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

2/9/11



Interface

- readerStart()
- readerFinish()
- writerStart()
- writerFinish()
- Many threads can be in between a readStart and readerFinish
- Only one thread can be between writerStart and writierFinish

2/10/11 2



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?

2/13/11



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?

2/13/11 4



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?

2/13/11



5

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)

2/13/11 6

```
Pasic Implementation

readerStart() { readerFinish() {

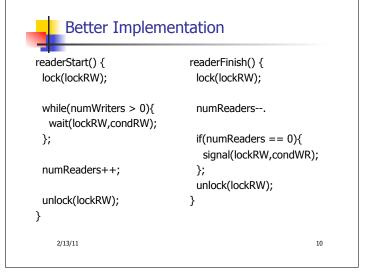
}
}

2/13/11 7
```

```
readerStart() { readerFinish() { lock(lockRW); lock(lockRW); numReaders--; wait(lockRW,condRW); }; broadcast(lockRW,condWR); numReaders++; unlock(lockRW); } unlock(lockRW); }

2/13/11 8
```

```
Basic Implementation
writerStart() {
                             writerFinish() {
 lock(lockRW);
                               lock(lockRW);
 while(numReaders > 0||
                               numWriters--;
      numWriters >0){
  wait(lockRW,condRW);
                               broadcast(lockRW,condWR);
 };
 numWriters++;
                               unlock(lockRW);
 unlock(lockRW);
}
    2/13/11
```



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?

```
2/13/11 11
```

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

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

numReaders++;

unlock(lockRW);
} 2/16/11 12
```

Write Priority

```
writerStart() {
 lock(lockRW);
 waitingWriters ++;
 while(numReaders > 0||
       numWriters >0){
  wait(lockRW,condRW);
 waitingWriters--;
 numWriters++;
 unlock(lockRW);
}
    2/16/11
                                                         13
```



Lock and Reader / Writer Locks

- Reader-writer functions are similar to standard Incks
 - 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
 - Thread in between writerStart and writerFinish "holds a write lock"
- Compare reader-writer locks with standard locks

2/13/11



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

2/13/11 15



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
 - Up() sets value to 1, waking up waiting down

2/13/11 16



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

2/16/11 17



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

```
down()
                do task
                up()
continue exec
```

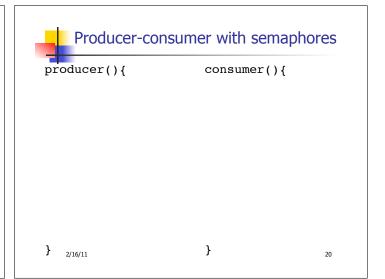
2/16/11 18



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)

2/16/11 19



Producer-consumer with semaphores

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

  down(mutex);
  down(mutex);

  numCokes++;
   numCokes--;

  up(mutex);
  up(mutex);

  up(fullBufs);
  up(emptyBufs);
}
```