Semáforos

Esquema de sincronização para Seção Crítica sem espera ocupada

Soluções para SC

- Algoritmo de Dekker, Peterson, Lamport, rápido, entre outros
  - Uso de instruções atômicas complexas pode facilitar
- Estas soluções garantem a exclusão mútua
  - Utilizam espera ocupada
- Problemas com espera ocupada
  - Uso do processador na sincronização e não na solução da tarefa
  - Pode dar origem ao problema da Prioridade Invertida
  - Processos de baixa prioridade bloqueiam de alta

Semáforos

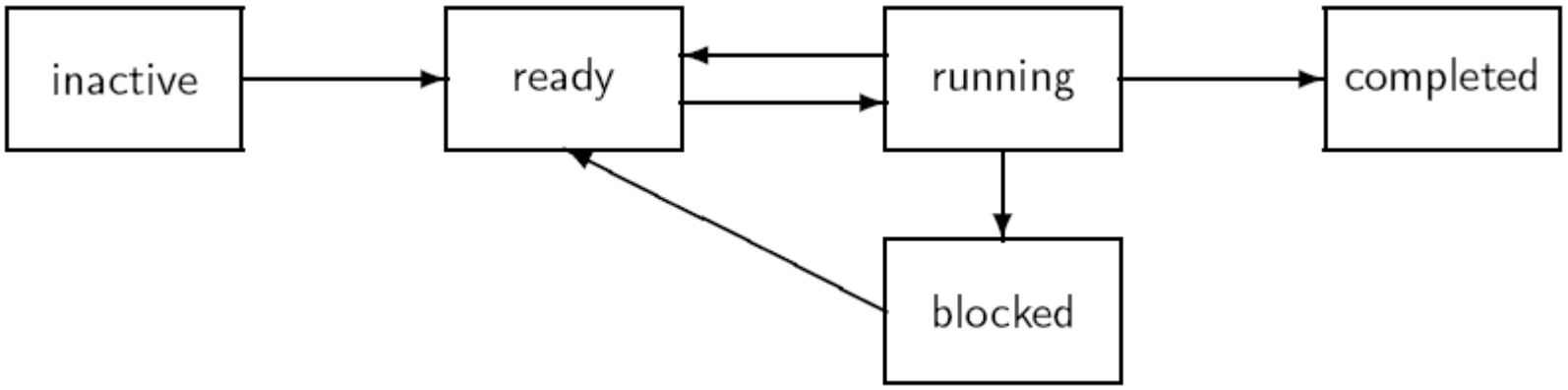
- Conceito introduzido por E. W. Dijkstra (1968)
- Simplificam os protocolos para sincronização
- Eliminam a necessidade de espera-ocupada (*busy-wait*)
- Um semáforo comporta-se como uma variável inteira, não negativa, na qual se pode fazer apenas duas operações (além da inicialização):
  - *P(s)=wait(s)*
  - *V(s)=signal(s)*
    - Curiosidade: em holandês:
      - *P = Probeer ('Try') e V = Verhoog ('Increment', 'Increase by one')*

Operações nos Semáforos

- *P(s)=wait(s)*
  - Se *s* > *0* decrementa *s*, senão suspende o processo
- *V(s)=signal(s)*
  - Se existe algum processo suspenso em *s*, libera um deles, senão incrementa *s*
    - Não é especificado qual processo é liberado
- As operações nos semáforos são atômicas

Ações dos Semáforos no S.O.

- Recursos do S.O. para resolver os problemas da espera ocupada
- O processo deve ser colocado no estado bloqueado
- Uma sinalização posterior coloca o processo novamente no estado pronto



Conforme M. Ben-Ari (2006) (Livro base)

- Semáforos são recursos do S.O. compostos basicamente por 2 campos de dados:
  - ***V*** número inteiro não negativo
  - ***L*** relação de processos que foram bloqueados neste semáforo

- Sobre um semáforo pode ser executadas 2 operações atômicas:
  - ***wait*** decrementa semáforo ou bloqueia processo
  - ***signal*** incrementa semáforo ou desbloqueia processo

Operações *Wait* e *Signal*

```
wait(S)
Se S.V>0
  S.V = S.V - 1
senão
  S.L = S.L U p
  p.estado = bloq.
```

```
signal(S)
Se S.L == ∅
  S.V = S.V + 1
Senão
  Escolhe q de S.L
  (arbitrário)
  S.L = S.L - {q}
  q.estado = pronto
```

Ao executar a operação *wait*:

- Caso o valor do semáforo seja maior do que zero
  - O valor é decrementado
  - O processo continua a execução normal
- Caso o valor seja zero
  - O processo que executou a operação *wait* é bloqueado
  - O valor continua em zero
  - O processo é acrescentado a relação (lista) de processos do semáforo
- Esta operação também é denominada de ***down*** ou ***P***

Ao executar a operação *signal*:

- Caso a relação de processos bloqueados no semáforo esteja vazia:
  - O valor do semáforo é incrementado
- Caso existam processos na relação (lista)
  - É escolhido arbitrariamente um deles
  - Retira-se o processo da relação do semáforo
  - O processo passa para o estado pronto
- Esta operação também é denominada de ***up*** ou **V**

Observações

- Semáforos que podem assumir qualquer valor positivo são denominados de **semáforos gerais**.

- Semáforos que podem assumir apenas os valores *0* e *1* são denominados de semáforos binários ou ***mutex***.

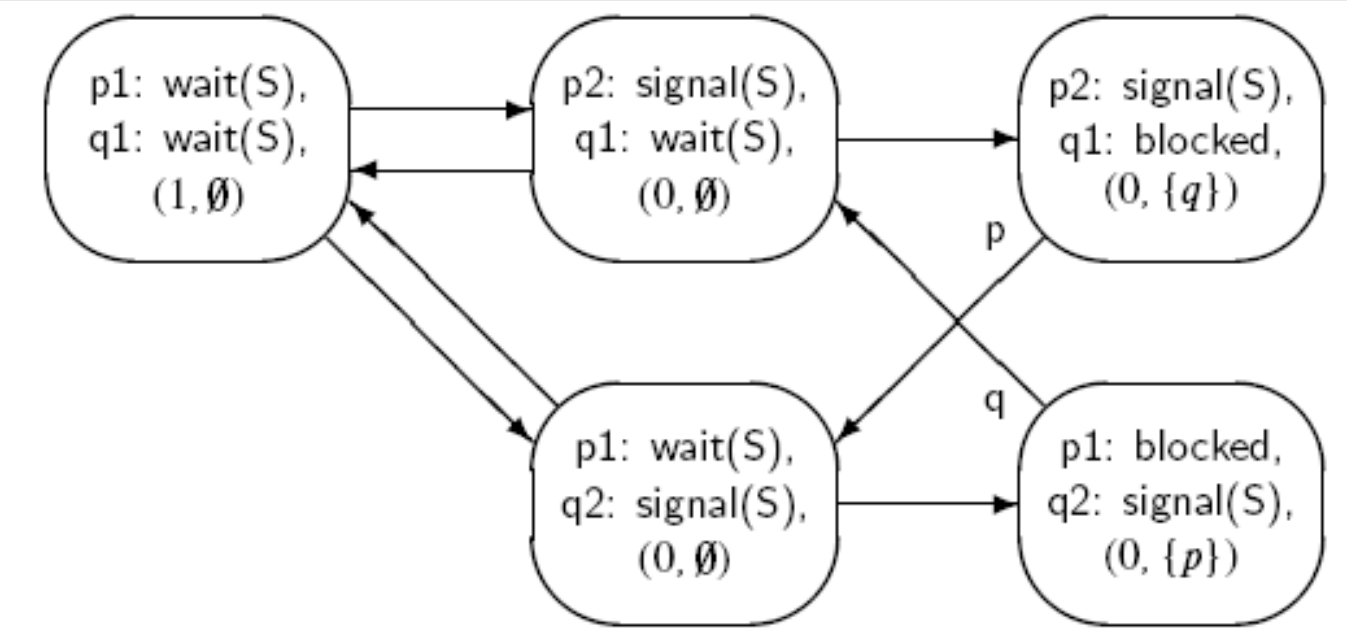
SC com semáforos e 2 processos

Algorithm 6.1: Critical section with semaphores (two processes)	
binary semaphore S ← (1, ∅)	
p	**q**
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: wait(S)	q2: wait(S)
p3: critical section	q3: critical section
p4: signal(S)	q4: signal(S)

Algoritmo Simplificado

Algorithm 6.2: Critical section with semaphores (two proc., abbrev.)	
binary semaphore S ← (1, ∅)	
p	**q**
loop forever	loop forever
p1: wait(S)	q1: wait(S)
p2: signal(S)	q2: signal(S)

Diagrama de estados (algoritmo simplificado)



Sem violação de exclusão mútua (não existe p2, q2, ...)
Livre de *Deadlock*
Livre de *Starvation*

Semáforos para N processos

Algorithm 6.3: Critical section with semaphores (N proc.)	
binary semaphore S $\leftarrow (1, \emptyset)$	
	loop forever
p1:	non-critical section
p2:	wait(S)
p3:	critical section
p4:	signal(S)