Q1

- Why can other threads still acquire the same mutex lock and enter the critical section when the current thread is in pthread_cond_wait()?
- Ans: When the thread is in pthread_cond_wait(), it will enter a waiting state and release the mutex lock, which allows other threads to acquire the same mutex lock and enter the critical section.

Q2

- What is the difference between the following implementations of wait() ?

```
1  void wait() {
2      pthread_mutex_lock(&mutex);
3
4      while (value <= 0) {
5          // block
6          pthread_cond_wait(&cond, &mutex);
7      }
8      --value;
9
10     pthread_mutex_unlock(&mutex);
11 }
```

while statement

```
1  void wait() {
2      pthread_mutex_lock(&mutex);
3
4      if (value <= 0) {
5          // block
6          pthread_cond_wait(&cond, &mutex);
7      }
8      --value;
9
10     pthread_mutex_unlock(&mutex);
11 }
```

if statement

Q2 (Answer)

- While statement is the correct implementation
- Key point
  - Pthread_cond_wait() will release the mutex lock when the thread go to wait state, hence it need to regain the mutex lock to execute the following program when other threads call pthread_cond_signal()
- Consider the following scenario
  - Thread A is in pthread_cond_wait()
  - Thread B calls pthread_cond_signal()
  - Thread A tries to get the mutex lock, but thread C gets the mutex lock first and enters the critical section, which causes the value--
  - Thread A gets the mutex lock, leave the pthread_cond_wait(), but at that moment value is still <= 0 (error)
  - It’s necessary to check the value again when leave pthread_cond_wait()