

COS 318: Operating Systems

Semaphores, Monitors and Condition Variables

Kai Li
Computer Science Department
Princeton University

(<http://www.cs.princeton.edu/courses/cos318/>)

Today's Topics

- ◆ Semaphores
- ◆ Monitors
- ◆ Mesa-style monitors
- ◆ Programming idiom

Semaphores (Dijkstra, 1965)

◆ Initialization

- Initialize a value atomically

◆ P (or Down or Wait) definition

- Atomic operation
- Wait for semaphore to become positive and then decrement

```
P(s) {  
    while (s <= 0)  
        ;  
    s--;  
}
```

◆ V (or Up or Signal) definition

- Atomic operation
- Increment semaphore by 1

```
V(s) {  
    s++;  
}
```

Bounded Buffer with Semaphores

```
producer() {  
    while (1) {  
        produce an item  
        P(emptyCount);  
  
        P(mutex);  
        put the item in buffer  
        V(mutex);  
  
        V(fullCount);  
    }  
}  
  
consumer() {  
    while (1) {  
        P(fullCount);  
  
        P(mutex);  
        take an item from buffer  
        V(mutex);  
  
        V(emptyCount);  
        consume the item  
    }  
}
```

- ◆ Initialization: emptyCount = N; fullCount = 0
- ◆ Are P(mutex) and V(mutex) necessary?

Implementation

```

P(sem) {
  while (!TAS(sem.guard))
    ;
  sem.value--;
  if (sem.value < 0) {
    enqueue self to sem.q;
    schedule & sem.guard = 0;
  };
  sem.guard = 0;
}

V(sem) {
  while (!TAS(sem.guard))
    ;
  sem.value++;
  if (sem.value <= 0)
    dequeue sem.q;
    make thread ready;
};
sem.guard = 0;
}
    
```

- ◆ Would this work?



Motivation

- ◆ A shared queue has Enqueue and Dequeue:

```

Enqueue(q, item)
{
  Acquire(mutex);
  put item into q;
  Release(mutex);
}

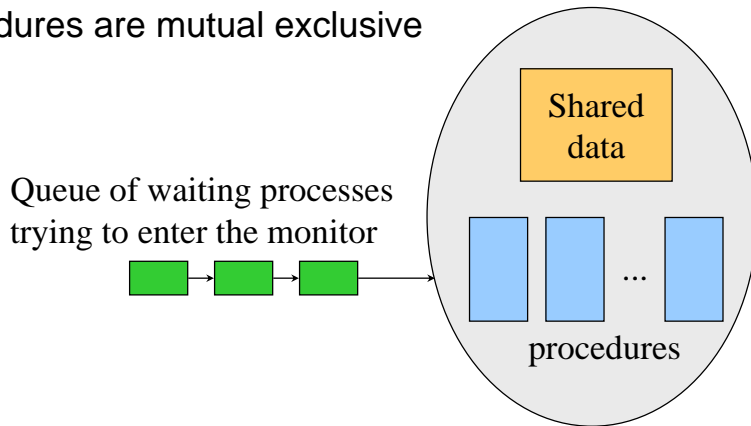
Dequeue(q)
{
  Acquire(mutex);
  take an item from q;
  Release(mutex);
  return item;
}
    
```

- ◆ It is a consumer and producer problem
 - Dequeue(q) blocks until q is not empty
- ◆ Semaphores are difficult to use: orders are important



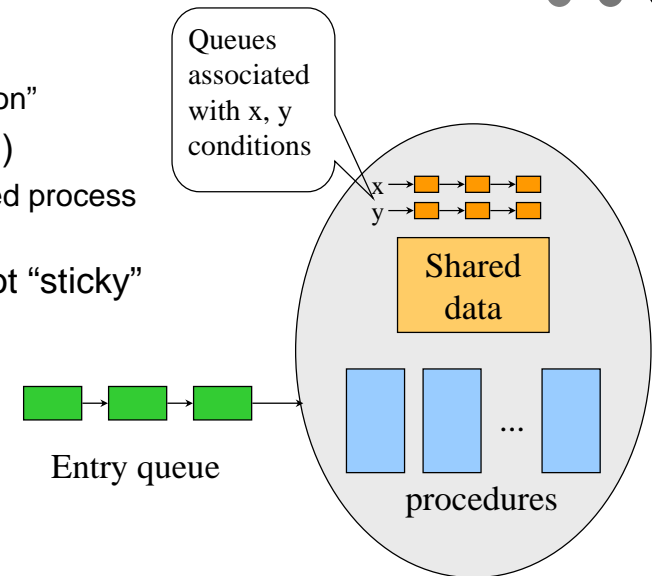
Monitor: Hide Mutual Exclusion

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



Condition Variables in A Monitor

- ◆ Wait(condition)
 - Block on "condition"
- ◆ Signal(condition)
 - Wakeup a blocked process on "condition"
- ◆ Conditions are not "sticky"



Producer-Consumer with Monitors

```
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;
```

```
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;
```

Options of the Signaler

- ◆ Run the signaled thread immediately and suspend the current one (Hoare)
 - If the signaler has other work to do, life is complex
 - It is difficult to make sure there is nothing to do, because the signal implementation is not aware of how it is used
 - It is easy to prove things
- ◆ Exit the monitor (Hansen)
 - Signal must be the last statement of a monitor procedure
- ◆ Continues its execution (Mesa)
 - Easy to implement
 - But, the condition may not be true when the awoken process actually gets a chance to run

10

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 awoken
- ◆ 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
- ◆ Original Mesa paper
 - B. Lampson and D. Redell. Experience with processes and monitors in Mesa. *Comm. ACM* 23, 2 (feb 1980), pp 106-117.

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?

12

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);
}
```



13

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);
```



14

Revisit the Motivation Example

```
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;
}
```

◆ Can this be further improved?



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



16

More on Mesa-Style Monitor

- ◆ Signaler continues execution
- ◆ Waiter simply put on ready queue, with no special priority
 - Must then spin and reevaluate the condition
- ◆ No constraints on when the waiting thread/process must run after a “signal”
- ◆ Simple to introduce a broadcast: wake up all
- ◆ No need to make signal sticky
- ◆ No constraints on signaler
 - Can execute after signal call (Hansen’s cannot)
 - Do not need to relinquish control to awaken thread/process



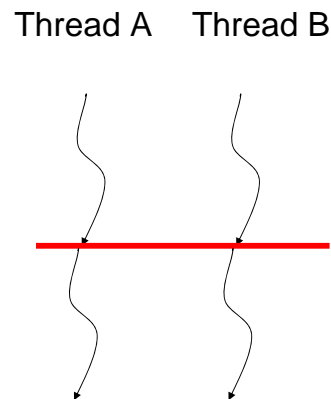
Evolution of Monitors

- ◆ 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



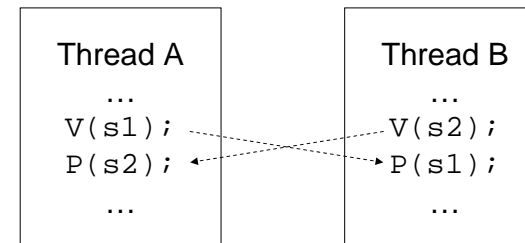
Example: A Simple Barrier

- ◆ Thread A and Thread B want to meet at a particular point and then go on
- ◆ How would you program this with monitor?



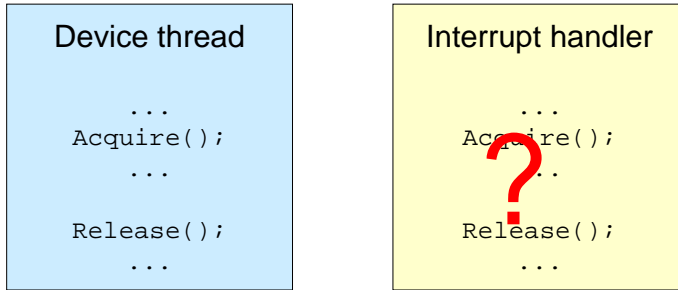
Using Semaphores as A Barrier

- ◆ Use two semaphores
- ```
init(s1, 0);
init(s2, 0);
```



## Example: Interrupt Handler

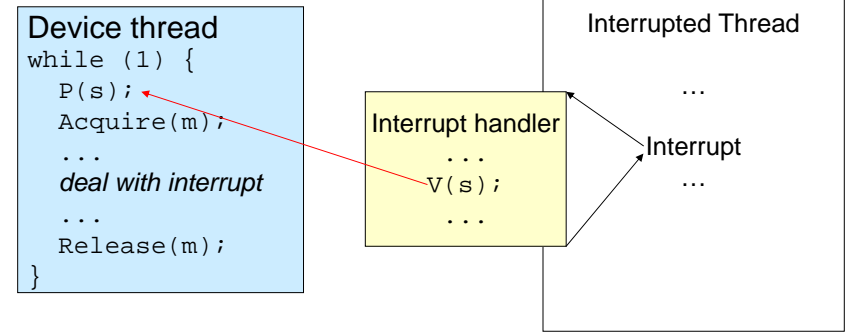
- ◆ A device thread works with an interrupt handler
- ◆ What to do with shared data?
- ◆ What if “m” is held by another thread or by itself?



21

## Use Semaphore to Signal

```
Init(s, 0);
```



22

## 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

23

## Summary

- ◆ Semaphores
- ◆ Monitors
  - Hoare's
  - Mesa-style and its idiom
- ◆ Examples
  - Use cases for semaphores

24