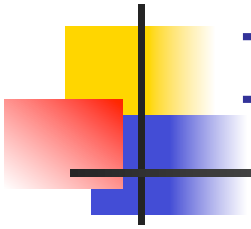
Operating Systems Design (CS 423)

Elsa L Gunter

2112 SC, UIUC

http://www.cs.illinois.edu/class/cs423/

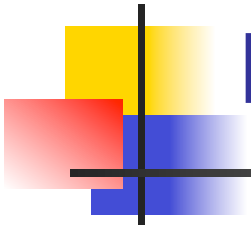
Based on slides by Roy Campbell, Sam King, and Andrew S Tanenbaum



Interface

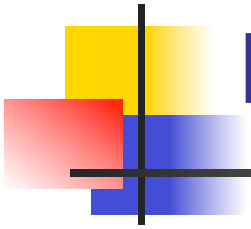
- readerStart()
- readerFinish()
- writerStart()
- writerFinish()

- Many threads can be in between a readStart and readerFinish
- Only one thread can be between writerStart and writitierFinish



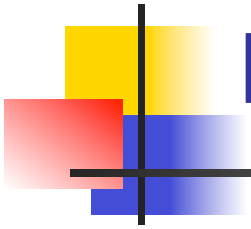
Reader – Writer Locks using Monitors

- Note: Implement Reader/Writer Locks as an abstractions, *not* as an integrated part of code
- Central Questions:
  - Shared Data?
  - Ordering Constraints?
  - How many Condition Variables?



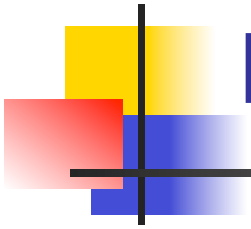
Reader – Writer Locks using Monitors

- Note: Implement Reader/Writer Locks as an abstractions, *not* as an integrated part of code
- Central Questions:
  - Shared Data? NumReaders NumWriters
  - Ordering Constraints?
  - How many Condition Variables?



Reader – Writer Locks using Monitors

- Note: Implement Reader/Writer Locks as an abstractions, *not* as an integrated part of code
- Central Questions:
  - Shared Data? NumReaders NumWriters
  - Ordering Constraints?
    - readerStart must wait if there are writers
    - writerStart must wait if there are readers or writes
  - How many Condition Variables?



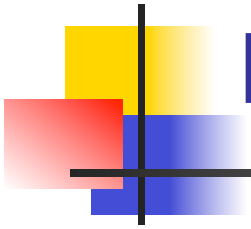
Reader – Writer Locks using Monitors

- Note: Implement Reader/Writer Locks as an abstractions, *not* as an integrated part of code
- Central Questions:
  - Shared Data? NumReaders NumWriters
  - Ordering Constraints?
    - readerStart must wait if there are writers
    - writerStart must wait if there are readers or writes
  - How many Condition Variables?
    - One: condRW (no readers or writers)

Basic Implementation

```
readerStart() {

}
```

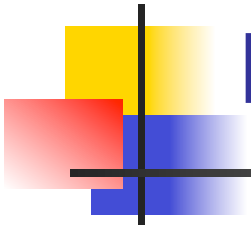
```
readerFinish() {

}
```



Basic Implementation

```
readerStart() {
  lock(lockRW);

  while(numWriters > 0){
    wait(lockRW,condRW);
  };

  numReaders++;

  unlock(lockRW);
}
```

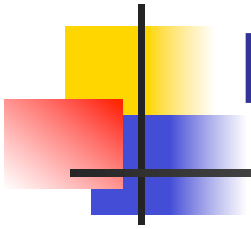
```
readerFinish() {
  lock(lockRW);

  numReaders--;

  broadcast(lockRW,condWR);

  unlock(lockRW);
}
```



Basic Implementation

```
writerStart() {
  lock(lockRW);

  while(numReaders > 0||
          numWriters >0){
    wait(lockRW,condRW);
  };

  numWriters++;

  unlock(lockRW);
}
```

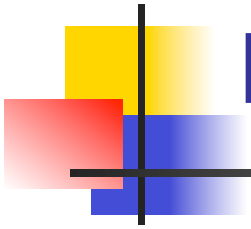
```
writerFinish() {
  lock(lockRW);

  numWriters--;


  broadcast(lockRW,condWR);

  unlock(lockRW);
}
```



Better Implementation

```
readerStart() {
  lock(lockRW);

  while(numWriters > 0){
    wait(lockRW,condRW);
  };

  numReaders++;

  unlock(lockRW);
}
```

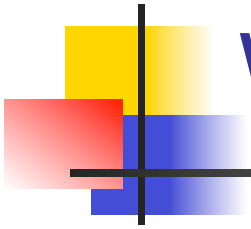
```
readerFinish() {
  lock(lockRW);

  numReaders--.

  if(numReaders == 0){
    signal(lockRW,condWR);
  };
  unlock(lockRW);
}
```



Better Implementation

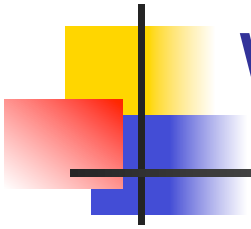
- Can we change broadcast to signal in writerFinish() in a similar way?

- Many Readers at a time, but only one Writer
- How long will one writer wait?
  - Starvation – process never gets a turn
- How to give priority to writer?



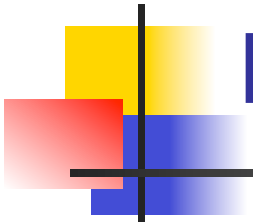
Write Priority

```
readerStart() {
  lock(lockRW);

  while(activeWriters + waitingWriters > 0){
    wait(lockRW,condRW);
  };

  numReaders++;

  unlock(lockRW);
}
```



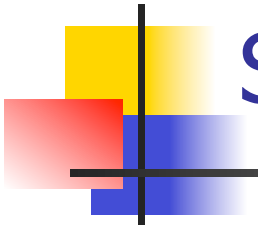
Write Priority

```
writerStart() {
  lock(lockRW);
  waitingWriters ++;
  while(numReaders > 0||
          numWriters >0){
    wait(lockRW,condRW);
  };
  waitingWriters--;
  numWriters++;

  unlock(lockRW);
}
```



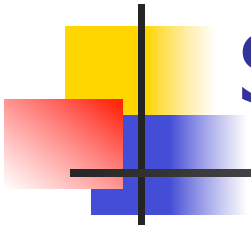
Lock and Reader / Writer Locks

- Reader-writer functions are similar to standard locks
  - Call readerStart before read shared data
  - Call readerFinish after done reading data
  - Call writerStart before writing shared data
  - Call writerFinish after done writing data
- These are known as "reader-writer locks"
  - Thread in between readerStart and readerFinish "holds a read lock"
  - Thread in between writerStart and writerFinish "holds a write lock"
- Compare reader-writer locks with standard locks



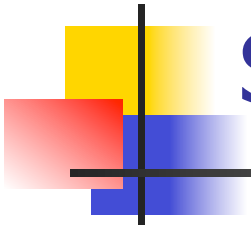
Semaphores (not used in this class)

- Like a generalized lock
- Semaphore has a non-negative integer value (>= 0) and supports
  - Down(): wait for semaphore to become positive, decrement semaphore by 1 (originally called "P" for Dutch "proberen")
  - Up(): increment semaphore by 1 (originally called "V" for Dutch "verhogen"). This wakes up a thread waiting in down(), if there are any.
  - Can also set the initial value for the semaphore



Semaphores – Quick Review

- The key parts in down() and up() are atomic
  - Two down calls at the same time cannot decrement the value below 0
- Binary semaphore
  - Value is either 0 or 1
  - Down() waits for value to become 1, then sets to 0
  - Up() sets value to 1, waking up waiting down

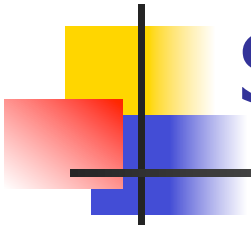


Semaphores

- Can be used for both types of sync
  - Mutual exclusion
  - Initial value of semaphore is 1

```
Down()
<critical section>
Up()
```

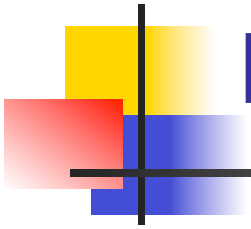
  - Like lock/unlock, but more general
  - Implement lock as a binary semaphore, initialized to 1



Semaphores

- Ordering constraints
  - Usually (not always) initial value is 0
  - Thread A wants to wait for thread B to finish before continuing
  - Semaphore init to 0

```
A                    B

down()

                     do task

                     up()

continue exec
```



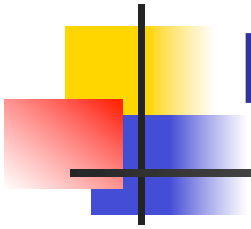
Producer-consumer with semaphores

- mutex: ensures mutual exclusion
- fullBufs: counts the number of full buffers (initialized to 0)
- emptyBufs: counts the number of empty buffers (initialized to N)

Producer-consumer with semaphores

```
producer(){

}
```

```
consumer(){

}
```



Producer-consumer with semaphores

```
producer(){
  down(emptyBufs);

  down(mutex);

  numCokes++;

  up(mutex);

  up(fullBufs);
}
```

```
consumer(){
  down(fullBufs);

  down(mutex);

  numCokes--;

  up(mutex);

  up(emptyBufs);
}
```