RAMA UNIVERSITY

www.ramauniversity.ac.in

FACULTY OF ENGINEERING & TECHNOLOGY

BCA-307 Operating System

Lecturer-13

Manisha Verma
Assistant Professor
Computer Science & Engineering

Process Synchronization

- Classic Problems of Synchronization
- Monitors
- Synchronization Examples
- Alternative Approaches

Process Synchronization

•Classical problems used to test newly-proposed synchronization scheme

- ➢Bounded-Buffer Problem
- ➢Readers and Writers Problem
- ➢Dining-Philosophers Problem

•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

The structure of the producer process

```
do {
    ...

  /* produce an item in next_produced */
    ...
  wait(empty);

  wait(mutex);
    ...
  /* add next produced to the buffer */
    ...
  signal(mutex);

  signal(full);

} while (true);
```

The structure of the consumer process

```
Do {

   wait(full);

   wait(mutex);
      ...
   /* remove an item from buffer to next_consumed */
      ...
   signal(mutex);

   signal(empty);
      ...
   /* consume the item in next consumed */
      ...
} 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

- Several variations of how readers and writers are considered – all involve some form of priorities

- Shared Data
  - Data set
  - Semaphore **rw_mutex** initialized to 1
  - Semaphore **mutex** initialized to 1
  - Integer **read_count** initialized to 0

The structure of a writer process

```
do {
   wait(rw_mutex);
        ...
   /* writing is performed */
        ...
   signal(rw_mutex);
} while (true);
```

The structure of a reader process

```
do {

    wait(mutex);

    read_count++;

    if (read_count == 1)

       wait(rw_mutex);

    signal(mutex);
        ...
    /* reading is performed */
        ...
    wait(mutex);

    read count--;

    if (read_count == 0)

    signal(rw_mutex);

    signal(mutex);

} while (true);
```

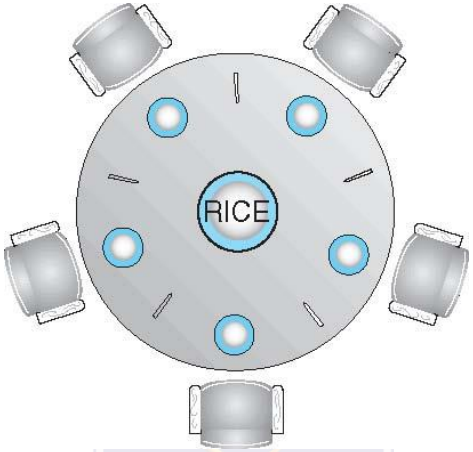
Readers-Writers Problem Variations

First variation – no reader kept waiting unless writer has permission to use shared object

Second variation – once writer is ready, it performs the write

- Both may have starvation leading to even more variations
- Problem is solved on some systems by kernel providing reader-writer locks

Dining-Philosophers Problem



•Philosophers spend their lives alternating thinking and eating

•Don’t interact with their neighbors, occasionally try to pick up 2 chopsticks (one at a time) to eat from bowl

Need both to eat, then release both when done

•In the case of 5 philosophers

Shared data

Bowl of rice (data set)

Semaphore chopstick [5] initialized to 1

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);
```

What is the problem with this algorithm?

Dining-Philosophers Problem Algorithm (Cont.)

- Deadlock handling
  - Allow at most 4 philosophers to be sitting simultaneously at the table.
  - Allow a philosopher to pick up the forks only if both are available (picking must be done in a critical section.
  - Use an asymmetric solution -- an odd-numbered philosopher picks up first the left chopstick and then the right chopstick. Even-numbered philosopher picks up first the right chopstick and then the left chopstick.

MCQ

wait() operation used for……. the semaphore value
A. decrements
B. Increment
C. Both
D. None

Dining-Philosophers Problem
A. an odd-numbered philosopher picks
B. Even-numbered philosopher picks
C. Both
D. None

Reader writer problem……
A. no reader kept waiting unless writer has permission to use shared object
B. once writer is ready, it performs the write
C. Both
D. None

synchronization schemes…….

A. Bounded-Buffer Problem
B. Readers and Writers Problem
C. Dining-Philosophers Problem
D. All of these

Bounded-Buffer Problem:-

A. *n* buffers, each can hold one item
B. Semaphore mutex initialized to the value 1
C. Never hold
D. All of these