Classical Problems of Synchronization

- Quick Review
  - mutex
  - semaphore
    - binary
    - counting
- Bounded-Buffer Problem (Producer-consumer)
  - Audio Player
- Readers and Writers Problem
  - Banking system: read acct balances versus update balances
- Dining-Philosophers Problem
  - Set of processes needing to lock multiple resources
    - Disk and Tape (backup)
    - Travel reservation: hotel, airline, car rental databases

Classical Problems of Synchronization

Using Semaphores

- Use is similar to our locks, but semantics are different

```
struct Semaphore {
   int value;
   Queue q;
} S;
withdraw (account, amount) {
   wait(S);
   balance = get_balance(account);
   balance = balance – amount;
   put_balance(account, balance);
   signal(S);
   return balance;
}
```

Threads block

critical section

```
wait(S);
balance = get_balance(account);
balance = balance – amount;
```

```
wait(S);
```

```
wait(S);
```

```
put_balance(account, balance);
signal(S);
```

```
…
signal(S);
```

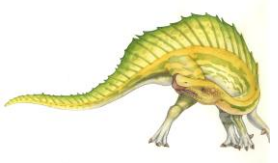
```
…
signal(S);
```

It is **undefined** which thread runs after a signal

http://www.cs.ucr.edu/~harsha/teaching/Winter2012/CS153/lectures/lec6.pdf

Bounded-Buffer Problem

- *N* buffers, each can hold one item
- Semaphore mutex initialized to the value 1
- Semaphore full initialized to the value 0
- Semaphore empty initialized to the value N.



Bounded Buffer Problem (Cont.)

- The structure of the producer process

```
do  {

        //   produce an item in nextp

      wait (empty);
      wait (mutex);

        //  add the item to the  buffer

       signal (mutex);
       signal (full);
  } while (TRUE);
```

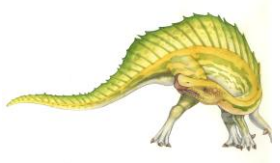
Bounded Buffer Problem (Cont.)

- The structure of the consumer process

```
do {
        wait (full);
        wait (mutex);

              //  remove an item from  buffer to nextc

        signal (mutex);
        signal (empty);

              //  consume the item in nextc

} while (TRUE);
```

Readers-Writers Problem

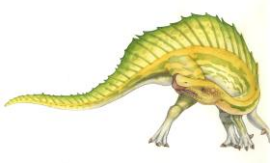
- A data set is shared among a number of concurrent processes
  - Readers – only read the data set; they do **not** perform any updates
  - Writers – can both read and write
- Problem – allow multiple readers to read at the same time. Only one single writer can access the shared data at the same time
- Shared Data
  - Data set
  - Integer readcount initialized to 0 // number of readers
  - Semaphore mutex initialized to 1 // mutual exclusion to readcount
  - Semaphore wrt initialized to 1 // exclusive reader or writer



Readers-Writers Problem (Cont.)

- The structure of a writer process

```
do {
        wait (wrt) ;

            //    writing is performed

        signal (wrt) ;
} while (TRUE);
```

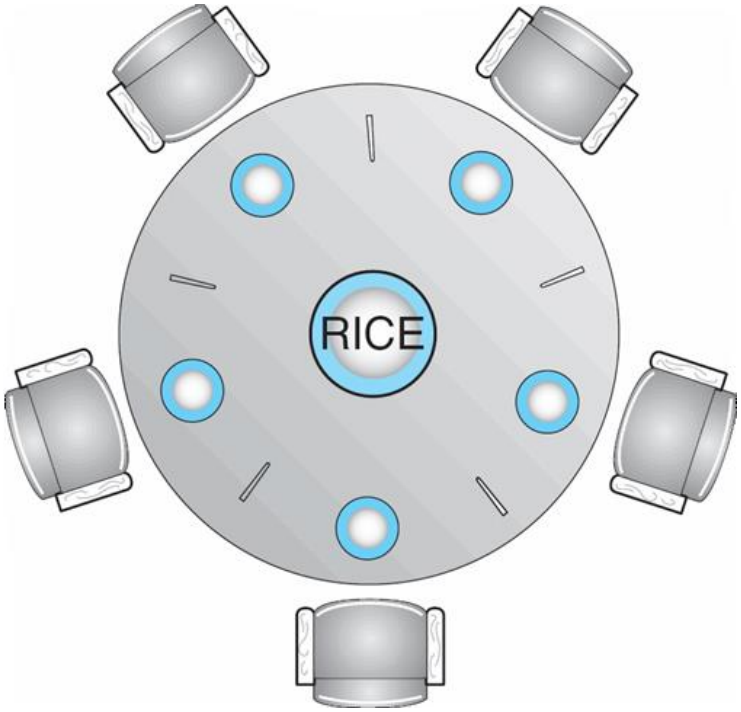


Readers-Writers Problem (Cont.)

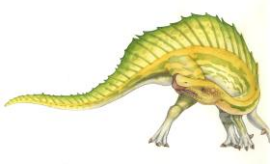
- The structure of a reader process for “first” readers-writers problem

```
do {
        wait (mutex) ;
        readcount ++ ;
        if (readcount == 1)
                wait (wrt) ;
        signal (mutex)

            // reading is performed

         wait (mutex) ;
         readcount  - - ;
         if (readcount  == 0)
                signal (wrt) ;
         signal (mutex) ;
    } while (TRUE);
```

Dining-Philosophers Problem



- Five philosophers think and eat
- Takes 2 chopsticks to eat
- Shared data
  - Bowl of rice (data set)
  - Semaphore chopstick [5] initialized to 1



Dining-Philosophers Problem (Cont.)

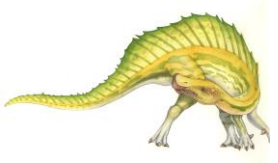
- The structure of Philosopher *i*:

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

            //  eat

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

            //  think

} while (TRUE);
```

Is this a solution?

Problem(s)?

Solution to Problem(s) ?



Problems with Semaphores

- Correct use of semaphore operations are imperative:
- Explain how each of the following can cause problems:
  - signal (mutex) …. wait (mutex)
  - wait (mutex) … wait (mutex)
  - Omitting wait (mutex) or signal (mutex) (or both)