Amirkabir University of Technology
(Tehran Polytechnic)

Operating Systems

Processes-Part4

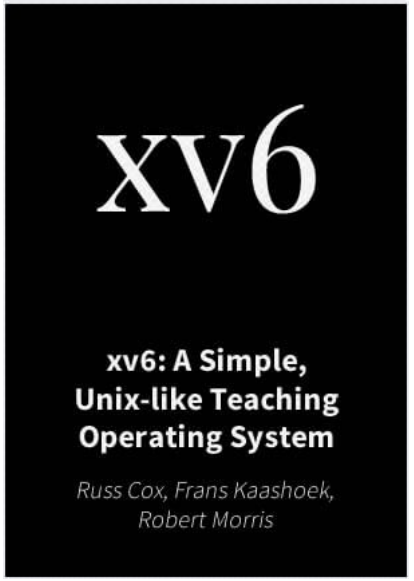
Seyyed Ahmad Javadi

sajavadi@aut.ac.ir

Fall 2021

Course logistics

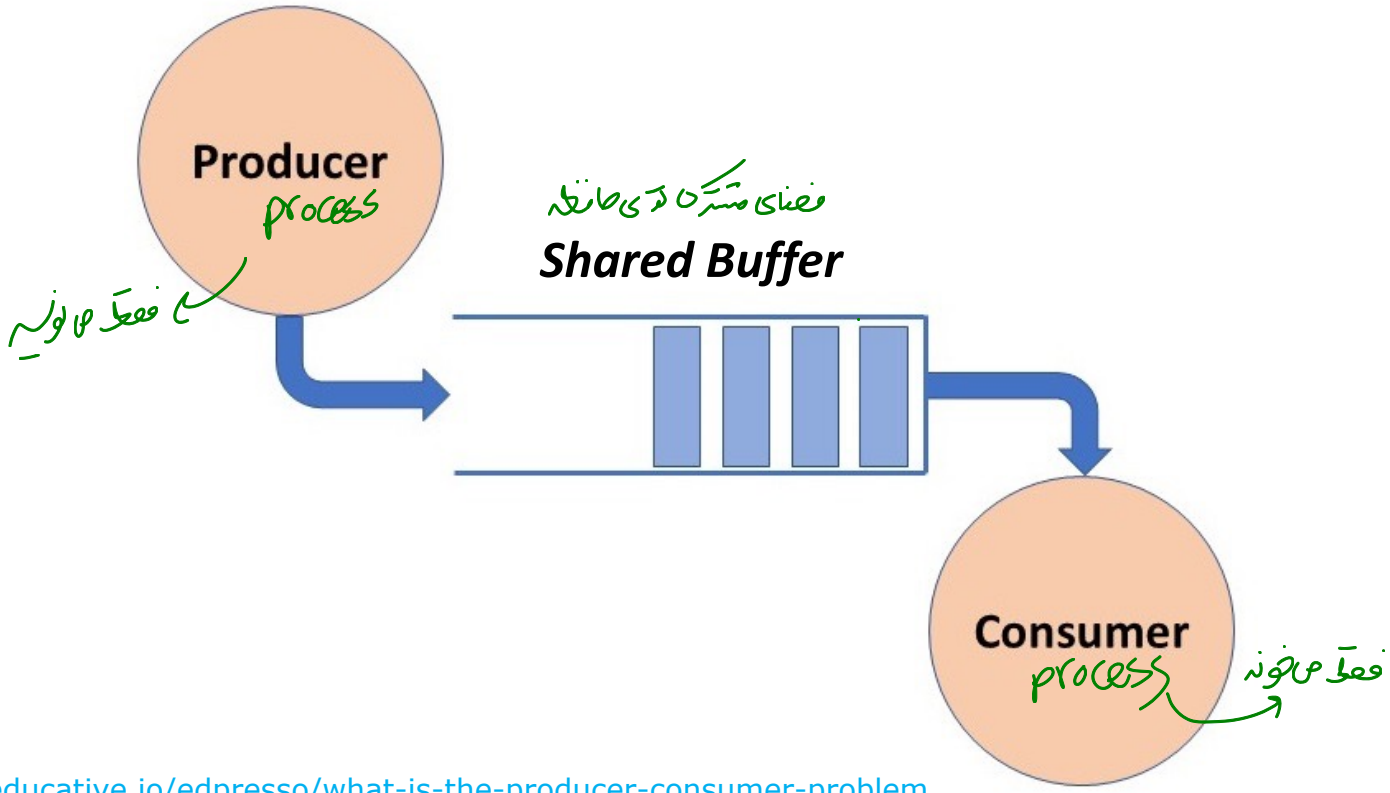
- You have **only few days** to complete
  - Phase 1 of the project
  - Your first homework



- We will have an extra session on fork
  - The session is handled by TAs
  - It covers both theoretical aspects and how use fork in practice
  - What time is best for you?

Producer-Consumer Problem

- Paradigm for cooperating processes:
  - **Producer** process produces information that is consumed by a **consumer** process.



https://www.educative.io/edpresso/what-is-the-producer-consumer-problem

Producer-Consumer Problem-Variations

1. **Unbounded-buffer** places no practical limit on the size of the buffer:
   - Producer never waits
   - Consumer waits if there is no buffer to consumer
2. **Bounded-buffer** assumes that there is a fixed buffer size
   - Producer must wait if all buffers are full
   - Consumer waits if there is no buffer to consume

IPC – Shared Memory

inter process communication

- An area of memory shared among the processes that wish to communicate.

  OS فضای مشترک رو فراهم می‌کنه اما اینکه چی نوشته بشه، چه زمانی نوشته بشه، کجا نوشته بشه دست اون دوتا process یه!

- The communication is ***under the control of the users*** processes ***not the operating system***.

  اگر ارتباط process ها به صورت synchron بشه، داده در معرض خطر نیست اما اگر نباشه، ممکن است داده به خطر بیفتد (از بین برود یا مثلاً هر دو نوشتن یک داده‌ی مشترک رو تغییر بدهند!)

- Major issues is to provide mechanism that will allow the user processes ***to synchronize their actions*** when they access shared memory.

  اگر براساس یک پروتکل مشترک عمل کنند، داده به درستی update میشه از هر دو طرف!

- Synchronization is discussed in great details in Chapters 6 & 7.

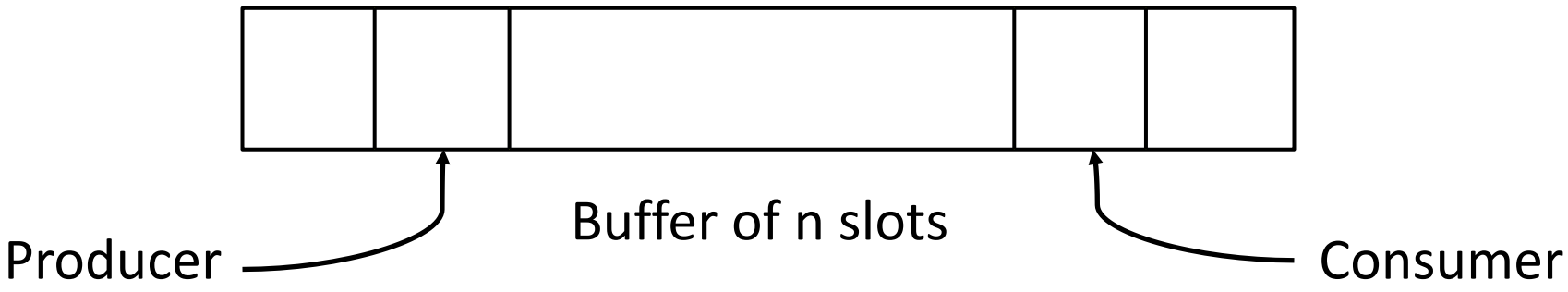
Bounded-Buffer – Shared-Memory Solution

- Shared data

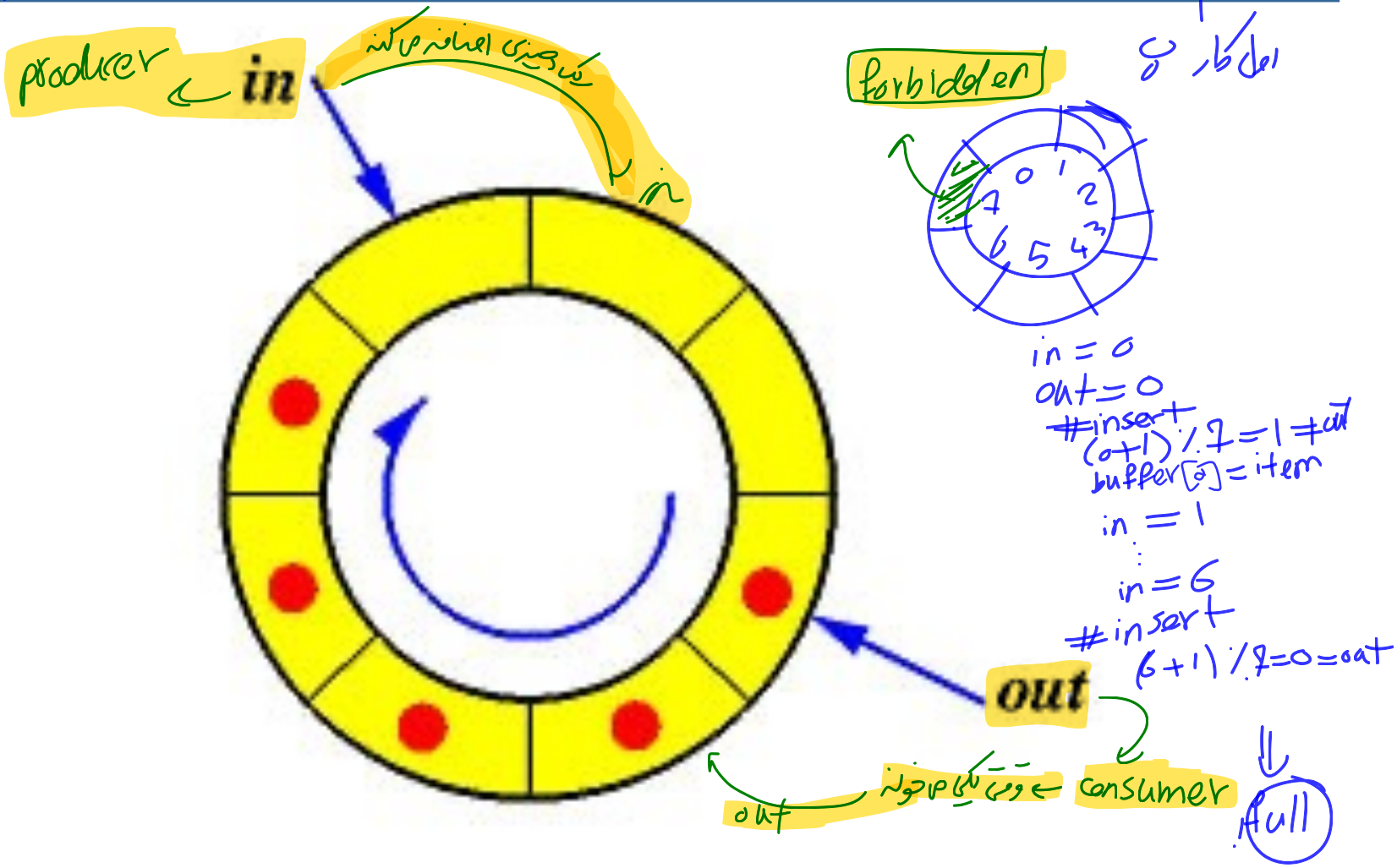
```
#define BUFFER_SIZE 10
typedef struct {
  . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

به اشتراک گذاشته می‌شود

- Solution is correct but can only use **BUFFER_SIZE - 1** elements.



Circular Bounded-Buffer

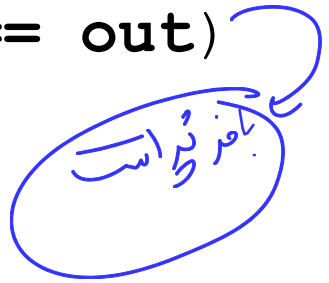


Source: https://pages.mtu.edu/~shene/NSF-3/e-Book/SEMA/TM-example-buffer.html

Producer Process – Shared Memory

```
item next_produced;

while (true) {
   /* produce an item in next produced */
   while (((in + 1) % BUFFER_SIZE) == out)
      ; /* do nothing */
   buffer[in] = next_produced;
   in = (in + 1) % BUFFER_SIZE;
}
```



Consumer Process – Shared Memory

```
item next_consumed;

while (true) {
        while (in == out)
                ; /* do nothing */
        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;

        /* consume the item in next consumed */
}
```

What about Filling all the Buffers?

- Suppose that we wanted to provide a solution to the consumer-producer problem that **fills all the buffers.**

چون قبلی این مثال از یک خونه می‌گذشتم استفاده کنیم!

- How can we do it?

می‌خواهیم counter نگهداریم

counter++ ← producer

counter-- ← consumer

What about Filling all the Buffers? (ont.)

- We can do so by having ***an integer counter*** that keeps track of the number of full buffers.
- Initially, counter is set to 0.
- The integer counter is incremented by the producer after it produces a new buffer.
- The integer counter is decremented by the consumer after it consumes a buffer.

Producer

```
while (true) {
        /* produce an item in next produced */

        while (counter == BUFFER_SIZE)
            ; /* do nothing */
        buffer[in] = next_produced;
        in = (in + 1) % BUFFER_SIZE;
        counter++;
}
```

Consumer

```
while (true) {
        while (counter == 0)
            ; /* do nothing */
        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
            counter--;
        /* consume the item in next
consumed */
}
```

Race Condition

- **counter++** could be implemented as

```
register1 = counter (load)
register1 = register1 + 1
counter = register1 (store)
```

- **counter--** could be implemented as

```
register2 = counter (load)
register2 = register2 - 1
counter = register2 (store)
```

Race Condition (cont.)

- 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** {counter = 6 }

S5: consumer execute **counter = register2** {counter = 4}

Race Condition (cont.)

Question – why was there no race condition in the first solution (where at most N – 1) buffers can be filled?

More in Chapter 6.

چون در اونجا produce از یک متغیر in استفاده میکرد و consumer از یک متغیر out استفاده میکرد. مشکلی نبود! چون جدا بودند!