desbloqueado, fica exmut=0 e P3 permanece bloqueado.

Comentário sobre realização 3

- Admitir exmut = 0 e dois processos P2 e P3 bloqueados no semáforo
- Admitir que P1 faz V():

-- incrementa: fica exmut = 1

-- desbloqueia P2 ou P3: admitir que é P2 mas este vai ter de retentar o teste de exmut...

A realização não se compromete que, ao desbloquear P2, este consiga ser o primeiro processo a decrementar o semáforo e ficar exmut = 0…

Comentário

- **Na prática, não podemos assumir a garantia de que o processo P2, no exemplo anterior, não possa ser ultrapassado por outro processo que seja escolhido pelo SO em primeiro lugar para execução e que consiga encontrar o valor de exmut = 1 (mas nesse caso: P2 manter-se-á bloqueado até que consiga completar o P())**
- **A ordem pela qual os processos são escalonados para execução depende do SO e da realização das operações P e V.**
- **Conclusão: não devemos assumir nada sobre a ordem pela qual os processos conseguem completar as operações P.**

Realização no Linux

- **Operação P(): como a realização 3**
- **Operação V(): S = S+1; (* incrementa SEMPRE*) e é-nos dito que isto permitirá que um dos processos bloqueados venha a conseguir completar o P()**
- **Deixa ao cuidado do SO a decisão de qual dos processos, quer já esteja bloqueado num P(), quer esteja a tentar iniciar um P(), consegue decrementar o valor S e continuar.**
- **Podem ocorrer situações de "injustiça" ('unfairness')**

P1 (inicio: exmut=1) P2

```
WHILE true DO                WHILE true DO
{ P(exmut);                  { P(exmut);
  Região Crítica;              Região Crítica;
  V(exmut);                    V(exmut)
}                            }
```

- Um cenário possível (entre outros…):

P1:P(); P1:RegiãoCrítica; P1: Time-slice; →

P2:P()→bloqueia:P2; …?…; P1: RegiãoCrítica; →

P1:V()→exmut=1,permitindo que P2 completasse P()

mas SO continua P1; →

P1:P()→encontra exmut=1; → P1:RegiãoCritica…..

Revisão do conceito de semáforo

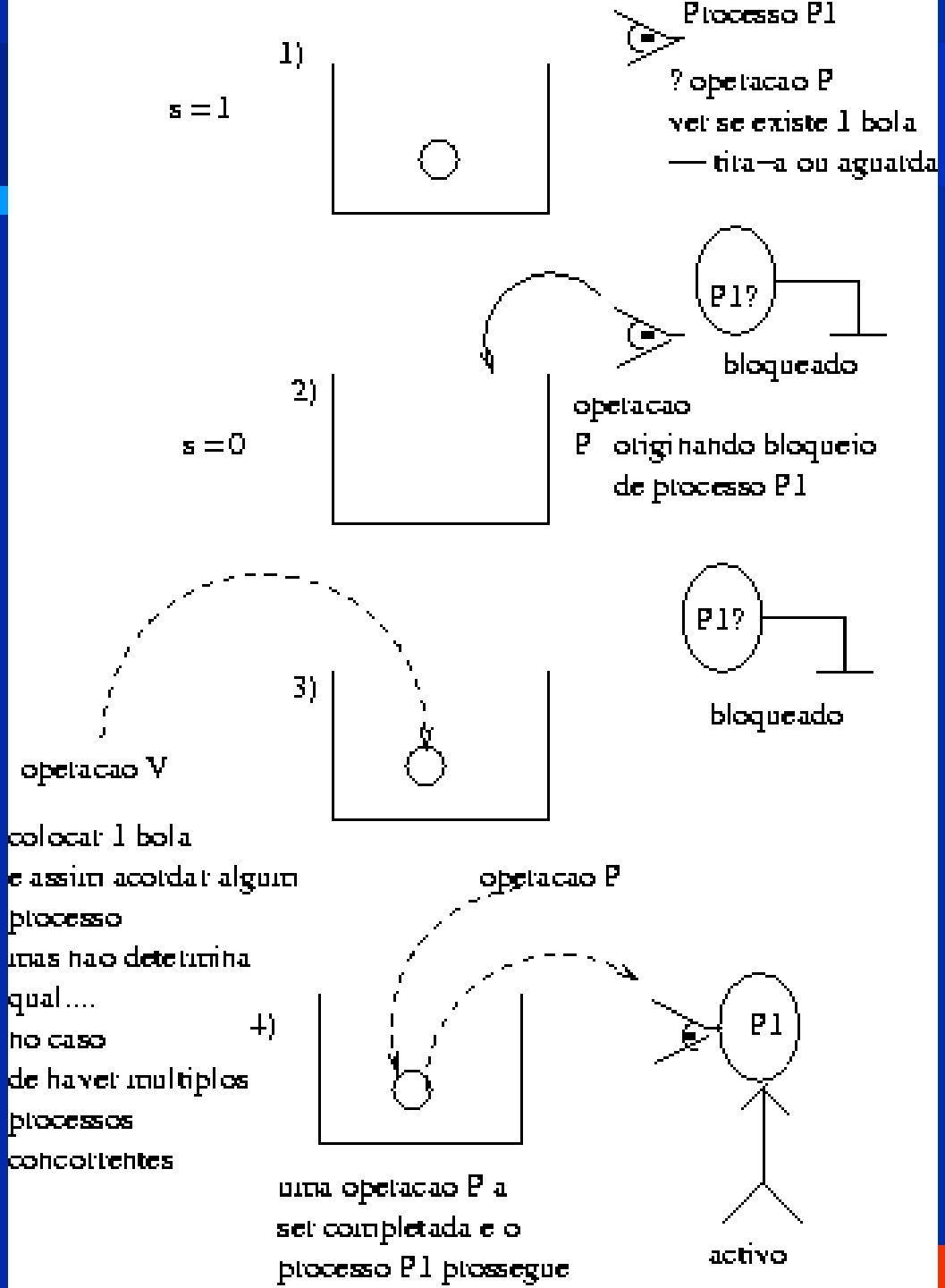
- **Um tipo de dados abstracto caracterizado por:**

-- um valor inteiro não negativo (S)

-- três operações indivisíveis:

- **Inicializar(S):**
- **P(S): o processo invocador bloqueia-se até que S>0 e, então, decrementa S de uma unidade**
- **V(S): o processo invocador incrementa S de uma unidade e prossegue sem se bloquear**

-- Nada mais é dito sobre a realização, nem sobre a ordem de execução dos processos.


Processo P1
1)
s = 1
? operacao P
vel se existe 1 bola
— tira-a ou aguarda
P1?
bloqueado
2)
s = 0
operacao
P originando bloqueio
de processo P1
P1?
bloqueado
3)
operacao V
colocar 1 bola
e assim acordar algum
processo
mas nao determina
qual....
no caso
de haver multiplos
processos
concorrentes
operacao P
4)
P1
uma operacao P a
ser completada e o
processo P1 prossegue
activo

Invariante do semáforo

s0 = Valor_inicial_do_semáforo

nV = número_de_operações_V_completadas

nP = número_de_operações_P_completadas

- **Antes e após ser completada qualquer das operações Inicializar(s) / P(s) / V(s), a seguinte expressão é sempre verdade:**

s0 + nV >= nP

- **Não é possível completar um número de operações P() superior à soma do valor inicial do semáforo com o número de operações V()**