EECS 482
Introduction to Operating Systems

Winter 2018

Baris Kasikci

Slides by: Harsha V. Madhyastha

Producer-consumer with monitors

Consumer

```
cokeLock.lock()

while (numCokes == 0) {
   waitingCons&Prod.wait()
}

take coke out of machine
numCokes--

waitingCons&Prod.signal()

cokeLock.unlock()
```

Producer

```
cokeLock.lock()

while (numCokes == MAX) {
    waitingCons&Prod.wait()
}

add coke to machine
numCokes++

waitingCons&Prod.signal()

cokeLock.unlock()
```

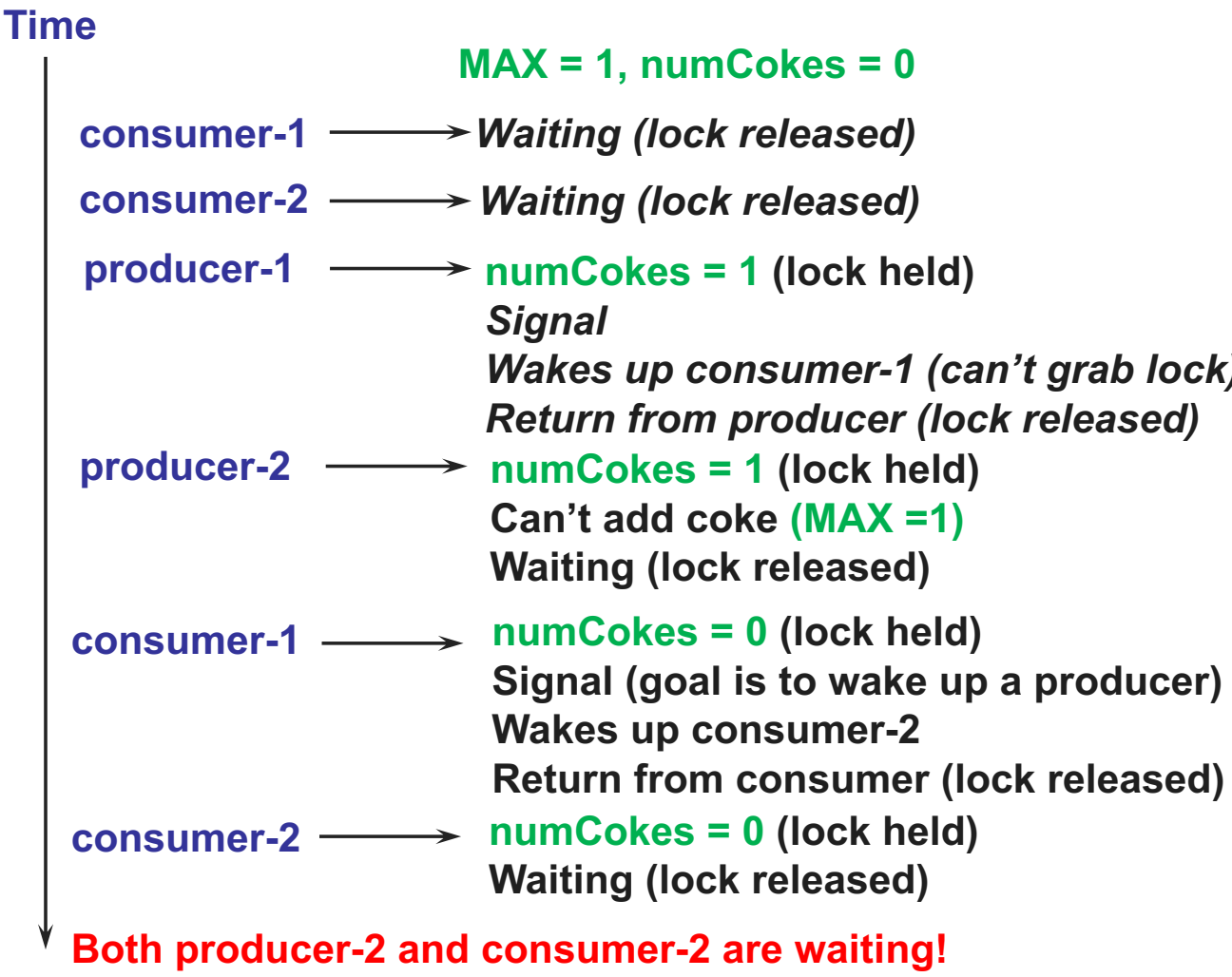
Producer-consumer with monitors

Consumer

```
cokeLock.lock()
while (numCokes == 0) {
   waitingCons&Prod.wait()
}
take coke out of machine
numCokes--
waitingCons&Prod.signal()
cokeLock.unlock()
```

Producer

```
cokeLock.lock()
while (numCokes == MAX) {
    waitingCons&Prod.wait()
}
add coke to machine
numCokes++
waitingCons&Prod.signal()
cokeLock.unlock()
```



Producer-consumer with monitors

Consumer

```
cokeLock.lock()

while (numCokes == 0) {
   waitingConsumers.wait()
}

take coke out of machine
numCokes--

waitingProducers.signal()

cokeLock.unlock()
```

Producer

```
cokeLock.lock()

while (numCokes == MAX) {
    waitingProducers.wait()
}

add coke to machine
numCokes++

if (numCokes == 1) {
    waitingConsumers.signal()
}

cokeLock.unlock()
```

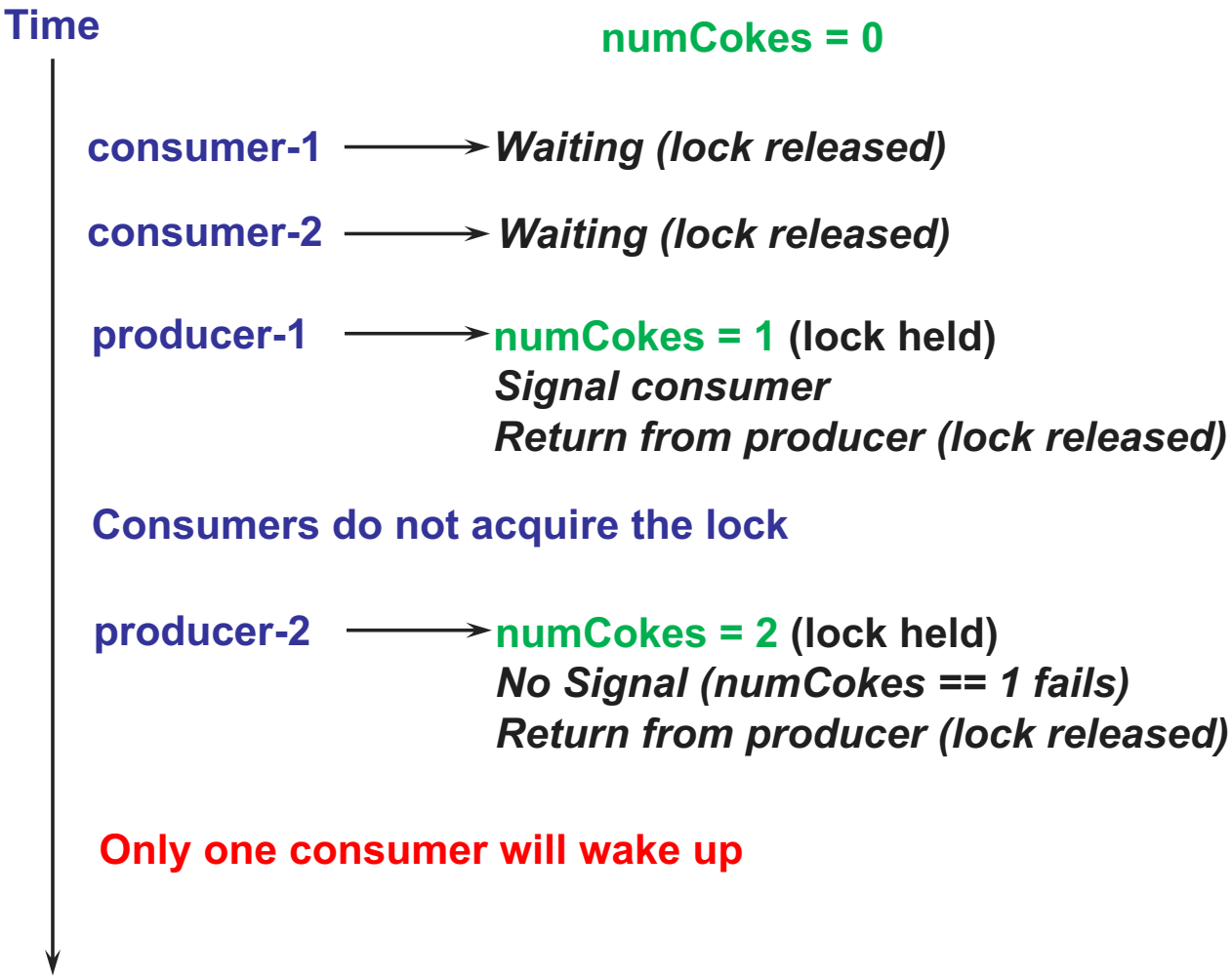
Producer-consumer with monitors

Consumer

```
cokeLock.lock()
while (numCokes == 0) {
   waitingConsumers.wait()
}
take coke out of machine
numCokes--
waitingProducers.signal()
cokeLock.unlock()
```

Producer

```
cokeLock.lock()
while (numCokes == MAX) {
    waitingProducers.wait()
}
add coke to machine
numCokes++
if (numCokes == 1) {
    waitingConsumers.signal()
}
cokeLock.unlock()
```



Implementing reader-writer locks with monitors

- Shared data needed to implement readerStart, readerFinish, writerStart, writerFinish?
  - *numReaders*
  - *numWriters*
- Use one lock (*rwLock*)
- Condition variables?
  - *waitingReaders*: readers must wait if there are writers
  - *waitingWriters*: writers must wait if there are readers or writers

Implementing reader-writer locks with monitors

```
readerStart () {
    rwLock.lock()
    while (numWriters > 0) {
        waitingReaders.wait()
    }
    numReaders++
    rwLock.unlock()
}
readerFinish() {
    rwLock.lock()
    numReaders--
    if (numReaders == 0) {
        waitingWriters.signal()
    }
    rwLock.unlock()
}
```

```
writerStart() {
    rwLock.lock()
    while (numReaders > 0 || numWriters > 0) {
        waitingWriters.wait()
    }
    numWriters++
    rwLock.unlock()
}
writerFinish() {
    rwLock.lock()
    numWriters--
    waitingReaders.broadcast()
    waitingWriters.signal()
    rwLock.unlock()
}
```

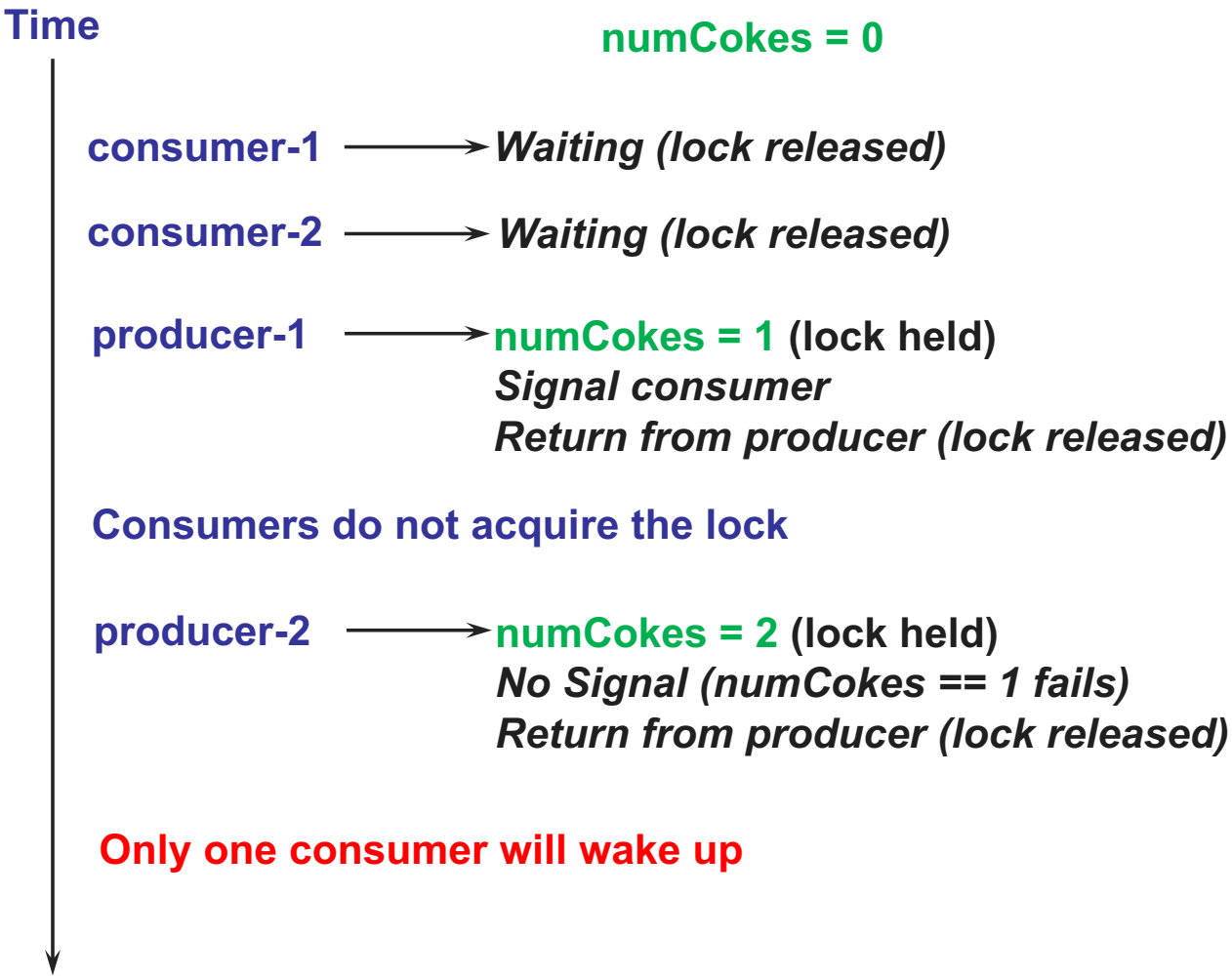
Producer-consumer with monitors

Consumer

```
cokeLock.lock()
while (numCokes == 0) {
   waitingConsumers.wait()
}
take coke out of machine
numCokes--
waitingProducers.signal()
cokeLock.unlock()
```

Producer

```
cokeLock.lock()
while (numCokes == MAX) {
    waitingProducers.wait()
}
add coke to machine
numCokes++
if (numCokes == 1) {
    waitingConsumers.signal()
}
cokeLock.unlock()
```



Implementing reader-writer locks with monitors

```
readerStart () {
    rwLock.lock()
    while (numWriters > 0) {
        waitingReaders.wait()
    }
    numReaders++
    rwLock.unlock()
}
readerFinish() {
    rwLock.lock()
    numReaders--
    if (numReaders == 0) {
        waitingWriters.signal()
    }
    rwLock.unlock()
}
```

```
writerStart() {
    rwLock.lock()
    while (numReaders > 0 || numWriters > 0) {
        waitingWriters.wait()
    }
    numWriters++
    rwLock.unlock()
}
writerFinish() {
    rwLock.lock()
    numWriters--
    waitingReaders.broadcast()
    waitingWriters.signal()
    rwLock.unlock()
}
```

Implementing reader-writer locks with monitors

```
readerStart () {
    rwLock.lock()
    while (numWriters > 0) {
        waitingReaders.wait()
    }
    numReaders++
    rwLock.unlock()
}
readerFinish() {
    rwLock.lock()
    if (numReaders == 1) {
        waitingWriters.signal()
    }
    numReaders--
    rwLock.unlock()
}
```

```
writerStart() {
    rwLock.lock()
    while (numReaders > 0 || numWriters > 0) {
        waitingWriters.wait()
    }
    numWriters++
    rwLock.unlock()
}
writerFinish() {
    rwLock.lock()
    numWriters--
    waitingReaders.broadcast()
    waitingWriters.signal()
    rwLock.unlock()
}
```

Implementing reader-writer locks with monitors

```
readerStart () {
    rwLock.lock()
    while (numWriters > 0) {
        waitingReaders.wait()
    }
    numReaders++
    rwLock.unlock()
}
readerFinish() {
    rwLock.lock()
    numReaders--
    if (numReaders == 0) {
        waitingWriters.signal()
    }
    rwLock.unlock()
}
```

```
writerStart() {
    rwLock.lock()
    while (numReaders > 0 || numWriters > 0) {
        waitingWriters.wait()
    }
    numWriters++
    rwLock.unlock()
}
writerFinish() {
    rwLock.lock()
    numWriters--
    waitingReaders.broadcast()
    waitingWriters.signal()
    rwLock.unlock()
}
```

Implementing reader-writer locks with monitors

- What will happen if a writer finishes and there are several waiting readers and writers?
- How long will a writer wait?
- How to give priority to a waiting writer?

Recap

- Multi-threaded code must synchronize access to shared data

- High-level synchronization primitives:
  - Locks: Mutual exclusion
  - Condition variables: Ordering constraints
  - Monitors: Lock + condition variables

- Today: Semaphores

Semaphores

- Generalized lock/unlock
- Definition:
  - A non-negative integer (initialized to user-specified value)
  - down(): wait for semaphore value to become positive, then atomically decrement semaphore value by 1

```
do {
    if (value > 0) {
        value--
        break
    }
} while (1)
```

Atomic

  - up(): increment semaphore value by 1

Two types of semaphores

- Mutex semaphore (or binary semaphore)
  - Represents single resource (critical section)
  - Guarantees mutual exclusion

- Counting semaphore (or general semaphore)
  - Represents a resource with many units (e.g. Coke machine), or a resource that allows concurrent access (e.g., reading)
  - Multiple threads can “hold” the semaphore
    - » Number determined by the semaphore “count”

Benefit of Semaphores

- Mutual exclusion
  - Initial value is 1

```
down()
critical section
up()
```

- Ordering constraints
  - Usually, initial value is 0
  - Example: thread A wants to wait for thread B to finish

Thread A

```
down()


continue execution
```

Thread B

```
do task
up()
```

Implementing producer-consumer with semaphores

- Semaphore assignments
  - *mutex*: ensures mutual exclusion around code that manipulates coke machine
  - *fullSlots*: counts no. of full slots in the coke machine
  - *emptySlots*: counts no. of empty slots in machine
- Initialization values?

Implementing producer-consumer with semaphores

```
Semaphore mutex = 1;   // mutual exclusion to shared set of slots
Semaphore emptySlots = N;  // count of empty slots (all empty to start)
Semaphore fullSlots = 0;    // count of full slots (none full to start)
```

```
producer {
    // wait for empty slot
    emptySlots.down();

    mutex.down();
    Add coke to the machine
    mutex.up();

    // note a full slot
    fullSlots.up();
}
```

```
consumer {
    // wait for full slot
    fullSlots.down();

    mutex.down();
    Take coke out of machine
    mutex.up();

    // note an empty slot
    emptySlots.up();
}
```

Implementing producer-consumer with semaphores

- Why do we need different semaphores for fullSlots and emptySlots?
- Does the order of down() (e.g., fullSlots, mutex) matter?
- Does the order of up() matter?
- What if there's 1 full slot, and multiple consumers call down() at the same time?
- What if a context switch happens between emptySlots.down() and mutex.down()?
- What if fullSlots.up() before mutex.down()?

Reminders

- Project 1 due on Monday
- Work through discussion questions about monitors before the discussion section

Comparing monitors and semaphores

- Semaphores provide 1 mechanism that can accomplish both mutual exclusion and ordering (monitors use different mechanisms for each)
  - Elegant
  - Can be difficult to use

- Monitor lock = binary semaphore (initialized to 1)
  - lock() = down()
  - unlock() = up()

Condition variable versus semaphore

Condition variable	Semaphore
while(!cond) {wait();}	down()
Can safely handle spurious wakeups	No spurious wakeups
Conditional code in user program	Conditional code in semaphore definition
User writes customized condition; more flexible	Condition specified by semaphore definition (wait if value == 0)
User provides shared variable; protects with lock	Semaphore provides shared variable (integer) and thread-safe operations on that variable (down, up)
No memory of past signals	Remembers past up calls

Condition variable versus semaphore

Condition variable	Semaphore
while(!cond) {wait();}	down()
Can safely handle spurious wakeups	No spurious wakeups
Conditional code in user program	Conditional code in semaphore defi
User w ; more f	Con defi nore
User provides shared variable; protects with lock	Semaphore provides shared variable (integer) and thread-safe operations on that variable (down, up)
No memory of past signals	Remembers past up calls

```
T1: wait()
T2: signal()
T3: signal()
T4: wait()
```

```
T1: down()
T2: up()
T3: up()
T4: down()
```

Implementing custom waiting condition with semaphores

- Semaphores work best if the shared integer and waiting condition (value==0) map naturally to problem domain

- How to implement custom waiting condition with semaphores?

Producer-consumer with monitors

Consumer

```
cokeLock.lock()

while (numCokes == 0) {
   waitingConsumers.wait()
}

take coke out of machine
numCokes--

waitingProducers.signal()

cokeLock.unlock()
```

Producer

```
cokeLock.lock()

while (numCokes == MAX) {
    waitingProducers.wait()
}

add coke to machine
numCokes++

waitingConsumers.signal()

cokeLock.unlock()
```

Producer-consumer with semaphores (monitor style)

Consumer

```
mutex.down()
while (numCokes == 0) {

    go to sleep

}
take coke out of machine
numCokes—

wake up waiting producer, if any

mutex.up()
```

Producer

```
mutex.down()
while (numCokes == MAX) {

    go to sleep

}
add coke to machine
numCokes++

wake up waiting consumer, if any

mutex.up()
```

Producer-consumer with semaphores (monitor style)

Consumer

```
mutex.down()
while (numCokes == 0) {
    semaphore s = 0
    waitingConsumers.push(&s)

    s.down()

}
take coke out of machine
numCokes—
if (!waitingProducers.empty()) {
    waitingProducers.front()->up()
    waitingProducers.pop()
}
mutex.up()
```

Producer

```
mutex.down()
while (numCokes == MAX) {
    semaphore s = 0
    waitingProducers.push(&s)

    s.down()

}
add coke to machine
numCokes++
if (!waitingConsumers.empty()) {
    waitingConsumers.front()->up()
    waitingConsumers.pop()
}
mutex.up()
```

Producer-consumer with semaphores (monitor style)

Consumer

```
mutex.down()
while (numCokes == 0) {
    semaphore s = 0
    waitingConsumers.push(&s)
    mutex.up()
    s.down()
    mutex.down()
}
take coke out of machine
numCokes—
if (!waitingProducers.empty()) {
    waitingProducers.front()->up()
    waitingProducers.pop()
}
mutex.up()
```

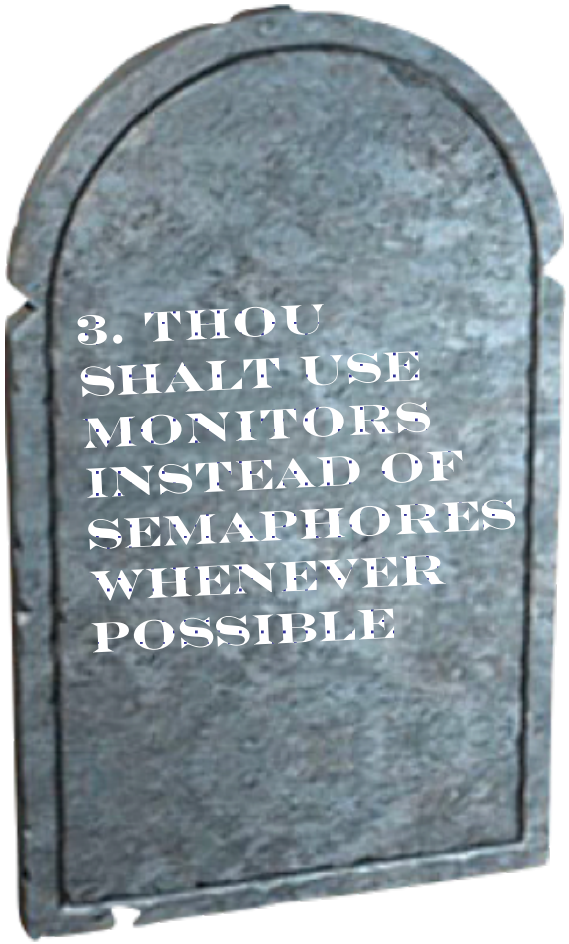
Producer

```
mutex.down()
while (numCokes == MAX) {
    semaphore s = 0
    waitingProducers.push(&s)
    mutex.up()
    s.down()
    mutex.down()
}
add coke to machine
numCokes++
if (!waitingConsumers.empty()) {
    waitingConsumers.front()->up()
    waitingConsumers.pop()
}
mutex.up()
```

The first commandment

Would you name your kid “Kid”?
Or “KidA”? Or “MyKid”? Or “k”?

The third commandment



We gave you monitors so you don't have to worship the ancient gods!

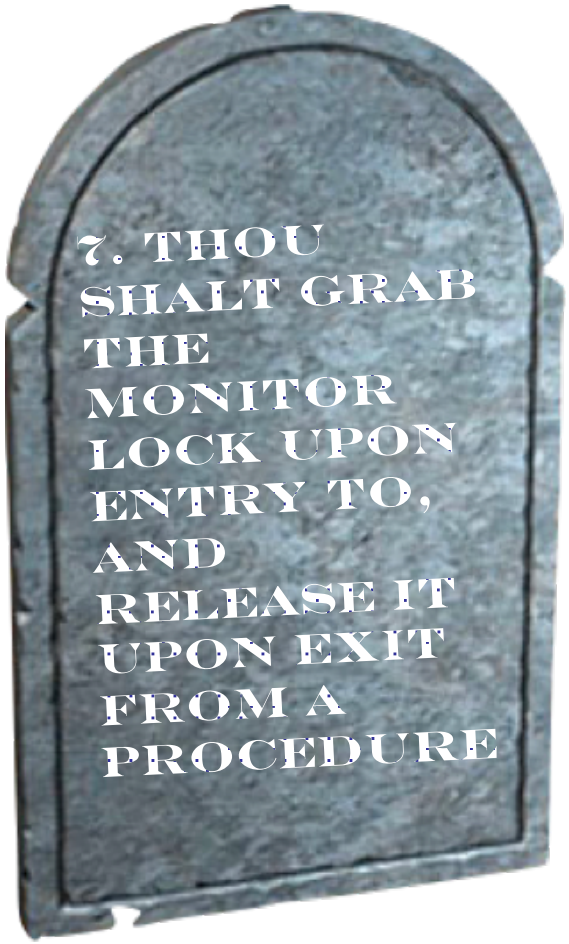
The fifth commandment

This is NOT OK:

```
while(true) {
    mutex.lock()
    if(condition) {
        mutex.unlock()
        break;
    } else {
        mutex.unlock()
        sleep(200);
    }
}
```

The sixth commandment

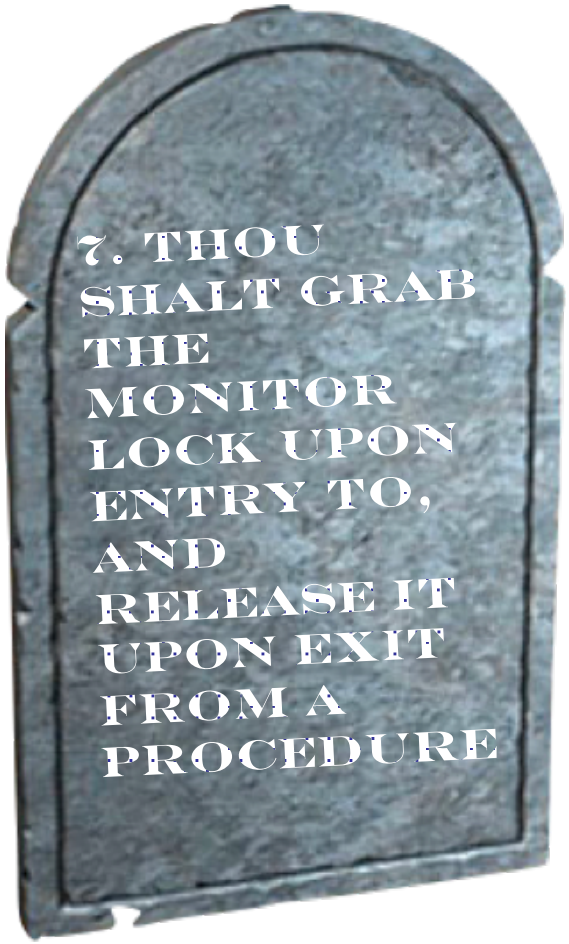
The seventh commandment




```
myAtomicFunction() {
    mutex.lock()
    …
    …
    …
    …
    mutex.unlock()
}
```


IRQL_NOT_DISPATCH_LEVEL
If this is the first time you've seen this error screen, restart your computer. If this screen appears again, follow these steps:
Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.
If problems continue, disable or remove any newly installed hardware. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.
Technical Information:
*** STOP: 0X00000ed (0X80F128D0, 0xc000009c, 0x00000000, 0x0000000)
Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further assistance.
More Info : https://msdn.microsoft.com/en-us/library/windows/hardware/ff559278(v=vs.85).aspx
For technical support assistance call : 1-855-596-2695 (USA-Canada)

The seventh commandment




```
myAtomicFunction() {
    mutex.lock()
    …
    …
    …
    …
    mutex.unlock()
}
```