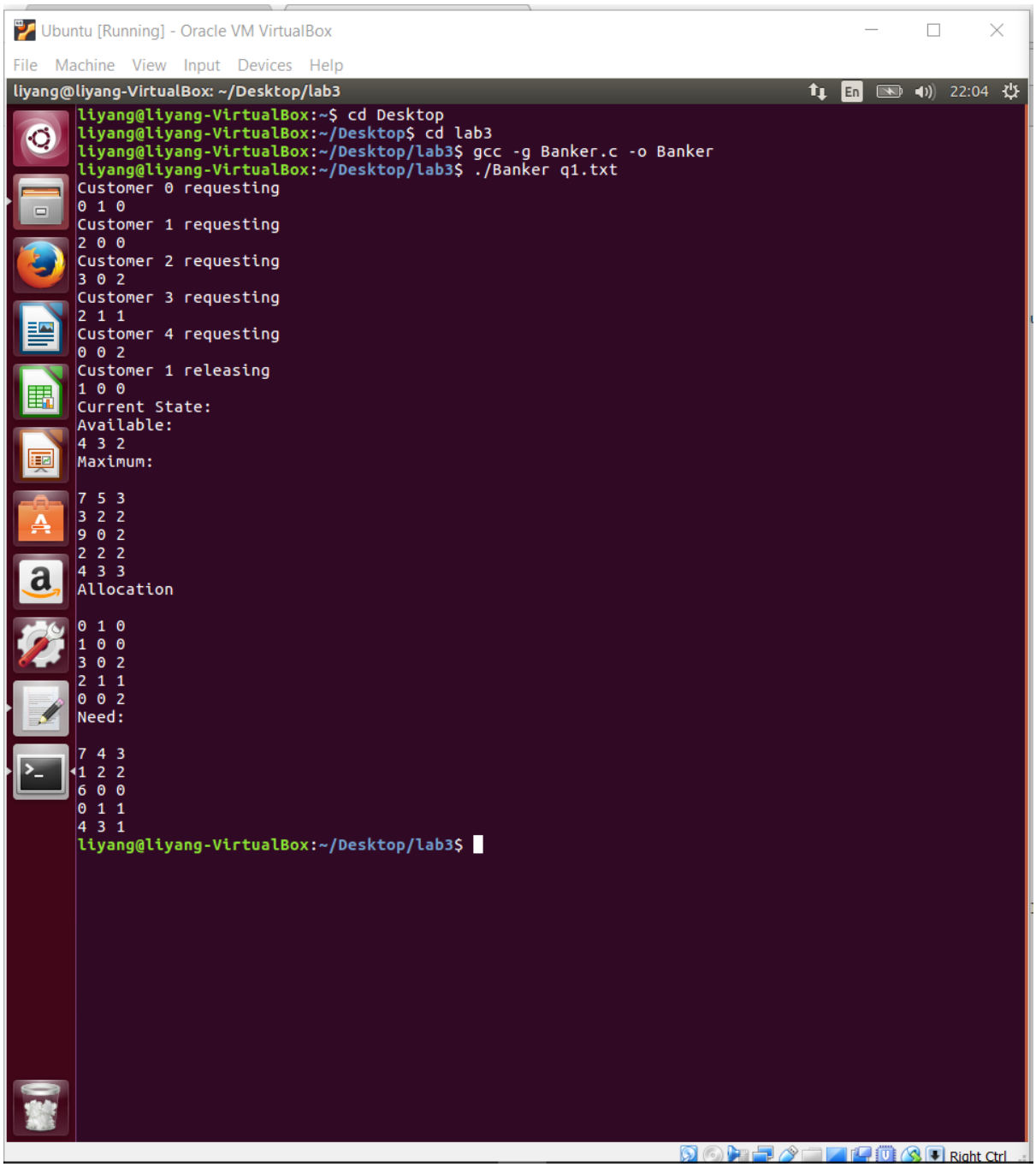
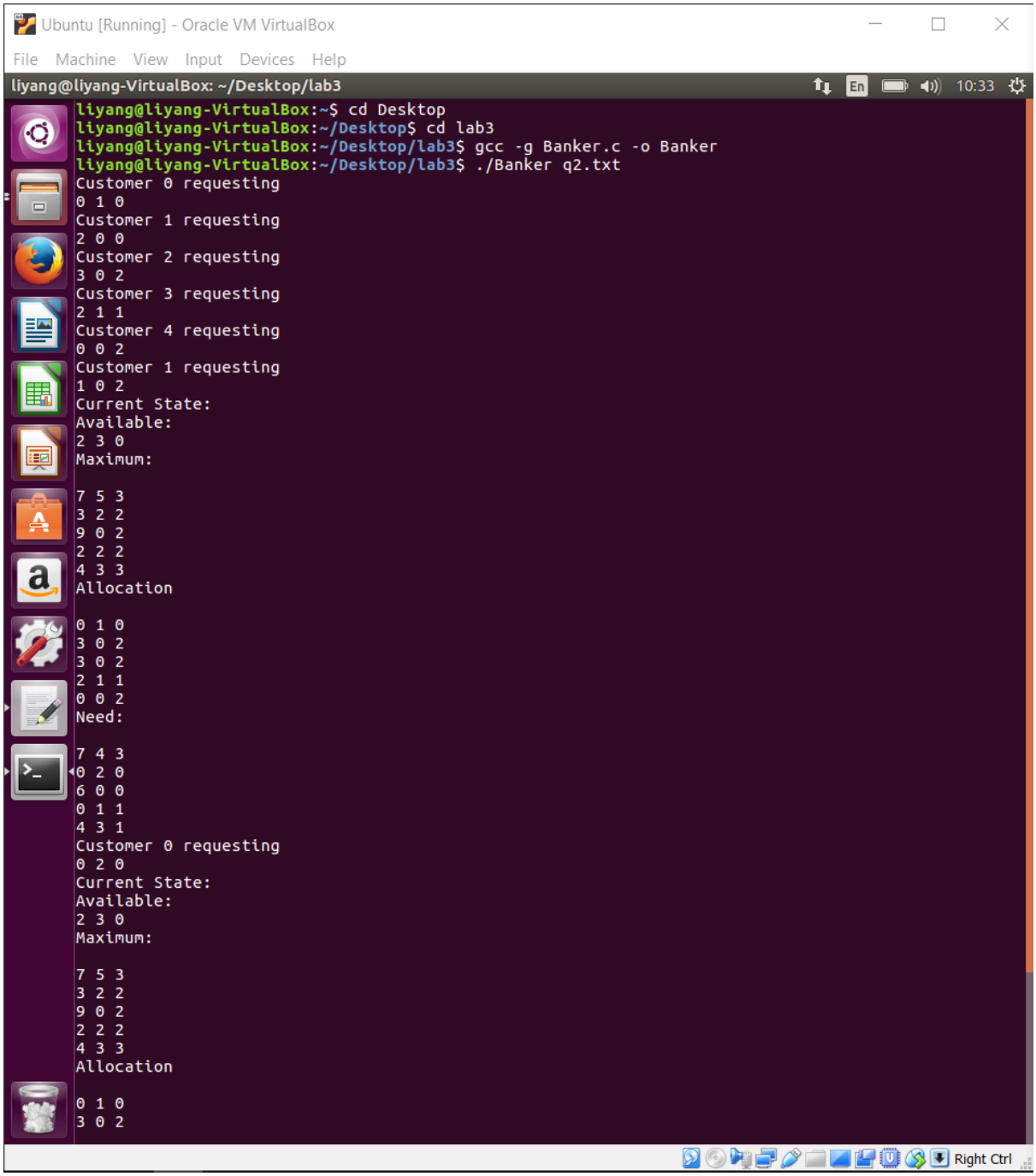
Lab 3

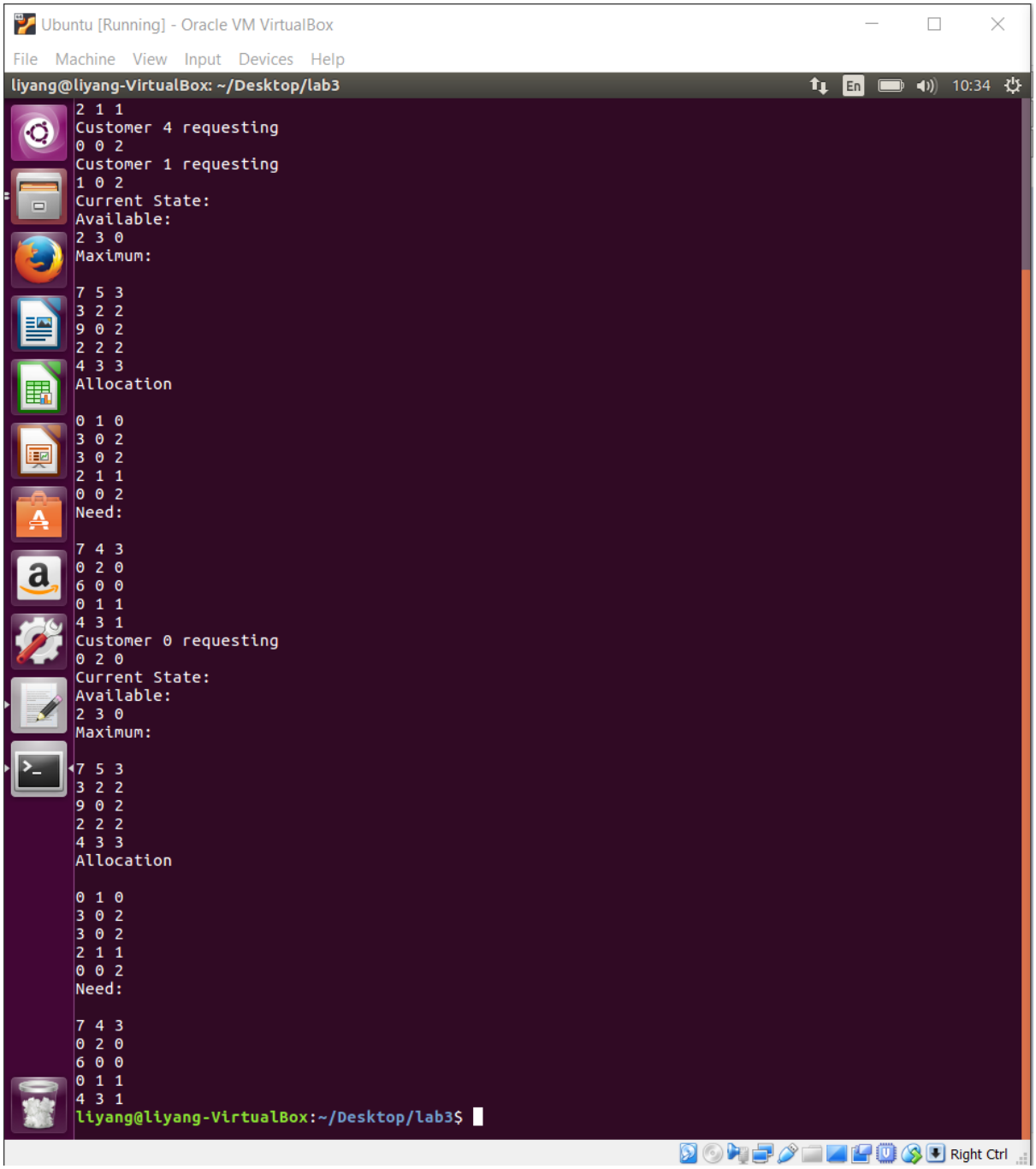
Q1: Implement a Basic Bank System



Q2: Implement a Safety Check Algorithm



```
liyang@liyang-VirtualBox:~$ cd Desktop
liyang@liyang-VirtualBox:~/Desktop$ cd lab3
liyang@liyang-VirtualBox:~/Desktop/lab3$ gcc -g Banker.c -o Banker
liyang@liyang-VirtualBox:~/Desktop/lab3$ ./Banker q2.txt
Customer 0 requesting
0 1 0
Customer 1 requesting
2 0 0
Customer 2 requesting
3 0 2
Customer 3 requesting
2 1 1
Customer 4 requesting
0 0 2
Customer 1 requesting
1 0 2
Current State:
Available:
2 3 0
Maximum:

7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Allocation

0 1 0
3 0 2
3 0 2
2 1 1
0 0 2
Need:

7 4 3
0 2 0
6 0 0
0 1 1
4 3 1
Customer 0 requesting
0 2 0
Current State:
Available:
2 3 0
Maximum:

7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Allocation

0 1 0
3 0 2
```



Q3: Discuss the complexity of Banker's Algorithm

In this case, with n = number of customers and m = number of resources, the time complexity would be $O(n^2*m)$. In the checkSafe function, where it must run a double for loop as many times as the number of customers as well as the number of resources, as shown below.

```
while (possible) {
    possible = false;
    for (int i = 0; i < numberOfCustomers; i++) {
        feasible = true;
        for (int j = 0; j < numberOfResources; j++) {
            if (tempNeed[i][j] > work[j]) feasible = false;
        }
        if (!finish[i] && feasible) {
                possible = true;
            finish[i] = true;
            for (int j = 0; j < numberOfResources; j++){
                work[j] += tempAllocation[i][j];
            }
        }
    }
}
```

This gives a O(n*m) complexity. The while loop would also need to run a maximum of n times. In the worst case, it is ordered in such a way that banker's algorithm would need to evaluate all remaining customers before the last one satisfies $Need_i$<=Work. This gives a final time complexity of $O(n^2*m)$.