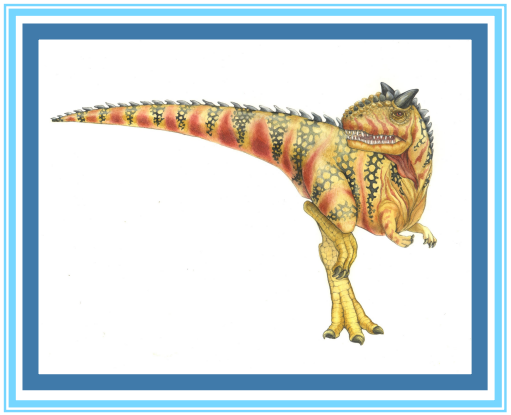
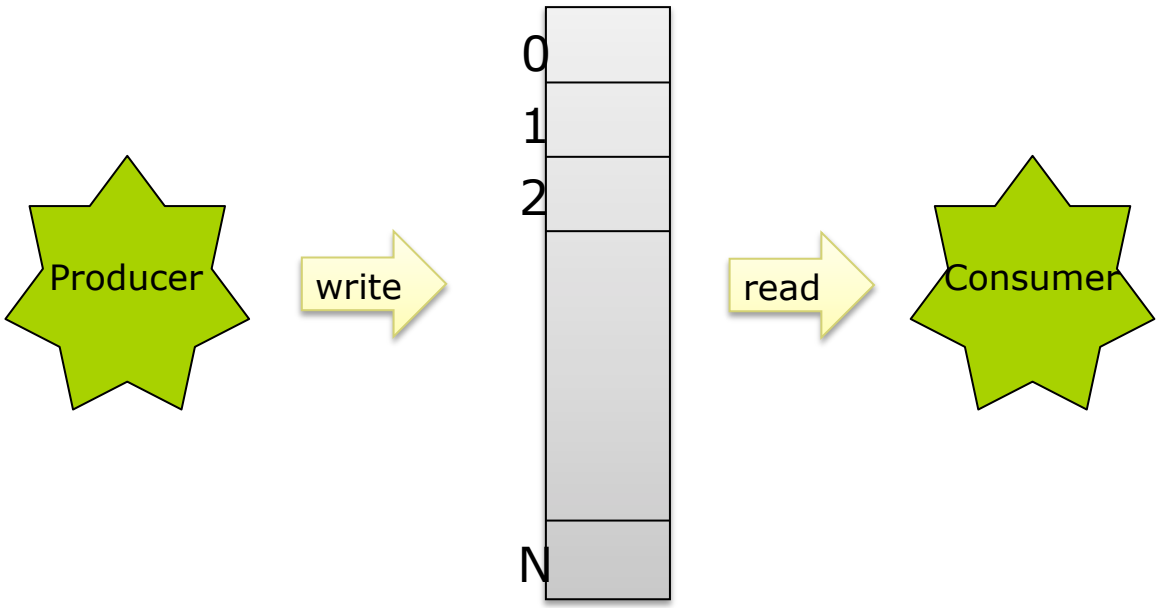
Chapter 7: Deadlocks



Bounded Buffer Problem

- Producer cannot write when full, but wait until not full
- Consumer cannot read when empty, but wait until not empty
- Producer and Consumer cannot write/read simultaneously



BBP v3

Producer

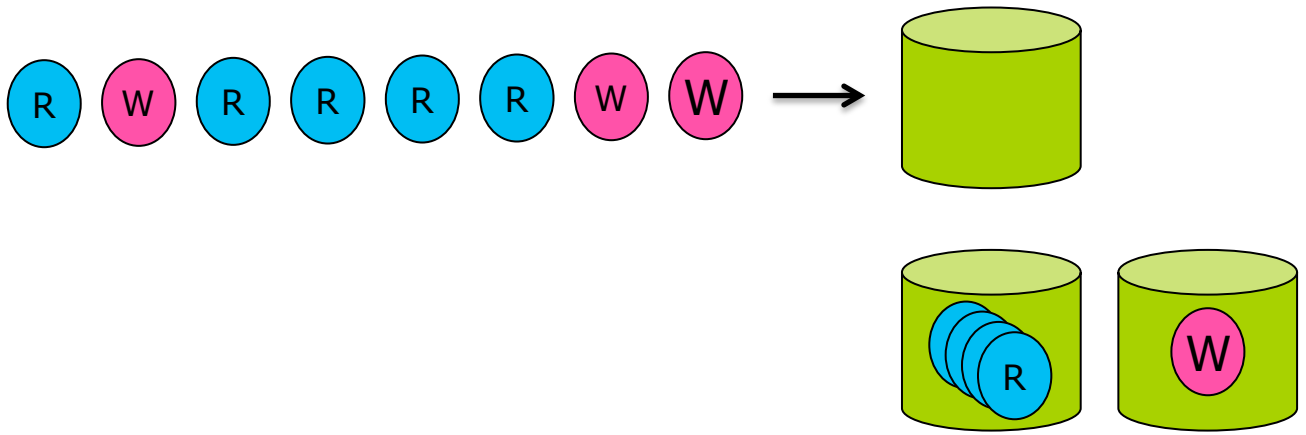
```
do {
    // produce an item
    wait (empty);
    wait (mutex);
    // add the item to the buffer
    signal (mutex);
    signal (full);
} while (TRUE);
```

Consumer

```
do {
    wait (full);
    wait (mutex);
    // remove an item from buffer
    signal (mutex);
    signal (empty);
    // consume the item
} while (TRUE);
```

First Reader-Writer problem

- Writer's problem
  - Cannot write if a writer is writing or a reader is reading
- Reader's problem
  - Cannot read if a writer is writing
  - First reader holds the lock
  - Last reader releases the lock



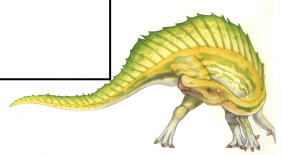
Readers-Writers Problem (Cont.)

- Reader process

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

        // Read data

        wait (mutex) ;
                readcount  - - ;
                if (readcount  == 0)
                        signal (wrt) ;
        signal (mutex) ;

    } while (TRUE);
```

- Writer process

```
do {
        wait (wrt) ;

        //    writing is performed

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



Cigarette Smokers Problem

- A cigarette requires
  - Tobacco
  - Smoking Paper
  - Match
- Three chain smokers sitting around the table, one has Tobacco, one has paper, and the other has a match
- One non-smoking arbiter do
  - When table is empty, pick two smokers at random, put their ingredients on the table, and let the third smoker to know ready
  - The third smoker makes a cigarette, and smoke the cigarette

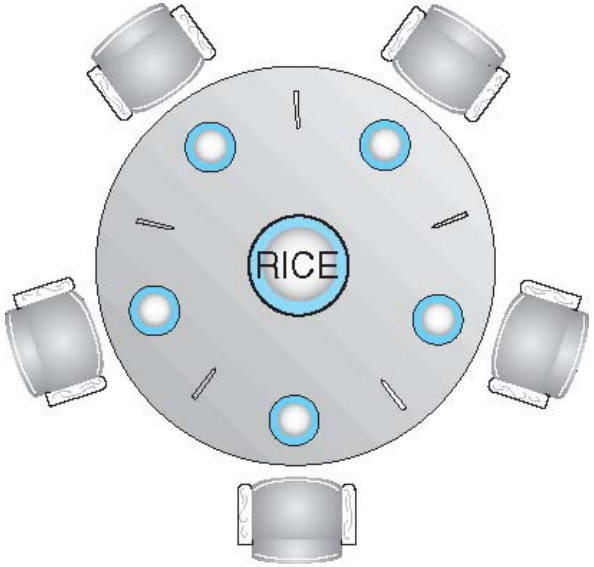
Cigarette Smokers Problem

- Arbiter
  - wait until table is empty
  - pick two smokers at random
  - notify the third smoker
- Smoker
  - wait until notified
  - make a cigarette
  - smoke cigarette

Cigarette Smokers Problem

- Arbiter
  - wait(T)
  - pick two smokers at random
  - k = third smoker
  - signal(A[k])
- Smoker-i
  - wait(A[i])
  - make a cigarette
  - signal(T)
  - smoke cigarette

Dining-Philosophers Problem



- Philosophers spend their lives 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
- 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);
```

- What is the problem with this algorithm?

Solutions

- At least one person is not hungry
- Pick if all the two chopsticks are free
- Asymmetric solution

The Deadlock Problem

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set

- Example
  - System has 2 disk drives
  - P_1 and P_2 each hold one disk drive and each needs another one

System Model

- Resource types R_1, R_2, . . ., R_m

  CPU cycles, memory space, I/O devices

- Each resource type R_i has W_i instances.

- Each process utilizes a resource as follows:
  - **request**
  - **use**
  - **release**

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task
- **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

Resource-Allocation Graph

A set of vertices V and a set of edges E.

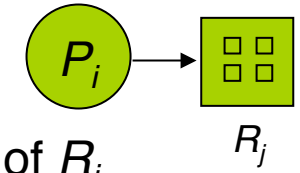
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system
- **request edge** – directed edge $P_i \rightarrow R_j$
- **assignment edge** – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph (Cont.)

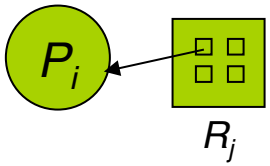
- Process

- Resource Type with 4 instances

- P_i requests instance of R_j



- P_i is holding an instance of R_j



Example of a Resource Allocation Graph