Project 6

Name: 刘　涵　之
ID：519021910102

Project 6: Banker's Algorithm

For this project, you 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. Although the code examples that describe this project are illustrated in C, you may also develop a solution using Java.

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 target, int req[]);
void release_resources(int target, int release[]);
```

The **request_resources()** function should return 0 if successful and –1 if unsuccessful.

Your program will initially read in a file containing the maximum number of requests for each customer. Your program will then have the user enter commands responding to a request of resources, a release of resources, or the current values of the different data structures. Use the command '**RQ**' for requesting resources, '**RL**' for releasing resources, and '*' to output the values of the different data structures.

Design: My design for this task is shown as follows:

- I use four arrays to track the current state: `available`, `maximum`, `allocation`, `need`.
- The content of `maximum` will be initialized from the file `max`.
- **check_safe()** function will use the banker algorithm to check the current state.
- **output()** function will output the current state.
- When requesting resources, changes will be done if and only if it passes **check_safe()** function.
- When releasing resources, changes will be done immediately.

The code of `banker.c` is shown as follows.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#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];

int check_safe();
int request_resources(int target, int req[]);
void release_resources(int target, int release[]);

int request_resources(int target, int req[]) {
    for (int i = 0; i < NUMBER_OF_RESOURCES; i++){
        if (req[i] < 0) {
            fprintf(stdout, "[Error] Request less than 0! \n");
            return 1;
        }
        if (req[i] > need[target][i]) {
            fprintf(stdout, "[Error] Request more than need! \n");
            return 1;
        }
    }

    for (int i = 0; i < NUMBER_OF_RESOURCES; i++){
        available[i] -= req[i];
        allocation[target][i] += req[i];
        need[target][i] = maximum[target][i] - allocation[target][i];
    }

    if (check_safe() == 1) {
        // unsafe, rollback
        for (int i = 0; i < NUMBER_OF_RESOURCES; i++){
            available[i] += req[i];
            allocation[target][i] -= req[i];
            need[target][i] = maximum[target][i] - allocation[target][i];
        }
        fprintf(stdout, "[Unsafe] Denied. \n");
        return 1;
    } else {
        fprintf(stdout, "[Safe] Accepted. \n");
        return 0;
    }
}

void release_resources(int target, int release[]){
    for (int i = 0; i < NUMBER_OF_RESOURCES; i++){
        if (release[i] < 0) {
            fprintf(stdout, "[Error] Release less than 0! \n");
            return;
        }
        if (release[i] > allocation[target][i]) {
            fprintf(stdout, "[Error] Release more than allocated! \n");
            return;
        }
```

```
65      }
66
67      for (int i = 0; i < NUMBER_OF_RESOURCES; i++){
68          available[i] += release[i];
69          allocation[target][i] -= release[i];
70          need[target][i] = maximum[target][i] - allocation[target][i];
71      }
72      fprintf(stdout, "[Safe] Accepted. \n");
73      return;
74  }
75
76  int initialize(int argc, char *argv[]) {
77      for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
78          available[i] = atoi(argv[i]);
79
80      FILE *stream = fopen("max", "r");
81      for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
82          fscanf(stream, "%d", &maximum[i][0]);
83          for (int j = 1; j < NUMBER_OF_RESOURCES; j++) {
84              fscanf(stream, ",%d", &maximum[i][j]);
85          }
86      }
87      fclose(stream);
88
89      for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
90          for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
91              need[i][j] = maximum[i][j];
92              allocation[i][j] = 0;
93          }
94      return 0;
95  }
96
97  int check_safe() {
98      int used[NUMBER_OF_CUSTOMERS];
99      int available_[NUMBER_OF_RESOURCES];
100
101     for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
102         available_[i] = available[i];
103
104     for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
105         used[i] = 0;
106
107     for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
108         // find a customer to use
109         int safe = 0;
110         for (int j = 0; j < NUMBER_OF_CUSTOMERS; j++) {
111             if (used[j] == 0) {
112                 int can = 1;
113                 for (int res = 0; res < NUMBER_OF_RESOURCES; res ++) {
114                     if (need[j][res] > available_[res]) {
115                         can = 0;
```

```
                        break;
                    }
                }
                if (can == 1) {
                    used[j] = 1;
                    safe = 1; // customer j can be satisfied!
                    for (int res = 0; res < NUMBER_OF_RESOURCES; res ++) {
                        available_[res] += allocation[j][res];
                    }
                    break;
                }
            }
        }
        if (safe == 0) {
            return 1;
        }
    }
    return 0;
}

void output() {
    fprintf(stdout, "#####     Current State     #####\n");
    fprintf(stdout, "  Available: [ ");
    for (int i = 0; i < NUMBER_OF_RESOURCES - 1; i++)
        fprintf(stdout, "%d , ", available[i]);
    fprintf(stdout, "%d ]\n", available[NUMBER_OF_RESOURCES - 1]);

    fprintf(stdout, "  Maximum: ");
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        if (i != 0) fprintf(stdout, "           ");
        fprintf(stdout, "[ ");
        for (int j = 0; j < NUMBER_OF_RESOURCES - 1; j++) {
            fprintf(stdout, "%d , ", maximum[i][j]);
        }
        fprintf(stdout, "%d ]\n", maximum[i][NUMBER_OF_RESOURCES - 1]);
    }

    fprintf(stdout, "  Allocation: ");
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        if (i != 0) fprintf(stdout, "              ");
        fprintf(stdout, "[ ");
        for (int j = 0; j < NUMBER_OF_RESOURCES - 1; j++) {
            fprintf(stdout, "%d , ", allocation[i][j]);
        }
        fprintf(stdout, "%d ]\n", allocation[i][NUMBER_OF_RESOURCES - 1]);
    }

    fprintf(stdout, "  Need: ");
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        if (i != 0) fprintf(stdout, "        ");
        fprintf(stdout, "[ ");
```

```
        for (int j = 0; j < NUMBER_OF_RESOURCES - 1; j++) {
            fprintf(stdout, "%d , ", need[i][j]);
        }
        fprintf(stdout, "%d ]\n", need[i][NUMBER_OF_RESOURCES - 1]);
    }
}

int main(int argc, char *argv[]) {
    if (argc != NUMBER_OF_RESOURCES + 1) {
        fprintf(stdout, "[Error] Wrong input. \n");
        exit(1);
    }
    initialize(argc, argv + 1);
    if (check_safe() == 1) {
        fprintf(stdout, "[Error] Unsafe initial state. \n");
        exit(1);
    }

    while (1) {
        char s1[100];
        fprintf(stdout, "Banker>>");
        fscanf(stdin, "%s", s1);
        if (strcmp(s1, "*") == 0) {
            output();
            continue;
        } else
        if (strcmp(s1, "RQ") == 0) {
            int target;
            fscanf(stdin, "%d", &target);

            int req[NUMBER_OF_RESOURCES];
            for (int i = 0; i < NUMBER_OF_RESOURCES; i++){
                fscanf(stdin, "%d", &req[i]);
            }

            request_resources(target, req);
        } else
        if (strcmp(s1, "RL") == 0) {
            int target;
            fscanf(stdin, "%d", &target);

            int req[NUMBER_OF_RESOURCES];
            for (int i = 0; i < NUMBER_OF_RESOURCES; i++){
                fscanf(stdin, "%d", &req[i]);
            }

            release_resources(target, req);
        } else {
            fprintf(stdout, "[Error] Unexpected input! \n");
        }
    }
```

```
    output();
}
```

Makefile for this task is shown as follow.

```
CC=gcc
CFLAGS=-Wall

all: banker.o
        $(CC) $(CFLAGS) -o banker banker.o

banker.o: banker.c
        $(CC) $(CFLAGS) -c banker.c

clean:
        rm -rf *.o
        rm -rf banker
```

The `max` file used here is:

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

The execution result is shown as follow. All functions mentioned above have been tested successfully.

```
misaka@MS-BVZPMBEQIPCD:/mnt/c/Projects/OS_Project/Project6/project6$ make all
gcc -Wall -c banker.c
gcc -Wall -o banker banker.o
misaka@MS-BVZPMBEQIPCD:/mnt/c/Projects/OS_Project/Project6/project6$ ./banker 10 5 7 8
[Error] Unsafe initial state.
misaka@MS-BVZPMBEQIPCD:/mnt/c/Projects/OS_Project/Project6/project6$ ./banker 10 6 7 7
Banker>>*
#####     Current State     #####
  Available: [ 10 , 6 , 7 , 7 ]
  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 ]
```

图 1: Banker's Algorithm

```
Banker>>RQ 0 6 4 7 3
[Safe] Accepted.
Banker>>*
#####     Current State     #####
  Available: [ 4 , 2 , 0 , 4 ]
  Maximum: [ 6 , 4 , 7 , 3 ]
           [ 4 , 2 , 3 , 2 ]
           [ 2 , 5 , 3 , 3 ]
           [ 6 , 3 , 3 , 2 ]
           [ 5 , 6 , 7 , 5 ]
  Allocation: [ 6 , 4 , 7 , 3 ]
              [ 0 , 0 , 0 , 0 ]
              [ 0 , 0 , 0 , 0 ]
              [ 0 , 0 , 0 , 0 ]
              [ 0 , 0 , 0 , 0 ]
  Need: [ 0 , 0 , 0 , 0 ]
        [ 4 , 2 , 3 , 2 ]
        [ 2 , 5 , 3 , 3 ]
        [ 6 , 3 , 3 , 2 ]
        [ 5 , 6 , 7 , 5 ]
Banker>>RQ 1 4 2 0 2
[Safe] Accepted.
Banker>>RQ 1 4 4 4 4
[Error] Request more than need!
Banker>>RQ 3 0 0 0 2
[Safe] Accepted.
Banker>>R*
[Error] Unexpected input!
```

图 2: Banker's Algorithm

```
Banker>>RL 0 1 1 1 1
[Safe] Accepted.
Banker>>RQ 4 1 1 1 1
[Unsafe] Denied.
Banker>>*
#####     Current State     #####
  Available: [ 1 , 1 , 1 , 1 ]
  Maximum: [ 6 , 4 , 7 , 3 ]
           [ 4 , 2 , 3 , 2 ]
           [ 2 , 5 , 3 , 3 ]
           [ 6 , 3 , 3 , 2 ]
           [ 5 , 6 , 7 , 5 ]
  Allocation: [ 5 , 3 , 6 , 2 ]
              [ 4 , 2 , 0 , 2 ]
              [ 0 , 0 , 0 , 0 ]
              [ 0 , 0 , 0 , 2 ]
              [ 0 , 0 , 0 , 0 ]
  Need: [ 1 , 1 , 1 , 1 ]
        [ 0 , 0 , 3 , 0 ]
        [ 2 , 5 , 3 , 3 ]
        [ 6 , 3 , 3 , 0 ]
        [ 5 , 6 , 7 , 5 ]
```

图 3: Banker's Algorithm