Banker's Algorithm

Overview

For this project, I will write a program that implements the banker's algorithm discussed in Section8.6.3. 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, I will develop a solution using C.

Program Structure

The major problem in the banker's algorithm is how to judge whether the system is safe or not. The basic idea is to execute processes that can be executed. If there remain more than one process that cannot be executed, the system is unsafe. Relative codes are as below

```
bool isSafe(int customer_num, int request[])
{
    int cnt = 0;
    bool finish[NUMBER_OF_CUSTOMERS];
    int work[NUMBER_OF_CUSTOMERS];

    for(int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
        finish[i] = false;
    for(int i = 0; i < NUMBER_OF_RESOURCES; i++)
        work[i] = available[i] - request[i];

    for(int i = 0; i < NUMBER_OF_RESOURCES; i++)
    {
        allocation[customer_num][i] += request[i];
        need[customer_num][i] -=request[i];
    }

    while(cnt!=NUMBER_OF_CUSTOMERS)
    {
        int i;
        for(i = 0; i < NUMBER_OF_CUSTOMERS; i++)
        {
            if(finish[i])
                continue;
            else
            {

                int j;
                for(j = 0; j < NUMBER_OF_RESOURCES; j++)
                {
                    if(work[j] < need[i][j])
```

```
                    break;
                }
                if(j == NUMBER_OF_RESOURCES)
                {
                    printf("%d ", i);
                    for(int k = 0; k < NUMBER_OF_RESOURCES; k++)
                    {
                        work[k] += allocation[i][k];
                    }
                    finish[i] = true;
                    cnt++;
                    break;
                }
            }
        }
        if(i == NUMBER_OF_CUSTOMERS)
            break;
    }
    printf("%d\n", cnt);
    for(int i = 0; i < NUMBER_OF_RESOURCES; i++)
    {
        allocation[customer_num][i] -= request[i];
        need[customer_num][i] +=request[i];
    }
    return cnt == NUMBER_OF_CUSTOMERS;

}
```

Then, I implement the `request_resources()` and `release_resources()` as below.

```
void request_resources(int customer_num, int request[])
{
    if(customer_num >= NUMBER_OF_CUSTOMERS || customer_num < 0)
    {
        printf("CUSTOMER ID INVALID\n");
        return;
    }

    for(int i = 0; i < NUMBER_OF_RESOURCES; i++)
    {
        if(request[i] > maximum[customer_num][i])
        {
            printf("REQUEST EXCEED MAXIMUM\n");
            return;
        }
    }


    for(int i = 0; i < NUMBER_OF_RESOURCES; i++)
    {
        allocation[customer_num][i] += request[i];
        available[i] -= request[i];
    }
    update();
```

```
}



void release_resources(int customer_num, int release[])
{
    if(customer_num >= NUMBER_OF_CUSTOMERS || customer_num < 0)
    {
        printf("CUSTOMER ID INVALID\n");
        return;
    }
    for(int i = 0; i < NUMBER_OF_RESOURCES; i++)
    {
        // judge whether released resources exceed allocated resources
        if(release[i] > allocation[customer_num][i])
        {
            printf("RELEASE EXCEED ALLOCATION\n");
            return;
        }

    }
    for(int i = 0; i < NUMBER_OF_RESOURCES; i++)
    {
        allocation[customer_num][i] -= release[i];
        available[i] += release[i];
    }
    update();

}
```

User interface

```
int main(int argc, char *argv[])
{
    //int args[NUMBER_OF_RESOURCES]

    if(argc < 1 + NUMBER_OF_RESOURCES)
    {
        fprintf(stderr, "Usage: %d is required", NUMBER_OF_RESOURCES);
        return -1;
    }

    /* initialize available vector */

    for(int i = 0; i < NUMBER_OF_RESOURCES; i++)
        available[i] = atoi(argv[i+1]);

    /* read the file */

    FILE *in = fopen("file.txt","r");

    char line[NUMBER_OF_RESOURCES * 2];
    char *tmp;
```

```
    char *token;

    for(int i = 0; i < NUMBER_OF_CUSTOMERS;i++)
    {
        fgets(line,NUMBER_OF_RESOURCES * 2,in);
        tmp = strdup(line);
        token = strsep(&tmp, DELIM);
        int cnt = 0;
        while(token)
        {
            maximum[i][cnt++] = atoi(token);
            printf("%d ", atoi(token));
            token = strsep(&tmp,DELIM);
        }
        printf("\n");
        fgets(line,NUMBER_OF_RESOURCES * 2,in);
        free(tmp);
    }

    fclose(in);
    update();



    /* Execute input commands */

    int args[NUMBER_OF_RESOURCES];
    char cmd[COMMAND_SIZE];
    int customer_num;

    while(fgets(cmd, COMMAND_SIZE, stdin))
    {
        if(cmd[0] == '*')
        {
            /* resources panel */
            display();
        }
        else if(cmd[0] == 'R' && cmd[1] == 'L')
        {
            /* release  resources */
            getArgs(cmd,&customer_num, args);
            release_resources(customer_num, args);

        }
        else if(cmd[0] == 'R' && cmd[1] == 'Q')
        {
            /* request resources */
            getArgs(cmd,&customer_num, args);
            if(isSafe(customer_num,args))
            {
                request_resources(customer_num,args);
            }
            else
```

```
            {
                printf("REQUEST NOT GRANTED\n");
            }

        }
        else if(cmd[0] == '!')
        {
            break;
        }
    }

}
```

How to run the program

To compile the files, enter `gcc -o test banker.c`

To run it, enter `./test a b c d` (a,b,c,d are 4 resources)