9/15/10

No office hours today

- in a meeting

How does waiter know when to serve an eat-in Customer's food?

- Who is involved?
  - Manager? } Bagger? No
  - OT? }
  - Waiter? Yes

Have a list of all "bagged" orders ~~wait~~ for eat-in Customers

An order is taken, food is not cooked completely. Order Taker cannot immediately bag food. It must be bagged in the future.

* When the food is finally ready, who tells the "bagger" to bag the order?

Item 1: Cooks don't know about orders.

Cooks just cook. Everytime they finish cooking an item they increase a monitor variable describing how many of that type of cooked items exist.

Item 2: No one tells the bagger to bag a specific order.

Manager/Cook interaction

An order comes in & cannot be bagged immediately.

Managers don't always start up a Cook when an order is placed.

A Manager needs to wake a $6 burger cook & a Veggie burger cook

- Manager must "tell" a cook what to cook
- Assume the Cooks are on break- waiting on a CoV.

"break" cook

Scenario — Acquire lock

- Manager sets a M.V (for $6 burger)

Release lock → & So & Signals the first cook

- Mgr checks & sees they need to wake up a Veggie burger, so they set M.V (for V burger) & signals another cook

Acquire & Release another lock

Acquire

- Mgr releases "break" cook lock

Proper Scenario - Waking Up a Cook

If I have to make a Cook - make thread

```
Thread *t = new Thread("Cook");
StartCookLock->Acquire();
• set the mv
if( /* I have to make a new Cook */ ){
    t->Fork(Cook, cook's ID);
} else {
    • Signal a cook
}
• Wait for the cook to read the mv

StartCookLock->Release();
```

Midterm #1 Topics

Intro to O.S.

Processes ⟿ Threads

Interprocess communication

- Msg passing
- Global memory

Thread synchronization

- Mutual Exclusion - Locks
- Condition Variables → Monitor
- Semaphores

Project 1

No CPU Scheduling

No Deadlock

Structure of Exam

* 15-20 points — Short Answer/Definition
  25-30 points — Nachos Project 1
  - extension problem of Restaurant simulation - you must write Nachos code.

  20-~~25~~ 30 points — Non-Nachos synchronization problem - pseudocode
  - Might be with Semaphores

* Something else

Semaphores

Can handle all of the problems monitors can handle

Have 2 Operations:

Book	Me	Original/Nachos
Wait	Down	P
Signal	Up	V

Semaphores also have state

- an integer
  - 0 → any positive value

Have wait queue

Up

Increments semaphore value by 1

Wakeup 1 waiting thread

Down

Decrements value by 1 - if not ∅

If value is ∅, thread goes to sleep

Semaphore value is private.

① Mutual Exclusion

Binary semaphore - 0 or 1
- Initialized to 1

to enter a C.R. - Down()

to exit a C.R - Up()

② Synchronization - w/ Semaphores

- Can start with any valid value
- Usage can be an Up, or Down, to start with

Application decision

Human Factors Issue w/ Semaphores

```
{
* a.Up();
                ] CR
  a.Down();
   b.Down();
      }

    c.Down();
      }

  b.Up();
      }
```

Soda Machine (with semaphores)

- Capacity 10
- Shared resource
  - Students take 1 soda
  - SodaGal ~~take~~ add 1 soda

need mutually exclusive access

When does a thread have to go to sleep

- machine is full - SodaGal's sleep
- machine is empty - Students sleep

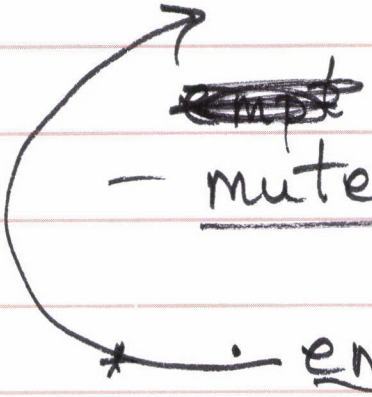
Need 3 semaphores

mutex - "lock" - binary semaphore

- empty - #empty slots - starts @ 10

Full - #full slots - starts @ 0

Soda Gal - Wrong



- mutex.Down(); //Acquire a lock

- empty.Down(); //
  // There is an empty slot I can fill

  //Add soda

full.Up(); // Tell Students 1 soda exists

mutex.Up(); // Release lock

Soda Gal – Right

* empty.Down(); // Reserve an empty slot for me

* mutex.Down();
  //add 1 soda
  mutex.Up();

  full.Up();

Student – Right

• full.Down(); //Reserve a soda for me

  mutex.Down();
  //Take 1 soda
  mutex.Up();

  empty.Up();