OS Project-6 Report

Banker's Algorithm

Zhou Yiyuan

516021910270
1025152261@qq.com

1 Data Structure

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];
```

2 Request Resources

```
int request_resources(int customer_num, int request[]){
    for(int i=0; i<NUMBER_OF_RESOURCES; ++i){
        if(request[i] > need[customer_num][i] || request[i] > available[i]){
            return -1;
        }
    }

    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];
    }

    if(is_safe() >= 0){
        return 0;
    }else{
        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;
    }
}
```

2.1 is_safe()

Determine if the state is safe.

```
int is_safe(){
    int work[NUMBER_OF_RESOURCES];
    for(int j=0; j<NUMBER_OF_RESOURCES; ++j){
        work[j] = available[j];
    }
    int finish[NUMBER_OF_CUSTOMERS];
    for(int i=0; i<NUMBER_OF_CUSTOMERS; ++i){
        finish[i] = -1;
    }

    while(1){
        int i;
        for(i=0; i<NUMBER_OF_CUSTOMERS; ++i){
            if(finish[i] >= 0) continue;
            int j;
            for(j=0; j<NUMBER_OF_RESOURCES; ++j){
                if(need[i][j] > work[j]) break;
            }
            if(j < NUMBER_OF_RESOURCES) continue;

            finish[i] = 1;
            for(int k=0; k<NUMBER_OF_RESOURCES; ++k){
                work[k] += allocation[i][k];
            }
            break;
        }

        if(i == NUMBER_OF_CUSTOMERS){
            for(int k=0; k<NUMBER_OF_CUSTOMERS; ++k){
                if(finish[k] < 0) return -1;
            }
            return 0;
        }
    }
}
```

3 Release Resources

```
void release_resources(int customer_num, int release[]){
    for(int i=0; i<NUMBER_OF_RESOURCES; ++i){
        int r;
        if(release[i] <= allocation[customer_num][i]){
            r = release[i];
        }else{
```

```
            r = allocation[customer_num][i];
        }
        available[i] += r;
        allocation[customer_num][i] -= r;
        need[customer_num][i] += r;
    }
}
```

4 Main()

The steps in the main() are shown as following:

(a) Initialize.

- initialize available from command line arguments
- initialize maximum from the file
- initialize allocation as all zeros
- initialize need as maximum

(b) Read a command from the command line.

(c) Analyze the command and execute the different functions:

- RQ: execute request_resources()
- RL: execute release_resources()
- * : output the values of the different data structures
- exit: break

(d) Jump back to step (b).

5 Results

The order of time is Figure 1-4.

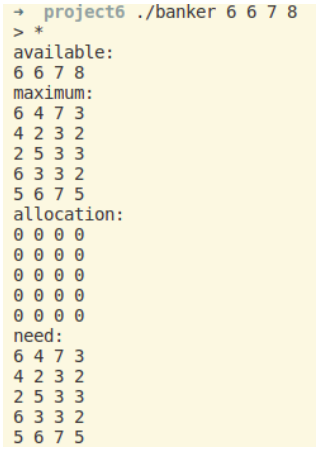

Figure 1: Initialization

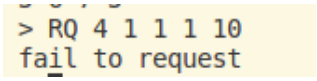

Figure 2: Fail to request resources, because request[3]=10 > need[4][3]=5.

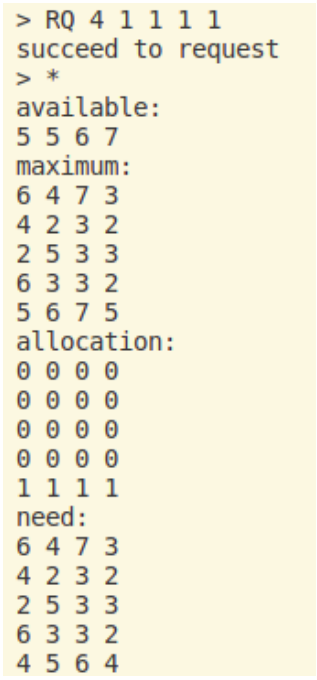

Figure 3: Succeed to request resources, because the state is safe after the request. The available and need[4] decrease and allocation[4] increase.

```
> RL 4 1 0 0 1
> *
available:
6 5 6 8
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 1 1 0
need:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 5 6 5
```

Figure 4: Release resources. The allocation[4] decrease and need[4] and available increase.