# Project6: **Banker's Algorithm**

Yanjie Ze 519021910706



# 1 Introduction

For this project, we will write a program that implements the banker's algorithm discussed in Section 8.6.3. Customers request and release resources from the bank. The banker will grant a request only if it leaves the system in a safe state. A request that leaves the system in an unsafe state will be denied.

We use **C** language and finish this project on **macOS Big Sur** operating system.

## 1.1 The Banker

The banker will consider requests from *n* customers for *m* resources types, as outlined in Section 8.6.3. The banker will keep track of the resources using the following data structures:

```
#define NUMBER_OF_CUSTOMERS 5
#define NUMBER_OF_RESOURCES 4

/* the available amount of each resource */
int available[NUMBER_OF_RESOURCES];

/*the maximum demand of each customer */
int maximum[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];

/* the amount currently allocated to each customer */
int allocation[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];

/* the remaining need of each customer */
int need[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];
```

The banker will grant a request if it satisfies the safety algorithm outlined in Section 8.6.3.1.

If a request does not leave the system in a safe state, the banker will deny it. Function prototypes for requesting and releasing resources are as follows:

```
int request_resources(int custome_num, int request[]);
void release_resources(int customer_num, int release[]);
```

The request resources() function should return 0 if successful and –1 if unsuccessful.

## 1.2 Safety Algorithm

The safety algorithm is shown below:

1. Let **Work** and **Finish** be vectors of length *m* and *n,* respectively. Initialize **Work** =

**Available** and **Finish**[*i*] = **false** for *i* = 0, 1, ..., *n* – 1.

2. Find an index *i* such that both

- **Finish**[*i*] == **false**
- $\mathbf{Need_i} \leq$ **Work**

   If no such *i* exists, go to step 4.

3. **Work** = **Work** + $\mathbf{Allocation_i}$

   **Finish**[*i*] = **true**

   Go to step 2.

4. If **Finish**[*i*] == **true** for all *i*, then the system is in a safe state.

This algorithm may require an order of $m \times n^2$ operations to determine whether a state is safe.

## 1.3 Requirements

We need to implement some commands:

```
RQ 0 3 1 2 1
```

Customer 0 is to request the resources (3, 1, 2, 1).

```
RL 4 1 2 3 1
```

Customer 4 is to release the resources (1, 2, 3, 1).

```
*
```

Output the values of the different data structures

What's more, we add one command `end` to end the program.

# 2 Implementation Details and Methods

## 2.1 Top-down Manner

We finish the program in a top-down manner, which is to say:

- First we implement the basic code frame and create the function name.
- Second we realize each function.

## 2.2 Define Macros and Global Variables

In the very first beginning, we need to define some Marcos and Global Variables, which will be used in all the functions.

Four arrays:

- **available** , to represent the available resources.
- **maximum** , to represent the maximum need of customers
- **allocation** , to represent the allocation of resources
- **need** , to represent the remaining need of customers

```
#define NUMBER_OF_CUSTOMERS 5
#define NUMBER_OF_RESOURCES 4
#define SIZE 100
/* the available amount of each resource */
int available[NUMBER_OF_RESOURCES];
/*the maximum demand of each customer */
int maximum[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];
/* the amount currently allocated to each customer */
int allocation[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];
/* the remaining need of each customer */
int need[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];
```

## 2.3 Input file and Parse

We use two files to include the information:

**max_need.txt**:

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

**available_resource.txt**:

```
10,10,10,10
```

Parsing file into information is the first thing we will do in **main()**.

Assign values to **maximum**:

```
// initialize array MAXIMUM with txt file
    FILE *in;
    in = fopen("max_need.txt", "r");
    char customer[SIZE];
    char *temp;
    for(int i=0;i<NUMBER_OF_CUSTOMERS;++i)
    {
        fgets(customer,SIZE,in);
        temp = strdup(customer);
        for(int j=0;j<NUMBER_OF_RESOURCES;j++)
            maximum[i][j]=need[i][j]=atoi(strsep(&temp, ",")),
allocation[i][j]=0;
    }
    fclose(in);
```

Assign values to **available**:

```
// initialize array **available**  with txt file
    in = fopen("available_resource.txt", "r");
    char ava[SIZE];
    fgets(ava, SIZE, in);
    temp = strdup(ava);
    for(int i=0;i<NUMBER_OF_RESOURCES;i++)
        available[i]=atoi(strsep(&temp, ","));
    fclose(in);
```

## 2.4 Execute Commands in main()

We need to implement four kinds of commands: **RQ, RL, *, end**.

**end**. This is the simplest command, and also the first command we need to implement.

```
  if(strcmp(input_line, "end\n")==0)
        {
            running = 0;
            printf("---------------Banker's Algorithm Finished-------------
\n");
            continue;
        }
```

**RQ & RL**. Finish the execution of RQ and RL with the help of two functions: `request()` and `release`.

If the execution of the functions successes, they will return 0. Otherwise, they will return 1.

The detailed content of `request()` and `release` will be shown in the following sections.

```
if(input_line[0]=='R')
```

```
        {
            if(input_line[1]=='Q')
            {
                //decode instruction
                temp = strdup(input_line);
                strsep(&temp," ");
                for(int i=0; i<=NUMBER_OF_RESOURCES;++i)
                    input_instruction[i] = atoi(strsep(&temp, " "));
                request_value = request(input_instruction);
                if(request_value!=0)
                    printf("Request Failed. Please Follow Rules.\n");
                continue;
            }
            else if(input_line[1]=='L')
            {
                temp = strdup(input_line);
                strsep(&temp," ");
                for(int i=0; i<=NUMBER_OF_RESOURCES;++i)
                    input_instruction[i] = atoi(strsep(&temp, " "));
                release_value = release(input_instruction);
                if(release_value!=0)
                    printf("Release Failed. Please Follow Rules.\n");
                continue;
            }
        }
```

`*` . This command is to show the status, using the function **`state_show()`** .

```
    else if(input_line[0]=='*')
                state_show();
```

Overall codes of `main()` are shown here.

```
char input_line[SIZE];
    int running = 1;
```

```
    while(running)
    {
        printf("banker>");
        fgets(input_line, SIZE, stdin);

        if(strcmp(input_line, "end\n")==0)
        {
            running = 0;
            printf("---------------Banker's Algorithm Finished-------------
\n");
            continue;
        }

        if(input_line[0]=='R')
        {
            if(input_line[1]=='Q')
            {
                //decode instruction
                temp = strdup(input_line);
                strsep(&temp," ");
                for(int i=0; i<=NUMBER_OF_RESOURCES;++i)
                    input_instruction[i] = atoi(strsep(&temp, " "));
                request_value = request(input_instruction);
                if(request_value!=0)
                    printf("Request Failed. Please Follow Rules.\n");
                continue;
            }
            else if(input_line[1]=='L')
            {
                temp = strdup(input_line);
                strsep(&temp," ");
                for(int i=0; i<=NUMBER_OF_RESOURCES;++i)
                    input_instruction[i] = atoi(strsep(&temp, " "));
                release_value = release(input_instruction);
                if(release_value!=0)
                    printf("Release Failed. Please Follow Rules.\n");
                continue;
```

```
            }
        }
        else if(input_line[0]=='*')
                state_show();

    }
```

## 2.5 request()

The function `request()` takes an array as input, which including the following information:

- customer id
- request quantity for resource 1
- request quantity for resource 2
- ...

**First**, a check of **whether the customer id is beyond the maximum** is needed:

```
    if(consumer_id>=NUMBER_OF_CUSTOMERS)
     {
         printf("Request Error: ID Exceed\n");
         return -1;
     }
```

**Second**, check whether the need exceeds the maximum need:

```
     // basic check
     for(int i=1;i<=NUMBER_OF_RESOURCES;++i)
     {
         if(a[i]>need[consumer_id][i-1])
         {
             printf("Request Error: Request Exceed Need\n");
             return -1;
```

```
        }
        if(a[i]>available[i-1])
        {
            printf("Request Error: Request Exceed Available\n");
            return -1;
        }
    }
```

**Third**, it's time to call `safety_algorithm` to check if we can satisfy this need.

The implementation of `safety_algotihm` will be described in the following section.

```
    /* safety algorithm*/
    int is_safe = safety_algorithm(a);
    if(is_safe!=0)
    {
        printf("Request Makes Unsafe State!!!\n");
        return 1;
    }
    else
    {
        printf("Request is satisfied.\n");
    }
```

**Forth**, if the state is safe, we can satisfy the requirements:

```
  // after the request, the state should be safe.
    for(int i=1;i<=NUMBER_OF_RESOURCES;++i)
    {
        available[i-1] -= a[i];
        need[consumer_id][i-1] -= a[i];
        allocation[consumer_id][i-1] += a[i];
    }
return 0;
```

Then the function `request()` has been finished. Here is the full version:

```
/* request resource*/
int request(int *a)
{

    int consumer_id = a[0];

    if(consumer_id>=NUMBER_OF_CUSTOMERS)
    {
        printf("Request Error: ID Exceed\n");
        return -1;
    }
    // basic check
    for(int i=1;i<=NUMBER_OF_RESOURCES;++i)
    {
        if(a[i]>need[consumer_id][i-1])
        {
            printf("Request Error: Request Exceed Need\n");
            return -1;
        }
        if(a[i]>available[i-1])
        {
            printf("Request Error: Request Exceed Available\n");
            return -1;
        }
    }
    /* safety algorithm*/
    int is_safe = safety_algorithm(a);
    if(!is_safe)
    {
        printf("Request Makes Unsafe State!!!\n");
        return 0;
    }
    else
    {
        printf("Request is satisfied.\n");
    }
```

```
    // after the request, the state should be safe.
    for(int i=1;i<=NUMBER_OF_RESOURCES;++i)
    {
        available[i-1] -= a[i];
        need[consumer_id][i-1] -= a[i];
        allocation[consumer_id][i-1] += a[i];
    }
    return 1;
}
```

## 2.6 safety_algorithm()

The function `safety_algorithm()` implements the saftey algorithm, which is shown below:

1. Let **Work** and **Finish** be vectors of length *m* and *n,* respectively. Initialize **Work** = **Available** and **Finish**[*i*] = **false** for *i* = 0, 1, ..., *n* − 1.
2. Find an index *i* such that both

- **Finish**[*i*] == **false**
- $\mathbf{Need_i} \leq$ **Work**

   If no such *i* exists, go to step 4.

3. **Work** = **Work** + $\mathbf{Allocation_i}$

   **Finish**[*i*] = **true**

   Go to step 2.

4. If **Finish**[*i*] == **true** for all *i,* then the system is in a safe state.

We just follow the steps of this algorithm to implement.

**First,** create arrays **work** and **finish** and initialize them.Initialize **Work** = **Available** and **Finish**[*i*] = **false** for *i* = 0, 1, ..., *n* − 1.

```
    int work[NUMBER_OF_RESOURCES];
    for(int i=0; i<NUMBER_OF_RESOURCES; i++)
        work[i] = available[i]-a[i+1];

    int finish[NUMBER_OF_CUSTOMERS];
    for(int i=0; i<NUMBER_OF_CUSTOMERS; ++i)
        finish[i]=0;
```

**Second**, find an index *i* such that both **Finish**[*i*] == **false** and $\mathbf{Need_i}$ ≤ **Work**. Then we do **Work** = **Work** + $\mathbf{Allocation_i}$, adn set **Finish**[*i*] = **true**, then go to step 2.

```
int all_satisfy;
    int i;
    for(i=0; i<NUMBER_OF_CUSTOMERS; ++i)
    {
        all_satisfy=0;//refresh

        if(finish[i]==0){
            for(int j=0; j<NUMBER_OF_RESOURCES; ++j)
            {
                if(i==a[0])
                {
                    if(need[i][j]-a[j+1]<=work[j])
                        all_satisfy++;
                }
                else if(need[i][j]<=work[j])
                    all_satisfy++;
            }
            if(all_satisfy==NUMBER_OF_RESOURCES)
            {   //this customer can be finished
                for(int j=0; j<NUMBER_OF_RESOURCES; ++j)
                    work[j] += allocation[i][j];
                finish[i]=1;
                i=-1;// back to the start
            }
        }
```

```
    }
```

**Third**, we check if **Finish**[*i*] == **true** for all i. If so, the system is in a safe state.

Wo do checking by adding **finish[i]** together :

```
all_satisfy = 0;
    for(i=0; i<NUMBER_OF_CUSTOMERS; ++i)
        if(finish[i]==1)
            all_satisfy++;
    if(all_satisfy==NUMBER_OF_CUSTOMERS)
        return 0; //safe
    else
        return -1; //unsafe
```

Finally, we have the full implementation of the safety algorithm:

```
int safety_algorithm(int *a)
{
    int work[NUMBER_OF_RESOURCES];
    for(int i=0; i<NUMBER_OF_RESOURCES; i++)
        work[i] = available[i]-a[i+1];

    int finish[NUMBER_OF_CUSTOMERS];
    for(int i=0; i<NUMBER_OF_CUSTOMERS; ++i)
        finish[i]=0;

    int all_satisfy;
    int i;
    for(i=0; i<NUMBER_OF_CUSTOMERS; ++i)
    {
        all_satisfy=0;//refresh

        if(finish[i]==0){
            for(int j=0; j<NUMBER_OF_RESOURCES; ++j)
            {
```

```
                if(i==a[0])
                {
                    if(need[i][j]-a[j+1]<=work[j])
                        all_satisfy++;
                }
                else if(need[i][j]<=work[j])
                    all_satisfy++;
            }
            if(all_satisfy==NUMBER_OF_RESOURCES)
            {   //this customer can be finished
                for(int j=0; j<NUMBER_OF_RESOURCES; ++j)
                    work[j] += allocation[i][j];
                finish[i]=1;
                i=-1;// back to the start
            }
        }
    }

    all_satisfy = 0;
    for(i=0; i<NUMBER_OF_CUSTOMERS; ++i)
        if(finish[i]==1)
            all_satisfy++;
    if(all_satisfy==NUMBER_OF_CUSTOMERS)
        return 0; //safe
    else
        return -1; //unsafe
}
```

## 2.7 release()

The function `release()` release the resources of the customers.

**First**, it will check whether the customer id exceeds the maximum.

```
  int consumer_id = a[0];
    if(consumer_id>=NUMBER_OF_CUSTOMERS)
    {
        printf("Request Error: ID Exceed\n");
        return -1;
    }
```

**Second**, it will check whether releasing is legal, which prevents the wrong releasing.

```
   // basic check
    for(int i=1;i<=NUMBER_OF_RESOURCES;++i)
    {
        if(a[i]>allocation[consumer_id][i-1])
        {
            printf("Release Error: Release Exceed Allocation\n");
            return -1;
        }
    }
```

**Third**, after checking, it will release the resources:

```
// no exceed
    for(int i=1;i<=NUMBER_OF_RESOURCES; ++i)
    {
        allocation[consumer_id][i-1] -= a[i];
    }
    return 0;
```

For the reference, codes below show the full implementation of `release()`.

```
/* release resource*/
int release(int *a)
{
    int consumer_id = a[0];
```

```
    if(consumer_id>=NUMBER_OF_CUSTOMERS)
    {
        printf("Request Error: ID Exceed\n");
        return -1;
    }
    // basic check
    for(int i=1;i<=NUMBER_OF_RESOURCES;++i)
    {
        if(a[i]>allocation[consumer_id][i-1])
        {
            printf("Release Error: Release Exceed Allocation\n");
            return -1;
        }
    }
    // no exceed
    for(int i=1;i<=NUMBER_OF_RESOURCES; ++i)
    {
        allocation[consumer_id][i-1] -= a[i];
    }
    return 0;
}
```

## 2.8 state_show()

The function **state_show()** displays the arrays we create.

To implement this, just do **printf()** .

```
/* show all the states*/
void state_show()
{
    int i;
    int j;
    printf("-----------------Available Resource---------------\n");
    for(i=0;i<NUMBER_OF_RESOURCES;++i)
        printf("resource[%d]=%d ", i, available[i]);
```

```
    printf("\n\n");

    printf("---------------------Maximum----------------------\n");
    for(i=0;i<NUMBER_OF_CUSTOMERS;++i)
    {
        printf("custom[%d]   ", i);
        for(j=0;j<NUMBER_OF_RESOURCES;++j)
            printf("resource[%d]=%d ", j, maximum[i][j]);
        printf("\n");
    }
    printf("\n");

    printf("---------------------Allocation--------------------\n");
    for(i=0;i<NUMBER_OF_CUSTOMERS;++i)
    {
        printf("custom[%d]   ", i);
        for(j=0;j<NUMBER_OF_RESOURCES;++j)
            printf("resource[%d]=%d ", j, allocation[i][j]);
        printf("\n");
    }
    printf("\n");

    printf("-----------------------Need-----------------------\n");
    for(i=0;i<NUMBER_OF_CUSTOMERS;++i)
    {
        printf("custom[%d]   ", i);
        for(j=0;j<NUMBER_OF_RESOURCES;++j)
            printf("resource[%d]=%d ", j, need[i][j]);
        printf("\n");
    }
    printf("\n");
    printf("-----------------------End-----------------------\n");

}
```

Our function achieves the effect like this:

```
┌─(~/Documents/studyBox/大二下/CS307 操作系统/project_github/lab6/code)──────────────────────────────────────────────(yanjieze@YanjieZedeMacBook-Pro:s000)─┐
└─(18:11:03 on master ● ★)──> ./banker                                                                                                          ──(六, 515)─┘
banker>*
-----------------Available Resource---------------
resource[0]=10 resource[1]=10 resource[2]=10 resource[3]=10

---------------------Maximum----------------------
custom[0]   resource[0]=6 resource[1]=4 resource[2]=7 resource[3]=3
custom[1]   resource[0]=4 resource[1]=2 resource[2]=3 resource[3]=2
custom[2]   resource[0]=2 resource[1]=5 resource[2]=3 resource[3]=3
custom[3]   resource[0]=6 resource[1]=3 resource[2]=3 resource[3]=2
custom[4]   resource[0]=5 resource[1]=6 resource[2]=7 resource[3]=5

---------------------Allocation---------------------
custom[0]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0
custom[1]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0
custom[2]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0
custom[3]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0
custom[4]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0

-----------------------Need------------------------
custom[0]   resource[0]=6 resource[1]=4 resource[2]=7 resource[3]=3
custom[1]   resource[0]=4 resource[1]=2 resource[2]=3 resource[3]=2
custom[2]   resource[0]=2 resource[1]=5 resource[2]=3 resource[3]=3
custom[3]   resource[0]=6 resource[1]=3 resource[2]=3 resource[3]=2
custom[4]   resource[0]=5 resource[1]=6 resource[2]=7 resource[3]=5

-----------------------End------------------------
banker>
```

# 3 Program Results

## 3.1 RQ

We do some requests, and show the results.

```
┌─(~/Documents/studyBox/大二下/CS307 操作系统/project_github/lab6/code)──────────────────────────────────────────────(yanjieze@YanjieZedeMacBook-Pro:s000)─┐
└─(18:12:30 on master ● ★)──> ./banker                                                                                                          ──(六, 515)─┘
banker>RQ 0 1 1 1 1
Request is satisfied.
banker>RQ 1 1 1 1 1
Request is satisfied.
banker>RQ 2 1 1 1 1
Request is satisfied.
banker>RQ 3 2 2 2 2
Request is satisfied.
banker>RQ 4 2 2 2 2
Request is satisfied.
banker>RQ 5 2 2 2 2
Request Error: ID Exceed
Request Failed. Please Follow Rules.
banker>RQ 4 2 2 2 2
Request Makes Unsafe State!!!
Request Failed. Please Follow Rules.
banker>
```

After some requests, the more requests are making the state unsafe.

The status is:

```
banker>*
-----------------Available Resource---------------
resource[0]=3 resource[1]=3 resource[2]=3 resource[3]=3

---------------------Maximum----------------------
custom[0]   resource[0]=6 resource[1]=4 resource[2]=7 resource[3]=3
custom[1]   resource[0]=4 resource[1]=2 resource[2]=3 resource[3]=2
custom[2]   resource[0]=2 resource[1]=5 resource[2]=3 resource[3]=3
custom[3]   resource[0]=6 resource[1]=3 resource[2]=3 resource[3]=2
custom[4]   resource[0]=5 resource[1]=6 resource[2]=7 resource[3]=5

---------------------Allocation--------------------
custom[0]   resource[0]=1 resource[1]=1 resource[2]=1 resource[3]=1
custom[1]   resource[0]=1 resource[1]=1 resource[2]=1 resource[3]=1
custom[2]   resource[0]=1 resource[1]=1 resource[2]=1 resource[3]=1
custom[3]   resource[0]=2 resource[1]=2 resource[2]=2 resource[3]=2
custom[4]   resource[0]=2 resource[1]=2 resource[2]=2 resource[3]=2

-----------------------Need-----------------------
custom[0]   resource[0]=5 resource[1]=3 resource[2]=6 resource[3]=2
custom[1]   resource[0]=3 resource[1]=1 resource[2]=2 resource[3]=1
custom[2]   resource[0]=1 resource[1]=4 resource[2]=2 resource[3]=2
custom[3]   resource[0]=4 resource[1]=1 resource[2]=1 resource[3]=0
custom[4]   resource[0]=3 resource[1]=4 resource[2]=5 resource[3]=3

-----------------------End-----------------------
```

# 3.2 RL

We do some release, and show the status:

```
banker>RL 4 2 2 2 2
banker>RL 1 1 1 1 1
banker>STAT
banker>*
-----------------Available Resource---------------
resource[0]=3 resource[1]=3 resource[2]=3 resource[3]=3

---------------------Maximum----------------------
custom[0]   resource[0]=6 resource[1]=4 resource[2]=7 resource[3]=3
custom[1]   resource[0]=4 resource[1]=2 resource[2]=3 resource[3]=2
custom[2]   resource[0]=2 resource[1]=5 resource[2]=3 resource[3]=3
custom[3]   resource[0]=6 resource[1]=3 resource[2]=3 resource[3]=2
custom[4]   resource[0]=5 resource[1]=6 resource[2]=7 resource[3]=5

---------------------Allocation--------------------
custom[0]   resource[0]=1 resource[1]=1 resource[2]=1 resource[3]=1
custom[1]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0
custom[2]   resource[0]=1 resource[1]=1 resource[2]=1 resource[3]=1
custom[3]   resource[0]=2 resource[1]=2 resource[2]=2 resource[3]=2
custom[4]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0

-----------------------Need-----------------------
custom[0]   resource[0]=5 resource[1]=3 resource[2]=6 resource[3]=2
custom[1]   resource[0]=3 resource[1]=1 resource[2]=2 resource[3]=1
custom[2]   resource[0]=1 resource[1]=4 resource[2]=2 resource[3]=2
custom[3]   resource[0]=4 resource[1]=1 resource[2]=1 resource[3]=0
custom[4]   resource[0]=3 resource[1]=4 resource[2]=5 resource[3]=3

-----------------------End-----------------------
```

## 3.3 * command

We use `*` to display the status, though we have shown in the previous parts.

```
┌─(~/Documents/studyBox/大二下/CS307 操作系统/project_github/lab6/code)──────────────────────────────────────────────(yanjieze@YanjieZedeMacBook-Pro:s
000)─┐
└─(18:17:52 on master ● ★)──> ./banker                                                                                                  ──(六, 515)─┘
banker>*
-----------------Available Resource---------------
resource[0]=10 resource[1]=10 resource[2]=10 resource[3]=10

---------------------Maximum----------------------
custom[0]   resource[0]=6 resource[1]=4 resource[2]=7 resource[3]=3
custom[1]   resource[0]=4 resource[1]=2 resource[2]=3 resource[3]=2
custom[2]   resource[0]=2 resource[1]=5 resource[2]=3 resource[3]=3
custom[3]   resource[0]=6 resource[1]=3 resource[2]=3 resource[3]=2
custom[4]   resource[0]=5 resource[1]=6 resource[2]=7 resource[3]=5

---------------------Allocation--------------------
custom[0]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0
custom[1]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0
custom[2]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0
custom[3]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0
custom[4]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0

-----------------------Need------------------------
custom[0]   resource[0]=6 resource[1]=4 resource[2]=7 resource[3]=3
custom[1]   resource[0]=4 resource[1]=2 resource[2]=3 resource[3]=2
custom[2]   resource[0]=2 resource[1]=5 resource[2]=3 resource[3]=3
custom[3]   resource[0]=6 resource[1]=3 resource[2]=3 resource[3]=2
custom[4]   resource[0]=5 resource[1]=6 resource[2]=7 resource[3]=5

-----------------------End------------------------
```

## 3.4 end

Enter `end`, and finish the execution of the program.

```
┌─(~/Documents/studyBox/大二下/CS307 操作系统/project_github/lab6/code)──────────────────────────────────────────────(yanjieze@YanjieZedeMacBook-Pro:s
000)─┐
└─(18:18:22 on master ● ★)──> ./banker                                                                                                  ──(六, 515)─┘
banker>end
---------------Banker's Algorithm Finished-------------
┌─(~/Documents/studyBox/大二下/CS307 操作系统/project_github/lab6/code)──────────────────────────────────────────────(yanjieze@YanjieZedeMacBook-Pro:s
000)─┐
└─(18:18:24 on master ● ★)──>                                                                                                           ──(六, 515)─┘
```

# 4 Conclusion and Thoughts

In project 6, we use **C** program language to implement the banker's algorithm, an algorithm performing **deadlock avoidance** to avoid entering the unsafe state. By implementing the algotihm, we clearly know the process of the bankers' algorithm and the safety algorithm.

The whole program needs no more than 300 lines to fulfill, while it's very interesting to have a programming purpose and implement a toy project which has the basic functions to show us the execution of the algorithm.

In conclusion, we go deeper into the banker's algorithm which is introduced in OS lectures and practise our programming skills. It's great enjoyment.