سوال ۶ | اول کار turn = 0 است. و ۲ تا بلاک های false هستند.

P1

```
blocked[1] = true

while(turn != 1){
    while(blocked[0]);
    turn = 1;
}

//critical

blocked[1] = false

//remainder
```

P0

```
blocked[0] = true

while(turn != 0){
    while(blocked[1]);
    turn = 0;
}
//critical
blocked[0] = false

//remainder
```

گزاره اول رد می‌شود | و گرسنگی هم داره.

گزاره ۱ و mutual exclusion یعنی فقط یک پردازه در هر لحظه توی ناحیه‌ی بحرانی باشد.

سوال ۱

گزاره ۱ و بررسی mutual exclusion ← یعنی نباید هر دو پردازه هم زمان در ناحیه بحرانی باشند ← نقض می‌شود چون اگه P0 اول اجرا شه (که می‌شه) ← ناحیه بحرانی را اجرا می‌کنه و می‌کند حالا اگه P1 اجرا بشه چون turn=0 است ← میره توی while() ← چون blocked[0] = false ← از روی while هم میاد بیرون ← میره سر خط turn=1 ← همین جا قبل از اینکه turn عوض شه و CPU ازش گرفته می‌شه ← P0 دوباره اجرا می‌شه ← چون turn=0 است هنوز ← وارد ناحیه بحرانی می‌شود ← همین جا دوباره CPU گرفته می‌شود به P1 داده می‌شود که سر خط turn=1 بود ← P1 هم وارد ناحیه بحرانی می‌شود ← هر دو رفتن تو ناحیه بحرانی

گزاره ۲ و چک می‌کنیم که وقتی ناحیه بحرانی آزاد است، فرآیندی نباشد که یک پردازه هیچ وقت نتواند وارد ناحیه بحرانی شود)

چون اینجا از یک flag مثل blocked استفاده می‌کنیم و هر زمان ناحیه بحرانی خالی باشد یکی از پردازه‌ها می‌تواند وارد بشه اگر نوبتش رعایت نشده باشد ← گزاره progress درست است

گزاره ۳ و چک کردن بن بست و چون turn داریم و یا turn=0 یا turn=1 است حالتی پیش نمیاد که هر دو پردازه و پشت ناحیه بحرانی گیر کنند ← بن بست نداریم

گزاره ۴ گرسنگی و آیا حالتی پیش میاد که یک پردازه مدام بره توی ناحیه‌ی بحرانی اما یک پردازه هیچ وقت نتواند و وقتی ناحیه‌ی بحرانی توی P0 کامل اجرا شود و وقتی به خط blocked[0] = false می‌رسیم و قبلش باشیم CPU را بدیم به P1 ← P1 میاد سر خط while(blocked[0]) و چون این خط هم چنان true است و برای خط گیر می‌کند

ادامه‌ی سوال ۱

چون blocked[i] هم‌زمان true است و ما هرچقدر صبر به P1 و CPU بدهیم تا این شرط را خراب کند باز هم چون خط blocked[i] = false و اجرا نشده هنوز و P1 نمی‌تواند وارد ناحیه بحرانی‌اش شود ← CPU از P1 گرفته می‌شود و دوباره به P0 داده می‌شود ← اینجا blocked[i] = false می‌شود و ممکن است دوباره P0 برود سر خط اول و بخواهد وارد ناحیه بحرانی‌اش شود ← اگر کل ماجرای قبلی دوباره تکرار شود ← P1 هیچ وقت نمی‌تواند وارد ناحیه‌ی بحرانی‌اش شود ← دچار گرسنگی می‌شود ← این شرط نقض می‌شود

سوال ۲

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

که مهمان

وقتی تعداد مهمان‌ها N شد ← یک signal به host میدهد که از while(count<N) بیدار شود و openDoor() را فراخوانی کند و سپس host باید cv_guest را signal کند تا مهمان اجازه ورود پیدا کند و بتواند از خواب wait(cv_guest) بیدار شود ←

وقتی که تعداد مهمان‌ها کمتر از N است ← هر مهمان یا هر ترد باید منتظر بماند ← همه مهمان‌ها تا قبل از آمدن آخرین مهمان پشت خط wait(cv_guest, m) باید منتظر بمانند

وقتی که آخرین مهمان میاید ← در تابع host بعد از فراخوانی openDoor، cv_guest را signal میکند ← یکی از تردهای مهمان (معمولا آخرین مهمان) آزاد می‌شود و میرسد سر خط signal(cv_guest) و یک ترد دیگر که مهمان بوده و منتظر آزاد شدن cv_guest بوده را آزاد می‌کند و ترد بعدی هم میاید signal را برای cv_guest فراخوانی می‌کند ← تا آخرین مهمان ← یکی یکی مهمان‌ها وارد خانه می‌شوند

سوال ۳

مسئله‌ی reader-writer با اولویت دهی به writer

```
int num_reads; // in progress
int num_writes; // waiting or in progress
pthread_cond_t reader_cv;
   "      "    writer_cv;
pthread_mutex_t lock;

read_lock(){
    pthread_mutex_lock(lock);
        while(num_writers > 0){
            cond_wait(&reader_cv, &lock);
        }
    num_reads++;
    mutex_unlock(lock);
}

read_unlock(){
    mutex_lock(&lock);
    num_reads--;
    if(num_writes > 0){
        cond_signal(&writer_cv);
    } else
        cond_broadcast(&reader_cv)
    mutex_unlock(&lock);
}
```

(یعنی تنها اگه دیگه reader ای نداریم، به writer سیگنال بده تا بتونه وارد بشه و بنویسه)

۱ → به جای این که ۲ خط بنویسیم:

```
if(num_reads == 0){
    cond_signal(&writer_cv);
}
```

ادامه سوال ۳

```
write_lock(){
    mutex_lock(&lock);
    num_writes++;
  while(num_writes > 1 || num_reads > 0)
        cond_wait(&writer_cv, &lock);
    mutex_unlock(&lock);
}
```

تعداد writer بیش از ۱ باشد

```
write_unlock(){
    mutex_lock(&lock);
    num_writes--;
    if(num_writes > 0)
        cond_signal(&writer_cv);
    else
        cond_broadcast(&reader_cv);
    mutex_unlock(&lock);
}
```

چون اولویت با writer است اگر writer داشتیم اولویت با او باشد

سوال ٤

```
semaphore  mutex_north = 1;
    ~      mutex_south = 1;
int   num_north = 0 , num_south = 0;

semaphore  max_car = 5;      // ماکس تعداد ماشین‌ها در پل
semaphore  bridge = 1;       // → هر وقت پل در دسترس است

north(){                     // ← تابع برای شمال
    wait(mutex_north);
        num_north ++;
        if(num_north == 1)
            wait(bridge);
    signal(mutex_north);
    wait(max_car);

    // با آمدن پل برای ماشین

    wait(mutex_north);
        signal(max_car);
        num_north --;
        if (num_north == 0)
            signal(bridge);
    signal(mutex_north);
}
```

```
south(){                     // تابع برای جنوب
    wait(mutex_south);
    num_south ++;
        if(num_south == 1)
            wait(bridge);
    signal(mutex_south);
    wait(max_car);

    // با آمدن پل برای

    wait(mutex_south);
        signal(max_car);
        num_south --;
        if(num_south == 0)
            signal(bridge);
    signal(mutex_south);
}
```

سوال ک

روش های synchronization در Qt:

روش 1: **Qmutex** یک basic class است برای اجرای mutual exclusion

یک thread، یک mutex را lock می کند تا به یک منبع مشترک دسترسی پیدا کند.

اگر یک thread دیگری سعی کند که mutex را lock کند درحالی که قبلاً lock شده:

thread دوم به حالت sleep میره تا وقتی که ترد اول، کارش را تمام کند و mutex را unlock کند.

```
Qmutex mutex;
mutex.lock();
// critical section
mutex.unlock();
```

monitor ها، کلاس هایی هستند که data member هاشون private هستند و member function هاشون هم به همراه mutual exclusion هستند ←

مثال برای Qmutex:

یک monitor تعریف کردیم

```
class BankAccount {
public:
    BankAccount() { balance = 0; }
    void withdraw(int amount) {
        mutex.lock();
        balance -= amount;
        mutex.unlock();
    }
    void deposit(int amount) {
        mutex.lock();
        balance += amount;
    }
private:
    Qmutex mutex;
    int balance;
};
```

private →

ادامه سوال ۵ | به کمک Qmutex و method های مانیتور و به صورت mutual exclusive

اجرا می‌شوند. یعنی وقتی مقداری از حساب می‌خواهد کم شود

توسط یک thread و thread های دیگر نمی‌توانند مقدار balance را تغییر دهد.

روش ۲

wait condition

QwaitCondition ← این روش و thread ها را سنکرون می‌کند نه به کمک mutual exclusion

بلکه به کمک یک condition value . یعنی thread ها را مجبور می‌کند

که منتظر اجرای یک شرط خاص شوند.

برای ادامه دادن اون thread هایی که sleep شدند ← از QwakeOne() یا از

QwakeAll() برای wake کردن همه‌ی ترد ها استفاده می‌کنیم

QwaitCondition() هم و باعث می‌شه ترد ها منتظر بمانند تا یک شرط خاص برقرار شود

به طور مثال و استفاده از کلاس QwaitCondition() در مسئله producer - consumer

```
QwaitCondition   notEmpty;
  ~        ~     notFull;
Qmutex  mutex;
```

کد producer

```
run() {
    for(int i=0; i<DataSize; i++) {
        mutex.lock();
        if(numUsedBytes == BufferSize)
            notFull.wait(&mutex);
        mutex.unlock();

        // ~  ~  produce
        mutex.lock();
        numUsedBytes++;
        notEmpty.wakeAll();
        mutex.unlock();
```

اگر بافر پر باشد باید صبر کنیم تا خالی شود

وقتی یک داده تولید شد و سیگنال می‌دهیم به

مصرف کننده (consumer) تا اگر خواست و consume کند

پیاده‌سازی مانیتور در wait Condition :

```
class BankAccount {
public:
    BankAccount() { balance = 0; }
    void withdraw(int amount) {
        mutex.lock();
        while (amount > balance)
            moremoney.wait(&mutex);
        balance -= amount;
        mutex.unlock();
    }
    void deposit(int amount) {
        mutex.lock();
        balance += amount;
        moremoney.wakeAll();
        mutex.unlock();
    }
private:
    Qmutex mutex;
    QwaitCondition moremoney;
    int balance;
};
```

← اگر اون مقداری که قراره از حساب کم بشه داری بیشتر از خود حساب باشه ←
باید بره تو حالت wait و صبر کنه تا مقداری که توی حسابه زیاد بشه
بعدش wake بشه و از حساب کم کنه

→ میگه می‌تونی الان از حساب چک کنی (البته اگر موجودی داشتی)

← به عنوان private تعریف می‌کنیم.

Condition variable