Name: Chillara V L N S Pavana Vamsi

Reg.no: 21BCE5095

Date: 16/02/2023

Faculty: M Sivagami

1. Develop a banker's algorithm for n processes and m resources. Assume that max, allocation and available matrix are given.

- Find the need matrix
- Check the system is in a safe state
- if a request from p1 arrives for (x1,y1,w1,z1), can the requested be granted immediately? will it lead safe state? if it is safe state print the final available matrix .
- if a request from p2 arrives for (x2,y2,w2,z2) can the requested be granted immediately? will it lead safe state? if it is safe state print the final available matrix.

Code:

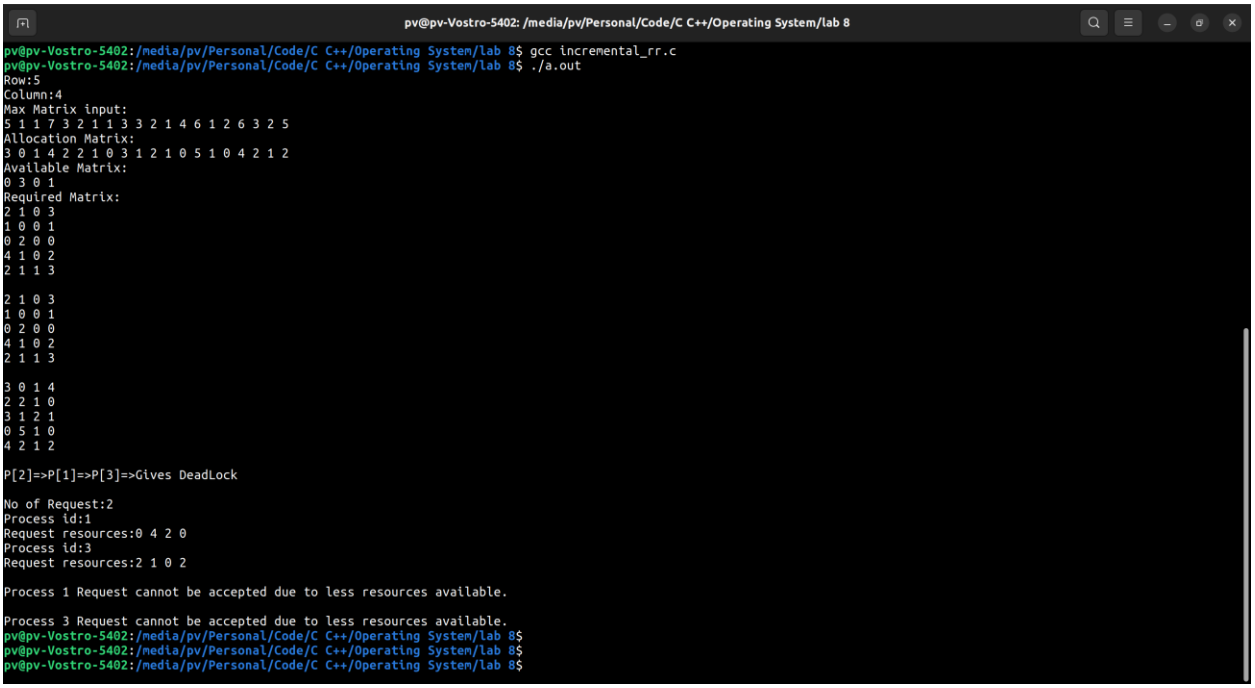
```c
#include <stdio.h>
#define row 5
#define col 4
int compare(int a[col], int b[col])
{
    int x = 0;
    for (int i = 0; i < col; i++)
    {
        if (a[i] <= b[i])
            x++;
    }
    if (x == col)
        return 1;
    else
        return 0;
}
void print_2d(int a[row][col])
{
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
            printf("%d ", a[i][j]);
        printf("\n");
    }
    printf("\n");
}
int safers(int required[][col], int allocation[][col], int available[col])
{
    int current[col];
    int flag = 0, rc = 0;
    int final[row];
    for (int i = 0; i < row; i++)
        final[i] = 0;
    for (int i = 0; i < col; i++)
        current[i] = available[i];
```

```
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < row; j++)
        {

            if (final[j] == 0)
            {
                flag=0;
                for (int k = 0; k < col; k++)
                {
                    if (current[k] >= required[j][k])
                        flag++;
                }
                if (flag == col)
                {
                    for (int k = 0; k < col; k++)
                    {
                        current[k] += allocation[j][k];
                    }
                    final[j] = 1;
                    printf("P[%d]=>", j);
                }
            }
        }
    }
    for (int i = 0; i < row; i++)
    {
        if (final[i] == 1)
            rc++;
    }

    if (rc == row)
    {
        return 0;
    }
    else
        return 1;
}
void process(int x, int resources_required[][col], int allocation[row][col], int
required[row][col], int available[col], int k)
{
    int t;
    int new_allocation[row][col], new_required[row][col], new_available[col];
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            new_allocation[i][j] = allocation[i][j];
            new_required[i][j] = required[i][j];
            new_available[j] = available[j];
        }
    }
    if (compare(resources_required[k], new_available))
```

```
    {
        for (int i = 0; i < col; i++)
        {
            t = resources_required[k][i];
            new_allocation[x][i] = new_allocation[x][i] + t;
            new_available[i] = new_available[i] - t;
            new_required[x][i] = new_required[x][i] - t;
        }
        if (safers(new_required, new_allocation, new_available) == 0)
        {
            printf("\nProcess %d Request is safe to add.\n", x + 1);
            for (int i = 0; i < col; i++)
            {
                t = resources_required[k][i];
                allocation[x][i] = allocation[x][i] + t;
                available[i] = available[i] - t;
                required[x][i] = required[x][i] - t;
            }
        }
        else
            printf("\nProcess %d required makes a deadlock.\n", x);
    }
    else
        printf("\nProcess %d Request cannot be accepted due to less resources
available.\n", x);
}
void resourcerequired(int allocation[row][col], int required[row][col], int
available[col])
{
    int n, x;
    printf("No of Request:");
    scanf("%d", &n);
    int process_required[n];
    int resources_required[n][col];
    for (int i = 0; i < n; i++)
    {
        printf("Process id:");
        scanf("%d", &process_required[i]);
        printf("Request resources:");
        for (int j = 0; j < col; j++)
            scanf("%d", &resources_required[i][j]);
    }
    for (int i = 0; i < n; i++)
    {
        process(process_required[i], resources_required, allocation, required, available,
i);
    }
}
void getRequired(int max[][col], int allocation[][col], int required[][col])
{
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
```

```
        {
            required[i][j] = max[i][j] - allocation[i][j];
        }
    }
}
int main()
{
    printf("Row:%d\nColumn:%d\n", row, col);
    printf("Max Matrix input:\n");
    int max[row][col]; //5 1 1 7 3 2 1 1 3 3 2 1 4 6 1 2 6 3 2 5
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            scanf("%d", &max[i][j]);
        }}
    printf("Allocation Matrix:\n");
    int allocation[row][col];// 3 0 1 4 2 2 1 0 3 1 2 1 0 5 1 0 4 2 1 2
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            scanf("%d", &allocation[i][j]);
        }
    }
    printf("Available Matrix:\n");
    int available[col];// 0 3 0 1
    for (int i = 0; i < col; i++)
        scanf("%d", &available[i]);
    int required[row][col];
    getRequired(max, allocation, required);
    printf("Required Matrix:\n");
    print_2d(required);
    int new_allocation[row][col], new_required[row][col], new_available[col];

    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            new_allocation[i][j] = allocation[i][j];
            new_required[i][j] = required[i][j];
            new_available[j] = available[j];
        }}
    print_2d(new_required);
        print_2d(new_allocation);
    int res = safers(required, allocation, available);
    if (res == 1)
        printf("Gives DeadLock\n\n");
    else
        printf("Safe\n\n");
    resourcerequired(new_allocation, new_required, new_available);
    return 0;
}
```

Output:

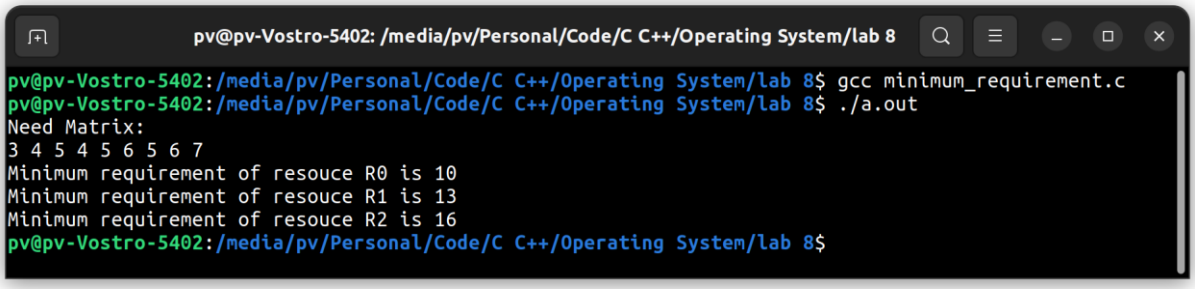


2. Develop a C program to identify the minimum requirement of each resources of m for n processes.

Code:

```c
#include <stdio.h>
int main()
{
    int i, j, count = 0, tot = 0, row = 3, col = 3;
    int request[row][col];//3 4 5 4 5 6 5 6 7
    printf("Need Matrix:\n");
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
                scanf("%d",&request[i][j]);
        }
    }
    for (i = 0; i < col; i++)
    {
        count = 0;
        tot = 0;
        for (j = 0; j < row; j++)
        {
            count += request[j][i] - 1;
        }
        tot += count + 1;
        printf("Minimum requirement of resouce R%d is %d\n", i, tot);
    }
    return 0;
}
```

Output:



3. Develop a c program as a multilevel feedback scheduler and show the output for the below scenario for n process with different arrival times and different burst for 3 level of schedulers

Note: Level1 - 8-time units level 2 - 16 time units and level3 – FCFS

Code:

```c
#include <stdio.h>

struct process
{
    char name;
    int AT, BT, WT, TAT, RT, CT;
} Q1[10], Q2[10], Q3[10];

int n;
void sortByArrival()
{
    struct process temp;
    int i, j;
    for (i = 0; i < n; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (Q1[i].AT > Q1[j].AT)
            {
                temp = Q1[i];
                Q1[i] = Q1[j];
                Q1[j] = temp;
            }
        }
    }
}

int main()
{
    int i, j, k = 0, r = 0, time = 0, tq1 = 8, tq2 = 16, flag = 0,count=0;
    char c;
    printf("Enter no of processes:");
    scanf("%d", &n);
    for (i = 0, c = 'A'; i < n; i++, c++)
    {
```

```
        Q1[i].name = c;
        printf("\nEnter the arrival time and burst time of process %c: ", Q1[i].name);
        scanf("%d%d", &Q1[i].AT, &Q1[i].BT);
        Q1[i].RT = Q1[i].BT;
    }
    sortByArrival();
    time = Q1[0].AT;
    printf("Process in first queue following RR with qt=%d", tq1);
    printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");
    for (i = 0; i < n; i++)
    {

        if (Q1[i].RT <= tq1)
        {

            time += Q1[i].RT;
            Q1[i].RT = 0;
            Q1[i].WT = time - Q1[i].AT - Q1[i].BT;
            Q1[i].TAT = time - Q1[i].AT;
            printf("\n%c\t\t%d\t\t%d\t\t%d", Q1[i].name, Q1[i].BT, Q1[i].WT, Q1[i].TAT);

        }
        else
        {
            Q2[k].WT = time;
            time += tq1;
            Q1[i].RT -= tq1;
            Q2[k].BT = Q1[i].RT;
            Q2[k].RT = Q2[k].BT;
            Q2[k].name = Q1[i].name;
            k = k + 1;
            flag = 1;
            count++;
        }
    }
    if(count==n)
    printf("\nNo Process ends in queue 1");
    if (flag == 1)
    {
        printf("\n\nProcess in second queue following RR with qt=%d", tq2);
        printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");
    }
    for (i = 0; i < k; i++)
    {
        if (Q2[i].RT <= tq2)
        {
            time += Q2[i].RT;
            Q2[i].RT = 0;
            Q2[i].WT = time - tq1 - Q2[i].BT;
            Q2[i].TAT = time - Q2[i].AT;
            printf("\n%c\t\t%d\t\t%d\t\t%d", Q2[i].name, Q2[i].BT, Q2[i].WT, Q2[i].TAT);
        }
        else
```

```
        {
            Q3[r].AT = time;
            time += tq2;
            Q2[i].RT -= tq2;
            Q3[r].BT = Q2[i].RT;
            Q3[r].RT = Q3[r].BT;
            Q3[r].name = Q2[i].name;
            r = r + 1;
            flag = 2;
        }
    }

    {
        if (flag == 2)
            printf("\n\nProcess in third queue following FCFS ");
    }
    for (i = 0; i < r; i++)
    {
        if (i == 0)
            Q3[i].CT = Q3[i].BT + time - tq1 - tq2;
        else
            Q3[i].CT = Q3[i - 1].CT + Q3[i].BT;
    }

    for (i = 0; i < r; i++)
    {
        Q3[i].TAT = Q3[i].CT;
        Q3[i].WT = Q3[i].TAT - Q3[i].BT;
        printf("\n%c\t\t%d\t\t%d\t\t%d\t\t", Q3[i].name, Q3[i].BT, Q3[i].WT, Q3[i].TAT);
    }
    printf("\n");
    return 0;
}
```

Output: