Higher-Level Synchronization Primitives

Steve Goddard

goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/CSCE451

The problem with semaphores

- Too general — one primitive for both mutual exclusion and condition synchronization

Example: A P/C system with multiple producers & consumers

```
globals
fullBuffers : semaphore := 0
emptyBuffers : semaphore := n
mutex : binary_semaphore :=
```

Producer
```
begin loop
<produce a character “c”>
down(emptyBuffers)
down(b(mutex))
buf[nextIn] := c
nextIn := nextIn+1 mod n
up(b(mutex))
up(fullBuffers)
end loop end
```

Consumer
```
begin loop
<consume a character “c”>
down(fullBuffers)
down(b(mutex))
c := buf[nextOut]
nextOut := nextOut+1 mod n
up(b(mutex))
up(emptyBuffers)
end loop end
```
Higher-Level Synchronization
Hoare Monitors

◆ Collect related shared objects together into a module
◆ Define data operations
  » Calls to any monitor entry guaranteed to be mutually exclusive
  
  ```
  monitor : BoundedBuffer
  var buffer         : ...
  nextIn,nextOut : ...
  fullCount      : ...
  entry deposit(c : char)
  entry remove(var c : char)
  end BoundedBuffer
  ```

◆ Condition synchronization via condition variables
  » `wait(cv)` — blocks the caller on a condition-specific queue
  » `signal(cv)` — wakes up a waiter if one exists
  » `empty(cv)` — indicates if any process is currently waiting

Monitor Example
Producer/Consumer synchronization

```java
monitor : BoundedBuffer
var buffer         : ...
nextIn,nextOut : ...
fullCount      : ...
entry deposit(c : char)
entry remove(var c : char)
end BoundedBuffer

process Producer
begin
  loop
    <produce a character "c">
    BoundedBuffer.deposit(c)
  end loop
end Producer

process Consumer
begin
  loop
    <consume a character "c">
    BoundedBuffer.remove(c)
  end loop
end Consumer
```
Monitor Example
Bounded buffer implementation

```plaintext
monitor : BoundedBuffer
var buffer : array [0..n-1] of char
nextIn, nextOut : 0..n-1 := 0
fullCount : 0..n := 0
notEmpty, notFull : condition
entry deposit(c : char)
begin
if (fullCount = n) then wait(notFull)
buffer[nextIn] := c
nextIn := nextIn+1 mod n
fullCount := fullCount + 1
signal(notEmpty)
end deposit

end deposit

entry remove(var c : char)
begin
if (fullCount = 0) then wait(notEmpty)
c := buffer[nextOut]
nextOut := nextOut+1 mod n
fullCount := fullCount - 1
signal(notFull)
end remove

end remove

end BoundedBuffer
```

A Different Synchronization Example
Readers/Writers synchronization

◆ A generalization of producer consumer systems
  » Typically involves a set of processes (> 2)
  » Reading is non-destructive
  » Writing updates data (rather than creating it)
◆ Rules
  » Multiple readers may be reading simultaneously
  » Only one writer may be active at a time
  » Reading and writing cannot proceed simultaneously
◆ Issues
  » Makes sure readers don’t starve writers (& vice versa)
Readers/Writers Synchronization
A monitor-based solution — structure

process Reader:
begin
loop
Sync_RW.StartRead
<read the desired data>
Sync_RW.EndRead
end loop
end Reader;

process Writer:
begin
loop
Sync_RW.StartWrite
<write the desired data>
Sync_RW.EndWrite
end loop
end Writer;

monitor : Sync_RW
var numReaders : int := 0
writerBusy : boolean := FALSE
OKtoRead, OKtoWrite : condition
entry StartRead()
entry EndRead()
entry StartWrite()
entry EndWrite()
eend Sync_RW

Readers/Writers Synchronization
A monitor-based solution — details

monitor : Sync_RW
var numReaders : int := 0
writerBusy : boolean := FALSE
OKtoRead, OKtoWrite : condition
entry StartRead()
begin
if (writerBusy OR NOT empty(OKtoWrite)) then
wait(OKtoRead)
end if
numReaders += 1
signal(OKtoRead)
end StartRead

entry StartWrite()
begin
if (writerBusy OR numReaders > 0) then
wait(OKtoWrite)
end if
writerBusy := TRUE
signal(OKtoWrite)
end StartWrite
end Sync_RW

entry EndRead()
begin
numReaders := numReaders - 1
if (numReaders = 0) then
signal(OKtoWrite)
end if
end EndRead

entry EndWrite()
begin
writerBusy := FALSE
if (NOT empty(OKtoRead)) then
signal(OKtoRead)
else
signal(OKtoWrite)
end if
end EndWrite
end Sync_RW
Semantics of synchronization
A discipline of concurrent programming

◆ What is the strongest statement we can make about the state of a monitor after a waiter wakes up?

entry deposit(c : char)
begin
  if (fullCount = n) then
    wait(notFull)
  end if
  buffer[nextIn] := c
  nextIn := (nextIn + 1) mod n
  fullCount := fullCount + 1
  signal(notEmpty)
end deposit

entry remove(var c : char)
begin
  if (fullCount = 0) then
    wait(notEmpty)
  end if
  c := buffer[nextOut]
  nextOut := (nextOut + 1) mod n
  fullCount := fullCount - 1
  signal(notFull)
end remove

Realizing the Semantics
Implementing Monitors

monitor BoundedBuffer
entry deposit(c : char)
begin
  if (fullCount = n) then
    wait(notFull)
  end if
  buffer[nextIn] := c
  nextIn := (nextIn + 1) mod n
  fullCount := fullCount + 1
  signal(notEmpty)
end deposit

entry remove(var c : char)
begin
  if (fullCount = 0) then
    wait(notEmpty)
  end if
  c := buffer[nextOut]
  nextOut := (nextOut + 1) mod n
  fullCount := fullCount - 1
  signal(notFull)
end remove
end BoundedBuffer

condition variable waiting queues
Producer Process
Consumer Process

Semantics of synchronization II

"Mesa" semantics

◆ Synchronization in the Mesa language from Xerox PARC

  » a signal (called notify()) is a "hint"

monitor BoundedBuffer
var ...
entry deposit(c : char) begin
  while (fullCount = n) do
    wait(notFull) and while
  buffer[nextIn] := c
  nextIn := nextIn + 1 mod n
  fullCount := fullCount + 1
  notify(notEmpty)
end deposit
end BoundedBuffer

entry remove(var c : char) begin
  while (fullCount = 0) do
    wait(notEmpty) and while
  c := buffer[nextOut]
  nextOut := nextOut + 1 mod n
  fullCount := fullCount - 1
  notify(notFull)
end remove