83 基于信号量的线程同步

83.1 资源计数

信号量是一种在较多线程瓜分较少资源时使用的停——等机制。一个信号量可以理解为表示某种资源份数的计数器。一个线程在获取一定份数的资源之前，会先尝试对信号量执行相应数值的减法操作。够减说明资源充足，线程可以获得并使用这些资源；不够减说明资源匮乏，线程应放弃对资源的获取，并进入等待状态。得到资源的线程，一旦完成对资源的使用，必须释放所持有的资源，即对信号量执行相应数值的加法操作。这时那些正在等待资源的线程将获得通知，并在对信号量执行减法成功后，拥有并使用相应份数的资源。

83.2

在Qt中，通过QSemaphore类表示信号量。该类提供如下方法：

```
void QSemaphore::acquire(int n = 1);             // 申请n份资源，阻塞版
bool QSemaphore::tryAcquire(int n = 1);          // 申请n份资源，非阻塞版
bool QSemaphore::tryAcquire(int n, int timeout); // 申请n份资源，超时版
void QSemaphore::release(int n = 1);             // 释放n份资源
int  QSemaphore::available() const;              // 获取资源的可用份数
```

83.4 案例

83.4.1 创建项目

通过QtCreator，在C:\Users\Minwei\Projects\Qt路径下，创建名为Semaphore的控制台（Console）项目。

83.4.2 实现功能

C:\Users\Minwei\Projects\Qt\Semaphore\main.cpp：

```
#include <iostream>
using namespace std;

#include <QCoreApplication>
#include <QThread>
#include <QMutex>
#include <QSemaphore>
#include <QRandomGenerator>

#define SEAT_COUNT 3
#define PASSENGER_COUNT 9

QMutex g_mutex;
QSemaphore g_seats(SEAT_COUNT);

class Passenger: public QThread
{
public:
    Passenger(QString name)
        : m_name(name)
        , m_rand(time(NULL))
```

```
    {
    }

protected:
    void run()
    {
        g_mutex.lock();
        cout << m_name.toStdString() << "> STANDING" << endl;
        g_mutex.unlock();

        g_seats.acquire();

        g_mutex.lock();
        cout << m_name.toStdString() << "> SIT DOWN" << endl;
        g_mutex.unlock();

        msleep(m_rand.bounded(1, 101));

        g_mutex.lock();
        cout << m_name.toStdString() << "> GET OFF" << endl;
        g_mutex.unlock();

        g_seats.release();
    }

private:
    QString m_name;
    QRandomGenerator m_rand;
};

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    cout << g_seats.available() << endl;

    Passenger* passengers[PASSENGER_COUNT];

    for (int i = 0; i < PASSENGER_COUNT; ++i)
    {
        passengers[i] = new Passenger(QString("PASSENGER-%1").arg(i + 1));
        passengers[i]->start();
    }

    for (int i = 0; i < PASSENGER_COUNT; ++i)
    {
        passengers[i]->wait();
        delete passengers[i];
    }

    cout << g_seats.available() << endl;

    return a.exec();
}

```

83.4.3 测试验证

运行效果如图所示：

```
3
PASSENGER-1> STANDING
PASSENGER-2> STANDING
PASSENGER-3> STANDING
PASSENGER-4> STANDING
PASSENGER-6> STANDING
PASSENGER-1> SIT DOWN
PASSENGER-5> STANDING
PASSENGER-7> STANDING
PASSENGER-8> STANDING
PASSENGER-9> STANDING
PASSENGER-2> SIT DOWN
PASSENGER-3> SIT DOWN
PASSENGER-1> GET OFF
PASSENGER-4> SIT DOWN
PASSENGER-3> GET OFF
PASSENGER-2> GET OFF
PASSENGER-5> SIT DOWN
PASSENGER-6> SIT DOWN
PASSENGER-4> GET OFF
PASSENGER-8> SIT DOWN
PASSENGER-5> GET OFF
PASSENGER-6> GET OFF
PASSENGER-7> SIT DOWN
PASSENGER-9> SIT DOWN
PASSENGER-8> GET OFF
PASSENGER-7> GET OFF
PASSENGER-9> GET OFF
3
```