CE 440 Introduction to Operating System

Lecture 8: Synchronization Exercises
Fall 2025

Prof. Yigong Hu

BOSTON UNIVERSITY

Slides courtesy of Manuel Egele, Ryan Huang and Baris Kasikci

Using Lock and Semaphores

We've looked at a simple example for using synchronization

- Mutual exclusion while accessing a bank account

Now let's use locks and semaphores to a more interesting example

- Bounded Buffers

Producer-Consumer With a Bounded Buffer

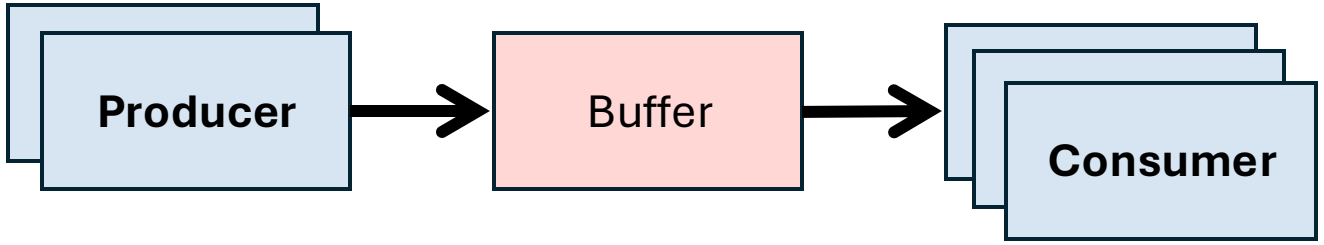
Problem: a set of buffers shared by producer and consumer threads

- Producer inserts resources into the buffer set
  - Output, disk blocks, memory pages, processes, etc.
- Consumer removes resources from the buffer set
  - Whatever is generated by the producer

Producer and consumer execute at different rates

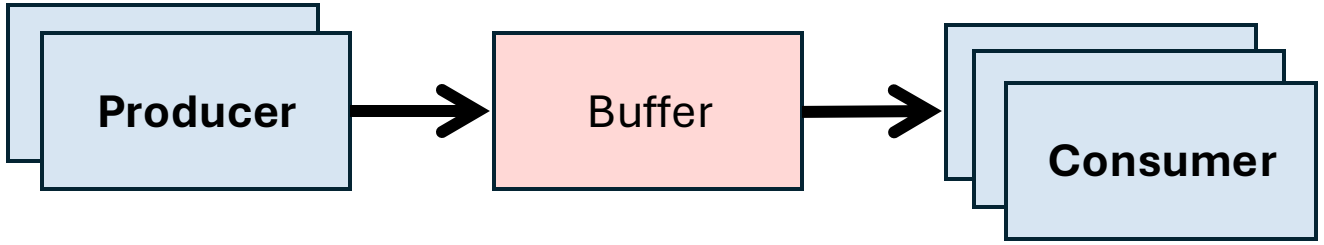
- No serialization of one behind the other
- Tasks are independent (easier to think about)
- The buffer set allows each to run without explicit handoff

Exercise 1.1: What Is the Safety Requirement?



https://app.sli.do/event/b8Co2oLQMSCHeYLA3DV8fr

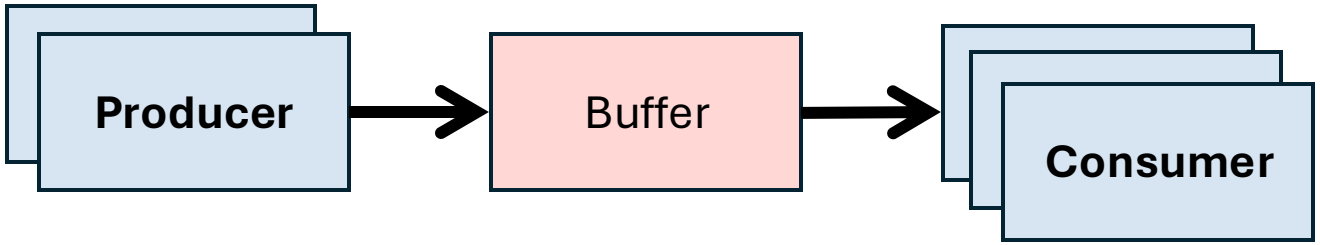
Exercise 1.2: What Is the Safety Requirement?



Follow up question:

- If nc is number consumed, np number produced, and N the size of the buffer, then:

Exercise 1.3: Buffer Data Structure



In the producer–consumer problem, the buffer pool must keep track of three essential pieces of information (excluding synchronization variables like semaphores or locks).

- Write down the three variables that the buffer pool structure needs in order to manage items produced and consumed. Use proper variable names and types.

```
typedef struct buf {

} buf_t;
```

Exercise 2.1: Using Lock

We now want to implement a bounded buffer that supports a **producer** thread adding items and a **consumer** thread removing items.

```
Mutex buf_lock = 0

void Producer (item) {
   …
}

void Consumer (item) {
   …
}
```

https://app.sli.do/event/j2U3ksJWRBuvu1PCedBZbS

Exercise 3.1: Semaphores

To implement a bounded buffer that supports a **producer** thread adding items and a **consumer** thread removing items, how many semaphores should we use?

https://app.sli.do/event/uoXX6nC5AJ9XEdEuFuaFtW

Exercise 3.2: Using Semaphore

Implement a bounded buffer that supports a **producer** thread adding items and a **consumer** thread removing items, how many semaphores should we use?

```
Semaphore mutex(1); // mutual exclusion to shared set of buffers
Semaphore empty(N); // count of empty buffers (all empty to start)
Semaphore full(0); // count of full buffers (none full to start)
```

```
void Producer (item) {
  …
}
```

```
void Consumer () {
  …
}
```

Exercise 4.1: Monitors for Bounded Buffer

To implement a bounded buffer that supports a **producer** thread adding items and a **consumer** thread removing items,

- How many methods do we need?
- What variables live inside monitor?
- Write down method names and variables. Use proper variable/method names.

```
Monitor Bounded_buffer {
    …
}
```

https://app.sli.do/event/9NCMHGFLUXrfJdHezfhEeK

Exercise 4.2: Monitor for Bounded Buffer

Write down the methods implementation

```
Monitor Bounded_buffer {
  Resource buffer[N];
  // Variables for indexing buffer
  // monitor invariant involves these vars
  Condition not_full; // space in buffer
  Condition not_empty; // value in buffer

  void put_resource (Resource R) {
    …
  }

  Resource get_resource() {
    …
  }

} // end monitor
```

More on Condition Variable and Monitor

Recap: Read-Write Problem

Readers/Writers Problem:

- An object is shared among several threads
- Some threads only read the object, others only write it
- We can allow multiple readers but only one writer
  - Let r be the number of readers, w be the number of writers
  - Safety: $(r \geq 0) \wedge (0 \leq w \leq 1) \wedge ((r > 0) \Rightarrow (w = 0))$

Recap: Using Semaphores for Readers/Writers

w_or_r provides mutex between readers and writers

- writer wait/signal, reader wait/signal when readcount goes from 0 to 1 or from 1 to 0

```
// exclusive writer or reader
Semaphore w_or_r(1);

// number of readers
int readcount = 0;

// mutual exclusion to readcount
Semaphore mutex(1);

writer() {
    wait(&w_or_r); // lock out others
    Write;
    signal(&w_or_r);// up for grabs
}
```

```
reader() {
    wait(&mutex); // lock readcount
    readcount += 1; // one more reader
    if (readcount == 1)
        wait(&w_or_r);// synch w/ writers
    signal(&mutex); // unlock readcount
    Read;
    wait(&mutex); // lock readcount
    readcount -= 1; // one less reader
    if (readcount == 0)
        signal(&w_or_r); // up for grabs
    signal(&mutex); // unlock readcount
}
```

Starvation on the Writers

R1 finishes, R2 blocks W1

w_or_r = 0, mutex =1, readcount = 2

```
// exclusive writer or reader
Semaphore w_or_r(1);

// number of readers
int readcount = 0;

// mutual exclusion to readcount
Semaphore mutex(1);

writer() {
    wait(&w_or_r); // lock out others
    Write;
    signal(&w_or_r);// up for grabs
}
```

```
reader() {
    wait(&mutex); // lock readcount
    readcount += 1; // one more reader
    if (readcount == 1)
        wait(&w_or_r);// synch w/ writers
    signal(&mutex); // unlock readcount
    Read;
    wait(&mutex); // lock readcount
    readcount -= 1; // one less reader
    if (readcount == 0)
        signal(&w_or_r); // up for grabs
    signal(&mutex); // unlock readcount
}
```

What if we have R3,R4,R5… coming now?

Exercise 5.1: Monitor Readers and Writers

Using Mesa Monitor semantics:

- How many methods do we need?
- What variables live inside monitor?
- Write down method names and variables. Use proper variable/method names.

```
Monitor RW {

}
```

https://app.sli.do/event/6hStgPuq44F3R3MgprtFn9

Exercise 5.2: Using Monitor

Using Mesa Monitor semantics:

- Write down the methods

```
Monitor RW {
  int nr = 0, nw = 0;
  Condition canRead, canWrite;

  void StartRead() {
    …
  }

  void EndRead () {
    …
  }
```

```
  void StartWrite() {
    …
  }

  void EndWrite () {
    …
  }

} // end monitor
```

Next Time…

Read Chapter 32