CS356 Project 6

Name : HaotianXue

Student ID : 518021910506

Email : xavihart@sjtu.edu.cn

Introduction

This is the report for project6 for CS356, also for COS book chapter8.

The project encouraged us to implement **Banker's Algorithm** and simulate the resource allocation process and safe-checking strategy.

The implementation is in C in Linux, we first initialize the status and fetch input from users in command to do relative operations.

General Ideas

Important Variables

Four matrices(array) is important for us to do the Banker's Algorithm:

- maximum : maximum number of instances different process needed for different resources
- need : temporary number of instances different process needed for different resources
- allocation : temporary number of instances different process allocated for different resources
- avail : available number of instances

It should be noted that they are actually not mutual independent with each other, since it is obvious that:

$$\forall (valid) i, j : need[i][j] + allocation[i][j] = maximum[i][j]$$

User Commands

The user input includes three types:

- `RQ` is used to request resources for certain process, so it should be followed by the process ID and needed instance number for each resources.
- `RL` is used to release resources
- `*` is used to show current status for the four matrices
- `exit` is used to break out of the program

If the input is `RQ`, we should use Banker's Algorithm to check is the state is safe is the resources is truly allocated, if not, we ignore that operation, if safe, we make the allocation.

If the input is `RL`, we should check if the process can free the number of instances, which means the allocation should be no more than the release number for each resources.

Code Structure

We store the maximum matrix in `req.txt`. We have three key functions: `release()`, `request()` and `check()`.

The simplest one is `release()`, where we just check if the release is sensible and to the release if it is.

For `request()` we should first generate a copy of the matrices assuming that the request is served, then we use `check()` to check if the allocation is safe. If not, we return back to the original status.

For `check` function, we just follow the Bankers Algorithm.

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

#define NUM_CONS 5
#define NUM_RES 4

int available[NUM_RES];
int maximum[NUM_CONS][NUM_RES];
int allocation[NUM_CONS][NUM_RES];
int need[NUM_CONS][NUM_RES];

char command[5];
int cosID;
int reqlist[NUM_RES];

int request(int , int[]);
int release(int, int[]);
```

```
int check(int [][NUM_RES], int [][NUM_RES], int []);
void init();
void show_mat();

int main(int argc, char *argv[]){

    // program inputs

    assert(argc == 5);
     //printf("Banker>>\n");
    for (int i = 0; i < NUM_RES;++i){
        //printf("%d\n", i);
        available[i] = atoi(argv[i + 1]);
    }

    FILE *fp;
    if ((fp = fopen("req.txt", "r")) == NULL){
        printf("File cannot open!\n");
        return 0;
    }

    for (int i = 0; i < NUM_CONS; ++i){
        for (int j = 0; j < NUM_RES; ++j){
            fscanf(fp, "%d", &maximum[i][j]);
            fgetc(fp);
        }
    }

    // initialize matrices

    init();

    // get user input

    while(1){
        printf("Banker>>");
        scanf("%s", &command);
        if(strcmp(command, "exit") == 0){
            break;
        }
        if(strcmp(command, "*") == 0){
            show_mat();
            continue;
        }
        scanf("%d %d %d %d %d", &cosID, &reqlist[0],
&reqlist[1], &reqlist[2], &reqlist[3]);
        if(strcmp(command, "RQ") == 0){
             request(cosID, reqlist);
        }
        else if(strcmp(command, "RL") == 0){
             release(cosID, reqlist);
        }
```

```
        else{
            printf("Wrong input(only RQ,RL,* are
acceptable)\n");
        }
    }

    return 0;
}


int request(int ID, int REQ[]){
    if(ID < 0 || ID > NUM_CONS - 1){
        printf("ConsID not exist!\n");
        return -1;
    }
    for(int i = 0;i < NUM_RES;++i){
        if(need[ID][i] < REQ[i]){
            printf("ERROR: request bigger than need!\n");
            return -1;
        }
    }

    // make a copy of the mat and check if they are in
safe state
    int avail_cp[NUM_RES];
    int need_cp[NUM_CONS][NUM_RES];
    int alloc_cp[NUM_CONS][NUM_RES];
    for(int i = 0;i < NUM_CONS;++i)
        for(int j = 0;j < NUM_RES;++j){
            need_cp[i][j] = need[i][j];
            alloc_cp[i][j] = allocation[i][j];
        }

    for(int i = 0;i < NUM_RES;++i){
        avail_cp[i] = available[i] - REQ[i];
        need_cp[ID][i] = need[ID][i] - REQ[i];
        alloc_cp[ID][i] = allocation[ID][i] + REQ[i];
    }

    int f;
    f = check(alloc_cp, need_cp, avail_cp);
    if(f == 0){
        printf("Request permitted!\n");
        for(int i = 0;i < NUM_RES;++i){
            available[i] -= REQ[i];
            need[ID][i] -= REQ[i];
            allocation[ID][i] += REQ[i];
        }
    }else{
        printf("Requset unsafe!\n");
    }
```

```
}
int release(int ID, int REQ[]){
    // return 0 if success, else -1
    if(ID < 0 || ID > NUM_CONS - 1){
        printf("ConsID not exist!\n");
        return -1;
    }
    for(int i = 0;i < NUM_RES;++i){
        if(allocation[ID][i] < REQ[i]){
            printf("ERROR: release bigger than alloc!\n");
            return -1;
        }
    }
    for(int i = 0;i < NUM_RES;++i){
        allocation[ID][i] -= REQ[i];
        available[i] += REQ[i];
        need[ID][i] += REQ[i];
    }
    return 0;
}

int check(int Alloc[][NUM_RES], int Need[][NUM_RES], int
avail[]){
    // return 0 if safe, else -1
    int alldone;
    int solveone = 0;
    int done[NUM_CONS] = {0,0,0,0,0};
    while(1){
        alldone = 1;
        solveone = 0;

        for(int i = 0;i < NUM_CONS;++i){
            if(done[i] == 1){
                continue;
            }else{
                alldone = 0;
                int f = 0;
                for(int j = 0;j < NUM_RES;++j){
                    if(avail[j] < Need[i][j]){
                        //printf("%d ",  Need[i][j]);
                        f = 1;
                        break;
                    }
                }
                if(!f){
                    solveone = 1;
                    //printf("solve %d\n", i);
                    for(int j = 0;j < NUM_RES;++j){
                        done[i] = 1;
                        avail[j] += Alloc[i][j];
                    }
                }else{
```

```
                    continue;
                }
            }
        }
        if(alldone) return 0;
        if(solveone == 0) return -1;
    }
}



void init(){
    for(int i = 0;i < NUM_CONS;++i){
        for(int j = 0;j < NUM_RES;++j){
            allocation[i][j] = 0;
            need[i][j] = maximum[i][j];
        }
    }
}

void show_mat(){
    printf("\n");
    printf("------Maximum Mat------\n");
    printf("   R1 R2 R3 R4\n");
    for(int i = 0;i < NUM_CONS;++i){
        printf("T%d ", i);
        for(int j = 0;j < NUM_RES;++j){
            printf("%d  ", maximum[i][j]);
        }
         printf("\n");
    }
    printf("\n");
    printf("------Need Mat------\n");
    printf("   R1 R2 R3 R4\n");
    for(int i = 0;i < NUM_CONS;++i){
        printf("T%d ", i);
        for(int j = 0;j < NUM_RES;++j){
            printf("%d  ", need[i][j]);
        }
         printf("\n");
    }
    printf("\n");
    printf("------Alloc Mat------\n");
    printf("   R1 R2 R3 R4\n");
    for(int i = 0;i < NUM_CONS;++i){
        printf("T%d ", i);
        for(int j = 0;j < NUM_RES;++j){
            printf("%d  ", allocation[i][j]);
        }
         printf("\n");
    }
    printf("\n");
```

```
    printf("------Avail List------\n");
    printf("R1 R2 R3 R4\n");
    for(int i = 0;i < NUM_RES;++i){
        printf(" %d", available[i]);
    }
    printf("\n");
}
```

Evaluation

We set the number of process to be 5 and the number of resources to be 4, and the maximum matrix is as follows:

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

Show Matrices

```
chrisxue@chrisxue-VirtualBox:~/OS-Lab/proj6$ ./run 6 6 7 5
Banker>>*

------Maximum Mat------
   R1 R2 R3 R4
T0 6  4  7  3
T1 4  2  3  2
T2 2  5  3  3
T3 6  3  3  2
T4 5  6  7  5

------Need Mat------
   R1 R2 R3 R4
T0 6  4  7  3
T1 4  2  3  2
T2 2  5  3  3
T3 6  3  3  2
T4 5  6  7  5

------Alloc Mat------
   R1 R2 R3 R4
T0 0  0  0  0
T1 0  0  0  0
T2 0  0  0  0
T3 0  0  0  0
T4 0  0  0  0

------Avail List------
R1 R2 R3 R4
 6 6 7 5
```

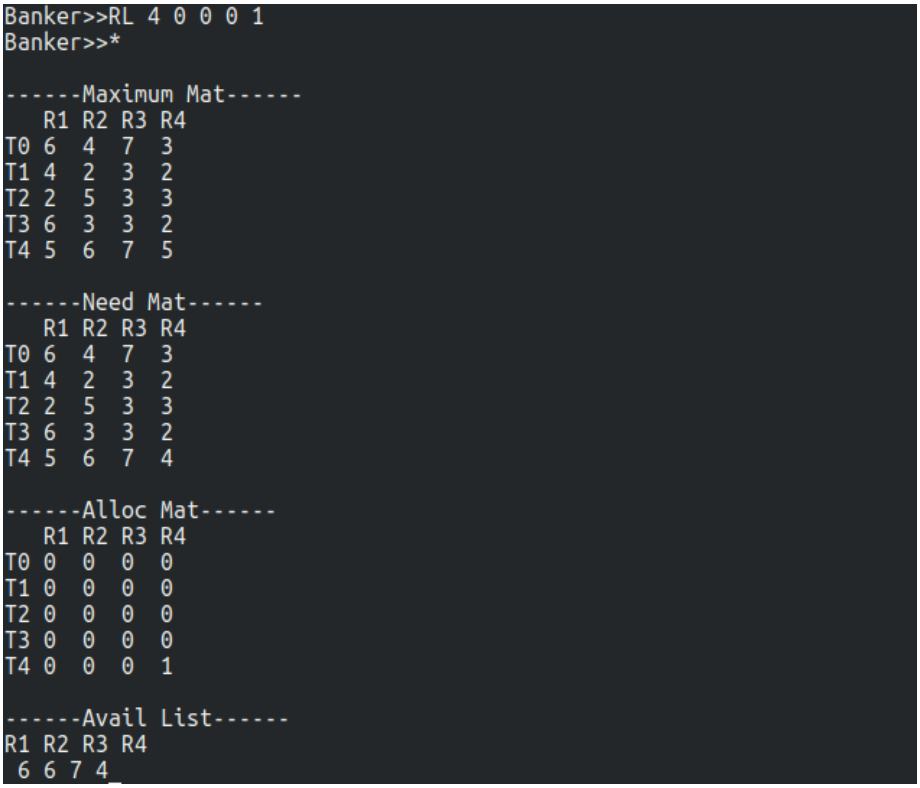
Validation Detection for Input

```
Banker>>RQ 0 7 6 5 4
ERROR: request bigger than need!
Banker>>RL 7 1 1 1 1
ConsID not exist!
Banker>>RL 4 1 1 1 8
ERROR: release bigger than alloc!
Banker>>RQ 0 3 3 3 3
Request permitted!
```

Request and Release

```
Banker>>RQ 4 0 0 0 5
Request permitted!
Banker>>*

------Maximum Mat------
   R1 R2 R3 R4
T0 6  4  7  3
T1 4  2  3  2
T2 2  5  3  3
T3 6  3  3  2
T4 5  6  7  5

------Need Mat------
   R1 R2 R3 R4
T0 6  4  7  3
T1 4  2  3  2
T2 2  5  3  3
T3 6  3  3  2
T4 5  6  7  0

------Alloc Mat------
   R1 R2 R3 R4
T0 0  0  0  0
T1 0  0  0  0
T2 0  0  0  0
T3 0  0  0  0
T4 0  0  0  5

------Avail List------
R1 R2 R3 R4
 6 6 7 0
Banker>>RQ 0 6 6 7 0
ERROR: request bigger than need!
Banker>>RQ 0 6 4 7 0
Requset unsafe!
```

```
------Maximum Mat------
   R1 R2 R3 R4
T0 6  4  7  3
T1 4  2  3  2
T2 2  5  3  3
T3 6  3  3  2
T4 5  6  7  5

------Need Mat------
   R1 R2 R3 R4
T0 6  4  7  3
T1 4  2  3  2
T2 2  5  3  3
T3 6  3  3  2
T4 5  6  7  3

------Alloc Mat------
   R1 R2 R3 R4
T0 0  0  0  0
T1 0  0  0  0
T2 0  0  0  0
T3 0  0  0  0
T4 0  0  0  2

------Avail List------
R1 R2 R3 R4
 6 6 7 3
Banker>>RL 4 0 0 0 1
```



Conclusion

This project helps us have better understanding of the Banker's Algorithm and its importance in avoid deadlock. The implementation can simulate the the resource allocation process and safe-checking strategy well. And I also gain a lot of experience in implementing this work.

Thanks for the instructions and useful help offered by Prof. Wu and all the TAs!