COS 318: Operating Systems

Semaphores, Monitors and Condition Variables

Kai Li and Andy Bavier
Computer Science Department
Princeton University

(http://www.cs.princeton.edu/courses/cos318/)
Today’s Topics

- Consumer-producer problem
- Semaphores
- Monitors
- Mesa-style monitor and its idioms
- Barriers
Revisit Mutex

- Mutex can solve the critical section problem
  ```
  Acquire( lock );
  
  Critical section
  
  Release( lock );
  ```

- Always use Mutex primitives when you access shared data structures
  E.g. shared “count” variable
  ```
  Acquire( lock );
  count++;
  Release( lock );
  ```

- Are mutex primitives adequate to solve all problems?
Producer-Consumer (Bounded Buffer) Problem

Producer:
while (1) {
    produce an item
    Insert item in buffer
    count++;
}

Consumer:
while (1) {
    remove an item from buffer
    count--;
    consume an item
}

count = 4

N = 12

Can we solve this problem with Mutex primitives?
Use Mutex, Block and Unblock

Producer:
while (1) {
    produce an item
    if (count == N)
        Block();
    Insert item in buffer
    Acquire(lock);
    count++;
    Release(lock);
    if (count == 1)
        Unblock(Consumer);
}

Consumer:
while (1) {
    if (!count)
        Block();
    remove an item from buffer
    Acquire(lock);
    count--;
    Release(lock);
    if (count == N-1)
        Unblock(Producer);
    consume an item
}
Use Mutex, Block and Unblock

- Race condition!
- Any way to make this work?
- These primitives are not enough

```c
Producer:
  while (1) {
    produce an item
    if (count == N)
    Block();
    Insert item in buffer
    Acquire(lock);
    count++;
    Release(lock);
    if (count == 1)
    Unblock(Consumer);
  }

N = 12

Consumer:
  while (1) {
    if (!count)
    {context switch}
    Block();
    remove an item from buffer
    Acquire(lock);
    count--; Release(lock);
    if (count == N-1)
    Unblock(Producer);
    consume an item
  }

count = 12
```
Semaphores (Dijkstra, 1965)

- Initialization
  - Initialize a semaphore \( s \)

- \( P \) (or Down or Wait or “Probeer”) definition
  - Atomic operation
  - Wait for semaphore to become positive and then decrement

\[
P(s)\
\begin{align*}
  &\text{while } (s <= 0) \\
  &\quad; \\
  &\quad s--; \\
\end{align*}
\]

- \( V \) (or Up or Signal or “Verhoog”) definition
  - Atomic operation
  - Increment semaphore by 1

\[
V(s)\
\begin{align*}
  &s++; \\
\end{align*}
\]
Bounded Buffer with Semaphores

- Initialization: emptyCount = N; fullCount = 0
- Are $P(\text{mutex})$ and $V(\text{mutex})$ necessary?
A device thread works with an interrupt handler

What to do with shared data?

What if “m” is held by another thread or by itself?
Use Semaphore

Init(s, 0);

Device thread
while (1) {
P(s);
Acquire(m);
... deal with interrupt
... V(s);
Release(m);
}

Interrupted Thread

Interrupt handler

Interrupt
Semaphores Are Not Always Convenient

- A shared queue with Enqueue and Dequeue:

  ```
  Enqueue(q, item)
  {
      Acquire(mutex);
      put item into q;
      Release(mutex);
  }
  
  Dequeue(q)
  {
      Acquire(mutex);
      take an item from q;
      Release(mutex);
      return item;
  }
  ```

- What if we want `Dequeue(q)` to block until `q` is not empty?
  - It is a consumer and producer problem

- Semaphores are difficult to use
  - Orders are important
Monitor: Hide Mutual Exclusion

- Brinch-Hansen (73), Hoare (74)
- Procedures are mutual exclusive

Queue of waiting processes trying to enter the monitor

Shared data

procedures
Condition Variables in A Monitor

- **Wait( condition )**
  - Block on “condition”

- **Signal( condition )**
  - Wake up a blocked process on “condition”
Producer-Consumer with Monitors

```plaintext
procedure Producer
begin
  while true do
  begin
    produce an item
    ProdCons.Enter();
  end;
end;

procedure Consumer
begin
  while true do
  begin
    ProdCons.Remove();
    consume an item;
  end;
end;

monitor ProdCons
condition full, empty;

procedure Enter;
begin
  if (buffer is full)
    wait(full);
  put item into buffer;
  if (only one item)
    signal(empty);
end;

procedure Remove;
begin
  if (buffer is empty)
    wait(empty);
  remove an item;
  if (buffer was full)
    signal(full);
end;
```
Hoare’s Signal Implementation (MOS p137)

- Run the signaled thread immediately and suspend the current one (Hoare)

- What if the current thread has more things to do?

```pascal
monitor ProdCons
condition full, empty;

procedure Enter;
begn
  if (buffer is full)
    wait(full);
  put item into buffer;
  if (only one item)
    signal(empty);
end;

procedure Remove;
begn
  if (buffer is empty)
    wait(empty);
  remove an item;
  if (buffer was full)
    signal(full);
end;
```
Hansen’s Signal Implementation (MOS p 137)

- Signal must be the last statement of a monitor procedure
- Exit the monitor

- Any issue with this approach?

```pascal
monitor ProdCons
condition full, empty;

procedure Enter;
begin
  if (buffer is full) then
    wait(full);
    put item into buffer;
    if (only one item) then
      signal(empty);
  end;
end;

procedure Remove;
begin
  if (buffer is empty) then
    wait(empty);
    remove an item;
    if (buffer was full) then
      signal(full);
  end;
end;
```
Mesa Signal Implementation

- Continues its execution
  
  ```
  if (only one item)
      signal(empty);
    something else
  end;
  ```


- This is easy to implement!

- Issues?
Mesa Style “Monitor” (Birrell’s Paper)

- Associate a condition variable with a mutex

- Wait( mutex, condition )
  - Atomically unlock the mutex and enqueued on the condition variable (block the thread)
  - Re-lock the lock when it is awaken

- Signal( condition )
  - No-op if there is no thread blocked on the condition variable
  - Wake up at least one if there are threads blocked

- Broadcast( condition )
  - Wake up all waiting threads
Consumer-Producer with Mesa-Style Monitor

static count = 0;
static Cond full, empty;
static Mutex lock;

Enter(Item item) {
    Acquire(lock);
    if (count==N)
        Wait(lock, full);
    insert item into buffer
    count++;
    if (count==1)
        Signal(empty);
    Release(lock);
}

Remove(Item item) {
    Acquire(lock);
    if (!count)
        Wait(lock, empty);
    remove item from buffer
    count--;  
    if (count==N-1)
        Signal(full);
    Release(lock);
}

Any issues with this?
Consumer-Producer with Mesa-Style Monitor

static count = 0;
static Cond full, empty;
static Mutex lock;

Enter(Item item) {
    Acquire(lock);
    while (count==N)
        Wait(lock, full);
    insert item into buffer
    count++;
    if (count==1)
        Signal(empty);
    Release(lock);
}

Remove(Item item) {
    Acquire(lock);
    while (!count)
        Wait(lock, empty);
    remove item from buffer
    count--;
    if (count==N-1)
        Signal(full);
    Release(lock);
}
The Programming Idiom

- Waiting for a resource

  ```
  Acquire( mutex );
  while ( no resource )
    wait( mutex, cond );
  ...
  (use the resource)
  ...
  Release( mutex );
  ```

- Make a resource available

  ```
  Acquire( mutex );
  ...
  (make resource available)
  ...
  Signal( cond );
  /* or Broadcast( cond );
  Release( mutex );
  ```
Revisit the Motivation Example

Does this work?

Enqueue(Queue q, Item item) {
    Acquire(lock);
    insert an item to q;
    Signal(Empty);
    Release(lock);
}

Item GetItem(Queue q) {
    Item item;
    Acquire(lock);
    while (q is empty)
        Wait(lock, Empty);
    remove an item;
    Release(lock);
    return item;
}
Condition Variables Primitives

- **Wait( mutex, cond )**
  - Enter the critical section (min busy wait)
  - Release mutex
  - Put my TCB to cond’s queue
  - Call scheduler
  - Exit the critical section . . . (blocked)

- **Waking up:**
  - Acquire mutex
  - Resume

- **Signal( cond )**
  - Enter the critical section (min busy wait)
  - Wake up a TCB in cond’s queue
  - Exit the critical section
More on Mesa-Style Monitor

- Signaler continues execution
- Waiters simply put on ready queue
  - Must reevaluate the condition
- No constraints on when the waiting thread must run
- No constrains on signaler
  - Can execute after signal call (Hansen’s cannot)
  - Do not need to relinquish control to awaken thread/process
Evolution of Monitors

- Brinch-Hansen (73) and Hoare Monitor (74)
  - Concept, but no implementation
  - Requires Signal to be the last statement (Hansen)
  - Requires relinquishing CPU to signaler (Hoare)

- Mesa Language (77)
  - Monitor in language, but signaler keeps mutex and CPU
  - Waiter simply put on ready queue, with no special priority

- Modula-2+ (84) and Modula-3 (88)
  - Explicit LOCK primitive
  - Mesa-style monitor

- Pthreads (95)
  - Started standard effort around 1989
  - Defined by ANSI/IEEE POSIX 1003.1 Runtime library

- Java threads
  - James Gosling in early 1990s without threads
  - Use most of the Pthreads primitives
Example: A Simple Barrier

- Thread A and Thread B want to meet at a particular point
- Then both go forward

- How would you program this with a monitor?
Using Semaphores as A Barrier

- Use two semaphore?
  
  ```
  init(s1, 0);
  init(s2, 0);
  ```

- What about more than two threads?
  
  Thread A
  
  ```
  ... 
  V(s1);
  P(s2);
  ... 
  ```

  Thread B
  
  ```
  ... 
  V(s2);
  P(s1);
  ... 
  ```
Barrier Primitive

- **Functions**
  - Take a barrier variable
  - Broadcast to n-1 threads
  - When barrier variable has reached n, go forward

- **Hardware support on some parallel machines**
  - Multicast network
  - Counting logic
  - User-level barrier variables
Equivalence

◆ Semaphores
  ● Good for signaling
  ● Not good for mutex because it is easy to introduce a bug

◆ Monitors
  ● Good for scheduling and mutex
  ● Maybe costly for a simple signaling
The Big Picture

<table>
<thead>
<tr>
<th>High-Level Atomic API</th>
<th>Mutex</th>
<th>Semaphores</th>
<th>Monitors</th>
<th>Barriers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low-Level Atomic Ops</td>
<td>Load/store</td>
<td>Interrupt disable/enable</td>
<td>Test&amp;Set</td>
<td>Other atomic instructions</td>
</tr>
<tr>
<td></td>
<td>Interrupts (I/O, timer)</td>
<td>Multiprocessors</td>
<td>CPU scheduling</td>
<td></td>
</tr>
</tbody>
</table>
Summary

- Mutex alone are not enough
- Semaphores
- Monitors
- Mesa-style monitor and its idiom
- Barriers