OPERATING SYSTEMS

Classical Problems of Synchronization

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

Course Syllabus - Unit 2

12 Hours

Unit 2: Threads & Concurrency

Introduction to Threads, types of threads, Multicore Programming, Multithreading Models, Thread creation, Thread Scheduling, PThreads and Windows Threads, Mutual Exclusion and Synchronization: software approaches, principles of concurrency, hardware support, Mutex Locks, Semaphores. Classic problems of Synchronization: Bounded-Buffer Problem, Readers -Writers problem, Dining Philosophers Problem concepts. Synchronization Examples - Synchronisation mechanisms provided by Linux/Windows/Pthreads. Deadlocks: principles of deadlock, tools for detection and Prevention.

OPERATING SYSTEMS
Course Outline

13	Introduction to Threads, types of threads, Multicore Programming, Multithreading Models	4.1 - 4.3	42.8
14	Thread creation, Thread Scheduling	5.4	
15	Pthreads and Windows Threads	4.4	
16	Mutual Exclusion and Synchronization: software approaches,	6.1-6.2	
17	principles of concurrency, hardware support	6.3-6.4	
18	Mutex Locks, Semaphores	6.5, 6.6	
19	Classic problems of Synchronization: Bounded-Buffer Problem, Readers-Writers problem	6.7-6.8	
20	Dining-Philosophers Problem	6.8	
21	Synchronization Examples: Synchronisation mechanisms provided by Linux/Windows/Pthreads.	6.9	
22	Deadlocks: principles of deadlock, Deadlock Characterization	7.1-7.3	
23	Deadlock Prevention, Deadlock example	7.4-7.5	
24	Deadlock Detection, Algorithm	7.6	

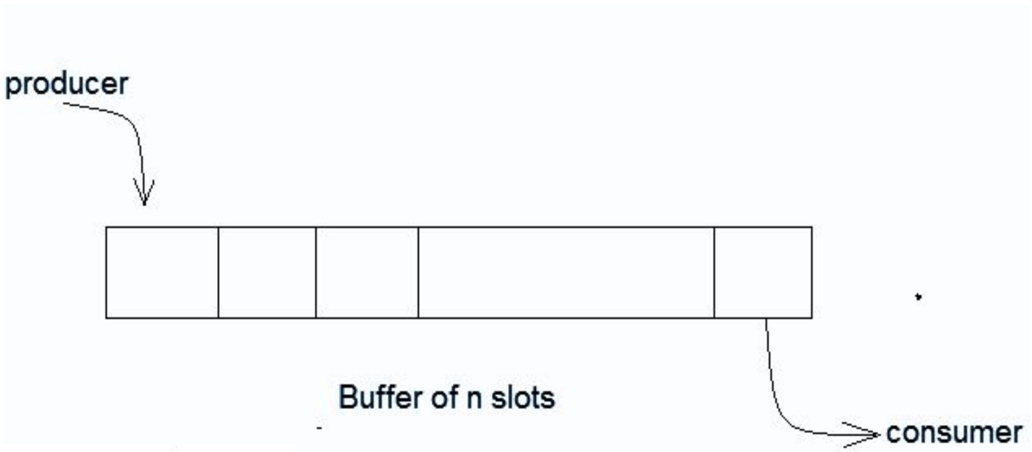
- Bounded Buffer Problem
- The Readers – Writers Problem

Classical Problems of Synchronization: Bounded Buffer Problem

- Bounded buffer problem, which is also called **producer consumer problem**, is one of the classical problems of synchronization.

- There is a buffer of n slots and each slot is capable of storing one unit of data.

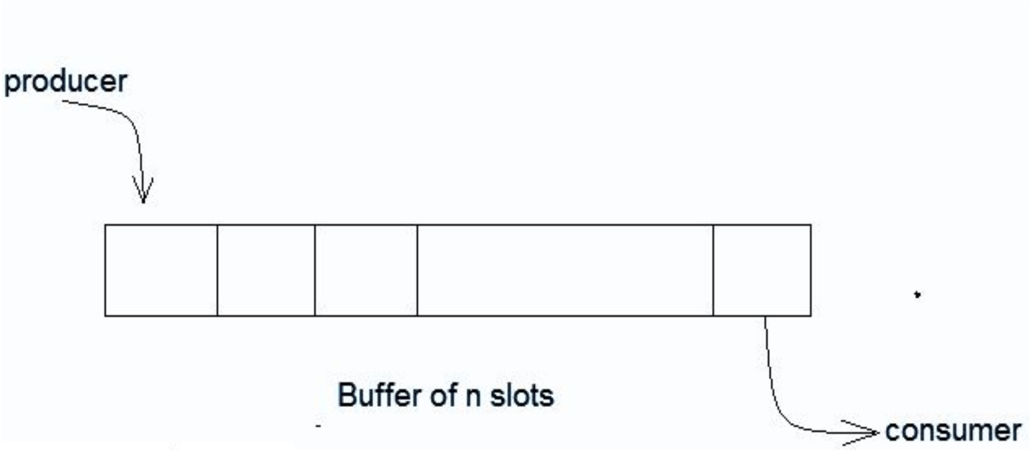
- There are two processes running, namely, **producer** and **consumer**, which are operating on the buffer.



Classical Problems of Synchronization: Bounded Buffer Problem

- 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 discussed earlier, 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



Classical Problems of Synchronization: Bounded Buffer Problem

- One solution of this problem is to use semaphores. The semaphores which will be used here are:
  - mutex - a **binary semaphore** which is used to acquire (wait) and release(signal) the lock.
  - empty a **counting semaphore** whose initial value is the number of slots in the buffer, since, initially all slots are empty.
  - full, a **counting semaphore** whose initial value is 0.

- At any instant, the current value of empty represents the number of empty slots in the buffer and full represents the number of occupied slots in the buffer.

Classical Problems of Synchronization: Bounded Buffer Problem

- The structure of the producer process

```
do {
      ...
      /* produce an item in next_produced */
      ...
   wait(empty);
   wait(mutex);
      ...
      /* add next produced to the buffer */
      ...
   signal(mutex);
   signal(full);
} while (true);
```

The structure of the consumer process

```
Do {
     wait(full);
     wait(mutex);
          ...
 /* remove an item from buffer to next_consumed */
          ...
     signal(mutex);
     signal(empty);
          ...
/* consume the item in next consumed */
          ...
  } while (true);
```

Classical Problems of Synchronization: Bounded Buffer Problem

- The structure of the producer process

```
do {
      ...
      /* produce an item in next_produced */
      ...
   wait(empty);
   wait(mutex);
      ...
      /* add next produced to the buffer */
      ...
   signal(mutex);
   signal(full);
} while (true);
```

The structure of the consumer process

```
Do {
    wait(full);
    wait(mutex);
        ...
 /* remove an item from buffer to next_consumed */
        ...
    signal(mutex);
    signal(empty);
        ...
/* consume the item in next consumed */
        ...
} while (true);
```

Classical Problems of Synchronization: Readers Writers Problem

- A data set is shared among a number of concurrent processes
    - Readers – only read the data set; they do not perform any updates
    - Writers – can both read and write

- Problem=> allow multiple readers to read at the same time

- Only one single writer can access the shared data at the same time

Classical Problems of Synchronization: Readers Writers Problem

- Several variations of how readers and writers are considered => all involve some form of priorities

- Shared Data
  - Data set
  - Semaphore rw_mutex initialized to 1
  - Semaphore mutex initialized to 1
  - Integer read_count initialized to 0

Classical Problems of Synchronization: Readers Writers Problem

- The structure of a reader process

```
do {
        wait(mutex);
        read_count++;
        if (read_count == 1)
         wait(rw_mutex);
      signal(mutex);
         ...
        /* reading is performed */
         ...
      wait(mutex);
        read count--;
        if (read_count == 0)
      signal(rw_mutex);
      signal(mutex);
} while (true);
```

- The structure of a writer process

```
do {
      wait(rw_mutex);
         ...
      /* writing is performed */
         ...
     signal(rw_mutex);
} while (true);
```

Classical Problems of Synchronization: Readers Writers Problem

- The structure of a reader process

```
do {
        wait(mutex);
        read_count++;
        if (read_count == 1)
         wait(rw_mutex);
      signal(mutex);
         ...
        /* reading is performed */
         ...
      wait(mutex);
        read_count--;
        if (read_count == 0)
      signal(rw_mutex);
      signal(mutex);
} while (true);
```

- The structure of a writer process

```
do {
      wait(rw_mutex);
         ...
      /* writing is performed */
         ...
     signal(rw_mutex);
} while (true);
```

Classical Problems of Synchronization: Readers Writers Problem

- First variation => no reader kept waiting unless writer has permission to use shared object
- Second variation => once writer is ready, it performs the write ASAP
- Both may have starvation leading to even more variations
- Problem is solved on some systems by kernel providing reader-writer locks

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

nitin.pujari@pes.edu

For Course Deliverables by the Anchor Faculty click on www.pesuacademy.com