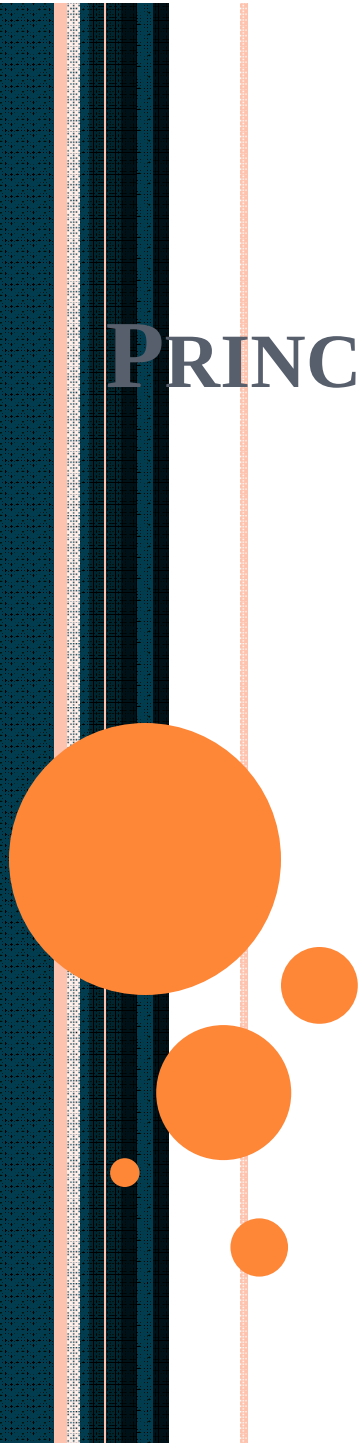
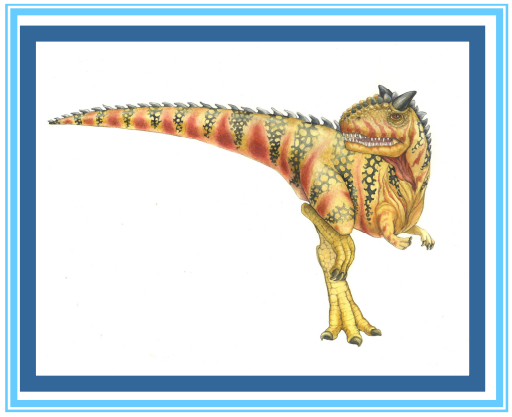
PRINCIPLES OF OPERATING SYSTEMS



Lecture 26
Bounded Buffer Problem



Modified from Silberschatz, Galvin and Gagne & Stallings

Chapter 6: Process Synchronization

- Background
- The Critical-Section Problem
- Peterson's Solution
- Synchronization Hardware
- Semaphores
- Classic Problems of Synchronization
- Monitors
- Synchronization Examples
- Atomic Transactions

Deadlock

- Deadlock – two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes
- Let S and Q be two semaphores initialized to 1

P_0	P_1
wait (S);	wait (Q);
wait (Q);	wait (S);
.	.
.	.
signal (S);	signal (Q);
signal (Q);	signal (S);

- Priority Inversion - Scheduling problem when lower-priority process holds a lock needed by higher-priority process

Starvation

Starvation – indefinite blocking.

A process may never be removed from the semaphore queue in which it is suspended

- Order of arrival retainment:
  - **Weak** semaphores:
    - The thread that will access the critical region next is selected randomly
    - *Starvation is possible*
  - **Strong** semaphores:
    - The thread that will access the critical region next is selected based on its arrival time, e.g. FIFO
    - *Starvation is not possible*

Classical Problems of Synchronization

- 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.

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);
```