Lecture Notes in Computer Science:
Producer-Consumer Problem

Truc Huynh[1]

[1] Computer Science Department, Indiana University-Purdue University
Fort Wayne, IN, 46805,

huyntl02@students.pfw.edu

Abstract. The producer/ consumer problem is a popular synchronization problem (proposed by Edger W. Dijkstra) [1]. It is also called Bound-Buffer problems. The problem describes two processes,[4] the producer and the consumer. The Producer generates and puts data into the buffer while the Consumer is consuming the data one piece at a time.

Keywords: **Producer, Consumer, Bound-Buffer**

1 Introduction

Introduce the producer-consumer problem. Implement conditional variables to solve the producer-consumer problem.

Understand the producer-consumer problem. Understand different methods to solve the problem.

2 Normal Operation

In normal operation, the Consumer Process and Producer Process will not operate at the same time (on the same Buffer). Which means one will sleep (or do other tasks) while the other one operates.[4] The Buffer is a fixed size buffer with queue implementation (data structure).[4] The Producer Process will generate and push data in the Buffer.[4] The Consumer Process is consuming the data (i.e. removing it from the buffer), one piece at a time.

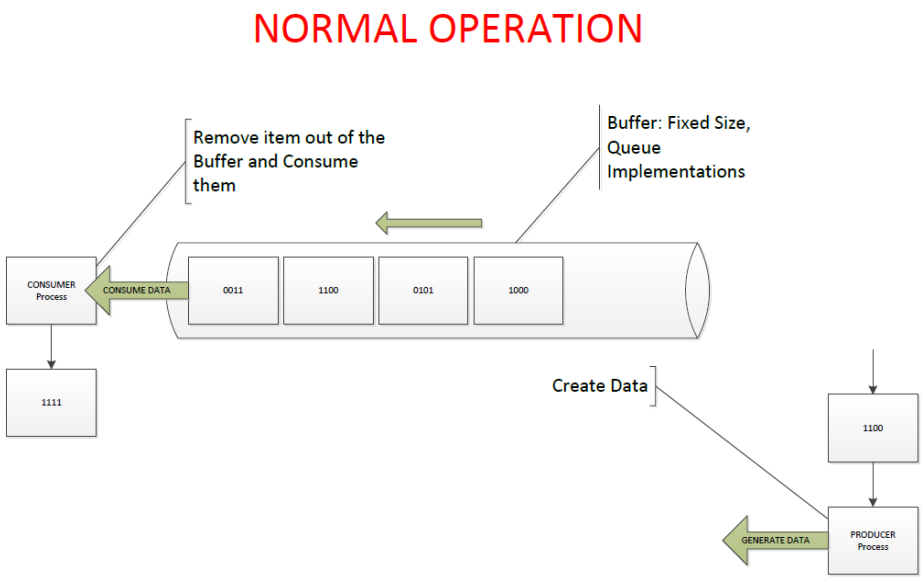


Diagram [A] designed by Truc Huynh

3 Synchronization problems:

Synchronization problems are popular multithreading issues which can cause deathlock. [4] Possible synchronization issues in the Producer-Consumer problem:

- A consumer attempts to consume a slot that is only half-filled by a producer.
- Producers do not block when the buffer is full.
- A Consumer consumes an empty slot in the buffer.
- Two producers write into the same slots
- Two consumers read the same slots
- There are many possible problems

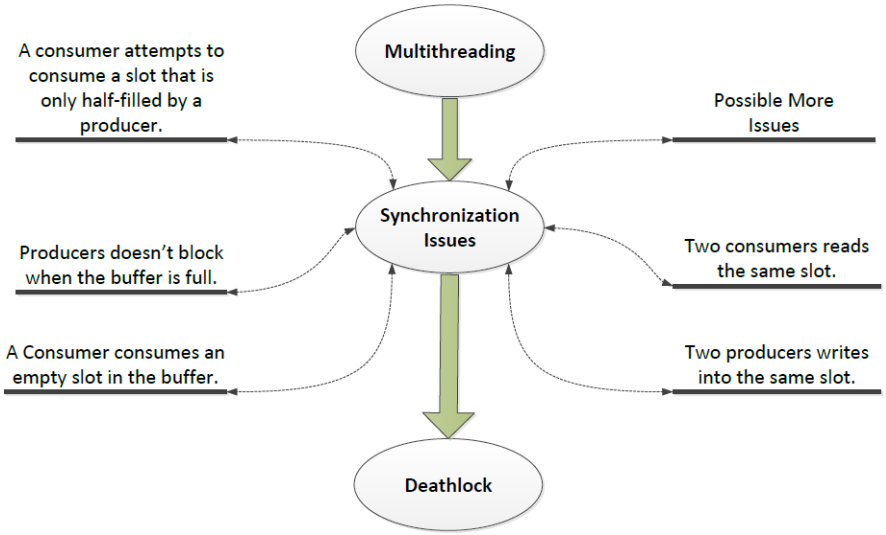


Diagram [B] designed by Truc Huynh

3.1 Deadlock:

This occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process [4].

In diagram [C], the Consumer Process enters the *waiting-process* (wait for the Producer) when the Buffer is empty. The Procedure Process generates data and fill-up the Buffer. After the Buffer is filled up, the Procedure Process enters the *waiting-process* (wait for the Consumer). The Resource in Buffer is allocated and reserve for the Consumer Process to consume. If the Consumer Process fails to wake up, they enter a deadlock

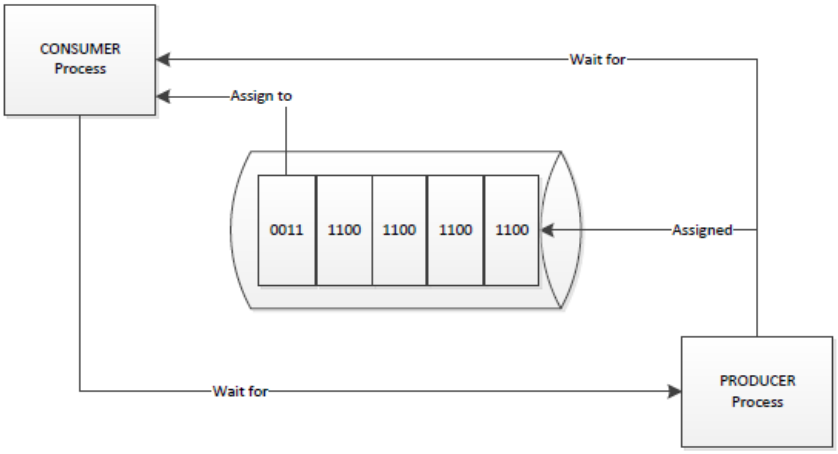


Diagram [C] designed by Truc Huynh

3.2 Consumer Processes and Producer Processes Operate at the same time:

In diagram [D], the Producer Process and the Consumer Process get access to the Buffer at the same time. By default, one process (either the Consumer or the Producer) must enter the *waiting-process* until the other finished its operation. The system will enter deadlock when both processes are in the *waiting-process.*

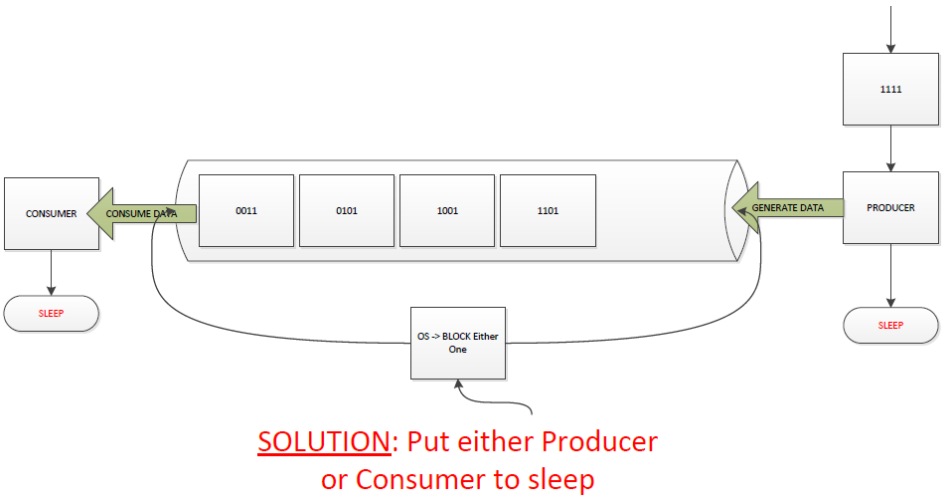


Diagram [D] designed by Truc Huynh

3.3 Consume Empty Buffer:

In normal operation, the Consumer Process can go to sleep if it finds the Buffer is empty [4]. If the Producer Process puts data in the Buffer, it will wake up the sleeping Consumer [4].

In diagram [E], the Producer fails to wake up the sleeping Consumer (in the *waiting-process*). When the Buffer is full, the Producer will enter the *waiting-process*. Since both processes are waiting to be awakened, they enter a deadlock.

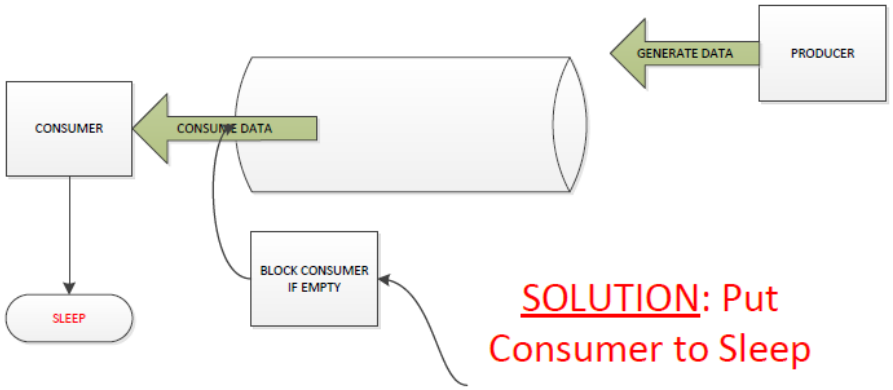


Diagram [E] designed by Truc Huynh

3.4 Add Data to Full Buffer:

In normal operation, the producer is either go to sleep or discard data if the buffer is full [4]. The next time the Consumer removes an item from the Buffer, it will notify the Producer (who starts to fill the Buffer again) [4]. This could result in a deadlock when both processes are waiting to be awakened.

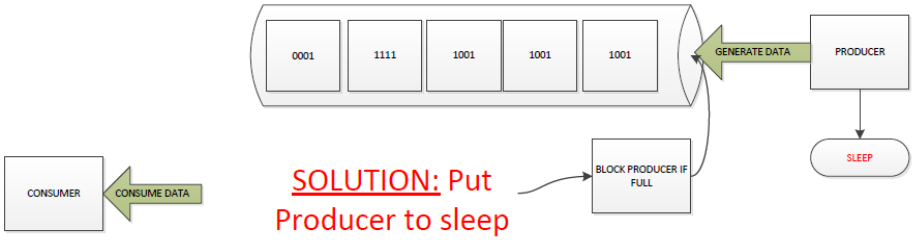


Diagram [F] designed by Truc Huynh

4 Solution Implementation:

Solutions to solve Bound-Buffer problems:

- Mutex Lock
- Conditional Variables
- Semaphores Algorithm
- Peterson Algorithm

JAVA IMPLEMENTATION

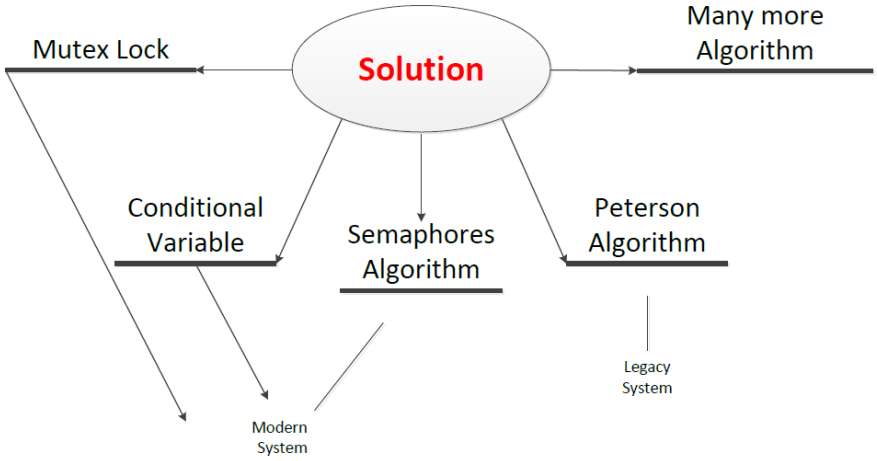
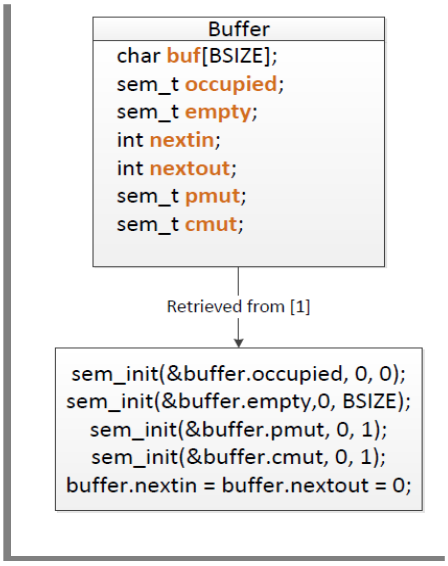


Diagram [G] designed by Truc Huynh

4.1 Producer and Consumer Problem with Semaphores:

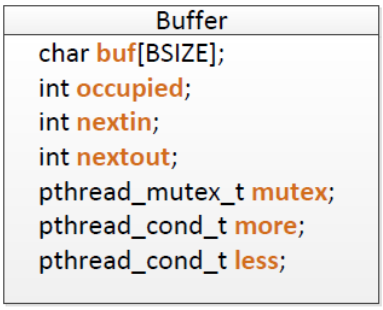
Semaphores were designed by E. W. Dijkstra in late 1960. To use a semaphore, the thread that wants access to the shared resource tries to acquire a permit [2].

- If the semaphore's count (sem_t) is greater than zero, then the thread acquires a permit, which causes the semaphore's count to be decremented.
- Otherwise, the thread will be blocked until a permit can be acquired.
- When the thread no longer needs access to the shared resource, it releases the permit, which causes the semaphore's count to be incremented.
- If there is another thread waiting for a permit, then that thread will acquire a permit at that time.
- Buffer's data-structures implementation [1]:



4.2 Producer and Consumer Problem with Conditional Variables:

- Buffer's data-structures implementation [1]:



- **Producer/Consumer Problem—the Producer:**

Example The Producer/Consumer Problem--the Producer

```
void producer(buffer_t *b, char item)
{
   pthread_mutex_lock(&b->mutex);
     while (b->occupied >= BSIZE)
      pthread_cond_wait(&b->less, &b->mutex);
   assert(b->occupied < BSIZE);
   b->buf[b->nextin++] = item;
   b->nextin %= BSIZE;
   b->occupied++;

   /* now: either b->occupied < BSIZE and b->nextin is the index
     of the next empty slot in the buffer, or
     b->occupied == BSIZE and b->nextin is the index of the
     next (occupied) slot that will be emptied by a consumer
     (such as b->nextin == b->nextout) */

   pthread_cond_signal(&b->more);
   pthread_mutex_unlock(&b->mutex);
}
```

- **Producer/Consumer Problem—the Consumer:**

Example The Producer/Consumer Problem--the Consumer

```
char consumer(buffer_t *b)
{
   char item;
   pthread_mutex_lock(&b->mutex);
   while(b->occupied <= 0)
      pthread_cond_wait(&b->more, &b->mutex);
   assert(b->occupied > 0);
   item = b->buf[b->nextout++];
   b->nextout %= BSIZE;
   b->occupied--;

   /* now: either b->occupied > 0 and b->nextout is the index
     of the next occupied slot in the buffer, or
     b->occupied == 0 and b->nextout is the index of the next
     (empty) slot that will be filled by a producer (such as
     b->nextout == b->nextin) */

   pthread_cond_signal(&b->less);
   pthread_mutex_unlock(&b->mutex);

   return(item);
}
```

References

Oracle(n.d.) *The Producer/Consumer Problem*. Retrieved from https://docs.oracle.com/cd/E19455-01/806-5257/sync-31/index.html **[1]**

Miglani., G.(2018) *Semaphore in Java.* Retrieved from https://www.geeksforgeeks.org /semaphore-in-java/ **[2]**

Shah., K.; Rithesh (2020) *Introduction of Deadlock in Operating System.* Retrieved from: https://www.geeksforgeeks.org/ introduction-of-deadlock-in-operating-system/ **[3]**

Yadav, G.,(2019) *Producer-Consumer solution using threads in Java.* Retrieved from https://www.geeksforgeeks.org/producer-consumer-solution-using-threads-java/ **[4]**