1- در این برنامه تمامی شرایط به غیر از starvation برقرار است

در عین واحد متغیر turn میتواند یک مقدار داشته باشد به همین علت دو برنامه نمیتوانند همزمان وارد محیط بحرانی شوند

اما، ممکن است شرایطی پیش بیاید که سیستم زمانی که نوبت را به یک برنامه میدهد، در حالت قفل باشد و نتواند ادامه دهد، و زمانی که نوبت برنامه دیگر میشود، اجرا میشود، قفل را باز میکند و دوباره قفل میکند، به این صورت حالتی پیش میاید که یک برنامه اصلا از حلقه قفل رد نشود

-2

```
lock(m);
guest_count++;
if (guest_count >= N)
        signal(cv_host);
wait(cv_guest,m);
unlock(m);
enterHouse();
```

-3

```
mutex queue_mutex;
mutex mutex;
conditional_variable rw_var;
conditional_variable read_var;

writerLock(){
        lock(queue_mutex)
        writerQueue++;
        unlock(queue_mutex);
        lock(mutex);
        while(readerCount>0)
                wait(rw_var,mutex);
}
```

```
writerUnlock(){
        unlock(mutex);
        lock(queue_mutex)
        writerQueue--;
        if(writerQueue == 0)
                signal(read_var);
        unlock(queue_mutex);
}
readerLock(){
        lock(mutex);
        while(writerQueue>0)
                wait(read_var,mutex);
        readerCount++;
        unlock(mutex);
}
readerUnlock(){
        lock(mutex);
        readerCount--;
        if(readerCount==0)
                signal(rw_var);
        unlock(mutex);
}
```

-4

-5

QMutex کلاس پایه برای ممانعت متقابل است، یک ترد هنگام دسترسی به یک منبع مشترک mutex را قفل میکند، اگر ترد دیگری سعی به قفل کردن آن کند در حال که از قبل قفل است، وارد sleep میشود تا زمانی که ترد قبلی قفل را باز کند

QReadWriteLock مشابه QMutex است، با این تفاوت که دسترسی خواندن و نوشتن را متمایز میکند، اگر اطلاعاتی مشغول نوشتن نباشد، میتوان در چند نخ آن را به صورت همزمان خواند، این کلاس این قابلیت را به ما میدهد و موازی سازی را بهبود میبخشد

کلاس QSemaphore یک تعمیم از QMutex است که اجازه چند دسترسی همزمان به یک منبع مشترک را میدهد، بر خلاف QMutex که فقط اجازه یک دسترسی میدهد

QWaitCondition قابلیت همزمانی با استفاده از condition variable را میدهد، در حالی که دیگر کلاس ها صبر میکنند تا یک منبعی آزاد شود، این کلاس صبر میکند تا یک شرط برقرار شود، برای

بیدار کردن نخ ها در این حالت میتوان از wakeOne (یک ترد تصادفی) یا wakeAll (همه ترد های خواب) استفاده کرد

برای پیاده سازی مانیتور میتوان هم از QMutex و QMutexLocker استفاده کرد

به این صورت که data member های یک کلاس به صورت private تعریف میشوند و برای دسترسی به آن ها تابع هایی تعریف میشوند، سپس در اجرای تابع ها از mutex ها استفاده میشود

کلاس QMutexLocker در constructor خود تابع mutex lock و در destructor خود تابع unlock آن را صدا میزند