Chapter 6
Synchronization

Images from Silberschatz

My code is slow

- Don't worry about speed at this point

- Later solutions:
  - use the optimizer with gcc: -O#
  - # is 0,1,2,3
    - 0 do not optimize
  - You will not be able to debug optimized code!

- gprof
  - profiling tool that measures how long you spend in each function
  - gcc -o exec exec.o -pg
  - ./exec
  - gprof ./exec

Race Condition

- How can count++ be executed?
- How can count-- be execute?
- Why is this a problem?
  - Why else is it a problem?
- Atomic

Critical Section Problem

- Critical Section

- Mutual Exclusion

- Progress

- Bounded Waiting

- Preemptive vs non-preemptive kernels

Two-lock Queue

```
structure node_t    {value: data type, next: pointer to node_t}
structure queue_t   {Head: pointer to node_t, Tail: pointer to node_t, H_lock: lock type, T_lock: lock type}

initialize(Q: pointer to queue_t)
        node = new_node()                     # Allocate a free node
        node->next.ptr = NULL                 # Make it the only node in the linked list
        Q->Head = Q->Tail = node              # Both Head and Tail point to it
        Q->H_lock = Q->T_lock = FREE          # Locks are initially free

enqueue(Q: pointer to queue_t, value: data type)
        node = new_node()                     # Allocate a new node from the free list
        node->value = value                   # Copy enqueued value into node
        node->next.ptr = NULL                 # Set next pointer of node to NULL
        lock(&Q->T_lock)                      # Acquire T_lock in order to access Tail
            Q->Tail->next = node              # Link node at the end of the linked list
            Q->Tail = node                    # Swing Tail to node
        unlock(&Q->T_lock)                    # Release T_lock
```

http://www.research.ibm.com/people/m/michael/podc-1996.pdf, Figure 2

Two-lock Queue

```
dequeue(Q: pointer to queue_t, pvalue: pointer to data type): boolean
    lock(&Q–>H_lock)                      # Acquire H_lock in order to access Head
        node = Q–>Head                    # Read Head
        new_head = node–>next             # Read next pointer
        if new_head == NULL               # Is queue empty?
            unlock(&Q–>H_lock)            # Release H_lock before return
            return FALSE                  # Queue was empty
        endif
        *pvalue = new_head–>value         # Queue not empty. Read value before release
        Q–>Head = new_head                # Swing Head to next node
    unlock(&Q–>H_lock)                    # Release H_lock
    free(node)                            # Free node
    return TRUE                           # Queue was not empty, dequeue succeeded
```

http://www.research.ibm.com/people/m/michael/podc-1996.pdf, Figure 2

Just a reminder.....

```
while (true)
{

    /* produce an item and put in nextProduced */

    while(count == BUFFER_SIZE)
        ; // do nothing

    buffer[in] = nextProduced;
    in = (in +1) % BUFFER_SIZE;
    count++;
}
```

```
while (true)
{
    /* consume an item */
    while(count == 0)
        ; // do nothing

    nextConsumed = buffer[out];
    out = (out +1) % BUFFER_SIZE;
    count--;

    /* use nextConsumed */
}
```

- These are two separate threads.
- What are we trying to do?
- What is the problem?

Peterson's Solution

- Assumptions:

- Are the 3 properties preserved?

```
while (true) {
        flag[i] = TRUE;
        turn = j;
        while ( flag[j] && turn == j);

            CRITICAL SECTION

        flag[i] = FALSE;

              REMAINDER SECTION

}
```

Hardware support

- Implement this on the processor
  - Machine instructions

```
boolean TestAndSet (boolean *target)
{
    boolean rv = *target;
    *target = TRUE;
    return rv:
}
```

```
while (true) {
        while ( TestAndSet (&lock ))
                ;   /* do nothing

                //    critical section

        lock = FALSE;

                //     remainder section

}
```

More hardware solutions

- **xchng** on Intel chips
- TestAndSet is really xchng & test

```
while (true)  {
        key = TRUE;
        while ( key == TRUE)
                Swap (&lock, &key );

                   //    critical section

         lock = FALSE;

                  //      remainder section

}
```

```
void Swap (boolean *a, boolean *b)
{
        boolean temp = *a;
        *a = *b;
        *b = temp:
}
```

CompareAndSwap

- **cmpxchg** on Intel Itanium and Intel x86_64
  - sets ZF-bit in EFLAGS to show if the swap occured
- pthreads eventually calls this instruction for pthread_mutex_lock()
- glibc
  - deep in the nptl directory
  - sysdeps/x86_64/bits/atomic.h
  - assembly in the .h file

```
int compareAndSwap(int *value, int compareTo, int setTo)
{
    int origValue = *value;
    if( *value == compareTo)
    {
        *value = setTo;
    }
    return origValue; // could return boolean indicating if swapped
}
```

```
// initialize to FALSE
boolean waiting[n];
boolean lock;
```

```
do{
    waiting[i] = TRUE;
    key = TRUE;
    while(waiting[i] && key)
    {
        key = TestAndSet(&lock);

    }
    waiting[i] = FALSE;

    // critical section
    j = (i + 1) % n;
    while((j != i) && !waiting[j])
    {
        j = (j + 1) %n;
    }

    if(j == i)
    {
        lock = FALSE;
    }
    else
    {
        waiting[j] = FALSE;
    }
    // non-critical section
}while(TRUE);
```

Semaphore

- Counting
- Binary
  - ??
- Spin lock
- Problems?
  - solutions?
- What can we say about Critical Sections?

```
wait (S) {
        while S <= 0
            ; // no-op
          S--;
    }
signal (S) {
      S++;
    }
```

```
Semaphore S;   //  initialized to 1
wait (S);
        Critical Section
signal (S);
```

Linux

- man sem_overview

- sem_init() // initialize, set initial value (may be 0, 1, >1)
- sem_wait() // decrement // block if semaphore is 0
- sem_post() // increment

unnamed semaphores are often shared across processes via shared memory

- sem_open(char*) // open a **named** semaphore
  // like opening a file.
- sem_close()
- sem_unlink() // delete from system

Example

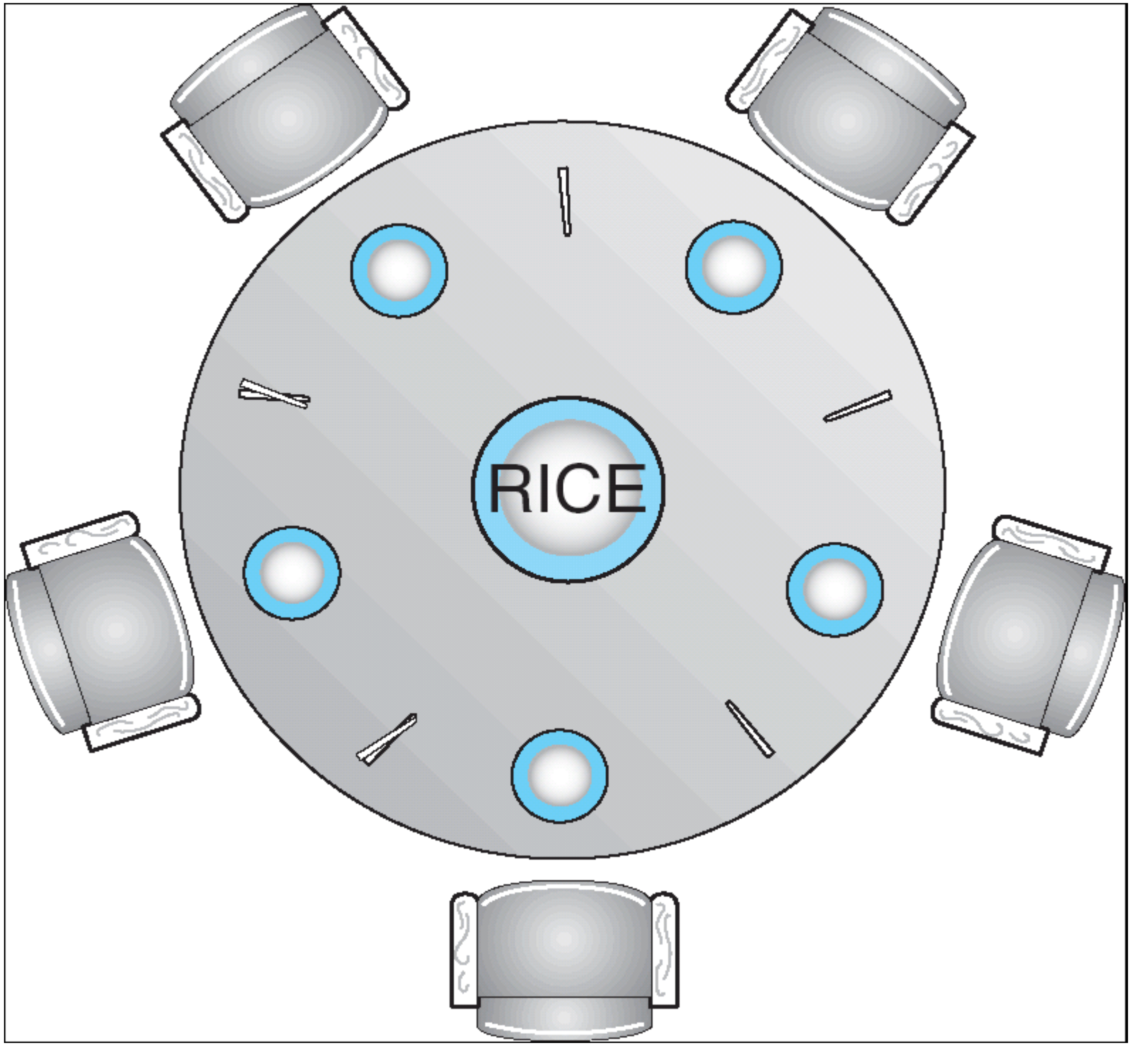
Dangers

- Deadlock
- Starvation
- Priority Inversion

Classic Problems of Synchronization

- Used to test new synchronization methods
- Bounded Buffer
- Readers-Writers
- Dining Philosophers
  - or, why you should never eat at a table full of computer scientists

Dining Philosophers



Dining Philosophers Solution

- Using semaphores
- Problems?
- Solutions?

```
while (true)  {
        wait ( chopstick[i] );
        wait ( chopStick[ (i + 1) % 5] );

            //  eat

        signal ( chopstick[i] );
        signal (chopstick[ (i + 1) % 5] );

            //  think

}
```

Problems with Semaphores

- What can you think of?

- Why are these problems bad?
  - Really, really, really bad?
    - Evil even.

Monitors

- High level coding practice
  - *design pattern*
  - Sometimes part of the language
    - Java: *synchronized*
    - C#: *Monitor* class
    - C++ .NET: *Monitor* class
  - Sometimes you code it yourself
    - C
- Only one process can be in a monitor at a time
- Why is this useful?

```
monitor monitor-name
{
   // shared variable declarations
   procedure P1 (…) { …. }
            …

   procedure Pn (…) {……}

    Initialization code ( ….) { … }
            …
   }
}
```

Log-Based Recovery

- Ensure atomicity
  - In case of a crash
  - Databases
  - Long running computations
    - Weather simulations
    - Nuclear reaction simulations
- Write-ahead logging
  - Start
  - Commit
  - Undo
  - Redo
- Problems?

Checkpoints

Transactional Memory

http://research.cs.wisc.edu/trans-memory/

http://arstechnica.com/hardware/news/2011/08/ibms-new-transactional-memory-make-or-break-time-for-multithreaded-revolution.ars