Bakers Algorithm

first get ticket number first 3-Lines

other processes cannot modify "ticket" values while another process is doing the same.

for all processes that can get into critical section and are allowed to look at critical section turn toward to = 0.

Advantages of hardware instruction
- works for any number of processes / processors as long as they share memory.
- simple and easy to verify.
- can support multiple critical sections

Con:
- Busy waiting
- Starvation is possible
- deadlock is possible

Intel Exchange instruction

Look @ page 212 in book

Symaphore - an operating system service

Classic problem with symaphore
- producer and consumer problem

Producer

```
while (true)
{  produce.item();
   if ( count == N) sleep();
   enter.item();
   count ++;
   if (count == 1 ) wakeup (consumer)
}
```

Consumer

```
while (true)
{  if (count == 0); sleep()
   remove . item ()
   count --;
   if (count == n-1) wakup (producer)
   Consumer. item()
   }
```

Problem with this method is the idea that the producer can create an item, the consumer can time out before it receives an item and turns over the the producer puter the consumer back to sleep.

This problem is used to introduce Symaphors

Symaphore - Edsgar Dykstra 1965
analogy - flag

2 operations defined in semaphores
Atomic operations
wait signal

Semaphore S; // int value

```
wait (S)
{  S--;
   if (S<0)
      block.process()   // sleep
}

signal (S)
```

```
{  S++;
   if(S <= 0)
      wake up (blocked. process)
}
```

weak Semaphore - randomly pick blocked process
Strong Semaphore - Queue blocked process

Producer + consumer semaphore

```
semaphore empty = N , full = 0 , mutex = 1;

Producer()                      Consumer()
{  while (true)                 {  while (true)
   {  produce.item();              {
      wait(empty);                    wait(full);
      wait(mutex);                    wait(mutex);
      enter.item()                    remove.item()
      signal(mutex);                  signal(mutex);
      signal(full);                   signal(empty);
   }                                  consume.item();
}                                  }
                                }
```

// no more busy waiting.

init semaphore and we can only do two operations on semaphore wait, & signal.

Semaphore S;

```
wait (S){
    S--;
   if (S<0)
    block.proces() }  // Sleep

signal (S) {
    S++;
    if(S <= 0)
      wake up (blocked.proces) }
```

```
mutex = 1
    wait (mutex)
    <C.S.>
    signal (mutex)
```

Note: What are the rules for mutual exclusion?

Spin lock = busy wait loop

Hw Example

① Semaphores matches_paper, tobacco_paper, tobacco_matches, vendor,

```
Vendor ()
{ while (true)
   { choose randomly from 3 ingredients ()
      if (matches_paper)
         signal (matches_paper)
      elif (tobacco_paper)
         signal(tobaccos_paper)
      else(tobacco_matches)
         signal (tobacco_matches)
      wait (vendor);
```

Smoker 1

```
{ while (true) {
  wait (matches_paper)}
  smoke(); signal (vendor);
 }
```

Smoker 2

```
{ while (true) {
  wait (tobacco_paper)
  smoke (); signal (vendo );
 }
```

}

Smoker 3

readers writers problem

wait (tabacco_ matches)

smoke() ; signal (vendor);

int readerCount ;

Semaphore mutex = 1, writer = 1 ;

```
reader ( )
{   wait (mutex)
    readerCount ++
    if (readercount == 1)
        wait(writer);
    signal ( mutex );
    readStuff();
    wait ( mutex );
    readerCount --;
    if (readercount == 0)
        signal (writer)
    signal ( mutex )
}

writer ( ) {
    wait (writer)
    writeStuff()
    signal (writer)
}
```

9/20/18 Day 16

```
Barber()
{
 wait(waitingroom);
 signal(barberchair)
 wait(mutex); chairs++;
 signal(mutex);
 cuthair();
}
```

```
customer() {
 wait(mutex)
 if (chairs >0)
    chairs--; signal(waitingroom);
else
    signal(mutex)
    Leave()
signal(mutex)
wait(barberchair)
```

Readers and writers problem

```
int readercount;
Semaphores mutex=1, writer=1;
```

```
reader()
{  wait(mutex)
   readercount++
   if (readercount ==1)
      wait(writer);
   signal(mutex);
```

```
writer() {
   wait(writer)
   writeStuff()
   signal(writer)
}
```

```
    read stuff();
    wait(mutex);
    reader count--;
    if (reader count == 0)
        signal(writer)
    signal(mutex)
}
```