School of Computer and Information Science

《**Operating-System-Concepts(10th)**》

Project Report

Student information：

Name:______张乐之______________________ID:____222022321102120__________
Major/Grade:______2022 计科中外 04______________Time:_____2024-11-17__________

Experiment information：

Topic:
Programming Projects in Chapter 8

Requirements:
1. Master the basic concepts, design ideas and status of the process. Understand the basic principles and ideas of process synchronization and mutual exclusion, and identify the key links, steps and constraints of engineering problems in this field. Master the banker algorithm of the process to avoid the OS deadlock.
2. To design implementation solutions to avoid OS deadlock. Master the management method of the system resources and establish the system view.

Procedure：
1. Define the algorithm and figure out an how to avoid deadlocks.
2. Define the behavior of each thread.
3. Code with the header pthread.h and unistd.h to create thread under one process.
4. Compile and link the codes so that it can run on Linux system.
5. Run the program and verify the result.

Results：
（Code and Figures）

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define NUMBER_OF_CUSTOMERS 5
#define NUMBER_OF_RESOURCES 4
```

```
int available[NUMBER_OF_RESOURCES];
int maximum[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];
int allocation[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];
int need[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];

void initialize_arrays(int argc, char *argv[], const char *filename);
int request_resources(int customer_num, int request[]);
int release_resources(int customer_num, int release[]);
int is_safe_state();
void print_status();

int main(int argc, char *argv[])
{
      if (argc < NUMBER_OF_RESOURCES + 1)
      {
            printf("Usage: %s <resource1> <resource2> ... <resourceN>\n", argv[0]);
            exit(1);
      }

      // Initialize available array and other arrays from file
      initialize_arrays(argc, argv, "maximum.txt");

      char command;
      int customer_num, resources[NUMBER_OF_RESOURCES];

      while (true)
      {
            printf("Enter command (RQ <cust_num> <r1> <r2> ... or RL <cust_num> <r1>
<r2> ... or * to display status): ");
            scanf(" %c", &command);

            if (command == '*')
            {
                  print_status();
            }
            else if (command == 'R')
            {
                  scanf("%c", &command);
                  if (command == 'Q')
                  {
                        scanf(" %d", &customer_num);
                        for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
                        {
                              scanf("%d", &resources[i]);
                        }
                        if (request_resources(customer_num, resources) == 0)
```

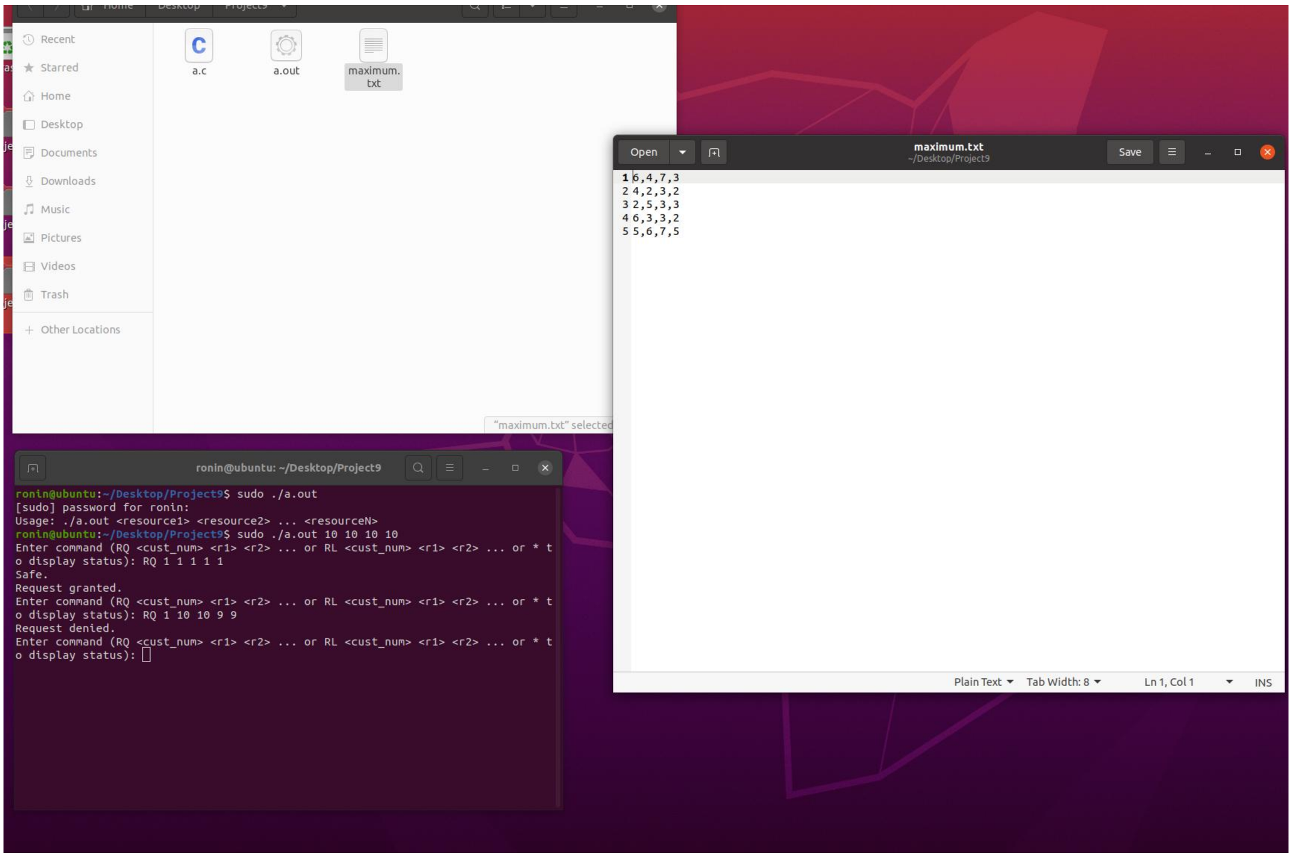
```
                    {
                        printf("Request granted.\n");
                    }
                    else
                    {
                        printf("Request denied.\n");
                    }
                }
                if (command == 'L')
                {
                    scanf("L %d", &customer_num);
                    for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
                    {
                        scanf("%d", &resources[i]);
                    }
                    release_resources(customer_num, resources);
                    printf("Resources released.\n");
                }
            }
            else
            {
                printf("Invalid command.\n");
            }
    }

    return 0;
}

void initialize_arrays(int argc, char *argv[], const char *filename)
{
    // Initialize available resources from command line
    for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
    {
        available[i] = atoi(argv[i + 1]);
    }

    // Read maximum matrix from file
    FILE *file = fopen(filename, "r");
    if (file == NULL)
    {
        perror("Error opening file");
        exit(1);
    }

    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
    {
            for (int j = 0; j < NUMBER_OF_RESOURCES; j++)
            {
                if (j == NUMBER_OF_RESOURCES - 1)
                {
                    // 最后一列不需要逗号
                    fscanf(file, "%d", &maximum[i][j]);
                }
                else
                {
                    // 处理逗号分隔符
                    fscanf(file, "%d,", &maximum[i][j]);
                }
                allocation[i][j] = 0;
                need[i][j] = maximum[i][j];
            }
        }

        fclose(file);
}

int request_resources(int customer_num, int request[])
{
        // Check if request is within need
        for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
        {
            if (request[i] > need[customer_num][i])
            {
                return -1; // Request exceeds need
            }
        }

        // Check if resources are available
        for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
        {
            if (request[i] > available[i])
            {
                return -1; // Not enough resources
            }
        }

        // 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 the system is in a safe state
    if (is_safe_state())
    {
        return 0; // Request granted
    }
    else
    {
        // Rollback allocation if state is unsafe
        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];
        }
        return -1; // Request denied
    }
}

int release_resources(int customer_num, int release[])
{
    // Release resources
    for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
    {
        allocation[customer_num][i] -= release[i];
        available[i] += release[i];
        need[customer_num][i] += release[i];
    }
    return 0;
}

int is_safe_state()
{
    int work[NUMBER_OF_RESOURCES];
    bool finish[NUMBER_OF_CUSTOMERS] = {false};

    // Initialize work = available
    for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
    {
        work[i] = available[i];
    }

    while (true)
    {
        bool found = false;
```

```
            for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
            {
                  if (!finish[i])
                  {
                        int j;
                        for (j = 0; j < NUMBER_OF_RESOURCES; j++)
                        {
                              if (need[i][j] > work[j])
                                    break;
                        }
                        if (j == NUMBER_OF_RESOURCES)
                        {
                              for (int k = 0; k < NUMBER_OF_RESOURCES; k++)
                              {
                                    work[k] += allocation[i][k];
                              }
                              finish[i] = true;
                              found = true;
                        }
                  }
            }
            if (!found)
                  break;
      }

      for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
      {
            if (!finish[i])
            {
                  printf("Not Safe.\n");
                  return 0; // Not safe
            }
      }
      printf("Safe.\n");
      return 1; // Safe
}

void print_status()
{
      printf("Available resources:\n");
      for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
      {
            printf("%d ", available[i]);
      }
      printf("\n\nMaximum resources:\n");
      for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
```

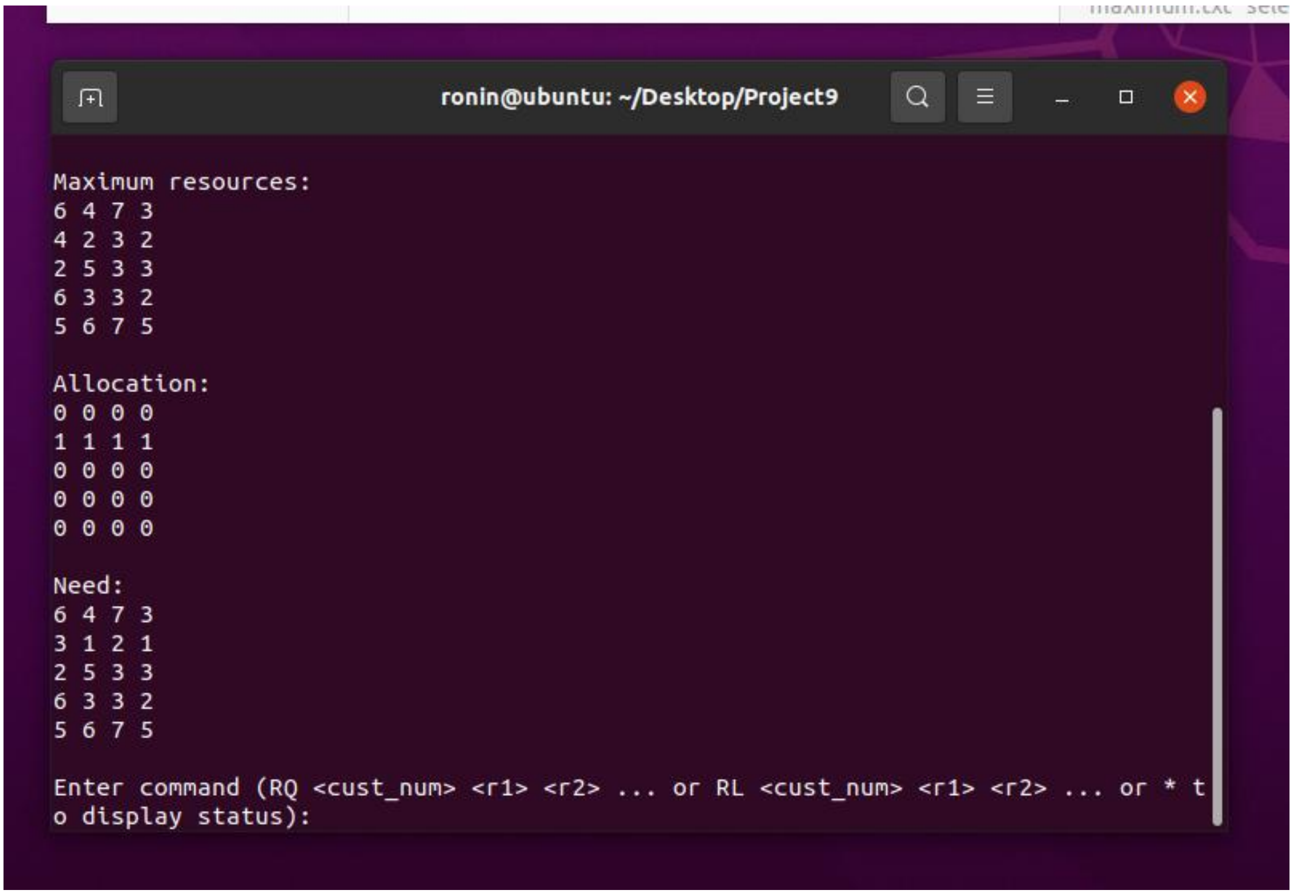
```
	{
		for (int j = 0; j < NUMBER_OF_RESOURCES; j++)
		{
			printf("%d ", maximum[i][j]);
		}
		printf("\n");
	}
	printf("\nAllocation:\n");
	for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
	{
		for (int j = 0; j < NUMBER_OF_RESOURCES; j++)
		{
			printf("%d ", allocation[i][j]);
		}
		printf("\n");
	}
	printf("\nNeed:\n");
	for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
	{
		for (int j = 0; j < NUMBER_OF_RESOURCES; j++)
		{
			printf("%d ", need[i][j]);
		}
		printf("\n");
	}
	printf("\n");
}
```

The algorithm use is_safe_state() function to check whether any of the customer’ s need can be fulfilled. If can, than this systemis safe, not safe elsewise. After a request is received, the request_resources() function allocate the resources after checking resources requested are available and doesn’ t exceed the need. Then it uses is_safe_state() to check the system health. If not safe than the change will be revoked, and the request is rejected.

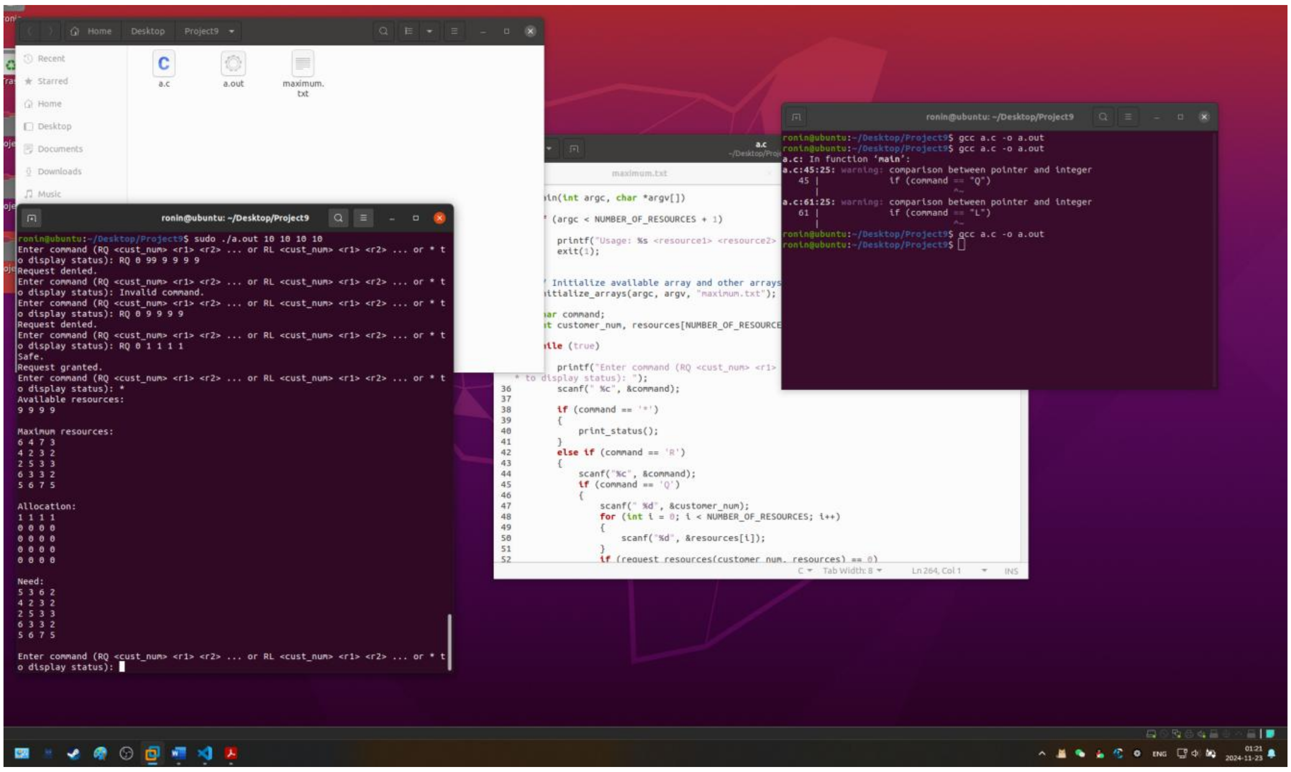


```
6,4,7,3
4,2,3,2
2,5,3,3
6,3,3,2
5,6,7,5
```

```
ronin@ubuntu:~/Desktop/Project9$ sudo ./a.out
[sudo] password for ronin:
Usage: ./a.out <resource1> <resource2> ... <resourceN>
ronin@ubuntu:~/Desktop/Project9$ sudo ./a.out 10 10 10 10
Enter command (RQ <cust_num> <r1> <r2> ... or RL <cust_num> <r1> <r2> ... or * t
o display status): RQ 1 1 1 1 1
Safe.
Request granted.
Enter command (RQ <cust_num> <r1> <r2> ... or RL <cust_num> <r1> <r2> ... or * t
o display status): RQ 1 10 10 9 9
Request denied.
Enter command (RQ <cust_num> <r1> <r2> ... or RL <cust_num> <r1> <r2> ... or * t
o display status): 
```



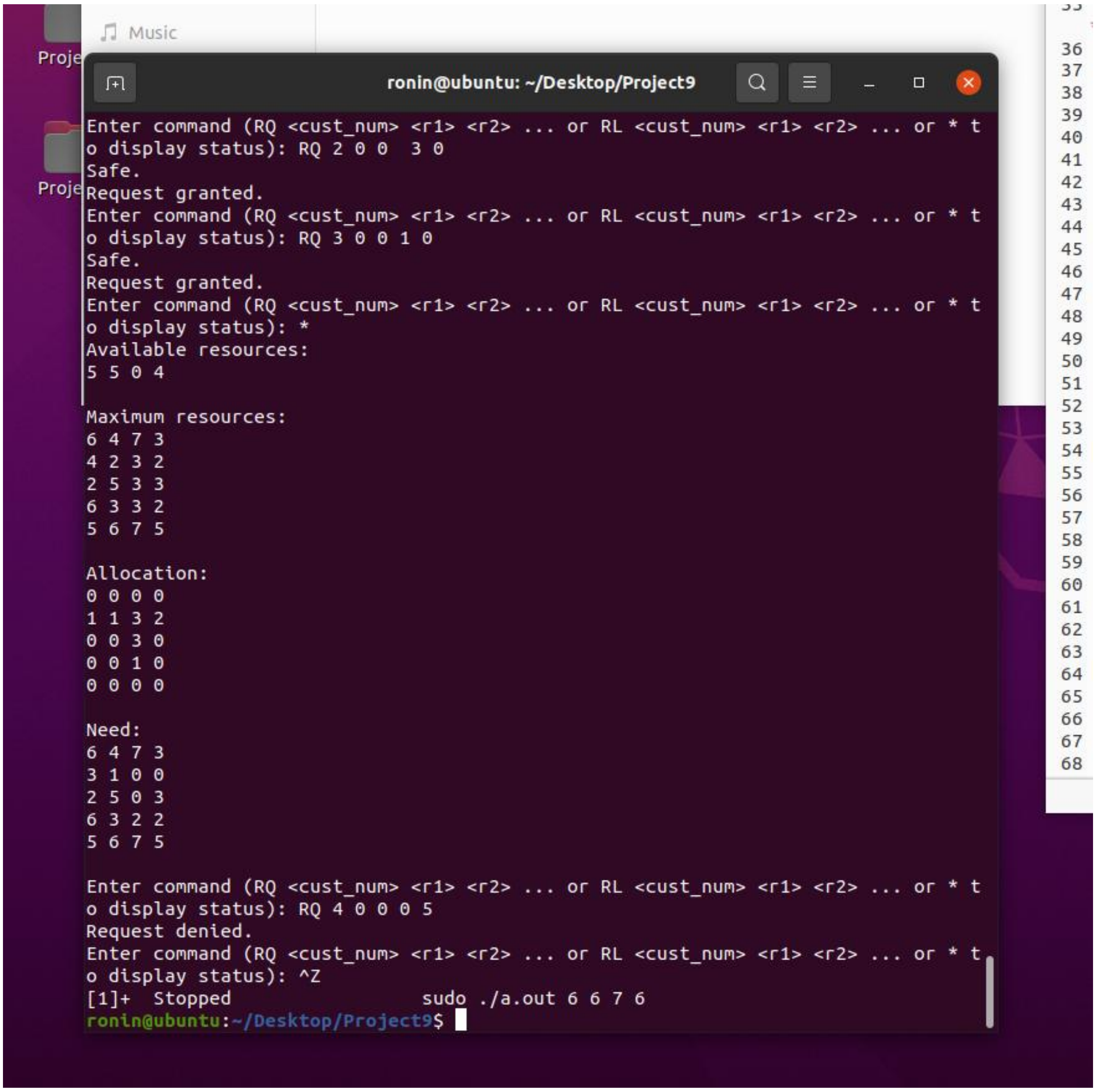
```
Maximum resources:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5

Allocation:
0 0 0 0
1 1 1 1
0 0 0 0
0 0 0 0
0 0 0 0

Need:
6 4 7 3
3 1 2 1
2 5 3 3
6 3 3 2
5 6 7 5

Enter command (RQ <cust_num> <r1> <r2> ... or RL <cust_num> <r1> <r2> ... or * t
o display status):
```



```
ronin@ubuntu:~/Desktop/Project9$ sudo ./a.out 10 10 10 10
Enter command (RQ <cust_num> <r1> <r2> ... or RL <cust_num> <r1> <r2> ... or * t
o display status): RQ 0 99 9 9 9 9
Request denied.
Enter command (RQ <cust_num> <r1> <r2> ... or RL <cust_num> <r1> <r2> ... or * t
o display status): Invalid command.
Enter command (RQ <cust_num> <r1> <r2> ... or RL <cust_num> <r1> <r2> ... or * t
o display status): RQ 0 9 9 9 9
Request denied.
Enter command (RQ <cust_num> <r1> <r2> ... or RL <cust_num> <r1> <r2> ... or * t
o display status): RQ 0 1 1 1 1
Safe.
Request granted.
Enter command (RQ <cust_num> <r1> <r2> ... or RL <cust_num> <r1> <r2> ... or * t
o display status): *
Available resources:
9 9 9 9

Maximum resources:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5

Allocation:
1 1 1 1
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

Need:
5 3 6 2
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5

Enter command (RQ <cust_num> <r1> <r2> ... or RL <cust_num> <r1> <r2> ... or * t
o display status): 
```

```c
ain(int argc, char *argv[])

 (argc < NUMBER_OF_RESOURCES + 1)

  printf("Usage: %s <resource1> <resource2>
  exit(1);


 Initialize available array and other arrays
itialize_arrays(argc, argv, "maximum.txt");

ar command;
t customer_num, resources[NUMBER_OF_RESOURCE

ile (true)

  printf("Enter command (RQ <cust_num> <r1>
* to display status): ");
        scanf(" %c", &command);

        if (command == '*')
        {
            print_status();
        }
        else if (command == 'R')
        {
            scanf("%c", &command);
            if (command == 'Q')
            {
                scanf(" %d", &customer_num);
                for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
                {
                    scanf("%d", &resources[i]);
                }
                if (request_resources(customer_num, resources) == 0)
```

```
ronin@ubuntu:~/Desktop/Project9$ gcc a.c -o a.out
ronin@ubuntu:~/Desktop/Project9$ gcc a.c -o a.out
a.c: In function 'main':
a.c:45:25: warning: comparison between pointer and integer
   45 |                 if (command == "Q")
      |                             ^~
a.c:61:25: warning: comparison between pointer and integer
   61 |                 if (command == "L")
      |                             ^~
ronin@ubuntu:~/Desktop/Project9$ gcc a.c -o a.out
ronin@ubuntu:~/Desktop/Project9$ 
```



```
Enter command (RQ <cust_num> <r1> <r2> ... or RL <cust_num> <r1> <r2> ... or * t
o display status): RQ 2 0 0  3 0
Safe.
Request granted.
Enter command (RQ <cust_num> <r1> <r2> ... or RL <cust_num> <r1> <r2> ... or * t
o display status): RQ 3 0 0 1 0
Safe.
Request granted.
Enter command (RQ <cust_num> <r1> <r2> ... or RL <cust_num> <r1> <r2> ... or * t
o display status): *
Available resources:
5 5 0 4

Maximum resources:
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
1 1 3 2
0 0 3 0
0 0 1 0
0 0 0 0

Need:
6 4 7 3
3 1 0 0
2 5 0 3
6 3 2 2
5 6 7 5

Enter command (RQ <cust_num> <r1> <r2> ... or RL <cust_num> <r1> <r2> ... or * t
o display status): RQ 4 0 0 0 5
Request denied.
Enter command (RQ <cust_num> <r1> <r2> ... or RL <cust_num> <r1> <r2> ... or * t
o display status): ^Z
[1]+  Stopped                 sudo ./a.out 6 6 7 6
ronin@ubuntu:~/Desktop/Project9$
```

As can be seen in the graph, the program rejected the request. Because resource 3 is occupied by customer 2 and 3, so they must be finished first. Customer 5 cannot take resource 4 otherwise the system will no longer be safe.