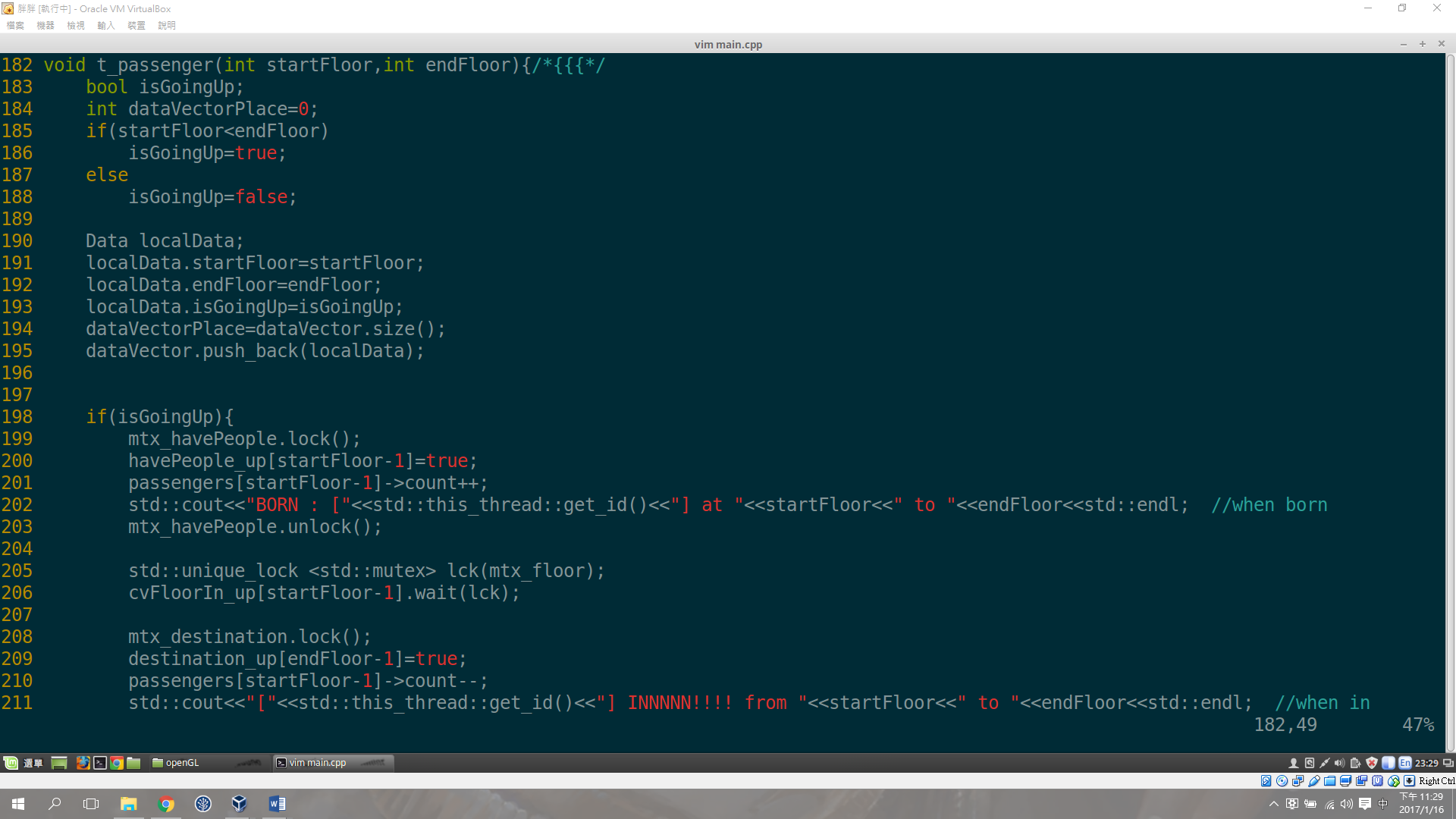
1. 執行緒 ( Threads ) 類型需求與各自功能:

依照題目要求，我們將要搭乘電梯的乘客與電梯分別做成執行緒

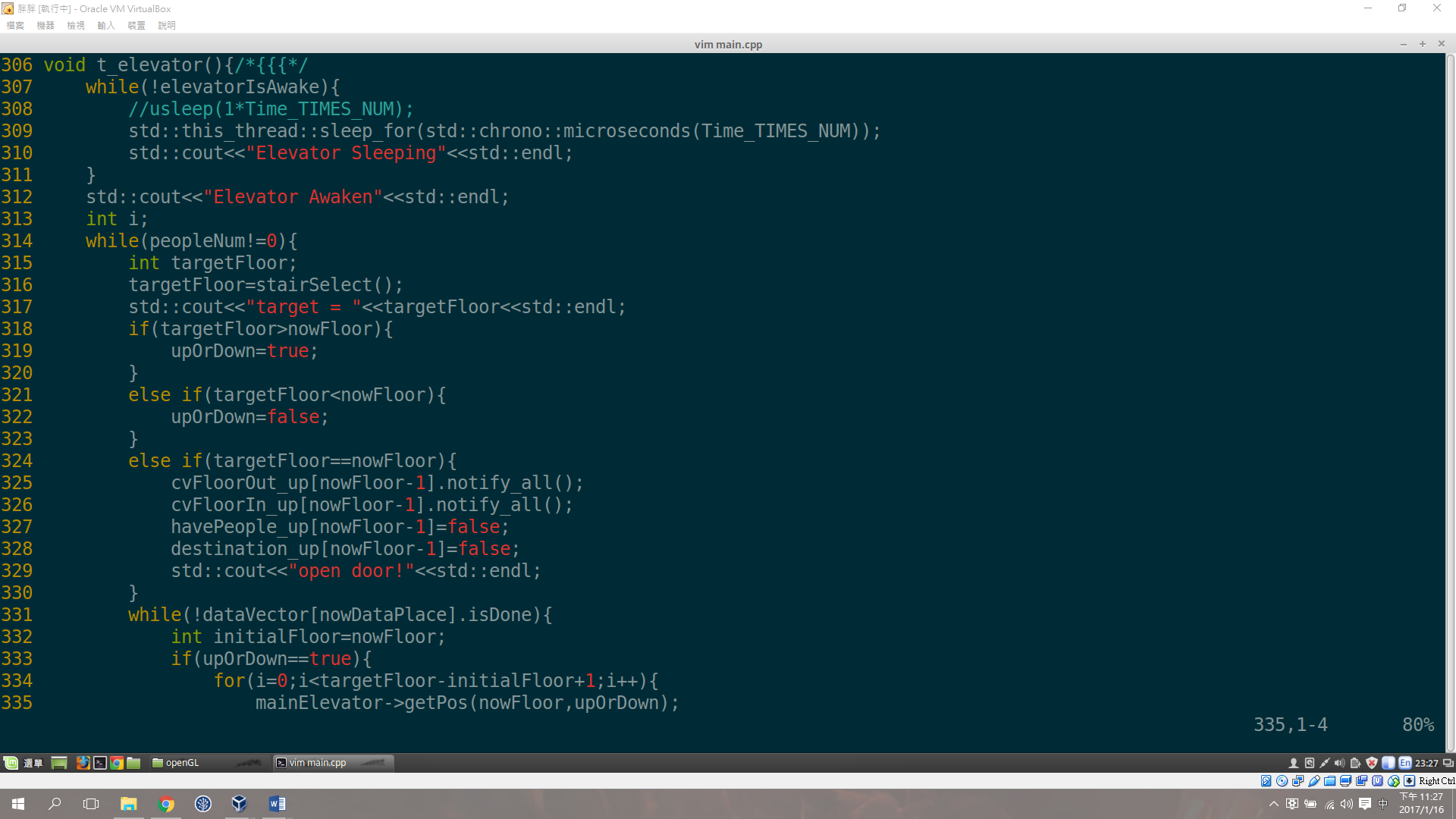
1. 乘客的執行緒(void t\_passenger(int startFloor,int endFloor)):

每個隨機產生出來的乘客，都各自擁有獨立的thread，裡面做的事情依序為:按電梯(上樓或下樓)->等電梯->進電梯並按下欲前往之樓層->等待電梯到達目的地->出電梯，當此執行緒被判定為出電梯之後，就結束。

(此圖並非所有程式碼)

1. 電梯的執行緒(void t\_elevator()):

此執行緒是用來模擬電梯的移動透過Shared Variables來取得資訊並移動，而電梯會有上樓、下樓，或者是沒人按時停止不動，此執行緒並不會因為沒有乘客而結束，而是繼續等待下一個隨機產生出來的乘客。

(此圖並非所有程式碼)

1. 共有資源與共享變數(Shared Variables)需要進行Multi-Thread (or -Process)之間的同步與合作:
2. Shared Variables:

bool havePeople\_up[NumberOfFloors];

有要上樓的乘客的樓層

bool destination\_up[NumberOfFloors];

上樓乘客要去的樓層

bool havePeople\_down[NumberOfFloors];

有要下樓的乘客的樓層

bool destination\_down[NumberOfFloors];

下樓乘客要去的樓層

以上的Shared Variables是為了要讓電梯與乘客去做溝通，例如:乘客在五樓想要去七樓，那麼havePeople\_up[4]就會被改成true，進電梯後，因為已經載到人了，電梯執行緒又會把它改成false，然後乘客執行緒就會將destination\_up[6]改成true，電梯將人載到目的地後，再改成false，以此類推。

1. Mutex:

std::mutex mtx\_havePeople;

為了不要讓havePeople\_up[NumberOfFloors]和havePeople\_down[NumberOfFloors]被乘客與電梯同時修改。

std::mutex mtx\_destination;

為了不要讓destination\_up[NumberOfFloors]和destination\_down[NumberOfFloors]被乘客與電梯同時修改。

std::mutex mtx\_floor;

與condition variable配合使用

1. Condition Variable:

std::condition\_variable cvFloorIn\_up[NumberOfFloors];

notify\_all()在N樓等上樓電梯的乘客進電梯

std::condition\_variable cvFloorOut\_up[NumberOfFloors];

notify\_all()要上到N樓的乘客下電梯

std::condition\_variable cvFloorIn\_down[NumberOfFloors];

notify\_all()在N樓等下樓電梯的乘客進電梯

std::condition\_variable cvFloorOut\_down[NumberOfFloors];

notify\_all()要下到N樓的乘客下電梯

1. Multi-Thread:

電梯先等待乘客，一有乘客之後，就取得data並且前往載客，並用Condition Variable來告訴乘客何時該進出電梯。