Project 6--Banker's Algorithm

陈思远 518021910567

1. 实验内容和目标

1. 编写程序实现银行家算法。
2. 采用合适的数据测试银行家算法是否正确。

2. 银行家算法

在银行家算法中，用户向银行请求或释放资源。银行家需要保证每个请求发生后，整个系统仍然处于安全状态，以及请求的资源不能多于可得的资源。如果这些条件无法被满足，就拒绝这个请求。

2.1数据结构

```
#define CUSTOMER_NUM 5
#define RESOURCE_NUM 4

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

/*the maximum demand of each customer */
int maximum[CUSTOMER_NUM][RESOURCE_NUM];

/* the amount currently allocated to each customer */
int allocation[CUSTOMER_NUM][RESOURCE_NUM]={0};

/* the remaining need of each customer */
int need[CUSTOMER_NUM][RESOURCE_NUM];
```

2.2 请求资源函数

在请求资源函数中，首先需要检验request是否超过need和available，若超过，则该请求必然无法被完成。接着，检查如果这个request被满足，系统是否处于安全状态，这时就要用到安全算法。描述如下。

1. work是长度为`RESOURCE_NUM`的向量，表示目前可得的资源；finish是长度为`CUSTOMER_NUM`的向量，表示任务是否完成。初始化`work=available`，`finish[i]=0`。
2. 查找这样的`i`使之满足`finish[i]==0 && need[i]<=work`，如果没有这样的`i`存在，就转到第4步。
3. `work=work+allocation[i]; finish[i]=true;` 返回第2步。

4. 如果对所有的`i`，`finish[i]==1`，那么系统处于安全状态，否则处于非安全状态。

如果发现系统处于安全状态，那就可以把资源分配给请求的进程；若处于非安全状态，则不分配资源并且把allocation和need矩阵恢复原先的状态。

```
int request_resources(int cust_num,int request[]){
    int work[RESOURCE_NUM];
    int finish[CUSTOMER_NUM]={0};

    for(int i=0;i<RESOURCE_NUM;i++){
        if(request[i]>need[cust_num][i]){
            printf("error! request more than need!\n");
            return -1;
        }
    }
    for(int i=0;i<RESOURCE_NUM;i++){
        if(request[i]>available[i]){
            printf("error! available resource is not enough!\n");
            return -1;
        }
    }
    for(int i=0;i<RESOURCE_NUM;i++){    //假设该请求被满足
        work[i]=available[i]-request[i];
        allocation[cust_num][i]+=request[i];
        need[cust_num][i]-=request[i];
    }
    while(1){   //检测系统是否处于安全状态
        int flag=0;
        for(int i=0;i<CUSTOMER_NUM;i++){
            if(finish[i]==0){
                int j;
                for(j=0;j<RESOURCE_NUM;j++){
                    if(need[i][j]>work[j])
                        break;
                }
                if(j==RESOURCE_NUM){
                    flag=1;
                    for(int k=0;k<RESOURCE_NUM;k++){
                        work[k]+=allocation[i][k];
                    }
                    finish[i]=1;
                    break;
                }
            }
        }
        if(flag==0){
            for(int i=0;i<CUSTOMER_NUM;i++){
                if(finish[i]==0){           //unsafe!
                    printf("the state is unsafe!\n");
                    for(int k=0;k<RESOURCE_NUM;k++){    //恢复原先的状态
                        allocation[cust_num][k]-=request[k];
                        need[cust_num][k]+=request[k];
                    }
                    return -1;
                }
            }
            for(int i=0;i<RESOURCE_NUM;i++){   //safe!
```

```
                available[i]-=request[i];
            }
            return 0;
        }
    }
}
```

2.3 释放资源函数

释放资源需要首先判断释放的资源数量是否多于分配的资源数量，若是，则无效；若否，则释放这些资源。

```
void release_resources(int cust_num,int release[]){
    for(int i=0;i<RESOURCE_NUM;i++){
        if(release[i]>allocation[cust_num][i]){
            printf("error! release more than allocated!");
            return;
        }
    }
    for(int i=0;i<RESOURCE_NUM;i++){
        allocation[cust_num][i]-=release[i];
        available[i]+=release[i];
        need[cust_num][i]+=release[i];
    }
    printf("the resource has been released!\n");
}
```

2.4 main函数

在主函数中，首先解析命令行，得到available数组，然后从文件中读取maximum数组，初始化need和allocation数组。接着，由用户输入请求资源、释放资源或打印信息的指令，解析这些指令后，调用对应函数处理用户请求。若用户输入“exit”，则程序结束。

```
int main(int argc,char*argv[]){
    //get command line arguments
    if(argc!=RESOURCE_NUM+1){
        printf("error: wrong argument!\n");
        return 1;
    }
    for(int i=1;i<argc;i++){
        available[i-1]=atoi(argv[i]);
    }
    FILE *in;
    in = fopen("max_request.txt","r");
    char space;
    for(int i=0;i<CUSTOMER_NUM;i++){
        fscanf(in,"%d,%d,%d,%d",&maximum[i][0],
               &maximum[i][1],&maximum[i][2],&maximum[i][3]);
        fscanf(in,"%c",&space);
    }
    for(int i=0;i<CUSTOMER_NUM;i++){
        for(int j=0;j<RESOURCE_NUM;j++){
            need[i][j]=maximum[i][j];
        }
    }
    while(1){
        printf(">");
```

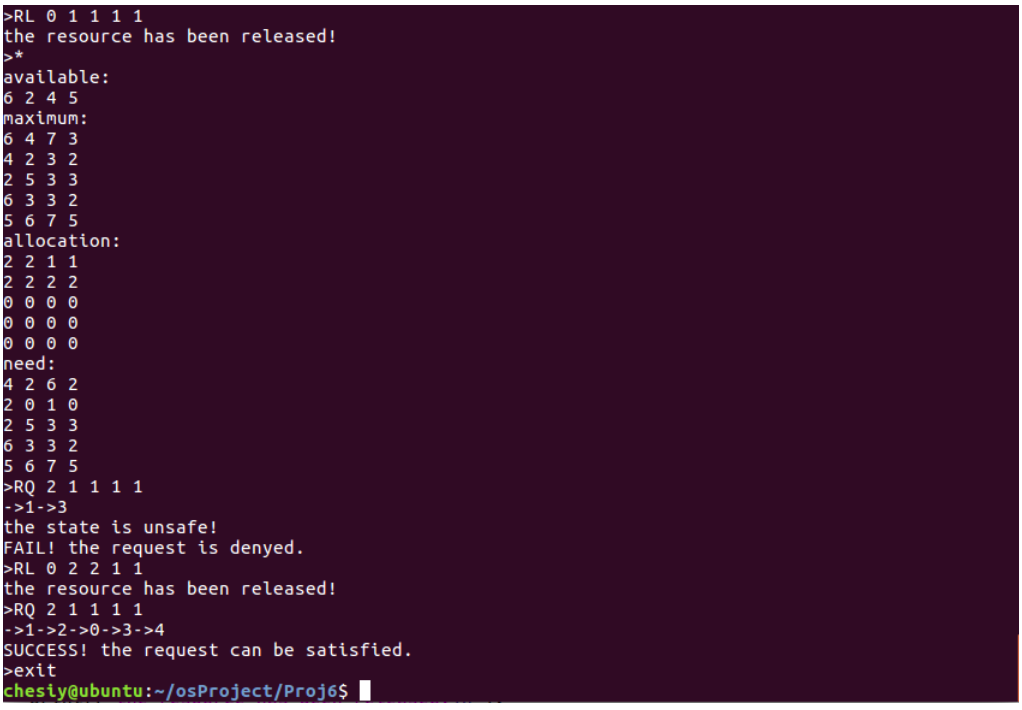
```
        char command[5];
        int cust_num;
        int request[RESOURCE_NUM];
        int release[RESOURCE_NUM];

        scanf("%s",command);
        if(strcmp(command,"RQ")==0){    //请求资源
            scanf("%c",&space);
            scanf("%d ",&cust_num);
            for(int i=0;i<RESOURCE_NUM;i++){
                scanf("%d",&request[i]);
                scanf("%c",&space);
            }
            int res=request_resources(cust_num,request);
            if(res==0){
                printf("SUCCESS! the request can be satisfied.\n");
            }else{
                printf("FAIL! the request is denyed.\n");
            }
        }else{
            if(strcmp(command,"RL")==0){    //释放资源
                scanf("%c",&space);
                scanf("%d ",&cust_num);
                for(int i=0;i<RESOURCE_NUM;i++){
                    scanf("%d",&release[i]);
                    scanf("%c",&space);
                }
                release_resources(cust_num,release);
            }else{
                if(strcmp(command,"*")==0){     //打印信息
                    printf("available:\n");
                    for(int i=0;i<RESOURCE_NUM;i++){
                        printf("%d ",available[i]);
                    }
                    printf("\n");
                    printf("maximum:\n");
                    for(int i=0;i<CUSTOMER_NUM;i++){
                        for(int j=0;j<RESOURCE_NUM;j++){
                            printf("%d ",maximum[i][j]);
                        }
                        printf("\n");
                    }
                    printf("allocation:\n");
                    for(int i=0;i<CUSTOMER_NUM;i++){
                        for(int j=0;j<RESOURCE_NUM;j++){
                            printf("%d ",allocation[i][j]);
                        }
                        printf("\n");
                    }
                    printf("need:\n");
                    for(int i=0;i<CUSTOMER_NUM;i++){
                        for(int j=0;j<RESOURCE_NUM;j++){
                            printf("%d ",need[i][j]);
                        }
                        printf("\n");
                    }
                }else{
                    if(strcmp(command,"exit")==0){      //退出
```

```
                    break;
                }
            }
        }
    }
    return 0;
}
```

2.5 结果

`available=[10 6 7 8]`。首先consumer 0 请求资源 (3 3 2 2)，请求成功并输出安全序列；之后consumer 1 请求资源 (2 2 2 2)，请求成功。这时`available=[5 1 3 4]`，consumer 2再请求资源 (2 2 3 3)，发现请求的量多于可得的量，故该请求失败。于是，consumer 2 减少请求资源的数量，请求 (1 1 1 1)，这时发现若接收该请求，系统就会处于不安全状态，故该请求也失败。释放consumer 0的资源(1 1 1 1)，consumer 2再请求(1 1 1 1)，仍然不行；将consumer 0的资源全部释放，consumer 2就可以成功请求。



3. 总结与思考

在本次project中，我编程实现了银行家算法，这让我在实践中又复习了一遍银行家算法的内容，我对它的理解也更加完善。在我看来，这次project的难点和要点在于编程结束后拟定一些测试数据来判断自己写的代码是否正确。构造测试数据的过程就是一个手动检查请求后系统是否处于安全状态的过程，需要在脑海中跑一遍安全算法，这很好地检验了我对这个算法的掌握程度。