Operating Systems
Condition Variables Advanced

Nipun Batra

Concurrency Objectives

Concurrency Objectives

- Mutual exclusion

Concurrency Objectives

- Mutual exclusion
  - A & B don’t run at the same time

Concurrency Objectives

- Mutual exclusion
  - A & B don’t run at the same time
  - Solved using locks

Concurrency Objectives

- Mutual exclusion
  - A & B don’t run at the same time
  - Solved using locks
- Ordering

Concurrency Objectives

- Mutual exclusion
  - A & B don’t run at the same time
  - Solved using locks
- Ordering
  - A runs after B

Concurrency Objectives

- Mutual exclusion
  - A & B don’t run at the same time
  - Solved using locks
- Ordering
  - A runs after B
- Solved with?

Concurrency Objectives

- Mutual exclusion
  - A & B don’t run at the same time
  - Solved using locks
- Ordering
  - A runs after B
  - Solved with?
    - Join

Concurrency Objectives

- Mutual exclusion
  - A & B don’t run at the same time
  - Solved using locks
- Ordering
  - A runs after B
  - Solved with?
    - Join
      - Implemented using condition variables

Condition Variables

Condition Variables

- **Condition variable**

Condition Variables

- **Condition variable**
  - Queue of sleeping threads

Condition Variables

- **Condition variable**
  - Queue of sleeping threads
  - Thread add themselves to queue with **wait**

Condition Variables

- **Condition variable**
  - Queue of sleeping threads
  - Thread add themselves to queue with **wait**
  - Thread wake up threads on the queue with **signal**

Condition Variables

- **Condition variable**
  - Queue of sleeping threads
  - Thread add themselves to queue with **wait**
  - Thread wake up threads on the queue with **signal**

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

Exercise: order using condition variables
Write thread_exit() and thread_join() using CVs

- **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

```
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; }
```

Rule of Thumb #1

Rule of Thumb #1

- In addition to condition variables use another variable to capture state

Rule of Thumb #1

- In addition to condition variables use another variable to capture state
- CVs can be used to nudge threads when state changes

Exercise: order using condition variables Correct Solution

```
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; }
```

Exercise: order using condition variables Correct Solution

```
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; }
```

Exercise: order using condition variables Correct Solution

```
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; }
```

Rule of Thumb #2

Rule of Thumb #2

- Wait and signal while holding the lock

The Producer Consumer Problem

The Producer Consumer Problem

- Producers produce data and place it on a shared resource

The Producer Consumer Problem

- Producers produce data and place it on a shared resource
- Example:

The Producer Consumer Problem

- Producers produce data and place it on a shared resource
- Example:
  - Multi-threaded web server:

The Producer Consumer Problem

- Producers produce data and place it on a shared resource
- Example:
  - Multi-threaded web server:
    - Multiple request coming in concurrently - Producers

The Producer Consumer Problem

- Producers produce data and place it on a shared resource
- Example:
  - Multi-threaded web server:
    - Multiple request coming in concurrently - Producers
  - Multiple responses concurrently - Consumers

The Producer Consumer Problem

- Producers produce data and place it on a shared resource
- Example:
  - Multi-threaded web server:
    - Multiple request coming in concurrently - Producers
    - Multiple responses concurrently - Consumers
- Bounded buffer

The Producer Consumer Problem

- Producers produce data and place it on a shared resource
- Example:
  - Multi-threaded web server:
    - Multiple request coming in concurrently - Producers
    - Multiple responses concurrently - Consumers
  - Bounded buffer
    - grep foo file.txt | wc –l

The Producer Consumer Problem

- Producers produce data and place it on a shared resource
- Example:
  - Multi-threaded web server:
    - Multiple request coming in concurrently - Producers
    - Multiple responses concurrently - Consumers
  - Bounded buffer
    - grep foo file.txt | wc –l
    - The grep process is the producer.

The Producer Consumer Problem

- Producers produce data and place it on a shared resource
- Example:
  - Multi-threaded web server:
    - Multiple request coming in concurrently - Producers
    - Multiple responses concurrently - Consumers
  - Bounded buffer
    - grep foo file.txt | wc –l
    - The grep process is the producer.
    - The wc process is the consumer.

The Producer Consumer Problem

- Producers produce data and place it on a shared resource
- Example:
  - Multi-threaded web server:
    - Multiple request coming in concurrently - Producers
    - Multiple responses concurrently - Consumers
  - Bounded buffer
    - grep foo file.txt | wc –l
    - The grep process is the producer.
    - The wc process is the consumer.
    - Between them is an in-kernel bounded buffer.

The Producer Consumer Problem

- Producers produce data and place it on a shared resource
- Example:
  - Multi-threaded web server:
    - Multiple request coming in concurrently - Producers
    - Multiple responses concurrently - Consumers
- Bounded buffer
  - grep foo file.txt | wc –l
  - The grep process is the producer.
  - The wc process is the consumer.
  - Between them is an in-kernel bounded buffer.

The Producer Consumer Problem

Bounded Buffer

The Producer Consumer Problem

Bounded Buffer

The Producer Consumer Problem

Producer adds to the buffer

Bounded Buffer

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]

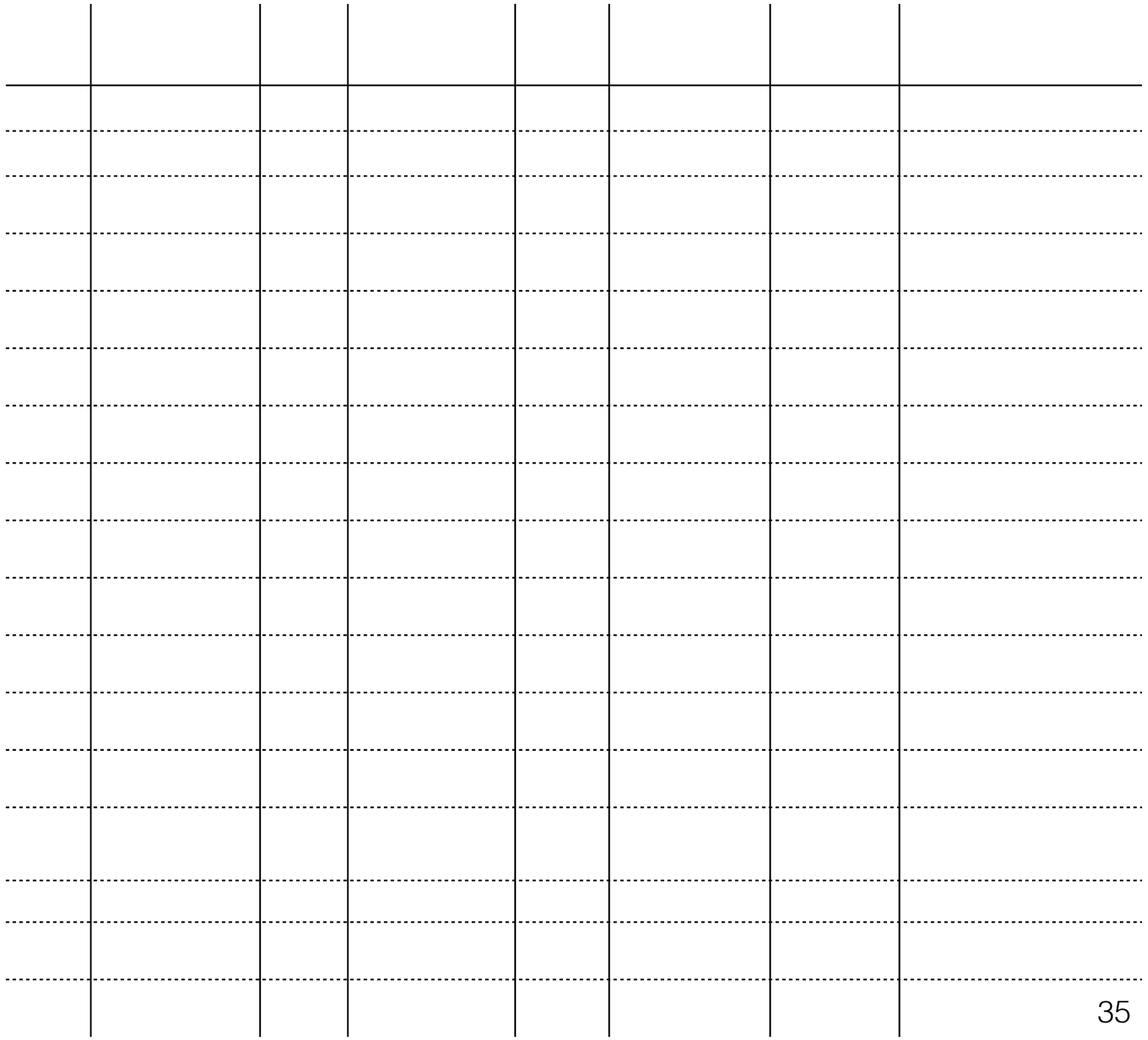
```
Pthread_mutex_lock(&mutex);             // p1
if (count == 1)                         // p2
    Pthread_cond_wait(&cond, &mutex)// p3
put(i);                                 // p4
Pthread_cond_signal(&cond);             // p5
Pthread_mutex_unlock(&mutex);           // p6
```

```
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);
}
}
```