CS307 Project6: Banker's Algorithm

Junjie Wang 517021910093

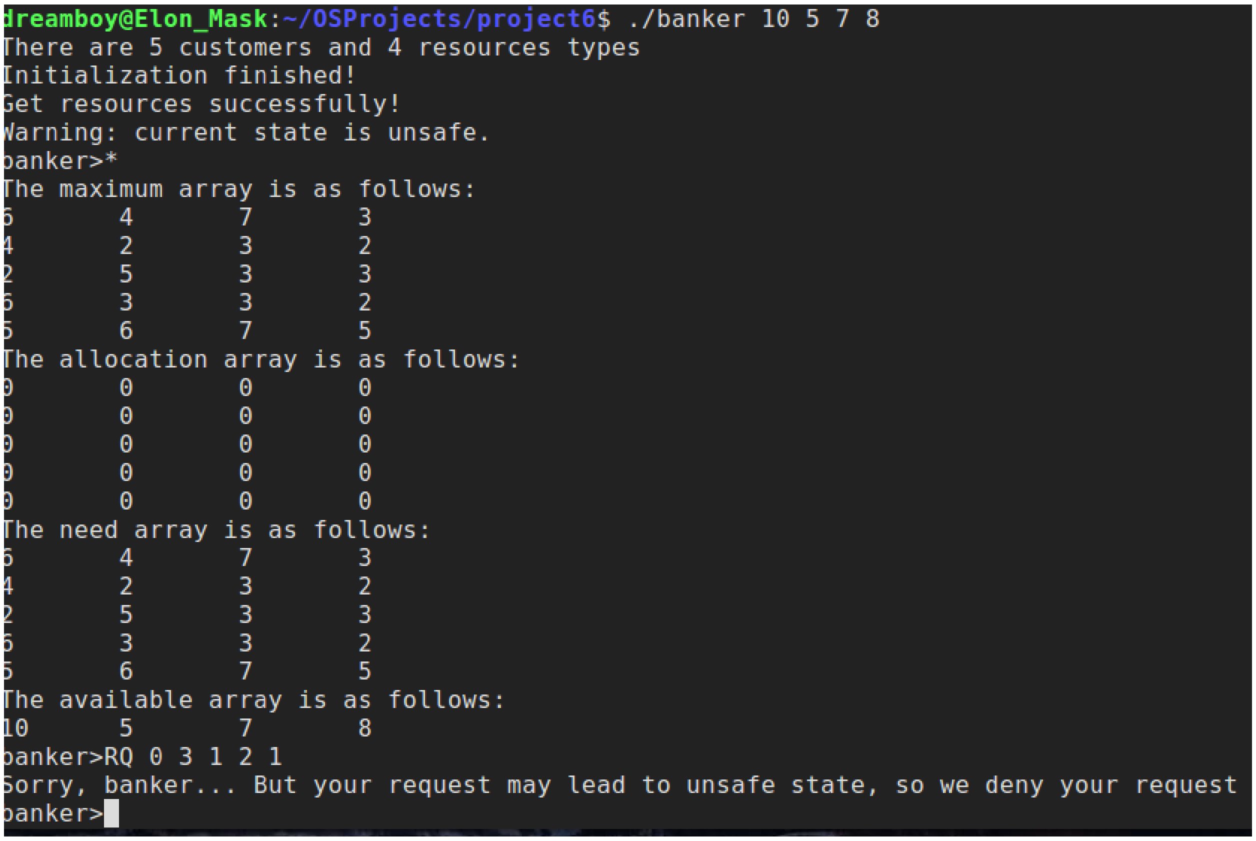
May 23, 2019

1 Programming Thoughts

Banker's algorithm is widely used to detect whether the state of a system is safe, then preventing deadlocks by denying requests which will cause a safe state into an unsafe state. We have to maintain 4 arrays(maximum, allocation, need, available) through the whole process. These 4 arrays will be used to decide whether we can find a sequence to execute all the processes(if such sequence can be found, we claim that the system is safe, otherwise it's unsafe). Note that once a process is finished, we release all the resources allocated to this process and update the 4 arrays accordingly.
The stratey here to handle the request of resources is: assume that we accept the request(in my implementation, I copy the 4 arrays as backup, actually this is not very efficient), then try to find a sequence to execute all the processes. If we can find, we turn back to accept the request, otherwise we deny it.
The strategy the handle the release request is much simpler, we only need to ensure that the release request does not exceed the process's allocation. Then we can update the 4 arrays accordingly.

2 Execution Results And Snapshots

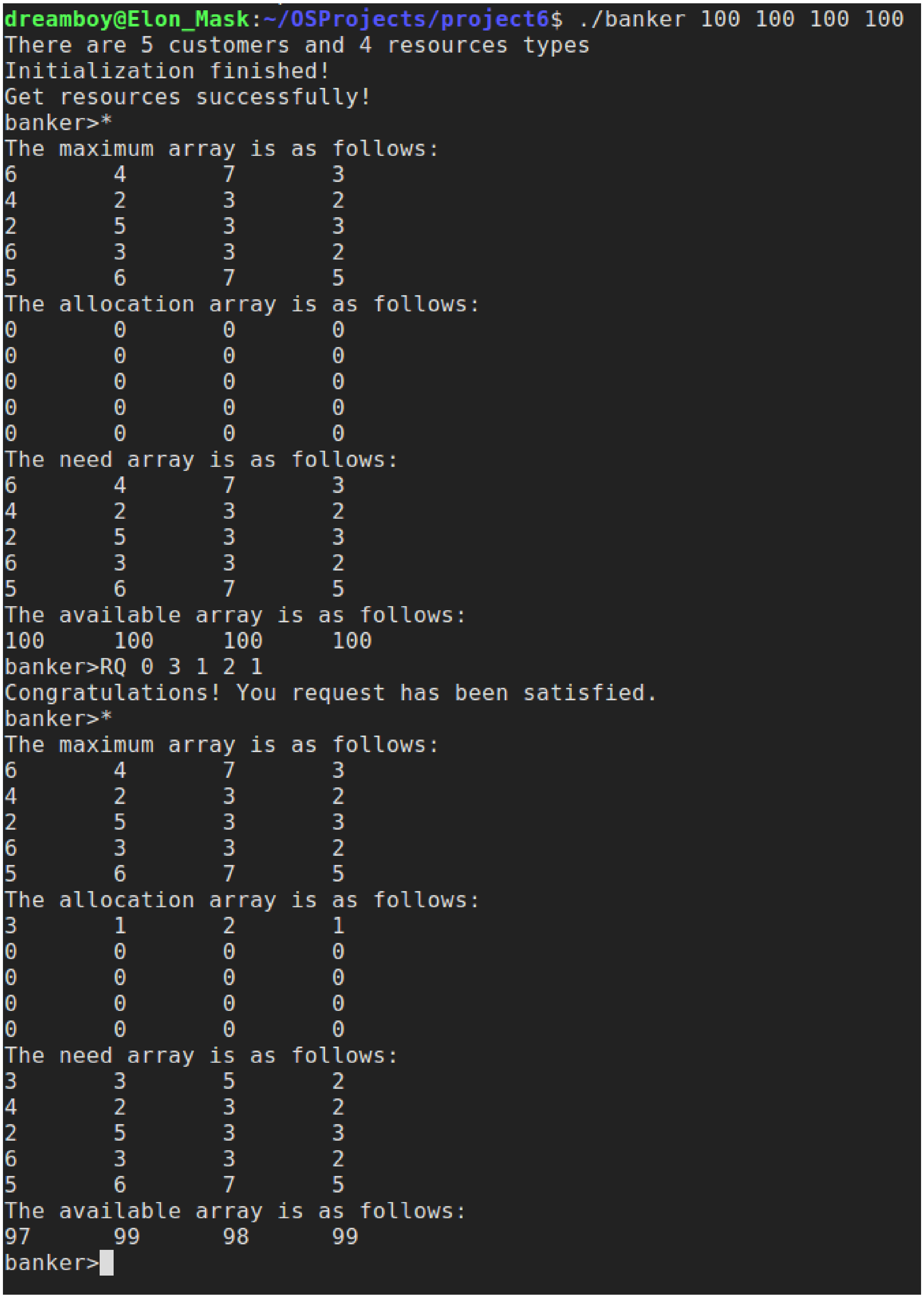
As we can see in 1, initialliy the allocation array is empty. If we use input (10,5,7,8)(claiming the maximum amout of the resources), since the maximum array of the fifth customer is (5,6,7,5), we can not find a sequence to satisfy all the customers, indicating that current state is unsafe. So the coming request will be denied for sure.



```
dreamboy@Elon_Mask:~/OSProjects/project6$ ./banker 10 5 7 8
There are 5 customers and 4 resources types
Initialization finished!
Get resources successfully!
Warning: current state is unsafe.
banker>*
The maximum array is as follows:
6       4       7       3
4       2       3       2
2       5       3       3
6       3       3       2
5       6       7       5
The allocation array is as follows:
0       0       0       0
0       0       0       0
0       0       0       0
0       0       0       0
0       0       0       0
The need array is as follows:
6       4       7       3
4       2       3       2
2       5       3       3
6       3       3       2
5       6       7       5
The available array is as follows:
10      5       7       8
banker>RQ 0 3 1 2 1
Sorry, banker... But your request may lead to unsafe state, so we deny your request
banker>
```

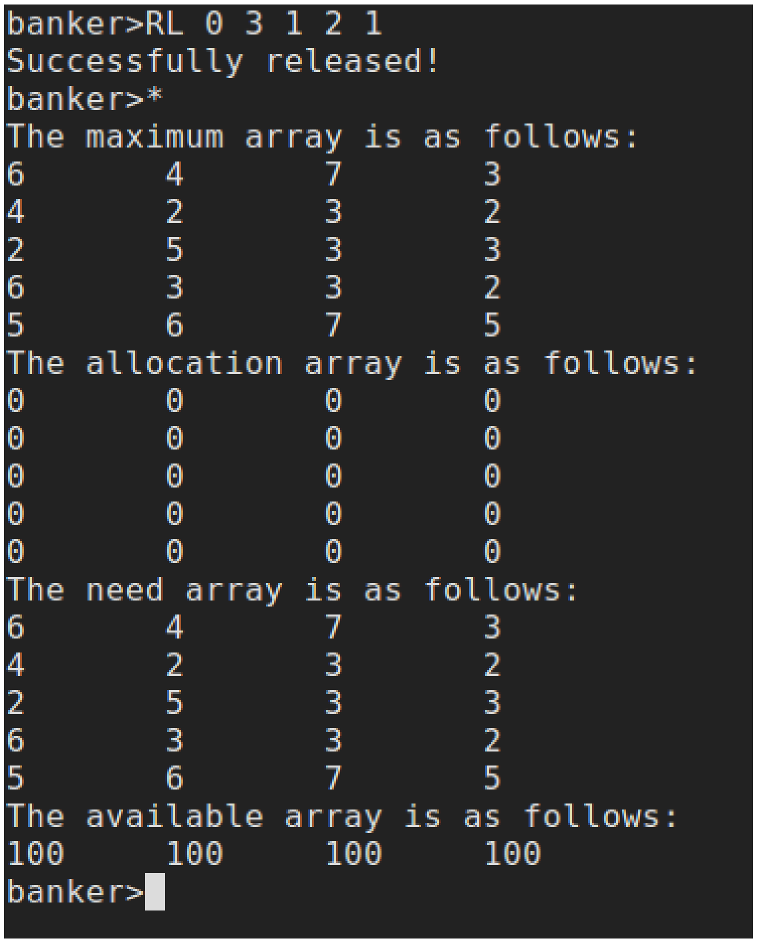
Figure 1: attempt 1

Next we use a much larger input, (100, 100, 100, 100) for 4 types of resources. This time we can see in 2 the resources are allocated successfully. And once we release the resources, the arrays are all restored.



```
dreamboy@Elon_Mask:~/OSProjects/project6$ ./banker 100 100 100 100
There are 5 customers and 4 resources types
Initialization finished!
Get resources successfully!
banker>*
The maximum array is as follows:
6       4       7       3
4       2       3       2
2       5       3       3
6       3       3       2
5       6       7       5
The allocation array is as follows:
0       0       0       0
0       0       0       0
0       0       0       0
0       0       0       0
0       0       0       0
The need array is as follows:
6       4       7       3
4       2       3       2
2       5       3       3
6       3       3       2
5       6       7       5
The available array is as follows:
100     100     100     100
banker>RQ 0 3 1 2 1
Congratulations! You request has been satisfied.
banker>*
The maximum array is as follows:
6       4       7       3
4       2       3       2
2       5       3       3
6       3       3       2
5       6       7       5
The allocation array is as follows:
3       1       2       1
0       0       0       0
0       0       0       0
0       0       0       0
0       0       0       0
The need array is as follows:
3       3       5       2
4       2       3       2
2       5       3       3
6       3       3       2
5       6       7       5
The available array is as follows:
97      99      98      99
banker>
```

Figure 2: attempt 2



```
banker>RL 0 3 1 2 1
Successfully released!
banker>*
The maximum array is as follows:
6       4       7       3
4       2       3       2
2       5       3       3
6       3       3       2
5       6       7       5
The allocation array is as follows:
0       0       0       0
0       0       0       0
0       0       0       0
0       0       0       0
0       0       0       0
The need array is as follows:
6       4       7       3
4       2       3       2
2       5       3       3
6       3       3       2
5       6       7       5
The available array is as follows:
100     100     100     100
banker>
```

Figure 3: attempt 2

3 Code Explanation

The core logic is to how to check whether the system will be in safe state. The idea is to use a *checkSafeWrapper*()(where the 4 arrays are restored for later restoration) function to call *checkSafe*() function, which will determine whether the system will be in safe state.

```
int checkSafe(int *finished){
  int i;
  for(i=0;i<num_customers && finished[i];i++) ;
  if(i == num_customers) return 1;

  /* currently not all the values in finished array are 1
     if we can not find a customer which can be satisfied, we fail
     i.e. the system is unsafe */
  int customerIndex = findSomeCustomer(finished);
  if(customerIndex == -1) return 0;
  else{
  releaseResources(customerIndex, allocation[customerIndex]);
```

```
   finished[customerIndex] = 1;
   checkSafe(finished);
   }
}

/* find some customer who can be sastisfied
   the logic is to compare the need array and available array */
int findSomeCustomer(int *finished){
   int i, j;
   for(i=0;i<num_customers;i++){
   for(j=0;j<num_resources;j++){
   if(need[i][j] > available[j]) break;
   }
   if(j == num_resources && !finished[i]) return i;
   }
   return -1;
}
```