Operating Systems
Condition Variables Advanced + Semaphores

Nipun Batra

Condition Variables

Condition Variables

- **wait (cond_t *cv, mutex_t *lock)**

Condition Variables

- **wait (cond_t *cv, mutex_t *lock)**
  - Assumes lock is held when wait() is called

Condition Variables

- **wait (cond_t *cv, mutex_t *lock)**
  - Assumes lock is held when wait() is called
  - Puts caller to sleep + atomically releases lock

Condition Variables

- **wait (cond_t *cv, mutex_t *lock)**
  - Assumes lock is held when wait() is called
  - Puts caller to sleep + atomically releases lock
  - When awoken, reacquires lock before returning

Condition Variables

- **wait (cond_t *cv, mutex_t *lock)**
  - Assumes lock is held when wait() is called
  - Puts caller to sleep + atomically releases lock
  - When awoken, reacquires lock before returning
- **signal (cond_t *cv)**

Condition Variables

- **wait (cond_t *cv, mutex_t *lock)**
  - Assumes lock is held when wait() is called
  - Puts caller to sleep + atomically releases lock
  - When awoken, reacquires lock before returning
- **signal (cond_t *cv)**
  - Wake a single waiting thread

Condition Variables

- **wait (cond_t *cv, mutex_t *lock)**
  - Assumes lock is held when wait() is called
  - Puts caller to sleep + atomically releases lock
  - When awoken, reacquires lock before returning
- **signal (cond_t *cv)**
  - Wake a single waiting thread
  - If there is no waiting thread, just return, do nothing

The Producer Consumer Problem

Producer adds to the buffer

Bounded Buffer

The Producer Consumer Problem

Producer adds to the buffer

Consumer removes from the buffer

Bounded Buffer

The Producer Consumer Problem

Buffer Full - Producer(s) have to wait

Bounded Buffer

The Producer Consumer Problem

Buffer Empty - Consumer(s) have to wait

Bounded Buffer

The Producer Consumer Problem (Buffer size =1)

```
1   int buffer;
2   int count = 0;   // initially, empty
3
4   void put(int value) {
5   assert(count == 0);
6   count = 1;
7   buffer = value;
8   }
9
10   int get() {
11   assert(count == 1);
12   count = 0;
13   return buffer;
14   }
```

The Producer Consumer Problem (Buffer size =1)

```
1   int buffer;
2   int count = 0;   // initially, empty
3
4   void put(int value) {
5   assert(count == 0);
6   count = 1;
7   buffer = value;
8   }
9
10  int get() {
11  assert(count == 1);
12  count = 0;
13  return buffer;
14  }
```

Insert into buffer (produce) only if buffer is empty

The Producer Consumer Problem (Buffer size =1)

```
1   int buffer;
2   int count = 0;   // initially, empty
3
4   void put(int value) {
5   assert(count == 0);
6   count = 1;
7   buffer = value;
8   }
9
10  int get() {
11  assert(count == 1);
12  count = 0;
13  return buffer;
14  }
```

Delete from buffer (consume) only if buffer is full

The Producer Consumer Problem (Buffer size =1) [Attempt #1]

```
1  void *producer(void *arg) {
2  int i;
3  int loops = (int) arg;
4  for (i = 0; i < loops; i++) {
5  put(i);
6  }
7  }
8
9  void *consumer(void *arg) {
10   int i;
11   while (1) {
12   int tmp = get();
13   printf("%d\n", tmp);
14   }
15   }
```

The Producer Consumer Problem (Buffer size =1) [Attempt #1]

```
1  void *producer(void *arg) {
2  int i;
3  int loops = (int) arg;
4  for (i = 0; i < loops; i++) {
5  put(i);
6  }
7  }
8
9  void *consumer(void *arg) {
10   int i;
11   while (1) {
12   int tmp = get();
13   printf("%d\n", tmp);
14   }
15   }
```

Producer puts an integer into the shared buffer loops number of times.

The Producer Consumer Problem (Buffer size =1) [Attempt #1]

```
1   void *producer(void *arg) {
2   int i;
3   int loops = (int) arg;
4   for (i = 0; i < loops; i++) {
5   put(i);
6   }
7   }
8
9   void *consumer(void *arg) {
10    int i;
11    while (1) {
12    int tmp = get();
13    printf("%d\n", tmp);
14    }
15    }
```

Consumer gets data out of the buffer.

The Producer Consumer Problem (Buffer size =1) [Attempt #1]

```
1  void *producer(void *arg) {
2  int i;
3  int loops = (int) arg;
4  for (i = 0; i < loops; i++) {
5  put(i);
6  }
7  }
8
9  void *consumer(void *arg) {
10   int i;
11   while (1) {
12   int tmp = get();
13   printf("%d\n", tmp);
14   }
15   }
```

The Producer Consumer Problem (Buffer size =1) [Attempt #1]

```
1  void *producer(void *arg) {
2  int i;
3  int loops = (int) arg;
4  for (i = 0; i < loops; i++) {
5  put(i);
6  }
7  }
8
9  void *consumer(void *arg) {
10   int i;
11   while (1) {
12   int tmp = get();
13   printf("%d\n", tmp);
14   }
15   }
```

What's the problem

The Producer Consumer Problem (Buffer size =1) [Attempt #1]

```
1  void *producer(void *arg) {
2  int i;
3  int loops = (int) arg;
4  for (i = 0; i < loops; i++) {
5  put(i);
6  }
7  }
8
9  void *consumer(void *arg) {
10   int i;
11   while (1) {
12   int tmp = get();
13   printf("%d\n", tmp);
14   }
15   }
```

What's the problem with this approach?

The Producer Consumer Problem (Buffer size =1) [Attempt #1]

```
1  void *producer(void *arg) {
2  int i;
3  int loops = (int) arg;
4  for (i = 0; i < loops; i++) {
5  put(i);
6  }
7  }
8
9  void *consumer(void *arg) {
10   int i;
11   while (1) {
12   int tmp = get();
13   printf("%d\n", tmp);
14   }
15   }
```

What's the problem with this approach?

The Producer Consumer Problem (Buffer size =1) [Attempt #1]

```
1  void *producer(void *arg) {
2  int i;
3  int loops = (int) arg;
4  for (i = 0; i < loops; i++) {
5  put(i);
6  }
7  }
8
9  void *consumer(void *arg) {
10   int i;
11   while (1) {
12   int tmp = get();
13   printf("%d\n", tmp);
14   }
15   }
```

What's the problem with this approach?

Multiple threads accessing

The Producer Consumer Problem (Buffer size =1) [Attempt #1]

```
1  void *producer(void *arg) {
2  int i;
3  int loops = (int) arg;
4  for (i = 0; i < loops; i++) {
5  put(i);
6  }
7  }
8
9  void *consumer(void *arg) {
10   int i;
11   while (1) {
12   int tmp = get();
13   printf("%d\n", tmp);
14   }
15   }
```

What's the problem with this approach?

Multiple threads accessing shared resource without locking

The Producer Consumer Problem (Buffer size =1) [Attempt #2]

```
1  void *producer(void *arg) {
2  int i;
3  int loops = (int) arg;
4  for (i = 0; i < loops; i++) {
5  Pthread_mutex_lock(&mutex);
6  put(i);
7  Pthread_mutex_unlock(&mutex);}
8  }
9    void *consumer(void *arg) {
10   int i;
11   while (1) { Pthread_mutex_lock(&mutex);
12   int tmp = get();
13   Pthread_mutex_unlock(&mutex);
14   printf("%d\n", tmp);
15   } }
```

The Producer Consumer Problem (Buffer size =1) [Attempt #2]

```
1   void *producer(void *arg) {
2   int i;
3   int loops = (int) arg;
4   for (i = 0; i < loops; i++) {
5   Pthread_mutex_lock(&mutex);
6   put(i);
7   Pthread_mutex_unlock(&mutex);}
8   }
9     void *consumer(void *arg) {
10  int i;
11  while (1) { Pthread_mutex_lock(&mutex);
12  int tmp = get();
13  Pthread_mutex_unlock(&mutex);
14  printf("%d\n", tmp);
15  } }
```

What's the problem

The Producer Consumer Problem (Buffer size =1) [Attempt #2]

```
1   void *producer(void *arg) {
2   int i;
3   int loops = (int) arg;
4   for (i = 0; i < loops; i++) {
5   Pthread_mutex_lock(&mutex);
6   put(i);
7   Pthread_mutex_unlock(&mutex);}
8   }
9     void *consumer(void *arg) {
10  int i;
11  while (1) { Pthread_mutex_lock(&mutex);
12  int tmp = get();
13  Pthread_mutex_unlock(&mutex);
14  printf("%d\n", tmp);
15  } }
```

What's the problem with this approach?

The Producer Consumer Problem (Buffer size =1) [Attempt #2]

```
1   void *producer(void *arg) {
2   int i;
3   int loops = (int) arg;
4   for (i = 0; i < loops; i++) {
5   Pthread_mutex_lock(&mutex);
6   put(i);
7   Pthread_mutex_unlock(&mutex);}
8   }
9     void *consumer(void *arg) {
10  int i;
11  while (1) { Pthread_mutex_lock(&mutex);
12  int tmp = get();
13  Pthread_mutex_unlock(&mutex);
14  printf("%d\n", tmp);
15  } }
```

What's the problem with this approach?

The Producer Consumer Problem (Buffer size =1) [Attempt #2]

```
1   void *producer(void *arg) {
2   int i;
3   int loops = (int) arg;
4   for (i = 0; i < loops; i++) {
5   Pthread_mutex_lock(&mutex);
6   put(i);
7   Pthread_mutex_unlock(&mutex);}
8   }
9     void *consumer(void *arg) {
10  int i;
11  while (1) { Pthread_mutex_lock(&mutex);
12  int tmp = get();
13  Pthread_mutex_unlock(&mutex);
14  printf("%d\n", tmp);
15  } }
```

What's the problem with this approach?

No explicit waiting

The Producer Consumer Problem (Buffer size =1) [Attempt #2]

```
1   void *producer(void *arg) {
2   int i;
3   int loops = (int) arg;
4   for (i = 0; i < loops; i++) {
5   Pthread_mutex_lock(&mutex);
6   put(i);
7   Pthread_mutex_unlock(&mutex);}
8   }
9     void *consumer(void *arg) {
10  int i;
11  while (1) { Pthread_mutex_lock(&mutex);
12  int tmp = get();
13  Pthread_mutex_unlock(&mutex);
14  printf("%d\n", tmp);
15  } }
```

What's the problem with this approach?

No explicit waiting on empty and full buffer

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

Write some traces/orderings of consumer/producer to see if this works for single producer and single consumer?

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

Count: 0

Lock held by:?

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

Count: 0

Lock held by: Producer

P1

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

Count: 0

Lock held by: Producer

Comments:count!=1

P1 **P2**

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

Count: **1**

Lock held by: Producer

Comments: Data put into buffer

P1 P2 **P4**

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

Count: **1**

Lock held by: Producer

Comments: Lock held by producer

P1 P2 P4 C1

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

Count: **1**

Lock held by: Producer

Comments: No waiting thread, just returns..

P1 P2 P4 C1 **P5**

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

Count: **1**

Lock held by: ?

Comments: Unlocked ..

P1 P2 P4 C1 P5 **P6**

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

Count: **1**

Comments: Producer will wait till buffer isn't empty

Lock held by: Producer

P1 P2 P4 C1 P5 P6 P1 P2 **P3**

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

Count: **1**

Comments: wait released lock, which consumer takes

Lock held by: ?

P1 P2 P4 C1 P5 P6 P1 P2 P3 **C1**

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

Count: **1**

Comments: wait released lock, which consumer takes

Lock held by: Consumer

P1 P2 P4 C1 P5 P6 P1 P2 P3 C1 **C2 C4**

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

Count: **1**

Comments: producer woken up —> can run now

Lock held by: Consumer

P1 P2 P4 C1 P5 P6 P1 P2 P3 C1 C2 C4 **C5**

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

Write some traces/orderings of consumer/producer to see if this works for single producer and 2 consumers?

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

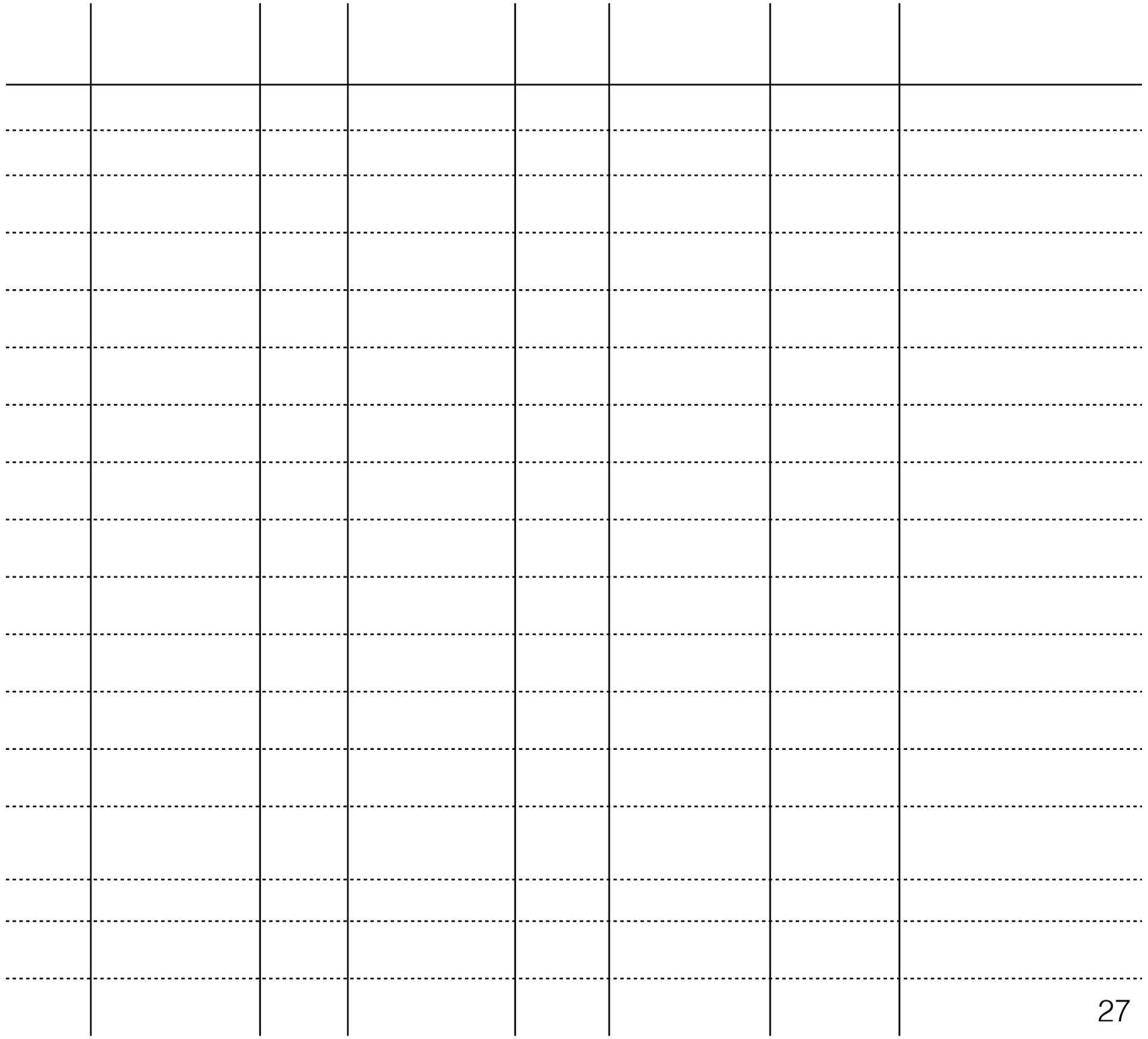
```
Pthread_mutex_lock(&mutex);             // p1
if (count == 1)                         // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                                 // p4
Pthread_cond_signal(&cond);             // p5
Pthread_mutex_unlock(&mutex);           // p6
```

```
Pthread_mutex_lock(&mutex);             // c1
if (count == 0)                         // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                        // c4
Pthread_cond_signal(&cond);             // c5
Pthread_mutex_unlock(&mutex);           // c6
```

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```



The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);                 // p1
if (count == 1)                             // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                                     // p4
Pthread_cond_signal(&cond);                 // p5
Pthread_mutex_unlock(&mutex);               // p6
```

```
Pthread_mutex_lock(&mutex);                 // c1
if (count == 0)                             // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                            // c4
Pthread_cond_signal(&cond);                 // c5
Pthread_mutex_unlock(&mutex);               // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);             // p1
if (count == 1)                         // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                                 // p4
Pthread_cond_signal(&cond);             // p5
Pthread_mutex_unlock(&mutex);           // p6
```

```
Pthread_mutex_lock(&mutex);             // c1
if (count == 0)                         // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                        // c4
Pthread_cond_signal(&cond);             // c5
Pthread_mutex_unlock(&mutex);           // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep		Ready	p1	Running	0	

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep		Ready	p1	Running	0	
	Sleep		Ready	p2	Running	0	

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep		Ready	p1	Running	0	
	Sleep		Ready	p2	Running	0	
	Sleep		Ready	p4	Running	1	Buffer now full

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep		Ready	p1	Running	0	
	Sleep		Ready	p2	Running	0	
	Sleep		Ready	p4	Running	1	Buffer now full
	Ready		Ready	p5	Running	1	T_c1 awoken

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep		Ready	p1	Running	0	
	Sleep		Ready	p2	Running	0	
	Sleep		Ready	p4	Running	1	Buffer now full
	Ready		Ready	p5	Running	1	T_c1 awoken
	Ready		Ready	p6	Running	1	

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep		Ready	p1	Running	0	
	Sleep		Ready	p2	Running	0	
	Sleep		Ready	p4	Running	1	Buffer now full
	Ready		Ready	p5	Running	1	T_c1 awoken
	Ready		Ready	p6	Running	1	
	Ready		Ready	p1	Running	1	

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep		Ready	p1	Running	0	
	Sleep		Ready	p2	Running	0	
	Sleep		Ready	p4	Running	1	Buffer now full
	Ready		Ready	p5	Running	1	T_c1 awoken
	Ready		Ready	p6	Running	1	
	Ready		Ready	p1	Running	1	
	Ready		Ready	p2	Running	1	

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep		Ready	p1	Running	0	
	Sleep		Ready	p2	Running	0	
	Sleep		Ready	p4	Running	1	Buffer now full
	Ready		Ready	p5	Running	1	T_c1 awoken
	Ready		Ready	p6	Running	1	
	Ready		Ready	p1	Running	1	
	Ready		Ready	p2	Running	1	
	Ready		Ready	p3	Sleep	1	Buffer full; sleep

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep		Ready	p1	Running	0	
	Sleep		Ready	p2	Running	0	
	Sleep		Ready	p4	Running	1	Buffer now full
	Ready		Ready	p5	Running	1	T_c1 awoken
	Ready		Ready	p6	Running	1	
	Ready		Ready	p1	Running	1	
	Ready		Ready	p2	Running	1	
	Ready		Ready	p3	Sleep	1	Buffer full; sleep
	Ready	c1	Running		Sleep	1	T_c2 sneaks in

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep		Ready	p1	Running	0	
	Sleep		Ready	p2	Running	0	
	Sleep		Ready	p4	Running	1	Buffer now full
	Ready		Ready	p5	Running	1	T_c1 awoken
	Ready		Ready	p6	Running	1	
	Ready		Ready	p1	Running	1	
	Ready		Ready	p2	Running	1	
	Ready		Ready	p3	Sleep	1	Buffer full; sleep
	Ready	c1	Running		Sleep	1	T_c2 sneaks in
	Ready	c2	Running		Sleep	1	

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep		Ready	p1	Running	0	
	Sleep		Ready	p2	Running	0	
	Sleep		Ready	p4	Running	1	Buffer now full
	Ready		Ready	p5	Running	1	T_c1 awoken
	Ready		Ready	p6	Running	1	
	Ready		Ready	p1	Running	1	
	Ready		Ready	p2	Running	1	
	Ready		Ready	p3	Sleep	1	Buffer full; sleep
	Ready	c1	Running		Sleep	1	T_c2 sneaks in
	Ready	c2	Running		Sleep	1	
	Ready	c4	Running		Sleep	0	… and grabs data

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep		Ready	p1	Running	0	
	Sleep		Ready	p2	Running	0	
	Sleep		Ready	p4	Running	1	Buffer now full
	Ready		Ready	p5	Running	1	T_c1 awoken
	Ready		Ready	p6	Running	1	
	Ready		Ready	p1	Running	1	
	Ready		Ready	p2	Running	1	
	Ready		Ready	p3	Sleep	1	Buffer full; sleep
	Ready	c1	Running		Sleep	1	T_c2 sneaks in
	Ready	c2	Running		Sleep	1	
	Ready	c4	Running		Sleep	0	… and grabs data
	Ready	c5	Running		Ready	0	T_p awoken

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep		Ready	p1	Running	0	
	Sleep		Ready	p2	Running	0	
	Sleep		Ready	p4	Running	1	Buffer now full
	Ready		Ready	p5	Running	1	T_c1 awoken
	Ready		Ready	p6	Running	1	
	Ready		Ready	p1	Running	1	
	Ready		Ready	p2	Running	1	
	Ready		Ready	p3	Sleep	1	Buffer full; sleep
	Ready	c1	Running		Sleep	1	T_c2 sneaks in
	Ready	c2	Running		Sleep	1	
	Ready	c4	Running		Sleep	0	… and grabs data
	Ready	c5	Running		Ready	0	T_p awoken
	Ready	c6	Running		Ready	0	

The Producer Consumer Problem (Buffer size =1) [Attempt #3]

```
Pthread_mutex_lock(&mutex);          // p1
if (count == 1)                      // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
if (count == 0)                      // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep		Ready	p1	Running	0	
	Sleep		Ready	p2	Running	0	
	Sleep		Ready	p4	Running	1	Buffer now full
	Ready		Ready	p5	Running	1	T_c1 awoken
	Ready		Ready	p6	Running	1	
	Ready		Ready	p1	Running	1	
	Ready		Ready	p2	Running	1	
	Ready		Ready	p3	Sleep	1	Buffer full; sleep
	Ready	c1	Running		Sleep	1	T_c2 sneaks in
	Ready	c2	Running		Sleep	1	
	Ready	c4	Running		Sleep	0	… and grabs data
	Ready	c5	Running		Ready	0	T_p awoken
	Ready	c6	Running		Ready	0	
c4	Running		Ready		Ready	0	**Whoops**

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
while (count == 0)                   // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

Replace if with while —> check condition again. Good rule of thumb!

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
while (count == 0)                   // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

Think of an order trace with 2 consumers and 1 producer where all the three threads go to sleeping state …

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

T_c1	State	T_c2	State	T_p	State	Count	Comment

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep	c1	Running		Ready	0	

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep	c1	Running		Ready	0	
	Sleep	c2	Running		Ready	0	

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep	c1	Running		Ready	0	
	Sleep	c2	Running		Ready	0	
	Sleep	c3	Sleep		Ready	0	Nothing to get

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep	c1	Running		Ready	0	
	Sleep	c2	Running		Ready	0	
	Sleep	c3	Sleep		Ready	0	Nothing to get
	Sleep		Sleep	p1	Running	0	

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep	c1	Running		Ready	0	
	Sleep	c2	Running		Ready	0	
	Sleep	c3	Sleep		Ready	0	Nothing to get
	Sleep		Sleep	p1	Running	0	
	Sleep		Sleep	p2	Running	0	

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep	c1	Running		Ready	0	
	Sleep	c2	Running		Ready	0	
	Sleep	c3	Sleep		Ready	0	Nothing to get
	Sleep		Sleep	p1	Running	0	
	Sleep		Sleep	p2	Running	0	
	Sleep		Sleep	p4	Running	1	Buffer now full

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep	c1	Running		Ready	0	
	Sleep	c2	Running		Ready	0	
	Sleep	c3	Sleep		Ready	0	Nothing to get
	Sleep		Sleep	p1	Running	0	
	Sleep		Sleep	p2	Running	0	
	Sleep		Sleep	p4	Running	1	Buffer now full
	Ready		Sleep	p5	Running	1	T_c1 awoken

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep	c1	Running		Ready	0	
	Sleep	c2	Running		Ready	0	
	Sleep	c3	Sleep		Ready	0	Nothing to get
	Sleep		Sleep	p1	Running	0	
	Sleep		Sleep	p2	Running	0	
	Sleep		Sleep	p4	Running	1	Buffer now full
	Ready		Sleep	p5	Running	1	T_c1 awoken
	Ready		Sleep	p6	Running	1	

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep	c1	Running		Ready	0	
	Sleep	c2	Running		Ready	0	
	Sleep	c3	Sleep		Ready	0	Nothing to get
	Sleep		Sleep	p1	Running	0	
	Sleep		Sleep	p2	Running	0	
	Sleep		Sleep	p4	Running	1	Buffer now full
	Ready		Sleep	p5	Running	1	T_c1 awoken
	Ready		Sleep	p6	Running	1	
	Ready		Sleep	p1	Running	1	

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep	c1	Running		Ready	0	
	Sleep	c2	Running		Ready	0	
	Sleep	c3	Sleep		Ready	0	Nothing to get
	Sleep		Sleep	p1	Running	0	
	Sleep		Sleep	p2	Running	0	
	Sleep		Sleep	p4	Running	1	Buffer now full
	Ready		Sleep	p5	Running	1	T_c1 awoken
	Ready		Sleep	p6	Running	1	
	Ready		Sleep	p1	Running	1	
	Ready		Sleep	p2	Running	1	

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep	c1	Running		Ready	0	
	Sleep	c2	Running		Ready	0	
	Sleep	c3	Sleep		Ready	0	Nothing to get
	Sleep		Sleep	p1	Running	0	
	Sleep		Sleep	p2	Running	0	
	Sleep		Sleep	p4	Running	1	Buffer now full
	Ready		Sleep	p5	Running	1	T_c1 awoken
	Ready		Sleep	p6	Running	1	
	Ready		Sleep	p1	Running	1	
	Ready		Sleep	p2	Running	1	
	Ready		Sleep	p3	Sleep	1	Must sleep (full)

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep	c1	Running		Ready	0	
	Sleep	c2	Running		Ready	0	
	Sleep	c3	Sleep		Ready	0	Nothing to get
	Sleep		Sleep	p1	Running	0	
	Sleep		Sleep	p2	Running	0	
	Sleep		Sleep	p4	Running	1	Buffer now full
	Ready		Sleep	p5	Running	1	T_c1 awoken
	Ready		Sleep	p6	Running	1	
	Ready		Sleep	p1	Running	1	
	Ready		Sleep	p2	Running	1	
	Ready		Sleep	p3	Sleep	1	Must sleep (full)
c2	Running		Sleep		Sleep	1	Recheck condition

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep	c1	Running		Ready	0	
	Sleep	c2	Running		Ready	0	
	Sleep	c3	Sleep		Ready	0	Nothing to get
	Sleep		Sleep	p1	Running	0	
	Sleep		Sleep	p2	Running	0	
	Sleep		Sleep	p4	Running	1	Buffer now full
	Ready		Sleep	p5	Running	1	T_c1 awoken
	Ready		Sleep	p6	Running	1	
	Ready		Sleep	p1	Running	1	
	Ready		Sleep	p2	Running	1	
	Ready		Sleep	p3	Sleep	1	Must sleep (full)
c2	Running		Sleep		Sleep	1	Recheck condition
c4	Running		Sleep		Sleep	0	T_c1 grabs data

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c1	State	T_c2	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep	c1	Running		Ready	0	
	Sleep	c2	Running		Ready	0	
	Sleep	c3	Sleep		Ready	0	Nothing to get
	Sleep		Sleep	p1	Running	0	
	Sleep		Sleep	p2	Running	0	
	Sleep		Sleep	p4	Running	1	Buffer now full
	Ready		Sleep	p5	Running	1	T_c1 awoken
	Ready		Sleep	p6	Running	1	
	Ready		Sleep	p1	Running	1	
	Ready		Sleep	p2	Running	1	
	Ready		Sleep	p3	Sleep	1	Must sleep (full)
c2	Running		Sleep		Sleep	1	Recheck condition
c4	Running		Sleep		Sleep	0	T_c1 grabs data
c5	Running		Ready		Sleep	0	Oops! Woke T_c2

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

T_c 1	State	T_c 2	State	T_p	State	Count	Comment

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
…	…	…	…	…	…	…	(cont.)

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
…	…	…	…	…	…	…	(cont.)
c6	Running		Ready		Sleep	0	

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
…	…	…	…	…	…	…	(cont.)
c6	Running		Ready		Sleep	0	
c1	Running		Ready		Sleep	0	

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
…	…	…	…	…	…	…	(cont.)
c6	Running		Ready		Sleep	0	
c1	Running		Ready		Sleep	0	
c2	Running		Ready		Sleep	0	

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
…	…	…	…	…	…	…	(cont.)
c6	Running		Ready		Sleep	0	
c1	Running		Ready		Sleep	0	
c2	Running		Ready		Sleep	0	
c3	Sleep		Ready		Sleep	0	Nothing to get

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
…	…	…	…	…	…	…	(cont.)
c6	Running		Ready		Sleep	0	
c1	Running		Ready		Sleep	0	
c2	Running		Ready		Sleep	0	
c3	Sleep		Ready		Sleep	0	Nothing to get
	Sleep	c2	Running		Sleep	0	

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
…	…	…	…	…	…	…	(cont.)
c6	Running		Ready		Sleep	0	
c1	Running		Ready		Sleep	0	
c2	Running		Ready		Sleep	0	
c3	Sleep		Ready		Sleep	0	Nothing to get
	Sleep	c2	Running		Sleep	0	
	Sleep	c3	Sleep		Sleep	0	Everyone asleep …

The Producer Consumer Problem (Buffer size =1) [Attempt #4]

```
Pthread_mutex_lock(&mutex);          // p1
While (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
```

```
Pthread_mutex_lock(&mutex);          // c1
while(count == 0)                    // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
```

T_c 1	State	T_c 2	State	T_p	State	Count	Comment
…	…	…	…	…	…	…	(cont.)
c6	Running		Ready		Sleep	0	
c1	Running		Ready		Sleep	0	
c2	Running		Ready		Sleep	0	
c3	Sleep		Ready		Sleep	0	Nothing to get
	Sleep	c2	Running		Sleep	0	
	Sleep	c3	Sleep		Sleep	0	Everyone asleep …

Key lesson : A consumer should not wake other consumers, only producers and vice versa

The Producer Consumer Problem (Buffer size =1) [Correct Attempt!]

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
while (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
While (count == 0)                   // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Correct Attempt!]

- Modify the code below to use 2 condition variables and solve the problem correctly …

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
while (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
While (count == 0)                   // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Correct Attempt!]

- Modify the code below to use 2 condition variables and solve the problem correctly …
- Wait and signal on fill and empty …

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // p1
while (count == 1)                   // p2
   Pthread_cond_wait(&cond, &mutex)// p3
put(i);                              // p4
Pthread_cond_signal(&cond);          // p5
Pthread_mutex_unlock(&mutex);        // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);          // c1
While (count == 0)                   // c2
  Pthread_cond_wait(&cond, &mutex)// c3
int tmp = get();                     // c4
Pthread_cond_signal(&cond);          // c5
Pthread_mutex_unlock(&mutex);        // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Correct Attempt!]

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);            // p1
while (count == 1)                     // p2
  Pthread_cond_wait(&empty, &mutex)// p3
put(i);                                // p4
Pthread_cond_signal(&fil);             // p5
Pthread_mutex_unlock(&mutex);          // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);            // c1
While (count == 0)                     // c2
  Pthread_cond_wait(&fil, &mutex).  // c3
int tmp = get();                       // c4
Pthread_cond_signal(&empty);           // c5
Pthread_mutex_unlock(&mutex);          // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Correct Attempt!]

- Producer threads wait on the condition empty, and signals fill.

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);            // p1
while (count == 1)                     // p2
  Pthread_cond_wait(&empty, &mutex)// p3
put(i);                                // p4
Pthread_cond_signal(&fill);            // p5
Pthread_mutex_unlock(&mutex);          // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);            // c1
While (count == 0)                     // c2
  Pthread_cond_wait(&fill, &mutex).  // c3
int tmp = get();                       // c4
Pthread_cond_signal(&empty);           // c5
Pthread_mutex_unlock(&mutex);          // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =1) [Correct Attempt!]

- Producer threads wait on the condition empty, and signals fill.
- Consumer threads wait on fill and signal empty.

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);            // p1
while (count == 1)                     // p2
  Pthread_cond_wait(&empty, &mutex)// p3
put(i);                                // p4
Pthread_cond_signal(&fill);            // p5
Pthread_mutex_unlock(&mutex);          // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);            // c1
While (count == 0)                     // c2
  Pthread_cond_wait(&fill, &mutex).  // c3
int tmp = get();                       // c4
Pthread_cond_signal(&empty);           // c5
Pthread_mutex_unlock(&mutex);          // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =n)

```
1   int buffer;
2   int count = 0;   // initially, empty
3
4   void put(int value) {
5   assert(count == 0);
6   count = 1;
7   buffer = value;
8   }
9
10  int get() {
11  assert(count == 1);
12  count = 0;
13  return buffer;
14  }
```

The Producer Consumer Problem (Buffer size =n)

```
1   int buffer;
2   int count = 0;   // initially, empty
3
4   void put(int value) {
5   assert(count == 0);
6   count = 1;
7   buffer = value;
8   }
9
10  int get() {
11  assert(count == 1);
12  count = 0;
13  return buffer;
14  }
```

Modify this code to define get and put on n sized buffer

The Producer Consumer Problem (Buffer size =n)

```
1   int buffer;
2   int count = 0;   // initially, empty
3
4   void put(int value) {
5   assert(count == 0);
6   count = 1;
7   buffer = value;
8   }
9
10  int get() {
11  assert(count == 1);
12  count = 0;
13  return buffer;
14  }
```

Modify this code to define get and put on n sized buffer

The Producer Consumer Problem (Buffer size =n)

```
1   int buffer[MAX];
2   int fill = 0;
3   int use = 0;
4   int count = 0;
5
6   void put(int value) {
7       buffer[fill] = value;
8       fill = (fill + 1) % MAX;
9       count++;
10  }
11
12  int get() {
13      int tmp = buffer[use];
14      use = (use + 1) % MAX;
15      count--;
16      return tmp;
17  }
```

The Producer Consumer Problem (Buffer size =n)

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);            // p1
while (count == 1)                     // p2
  Pthread_cond_wait(&empty, &mutex)// p3
put(i);                                // p4
Pthread_cond_signal(&fil);             // p5
Pthread_mutex_unlock(&mutex);          // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);            // c1
While (count == 0)                     // c2
  Pthread_cond_wait(&fil, &mutex).  // c3
int tmp = get();                       // c4
Pthread_cond_signal(&empty);           // c5
Pthread_mutex_unlock(&mutex);          // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =n)

- Modify this program to work for n sized buffer

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);            // p1
while (count == 1)                     // p2
  Pthread_cond_wait(&empty, &mutex)// p3
put(i);                                // p4
Pthread_cond_signal(&fil);             // p5
Pthread_mutex_unlock(&mutex);          // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);            // c1
While (count == 0)                     // c2
  Pthread_cond_wait(&fil, &mutex).  // c3
int tmp = get();                       // c4
Pthread_cond_signal(&empty);           // c5
Pthread_mutex_unlock(&mutex);          // c6
printf("%d\n", tmp);
}
}
```

The Producer Consumer Problem (Buffer size =n) [Correct Attempt!]

- Modify this program to work for n sized buffer

```
void *producer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);            // p1
while (count == N)                     // p2
  Pthread_cond_wait(&empty, &mutex)// p3
put(i);                                // p4
Pthread_cond_signal(&fil);             // p5
Pthread_mutex_unlock(&mutex);          // p6
}
}
```

```
void *consumer(void *arg) {

for (int i = 0; i < loops; i++) {
Pthread_mutex_lock(&mutex);            // c1
While (count == 0)                     // c2
  Pthread_cond_wait(&fil, &mutex).  // c3
int tmp = get();                       // c4
Pthread_cond_signal(&empty);           // c5
Pthread_mutex_unlock(&mutex);          // c6
printf("%d\n", tmp);
}
}
```

CV Rules of Thumb

CV Rules of Thumb

- Keep state in addition to CV

CV Rules of Thumb

- Keep state in addition to CV
- Always do wait/signal with the lock held

CV Rules of Thumb

- Keep state in addition to CV
- Always do wait/signal with the lock held
- Whenever you reacquire lock, check the state

CV Rules of Thumb

- Keep state in addition to CV
- Always do wait/signal with the lock held
- Whenever you reacquire lock, check the state

Design Tip

Design Tip

- If it is always recommended to use an abstraction the same way —>

Design Tip

- If it is always recommended to use an abstraction the same way —>
  - Build a better abstraction over the first one

Design Tip

- If it is always recommended to use an abstraction the same way —>
  - Build a better abstraction over the first one

Semaphore: Synchronisation Primitive

Semaphore: Synchronisation Primitive

- An object with an integer value

Semaphore: Synchronisation Primitive

- An object with an integer value
- We can manipulate with two routines; sem_wait() and sem_post().

Semaphore: Synchronisation Primitive

- An object with an integer value
- We can manipulate with two routines; sem_wait() and sem_post().

Semaphore: Synchronisation Primitive

- An object with an integer value
- We can manipulate with two routines; sem_wait() and sem_post().

```
1 #include <semaphore.h>
2 sem_t s;
3 sem_init(&s, 0, 1); // initialize s to the value 1
```

Semaphore: Synchronisation Primitive

- An object with an integer value
- We can manipulate with two routines; sem_wait() and sem_post().

```
1 #include <semaphore.h>
2 sem_t s;
3 sem_init(&s, 0, 1); // initialize s to the value 1
```


Semaphore is shared between threads of same process

Semaphore: Synchronisation Primitive

- An object with an integer value
- We can manipulate with two routines; sem_wait() and sem_post().

```
1 #include <semaphore.h>
2 sem_t s;
3 sem_init(&s, 0, 1); // initialize s to the value 1
```

Semaphore is shared between threads of same process

Initial value of semaphore

Semaphore: Synchronisation Primitive

Semaphore: Synchronisation Primitive

```
1  int sem_init(sem_t *s, int init_val) {
2    s->value=init_val;
3    }
```

Semaphore: Synchronisation Primitive

```
1  int sem_init(sem_t *s, int init_val) {
2    s->value=init_val;
3    }
```

```
1  int sem_wait(sem_t *s) {
2    s->value -= 1
3     wait if s->value <0
4  }
```

Semaphore: Synchronisation Primitive

```
1  int sem_init(sem_t *s, int init_val) {
2    s->value=init_val;
3    }
```

```
1  int sem_wait(sem_t *s) {
2    s->value -= 1
3     wait if s->value <0
4  }
```

```
1  int sem_post(sem_t *s) {
2     s->value += 1
3     wake one waiting thread if any
4  }
```

Semaphore: Synchronisation Primitive

```
1  int sem_init(sem_t *s, int init_val) {
2    s->value=init_val;
3    }
```

```
1  int sem_wait(sem_t *s) {
2    s->value -= 1
3     wait if s->value <0
4  }
```

sem_wait and sem_post are atomic

```
1  int sem_post(sem_t *s) {
2     s->value += 1
3     wake one waiting thread if any
4  }
```

Exercise: Build a lock using semaphores

Refresher Notes

Exercise: Build a lock using semaphores

Refresher Notes

```
1  int sem_wait(sem_t *s) {
2    s->value -= 1
3    wait if s->value <0
4  }
```

Exercise: Build a lock using semaphores

Refresher Notes

```
1  int sem_wait(sem_t *s) {
2    s->value -= 1
3    wait if s->value <0
4  }
```

```
1  int sem_post(sem_t *s) {
2    s->value += 1
3    wake one waiting thread if any
4  }
```

Exercise: Build a lock using semaphores

```
1  sem_t m;
2  sem_init(&m, 0, X);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Refresher Notes

```
1  int sem_wait(sem_t *s) {
2    s->value -= 1
3    wait if s->value <0
4  }
```

```
1  int sem_post(sem_t *s) {
2    s->value += 1
3    wake one waiting thread if any
4  }
```

Exercise: Build a lock using semaphores

```
1   sem_t m;
2   sem_init(&m, 0, 1);
3
4   sem_wait(&m);
5   //critical section here
6   sem_post(&m);
```

Refresher Notes

```
1  int sem_wait(sem_t *s) {
2    s->value -= 1
3    wait if s->value <0
4  }
```

```
1  int sem_post(sem_t *s) {
2    s->value += 1
3    wake one waiting thread if any
4  }
```

Thread Trace: Single Thread Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Thread Trace: Single Thread Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value of Semaphore	Thread 0	Thread 1

Thread Trace: Single Thread Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value of Semaphore	Thread 0	Thread 1
1		

Thread Trace: Single Thread Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value of Semaphore	Thread 0	Thread 1
1		
1	call sem_wait()	

Thread Trace: Single Thread Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value of Semaphore	Thread 0	Thread 1
1		
1	call sem_wait()	
0	sem_wait() returns	

Thread Trace: Single Thread Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value of Semaphore	Thread 0	Thread 1
1		
1	call sem_wait()	
0	sem_wait() returns	
0	(crit sect)	

Thread Trace: Single Thread Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value of Semaphore	Thread 0	Thread 1
1		
1	call sem_wait()	
0	sem_wait() returns	
0	(crit sect)	
0	call sem_post()	

Thread Trace: Single Thread Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value of Semaphore	Thread 0	Thread 1
1		
1	call sem_wait()	
0	sem_wait() returns	
0	(crit sect)	
0	call sem_post()	
1	sem_post() returns	

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0 State	Thread 1 State

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready
0	(crit set: begin)	Running		Ready

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready
0	(crit set: begin)	Running		Ready
0	Interrupt; Switch → T1	Ready		Running

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready
0	(crit set: begin)	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	call sem_wait()	Running

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready
0	(crit set: begin)	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	call sem_wait()	Running
-1		Ready	decrement sem	Running

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready
0	(crit set: begin)	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	call sem_wait()	Running
-1		Ready	decrement sem	Running
-1		Ready	(sem < 0)→sleep	sleeping

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready
0	(crit set: begin)	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	call sem_wait()	Running
-1		Ready	decrement sem	Running
-1		Ready	(sem < 0)→sleep	sleeping
-1		Running	Switch → T0	sleeping

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready
0	(crit set: begin)	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	call sem_wait()	Running
-1		Ready	decrement sem	Running
-1		Ready	(sem < 0)→sleep	sleeping
-1		Running	Switch → T0	sleeping
-1	(crit sect: end)	Running		sleeping

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready
0	(crit set: begin)	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	call sem_wait()	Running
-1		Ready	decrement sem	Running
-1		Ready	(sem < 0)→sleep	sleeping
-1		Running	Switch → T0	sleeping
-1	(crit sect: end)	Running		sleeping
-1	call sem_post()	Running		sleeping

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready
0	(crit set: begin)	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	call sem_wait()	Running
-1		Ready	decrement sem	Running
-1		Ready	(sem < 0)→sleep	sleeping
-1		Running	Switch → T0	sleeping
-1	(crit sect: end)	Running		sleeping
-1	call sem_post()	Running		sleeping
0	increment sem	Running		sleeping

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready
0	(crit set: begin)	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	call sem_wait()	Running
-1		Ready	decrement sem	Running
-1		Ready	(sem < 0)→sleep	sleeping
-1		Running	Switch → T0	sleeping
-1	(crit sect: end)	Running		sleeping
-1	call sem_post()	Running		sleeping
0	increment sem	Running		sleeping
0	wake(T1)	Running		Ready

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready
0	(crit set: begin)	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	call sem_wait()	Running
-1		Ready	decrement sem	Running
-1		Ready	(sem < 0)→sleep	sleeping
-1		Running	Switch → T0	sleeping
-1	(crit sect: end)	Running		sleeping
-1	call sem_post()	Running		sleeping
0	increment sem	Running		sleeping
0	wake(T1)	Running		Ready
0	sem_post() returns	Running		Ready

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready
0	(crit set: begin)	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	call sem_wait()	Running
-1		Ready	decrement sem	Running
-1		Ready	(sem < 0)→sleep	sleeping
-1		Running	Switch → T0	sleeping
-1	(crit sect: end)	Running		sleeping
-1	call sem_post()	Running		sleeping
0	increment sem	Running		sleeping
0	wake(T1)	Running		Ready
0	sem_post() returns	Running		Ready
0	Interrupt; Switch → T1	Ready		Running

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready
0	(crit set: begin)	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	call sem_wait()	Running
-1		Ready	decrement sem	Running
-1		Ready	(sem < 0)→sleep	sleeping
-1		Running	Switch → T0	sleeping
-1	(crit sect: end)	Running		sleeping
-1	call sem_post()	Running		sleeping
0	increment sem	Running		sleeping
0	wake(T1)	Running		Ready
0	sem_post() returns	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	sem_wait() retruns	Running

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready
0	(crit set: begin)	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	call sem_wait()	Running
-1		Ready	decrement sem	Running
-1		Ready	(sem < 0)→sleep	sleeping
-1		Running	Switch → T0	sleeping
-1	(crit sect: end)	Running		sleeping
-1	call sem_post()	Running		sleeping
0	increment sem	Running		sleeping
0	wake(T1)	Running		Ready
0	sem_post() returns	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	sem_wait() retruns	Running
0		Ready	(crit sect)	Running

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready
0	(crit set: begin)	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	call sem_wait()	Running
-1		Ready	decrement sem	Running
-1		Ready	(sem < 0)→sleep	sleeping
-1		Running	Switch → T0	sleeping
-1	(crit sect: end)	Running		sleeping
-1	call sem_post()	Running		sleeping
0	increment sem	Running		sleeping
0	wake(T1)	Running		Ready
0	sem_post() returns	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	sem_wait() retruns	Running
0		Ready	(crit sect)	Running
0		Ready	call sem_post()	Running

Thread Trace: Two Threads Using a Semaphore

```
1  sem_t m;
2  sem_init(&m, 0, 1);
3
4  sem_wait(&m);
5  //critical section here
6  sem_post(&m);
```

Value	Thread 0	State	Thread 1	State
1		Running		Ready
1	call sem_wait()	Running		Ready
0	sem_wait() retruns	Running		Ready
0	(crit set: begin)	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	call sem_wait()	Running
-1		Ready	decrement sem	Running
-1		Ready	(sem < 0)→sleep	sleeping
-1		Running	Switch → T0	sleeping
-1	(crit sect: end)	Running		sleeping
-1	call sem_post()	Running		sleeping
0	increment sem	Running		sleeping
0	wake(T1)	Running		Ready
0	sem_post() returns	Running		Ready
0	Interrupt; Switch → T1	Ready		Running
0		Ready	sem_wait() retruns	Running
0		Ready	(crit sect)	Running
0		Ready	call sem_post()	Running
1		Ready	sem_post() returns	Running

join() using CVs

```
1   void *child(void *arg) {
2       printf("child\n");
3       thread_exit()
4       return NULL; }

7   int main(int argc, char *argv[]) {
8       printf("parent: begin\n");
9       pthread_t c;
10        Pthread_create(&c, NULL, child, NULL); // create child
11        thread_join()
12        printf("parent: end\n");
13        return 0; }
```

join() using CVs

```
void thread_exit {
    mutex_lock(&m)
    Done  = 1
    cond_signal(&c)
    mutex_unlock(&m)
```

```
1   void *child(void *arg) {
2       printf("child\n");
3       thread_exit()
4       return NULL; }

7   int main(int argc, char *argv[]) {
8       printf("parent: begin\n");
9       pthread_t c;
10        Pthread_create(&c, NULL, child, NULL); // create child
11        thread_join()
12        printf("parent: end\n");
13        return 0; }
```

join() using CVs

```
void thread_exit {
    mutex_lock(&m)
    Done  = 1
    cond_signal(&c)
    mutex_unlock(&m)
```

```
void thread_join {
     mutex_lock(&m)              //w
    while (done==0)              //x
        cond_wait(&c, &m)        //y
    mutex_unlock(&m) }           //z
```

```
1   void *child(void *arg) {
2       printf("child\n");
3       thread_exit()
4       return NULL; }

7   int main(int argc, char *argv[]) {
8       printf("parent: begin\n");
9       pthread_t c;
10        Pthread_create(&c, NULL, child, NULL); // create child
11        thread_join()
12        printf("parent: end\n");
13        return 0; }
```

join() using semaphores

```
1  void *child(void *arg) {
2      printf("child\n");
3      thread_exit()
4      return NULL; }

7  int main(int argc, char *argv[]) {
8      printf("parent: begin\n");
9      pthread_t c;
10     sem_init(&s, 0, X);
11       Pthread_create(&c, NULL, child, NULL);
12       thread_join()
13       printf("parent: end\n");
14       return 0; }
```

Refresher Notes

```
1  int sem_wait(sem_t *s) {
2    s->value -= 1
3    wait if s->value <0
4  }
```

```
1  int sem_post(sem_t *s) {
2    s->value += 1
3    wake one waiting thread if any
4  }
```

join() using semaphores

```
void thread_exit {

}
```

```
1  void *child(void *arg) {
2      printf("child\n");
3      thread_exit()
4      return NULL; }

7  int main(int argc, char *argv[]) {
8      printf("parent: begin\n");
9      pthread_t c;
10     sem_init(&s, 0, X);
11       Pthread_create(&c, NULL, child, NULL);
12       thread_join()
13       printf("parent: end\n");
14       return 0; }
```

Refresher Notes

```
1  int sem_wait(sem_t *s) {
2    s->value -= 1
3    wait if s->value <0
4  }

1  int sem_post(sem_t *s) {
2    s->value += 1
3    wake one waiting thread if any
4  }
```

join() using semaphores

```
void thread_exit {

}
```

```
void thread_join {

}
```

```
1  void *child(void *arg) {
2      printf("child\n");
3      thread_exit()
4      return NULL; }

7  int main(int argc, char *argv[]) {
8      printf("parent: begin\n");
9      pthread_t c;
10     sem_init(&s, 0, X);
11       Pthread_create(&c, NULL, child, NULL);
12       thread_join()
13       printf("parent: end\n");
14       return 0; }
```

Refresher Notes

```
1  int sem_wait(sem_t *s) {
2    s->value -= 1
3    wait if s->value <0
4  }
```

```
1  int sem_post(sem_t *s) {
2    s->value += 1
3    wake one waiting thread if any
4  }
```