Sincronização - Semáforos (15.7)

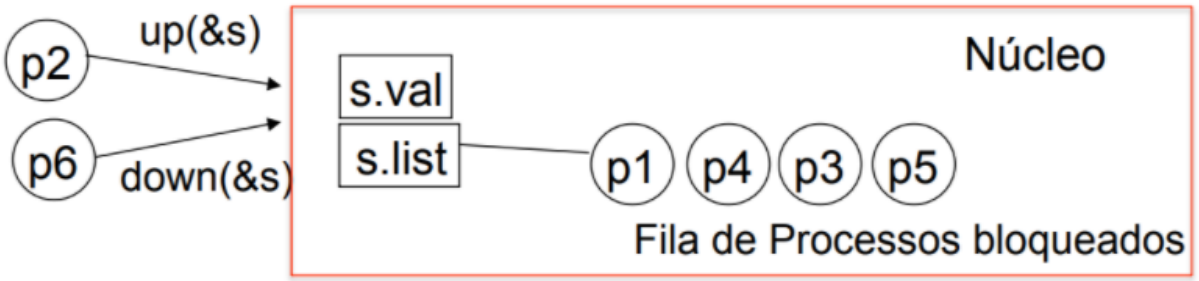
Semáforos

O semáforo é uma variável inteira que pode ser mudada por apenas duas operações primitivas (atômicas): P (lock) e V (unlock)

- P = proberen (testar) / down ou wait
  Nessa operação, se o semáforo for maior que “0”, o valor do semáforo é decrementado e se ele for igual a zero, o processo é bloqueado e inserido na fila de espera do semáforo.

- V = verhogen (incrementar) / up ou signal
  Numa operação V, o semáforo é incrementado e, um processo que aguarda na fila de espera deste semáforo é acordado.

Semáforos que assumem somente os valores 0 e 1 são denominados semáforos binários ou mutex.
A ideia do semáforo é como se ele fosse um estrutura de dados especial, contendo uma variável inteira e uma fila, entretanto os processos não podem consultar diretamente essa variável inteira e nem a fila, pois são implementadas como chamada de núcleo (system call) e para garantir a atomicidade durante a execução o núcleo desabilita temporariamente as interrupções.



O semáforo é utilizado em códigos para garantir a exclusão mútua em um processo P.

```
Processo P1     Processo P2     ...Processo Pn
...               ...               ...
P(mutex)          P(mutex)          P(mutex)
// R.C.           // R.C.           // R.C.
V(mutex)          V(mutex)          V(mutex)
...               ...               ...
```

Semáforos com Recursos compartilhados

Supondo que há 3 processos, o iniciado o semáforo com quantidade de 3 e ao utilizar um recurso, ocorre um down e o semáforo ganha o valor 2, simbolizando que ainda há espaço para mais dois processos, ou seja, mais dois processos podem acessar um recurso e quando um quarto processo vir a necessitar de um recurso, o semáforo irá conter o valor zero e o processo irá ser bloqueado até a liberação de algum processo anterior.

```
...
Semaphore S := 3;     /*var. semáforo, iniciado com
                      qualquer valor inteiro */

Processo P1          Processo P2          Processo P3
...                      ...                  ...
P(S)                     P(S)                 P(S)
//usa recurso        //usa recurso        //usa recurso
V(S)                     V(S)                 V(S)
...                      ...                  ...
```

Semáforos com Sincronização e relação de precedência

Utilizando semáforo para garantir precedência de alguma rotina, para que a rotina 1 rode primeiro que a rotina 2, para isso o semáforo tem que ser iniciado em 0.

Ex: Um problema em que o processo B depende a saída do processo A, então A deve ser ter precedência no programa em relação a B.

```
semaphore S := 0 ;
parbegin
   begin            /* processo P0*/
      p0_rot1()
      Up(S)
      p0_rot2()
   end
   begin            /* processo P1*/
      p1_rot1()
      Down(S)
      p1_rot2()
   end
parend
```

Caso o processo p1 ganhe a posse da CPU primeiro que o processo p0, ele vai executar a primeira rotina “p1_rot1()” e na sequência vai se bloquear até que o processo p0 ganhe a posse da CPU, e após executar a rotina “p0_rot1()”, o processo que executa ela vai fazer um Up sinalizando para os outros processos que essa rotina foi executada e se algum processo estiver esperando para rodar “p1_rot2()”, irá ser desbloqueado.
Se o processo p0 ganhar a posse da CPU primeiro,vai ser executado o processo “p0_rot1()” e em seguida irá fazer um Up, incrementando o semáforo em +1 e em algum momento, o p1 vai ganhar a posse da CPU e executar a rotina “p1_rot1()”, em seguida, e ao inves de bloquear, irá verificar se o semáforo possui um valor maior que zero, e se for verdadeiro, sera decrementado -1 ao semáforo e vai executar a rotina “p1_rot2()”.

Problema do Produtor-Consumidor

O problema descreve dois processos, o produtor e o consumidor, que compartilham um buffer de tamanho fixo. O trabalho do produtor é gerar um dado, colocá-lo no buffer e repetir essa operação indefinidamente. Ao mesmo tempo, a tarefa do consumidor é consumir tais dados (i.e. removendo do buffer), um de cada vez.
Devendo assim assegurar que o produtor não irá tentar adicionar dados no buffer quando este estiver cheio, e que o consumidor não tentará remover dados quando o buffer estiver vazio.
Ex: Produtor Consumidor c/ Buffer Circular

Variáveis : proxima_insercao, proxima_remocao.
Após o valor N-1 elas voltam a apontar para a entrada zero do vetor
% representa a operação “resto da divisão”

Três semáforos, duas funcionalidades diferentes:

Exclusão mútua (mutex) garante a exclusão mútua. Deve ser iniciado com “1”.

Sincronização.

Espera_dado: bloqueia o consumidor se o buffer está vazio. Iniciado com “0”.
Espera_vaga: bloquear produtor se o buffer está cheio. Iniciado com “N”

// não irá tentar adicionar dados no buffer quando este estiver cheio

```
void produtor(void){
...
down(espera_vaga);
down(mutex);
buffer[proxima_insercao] := dado_produzido;
proxima_insercao := (proxima_insercao + 1) % N;
up(mutex);
up(espera_dado);
... }
```

// não tentará remover dados quando o buffer estiver vazio.

```
void consumidor(void){
...
down(espera_dado);
down(mutex);
dado_a_consumir := buffer[proxima_remocao];
proxima_remocao := (proxima_remocao + 1) % N;
up(mutex);
up(espera_vaga);
... }
```

Deficiência dos Semáforos

Semáforos são uma abstração de alto nível baseada em primitivas de baixo nível, que provêm atomicidade e mecanismo de bloqueio, com manipulação de filas de espera e de escalonamento. Tudo isso contribui para que a operação seja lenta, para alguns recursos, isso pode ser tolerado; para outros esse tempo mais longo é inaceitável