A Hierarchy of Critical Sections

- We have two different critical sections...
- ...that occur at different levels of abstraction
  - the first relies on a R/W lock
    - protects access to some database (say)
    - allows multiple readers in the CS
  - the second relies on split binary semaphores
    - protects the shared variables \( nreaders, r\_gate\_count \), etc) and implements the conditions we use to implement R/W locks
    - allows only one thread at a time in CS
Starvation

- Our R/W implementation can starve writers
- Change the waiting and release conditions:
  - when a reader tries to enter the CS, wait if
    there is a writer in the CS or
    writers at the write gate waiting to enter CS
  - exiting reader prioritizes releasing a waiting writer
  - exiting writer prioritizes releasing a waiting reader

See Chapter 17 in the Harmony book
Conditional Critical Sections

We know of two ways to implement them:

<table>
<thead>
<tr>
<th>Busy Waiting</th>
<th>Split Binary Semaphores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait for condition in loop, acquiring lock before testing for condition, and releasing it if condition does not hold</td>
<td>Use a collection of binary semaphores and keep track of state, including information about waiting threads</td>
</tr>
<tr>
<td>Easy to understand the code</td>
<td>State tracking is complicated</td>
</tr>
<tr>
<td>OK-ish for true multi-core, but bad for virtual threads</td>
<td>Good for both multicore and virtual threading</td>
</tr>
</tbody>
</table>
Language support?

- Can the programming language be more helpful here?
  - Helpful syntax
  - Or at least some library support
Enter Monitors

- Collect shared data into an object/module
- Define methods for accessing shared data
- Separate the concerns of mutual exclusion and condition synchronization

Monitors are comprised of

- one mutex lock, and
- zero or more condition variables for managing concurrent access to shared data
An abstraction for conditional synchronization associated with a monitor

Enable threads to wait for a given condition to hold while inside the monitor (after releasing the monitor lock) and be alerted when the condition holds.

A misnomer
- can neither be read nor set to a value
- think of a condition variable as a label associated with a condition and a queue
- threads wait in the queue (inside the monitor) until notified that condition holds

Condition Variables
Resource Variables

- Each condition variable should be associated with a **resource variable (RV)** tracking the state of that resource
  - e.g., the number of reader slots that have been claimed
  - It is your job to maintain the RV!

- Check its RV before calling `wait()` on a condition variable to ensure the resource is truly unavailable

- Once the resource is available, claim it (subtract the amount you are using!)

- Before notifying you are releasing a resource, indicate it has become available by increasing the corresponding RV
Two Types of Monitors

Hoare Monitors

Mesa Monitors

Different semantics as to what happens when a thread waiting on a condition is alerted that the condition holds
Hoare Monitors
Tony Hoare, 1974

- Syntactic sugar above split binary semaphores
  - monitor: one thread can execute at a time
  - wait(cond. var.): thread waits for given condition
  - signal(cond. var.): transfer control to a thread waiting for the given condition, if any

```
single resource: monitor
begin
  busy: Boolean;
  nonbusy: condition;
  procedure acquire;
  begin
    if busy then nonbusy.wait;
    busy := true
  end;
  procedure release;
  begin
    busy := false;
    nonbusy.signal
  end;
  busy := false; comment initial value;
end single resource
```

Similar construct proposed by Per Brinch Hansen in 1973
import synch

def Monitor():
    result = synch.Lock()

def enter(mon):
    synch.acquire(mon)

def exit(mon):
    synch.release(mon)

def Condition():
    result = { .sema: synch.BinSema(True), .count: 0 }

def wait(cond, mon):
    cond→count += 1
    exit(mon)
    synch.acquire(?cond→sema)
    cond→count -= 1

def signal(cond, mon):
    if cond→count > 0:
        synch.release(?cond→sema)
        enter(mon)
What happens when a thread signals?

**Hoare semantics:**
- Signaling thread is suspended and, atomically, ownership of the lock is passed to one of the waiting threads, whose execution is immediately resumed.
- Signaling thread is resumed if former waiter exits monitor, or if it waits again.
Producer/Consumer with Bounded Buffer

```
import hoare

def BB(size):
    result = {
        .mon: hoare.Monitor(),
        .prod: hoare.Condition(), .cons: hoare.Condition(),
        .buf: { x(): for x in {1..size} },
        .head: 1, .tail: 1,
        .count: 0, .size: size
    }

def put(bb, item):
    hoare.enter(?bb->mon)
    if bb->count == bb->size:
        hoare.wait(?bb->prod, ?bb->mon)
    bb->buf[bb->tail] = item
    bb->tail = (bb->tail % bb->size) + 1
    bb->count += 1
    hoare.signal(?bb->cons, ?bb->mon)
    hoare.exit(?bb->mon)
```
Producer/Consumer with Bounded Buffer

```
import hoare

def BB(size):
    result = {
        .mon: hoare.Monitor(),
        .prod: hoare.Condition(), .cons: hoare.Condition(),
        .buf: { x(): for x in {1..size} },
        .head: 1, .tail: 1,
        .count: 0, .size: size
    }

def put(bb, item):
    hoare.enter(?bb→mon)
    if bb→count == bb→size:
        hoare.wait(?bb→prod, ?bb→mon)
    bb→buf[bb→tail] = item
    bb→tail = (bb→tail % bb→size) + 1
    bb→count += 1
    hoare.signal(?bb→cons, ?bb→mon)
    hoare.exit(?bb→mon)
```

- N + 1 semaphores abstracted away
- Circular buffer
- Enter monitor
- Exit monitor
- Wait if full
- Signal() passes the baton immediately if there are waiting consumers
Producer/Consumer with Bounded Buffer

```python
def get(bb):
    hoare.enter(bb->mon)
    if bb->count == 0:
        hoare.wait(bb->cons, bb->mon)
    result = bb->buf[bb->head]
    bb->head = (bb->head % bb->size) + 1
    bb->count -= 1
    hoare.signal(bb->prod, bb->mon)
    hoare.exit(bb->mon)
```

- **enter monitor**
- **exit monitor**
- **wait if empty**
- **signal a producer**
- **signal() passes the baton immediately if there are waiting producers**
Mesa Monitors
Mesa Language, Xerox PAak 1980

- Syntactically similar to Hoare monitors
  - monitors and condition variables

- Semantically closer to busy waiting
  - `wait(cond. var.)`: wait for condition, but may get back the CPU when condition is not satisfied (!)
  - `notify(cond. var.)`: move to ready queue a thread waiting for the condition, if any, but don’t transfer control to it
  - `notifyAll(cond. var.)`: move to ready queue all threads waiting for the condition, but don’t transfer control to any of them
<table>
<thead>
<tr>
<th><strong>Hoare</strong></th>
<th><strong>Mesa</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Signaling is atomic with the resumption of waiting thread</td>
<td>- notify() and notifyAll() are hints</td>
</tr>
<tr>
<td>- shared state cannot change before waiting thread is resumed</td>
<td>- adding them affects performance, never safety</td>
</tr>
<tr>
<td>- safety requires to signal <strong>only</strong> when condition holds</td>
<td>- Shared state must be checked in a loop (the condition could have changed since the thread was notified!)</td>
</tr>
<tr>
<td>- Shared state can be checked using an if statement</td>
<td>- Simple implementation</td>
</tr>
<tr>
<td>- Makes it easier to prove liveness</td>
<td>- Resilient to <strong>spurious</strong> wakeup</td>
</tr>
<tr>
<td>- Tricky to implement</td>
<td></td>
</tr>
</tbody>
</table>
## Hoare vs Mesa Monitors

<table>
<thead>
<tr>
<th>Hoare Monitors</th>
<th>Mesa Monitors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baton passing approach</td>
<td>If at first you don’t succeed... sleep &amp; try again!</td>
</tr>
<tr>
<td><em>signal</em> passes baton</td>
<td><em>notify (all)</em> moves waiting threads back to ready queue</td>
</tr>
<tr>
<td>used by most books</td>
<td>used by most real systems</td>
</tr>
</tbody>
</table>

*Mesa monitors won the test of time...*
Mesa Monitors in Harmony

```python
def Condition():
    result = bag.empty()

def wait(c, lk):
    var cnt = 0
    let _, ctx = save():
        atomically:
            cnt = bag.multiplicity(!c, ctx)
            !c = bag.add(!c, ctx)
            !lk = False
            atomically when (not !lk) and (bag.multiplicity(!c, ctx) <= cnt):
                !lk = True

def notify(c):
    atomically if !c != bag.empty():
        !c = bag.remove(!c, bag.bchoose(!c))

def notifyAll(c):
    !c = bag.empty()
```

**Condition:** consists of a bag of threads waiting

**wait:** unlock+add thread context to bag of waiters

**notify:** remove one waiter from the bag of suspended threads

**notifyAll:** remove all waiters from the bag of suspended threads
Reader/Writer Lock Specification (again)

```python
def RWLock():
    result = { .nreaders: 0, .nwriters: 0 }

    def read_acquire(rw):
        atomically when rw→nwriters == 0:
            rw→nreaders += 1

    def read_release(rw):
        atomically rw→nreaders -= 1

    def write_acquire(rw):
        atomically when (rw→nreaders + rw→nwriters) == 0:
            rw→nwriters = 1

    def write_release(rw):
        atomically rw→nwriters = 0
```
Busy-Waiting Implementation

```python
from synch import Lock, acquire, release

def RWLock():
    result = { .lock: Lock(), .nreaders: 0, .nwriters: 0 }

def read_acquire(rw):
    acquire(rw→lock)
    while rw→nwriters > 0:
        release(rw→lock)
        acquire(rw→lock)
    rw→nreaders += 1
    release(rw→lock)

def read_release(rw):
    acquire(rw→lock)
    rw→nreaders -= 1
    release(rw→lock)

def write_acquire(rw):
    acquire(rw→lock)
    while (rw→nreaders + rw→nwriters) > 0:
        release(rw→lock)
        acquire(rw→lock)
    rw→nwriters = 1
    release(rw→lock)

def write_release(rw):
    acquire(rw→lock)
    rw→nwriters = 0
    release(rw→lock)
```

It has the same behaviors as the implementation!

Process continuously scheduled to try to get the lock even if it is not available.
Reader/Writer lock with Mesa monitors

$$\text{From synch import } \ast$$

$$\text{def RWlock():}$$

$$\text{result = \{ }$$

$$\text{.nreaders: 0, .nwriters: 0, .mutex: Lock(), }$$

$$\text{.r\_cond: Condition(), .w\_cond: Condition() }$$

$$\text{\}}$$

It is the mutex that protects nreaders and nwriters, not the R/W lock!

**Invariants**

- If $n$ readers in the critical section, then $nreaders \geq n$
- If $n$ writers in the critical section, then $nwriters \geq n$
- $(nreaders \geq 0 \land nwriters = 0) \lor (nreaders = 0 \land nwriters = \leq 1)$
R/W Lock, Reader

```python
def read_acquire(rw):
    acquire(rw→mutex)
    while rw→nwriters > 0:
        wait(rw→r_cond, rw→mutex)
    rw→nreaders += 1
    release(rw→mutex)

def read_release(rw):
    acquire(rw→mutex)
    rw→nreaders -= 1
    if rw→nreaders == 0:
        notify(rw→w_cond)
    release(rw→mutex)
```

Similar to Busy Waiting

but needs this
R/W Lock, Writer

```python
def write_acquire(rw):
    acquire(?rw→mutex)
    while (rw→nreaders + rw→nwriters) > 0:
        wait(?rw→w_cond, ?rw→mutex)
    rw→nwriters = 1
    release(?rw→mutex)

def write_release(rw):
    acquire(?rw→mutex)
    rw→nwriters = 0
    notifyAll(?rw→r_cond)
    notify(?rw→w_cond)
    release(?rw→mutex)
```

Similar to Busy Waiting

don’t forget anyone!
## Conditional Critical Sections

Let me count the ways...

<table>
<thead>
<tr>
<th></th>
<th>Busy Waiting</th>
<th>Split Binary Semaphores</th>
<th>Mesa Monitors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use a lock and a loop</td>
<td>Use a collection of binary semaphores</td>
<td>Use a lock, a collection of condition variables, and a loop</td>
<td></td>
</tr>
<tr>
<td>Easy to write the code</td>
<td>Just follow the recipe</td>
<td>Notifying is tricky</td>
<td></td>
</tr>
<tr>
<td>Easy to understand the code</td>
<td>Tricky to understand if you don’t know the recipe</td>
<td>Easy to understand the code</td>
<td></td>
</tr>
<tr>
<td>Ok-ish for true multicore, but bad for virtual threads</td>
<td>Good for virtual threading. Thread only runs when it can make progress</td>
<td>Good for both multicore and virtual threading</td>
<td></td>
</tr>
</tbody>
</table>
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;

    int numburers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburers==0)
            hungrykid.wait()
        numburers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Kid and Cook Threads

Monitor BurgerKing {
  Lock mlock;

  int numburers = 0;
  condition hungrykid;

  void kid_eat() {
    mlock.acquire()
    while (numburers == 0)
      hungrykid.wait()
    numburers -= 1
    mlock.release()
  }

  void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
  }
}

cook_main() {
  wake();
  shower();
  drive_to_work();
  while (not_5pm)
    BK.makeburger();
  drive_to_home();
  watch_got();
  sleep();
}

kid_main() {
  dig_in_mud();
  BK.kid_eat();
  bathe();
  draw_on_walls();
  BK.kid_eat();
  facetime_Karthik();
  facetime_Gma();
  BK.kid_eat();
}

girl gets the CPU
Kid and Cook Threads

```
kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```

Monitor BurgerKing {
    Lock mlock;
    int numburtle = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburtle==0)
            hungrykid.wait()
        numburtle -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburtle;
        hungrykid.notify();
        mlock.release()
    }
}
Monitor BurgerKing {
    Lock mlock;
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        numburger++
        hungrykid.notify()
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

dig_in_mud();
BK.kid_eat();
bathe();
draw_on_walls();
BK.kid_eat();
facetime_Karthik();
facetime_Gma();
BK.kid_eat();

girl back on ready Q
Kid and Cook Threads

Monitor BurgerKing {
  Lock mlock;

  int numbburgers = 0;
  condition hungrykid;

  void kid_eat() {
    mlock.acquire()
    while (numburgers == 0)
      hungrykid.wait()
    numbburgers -= 1
    mlock.release()
  }

  void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
  }
}

cook_main() {
  wake();
  shower();
  drive_to_work();
  while(not_5pm)
    BK.makeburger();
  drive_to_home();
  watch_got();
  sleep();
}
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;

    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers==0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}
Kid and Cook Threads

monitor BurgerKing {
  lock mlock;

  int numbburgers = 0;
  condition hungrykid;

  void kid_eat()
  {
    mlock.acquire()
    while (numburgers == 0)
      hungrykid.wait()
    numbburgers -= 1
    mlock.release()
  }

  void makeburger()
  {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
  }
}

cook_main()
{
  wake();
  shower();
  drive_to_work();
  while (not_5pm)
    BK.makeburger();
  drive_to_home();
  watch_got();
  sleep();
}
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;

    int numbigers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire();
        while (numbigers==0)
                hungrykid.wait();
        numbigers -= 1
        mlock.release();
    }

    void makeburger() {
        mlock.acquire();
        ++numburger;
        hungrykid.notify();
        mlock.release();
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;

    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers==0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;

    int numbburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numbburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

boy tries to enter monitor

Running

Ready
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;
    int numburgers = 0;
    condition hungrykid;
    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }
    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

Running

Boy back on ready Q
Kid and Cook Threads

```c
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;

    int numbgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numbgers==0)
            hungrykid.wait()
        numbgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

Kid main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```

Ready

Cook gets the CPU

Running
**Kid and Cook Threads**

```c
kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}
```

```c
cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```

Monitor BurgerKing {
    Lock mlock;
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers==0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}
Kid and Cook Threads

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

Monitor BurgerKing {
    Lock mlock;
    int numburegs = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburegs -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}
Kid and Cook Threads

```c
kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```

Monitor BurgerKing {
    Lock mlock;
    int numbburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numbburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}
kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

Monitor BurgerKing {
    Lock mlock;
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Kid and Cook Threads

```c
kid_main()
{
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main()
{
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```

Monitor BurgerKing
{
    Lock mlock;
    int numbrowsers = 0;
    condition hungrykid;

    void kid_eat()
    {
        mlock.acquire()
        while (numbrowsers==0)
            hungrykid.wait()
        numbrowsers -= 1
        mlock.release()
    }

    void makeburger()
    {
        mlock.acquire()
        ++numbrowsers
        hungrykid.notify()
        mlock.release()
    }
}
Kid and Cook Threads

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

Monitor BurgerKing {
    Lock mlock;
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers==0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}
Kid and Cook Threads

```c
// kid_main()
{
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

// cook_main()
{
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```

Monitor BurgerKing

```c
Monitor BurgerKing {
    Lock mlock;
    int numbrowsers = 0;
    condition hungrykid;
    
    void kid_eat()
    {
        mlock.acquire();
        while (numbrowsers == 0)
            hungrykid.wait();
        numbrowsers -= 1;
        mlock.release();
    }

    void makeburger()
    {
        mlock.acquire();
        ++numbrowsers;
        hungrykid.notify();
        mlock.release();
    }
}
```

- Ready
- Waiting
- Running
**Kid and Cook Threads**

```c
// Kid thread
void kid_eat() {
    mlock.acquire();
    while (numburgers == 0) {
        hungrykid.wait();
        numburgers -= 1;
    }
    mlock.release();
}

// Make burger
void makeburger() {
    mlock.acquire();
    ++numburger;
    hungrykid.notify();
    mlock.release();
}

// Monitor BurgerKing
Monitor BurgerKing {
    Lock mlock;
    int numburgers = 0;
    condition hungrykid;
}

// Cook thread
void cook_main() {
    wake();
    shower();
    drive_to_work();
    while (not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```

**Notes:**
- The `cook_main()` thread wakes up and performs a series of activities before starting to make burgers.
- The `kid_main()` thread挖泥巴 (`dig_in_mud()`), 浴 (`bathe()`), 得到墙壁 (`draw_on_walls()`), 再次吃饭 (`BK.kid_eat()`), 林志颖的视频电话 (`facetime_Karthik()`) 和爷爷的电话 (`facetime_Gma()`) 后再吃饭 (`BK.kid_eat()`).
- The `monitor` blocks the `hungrykid` until the number of burgers is greater than zero.
- The number of burgers is incremented (`++numburger`) before notifying the `hungrykid` thread.

**Thread Status:**
- **Ready:** The `cook_main()` thread is ready to run.
- **Running:** The `cook_main()` thread is currently running.
- **Waiting:** The `kid_main()` thread is waiting for the `numburgers` to be greater than zero.

**Challenge:**
- Boy (with lock) gets the CPU.
Kid and Cook Threads

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

Monitor BurgerKing {
    Lock mlock;
    int numbúrgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers==0)
            hungrykid.wait()
        numbúrgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

Kid and Cook Threads

Ready

Waiting

Q: 0

Waiting

0

Running

no burgers
Kid and Cook Threads

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

Monitor BurgerKing {
    Lock mlock;
    int numbursgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numbursgers==0)
            hungrykid.wait()
        numbursgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

Q: boy releases monitor lock & waits for hungrykid signal

Ready

Waiting

Running

Waiting
Kid and Cook Threads

```
kid_main()
{
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}
```

```
cook_main()
{
    cook;
    joins;
    ready;
    Q;
    with;
    release;
    of;
    monitor;
    lock;
}
```

Monitor BurgerKing {
    Lock mlock;
    int numbburgers = 0;
    condition hungrykid;
    void kid_eat()
    {
        mlock.acquire()
        while (numburgers==0)
            hungrykid.wait()
        numbburgers -= 1
        mlock.release()
    }
    void makeburger()
    {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}
# Kid and Cook Threads

```c
/* Kid routine */
kid_main()
{
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}
```

```c
/* Cook routine */
cook_main()
{
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
    
        BK.makeburger();
        drive_to_home();
        watch_got();
        sleep();
}
```

```c
/* BurgerKing Monitor */
Monitor BurgerKing {
    Lock mlock;
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}
```
Kid and Cook Threads

Monitor BurgerKing {
  Lock mlock;
  int numburgers = 0;
  condition hungrykid;
  void kid_eat() {
    mlock.acquire();
    while (numburgers == 0)
      hungrykid.wait();
    numburgers -= 1
    mlock.release();
  }
  void makeburger() {
    mlock.acquire();
    ++numburger;
    hungrykid.notify();
    mlock.release();
  }
}

kid_main() {
  dig_in_mud();
  BK.kid_eat();
  bathe();
  draw_on_walls();
  BK.kid_eat();
  facetime_Karthik();
  facetime_Gma();
  BK.kid_eat();
}

cook_main() {
  wake();
  shower();
  drive_to_work();
  while(not_5pm)
    BK.makeburger();
  drive_to_home();
  watch_got();
  sleep();
}
Monitor BurgerKing {
    Lock mlock;
    int numbburgers = 0;
    condition hungrykid;
}

void kid_eat() {
    mlock.acquire()
    while (numburgers == 0)
        hungrykid.wait()
    numbburgers -= 1
    mlock.release()
}

void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Kid and Cook Threads

Kid main() {  
dig_in_mud();  
BK.kid_eat();  
bathe();  
draw_on_walls();  
BK.kid_eat();  
facetime_Karthik();  
facetime_Gma();  
BK.kid_eat();  
}

cook main() {  
wake();  
shower();  
drive_to_work();  
while(not_5pm)  
    BK.makeburger();  
drive_to_home();  
watch_got();  
sleep();  
}

Monitor BurgerKing {  
lock mlock;  

int numbubers = 0;  
condition hungrykid;  

void kid_eat() {  
    mlock.acquire()  
    while (numbubers == 0)    
        hungrykid.wait()  
        numbubers -= 1  
    mlock.release()  
}

void makeburger() {  
    mlock.acquire()  
    ++numburger;  
    hungrykid.notify();  
    mlock.release()  
}

cook signals a hungry kid
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;
    int numbburgers = 0;
    condition hungrykid;

    void kid_eat()
    {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numbburgers -= 1
        mlock.release()
    }

    void makeburger()
    {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

kid_main()
{
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main()
{
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Monitor BurgerKing {
Lock mlock;
int numburers = 0;
condition hungrykid;
}

void kid_eat() {
mlock.acquire()
while (numburgers==0)
hungrykid.wait()
numburgers -= 1
mlock.release()
}

void makeburger() {
mlock.acquire()
++numburger;
hungrykid.notify();
mlock.release()
}

cook_main() {
  wake();
  shower();
  drive_to_work();
  while(not_5pm)
    BK.makeburger();
  drive_to_home();
  watch_got();
  sleep();
}

dig_in_mud();
BK.kid_eat();
bathe();
draw_on_walls();
BK.kid_eat();
facetime_Karthik();
facetime_Gma();
BK.kid_eat();

kid_main() {
  BK.kid_eat();
  bathe();
draw_on_walls();
  BK.kid_eat();
  facetime_Karthik();
facetime_Gma();
  BK.kid_eat();
}
Monitor BurgerKing {
    Lock mlock;
    int numboungers = 0;
    condition hungrykid;
}

void kid_eat() {
    mlock.acquire()
    while (numburgers == 0)
        hungrykid.wait()
    numboungers -= 1
    mlock.release()
}

void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while (not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Monitor BurgerKing {
    Lock mlock;

    int numb permanents = 0;
    condition hungry kid;

    void kid_eat() {
        mlock.acquire()
        while (numb permanents == 0)
            hungry kid. wait ()
        numb permanents -= 1
        mlock.release()
    }

    void make burger () {
        mlock.acquire()
        + num burger;
        hungry kid. notify ()
        numb permanents -= 1
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while (not 5 pm)
        BK.make burger ()
    drive_to home();
    watch got();
    sleep();
}
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Monitor BurgerKing {
    Lock mlock;
    int numburgers = 0;
    condition hungrykid;
    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }
    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while (not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}
Monitor BurgerKing {
    Lock mlock;
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers==0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

dig_in_mud();
BK.kid_eat();
bathe();
draw_on_walls();
BK.kid_eat();
facetime_Karthik();
facetime_Gma();
BK.kid_eat();

cook_main() {
    wake();
    shower();
drive_to_work();
    while(not_5pm)
        BK.makeburger();
drive_to_home();
    watch_got();
sleep();
}

Running
girl executes

Q:
1

Ready
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;
    
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Monitor BurgerKing {
    Lock mlock;
    int numb burgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numb burgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;
    int numbburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers==0)
            hungrykid.wait()
        numbburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Monitor BurgerKing {
    Lock mlock;
    int numborners = 0;
    condition hungrykid;
    void kid_eat() {
        mlock.acquire()
        while (numburngers == 0)
            hungrykid.wait()
        numborners -= 1
        mlock.release()
    }
    void makeburger() {
        mlock.acquire()
        ++numburners;
        hungrykid.notify();
        mlock.release()
    }
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Monitor BurgerKing {
  Lock mlock;
  int numburgers = 0;
  condition hungrykid;

  void kid_eat() {
    mlock.acquire()
    while (numburgers == 0)
      hungrykid.wait()
    numburgers -= 1
    mlock.release()
  }

  void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
  }
}

cook_main() {
  wake();
  shower();
  drive_to_work();
  while(not_5pm)
    BK.makeburger();
  drive_to_home();
  watch_got();
  sleep();
}

kid_main() {
  dig_in_mud();
  BK.kid_eat();
  bathe();
  draw_on_walls();
  BK.kid_eat();
  facetime_Karthik();
  facetime_Gma();
  BK.kid_eat();
}
Kid and Cook Threads

Monitor BurgerKing {
  Lock mlock;
  condition hungrykid;

  int numburgers = 0;

  void kid_eat() {
    mlock.acquire()
    while (numburgers == 0)
      hungrykid.wait()
    numburgers -= 1
    mlock.release()
  }

  void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
  }
}

cook_main() {
  wake();
  shower();
  drive_to_work();
  while(not_5pm)
    BK.makeburger();
  drive_to_home();
  watch_got();
  sleep();
}

kid_main() {
  dig_in_mud();
  BK.kid_eat();
  bathe();
  draw_on_walls();
  BK.kid_eat();
  facetime_Karthik();
  facetime_Gma();
  BK.kid_eat();
}

Running

girl moved to ready Q

Ready
Monitor BurgerKing {
    Lock mlock;
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Monitor BurgerKing {
    lock mlock;
    int numburers = 0;
    condition hungrykid;
}

void kid_eat() {
    mlock.acquire()
    while (numburers == 0)
        hungrykid.wait()
    numburers -= 1
    mlock.release()
}

void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while (not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}
Monitor BurgerKing {
    Lock mlock;
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        numburgers += 1
        hungrykid.notify()
        mlock.release()
    }
}

dig_in_mud();
BK.kid_eat();
bathe();
draw_on_walls();
BK.kid_eat();
facetime_Karthik();
facetime_Gma();
BK.kid_eat();
kid_main() {
    wake();
    shower();
drive_to_work();
    while(not_5pm)
        BK.makeburger();
drive_to_home();
    watch_got();
    sleep();
}

cook_main() {
}

boy returns from wait

Ready

Running
Monitor BurgerKing {
  Lock mlock;
  int numburers = 0;
  condition hungrykid;
  void kid_eat() {
    mlock.acquire()
    while (numburgers == 0)
      hungrykid.wait()
    numburers -= 1
    mlock.release()
  }
  void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify()
    mlock.release()
  }
}

kid_main() {
  dig_in_mud();
  BK.kid_eat();
  bathe();
  draw_on_walls();
  BK.kid_eat();
  facetime_Karthik();
  facetime_Gma();
  BK.kid_eat();
}

cook_main() {
  wake();
  shower();
  drive_to_work();
  while(not_5pm)
    BK.makeburger();
  drive_to_home();
  watch_got();
  sleep();
}
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Monitor BurgerKing {
    Lock mlock;
    int numbudget = 0;
    condition hungrykid;
}

void kid_eat() {
    mlock.acquire()
    while (numbudger==0)
        hungrykid.wait()
    numbudget -= 1
    mlock.release()
}

void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Monitor BurgerKing {
    Lock mlock;
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}
and so forth...