Operating Systems

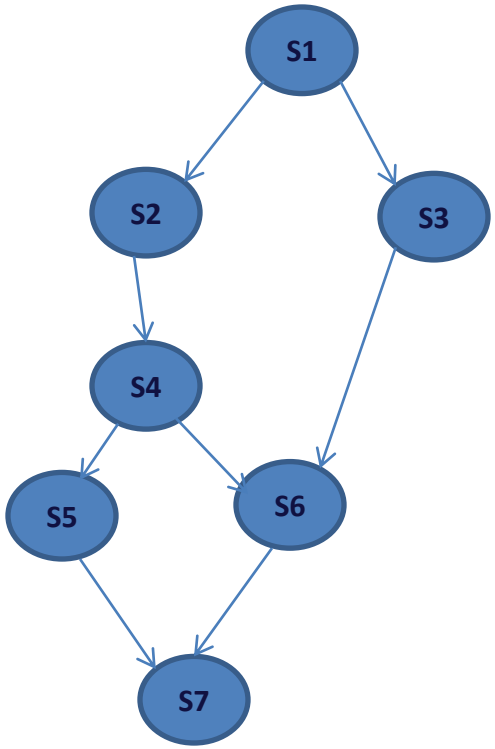
BITS Pilani
K K Birla Goa Campus

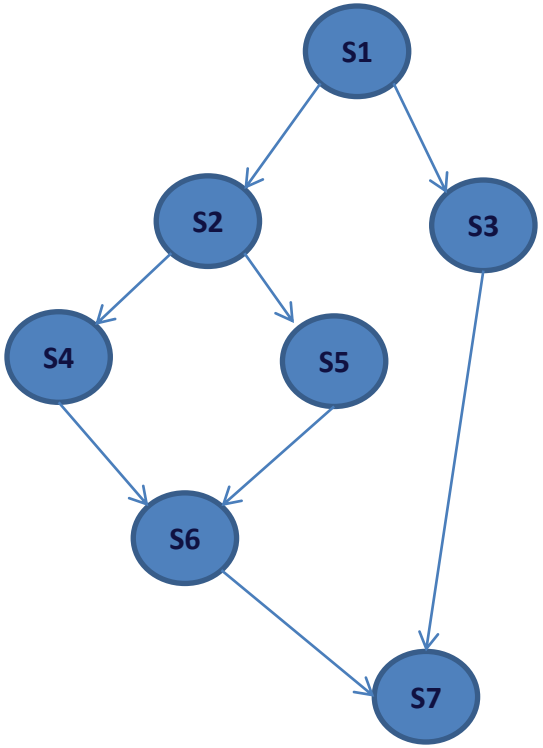
Dr. Lucy J. Gudino
Dept. of CS and IS

Process Synchronization

Last class

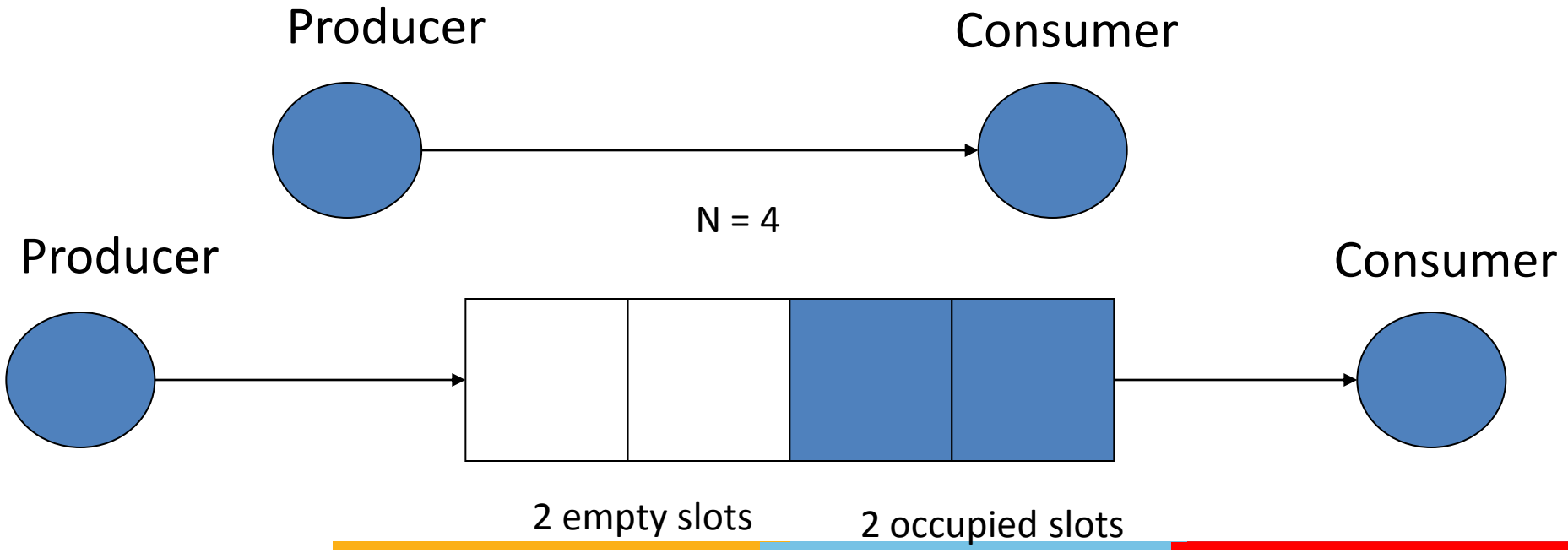
- Concurrent Processing
- fork and join construct
- cobegin and coend





contd…

- Concurrent access to shared data may result in data inconsistency
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes
- Producer / Consumer Problem



Producer / Consumer Problem…

- Two types of buffers:
  - unbounded buffer:
    - No limit on the size of the buffer
    - always space for producer to store data items
    - if producer is slower consumer needs to wait
  - bounded buffer: 2 cases
    - if the producer is faster than consumer
      - some point in time buffer will be full, producer has to wait
    - if the consumer is faster than producer
      - some point in time buffer will be empty, consumer has to wait
- use shared memory

contd..

- Bounded buffer can be used to enable processes to share memory.
- Shared bounded buffer is implemented using circular array.
- Assume size of buffer is 5
- Two logical pointers “in” and “out”
- variable “in” is used by producer which points to the next free position in the buffer
- variable “out” is used by consumer which points to the first full position in the buffer.

Contd…


```
#define BUFFER_SIZE 5
int buffer [BUFFER_SIZE ];
int in = 0;
int out = 0
```

- when in = out, buffer is empty
- when ((in+1) % BUFFER_SIZE) == out , buffer is full when

The producer process

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

Consumer Process

```
while (true)  {
        int nextConsumed;
         while (in == out)
              ; // do nothing, but wait
              nextConsumed =  buffer[out];
               out = (out + 1) % BUFFER_SIZE;
              /*  consume the item in nextConsumed
   }
```

Producer

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

Consumer

```
while (true)  {
        int nextConsumed;
         while (in == out)
                ; // do nothing, but wait
                nextConsumed =  buffer[out];
                 out = (out + 1) % BUFFER_SIZE;
                /*  consume the item in nextConsumed
 }
```

Contd…

- Main flaw: one location has to be empty
- use counter to eliminate above problem
- Initially counter is set to zero
  - increment the counter whenever producer produces an item
  - decrement the counter whenever consumer consumes an item
  - When counter = 0 ➔ buffer is empty
  - When counter = BUFFER_SIZE ➔ buffer is full

The producer process


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

Consumer Process

```
while (true)  {
        int nextConsumed;
       while (count == 0)
            ; // do nothing
            nextConsumed =  buffer[out];
             out = (out + 1) % BUFFER_SIZE;
               count--;
               /*  consume the item in nextConsumed
   }
```

Producer - Consumer

The producer process

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

Consumer Process

```
while (true)  {
        int nextConsumed;
        while (count == 0)
                ; // do nothing
        nextConsumed =  buffer[out];
        out = (out + 1) % BUFFER_SIZE;
        count--;
        /*  consume the item in nextConsumed
}
```

Contd…

Implementation of counter

Producer Process

A : R1 = count

B: R1 = R1+1

C: count = R1

Consumer Process

D : R2 = count

E: R2 = R2-1

F: count = R2

A

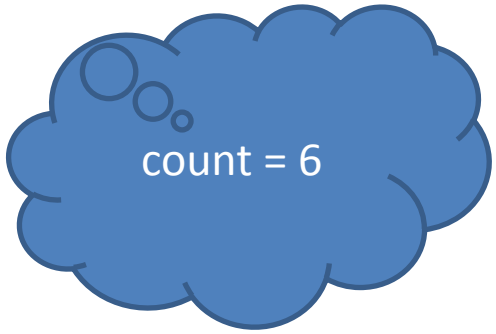
B

D

E

C

F



A : R1 = 6

B : R1 = 7

D : R2 = 6

E : R2 = 5

C : count = 7

F : count = 5

Race Condition

- Several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called a **race condition.**