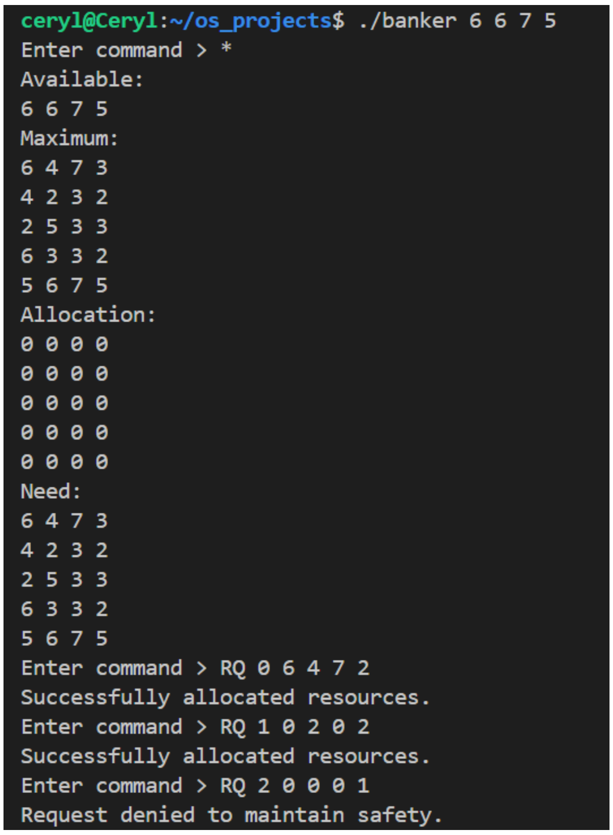
Project 6

Banker’s Algorithm

Introduction

The Banker's Algorithm is a deadlock avoidance algorithm that is used to avoid deadlock in a system. It is named so because it is based on the principles of a bank. A bank never allocates available cash in such a way that it can no longer satisfy the needs of all of its customers.

The essential part of the Banker's Algorithm is the safety algorithm, which is used to check whether a system is in a safe state or not. A system is in a safe state only if there is a safe sequence of processes that, by running in that sequence, will allow each process to acquire all the resources it needs, finish the execution, and release all the resources it has acquired. If a system is in a safe state after an allocation request, the Banker's Algorithm will grant the request; otherwise, it will deny the request and force the process to wait until the system is in a safe state.



```
ceryl@Ceryl:~/os_projects$ ./banker 6 6 7 5
Enter command > *
Available:
6 6 7 5
Maximum:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Allocation:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
Need:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Enter command > RQ 0 6 4 7 2
Successfully allocated resources.
Enter command > RQ 1 0 2 0 2
Successfully allocated resources.
Enter command > RQ 2 0 0 0 1
Request denied to maintain safety.
```

Figure 1: Test using command-line interface

Implementation

Algorithm

The Banker's Algorithm is implemented by following functions:

```c
void request_resources(int customer_num, int request[])
{
  // Check if request exceeds need or available
  ...

  // Pretend to allocate resources
  for (int i = 0; i < NUMBER_OF_RESOURCES; ++i)
  {
    available[i]                 -= request[i];
    allocation[customer_num][i] += request[i];
    need[customer_num][i]       -= request[i];
  }

  // Check if system is in safe state
  if (check_safe())
    printf("Successfully allocated resources.\n");
  else
  {
    // Rollback allocation
    for (int i = 0; i < NUMBER_OF_RESOURCES; ++i)
    {
      available[i]                 += request[i];
      allocation[customer_num][i] -= request[i];
      need[customer_num][i]       += request[i];
    }
    printf("Request denied to maintain safety.\n");
  }
}

void release_resources(int customer_num, int release[])
{
  // Check if release exceeds allocation
  // Release the resources
}

int check_safe(void)
{
  // The available resources in the simulation
  int work[NUMBER_OF_RESOURCES];
  // The finish status of each customer
  int finish[NUMBER_OF_CUSTOMERS];
  int finish_count = 0;

  for (int i = 0; i < NUMBER_OF_RESOURCES; ++i)
    work[i] = available[i];
  memset(finish, 0, sizeof(finish));

  while (finish_count < NUMBER_OF_CUSTOMERS)
  {
    // Search for a customer that can finish
    int found = 0;

    for (int i = 0; i < NUMBER_OF_CUSTOMERS; ++i)
      if (!finish[i])
      {
        // Check if customer can finish
        int enough = 1;
        for (int j = 0; j < NUMBER_OF_RESOURCES; ++j)
          if (need[i][j] > work[j])
          {
            enough = 0;
            break;
          }
        // If customer can finish, release all its resources
        if (enough)
        {
          found     = 1;
          finish[i] = 1;
          ++finish_count;
          for (int j = 0; j < NUMBER_OF_RESOURCES; ++j)
            work[j] += allocation[i][j];
        }
      }

    // If no customer can finish, the system is in an unsafe state
    if (!found)
        return 0;
  }

  // Found a safe sequence, the system is in a safe state
  return 1;
}
```

Command-line Interface

The Banker's Algorithm is tested using a command-line interface. The user can request resources and release resources using the following commands:

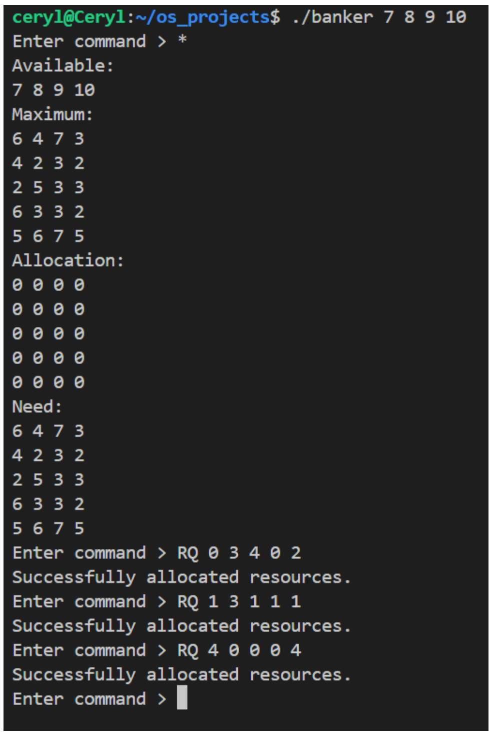
- `RQ <customer_num> <request1> <request2> ...`: Request resources for a customer.
- `RL <customer_num> <release1> <release2> ...`: Release resources for a customer.
- `*`: Print the current state of the system.
- `exit`: Exit the program.

The main function parses the user input and calls the corresponding functions to request or release resources, print the current state of the system, or exit the program.

If one of the customers is finished, the resources allocated to the customer are released and its need is set to `0`, in order to indicate that the customer has finished its execution when the state of the system is printed.

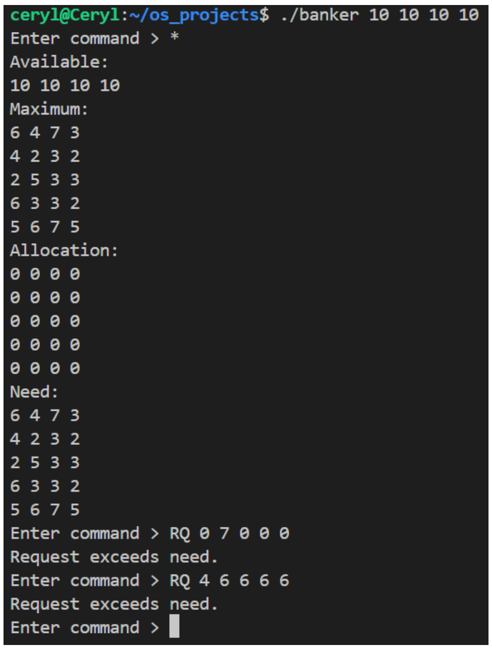
Correctness

The correctness test of the Banker's Algorithm is shown in the following figure:



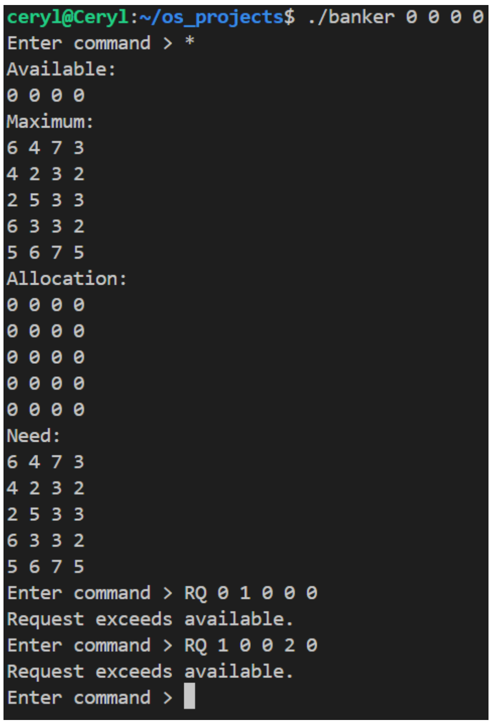
```
ceryl@Ceryl:~/os_projects$ ./banker 7 8 9 10
Enter command > *
Available:
7 8 9 10
Maximum:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Allocation:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
Need:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Enter command > RQ 0 3 4 0 2
Successfully allocated resources.
Enter command > RQ 1 3 1 1 1
Successfully allocated resources.
Enter command > RQ 4 0 0 0 4
Successfully allocated resources.
Enter command >
```

Figure 2: Request is granted



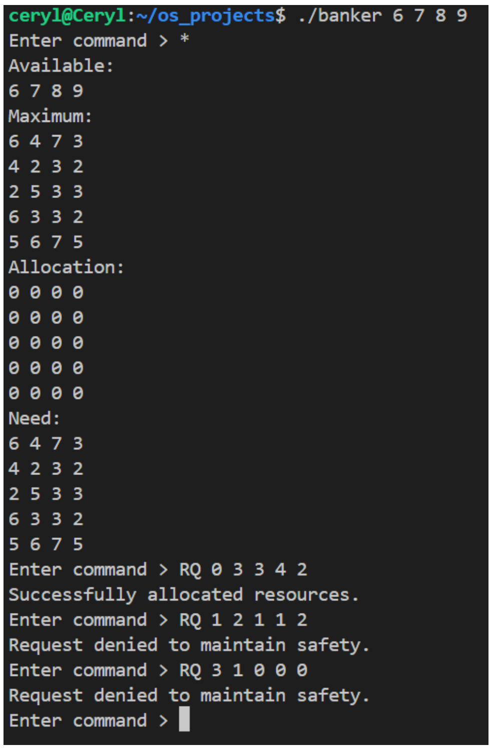
```
ceryl@Ceryl:~/os_projects$ ./banker 10 10 10 10
Enter command > *
Available:
10 10 10 10
Maximum:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Allocation:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
Need:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Enter command > RQ 0 7 0 0 0
Request exceeds need.
Enter command > RQ 4 6 6 6 6
Request exceeds need.
Enter command >
```

Figure 3: Request exceeds need



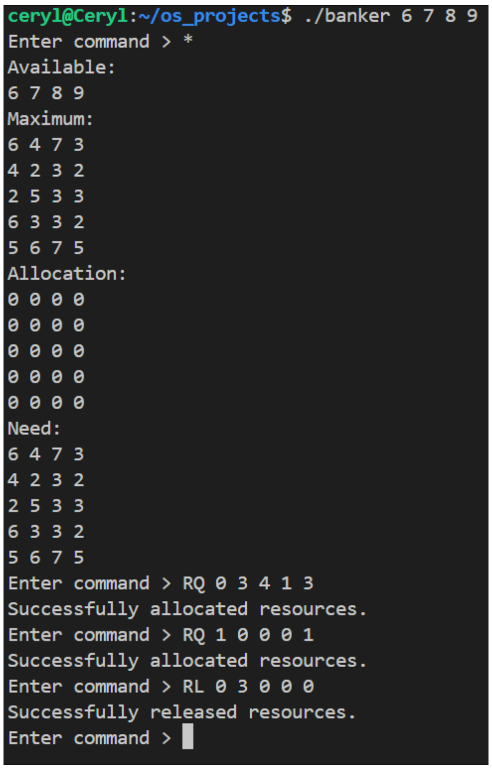
```
ceryl@Ceryl:~/os_projects$ ./banker 0 0 0 0
Enter command > *
Available:
0 0 0 0
Maximum:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Allocation:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
Need:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Enter command > RQ 0 1 0 0 0
Request exceeds available.
Enter command > RQ 1 0 0 2 0
Request exceeds available.
Enter command >
```

Figure 4: Request exceeds available



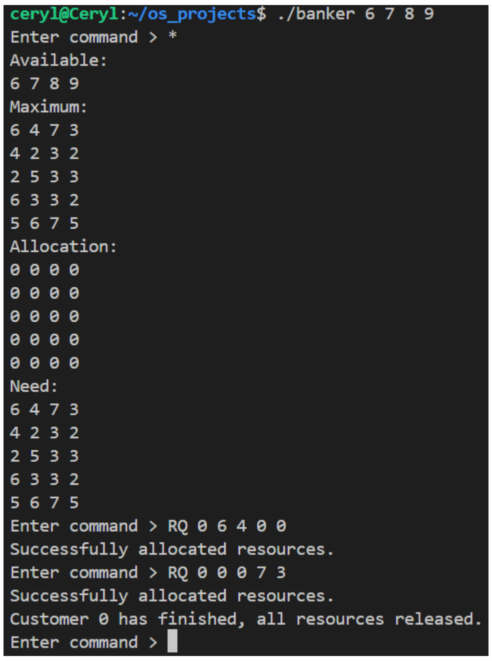
```
ceryl@Ceryl:~/os_projects$ ./banker 6 7 8 9
Enter command > *
Available:
6 7 8 9
Maximum:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Allocation:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
Need:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Enter command > RQ 0 3 3 4 2
Successfully allocated resources.
Enter command > RQ 1 2 1 1 2
Request denied to maintain safety.
Enter command > RQ 3 1 0 0 0
Request denied to maintain safety.
Enter command >
```

Figure 5: Request is denied to maintain safety



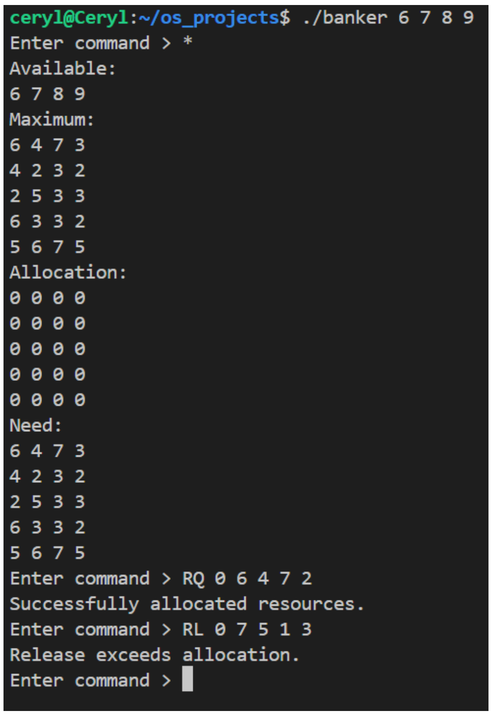
```
ceryl@Ceryl:~/os_projects$ ./banker 6 7 8 9
Enter command > *
Available:
6 7 8 9
Maximum:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Allocation:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
Need:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Enter command > RQ 0 3 4 1 3
Successfully allocated resources.
Enter command > RQ 1 0 0 0 1
Successfully allocated resources.
Enter command > RL 0 3 0 0 0
Successfully released resources.
Enter command >
```

Figure 6: Release resources



```
ceryl@Ceryl:~/os_projects$ ./banker 6 7 8 9
Enter command > *
Available:
6 7 8 9
Maximum:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Allocation:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
Need:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Enter command > RQ 0 6 4 0 0
Successfully allocated resources.
Enter command > RQ 0 0 0 7 3
Successfully allocated resources.
Customer 0 has finished, all resources released.
Enter command >
```

Figure 7: Release resources when customer is finished



```
ceryl@Ceryl:~/os_projects$ ./banker 6 7 8 9
Enter command > *
Available:
6 7 8 9
Maximum:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Allocation:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
Need:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Enter command > RQ 0 6 4 7 2
Successfully allocated resources.
Enter command > RL 0 7 5 1 3
Release exceeds allocation.
Enter command >
```

Figure 8: Release exceeds allocation