Proj6 Banker‘s Algorithm

by 郑航 520021911347

1 Abstract

实现一个Banker’s Algorithm的模拟程序

功能：模拟一个bank，允许Customers申请或释放资源，当且仅当为Customers分配资源后系统仍处于安全状态时予以分配，即始终维持系统的安全状态，不允许破坏安全状态的操作被执行。可以与用户进行交互，允许用户随时为某个customer提交资源分配或释放的申请、查看当前资源分配状态或退出程序

算法的思路参考课本 8.6.3 即可，主要分以下六部分进行实现：

- 全局变量的设计
- main函数的设计与实现
- safe_check的实现
- request_resources的实现
- release_resources的实现
- print_state的实现

2 全局变量的设计

在本project中，我们总共设计了如下全局变量：

- num_of_customers：customers数量
- num_of_resources：resources数量
- int *available：表示各资源当前可用资源数
- int **maximum： 表示各customer对各资源的最大可能使用量
- int **allocation：表示各customer对各资源的当前分配量
- int **need：表示各customer对各资源的目前最大需求量
- int *args：用于RQ和RL命令中存储参数

- int *work, *finish：用于安全检测算法中的工具数组，为了避免每次进行动态内存分配，选择直接设计为全局变量

3 main函数的设计与实现

main函数主要分为两个部分：

- 初始化部分，依据命令行参数从文件读入数据，并检查初始状态是否是安全状态
- 循环命令执行部分，主体是一个while循环，每个循环读入一行命令并执行

3.1 初始化部分

主要分为两个部分：

- 依据命令行参数从指定文件读入数据，获取num_of_customers和num_of_resources，并对各数据结构进行动态内存分配和数据初始化，该部分由init函数来完成，主要内容如下：
  - 根据命令行参数数目，判断其是否有效，如是的话，argc-1即为num_of_resources，否则输出错误信息并退出程序
  - 根据命令行参数，对available进行动态内存分配和初始化
  - 只读模式打开init.txt文件，统计其行数，即可得到num_of_customers（利用fgets函数可每次读入文件中的一行）
  - 根据num_of_customers和num_of_resources对maximum，allocation，need分别进行动态内存分配，并再次打开init.txt读入数据，对maximum进行初始化，allocation全为0，need[i][j]=maximum[i][j]
  - 对work，finish和args进行动态内存分配，方便后续使用
- 检查初始状态是否为安全状态，是的话则输出提示文字，并开始等待命令；否则输出错误信息，打印当前状态并退出程序
  - 检查初始状态由safe_check()函数完成，该函数在当前状态安全时返回1，否则返回0

    safe_check()的实现后续会在 4 safe_check()的实现 中详细说明
  - 打印当前状态由print_state()函数完成

    print_state()的实现后续会在 7 print_state()的实现 中详细说明

该部分代码如下：

```c
// initialization
init(argc, argv);
if (!safe_check())
{
    printf("Error: initial state is unsafe!\n");
    print_state();
    return 1;
}
printf("Banker's Algorithm\n(input command \"q\" to quit the program)\n");
```

init()代码如下：

```c
void init(int argc, char *argv[])
{
    if (argc <= 1)
    {
        printf("Error: invalid arguments!\n");
        exit(1);
    }
    num_of_resources = argc - 1;

    // available initialization
    available = (int *)malloc(sizeof(int) * num_of_resources);
    for (int i = 1; i < argc; ++i)
    {
        available[i - 1] = atoi(argv[i]);
    }
    // fetch num_of_customers
    FILE *fp = fopen("init.txt", "r");
    char *tmp = (char *)(malloc(sizeof(char) * num_of_resources * 10));
    while (!feof(fp))
    {
        char *ret = fgets(tmp, num_of_resources * 10, fp);
        if (ret)
            num_of_customers++;
    }
    free(tmp);
    fclose(fp);

    maximum = (int **)malloc(sizeof(int *) * num_of_customers);
    allocation = (int **)malloc(sizeof(int *) * num_of_customers);
    need = (int **)malloc(sizeof(int *) * num_of_customers);
    for (int i = 0; i < num_of_customers; i++)
    {
        maximum[i] = (int *)malloc(sizeof(int) * num_of_resources);
        allocation[i] = (int *)malloc(sizeof(int) * num_of_resources);
        need[i] = (int *)malloc(sizeof(int) * num_of_resources);
    }

    // read data into maximum
    fp = fopen("init.txt", "r");
    for (int i = 0; i < num_of_customers; i++)
    {
        int j = 0;
        fscanf(fp, "%d", &(maximum[i][j]));
        for (j = 1; j < num_of_resources; j++)
        {
            fscanf(fp, ",%d", &(maximum[i][j]));
        }
    }
    fclose(fp);

    // initialization of allocation and need
    for (int i = 0; i < num_of_customers; i++)
    {
        for (int j = 0; j < num_of_resources; j++)
        {
            allocation[i][j] = 0;
            need[i][j] = maximum[i][j];
        }
    }
    // initialization of work and finish (prepared for safe_check)
    work = (int *)malloc(sizeof(int) * num_of_resources);
    finish = (int *)malloc(sizeof(int) * num_of_customers);

    args = (int *)malloc(sizeof(int) * (num_of_resources + 1));
```

```
}
```

3.2 循环命令执行部分

一个条件恒为true的while循环中，首先利用函数read_cmd()读入新的一行命令并判断命令类型，根据不同的返回值，分别执行switch语句中的一个case，对指令进行执行

- 其中，read_cmd()的返回值存入一个int类型的变量mode中，mode值和命令类型有如下对应关系：
  - mode 0 represents "RQ"
  - mode 1 represents "RL"
  - mode 2 represents "*"
  - mode 3 represents "quit"
  - mode -1 represents invalid command

  read_cmd()的实现较为简单，主要就是读入输入的命令并利用一系列条件判断，判断命令类型；若是RQ或RL命令还需要将后续的参数暂时存入args中。
- read_cmd()后需要利用如下语句，对缓冲区中多余的字符读出删去

  ```
  scanf("%*[^\n]%*c");
  ```

- case mode=0：

  RQ命令，调用request_resources() 函数，并对返回结果进行判断，若返回0代表申请成功，打印成功的提示信息；返回-1表示申请失败，不会执行该次分配，并打印对应的错误信息

  request_resources()的具体实现在 5 request_resources()的实现 中详细说明
- case mode=1：

  RL命令，调用release_resources() 函数，并对返回结果进行判断，若返回0代表申请成功，打印成功的提示信息；返回-1表示申请失败，不会执行该次释放，并打印错误信息

  release_resources()的具体实现在 6 release_resources()的实现 中中详细说明
- case mode=2：

  *命令，调用print_state() 函数打印当前状态

  print_state()的具体实现在 7 print_state()的实现 中详细说明
- case mode=3：

  quit命令，调用clean() 函数完成资源释放等工作后，打印退出成功的信息，并直接return 0表示程序正常退出即可
- default 情况：

  表示程序接受到了无效的命令，会打印Error信息后break，进入下一次循环重新等待新的命令

read_cmd()具体代码如下：

```
/*
mode 0 represents "RQ"
mode 1 represents "RL"
mode 2 represents "*"
mode 3 represents "quit"
mode -1 represents invalid command
*/
int read_cmd(void)
{
```

```c
    printf("cmd >>  ");
    char cmd[2];
    scanf("%c", &cmd[0]);
    if (cmd[0] == '\n')
        scanf("%c", &cmd[0]);

    if (cmd[0] == '*')
    {
        return 2;
    }
    else if (cmd[0] == 'R')
    {
        scanf("%c", &cmd[1]);
        if (cmd[1] != 'Q' && cmd[1] != 'L')
            return -1;
    }
    else if (cmd[0] == 'q')
        return 3;
    else
        return -1;

    // RQ & RL
    for (int i = 0; i < num_of_resources + 1; i++)
        scanf(" %d", &args[i]);
    if (cmd[1] == 'Q')
        return 0;
    else
        return 1;
}
```

clean()具体代码如下：

```c
void clean(void)
{
    for (int i = 0; i < num_of_customers; i++)
    {
        free(maximum[i]);
        free(allocation[i]);
        free(need[i]);
    }
    free(maximum);
    free(allocation);
    free(need);
    free(available);
    free(args);
    free(work);
    free(finish);
}
```

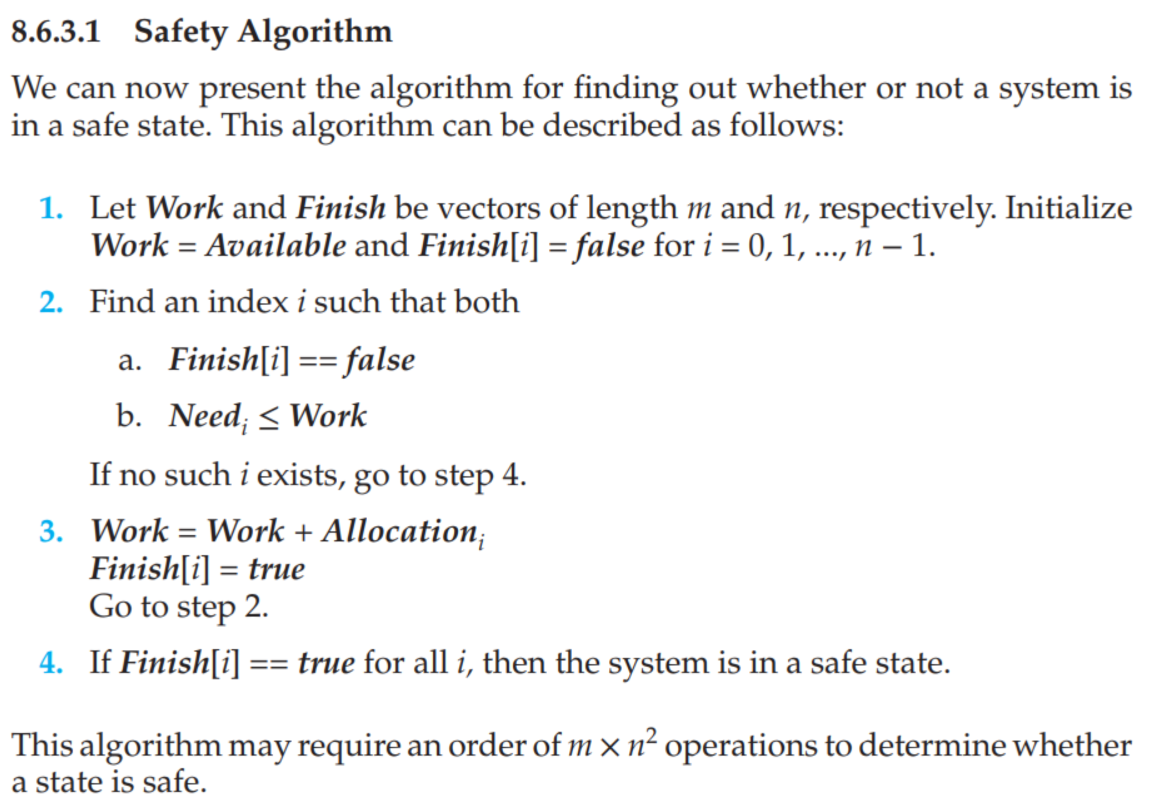
main()具体代码如下：

```c
int main(int argc, char *argv[])
{
    // initialization
    init(argc, argv);
    if (!safe_check())
    {
        printf("Error: initial state is unsafe!\n");
        print_state();
        return 1;
    }
    printf("Banker's Algorithm\n(input command \"q\" to quit the program)\n");

    while (TRUE)
    {
        int mode = read_cmd();
        scanf("%*[^\n]%*c");
        int ret;

        switch (mode)
        {
        // RQ
        case 0:
        {
            ret = request_resources(args[0], args + 1);
            if (!ret)
            {
                printf("The resources have been successfully allocated!\n");
            }
            break;
        }
        // RL
        case 1:
        {
            ret = release_resources(args[0], args + 1);
            if (ret)
            {
                printf("Error: The released resources exceed that has been allocated for customer %d!\n", args[0]);
            }
            else
            {
                printf("The resources have been successfully released!\n");
            }
            break;
        }
        //*
        case 2:
        {
            print_state();
            break;
        }
        // quit
        case 3:
        {
            printf("quit successfully!\n");
            clean();
            return 0;
            break;
        }
        // error
        default:
        {
```

```c
                printf("Error:invalid command!\n");
                break;
            }
        }
    }
    return 0;
}
```

4 safe_check的实现

safe_check() 的实现完全以课本给的思路为参考，具体如下：



8.6.3.1 Safety Algorithm

We can now present the algorithm for finding out whether or not a system is in a safe state. This algorithm can be described as follows:

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
   a. *Finish*[i] == *false*
   b. $Need_i \leq Work$

   If no such i exists, go to step 4.
3. *Work* = *Work* + $Allocation_i$
   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.

根据该思路，执行如下步骤：

- 按照step1的要求对work和finish进行初始化
- 执行step2，可能需要三层循环，利用一个简单的操作可以使得只需要写两层循环：每次找到一个可用的i，跳出执行step3，同时将i重新置为0（下次重新从头开始寻找）
- 执行step3，进行work和finish的赋值
- 执行step4，利用一个unsafe_flag作为标志，遍历finish，存在finish[i]==0则将unsafe_flag置为1表不安全，否则unsafe_flag为0

safe_check() 具体代码如下：

```c
// return 1 represents safe, 0 represents unsafe
int safe_check(void)
{
    // step 1
    for (int i = 0; i < num_of_resources; i++)
        work[i] = available[i];
    for (int i = 0; i < num_of_customers; i++)
```

```
        finish[i] = 0;

    for (int i = 0; i < num_of_customers; i++)
    {
        // step 2
        if (finish[i])
            continue;
        int flag = 0;
        for (int j = 0; j < num_of_resources; j++)
        {
            if (need[i][j] > work[j])
                flag = 1;
        }
        if (flag)
            continue;

        // step 3
        for (int j = 0; j < num_of_resources; j++)
        {
            work[j] += allocation[i][j];
            finish[i] = 1;
            i = 0;
        }
    }
    // step 4
    int unsafe_flag = 0;
    for (int i = 0; i < num_of_customers; i++)
    {
        if (finish[i] == 0)
            unsafe_flag = 1;
    }
    return !unsafe_flag;
}
```

5 request_resources的实现

request_resources()的实现也参考课本内Resource-Request Algorithm的思路，主要分以下几个步骤：

- 分配资源并判断该分配是否合法，交由工具函数decrease_resources()来实现，函数功能为当前状态基础上，直接减去request_resources传入的申请对应的资源量，即无条件进行资源分配，并返回该资源分配的合法性情况

  资源分配有两种可能的不合法情况：

  - customer[i]申请的资源量超过其need的资源量，即假如分配成功，则customer[i]的资源量会超过其可能的最大申请资源量maximum。decrease_resources()返回1表示此种不合法情况
  - customer[i]申请的资源量超过当前可以分配的最大资源量available，即系统没有这么多的资源可供分配给customer[i]。decrease_resources()返回2表示此种不合法情况

  我们将其他情况都定义为合法的情况，此时decrease_resources()返回0

- 若资源分配不合法，打印对应的不合法信息；否则对当前状态进行安全状态检查，即调用safe_check()，若当前状态安全则返回0

- 若资源分配不合法或者合法但资源分配后系统不处于安全状态，则调用工具函数increase_resources()重新将该部分资源释放

increase_resources()的功能就是无条件地将传入的参数对应的资源进行释放

工具函数 **decrease_resources()**具体代码如下:

```c
/*
return 0 represents successful
return 1 represents allocation[i][j]>maximum[i][j]
return 2 represents request[i]>available[i]
*/
int decrease_resources(int customer_num, int resources[])
{
    for (int i = 0; i < num_of_resources; i++)
    {
        allocation[customer_num][i] += resources[i];
        need[customer_num][i] -= resources[i];
        available[i] -= resources[i];
    }
    int out_maximum = 0;
    int out_available = 0;
    for (int i = 0; i < num_of_resources; i++)
    {
        if (need[customer_num][i] < 0)
            out_maximum = 1;
        if (available[i] < 0)
            out_available = 1;
    }
    if (out_maximum)
        return 1;
    if (out_available)
        return 2;
    return 0;
}
```

工具函数 **increase_resources()**具体代码如下:

```c
void increase_resources(int customer_num, int resources[])
{
    for (int i = 0; i < num_of_resources; i++)
    {
        allocation[customer_num][i] -= resources[i];
        need[customer_num][i] += resources[i];
        available[i] += resources[i];
    }
}
```

request_resources()具体代码如下:

```c
int request_resources(int customer_num, int request[])
{
    int ret = decrease_resources(customer_num, request);
    if (ret == 0)
    {
        if (safe_check())
            return 0;
        else
```

```c
            printf("Request failed! The request from customer %d will lead to an unsafe state.\n", customer_num);
    }
    else if (ret == 1)
    {
        printf("Error: The requested resources exceed customer %d's needed resources!\n", customer_num);
    }
    else if (ret == 2)
    {
        printf("Error: The requested resources exceed the currently available ones!\n");
    }

    increase_resources(customer_num, request);
    return -1;
}
```

6 release_resources的实现

　　release_resources()的实现较为简单，只需要首先检查一下申请release的资源是否合法，若合法则直接调用 5 request_resources的实现 中的工具函数increase_resources()进行资源释放，并返回0表示释放成功即可；若不合法则不予释放，并返回1表示释放失败

　　其中，不合法的可能情况只有一种，即customer[i]试图申请释放超过其被分配资源量（即allocation[i]）的资源

release_resources()具体代码如下：

```c
int release_resources(int customer_num, int release[])
{
    // check
    for (int i = 0; i < num_of_resources; ++i)
        if (release[i] > allocation[customer_num][i])
        {
            return 1;
        }

    increase_resources(customer_num, release);
    return 0;
}
```

7 print_state的实现

print_state()函数就是简单的打印当前的状态即可，我们打印的信息包括

- Number of customers
- Number of resources
- Available
- Maximum
- Allocation
- Need

print_state()具体代码如下：

```c
void print_state(void)
{
    int i, j;
    printf("The current state is shown as below:\n\n");
    printf("Number of customers:\t%d\n",num_of_customers);
    printf("Number of resources:\t%d\n\n",num_of_resources);
    printf("Available:\n\t\t[ ");
    for (i = 0; i < num_of_resources - 1; i++)
    {
        printf("%d, ", available[i]);
    }
    printf("%d ]\n", available[i]);
    printf("\nMaximum:\n");
    for (i = 0; i < num_of_customers; i++)
    {
        printf("customer %d :\t[ ", i);
        for (j = 0; j < num_of_resources - 1; j++)
        {
            printf("%d, ", maximum[i][j]);
        }
        printf("%d ]\n", maximum[i][j]);
    }
    printf("\nAllocation:\n");
    for (i = 0; i < num_of_customers; i++)
    {
        printf("customer %d :\t[ ", i);
        for (j = 0; j < num_of_resources - 1; j++)
        {
            printf("%d, ", allocation[i][j]);
        }
        printf("%d ]\n", allocation[i][j]);
    }
    printf("\nNeed:\n");
    for (i = 0; i < num_of_customers; i++)
    {
        printf("customer %d :\t[ ", i);
        for (j = 0; j < num_of_resources - 1; j++)
        {
            printf("%d, ", need[i][j]);
        }
        printf("%d ]\n", need[i][j]);
    }
    printf("\n");
}
```

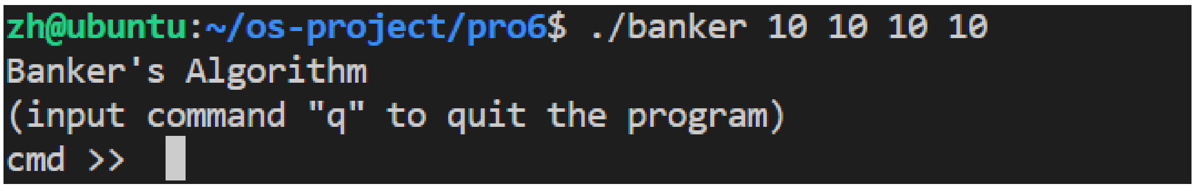
8 运行结果

init.txt文件内容如下：

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

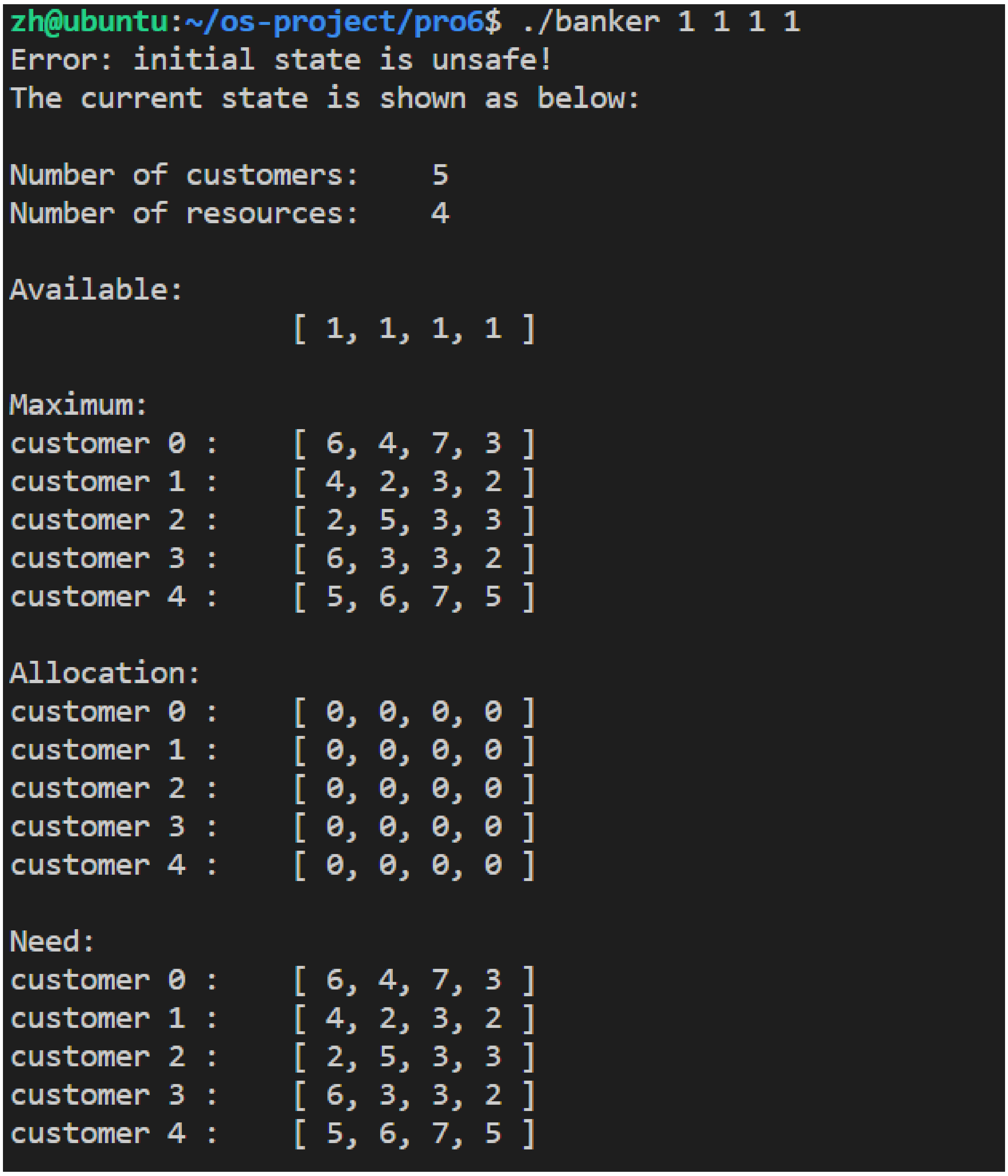
首先测试对于初始状态的safe情况判断：

```
./banker 10 10 10 10
```



```
zh@ubuntu:~/os-project/pro6$ ./banker 10 10 10 10
Banker's Algorithm
(input command "q" to quit the program)
cmd >>
```

```
./banker 1 1 1 1
```



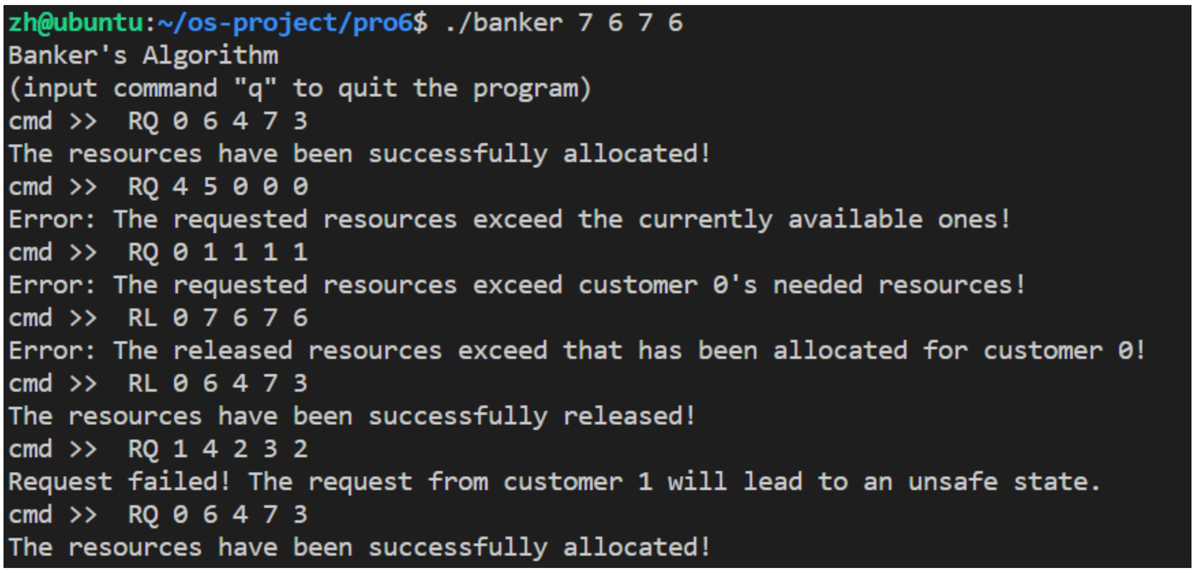
```
zh@ubuntu:~/os-project/pro6$ ./banker 1 1 1 1
Error: initial state is unsafe!
The current state is shown as below:

Number of customers:    5
Number of resources:    4

Available:
                [ 1, 1, 1, 1 ]

Maximum:
customer 0 :    [ 6, 4, 7, 3 ]
customer 1 :    [ 4, 2, 3, 2 ]
customer 2 :    [ 2, 5, 3, 3 ]
customer 3 :    [ 6, 3, 3, 2 ]
customer 4 :    [ 5, 6, 7, 5 ]

Allocation:
customer 0 :    [ 0, 0, 0, 0 ]
customer 1 :    [ 0, 0, 0, 0 ]
customer 2 :    [ 0, 0, 0, 0 ]
customer 3 :    [ 0, 0, 0, 0 ]
customer 4 :    [ 0, 0, 0, 0 ]

Need:
customer 0 :    [ 6, 4, 7, 3 ]
customer 1 :    [ 4, 2, 3, 2 ]
customer 2 :    [ 2, 5, 3, 3 ]
customer 3 :    [ 6, 3, 3, 2 ]
customer 4 :    [ 5, 6, 7, 5 ]
```

接下来测试各个命令的工作情况：

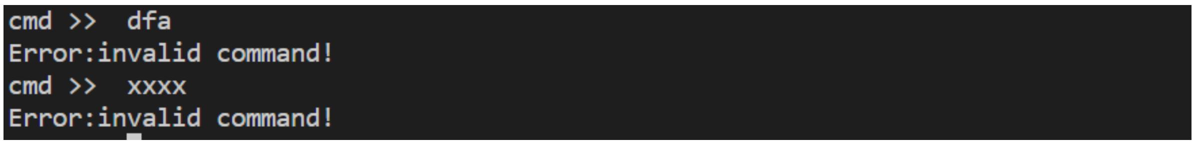
命令如下：

（可以测试所有可能情况）

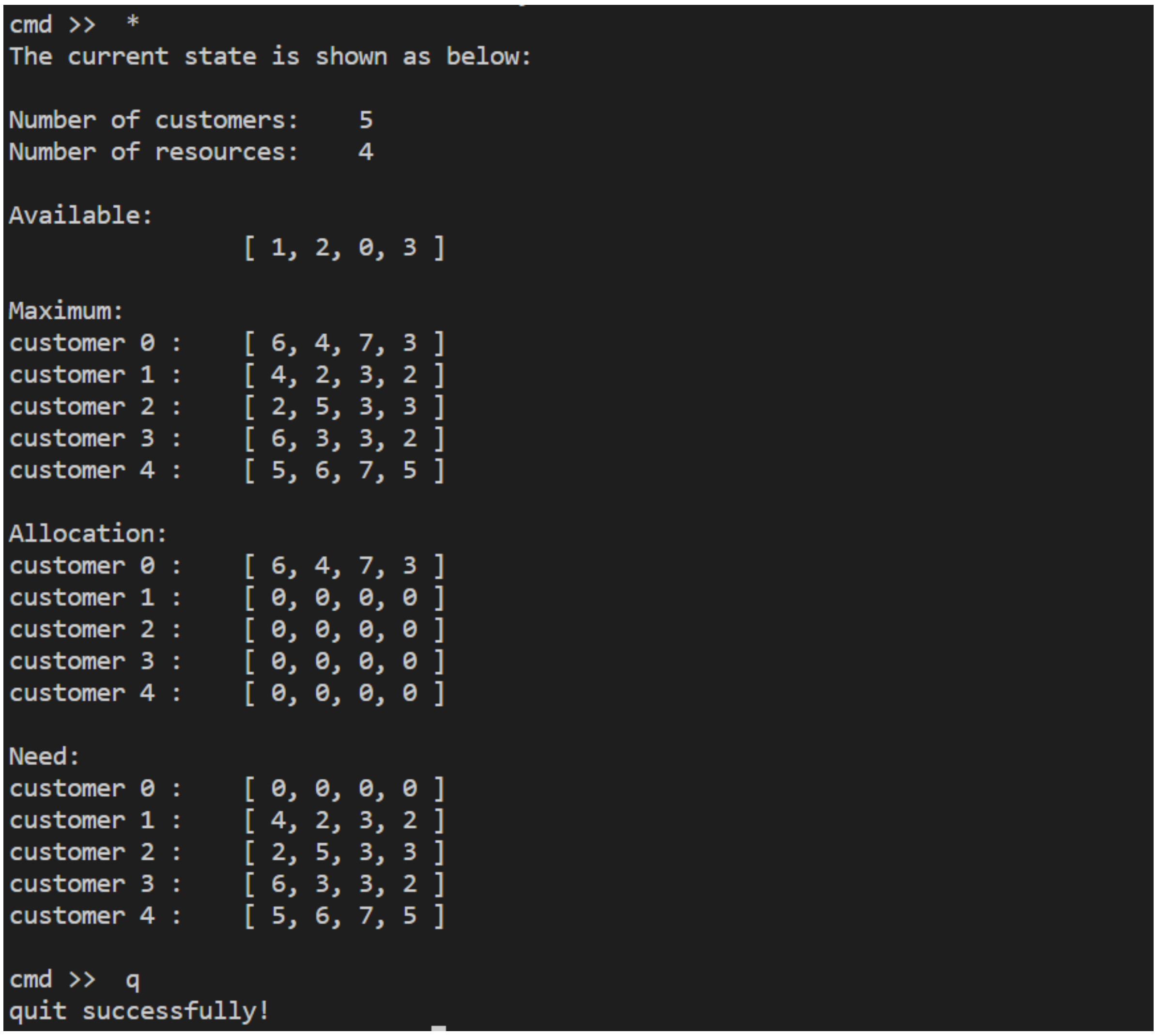
```
./banker 7 6 7 6

RQ 0 6 4 7 3
RQ 4 5 0 0 0
RQ 0 1 1 1 1
RL 0 7 6 7 6
RL 0 6 4 7 3
RQ 1 4 2 3 2
RQ 0 6 4 7 3
dfa
xxxx
*
q
```



```
zh@ubuntu:~/os-project/pro6$ ./banker 7 6 7 6
Banker's Algorithm
(input command "q" to quit the program)
cmd >>  RQ 0 6 4 7 3
The resources have been successfully allocated!
cmd >>  RQ 4 5 0 0 0
Error: The requested resources exceed the currently available ones!
cmd >>  RQ 0 1 1 1 1
Error: The requested resources exceed customer 0's needed resources!
cmd >>  RL 0 7 6 7 6
Error: The released resources exceed that has been allocated for customer 0!
cmd >>  RL 0 6 4 7 3
The resources have been successfully released!
cmd >>  RQ 1 4 2 3 2
Request failed! The request from customer 1 will lead to an unsafe state.
cmd >>  RQ 0 6 4 7 3
The resources have been successfully allocated!
```



```
cmd >>  dfa
Error:invalid command!
cmd >>  xxxx
Error:invalid command!
```



```
cmd >>  *
The current state is shown as below:

Number of customers:    5
Number of resources:    4

Available:
                [ 1, 2, 0, 3 ]

Maximum:
customer 0 :    [ 6, 4, 7, 3 ]
customer 1 :    [ 4, 2, 3, 2 ]
customer 2 :    [ 2, 5, 3, 3 ]
customer 3 :    [ 6, 3, 3, 2 ]
customer 4 :    [ 5, 6, 7, 5 ]

Allocation:
customer 0 :    [ 6, 4, 7, 3 ]
customer 1 :    [ 0, 0, 0, 0 ]
customer 2 :    [ 0, 0, 0, 0 ]
customer 3 :    [ 0, 0, 0, 0 ]
customer 4 :    [ 0, 0, 0, 0 ]

Need:
customer 0 :    [ 0, 0, 0, 0 ]
customer 1 :    [ 4, 2, 3, 2 ]
customer 2 :    [ 2, 5, 3, 3 ]
customer 3 :    [ 6, 3, 3, 2 ]
customer 4 :    [ 5, 6, 7, 5 ]

cmd >>  q
quit successfully!
```

可以看到，程序正确地给出了所需的结果

9 Summary

本次通过编写模拟banker's algorithm的程序，对课内内容进行实践，使得我对该算法有了更为深入的理解。同时，本次project函数数目较多，需要好好组织代码结构并且合理添加注释，才可以使得代码可读性更强一些。此外，本次project需要用到诸如文件处理，字符串分解等一系列方法，虽然并不是很难，但是编写代码的过程也锻炼了我的编程能力。总之，本次project难度不算很大，而且完成过程也富有乐趣（比如调整打印信息以及拆分出一些工具函数使得代码结构更加清晰等等的过程），收获良多。