

Operating Systems and Concurrency Lecture 10&11: Concurrency

University of Nottingham, Ningbo China, 2024



- Concurrency using semaphores and mutexes
 - Mutex is a spinlock (busy waiting)
 - Semaphore puts the process to sleep
- Practical examples of how to use (code) semaphores
- Solve producer consumer problem using semaphore



Today class

- The Consumer-producer problem
 - Scenarios
 - Solutions

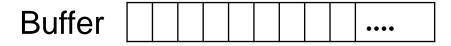


The Producer/Consumer Problem One Consumer, One Producer, Unbounded Buffer

- It is obvious that any manipulations of "item" will have to be synchronized.
- Race conditions still exist:
 - When the consumer has exhausted the buffer, should have gone to sleep, but the producer increments items before the consumer checks it.

```
void * consumer(void * p)
                                          void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                            while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items --:
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   sem_post(&sync); 0 => 1
                                             if(items == 1)
                                               sem_post(&delay_consumer);
   if(items = 0)
     sem_wait(&delay_consumer);
                                             sem_post(&sync):
```





Action	delay_cons=0	Syn=1	Item=0
C_blocked	-1	1	0

```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);0=>-1
 while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   sem_post(&sync);
                                             if(items == 1)
   if(items = 0)
                                               sem_post(&delay_consumer);
     sem_wait(&delay_consumer);
                                             sem_post(&sync):
```





Action	delay_cons=0	Syn=1	Item=0
C_blocked	-1	1	0
Enter_CS	-1	0	0

```
void * consumer(void * p)
                                          void * producer(void * p)
  sem wait(&delay consumer);
  while(1)
                                            while(1)
   sem wait(&sync);
                                             sem_wait(&sync); 1=>0
   items--:
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   sem_post(&sync);
                                             if(items = 1)
   if(items == 0)
                                               sem_post(&delay_consumer);
     sem_wait(&delay_consumer);
                                             sem_post(&sync);
```



Buffer	A									••••
--------	---	--	--	--	--	--	--	--	--	------

Action	delay_cons=0	Syn=1	Item=0
C_blocked	-1	1	0
Enter_CS	-1	0	0
	-1	0	1

```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
                                             sem wait(&sync);
   sem_wait(&sync);
                                             items++: 0 => 1
   items--:
   printf("%d\n", items);
                                             printf("%d\n", items);
   sem_post(&sync):
                                            if(items = 1)
   if(items = 0)
                                               sem post(&delay consumer);
     sem_wait(&delay_consumer);
                                             sem_post(&sync);
```

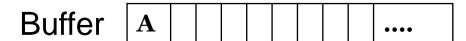


Buffer	A									••••	
--------	---	--	--	--	--	--	--	--	--	------	--

Action	delay_cons=0	Syn=1	Item=0
C_blocked	-1	1	0
Enter_CS	-1	0	0
	-1	0	1
	-1	0	1

```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items):
                                             printf("%d\n", items);
   sem_post(&sync);
                                             if(items == 1)
                                               sem post(&delay consumer);
   if(items = 0)
     sem_wait(&delay_consumer);
                                             sem_post(&sync):
```





Action	delay_cons=0	Syn=1	Item=0
C_blocked	-1	1	0
Enter_CS	-1	0	0
	-1	0	1
	-1	0	1
	-1	0	1

```
void * consumer(void * p)
                                          void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                            while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   sem_post(&sync):
                                             if(items == 1)
   if(items == 0)
                                               sem_post(&delay_consumer);
     sem_wait(&delay_consumer);
                                             sem_post(&sync);
```





```
void * consumer(void * p)
{
    sem_wait(&delay_consumer); (wakeup)
    while(1)
    {
        sem_wait(&sync);
        items--;
        printf("%d\n", items);
        sem_post(&sync);
        if(items == 0)
            sem_wait(&delay_consumer);
    }
}
```

```
void * producer(void * p)
{
  while(1)
{
    sem_wait(&sync);
    items++;
    printf("%d\n", items);
    if(items == 1)
        sem_post(&delay_consumer); -1=>0
    sem_post(&sync);
}
```

Action	delay_cons=0	Syn=1	Item=0
C_blocked	-1	1	0
Enter_CS	-1	0	0
	-1	0	1
	-1	0	1
	-1	0	1
Wakeup_C	0	0	1



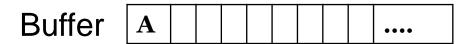
Buffer	A									••••	
--------	---	--	--	--	--	--	--	--	--	------	--

```
void * consumer(void * p)
{
   sem_wait(&delay_consumer);
   while(1)
   {
      sem_wait(&sync);
      items--;
      printf("%d\n", items);
      sem_post(&sync);
      if(items == 0)
           sem_wait(&delay_consumer);
   }
}
```

```
void * producer(void * p)
{
  while(1)
  {
    sem_wait(&sync);
    items++;
    printf("%d\n", items);
    if(items == 1)
        sem_post(&delay_consumer);
    sem_post(&sync); 0=>1
}
```

Action	delay_cons=0	Syn=1	Item=0
C_blocked	-1	1	0
Enter_CS	-1	0	0
	-1	0	1
	-1	0	1
	-1	0	1
Wakeup_C	0	0	1
Exit_CS	0	1	1





```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
 while(1)
                                            while(1)
   sem_wait(&sync); 1 => 0
                                             sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   sem_post(&sync):
                                             if(items == 1)
                                               sem_post(&delay_consumer);
   if(items = 0)
     sem_wait(&delay_consumer);
                                             sem post(&sync):
```

Action	delay_cons=0	Syn=1	Item=0
C_blocked	-1	1	0
Enter_CS	-1	0	0
	-1	0	1
	-1	0	1
	-1	0	1
Wakeup_C	0	0	1
Exit_CS	0	1	1
Enter_CS	0	0	1





```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                            while(1)
                                             sem_wait(&sync);
   sem_wait(&sync);
   items--: 1 => 0
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   sem_post(&sync);
                                             if(items == 1)
   if(items == 0)
                                               sem_post(&delay_consumer);
     sem_wait(&delay_consumer);
                                             sem_post(&sync);
```

Action	delay_cons=0	Syn=1	Item=0
C_blocked	-1	1	0
Enter_CS	-1	0	0
	-1	0	1
	-1	0	1
	-1	0	1
Wakeup_C	0	0	1
Exit_CS	0	1	1
Enter_CS	0	0	1
	0	0	0





```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   sem_post(&sync);
                                             if(items == 1)
   if(items = 0)
                                               sem_post(&delay_consumer);
     sem_wait(&delay_consumer);
                                             sem_post(&sync);
```

Action	delay_cons=0	Syn=1	Item=0
C_blocked	-1	1	0
Enter_CS	-1	0	0
	-1	0	1
	-1	0	1
	-1	0	1
Wakeup_C	0	0	1
Exit_CS	0	1	1
Enter_CS	0	0	1
	0	0	0
	0	0	0





```
void * consumer(void * p)
                                         void * producer(void * p)
  sem wait(&delay consumer);
  while(1)
                                           while(1)
   sem wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items):
                                             printf("%d\n", items);
   sem_post(&sync); 0 => 1
                                            if(items == 1)
   if(items = 0)
                                              sem post(&delay consumer);
     sem_wait(&delay_consumer);
                                             sem_post(&sync):
```

Action	delay_cons=0	Syn=1	Item=0
C_blocked	-1	1	0
Enter_CS	-1	0	0
	-1	0	1
	-1	0	1
	-1	0	1
Wakeup_C	0	0	1
Exit_CS	0	1	1
Enter_CS	0	0	1
	0	0	0
	0	0	0
Exit_CS	0	1	0





sem_wait(&delay_consumer);

Action	delay_cons=0	Syn=1	Item=0
C_blocked	-1	1	0
Enter_CS	-1	0	0
	-1	0	1
	-1	0	1
	-1	0	1
Wakeup_C	0	0	1
Exit_CS	0	1	1
Enter_CS	0	0	1
	0	0	0
	0	0	0
Exit_CS	0	1	0
Enter_CS	0	0	0

Figure: Single producer/consumer with unbounded buffer: Race condition (non-existing element => items=-1)

sem_post(&sync);



Buffer

A						••••
---	--	--	--	--	--	------

Action	delay_cons=0	Syn=1	Item=0
Enter_CS	0	0	0
	0	0	1

```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                              while(1)
                                             sem_wait(&sync);
   sem_wait(&sync);
                                             items++: 0 => 1
   items--:
   printf("%d\n", items);
                                             printf("%d\n", items);
   sem_post(&sync);
                                             if(items == 1)
   if(items = 0)
                                               sem_post(&delay_consumer);
     sem_wait(&delay_consumer);
                                             sem_post(&sync):
```



Buffer	A									••••	
--------	---	--	--	--	--	--	--	--	--	------	--

Action	delay_cons=0	Syn=1	Item=0
Enter_CS	0	0	0
	0	0	1
	0	0	1

```
void * consumer(void * p)
                                          void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                            while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   sem_post(&sync);
                                             if(items == 1)
   if(items == 0)
                                               sem_post(&delay_consumer);
     sem_wait(&delay_consumer);
                                             sem_post(&sync);
```

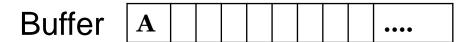


Buffer	A									••••
--------	---	--	--	--	--	--	--	--	--	------

Action	delay_cons=0	Syn=1	Item=0
Enter_CS	0	0	0
	0	0	1
	0	0	1
	0	0	1

```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                            while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items):
                                             if(items == 1)
   sem_post(&sync);
                                               sem_post(&delay_consumer);
   if(items = 0)
     sem wait(&delay consumer);
                                             sem_post(&sync);
```





Action	delay_cons=0	Syn=1	Item=0
Enter_CS	0	0	0
	0	0	1
	0	0	1
	0	0	1
	1	0	1

```
void * consumer(void * p)
{
   sem_wait(&delay_consumer);
   while(1)
   {
      sem_wait(&sync);
      items--;
      printf("%d\n", items);
      sem_post(&sync);
      if(items == 0)
           sem_wait(&delay_consumer);
   }
}
```

```
void * producer(void * p)
{
  while(1)
  {
    sem_wait(&sync);
    items++;
    printf("%d\n", items);
    if(items == 1)
        sem_post(&delay_consumer);0 => 1
    sem_post(&sync);
  }
}
```





```
void * producer(void * p)
void * consumer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                            while(1)
                                             sem_wait(&sync);
   sem_wait(&sync);
   items --:
                                             items++:
   printf("%d\n", items):
                                             printf("%d\n", items):
   sem post(&sync):
                                             if(items = 1)
   if(items = 0)
                                               sem_post(&delay_consumer);
     sem_wait(&delay_consumer);
                                             sem_post(&sync); 0 => 1
```

Action	delay_cons=0	Syn=1	Item=0
Enter_CS	0	0	0
	0	0	1
	0	0	1
	0	0	1
	1	0	1
Exit_CS	1	1	1





```
void * consumer(void * p)
                                          void * producer(void * p)
  sem_wait(&delay_consumer);
                                            while(1)
  while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   sem_post(&sync);
                                             if(items = 1)
   if(items == 0)
                                               sem_post(&delay_consumer);
     sem_wait(&delay_consumer);
                                             sem_post(&sync);
```

Action	delay_cons=0	Syn=1	Item=0
Enter_CS	0	0	0
	0	0	1
	0	0	1
	0	0	1
	1	0	1
Exit_CS	1	1	1
	1	1	1





```
void * consumer(void * p)
                                          void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                            while(1)
   sem_wait(&sync); 1 => 0
                                             sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   sem_post(&sync);
                                             if(items == 1)
   if(items == 0)
                                               sem_post(&delay_consumer);
     sem_wait(&delay_consumer);
                                             sem_post(&sync);
```

Action	delay_cons=0	Syn=1	Item=0
Enter_CS	0	0	0
	0	0	1
	0	0	1
	0	0	1
	1	0	1
Exit_CS	1	1	1
	1	1	1
Enter_CS	1	0	1

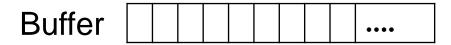




```
void * consumer(void * p)
                                         void * producer(void * p)
  sem wait(&delay consumer);
 while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items -: 1 => 0
                                             items++:
   printf("%d\n", items):
                                             printf("%d\n", items);
   sem_post(&sync);
                                             if(items = 1)
   if(items == 0)
                                               sem_post(&delay_consumer);
     sem_wait(&delay_consumer);
                                             sem_post(&sync);
```

Action	delay_cons=0	Syn=1	Item=0
Enter_CS	0	0	0
	0	0	1
	0	0	1
	0	0	1
	1	0	1
Exit_CS	1	1	1
	1	1	1
Enter_CS	1	0	1
	1	0	0





```
void * consumer(void * p)
                                          void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                            while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   sem_post(&sync);
                                             if(items == 1)
                                               sem_post(&delay_consumer);
   if(items == 0)
     sem_wait(&delay_consumer);
                                             sem_post(&sync);
```

Action	delay_cons=0	Syn=1	Item=0
Enter_CS	0	0	0
	0	0	1
	0	0	1
	0	0	1
	1	0	1
Exit_CS	1	1	1
	1	1	1
Enter_CS	1	0	1
	1	0	0
	1	0	0





```
void * consumer(void * p)
                                           void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                             while(1)
   sem wait(&sync);
                                               sem_wait(&sync);
   items--:
                                               items++:
   printf("%d\n", items);
                                               printf("%d\n", items);
   sem_post(\&sync); 0 \Rightarrow 1
                                              if(items == 1)
   if(items == 0)
                                                sem post(&delay consumer);
     sem_wait(&delay_consumer);
                                               sem_post(&sync);
```

Action	delay_cons=0	Syn=1	Item=0
Enter_CS	0	0	0
	0	0	1
	0	0	1
	0	0	1
	1	0	1
Exit_CS	1	1	1
	1	1	1
Enter_CS	1	0	1
	1	0	0
	1	0	0
Exit_CS	1	1	0

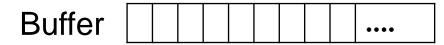




```
void * consumer(void * p)
                                          void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                            while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   sem_post(&sync);
                                             if(items == 1)
   if(items == 0)
                                               sem_post(&delay_consumer);
     sem_wait(&delay_consumer);
                                             sem_post(&sync);
```

Action	delay_cons=0	Syn=1	Item=0
Enter_CS	0	0	0
	0	0	1
	0	0	1
	0	0	1
	1	0	1
Exit_CS	1	1	1
	1	1	1
Enter_CS	1	0	1
	1	0	0
	1	0	0
Exit_CS	1	1	0
	1	1	0





Action	delay_cons=0	Syn=1	Item=0
	1	1	0
	0	1	0

<pre>void * consumer(void * p) {</pre>	<pre>void * producer(void * p) {</pre>
<pre>sem_wait(&delay_consumer); while(1)</pre>	while(1)
<pre>{ sem_wait(&sync); items; printf("%d\n", items); sem_post(&sync); if(items == 0) sem_wait(&delay_consumer); 1 => 0</pre>	<pre>{ sem_wait(&sync); items++; printf("%d\n", items); if(items == 1) sem_post(&delay_consumer); sem_post(&sync);</pre>
} }	}





Action	delay_cons=0	Syn=1	Item=0
	1	1	0
	0	1	0
Enter_CS	0	0	0

```
void * consumer(void * p)
                                          void * producer(void * p)
  sem_wait(&delay_consumer);
 while(1)
                                            while(1)
   sem\ wait(\&sync); 1 => 0
                                              sem wait(&sync);
   items--:
                                              items++:
   printf("%d\n", items);
                                              printf("%d\n", items);
   sem_post(&sync);
                                             if(items == 1)
   if(items == 0)
                                               sem post(&delay consumer);
     sem_wait(&delay_consumer);
                                              sem_post(&sync);
```



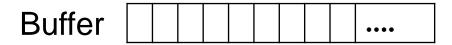


Consumer trying to access non-existing item

Action	delay_cons=0	Syn=1	Item=0
	1	1	0
	0	1	0
Enter_CS	0	0	0
Race_C	0	0	-1

<pre>void * consumer(void * p) {</pre>	<pre>void * producer(void * p) {</pre>
sem_wait(&delay_consumer);	
while(1)	while(1)
<pre>{ sem_wait(&sync); items; 0 => -1 • printf("%d\n", items); sem_post(&sync); if(items == 0)</pre>	<pre>{ sem_wait(&sync); items++; printf("%d\n", items); if(items == 1) sem_post(&delay_consumer);</pre>
<pre>sem_wait(&delay_consumer);</pre>	sem_post(&sync);
}	}
}	}





```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++;
   printf("%d\n", items);
                                             printf("%d\n", items);
   sem_post(&sync);
                                             if(items == 1)
                                               sem post(&delay consumer);
   if(items == 0)
     sem_wait(&delay_consumer);
                                             sem post(&sync);
```

Action	delay_cons=0	Syn=1	Item=0
	1	1	0
	0	1	0
Enter_CS	0	0	0
Race_C	0	0	-1
	0	0	-1





```
void * consumer(void * p)
                                          void * producer(void * p)
 sem_wait(&delay_consumer);
 while(1)
                                            while(1)
   sem_wait(&sync);
                                              sem_wait(&sync);
   items--:
                                              items++:
   printf("%d\n", items);
                                              printf("%d\n", items);
   sem post(\&sync); 0 \Rightarrow 1
                                              if(items == 1)
   if(items == 0)
                                                sem_post(&delay_consumer);
     sem_wait(&delay_consumer);
                                              sem post(&sync);
```

Action	delay_cons=0	Syn=1	Item=0
	1	1	0
	0	1	0
Enter_CS	0	0	0
Race_C	0	0	-1
	0	0	-1
Exit_CS	0	1	-1



```
Buffer ....
```

```
void * producer(void * p)
void * consumer(void * p)
  sem_wait(&delay_consumer);
 while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   sem_post(&sync);
                                            if(items == 1)
   if(items == 0)
                                               sem_post(&delay_consumer);
     sem_wait(&delay_consumer);
                                             sem_post(&sync);
```

}	}
Figure: Single producer/consumer	with unbounded buffer: Race
condition (non-existing element =>	> items=-1)

Action	delay_cons=0	Syn=1	Item=0
	1	1	0
	0	1	0
Enter_CS	0	0	0
Race_C	0	0	-1
	0	0	-1
Exit_CS	0	1	-1
	0	1	-1

The Producer/Consumer Problem One Consumer, One Producer, Unbounded Buffer

 Deadlock could happen when the consumer interrupted after executing sema_wait(&delay_Consumer) (the consumer putting it self to sleep before exiting the critical section) and the producer tried to enter the critical section.

```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
 while(1)
                                            while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   if(items == 0)
                                             if(items == 1)
     sem_wait(&delay_consumer); 0=>-1(sleep) sem_post(&delay_consumer);
   sem_post(&sync);
                                             sem_post(&sync);
```

Figure: Single producer/consumer with unbounded buffer: deadlocks



The Producer/Consumer Problem One Consumer, One Producer, Unbounded Buffer (Deadlocks)



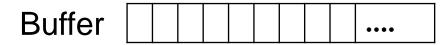
Action	delay_cons=0	Syn=1	Item=0
	-1	1	0

```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer); 0 => -1
  while(1)
                                           while(1)
                                             sem_wait(&sync);
   sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items);
                                            printf("%d\n", items);
   if(items == 0)
                                            if(items == 1)
     sem_wait(&delay_consumer);
                                              sem_post(&delay_consumer);
   sem_post(&sync);
                                             sem_post(&sync);
```

Figure: Single producer/consumer with unbounded buffer: deadlocks



The Producer/Consumer Problem One Consumer, One Producer, Unbounded Buffer (Deadlocks)

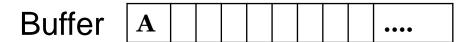


Action	delay_cons=0	Syn=1	Item=0
	-1	1	0
Enter_CS	-1	0	0

```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                            while(1)
   sem_wait(&sync);
                                              sem_wait(\&sync); 1 => 0
   items--:
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   if(items == 0)
                                             if(items == 1)
     sem_wait(&delay_consumer);
                                               sem_post(&delay_consumer);
   sem_post(&sync);
                                             sem_post(&sync);
```

Figure: Single producer/consumer with unbounded buffer: deadlocks



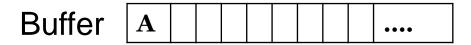


Action	delay_cons=0	Syn=1	Item=0
	-1	1	0
Enter_CS	-1	0	0
	-1	0	1

```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
                                             items++; 0 => 1
   items--:
   printf("%d\n", items);
                                             printf("%d\n", items);
   if(items == 0)
                                             if(items == 1)
     sem wait(&delay consumer);
                                               sem post(&delay consumer);
   sem_post(&sync);
                                             sem_post(&sync);
```

Figure: Single producer/consumer with unbounded buffer: deadlocks





Action	delay_cons=0	Syn=1	Item=0
	-1	1	0
Enter_CS	-1	0	0
	-1	0	1
	-1	0	1

```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
 while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
                                             printf("%d\n", items);
   printf("%d\n", items);
   if(items == 0)
                                             if(items == 1)
     sem_wait(&delay_consumer);
                                               sem_post(&delay_consumer);
   sem_post(&sync);
                                             sem_post(&sync);
```





```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer); (wakeup)
  while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   if(items == 0)
                                             if(items == 1)
     sem wait(&delay consumer);
                                               sem_post(&delay_consumer); -1=>0
   sem_post(&sync);
                                             sem_post(&sync);
```

Action	delay_cons=0	Syn=1	Item=0
	-1	1	0
Enter_CS	-1	0	0
	-1	0	1
	-1	0	1
	-1	0	1
Wakeup_C	0	0	1



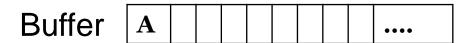


```
void * consumer(void * p)
{
   sem_wait(&delay_consumer);
   while(1)
   {
      sem_wait(&sync);
      items--;
      printf("%d\n", items);
      if(items == 0)
           sem_wait(&delay_consumer);
      sem_post(&sync);
   }
}
```

```
void * producer(void * p)
{
  while(1)
  {
    sem_wait(&sync);
    items++;
    printf("%d\n", items);
    if(items == 1)
        sem_post(&delay_consumer);
    sem_post(&sync); 0 => 1
  }
}
```

Action	delay_cons=0	Syn=1	Item=0
	-1	1	0
Enter_CS	-1	0	0
	-1	0	1
	-1	0	1
	-1	0	1
Wakeup_C	0	0	1
Exit_CS	0	0	1





```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
   sem_wait(\&sync); 1 => 0
                                             sem wait(&sync);
   items--:
                                             items++:
                                             printf("%d\n", items);
   printf("%d\n", items);
   if(items == 0)
                                             if(items == 1)
     sem_wait(&delay_consumer);
                                               sem_post(&delay_consumer);
   sem_post(&sync);
                                             sem_post(&sync);
```

Action	delay_cons=0	Syn=1	Item=0
	-1	1	0
Enter_CS	-1	0	0
	-1	0	1
	-1	0	1
	-1	0	1
Wakeup_C	0	0	1
Exit_CS	0	0	1
Enter_CS	0	0	1

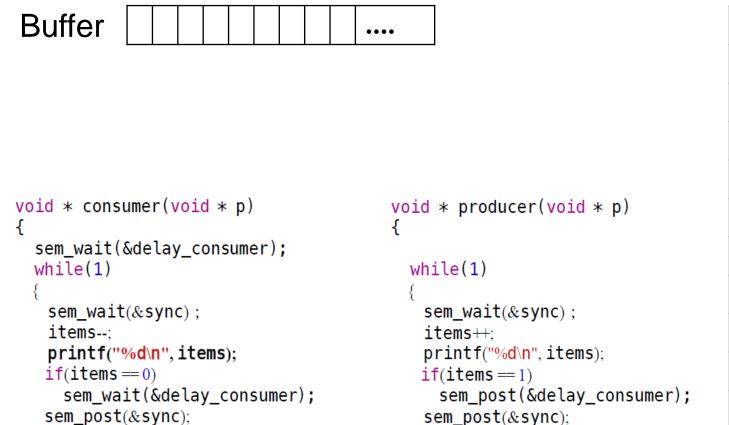


```
Buffer
                                        ••••
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
                                           while(1)
  while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--: 1 => 0
                                             items++:
   printf("%d\n", items);
                                             printf("%d\n", items);
   if(items == 0)
                                            if(items == 1)
     sem_wait(&delay_consumer);
                                              sem_post(&delay_consumer);
   sem_post(&sync);
                                             sem_post(&sync);
```

Figure: Single producer/consumer with unbounded buffer: deadlo	Figure:	Sinale	producer	/consumer	with	unbounded	buffer:	deadloc
--	---------	--------	----------	-----------	------	-----------	---------	---------

Action	delay_cons=0	Syn=1	Item=0
	-1	1	0
Enter_CS	-1	0	0
	-1	0	1
	-1	0	1
	-1	0	1
Wakeup_C	0	0	1
Exit_CS	0	0	1
Enter_CS	0	0	1
	0	0	0

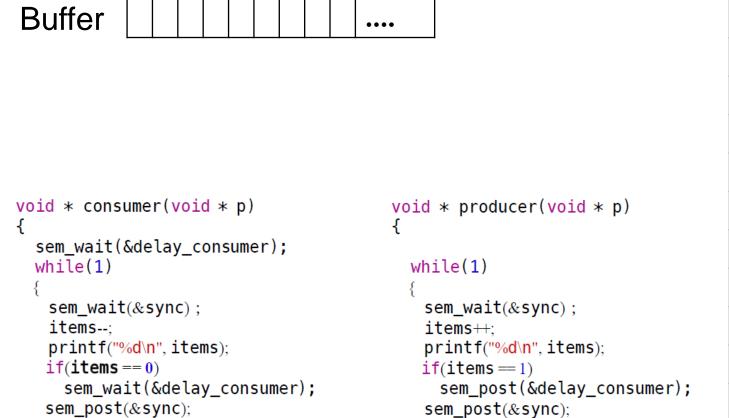




Action	delay_cons=0	Syn=1	Item=0
	-1	1	0
Enter_CS	-1	0	0
	-1	0	1
	-1	0	1
	-1	0	1
Wakeup_C	0	0	1
Exit_CS	0	0	1
Enter_CS	0	0	1
	0	0	0
	0	0	0

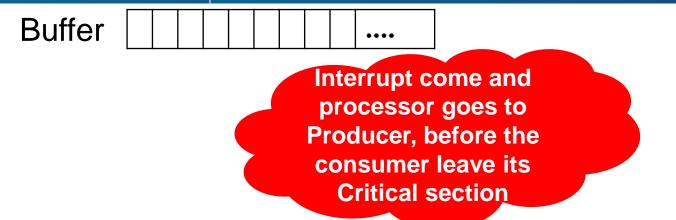
Figure: Single producer/consumer with unbounded buffer: deadlocks





Action	delay_cons=0	Syn=1	Item=0
	-1	1	0
Enter_CS	-1	0	0
	-1	0	1
	-1	0	1
	-1	0	1
Wakeup_C	0	0	1
Exit_CS	0	0	1
Enter_CS	0	0	1
	0	0	0
	0	0	0
	0	0	0





Action	delay_cons=0	Syn=1	Item=0
	0	0	0
Block_C	-1	0	0
	-1	0	0

```
void * consumer(void * p)
                                          void * producer(void * p)
  sem wait(&delay consumer);
  while(1)
                                            while(1)
   sem_wait(&sync);
                                              sem_wait(&sync);
   items--:
                                             items++:
   printf("%d\n", items);
                                              printf("%d\n", items);
   if(items == 0)
                                             if(items == 1)
     sem_wait(&delay_consumer); 0=>-1(sleep) sem_post(&delay_consumer);
   sem_post(&sync);
                                              sem_post(&sync);
```





Action	delay_cons=0	Syn=1	Item=0
	0	0	0
Block_C	-1	0	0
Deadlock	-1	-1	0

```
void * consumer(void * p)
                                           void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                             while(1)
                                              sem_wait(\&sync); 0 \Rightarrow -1 (sleep)
   sem wait(&sync);
   items--:
                                              items++:
   printf("%d\n", items);
                                              printf("%d\n", items);
   if(items == 0)
                                              if(items == 1)
     sem_wait(&delay_consumer);
                                                sem_post(&delay_consumer);
   sem_post(&sync);
                                              sem_post(&sync);
```





Action	delay_cons=0	Syn=1	Item=0	temp
Block_C	-1	1	0	0
Enter_CS	-1	0	0	0

```
void * consumer(void * p)
                                          void * producer(void * p)
  sem_wait(&delay_consumer); 0 => -1
                                            while(1)
  while(1)
                               (sleep)
   sem_wait(&sync);
                                              sem_wait(\&sync); 1 => 0
   items--:
                                              items++:
   temp = items:
                                              printf("%d\n", items);
   printf("%d\n", items);
                                              if(items == 1)
   sem_post(&sync);
                                                sem_post(&delay_consumer);
   if(temp = 0)
                                              sem_post(&sync);
      sem wait(&delay consumer);
```

Tips:

- Add temp, private variable(not shared variable) &
- Reduces the size of critical section(the probability of deadlock decreases)

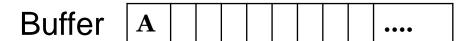


Buffer A |

Action	delay_cons=0	Syn=1	Item=0	temp
Block_C	-1	1	0	0
	-1	0	0	0
	-1	0	1	0

```
void * consumer(void * p)
                                          void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                            while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++: 0 => 1
   temp = items;
                                             printf("%d\n", items);
   printf("%d\n", items);
                                             if(items == 1)
                                               sem post(&delay consumer);
   sem_post(&sync);
   if(temp == 0)
                                             sem_post(&sync);
      sem_wait(&delay_consumer);
```





Action	delay_cons=0	Syn=1	Item=0	temp
Block_C	-1	1	0	0
	-1	0	0	0
	-1	0	1	0
	-1	0	1	0

```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   temp = items;
                                             printf("%d\n", items);
   printf("%d\n", items);
                                             if(items == 1)
                                               sem_post(&delay_consumer);
   sem_post(&sync);
                                             sem_post(&sync);
   if(temp = 0)
      sem_wait(&delay_consumer);
```

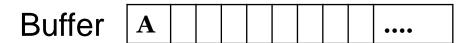




Action	delay_cons=0	Syn=1	Item=0	temp
Block_C	-1	1	0	-
	-1	0	0	-
	-1	0	1	-
	-1	0	1	-
	-1	0	1	-

```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   temp = items;
                                             printf("%d\n", items);
   printf("%d\n", items);
                                             if(items == 1)
                                               sem_post(&delay_consumer);
   sem_post(&sync);
   if(temp = 0)
                                             sem_post(&sync);
      sem_wait(&delay_consumer);
```



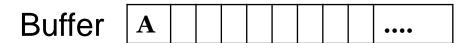


```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer); (wakeup)
 while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
                                             printf("%d\n", items);
   temp = items:
   printf("%d\n", items);
                                             if(items == 1)
                                               sem_post(&delay_consumer); -1=>0
   sem_post(&sync);
   if(temp == 0)
                                             sem_post(&sync);
      sem_wait(&delay_consumer);
```

Figure: Single	producer/consumer	with unbounded	buffer: correct	solution
----------------	-------------------	----------------	-----------------	----------

Action	delay_cons=0	Syn=1	Item=0	temp
Block_C	-1	1	0	_
	-1	0	0	_
	-1	0	1	_
	-1	0	1	_
	-1	0	1	-
Wakeup_C	0	0	1	_





```
void * consumer(void * p)
                                          void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                             while(1)
   sem_wait(&sync);
                                              sem_wait(&sync);
   items--:
                                              items++:
   temp = items;
                                              printf("%d\n", items);
   printf("%d\n", items);
                                              if(items == 1)
                                                sem_post(&delay_consumer);
   sem_post(&sync);
                                              sem_post(\&sync); 0 \Rightarrow 1
   if(temp = 0)
      sem_wait(&delay_consumer);
```

Action	delay_cons=0	Syn=1	Item=0	temp
Block_C	-1	1	0	-
	-1	0	0	-
	-1	0	1	-
	-1	0	1	-
	-1	0	1	-
Wakeup_C	0	0	1	-
Exit_CS	0	1	1	_

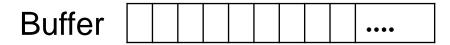




```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
   sem_wait(\&sync); 1 => 0
                                             sem_wait(&sync);
   items--:
                                             items++:
   temp = items;
                                             printf("%d\n", items);
   printf("%d\n", items);
                                             if(items == 1)
                                               sem_post(&delay_consumer);
   sem_post(&sync);
   if(temp = 0)
                                             sem_post(&sync);
      sem_wait(&delay_consumer);
```

Action	delay_cons=0	Syn=1	Item=0	temp
Block_C	-1	1	0	_
	-1	0	0	-
	-1	0	1	-
	-1	0	1	-
	-1	0	1	-
Wakeup_C	0	0	1	-
Exit_CS	0	1	1	_
Enter_CS	0	0	1	_





```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items-: 1 => 0
                                             items++;
   temp = items;
                                             printf("%d\n", items);
   printf("%d\n", items);
                                             if(items == 1)
                                               sem_post(&delay_consumer);
   sem_post(&sync);
   if(temp == 0)
                                             sem_post(&sync);
      sem_wait(&delay_consumer);
```

Action	delay_cons=0	Syn=1	Item=0	temp
Block_C	-1	1	0	-
	-1	0	0	-
	-1	0	1	-
	-1	0	1	-
	-1	0	1	-
Wakeup_C	0	0	1	-
Exit_CS	0	1	1	-
Enter_CS	0	0	1	-
	0	0	0	_





```
void * consumer(void * p)
{
   sem_wait(&delay_consumer);
   while(1)
   {
      sem_wait(&sync);
      items--;
      temp = items;
      printf("%d\n", items);
      sem_post(&sync);
      if(temp = 0)
           sem_wait(&delay_consumer);
    }
}
```

```
void * producer(void * p)
{
  while(1)
  {
    sem_wait(&sync);
    items++;
    printf("%d\n", items);
    if(items == 1)
        sem_post(&delay_consumer);
    sem_post(&sync);
  }
}
```

Action	delay_cons=0	Syn=1	Item=0	temp
Block_C	-1	1	0	-
	-1	0	0	_
	-1	0	1	-
	-1	0	1	-
	-1	0	1	_
Wakeup_C	0	0	1	_
Exit_CS	0	1	1	_
Enter_CS	0	0	1	_
	0	0	0	_
	0	0	0	0

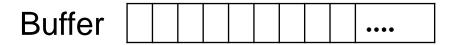




```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
   sem wait(&sync);
                                             sem wait(&sync);
   items--:
                                             items++:
   temp = items;
                                             printf("%d\n", items);
   printf("%d\n", items);
                                            if(items == 1)
   sem_post(&sync);
                                              sem_post(&delay_consumer);
   if(temp = 0)
                                             sem_post(&sync);
      sem_wait(&delay_consumer);
```

Action	delay_cons=0	Syn=1	Item=0	temp
Block_C	-1	1	0	-
	-1	0	0	-
	-1	0	1	-
	-1	0	1	-
	-1	0	1	-
Wakeup_C	0	0	1	-
Exit_CS	0	1	1	-
Enter_CS	0	0	1	-
	0	0	0	-
	0	0	0	0
	0	0	0	0

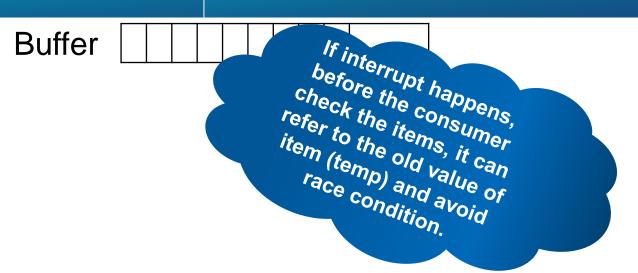




```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                            while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   temp = items;
                                             printf("%d\n", items);
   printf("%d\n", items);
                                             if(items == 1)
   sem_post(&sync); 0 => 1
                                               sem post(&delay consumer);
   if(temp = 0)
                                             sem_post(&sync);
      sem_wait(&delay_consumer);
```

Action	delay_cons=0	Syn=1	Item=0	temp
Block_C	-1	1	0	_
	-1	0	0	-
	-1	0	1	-
	-1	0	1	_
	-1	0	1	-
Wakeup_C	0	0	1	-
Exit_CS	0	1	1	-
Enter_CS	0	0	1	_
	0	0	0	_
	0	0	0	0
	0	0	0	0
Exit_CS	0	1	0	0





Action	delay_cons=0	Syn=1	Item=0	temp
Exit_CS	0	1	0	0
Enter_CS	0	0	0	0

```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
                                             sem_wait(&sync); 1 => 0
   sem_wait(&sync);
   items--:
                                             items++:
                                             printf("%d\n", items);
   temp = items:
   printf("%d\n", items);
                                            if(items == 1)
                                               sem_post(&delay_consumer);
   sem_post(&sync);
   if(temp = 0)
                                             sem_post(&sync);
      sem_wait(&delay_consumer);
```



Buffer	A									••••	
--------	---	--	--	--	--	--	--	--	--	------	--

Action	delay_cons=0	Syn=1	Item=0	temp
Exit_CS	0	1	0	0
Enter_CS	0	0	0	0
	0	0	1	0

```
void * consumer(void * p)
                                         void * producer(void * p)
 sem_wait(&delay_consumer);
 while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem wait(&sync);
   items--:
                                            items++: 0 => 1
   temp = items;
                                             printf("%d\n", items);
   printf("%d\n", items);
                                            if(items == 1)
                                              sem_post(&delay_consumer);
   sem_post(&sync);
   if(temp = 0)
                                             sem_post(&sync);
     sem_wait(&delay_consumer);
```

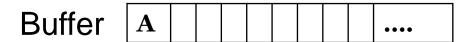




```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   temp = items;
                                             printf("%d\n", items);
   printf("%d\n", items);
                                            if(items = 1)
                                              sem_post(&delay_consumer);
   sem_post(&sync);
   if(temp = 0)
                                             sem post(&sync);
      sem_wait(&delay_consumer);
```

Action	delay_cons=0	Syn=1	Item=0	temp
Exit_CS	0	1	0	0
Enter_CS	0	0	0	0
	0	0	1	0
	0	0	1	0

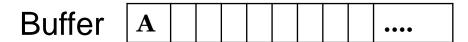




```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
                                             sem_wait(&sync);
   sem_wait(&sync);
   items--:
                                             items++:
                                             printf("%d\n", items);
   temp = items;
   printf("%d\n", items);
                                             if(items == 1)
                                               sem_post(&delay_consumer);
   sem_post(&sync);
                                             sem_post(&sync);
   if(temp = 0)
      sem_wait(&delay_consumer);
```

Action	delay_cons=0	Syn=1	Item=0	temp
Exit_CS	0	1	0	0
Enter_CS	0	0	0	0
	0	0	1	0
	0	0	1	0
	0	0	1	0





<pre>void * consumer(void * p) { sem_wait(&delay_consumer); while(1) { sem_wait(&sync); items; temp = items; printf("%d\n", items); sem_post(&sync); if(temp == 0) sem_wait(&delay_consumer);</pre>	<pre>void * producer(void * p) { while(1) { sem_wait(&sync); items++; printf("%d\n", items); if(items == 1) sem_post(&delay_consumer); 0=>1 sem_post(&sync); }</pre>
}	}
}	

Action	delay_cons=0	Syn=1	Item=0	temp
Exit_CS	0	1	0	0
Enter_CS	0	0	0	0
	0	0	1	0
	0	0	1	0
	0	0	1	0
	1	0	1	0





```
void * consumer(void * p)
                                          void * producer(void * p)
 sem_wait(&delay_consumer);
 while(1)
                                             while(1)
                                              sem_wait(&sync);
   sem_wait(&sync);
   items--:
                                              items++:
                                              printf("%d\n", items);
   temp = items;
   printf("%d\n", items);
                                              if(items == 1)
                                                sem_post(&delay_consumer);
   sem_post(&sync);
   if(temp == 0)
                                              sem_post(\&sync); 0 \Rightarrow 1
      sem_wait(&delay_consumer);
```

Action	delay_cons=0	Syn=1	Item=0	temp
Exit_CS	0	1	0	0
Enter_CS	0	0	0	0
	0	0	1	0
	0	0	1	0
	0	0	1	0
	1	0	1	0
Exit_CS	1	1	1	0





```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                            while(1)
                                             sem_wait(&sync);
   sem_wait(&sync);
   items--:
                                             items++:
   temp = items;
                                             printf("%d\n", items);
   printf("%d\n", items);
                                             if(items == 1)
                                               sem_post(&delay_consumer);
   sem_post(&sync);
   if(temp == 0)
                                             sem_post(&sync);
      sem_wait(&delay_consumer);
```

Action	delay_cons=0	Syn=1	Item=0	temp
Exit_CS	0	1	0	0
Enter_CS	0	0	0	0
	0	0	1	0
	0	0	1	0
	0	0	1	0
	1	0	1	0
Exit_CS	1	1	1	0
	1	1	1	0





```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   temp = items;
                                             printf("%d\n", items);
   printf("%d\n", items);
                                             if(items == 1)
                                               sem post(&delay consumer);
   sem post(&sync);
   if(temp == 0)
                                             sem_post(&sync);
      sem_wait(&delay_consumer); 1 => 0
```

Action	delay_cons=0	Syn=1	Item=0	temp
Exit_CS	0	1	0	0
Enter_CS	0	0	0	0
	0	0	1	0
	0	0	1	0
	0	0	1	0
	1	0	1	0
Exit_CS	1	1	1	0
	1	1	1	0
	0	1	1	0

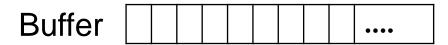


Buffer A

```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
   sem_wait(\&sync); 1 => 0
                                             sem_wait(&sync);
   items--:
                                             items++:
   temp = items:
                                             printf("%d\n", items);
   printf("%d\n", items);
                                             if(items == 1)
                                               sem_post(&delay_consumer);
   sem_post(&sync);
   if(temp = 0)
                                             sem_post(&sync);
      sem_wait(&delay_consumer);
```

Action	delay_cons=0	Syn=1	Item=0	temp
Exit_CS	0	1	0	0
Enter_CS	0	0	0	0
	0	0	1	0
	0	0	1	0
	0	0	1	0
	1	0	1	0
Exit_CS	1	1	1	0
	1	1	1	0
	0	1	1	0
Enter_CS	0	0	1	0





```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
 while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--: 1 => 0
                                             items++:
                                             printf("%d\n", items);
   temp = items;
   printf("%d\n", items);
                                             if(items == 1)
                                              sem_post(&delay_consumer);
   sem_post(&sync);
   if(temp == 0)
                                             sem_post(&sync);
      sem wait(&delay consumer);
```

Action	delay_cons=0	Syn=1	Item=0	temp
Exit_CS	0	1	0	0
Enter_CS	0	0	0	0
	0	0	1	0
	0	0	1	0
	0	0	1	0
	1	0	1	0
Exit_CS	1	1	1	0
	1	1	1	0
	0	1	1	0
Enter_CS	0	0	1	0
	0	0	0	0





```
void * consumer(void * p)
                                           void * producer(void * p)
  sem_wait(&delay_consumer);
  whi\overline{l}e(1)
                                             while(1)
   sem_wait(&sync);
                                               sem_wait(&sync);
   items--:
                                               items++:
   temp = items;
                                               printf("%d\n", items);
   printf("%d\n", items);
                                              if(items == 1)
   sem_post(&sync);
                                                sem_post(&delay_consumer);
   if(temp == 0)
                                               sem_post(&sync);
      sem wait(&delay consumer);
```

Action	delay_cons=0	Syn=1	Item=0	temp
Exit_CS	0	1	0	0
Enter_CS	0	0	0	0
	0	0	1	0
	0	0	1	0
	0	0	1	0
	1	0	1	0
Exit_CS	1	1	1	0
	1	1	1	0
	0	1	1	0
Enter_CS	0	0	1	0
	0	0	0	0
	0	0	0	0

Figure: Single producer/consumer with unbounded buffer: correct solution



Buffer										••••	
--------	--	--	--	--	--	--	--	--	--	------	--

Action	delay_cons=0	Syn=1	Item=0	temp
	0	0	0	0
	0	0	0	0

```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   temp = items;
                                             printf("%d\n", items);
   printf("%d\n", items);
                                             if(items == 1)
   sem_post(&sync);
                                               sem_post(&delay_consumer);
   if(temp == 0)
                                             sem_post(&sync);
      sem wait(&delay consumer);
```



Buffer										••••	
--------	--	--	--	--	--	--	--	--	--	------	--

Action	delay_cons=0	Syn=1	Item=0	temp	
	0	0	0	0	
	0	0	0	0	
Exit_CS	0	1	0	0	

```
void * consumer(void * p)
                                           void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                             while(1)
   sem_wait(&sync);
                                              sem_wait(&sync);
   items--:
                                              items++:
   temp = items;
                                              printf("%d\n", items);
   printf("%d\n", items);
                                              if(items == 1)
                                                sem_post(&delay_consumer);
   sem_post(\&sync); 0 \Rightarrow 1
                                              sem_post(&sync);
   if(temp = 0)
      sem_wait(&delay_consumer);
```



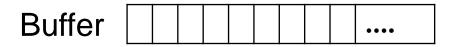


```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
                                             printf("%d\n", items);
   temp = items;
   printf("%d\n", items);
                                             if(items == 1)
                                               sem_post(&delay_consumer);
   sem_post(&sync);
   if(temp == 0)
                                             sem_post(&sync);
      sem_wait(&delay_consumer);
```

Action	delay_cons=0	Syn=1	Item=0	temp	
	0	0	0	0	
	0	0	0	0	
Exit_CS	0	1	0	0	
	0	1	0	0	

Figure: Single producer/consumer with unbounded buffer: correct solution





```
void * consumer(void * p)
                                         void * producer(void * p)
  sem_wait(&delay_consumer);
  while(1)
                                           while(1)
   sem_wait(&sync);
                                             sem_wait(&sync);
   items--:
                                             items++:
   temp = items;
                                             printf("%d\n", items);
   printf("%d\n", items);
                                             if(items == 1)
   sem_post(&sync);
                                               sem_post(&delay_consumer);
   if(temp = 0)
                                             sem_post(&sync);
      sem_wait(&delay_consumer);0 => -1
```

Action	delay_cons=0	Syn=1	Item=0	temp
	0	0	0	0
	0	0	0	0
Exit_CS	0	1	0	0
	0	1	0	0
Block_C	-1	1	0	0

Figure: Single producer/consumer with unbounded buffer: correct solution

- Modern Operating Systems (Tanenbaum): Chapter 2(2.3.5,
 2.5.1)
- Operating System Concepts (Silberschatz): Chapter 6(6.6-7)
- Operating Systems: Internals and Design Principles (Starlings): **Chapter 5(5.3, 5.6)**