Implementation of General Semaphores Using Binary Semaphores

Anthony Howe
June 18, 2001
The Problem

Can we implement a general semaphore from a binary semaphore?
Semaphores Review

- One of the first mechanisms proposed to handle inter-process synchronization
- A semaphore is an integer value that is only accessed through two atomic operations: wait and signal
- Busy-wait version

```plaintext
Wait(S):
    while S <= 0 do no-op;
    S := S - 1;

Signal(S):
    S := S + 1;
```
Semaphores Continued

- **Blocking version**

  \[
  \text{Wait}(S):\quad \begin{cases} 
  \text{if } S > 0 \text{ then} \\
  S := S - 1 \\
  \text{else} \\
  \text{block execution of calling process}
  \end{cases}
  \]

  \[
  \text{Signal}(S):\quad \begin{cases} 
  \text{if processes blocked on } S \text{ then} \\
  \text{awaken one of them} \\
  \text{else} \\
  S := S + 1
  \end{cases}
  \]

- Binary semaphores only allow the integer to hold the values 0 and 1
- Binary semaphores are easier to implement than general semaphores
Solution #1

var
  mutex=1: binary-semaphore;
  delay=0: binary-semaphore;
  C={initvalue}: integer;

Procedure Wait()
  begin
    wait(mutex);
    C:=C-1;
    if C < 0 then begin
      signal(mutex);
      wait(delay);
    end
    else
      signal(mutex);
  end

Procedure Signal()
  begin
    wait(mutex);
    C:=C+1;
    if C <= 0 then
      signal(delay)
      signal(mutex)
  end
Solution #2

```plaintext
var
    mutex=1: binary-semaphore;
    delay=0: binary-semaphore;
    C={initvalue}: integer;

Procedure Wait()
    begin
        wait(mutex);
        C:=C-1;
        if C < 0 then begin
            signal(mutex);
            wait(delay);
        end
        signal(mutex);
    end

Procedure Signal()
    begin
        wait(mutex);
        C:=C+1;
        if C <= 0 then
            signal(delay)
        else
            signal(mutex)
    end
```
Solution #3

```plaintext
var
mutex=1: binary-semaphore;
delay=0: binary-semaphore;
barrier=1: binary-semaphore;
C={initvalue}: integer;

Procedure Wait()
begin
    wait(barrier);
    wait(mutex);
    C:=C-1;
    if C < 0 then begin
        signal(mutex);
        wait(delay);
        end
    else
        signal(mutex);
    signal(barrier);
end

Procedure Signal()
begin
    wait(mutex);
    C:=C+1;
    if C = 1 then
        signal(delay)
    signal(mutex)
end
```
Solution #4

```plaintext
var
  mutex=1: binary-semaphore;
  delay={min(1,initvalue)}: binary-semaphore;
  C={initvalue}: integer;

Procedure Wait()
  begin
    wait(delay);
    wait(mutex);
    C:=C-1;
    if C > 0 then
      signal(delay);
      signal(mutex);
    end

Procedure Signal()
  begin
    wait(mutex);
    C:=C+1;
    if C = 1 then
      signal(delay)
      signal(mutex)
    end
```
Performance

- Semaphore Operations

<table>
<thead>
<tr>
<th></th>
<th>c&lt;=0</th>
<th>c=1</th>
<th>c&gt;1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Wait() Signal()</td>
<td>Wait() Signal()</td>
<td>Wait() Signal()</td>
</tr>
<tr>
<td>Semaphore</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Solution #1</td>
<td>Incorrect Incorrect Incorrect Incorrect Incorrect Incorrect Incorrect</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solution #2</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Solution #3</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Solution #4</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

- Restrictive/Unrestrictive
Conclusions

- The implementation of general semaphores using binary semaphores must be implemented carefully so no concurrency errors are introduced.

- Various solutions exist, when choosing a solution examine the performance characteristics of each that best suits your needs.

- Implementation of general semaphores using binary semaphores is not recommended when efficiency is a concern.


