Group Project 3

All about deadlock

03/31/2024

CECS 326 - Operating Systems

Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Brandon Quach (#029133232)

Michael Kim (#025633788)

Project demo:

Link: https://youtu.be/0G1UIPzm-4A?feature=shared

Project Description:

In this project, we created a program that applies the banker's algorithm as we went over class 2 weeks ago. Customers will make resource requests and release them to the system. The banker will approve the request only if the system remains in its safe state; otherwise, it will be rejected. In the project, we decided to use C.

Steps:

1. Download the bankersalg.c file and input.txt file.
2. Navigate to the folder using cd where your files are saved and compile them using these commands
    a. gcc bankersalg.c
    b. ./a.exe 10 5 7 8
3. Use the command "RQ 0 3 1 2 1" to make customer 0 request the resources 3,1,2,1.
    a. By requesting more than the available resources or maximum resources for each customer, the request will be denied.
4. Use the command "RL 0 3 1 2 1" to make customer 0 release the resources it requested.
    a. Error message should pop up if there are resources the customers doesn't have
5. Use the command "*" to view the outputs of the values of the available, maximum, allocation, and needed arrays.

Code analysis:

The banker will keep track of the resources using these data structures:

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

#define NUMBER_OF_CUSTOMERS 5   // Maximum number of customers
#define NUMBER_OF_RESOURCES 4    // Maximum number of resources

int available[NUMBER_OF_RESOURCES];        // Available resources
int maximum[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];   // Maximum resources required by each customer
int allocation[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];    // Resources currently allocated to each customer
int need[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];      // Remaining needs of each customer

int num_res;   // Number of resource types

void init();
bool request_resources(int customer_num, int request[]);
void release_resources(int customer_num, int release[]);
void display();
```

void Init() function initializes all arrays to 0:

```c
void init() {
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
            maximum[i][j] = 0;
            allocation[i][j] = 0;
            need[i][j] = 0;
        }
    }
}
```

bool request_resources() function handles resource requests:

```c
bool request_resources(int customer_num, int request[]) {
    // Check if request is valid
    for (int i = 0; i < NUMBER_OF_RESOURCES; i++) {
        if (request[i] > need[customer_num][i] || request[i] > available[i]) {
            return false; // Request cannot be granted
        }
    }

    // Try allocating resources temporarily
    for (int i = 0; i < num_res; i++) {
        available[i] -= request[i];
        allocation[customer_num][i] += request[i];
        need[customer_num][i] -= request[i];
    }

    return true; // Request granted
}
```

void release_resources() function handles release resources:

```c
void release_resources(int customer_num, int release[]) {
    // Check if release is valid
    for (int i = 0; i < num_res; i++) {
        if (release[i] > allocation[customer_num][i]) {
            printf("Error: Attempting to release more resources than allocated.\n");
            return;
        }
    }

    // Release resources
    for (int i = 0; i < num_res; i++) {
        available[i] += release[i];
        allocation[customer_num][i] -= release[i];
        need[customer_num][i] += release[i];
    }

    printf("Resources released.\n");
}
```

void display() function displays current state

```c
void display() {
    printf("\nAvailable resources:\n");
    for (int i = 0; i < num_res; i++) {
        printf("%d ", available[i]);
    }
    printf("\n");

    printf("\nMaximum resources:\n");
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        for (int j = 0; j < num_res; j++) {
            printf("%d ", maximum[i][j]);
        }
        printf("\n");
    }

    printf("\nAllocated:\n");
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        for (int j = 0; j < num_res; j++) {
            printf("%d ", allocation[i][j]);
        }
        printf("\n");
    }

    printf("\nNeed:\n");
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        for (int j = 0; j < num_res; j++) {
            printf("%d ", need[i][j]);
        }
        printf("\n");
    }
}
```

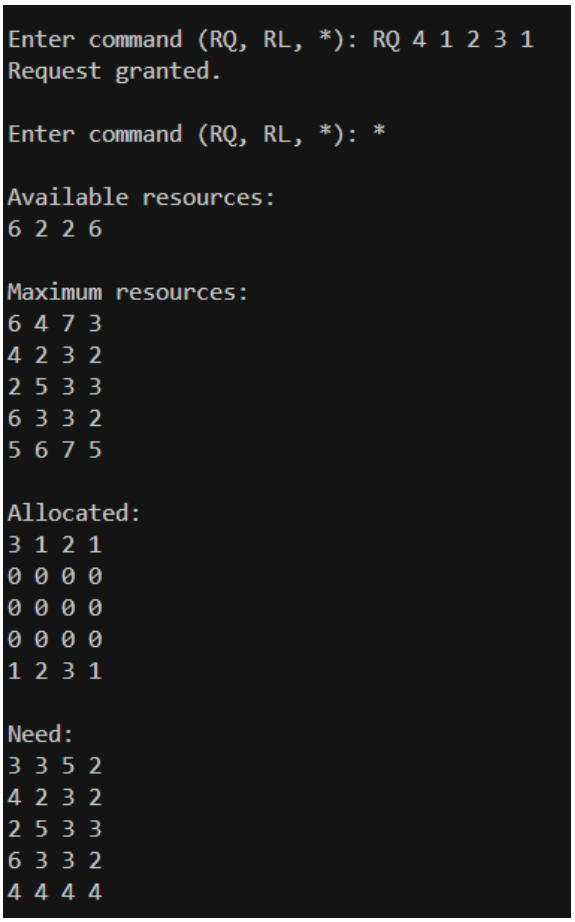
This function displays an interactive menu for user input:

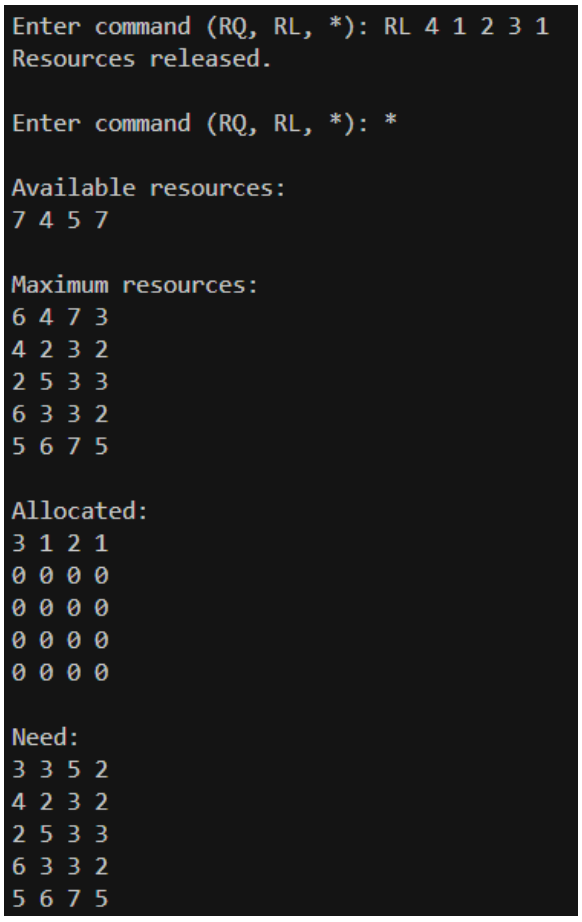
```
while (true) {
    printf("\nEnter command (RQ, RL, *): ");
    scanf("%2s", command);   // Read command as a string of maximum 2 characters

    // Process the command entered by the user
    if (strcmp(command, "RQ") == 0) { // Compare cmd with "RQ"
        scanf("%d", &customers);
        for (int i = 0; i < num_res; i++) {
            scanf("%d", &request[i]);
        }
        // Call function to handle resource request
        if (request_resources(customers, request)) {
            printf("Request granted.\n");
        } else {
            printf("Request denied.\n");
        }
    } else if (strcmp(command, "RL") == 0) { // Compare cmd with "RL"
        scanf("%d", &customers);
        for (int i = 0; i < num_res; i++) {
            scanf("%d", &releases[i]);
        }
        // Call function to handle resource release
        release_resources(customers, releases);
    } else if (strcmp(command, "*") == 0) { // Compare cmd with "*"
        // Display current state
        display();
    } else {
        printf("Invalid command. Please enter RQ, RL, or *.\n");
    }
}

return 0;
```

int main() controls the program:

```c
int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Usage: %s <av res>\n", argv[0]);
        return 1;
    }

    // Initialize available resources from command line arguments
    num_res = argc - 1;
    for (int i = 0; i < num_res; i++) {
        available[i] = atoi(argv[i + 1]);
    }

    // Read maximum resources from input file
    FILE *file = fopen("input.txt", "r");
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    int num_cust = 0;
    // Read each line from the file to get maximum resource allocation for each customer
    while (!feof(file) && num_cust < NUMBER_OF_CUSTOMERS) {
        for (int i = 0; i < num_res; i++) {
            fscanf(file, "%d,", &maximum[num_cust][i]);
            allocation[num_cust][i] = 0;
            need[num_cust][i] = maximum[num_cust][i];
        }
        num_cust++;
    }
    fclose(file);

    char command[3];   // Command entered by user
    int customers;  // Customer number
    int request[NUMBER_OF_RESOURCES];  // Resource request
    int releases[NUMBER_OF_RESOURCES];  // Resource release
```

Program output:





Outcome when customer 4 requests the resources then releases them. The 2nd example shows that the error message pops up when customer 4 has no resources.

Output from example on project 3 pdf:

```
$ ./a.exe 10 5 7 8

Enter command (RQ, RL, *): RQ 0 3 1 2 1
Request granted.

Enter command (RQ, RL, *): RL 4 1 2 3 1
Error: Attempting to release more resources than allocated.

Enter command (RQ, RL, *): *

Available resources:
7 4 5 7

Maximum resources:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

Allocated:
3 1 2 1
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

Need:
3 3 5 2
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

Enter command (RQ, RL, *):
```

What each individual did:

For this project we kept the roles the same to avoid any confusion.

Brandon: Brandon took the role in the completion of the lab report and ensuring that all project deliverables were included. In addition to his lab report responsibilities, he also worked on the code with Michael and input the proper comments to show the design.

Michael: Michael took a technical role within the group, with a primary focus on both software. His responsibilities encompassed software development, creation, and software components. In addition to his software responsibilities, he also focused on the video.

Work split was mostly 50/50 when it came to the software and its deliverables.

References:

https://www.geeksforgeeks.org/program-bankers-algorithm-set-1-safety-algorithm/

https://www.youtube.com/watch?v=5v5xpQi7fHk