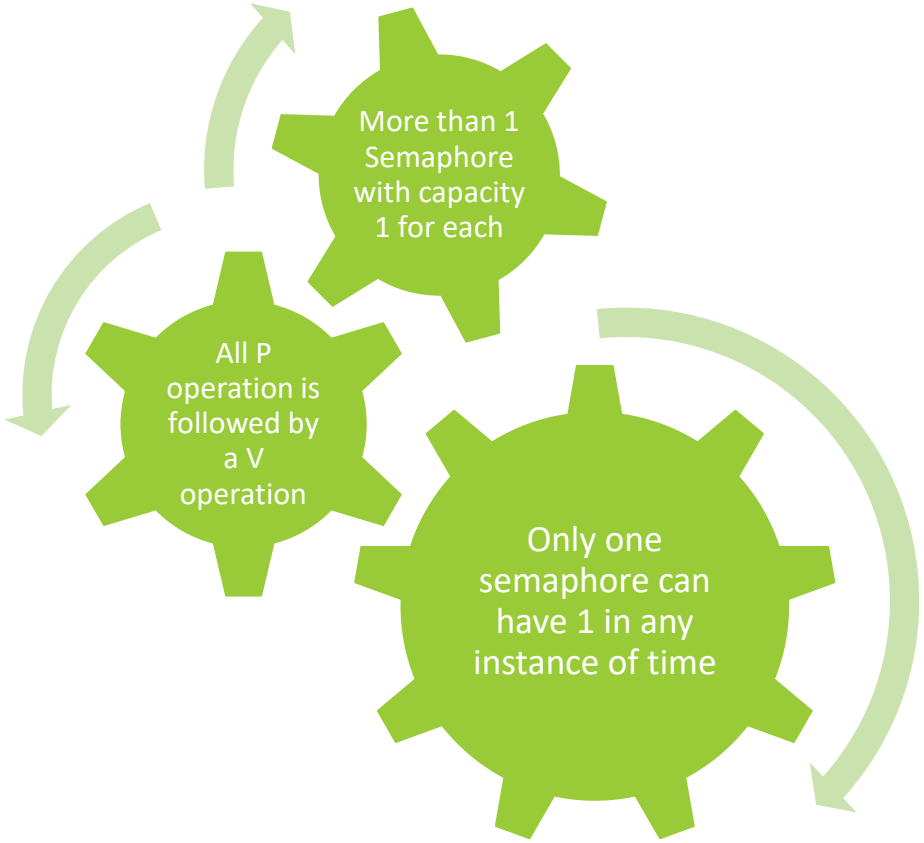
ENGI 7894/9869

Split Binary Semaphores

Presented by
Md Monjur Ul Hasan

Question or Comments From Previous Class?

Split Binary Semaphores



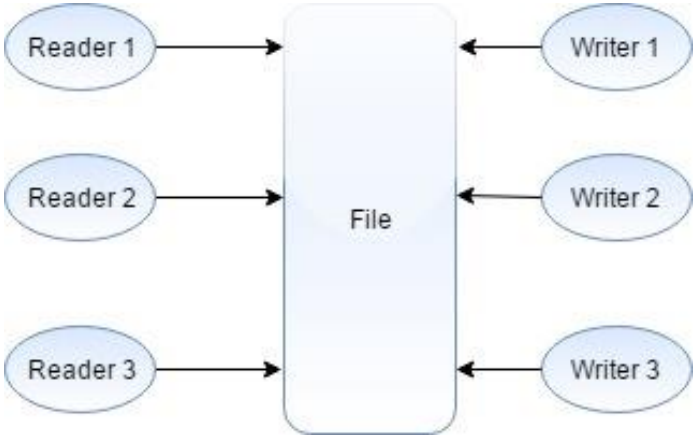
$$0 \leq s_0 + s_1 + \ldots + s_n \leq 1$$

Reader Writer Problem

Multiple Reader

One Writer

Maximize Concurrency



Reader Writer Problem: Coarse-grain Solution

```
rw = Semaphore(1);
```

Reader

```
while (true)
{
        rw.P();
        read file
        rw.V();
}
```

Writer

```
while (true)
{
        rw.P();
        write file
        rw.V();
}
```

Sequential access → Only one reader can read → Only one writer can write

Reader Writer Problem: Fine-grain Solution

```
Semaphore resource  = new Semaphore(1);
Semaphore mutex     = new Semaphore(1);
int readcount       = 0;
```

Reader

```
mutex.P(); // entry protocol
    readcount++;
    if (readcount == 1)
        resource.P();
mutex.V();

        READ OPERATION...

mutex.P();
    readcount--;
    if (readcount == 0)
        resource.V();
mutex.V();
```

Writer

```
resource.P();

    WRITE OPERATIONS…

resource.V();
```

Problem!!

Starvation

Concurrent Readers

Reader Writer Problem: Fair Solution

```
int readcount = 0;
int writecount = 0;
Semaphore rmutex    = new Semaphore(1);
Semaphore wmutex    = new Semaphore(1);
Semaphore readTry   = new Semaphore(1);
Semaphore resource  = new Semaphore(1);
```

Reader

```
readTry.P();
        rmutex.P();
        readcount++;
        if (readcount == 1)
                resource.P();
        rmutex.V();
readTry.V();

READ OPERATION...

rmutex.P();
readcount--;
if (readcount == 0)
        resource.V();
rmutex.V();
```

Writer

```
wmutex.P();
        writecount++;
        if (writecount == 1)
                readTry.P();
wmutex.V();

resource.P();
        WRITE OPERATION...
resource.V();

wmutex.P();
        writecount--;
        if (writecount == 0)
                readTry.V();
wmutex.V();
```

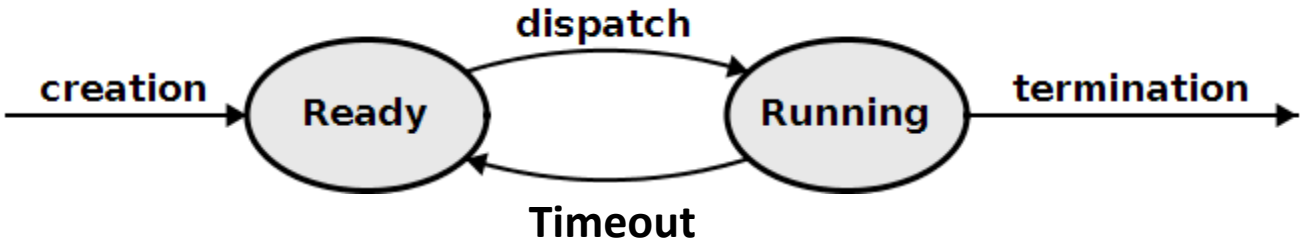
Question or Comments?

Semaphores: Implementation Example

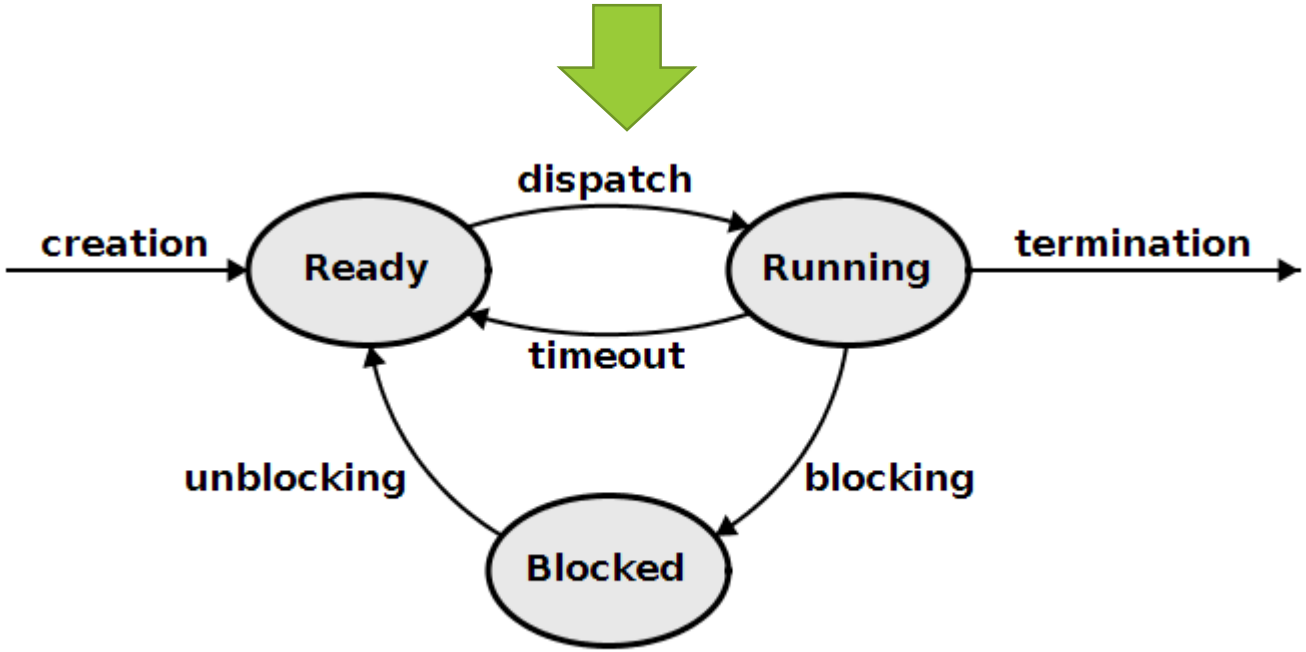
```
public class Semaphore {
        int state;
        iLock lock;
        Condition condition;
        public Semaphore(int c) {
                state = c;
                lock = new Lock();
        }
        public void P() {
                lock.lock();
                try {
                        while (state == 0) {} ;
                        state--;
                }
                finally {
                       lock.unlock();
                }
        }
       public void V() {
                lock.lock();
                try {
                        state++;
                }
                finally {
                          lock.unlock();
                }
        }
}
```

Efficient Locks

Thread Lifecycles



Traditional Thread Lifecycle



Modern (Mutable) Thread Lifecycle

ReentrantLock

- A ReentrantLock is owned by the thread last successfully locking, but not yet unlocking it. A thread invoking lock will return, successfully acquiring the lock, when the lock is not owned by another thread. The method will return immediately if the current thread already owns the lock
- ReentrantLock allow threads to enter into lock on a resource more than once. When the thread first enters into lock, a hold count is set to one. Before unlocking the thread can re-enter into lock again and every time hold count is incremented by one.
- Reentrant Locks also offer fairness.

ReentrantLock Methods

- lock()
- unlock()
- tryLock()
- lockInterruptibly()
- isHeldByCurrentThread()
- getHoldCount()

Condition Variable

Conditional variable creates on ReentrantLock.

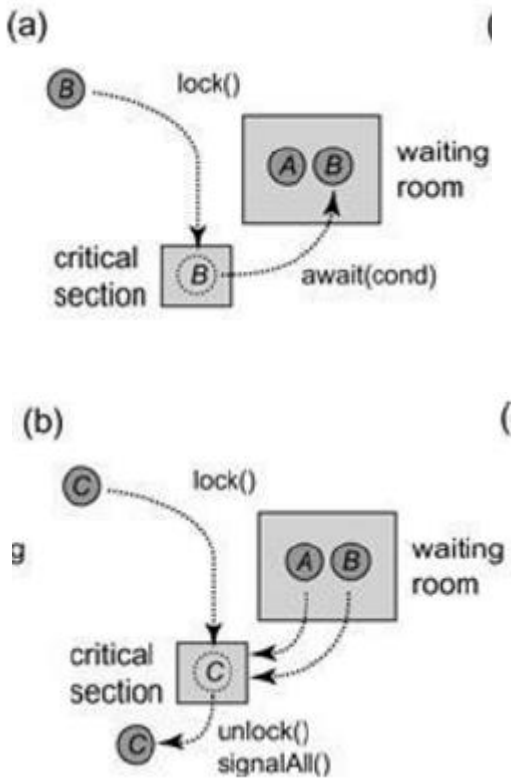
- newCondition() is used on existing lock to create condition variable

Wait on a lock use condition variable.

- Waiting threads are not scheduled to execute
- Use await()
- On awaite() lock is also released

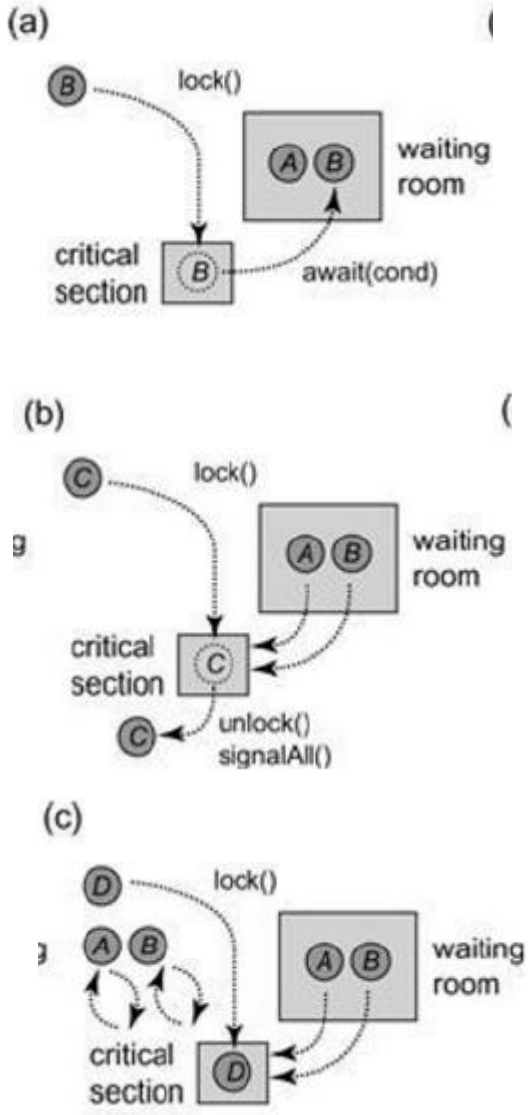
Signal() and signalAll() wake up waiting methods

- signal() will only unblock one waiting thread
- signalAll() unblocks all of them



Semaphore with ReentranLock

```
public class Semaphore {
    final int capacity;
    int state;
    Lock lock;
    Condition condition;
    public Semaphore(int c) {
                capacity = c;
                state = c;
                lock = new ReentrantLock();
                condition = lock.newCondition();
    }
    public void acquire() { //p()
                lock.lock();
                try {
                        while (state == 0) {
                                condition.await();
                        }
                        state--;
                }
                finally {
                        lock.unlock();
                }
    }
    public void release() {  //v()
                lock.lock();
                try {
                      state++;
                      condition.signalAll();
                }
                finally {
                      lock.unlock();
                }
    }
}
```

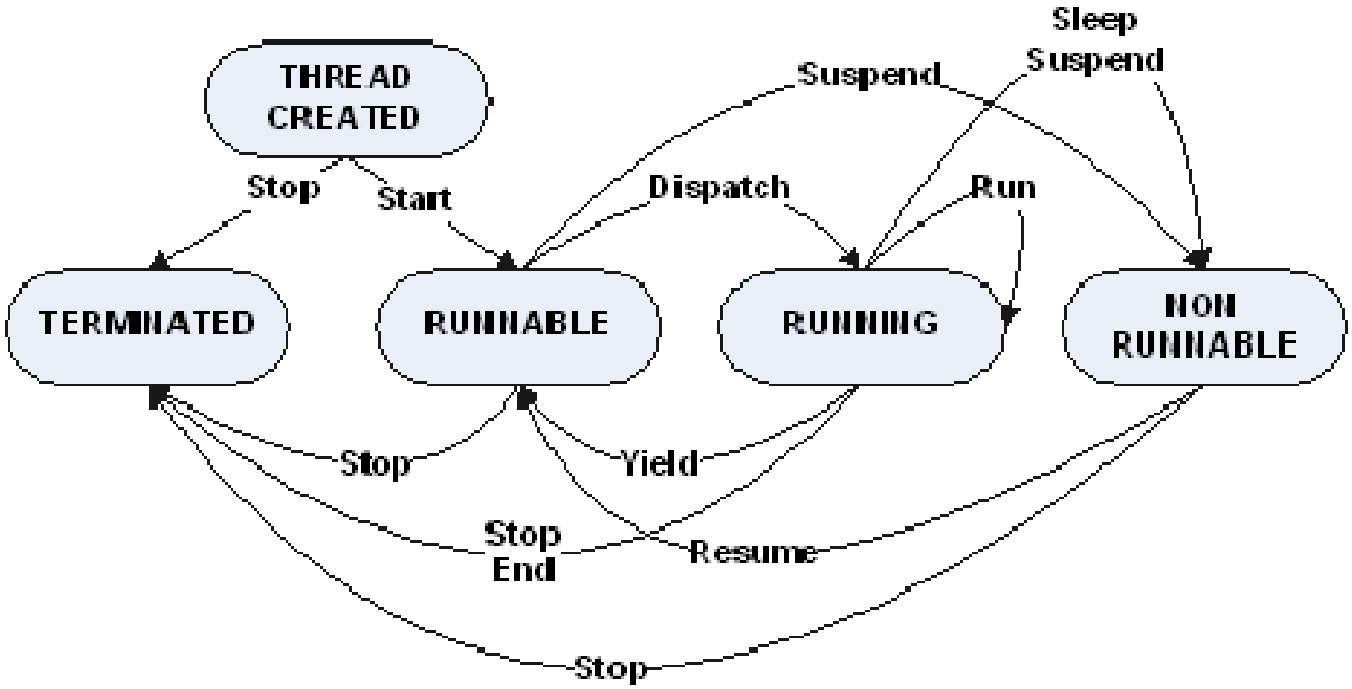


Question or Comments?

Producer – Consumer Solution: ReentrantLock()

```
class BoundedBuffer {
        final Lock lock = new ReentrantLock();
        final Condition notFull = lock.newCondition();
        final Condition notEmpty = lock.newCondition();
        final Object[] items = new Object[100];
        int putptr, takeptr, count;
        public void put(Object x) throws InterruptedException {
                lock.lock();
                try {
                        while (count == items.length)
                                notFull.await();
                        items[putptr] = x;
                        if (++putptr == items.length)   putptr = 0;
                                ++count;
                        notEmpty.signal();
                } finally {
                        lock.unlock();
                }
        }
        public Object take() throws InterruptedException {
                lock.lock();
                try {
                        while (count == 0)
                                notEmpty.await();
                        Object x = items[takeptr];
                        if (++takeptr == items.length) takeptr = 0;
                                --count;
                        notFull.signal();
                        return x;
                } finally {
                        lock.unlock();
                }
         }
}
```

Lifecycle of Threads



ReadWriteLock

A ReadWriteLock maintains a pair of associated locks, one for read-only operations and one for writing. The read lock may be held simultaneously by multiple reader threads, so long as there are no writers

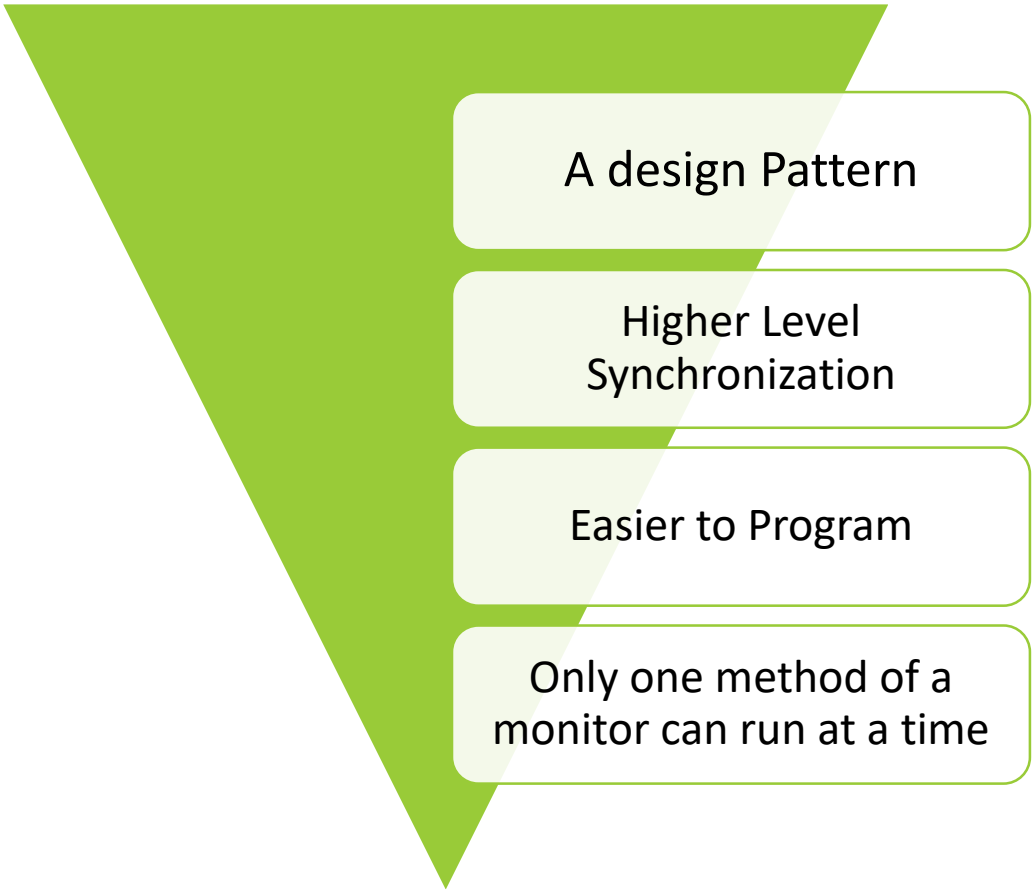
Multiple read Locks, but only a single Thread can acquire mutually-exclusive write Lock

- Lock readLock() - returns the lock used for reading.
- Lock writeLock() - returns the lock used for writing.

```
ReadWriteLock readWriteLock = new ReentrantReadWriteLock();
readWriteLock.readLock().lock();
// multiple readers can enter this section
// if not locked for writing, and not writers waiting
// to lock for writing.
readWriteLock.readLock().unlock();
readWriteLock.writeLock().lock();
// only one writer can enter this section,
// and only if no threads are currently reading.
readWriteLock.writeLock().unlock();
```

Question or Comments?

Monitor


A design Pattern
Higher Level Synchronization
Easier to Program
Only one method of a monitor can run at a time

Semaphores

```
public class Semaphore {
    final int capacity;
    int state;
    Lock lock;
    Condition condition;
    public Semaphore(int c) {
            capacity = c;
            state = c;
            lock = new ReentrantLock();
            condition = lock.newCondition();
    }
    public void acquire() {
            lock.lock();
            try {
                while (state == 0) {
                        condition.await();
                 }
                 state--;
            }
            finally {
                 lock.unlock();
            }
    }
    public void release() {
            lock.lock();
            try {
                state++;
                condition.signalAll();
            }
            finally {
                lock.unlock();
            }
    }
}
```

Topics

- Split Binary Semaphore
- Reader Writer Problem
- Practical Lock
  - ReentrantLock
  - Condition Variable
  - Semaphore (Reimplemented)
  - Producer Consumer Problem (Reimplemented)
  - ReadWriteLock
- Monitor

Thank you for your attention

Any Questions?