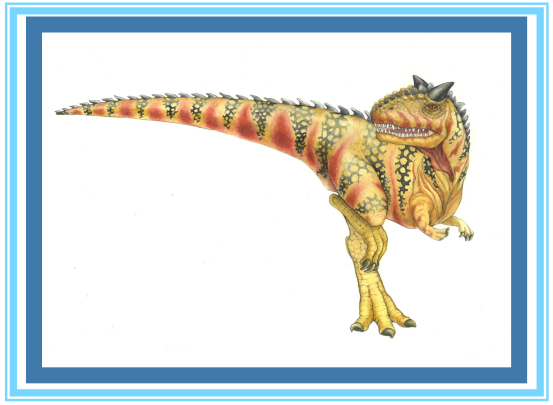
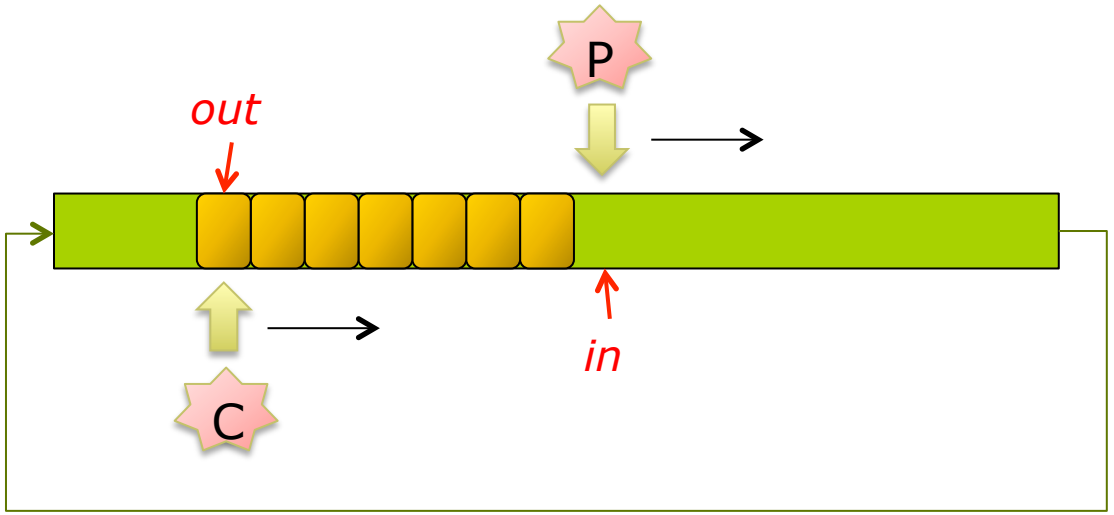
Chapter 6: Process Synchronization



Concurrency Problem

- Concurrent access to shared data may result in data inconsistency
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes
- Think about shared memory problem

Shared Buffer by Circular Array




```
#define BS 100
typedef struct {…} item;

item buf[BS]
int in = 0
int out = 0
```

* Buffer is empty if
  in == out
* Buffer is full if
  (in+1)%BS == out
* Maximum items count
  BS-1

Bounded-Buffer – Producer

```
while (true) {
   /* Produce an item */
      while ((in + 1) % BUFFER_SIZE == out)
       ;   /* do nothing -- no free buffers */
     buffer[in] = item;
     in = (in + 1) % BUFFER SIZE;
   }
```

Bounded Buffer – Consumer

```
while (true) {
        while (in == out)
                ; // do nothing --
nothing to consume

     // remove an item from the buffer
     item = buffer[out];
     out = (out + 1) % BUFFER SIZE;

}
```

Fix

- We can use only up to BUFFER_SIZE-1 spaces
  - empty if in == out
  - full if (in + 1)%BUFFER_SIZE == out
- How can we use all the space of the buffer?

- Use a *counter* variable to indicate the number of data
  - empty if counter == 0
  - full if counter == BUFFER_SIZE
  - Producer counter++ after writing a data
  - Consumer counter– after reading a data

Producer

```
while (true) {

        /*  produce an item and put in nextProduced  */
         while (counter == BUFFER_SIZE)
                  ; // do nothing
              buffer [in] = nextProduced;
              in = (in + 1) % BUFFER_SIZE;
              counter++;
}
```

Consumer

```
while (true)  {
        while (counter == 0)
            ; // do nothing
        nextConsumed =  buffer[out];
        out = (out + 1) % BUFFER_SIZE;
        counter--;

        /*  consume the item in nextConsumed  */
}
```

Race Condition

- counter++ could be implemented as

```
register1 = counter
register1 = register1 + 1
counter = register1
```

- counter-- could be implemented as

```
register2 = counter
register2 = register2 - 1
count = register2
```

- Consider this execution interleaving with "count = 5" initially:

S0: producer execute register1 = counter {register1 = 5}
S1: producer execute register1 = register1 + 1 {register1 = 6}
S2: consumer execute register2 = counter {register2 = 5}
S3: consumer execute register2 = register2 - 1 {register2 = 4}
S4: producer execute counter = register1 {count = 6 }
S5: consumer execute counter = register2 {count = 4}