Operating Systems
CS F372

Synchronization

BIJU K RAVEENDRAN

Classical Problems of Synchronization

- Bounded-Buffer Problem
- Readers and Writers Problem
- Dining-Philosophers Problem
- Sleeping Barber Problem

Producer – Consumer Problem

Buffer[ ] (infinite size), in ← 0, out ← 0;

PRODUCER

```
while (1) {
    // Produce item;
    Buffer[in] = item;
    in = in + 1;
}
```

CONSUMER

```
while (1) {
    while (in == out);
    item = Buffer[out];
    out = out +1;
}
```

Bounded Buffer Problem

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

Producer – Consumer Problem

semaphore mutex ← 1, full ← 0; in ← 0, out ← 0;

PRODUCER

```
while (1) {
   // Produce an item;
    wait (mutex);
          Buffer[in] = item;
          in = in + 1;
    signal (mutex);
   signal(full) ;
}
```

CONSUMER

```
while (1) {
    wait (full) ;
    wait (mutex);
          item=Buffer[out];
          out = out +1;
    signal (mutex);
}
```

Producer – Consumer Problem

in ← 0, out ← 0; count ← 0;

PRODUCER

```
while (1) {
    // produce an item
    while (count = = BSIZE);
    buffer[in] = item;
    in = (in + 1) % BSIZE;
    count = count + 1;
}
```

CONSUMER

```
while (1) {
    while (count = = 0);
     item = buffer[out];
    out = (out + 1) % BSIZE;
    count = count – 1;
}
```

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.

Producer – Consumer Problem

semaphore mutex ← 1, full ← 0, empty ← BSIZE;
in ← 0, out ← 0;

PRODUCER

```
while (1) {
   // Produce an item;
   wait(empty);
   wait (mutex);
        Buffer[in] = item;
         in = (in + 1) % BSIZE;
   signal (mutex);
  signal(full) ;
}
```

CONSUMER

```
while (1) {
     wait (full) ;
     wait (mutex);
             item = Buffer[out];
              out = (out + 1) % BSIZE;
     signal (mutex);
     signal (empty);
}
```

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
    - Reader and writer cannot access simultaneously.
    - No other processes are allowed to enter in the critical section when a writer is executing the critical section

Readers – Writers Problem

- Data structure support needed
  - semaphore mutex, wrt ;
  - int readcount ;
- Data structure Initialization
  - mutex = 1
  - wrt = 1
  - readcount = 0 ;

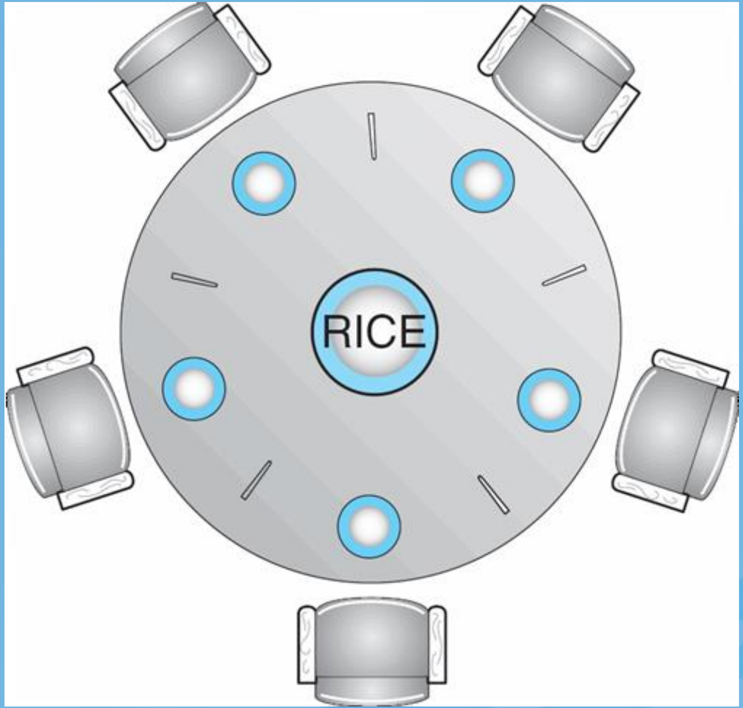
Reader

```
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);
```

Writer

```
do {
      wait ( wrt ) ;
      // writing is performed
      signal ( wrt ) ;
}while(TRUE);
```

Dining Philosophers Problem



Dining Philosophers Problem

- Data structure support needed
  - semaphore chopstick [N] ;

- Data structure Initialization
  - for(int i=0; i< N; i + +)

    chopstick [i] = 1;

Dining Philosophers Problem

- 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);
```