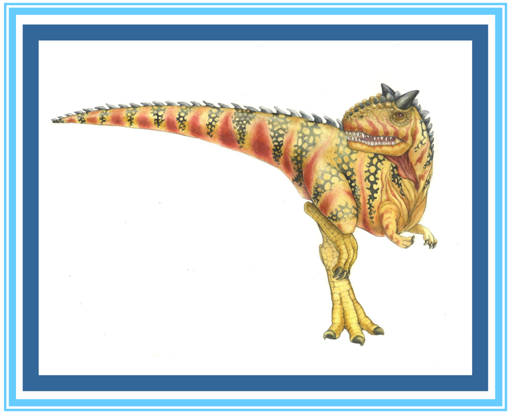
Chapter 5: Process Synchronization Lecture # 14



Previous Lecture

- Peterson’s Solution
- Synchronization Hardware
- Mutex Locks

Semaphore

- Synchronization tool that provides more sophisticated ways (than Mutex locks) for process to synchronize their activities.
- Semaphore ***S*** – integer variable
- Can only be accessed via two indivisible (atomic) operations
  - **`wait()`** and **`signal()`**
    - 4 Originally called **`P()`** and **`V()`**
- Definition of the **`wait() operation`**

```
wait(S) {
    while (S <= 0)
        ; // busy wait
    S--;
}
```

- Definition of the **`signal() operation`**

```
signal(S) {
    S++;
}
```

Semaphore

```
wait(S) {
    while (S <= 0)
       ; // busy wait
    S--;
}
```

```
signal(S) {
    S++;
}
```

```
do {
Entry section  wait(S)
//critical section
Exit section Signal (S)
//remainder section
 } while (true);
```

S =1

Classical Problems of Synchronization

- Classical problems used to test newly-proposed synchronization schemes
  - Bounded-Buffer Problem
  - Readers and Writers Problem
  - Dining-Philosophers Problem

Classical Problems of Synchronization

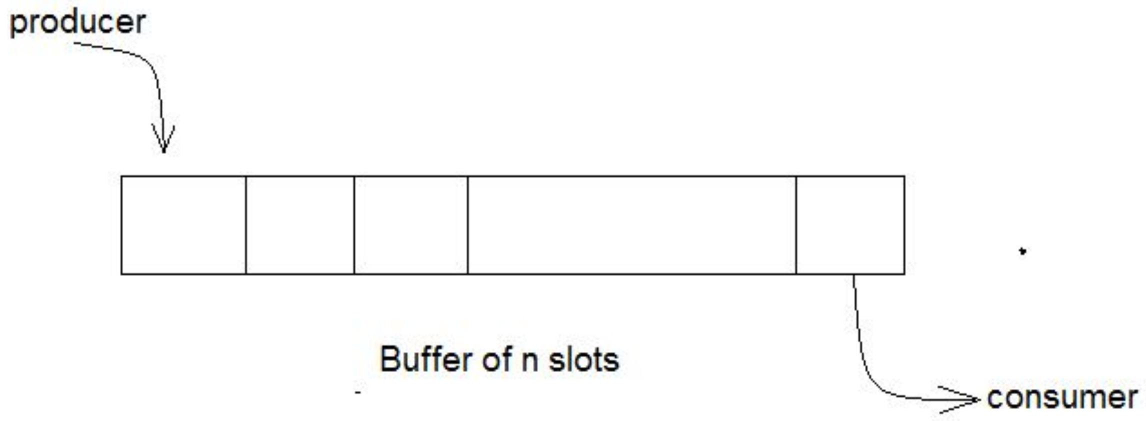
- Classical problems used to test newly-proposed synchronization schemes
  - **Bounded-Buffer Problem**
  - Readers and Writers Problem
  - Dining-Philosophers Problem

Bounded-Buffer Problem

- ***n*** buffers, each can hold one item
- Semaphore **`S`** initialized to the value 1
- Semaphore **`full`** initialized to the value 0
- Semaphore **`empty`** initialized to the value n

Bounded Buffer Problem

- ❑ There is a buffer of n slots and each slot can store one unit of data.
- ❑ There are two processes running, namely,
  - ❑ **producer** and **consumer**, which are operating on the buffer.



- ❑ A producer tries to insert data into an empty slot of the buffer. A consumer tries to remove data from a filled slot in the buffer. As you might have guessed by now, those two processes won't produce the expected output if they are being executed concurrently.
- ❑ There needs to be a way to make the producer and consumer work in an independent manner

Bounded Buffer Problem (Cont.)

- The structure of the producer process

```
void producer
{
while(true)
{
Produce(); /* produce an item in next_produced */
wait(E);
wait(S);
append();
  ...
                /* add next produced to the buffer */
                ...
signal(S);
signal(F);
}
}
```

1	2	3	4	5
a	b	C	D	e

S	E	F
1	0	5

```
wait(S) {
    while (S <= 0)
       ; // busy wait
    S--;
}

signal(S) {
    S++;
}
```

Bounded Buffer Problem (Cont.)

- The structure of the consumer process

```
void consumer
{
while(true)
{
wait(F);
wait(S);
take();
 ...
          /* consume the item in next consumed */
             ...
signal(S);
signal(E);
}
}
```

1	2	3	4	5

S	E	F
1	5	0

```
wait(S) {
     while (S <= 0)
         ; // busy wait
     S--;
}

signal(S) {
     S++;
}
```