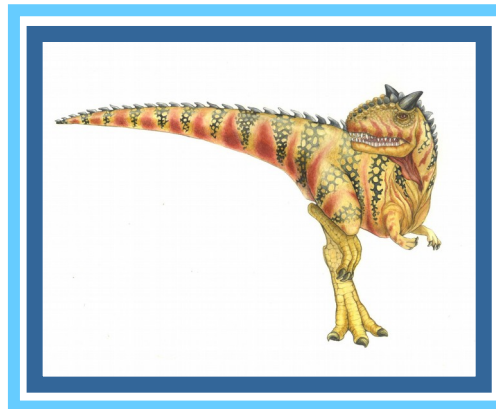


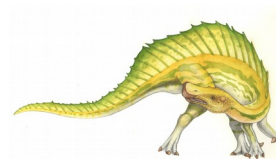
Chapter 7: Synchronization Examples





Chapter 7: Synchronization Examples

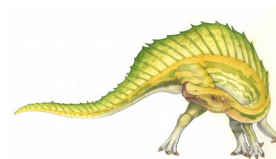
- Classic Problems of Synchronization
- Synchronization Examples
- Alternative Approaches





Objectives

- To examine several classical process-synchronization problems
- To explore several tools that are used to solve process synchronization problems





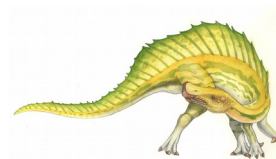
Classical Problems of Synchronization

- Classical problems used to test newly-proposed synchronization schemes
 - Bounded-Buffer Problem
 - Readers and Writers Problem
 - Dining-Philosophers Problem
- We will present solutions using:
 - Semaphores.
 - Monitors
 - Various operating systems





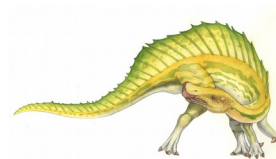
Semaphore Solutions





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

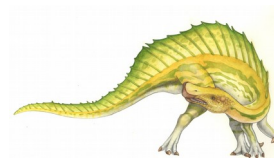




Bounded Buffer Problem (Cont.)

- The structure of the producer process

```
do {  
    ...  
    /* produce an item in next_produced */  
    ...  
    wait(empty);  
    wait(mutex);  
    ...  
    /* add next produced to the buffer */  
    ...  
    signal(mutex);  
    signal(full);  
} while (true);
```

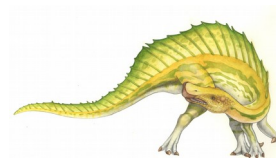


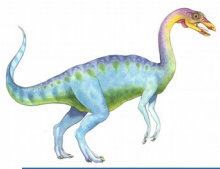


Bounded Buffer Problem (Cont.)

- The structure of the consumer process

```
Do {  
    wait(full);  
    wait(mutex);  
    ...  
    /* remove an item from buffer to 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
 - Semaphore **rw_mutex** initialized to 1
 - Semaphore **mutex** initialized to 1
 - Integer **read_count** initialized to 0

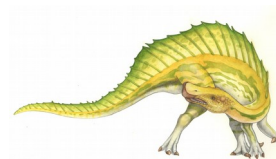




Readers-Writers Problem (Cont.)

- The structure of a writer process

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





Readers-Writers Problem (Cont.)

- 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

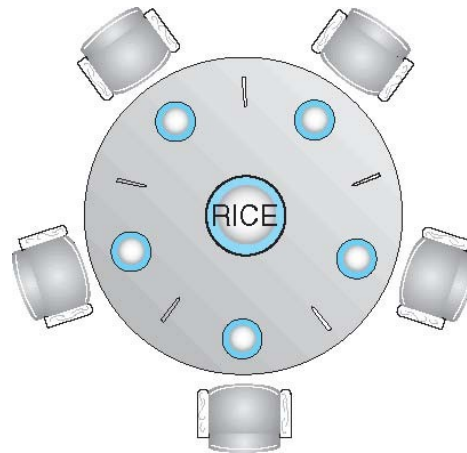
- **First** variation – no reader kept waiting unless writer has permission to use shared object
- **Second** variation – once writer is ready, it performs the write ASAP
- Both may have starvation leading to even more variations
- Problem is solved on some systems by kernel providing reader-writer locks





Dining-Philosophers Problem

- Philosophers spend their lives alternating thinking and eating
- They do not 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, the shared data:
 - Bowl of rice (data set)
 - Semaphore **chopstick [5]** initialized to 1

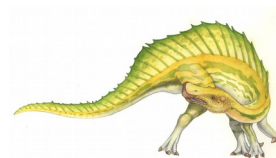




Dining-Philosophers Problem Algorithm

- 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);
```





Dining-Philosophers Problem Algorithm (Cont.)

- This solution guarantees that no two neighbors are eating simultaneously.
- Possibility of a deadlock. Suppose that all five philosophers become hungry at the same time and each grabs the left chopstick.
- Solution:
 - 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.



End of Chapter 7

