Slip no 1

Q.1) Write a C Menu driven Program to implement following functionality

a) Accept Available
b) Display Allocation, Max
c) Display the contents of need matrix
d) Display Available.

```
#include <stdio.h>
#define PROCESS_COUNT 5
#define RESOURCE_COUNT 3
int available[RESOURCE_COUNT];
int allocation[PROCESS_COUNT][RESOURCE_COUNT] = {
    {2, 3, 2},
    {4, 0, 0},
    {5, 0, 4},
    {4, 3, 3},
    {2, 2, 4}
};
int max[PROCESS_COUNT][RESOURCE_COUNT] = {
    {9, 7, 5},
    {5, 2, 2},
    {1, 0, 4},
    {4, 4, 4},
    {6, 5, 5}
};
int need[PROCESS_COUNT][RESOURCE_COUNT];
void calculateNeedMatrix() {
    for (int i = 0; i < PROCESS_COUNT; i++) {
        for (int j = 0; j < RESOURCE_COUNT; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
}
void displayAllocationAndMax() {
```

```
    printf("Process\tAllocation\tMax\n");
    printf("\tA B C\t\tA B C\n");
    for (int i = 0; i < PROCESS_COUNT; i++) {
        printf("P%d\t", i);
        for (int j = 0; j < RESOURCE_COUNT; j++) {
            printf("%d ", allocation[i][j]);
        }
        printf("\t\t");
        for (int j = 0; j < RESOURCE_COUNT; j++) {
            printf("%d ", max[i][j]);
        }
        printf("\n");
    }
}
void displayNeedMatrix() {
    printf("Process\tNeed\n");
    printf("\tA B C\n");
    for (int i = 0; i < PROCESS_COUNT; i++) {
        printf("P%d\t", i);
        for (int j = 0; j < RESOURCE_COUNT; j++) {
            printf("%d ", need[i][j]);
        }
        printf("\n");
    }
}
void displayAvailable() {
    printf("Available Resources: ");
    for (int i = 0; i < RESOURCE_COUNT; i++) {
        printf("%d ", available[i]);
    }
    printf("\n");
}
void acceptAvailable() {
    printf("Enter available resources (A B C): ");
    for (int i = 0; i < RESOURCE_COUNT; i++) {
        scanf("%d", &available[i]);
```

```
    }
}
int main() {
    int choice;
    calculateNeedMatrix();
    while (1) {
        printf("\nMenu:\n");
        printf("1. Accept Available\n");
        printf("2. Display Allocation and Max\n");
        printf("3. Display Need Matrix\n");
        printf("4. Display Available\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                acceptAvailable();
                break;
            case 2:
                displayAllocationAndMax();
                break;
            case 3:
                displayNeedMatrix();
                break;
            case 4:
                displayAvailable();
                break;
            case 5:
                return 0;
            default:
                printf("Invalid choice, try again.\n");
        }
    }
    return 0;
}
```

Q.2 Write a simulation program for disk scheduling using FCFS algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 55, 58, 39, 18, 90, 160, 150, 38, 184 Start Head Position: 50

```c
#include <stdio.h>
#include <stdlib.h>
void fcfs(int requests[], int n, int head) {
    int totalMovement = 0;
    printf("Request served in order: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", requests[i]);
        totalMovement += abs(requests[i] - head);
        head = requests[i];
    }
    printf("\nTotal head movements: %d\n", totalMovement);
}
int main() {
    int n, head;
    printf("Enter total number of requests: ");
    scanf("%d", &n);
    int requests[n];
    printf("Enter the disk request string: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }
    printf("Enter the starting head position: ");
    scanf("%d", &head);
    fcfs(requests, n, head);
    return 0;
}
```

Slip no 2

Q.1 Write a program to simulate Linked file allocation method. Assume disk with n number of blocks. Give value of n as input. Randomly mark some block as allocated and accordingly maintain the list of free blocks Write menu driver program with menu options as mentioned below and implement each option.

- Show Bit Vector
- Create New File
- Show Directory
- Exit

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX_FILES 10
#define MAX_BLOCKS 100
int bitVector[MAX_BLOCKS];
int directory[MAX_FILES][MAX_BLOCKS];
int fileStart[MAX_FILES];
int fileCount = 0;
void initializeDisk(int n) {
    srand(time(0));
    for (int i = 0; i < n; i++) {
        bitVector[i] = rand() % 2;
    }
}
void showBitVector(int n) {
    printf("Bit Vector: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", bitVector[i]);
    }
    printf("\n");
}
void createNewFile(int n) {
    if (fileCount >= MAX_FILES) {
        printf("Directory full! Cannot create more files.\n");
        return;
```

```
    }
    int fileSize, block, currentBlock = -1, previousBlock = -1;
    printf("Enter file size: ");
    scanf("%d", &fileSize);
    if (fileSize > n) {
        printf("File size too large.\n");
        return;
    }
    fileStart[fileCount] = -1;
    for (int i = 0; i < fileSize; i++) {
        do {
            block = rand() % n;
        } while (bitVector[block] != 0);
        bitVector[block] = 1;
        if (previousBlock == -1) {
            fileStart[fileCount] = block;
        } else {
            directory[fileCount][previousBlock] = block;
        }
        previousBlock = block;
        directory[fileCount][block] = -1;
    }
    printf("File created with starting block %d\n", fileStart[fileCount]);
    fileCount++;
}
void showDirectory() {
    printf("Directory:\n");
    for (int i = 0; i < fileCount; i++) {
        printf("File %d: ", i + 1);
        int block = fileStart[i];
        while (block != -1) {
            printf("%d -> ", block);
            block = directory[i][block];
        }
        printf("NULL\n");
    }
```

```
}
int main() {
    int n, choice;
    printf("Enter total number of blocks: ");
    scanf("%d", &n);
    initializeDisk(n);
    while (1) {
        printf("\nMenu:\n");
        printf("1. Show Bit Vector\n");
        printf("2. Create New File\n");
        printf("3. Show Directory\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                showBitVector(n);
                break;
            case 2:
                createNewFile(n);
                break;
            case 3:
                showDirectory();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }
    return 0;
}
```

Q.2 Write an MPI program to calculate sum of randomly generated 1000 numbers (stored in array) on a cluster

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define ARRAY_SIZE 1000
int main(int argc, char *argv[]) {
    int rank, size, i, sum = 0, total_sum = 0;
    int numbers[ARRAY_SIZE], local_sum = 0;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (rank == 0) {
        srand(time(NULL));
        for (i = 0; i < ARRAY_SIZE; i++) {
            numbers[i] = rand() % 100;
        }
    }
    int chunk_size = ARRAY_SIZE / size;
    int local_numbers[chunk_size];
    MPI_Scatter(numbers, chunk_size, MPI_INT, local_numbers, chunk_size, MPI_INT,
0, MPI_COMM_WORLD);
    for (i = 0; i < chunk_size; i++) {
        local_sum += local_numbers[i];
    }
    MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);
    if (rank == 0) {
        printf("Total sum of 1000 numbers: %d\n", total_sum);
    }
    MPI_Finalize();
    return 0;
}
```

Slip no 3

Q.1 Write a C program to simulate Banker's algorithm for the purpose of deadlock avoidance. Consider the following snapshot of system, A, B, C and D is the resource type. Process Allocation Max Available

	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

a)Calculate and display the content of need matrix?
b)Is the system in safe state? If display the safe sequence.

```
#include <stdio.h>
#define PROCESS_COUNT 5
#define RESOURCE_COUNT 4
int allocation[PROCESS_COUNT][RESOURCE_COUNT] = {
    {0, 0, 1, 2},
    {1, 0, 0, 0},
    {1, 3, 5, 4},
    {0, 6, 3, 2},
    {0, 0, 1, 4}
};
int max[PROCESS_COUNT][RESOURCE_COUNT] = {
    {0, 0, 1, 2},
    {1, 7, 5, 0},
    {2, 3, 5, 6},
    {0, 6, 5, 2},
    {0, 6, 5, 6}
};
int available[RESOURCE_COUNT] = {1, 5, 2, 0};
int need[PROCESS_COUNT][RESOURCE_COUNT];
void calculateNeedMatrix() {
    for (int i = 0; i < PROCESS_COUNT; i++) {
```

```
        for (int j = 0; j < RESOURCE_COUNT; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
}
int isSafeState() {
    int work[RESOURCE_COUNT];
    int finish[PROCESS_COUNT] = {0};
    int safeSequence[PROCESS_COUNT];
    int count = 0;
    for (int i = 0; i < RESOURCE_COUNT; i++) {
        work[i] = available[i];
    }
    while (count < PROCESS_COUNT) {
        int found = 0;
        for (int i = 0; i < PROCESS_COUNT; i++) {
            if (finish[i] == 0) {
                int j;
                for (j = 0; j < RESOURCE_COUNT; j++) {
                    if (need[i][j] > work[j])
                        break;
                }
                if (j == RESOURCE_COUNT) {
                    for (int k = 0; k < RESOURCE_COUNT; k++) {
                        work[k] += allocation[i][k];
                    }
                    safeSequence[count++] = i;
                    finish[i] = 1;
                    found = 1;
                }
            }
        }
        if (!found) {
            printf("System is not in a safe state.\n");
            return 0;
        }
```

```
    }
    printf("System is in a safe state.\nSafe sequence is: ");
    for (int i = 0; i < PROCESS_COUNT; i++) {
        printf("P%d ", safeSequence[i]);
    }
    printf("\n");
    return 1;
}
int main() {
    calculateNeedMatrix();
    printf("Need Matrix:\n");
    for (int i = 0; i < PROCESS_COUNT; i++) {
        for (int j = 0; j < RESOURCE_COUNT; j++) {
            printf("%d ", need[i][j]);
        }
        printf("\n");
    }
    isSafeState();
    return 0;
}
```

Q.2 Write an MPI program to calculate sum and average of randomly generated 1000 numbers (stored in array) on a cluster

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define ARRAY_SIZE 1000
int main(int argc, char *argv[]) {
    int rank, size, i;
    int numbers[ARRAY_SIZE], local_sum = 0, total_sum = 0;
    double average;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (rank == 0) {
        srand(time(NULL));
        for (i = 0; i < ARRAY_SIZE; i++) {
            numbers[i] = rand() % 100;
        }
    }
    int chunk_size = ARRAY_SIZE / size;
    int local_numbers[chunk_size];
    MPI_Scatter(numbers, chunk_size, MPI_INT, local_numbers, chunk_size, MPI_INT,
0, MPI_COMM_WORLD);
    for (i = 0; i < chunk_size; i++) {
        local_sum += local_numbers[i];
    }
    MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);
    if (rank == 0) {
        average = total_sum / (double)ARRAY_SIZE;
        printf("Total sum of 1000 numbers: %d\n", total_sum);
        printf("Average of 1000 numbers: %f\n", average);
    }
    MPI_Finalize();
    return 0;
}
```

Slip no 4

Q.1 Implement the Menu driven Banker's algorithm for accepting Allocation, Max fromuser. a)Accept Available

b)Display Allocation, Max

c)Find Need and display It,

d)Display Available Consider the system with 3 resources types A,B, and C with 7,2,6 instances respectively.

Consider the following snapshot: Process Allocation Request

A B C A B C

P0 0 1 0 0 0 0

P1 4 0 0 5 2 2

P2 5 0 4 1 0 4

P3 4 3 3 4 4 4

P4 2 2 4 6 5 5

```
#include <stdio.h>
#define PROCESS_COUNT 5
#define RESOURCE_COUNT 3
int allocation[PROCESS_COUNT][RESOURCE_COUNT] = {
    {0, 1, 0},
    {4, 0, 0},
    {5, 0, 4},
    {4, 3, 3},
    {2, 2, 4}
};
int max[PROCESS_COUNT][RESOURCE_COUNT] = {
    {0, 0, 0},
    {5, 2, 2},
    {1, 0, 4},
    {4, 4, 4},
    {6, 5, 5}
};
int available[RESOURCE_COUNT] = {7, 2, 6};
int need[PROCESS_COUNT][RESOURCE_COUNT];
void calculateNeedMatrix() {
    for (int i = 0; i < PROCESS_COUNT; i++) {
        for (int j = 0; j < RESOURCE_COUNT; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
}
void displayMatrix(int matrix[PROCESS_COUNT][RESOURCE_COUNT], char *name) {
    printf("%s Matrix:\n", name);
    for (int i = 0; i < PROCESS_COUNT; i++) {
        for (int j = 0; j < RESOURCE_COUNT; j++) {
            printf("%d ", matrix[i][j]);
```

```
        }
        printf("\n");
    }
}
void displayAvailable() {
    printf("Available Resources:\n");
    for (int i = 0; i < RESOURCE_COUNT; i++) {
        printf("%d ", available[i]);
    }
    printf("\n");
}
int main() {
    int choice;
    while (1) {
        printf("\nMenu:\n");
        printf("1. Accept Available Resources\n");
        printf("2. Display Allocation and Max\n");
        printf("3. Find and Display Need\n");
        printf("4. Display Available Resources\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter available resources for A, B, and C: ");
                for (int i = 0; i < RESOURCE_COUNT; i++) {
                    scanf("%d", &available[i]);
                }
                break;
            case 2:
                displayMatrix(allocation, "Allocation");
                displayMatrix(max, "Max");
                break;
            case 3:
                calculateNeedMatrix();
                displayMatrix(need, "Need");
```

```
            break;
        case 4:
            displayAvailable();
            break;
        case 5:
            return 0;
        default:
            printf("Invalid choice!\n");
        }
    }
    return 0;
}
```

Q.2 Write a simulation program for disk scheduling using SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 86, 147, 91, 170, 95, 130, 102, 70 Starting Head position= 125 Direction: Left

```
#include <stdio.h>
#include <stdlib.h>
void sort(int arr[], int n) {
    int temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
void scan(int requests[], int n, int head, int direction, int disk_size) {
    int total_movement = 0;
```

```
int current = head;
// Sort the requests in ascending order
sort(requests, n);
int index;
for (index = 0; index < n; index++) {
    if (requests[index] > head) {
        break;
    }
}
printf("Order of servicing requests: ");
if (direction == 0) { // Move left
    for (int i = index - 1; i >= 0; i--) {
        printf("%d ", requests[i]);
        total_movement += abs(current - requests[i]);
        current = requests[i];
    }
    total_movement += current;
    current = 0;
    for (int i = index; i < n; i++) {
        printf("%d ", requests[i]);
        total_movement += abs(current - requests[i]);
        current = requests[i];
    }
} else { // Move right
    for (int i = index; i < n; i++) {
        printf("%d ", requests[i]);
        total_movement += abs(current - requests[i]);
        current = requests[i];
    }
    total_movement += abs(disk_size - 1 - current);
    current = disk_size - 1;
    for (int i = index - 1; i >= 0; i--) {
        printf("%d ", requests[i]);
        total_movement += abs(current - requests[i]);
        current = requests[i];
    }
```

```
    }
    printf("\nTotal head movement: %d\n", total_movement);
}
int main() {
    int n, head, direction, disk_size;
    printf("Enter total number of disk blocks: ");
    scanf("%d", &disk_size);
    printf("Enter number of requests: ");
    scanf("%d", &n);
    int requests[n];
    printf("Enter the request string: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }
    printf("Enter starting head position: ");
    scanf("%d", &head);
    printf("Enter direction (0 for Left, 1 for Right): ");
    scanf("%d", &direction);
    scan(requests, n, head, direction, disk_size);
    return 0;
}
```

Slip no 5

Q.1 Consider a system with ‘m’ processes and ‘n’ resource types. Accept number of instances for every resource type. For each process accept the allocation and maximum requirement matrices. Write a program to display the contents of need matrix and to check if the given request of a process can be granted immediately or not

```
#include <stdio.h>
int main() {
    int m, n;
    printf("Enter number of processes (m): ");
    scanf("%d", &m);
    printf("Enter number of resource types (n): ");
```

```
scanf("%d", &n);
int allocation[m][n], max[m][n], need[m][n], available[n];
int request[n], process;
// Input Available resources for each resource type
printf("Enter the number of available instances for each resource type: \n");
for (int i = 0; i < n; i++) {
    printf("Resource %d: ", i + 1);
    scanf("%d", &available[i]);
}
// Input Allocation matrix
printf("Enter the Allocation matrix:\n");
for (int i = 0; i < m; i++) {
    printf("Process %d Allocation: ", i);
    for (int j = 0; j < n; j++) {
        scanf("%d", &allocation[i][j]);
    }
}
// Input Maximum requirement matrix
printf("Enter the Maximum requirement matrix:\n");
for (int i = 0; i < m; i++) {
    printf("Process %d Maximum: ", i);
    for (int j = 0; j < n; j++) {
        scanf("%d", &max[i][j]);
    }
}
// Calculate the Need matrix (Need = Max - Allocation)
for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        need[i][j] = max[i][j] - allocation[i][j];
    }
}
// Display the Need matrix
printf("\nNeed Matrix:\n");
for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        printf("%d ", need[i][j]);
```

```
        }
        printf("\n");
    }
    // Accept a request from a process
    printf("\nEnter the process number making the request (0 to %d): ", m - 1);
    scanf("%d", &process);
    printf("Enter the resource request for process %d:\n", process);
    for (int i = 0; i < n; i++) {
        printf("Resource %d: ", i + 1);
        scanf("%d", &request[i]);
    }
    // Check if the request can be granted immediately
    int can_be_granted = 1;
    // Check if the request is less than the available resources
    for (int i = 0; i < n; i++) {
        if (request[i] > available[i]) {
            can_be_granted = 0;
            break;
        }
    }
    // Check if the request is less than the need for the process
    for (int i = 0; i < n; i++) {
        if (request[i] > need[process][i]) {
            can_be_granted = 0;
            break;
        }
    }
    if (can_be_granted) {
        printf("\nThe request can be granted immediately.\n");
    } else {
        printf("\nThe request cannot be granted immediately.\n");
    }
    return 0;
}
```

Q.2 Write an MPI program to find the max number from randomly generated 1000 numbers (stored in array) on a cluster (Hint: Use MPI_Reduce)

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <time.h>
#define N 1000
int main(int argc, char *argv[]) {
    int rank, size;
    int arr[N], local_max, global_max;
    int local_start, local_end;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    srand(time(NULL) + rank);
    int local_array_size = N / size;
    int local_array[local_array_size];
    if (rank == 0) {
        for (int i = 0; i < N; i++) {
            arr[i] = rand() % 1000;
        }
    }
    MPI_Scatter(arr, local_array_size, MPI_INT, local_array, local_array_size, MPI_INT,
0, MPI_COMM_WORLD);
    local_max = local_array[0];
    for (int i = 1; i < local_array_size; i++) {
        if (local_array[i] > local_max) {
            local_max = local_array[i];
        }
    }
    MPI_Reduce(&local_max, &global_max, 1, MPI_INT, MPI_MAX, 0,
MPI_COMM_WORLD);
    if (rank == 0) {
        printf("The maximum value is %d\n", global_max);
```

```
    }
    MPI_Finalize();
    return 0;
}
```

Slip no 6

Q.1 Write a program to simulate Linked file allocation method. Assume disk with n number of blocks. Give value of n as input. Randomly mark some block as allocated and accordingly maintain the list of free blocks Write menu driver program with menu options as mentioned below and implement each option. • Show Bit Vector • Create New File • Show Directory • Exit

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_BLOCKS 100
int n;  // Number of blocks
int bitVector[MAX_BLOCKS];  // Bit vector for allocation
int directory[MAX_BLOCKS];  // Directory to store file block allocations
void showBitVector() {
    printf("Bit Vector (0=Free, 1=Allocated):\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", bitVector[i]);
    }
    printf("\n");
}
void createNewFile() {
    int fileSize;
    printf("Enter the size of the new file (number of blocks): ");
    scanf("%d", &fileSize);

    int allocatedBlocks = 0;
    int fileBlocks[fileSize];
    for (int i = 0; i < n && allocatedBlocks < fileSize; i++) {
        if (bitVector[i] == 0) {  // Free block found
            bitVector[i] = 1;
```

```
            fileBlocks[allocatedBlocks] = i;
            allocatedBlocks++;
        }
    }
    if (allocatedBlocks == fileSize) {
        printf("File created successfully with blocks: ");
        for (int i = 0; i < fileSize; i++) {
            printf("%d ", fileBlocks[i]);
        }
        printf("\n");
    } else {
        printf("Not enough free blocks to create the file.\n");
    }
}
void showDirectory() {
    printf("Directory (File Allocations):\n");
    for (int i = 0; i < n; i++) {
        if (bitVector[i] == 1) {
            printf("Block %d allocated\n", i);
        }
    }
}
int main() {
    int choice;
    printf("Enter the total number of blocks: ");
    scanf("%d", &n);
    // Initialize bit vector and directory
    for (int i = 0; i < n; i++) {
        bitVector[i] = 0;  // 0 means free block
        directory[i] = -1;  // No file allocated yet
    }
    do {
        printf("\nMenu:\n");
        printf("1. Show Bit Vector\n");
        printf("2. Create New File\n");
        printf("3. Show Directory\n");
```

```
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                showBitVector();
                break;
            case 2:
                createNewFile();
                break;
            case 3:
                showDirectory();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);
    return 0;
}
```

Q.2 Write a simulation program for disk scheduling using C-SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.. 80, 150, 60,135, 40, 35, 170 Starting Head Position: 70 Direction: Right

```
#include <stdio.h>
#include <stdlib.h>
void sortRequests(int requests[], int n) {
    int temp;
    for (int i = 0; i < n-1; i++) {
        for (int j = i+1; j < n; j++) {
```

```
            if (requests[i] > requests[j]) {
                temp = requests[i];
                requests[i] = requests[j];
                requests[j] = temp;
            }
        }
    }
}
void cscan(int requests[], int n, int head, int diskSize, char direction) {
    int totalHeadMovements = 0;
    int index = 0;
    int servedRequests[n];

    sortRequests(requests, n);

    if (direction == 'R') {
        for (int i = 0; i < n; i++) {
            if (requests[i] >= head) {
                index = i;
                break;
            }
        }

        for (int i = index; i < n; i++) {
            servedRequests[i - index] = requests[i];
        }

        for (int i = 0; i < index; i++) {
            servedRequests[n - index + i] = requests[i];
        }

        totalHeadMovements += (diskSize - 1 - head) + (diskSize - 1) - servedRequests[n-
1];

        printf("Requests in the order they are served:\n");
        for (int i = 0; i < n; i++) {
```

```
            printf("%d ", servedRequests[i]);
        }
    }
    printf("\nTotal head movements: %d\n", totalHeadMovements);
}
int main() {
    int requests[] = {80, 150, 60, 135, 40, 35, 170};
    int n = sizeof(requests) / sizeof(requests[0]);
    int head, diskSize = 200;
    char direction;
    printf("Enter the starting head position: ");
    scanf("%d", &head);
    printf("Enter the direction (R for Right, L for Left): ");
    scanf(" %c", &direction);
    cscan(requests, n, head, diskSize, direction);
    return 0;
}
```

Slip no 7

Q.1 Consider the following snapshot of the system. Proces s Allocation Max Available A B C D A B C D A B C D

P0 2 0 0 1 4 2 1 2 3 3 2 1

P1 3 1 2 1 5 2 5 2

P2 2 1 0 3 2 3 1 6

P3 1 3 1 2 1 4 2 4

P4 1 4 3 2 3 6 6 5

Using Resource –Request algorithm to Check whether the current system is in safe state or not

```
#include <stdio.h>
#include <stdbool.h>
#define P 5
#define R 4
void calculateNeed(int need[P][R], int max[P][R], int allocation[P][R]) {
```

```
    for (int i = 0; i < P; i++) {
        for (int j = 0; j < R; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
}
bool isSafe(int available[], int max[][R], int allocation[][R], int need[][R]) {
    int work[R];
    bool finish[P] = {false};

    for (int i = 0; i < R; i++) {
        work[i] = available[i];
    }

    while (true) {
        bool found = false;
        for (int i = 0; i < P; i++) {
            if (!finish[i]) {
                bool canAllocate = true;
                for (int j = 0; j < R; j++) {
                    if (need[i][j] > work[j]) {
                        canAllocate = false;
                        break;
                    }
                }
                if (canAllocate) {
                    for (int j = 0; j < R; j++) {
                        work[j] += allocation[i][j];
                    }
                    finish[i] = true;
                    found = true;
                    break;
                }
            }
        }
        if (!found) {
```

```
            break;
        }
    }

    for (int i = 0; i < P; i++) {
        if (!finish[i]) {
            return false;
        }
    }
    return true;
}
int main() {
    int allocation[P][R] = {
        {2, 0, 0, 1},
        {3, 1, 2, 1},
        {2, 1, 0, 3},
        {1, 3, 1, 2},
        {1, 4, 3, 2}
    };
    int max[P][R] = {
        {4, 2, 1, 2},
        {5, 2, 5, 2},
        {2, 3, 1, 6},
        {1, 4, 2, 4},
        {3, 6, 6, 5}
    };
    int available[R] = {3, 3, 2, 1};
    int need[P][R];

    calculateNeed(need, max, allocation);

    if (isSafe(available, max, allocation, need)) {
        printf("The system is in a safe state.\n");
    } else {
        printf("The system is not in a safe state.\n");
    }
```

```
    return 0;
}
```

Q.2 Write a simulation program for disk scheduling using SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 82, 170, 43, 140, 24, 16, 190 Starting Head Position: 50 Direction: Right

```
#include <stdio.h>
#include <stdlib.h>
void sortRequests(int requests[], int n) {
    int temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (requests[i] > requests[j]) {
                temp = requests[i];
                requests[i] = requests[j];
                requests[j] = temp;
            }
        }
    }
}
void scan(int requests[], int n, int head, int diskSize, char direction) {
    int totalHeadMovements = 0;
    int left[100], right[100];
    int leftCount = 0, rightCount = 0;

    for (int i = 0; i < n; i++) {
        if (requests[i] < head) {
            left[leftCount++] = requests[i];
        } else {
            right[rightCount++] = requests[i];
        }
    }
```

```
    sortRequests(left, leftCount);
    sortRequests(right, rightCount);
    if (direction == 'R') {
        totalHeadMovements += (diskSize - 1 - head);
        for (int i = 0; i < rightCount; i++) {
            totalHeadMovements += abs(head - right[i]);
            head = right[i];
        }
        totalHeadMovements += (head - 0);
        head = 0;
        for (int i = leftCount - 1; i >= 0; i--) {
            totalHeadMovements += abs(head - left[i]);
            head = left[i];
        }
    }
    printf("Requests in the order they are served:\n");
    for (int i = 0; i < rightCount; i++) {
        printf("%d ", right[i]);
    }
    for (int i = leftCount - 1; i >= 0; i--) {
        printf("%d ", left[i]);
    }
    printf("\nTotal head movements: %d\n", totalHeadMovements);
}
int main() {
    int requests[] = {82, 170, 43, 140, 24, 16, 190};
    int n = sizeof(requests) / sizeof(requests[0]);
    int head, diskSize = 200;
    char direction;
    printf("Enter the starting head position: ");
    scanf("%d", &head);
    printf("Enter the direction (R for Right, L for Left): ");
    scanf(" %c", &direction);
    scan(requests, n, head, diskSize, direction);
    return 0;
}
```

Slip no 8

Q.1 Write a program to simulate Contiguous file allocation method. Assume disk with n number of blocks. Give value of n as input. Randomly mark some block as allocated and accordingly maintain the list of free blocks Write menu driver program with menu options as mentioned above and implement each option. • Show Bit Vector • Create New File • Show Directory • Exit

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_BLOCKS 100
int n;
int bitVector[MAX_BLOCKS];
int directory[MAX_BLOCKS];
void showBitVector() {
    printf("Bit Vector: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", bitVector[i]);
    }
    printf("\n");
}
void createNewFile(int fileSize) {
    int start = -1;
    for (int i = 0; i <= n - fileSize; i++) {
        int found = 1;
        for (int j = i; j < i + fileSize; j++) {
            if (bitVector[j] == 1) {
                found = 0;
                break;
            }
        }
        if (found) {
            start = i;
            break;
```

```
        }
    }
    if (start == -1) {
        printf("Not enough space to allocate the file.\n");
        return;
    }
    for (int i = start; i < start + fileSize; i++) {
        bitVector[i] = 1;
        directory[i] = 1;
    }
    printf("File allocated starting from block %d.\n", start);
}
void showDirectory() {
    printf("Directory: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", directory[i]);
    }
    printf("\n");
}
int main() {
    int choice, fileSize;

    printf("Enter the number of blocks in the disk: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        bitVector[i] = 0;
        directory[i] = 0;
    }
    while (1) {
        printf("\nMenu:\n");
        printf("1. Show Bit Vector\n");
        printf("2. Create New File\n");
        printf("3. Show Directory\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                showBitVector();
                break;
            case 2:
                printf("Enter the size of the file to create: ");
                scanf("%d", &fileSize);
                createNewFile(fileSize);
                break;
            case 3:
                showDirectory();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

Q.2 Write a simulation program for disk scheduling using SSTF algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 186, 89, 44, 70, 102, 22, 51, 124 Start Head Position: 70

```
#include <stdio.h>
#include <stdlib.h>
void sortRequests(int requests[], int n, int head) {
    int temp;
    int minDist, minIndex;
    for (int i = 0; i < n - 1; i++) {
        minDist = abs(requests[i] - head);
        minIndex = i;
        for (int j = i + 1; j < n; j++) {
```

```
            if (abs(requests[j] - head) < minDist) {
                minDist = abs(requests[j] - head);
                minIndex = j;
            }
        }
        if (minIndex != i) {
            temp = requests[i];
            requests[i] = requests[minIndex];
            requests[minIndex] = temp;
        }
    }
}
void sstf(int requests[], int n, int head) {
    int totalHeadMovements = 0;
    int served[n];
    for (int i = 0; i < n; i++) {
        served[i] = 0;
    }
    printf("Request order: ");
    for (int i = 0; i < n; i++) {
        int minDist = -1;
        int index = -1;
        for (int j = 0; j < n; j++) {
            if (!served[j]) {
                int dist = abs(requests[j] - head);
                if (minDist == -1 || dist < minDist) {
                    minDist = dist;
                    index = j;
                }
            }
        }
        served[index] = 1;
        totalHeadMovements += minDist;
        head = requests[index];
        printf("%d ", requests[index]);
    }
```

```
    printf("\nTotal head movements: %d\n", totalHeadMovements);
}
int main() {
    int requests[] = {186, 89, 44, 70, 102, 22, 51, 124};
    int n = sizeof(requests) / sizeof(requests[0]);
    int head;
    printf("Enter the starting head position: ");
    scanf("%d", &head);
    sstf(requests, n, head);
    return 0;
}
\
```

Slip no 9 repeat please refer previous slips

Slip no 10

Q.1 Write an MPI program to calculate sum and average of randomly generated 1000 numbers (stored in array) on a cluster

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define SIZE 1000
int main(int argc, char *argv[]) {
    int rank, size;
    int numbers[SIZE], local_sum = 0, total_sum = 0;
    float average;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    // Generate random numbers on the root process
    if (rank == 0) {
        for (int i = 0; i < SIZE; i++) {
            numbers[i] = rand() % 1000;  // Random numbers between 0 and 999
```

```
        }
    }
    // Distribute the data among all processes
    int chunk_size = SIZE / size;
    int local_numbers[chunk_size];
    MPI_Scatter(numbers, chunk_size, MPI_INT, local_numbers, chunk_size, MPI_INT,
0, MPI_COMM_WORLD);
    // Calculate local sum
    for (int i = 0; i < chunk_size; i++) {
        local_sum += local_numbers[i];
    }
    // Reduce the local sums to compute the total sum
    MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);
    // Calculate average on the root process
    if (rank == 0) {
        average = (float)total_sum / SIZE;
        printf("Total Sum: %d\n", total_sum);
        printf("Average: %.2f\n", average);
    }
    MPI_Finalize();
    return 0;
}
```

Q.2 Write a simulation program for disk scheduling using C-SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 33, 99, 142, 52, 197, 79, 46, 65 Start Head Position: 72 Direction: Left

```
#include <stdio.h>
#include <stdlib.h>
void sortRequests(int requests[], int n) {
    int temp;
    for (int i = 0; i < n - 1; i++) {
```

```
        for (int j = i + 1; j < n; j++) {
            if (requests[i] > requests[j]) {
                temp = requests[i];
                requests[i] = requests[j];
                requests[j] = temp;
            }
        }
    }
}
void cscan(int requests[], int n, int head, int total_blocks, int direction) {
    int total_head_movements = 0;
    int served[n];
    for (int i = 0; i < n; i++) {
        served[i] = 0;
    }
    sortRequests(requests, n);

    int left[n], right[n], left_count = 0, right_count = 0;
    for (int i = 0; i < n; i++) {
        if (requests[i] < head) {
            left[left_count++] = requests[i];
        } else {
            right[right_count++] = requests[i];
        }
    }
    if (direction == 0) {
        for (int i = left_count - 1; i >= 0; i--) {
            total_head_movements += abs(head - left[i]);
            head = left[i];
            printf("%d ", head);
        }
        total_head_movements += abs(head - 0);
        head = 0;
        for (int i = 0; i < right_count; i++) {
            total_head_movements += abs(head - right[i]);
            head = right[i];
```

```
            printf("%d ", head);
        }
    } else {
        for (int i = 0; i < right_count; i++) {
            total_head_movements += abs(head - right[i]);
            head = right[i];
            printf("%d ", head);
        }
        total_head_movements += abs(head - (total_blocks - 1));
        head = total_blocks - 1;
        for (int i = left_count - 1; i >= 0; i--) {
            total_head_movements += abs(head - left[i]);
            head = left[i];
            printf("%d ", head);
        }
    }
    printf("\nTotal head movements: %d\n", total_head_movements);
}
int main() {
    int requests[] = {33, 99, 142, 52, 197, 79, 46, 65};
    int n = sizeof(requests) / sizeof(requests[0]);
    int head, direction;
    printf("Enter the starting head position: ");
    scanf("%d", &head);
    printf("Enter the direction (0 for left, 1 for right): ");
    scanf("%d", &direction);
    cscan(requests, n, head, 200, direction);
    return 0;
}
```

Slip no 11 repeated please refer to previous slips

Slip no 12 repeated please refer to previous slips

Slip no 13 repeated please refer to previous slips

Slip no 14

Q.1 Write a program to simulate Sequential (Contiguous) file allocation method. Assume disk with n number of blocks. Give value of n as input. Randomly mark some block as allocated and accordingly maintain the list of free blocks Write menu driver program with menu options as mentioned below and implement each option.

- Show Bit Vector
- Show Directory
- Delete File
- Exit

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_BLOCKS 100
int disk[MAX_BLOCKS];
int n;
void showBitVector() {
    printf("Bit Vector: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", disk[i]);
    }
    printf("\n");
}
void showDirectory() {
    printf("Directory (Allocated blocks): ");
    for (int i = 0; i < n; i++) {
        if (disk[i] == 1) {
            printf("%d ", i);
        }
    }
    printf("\n");
}
void deleteFile(int start, int length) {
    int i;
```

```
    for (i = start; i < start + length && i < n; i++) {
        if (disk[i] == 1) {
            disk[i] = 0;
        }
    }
    printf("File deleted from blocks %d to %d\n", start, i-1);
}
int main() {
    int option, start, length;

    printf("Enter the total number of disk blocks: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        disk[i] = rand() % 2;
    }
    while (1) {
        printf("\nMenu:\n");
        printf("1. Show Bit Vector\n");
        printf("2. Show Directory\n");
        printf("3. Delete File\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &option);
        switch (option) {
            case 1:
                showBitVector();
                break;
            case 2:
                showDirectory();
                break;
            case 3:
                printf("Enter the starting block and length of the file to delete: ");
                scanf("%d %d", &start, &length);
                deleteFile(start, length);
                break;
```

```
            case 4:
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

Q.2 Write a simulation program for disk scheduling using SSTF algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 55, 58, 39, 18, 90, 160, 150, 38, 184 Start Head Position: 50

```
#include <stdio.h>
#include <stdlib.h>
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
int findClosestRequest(int requests[], int n, int head) {
    int min_dist = 99999, index = -1;
    for (int i = 0; i < n; i++) {
        if (requests[i] != -1) {
            int dist = abs(requests[i] - head);
            if (dist < min_dist) {
                min_dist = dist;
                index = i;
            }
        }
    }
    return index;
}
void sstf(int requests[], int n, int head) {
```

```
    int total_head_movements = 0;
    int served[n];
    for (int i = 0; i < n; i++) {
        served[i] = 0;
    }
    printf("Disk Requests Order: ");
    for (int i = 0; i < n; i++) {
        int index = findClosestRequest(requests, n, head);
        total_head_movements += abs(head - requests[index]);
        head = requests[index];
        requests[index] = -1;  // Mark the request as served
        printf("%d ", head);
    }
    printf("\nTotal head movements: %d\n", total_head_movements);
}
int main() {
    int requests[] = {55, 58, 39, 18, 90, 160, 150, 38, 184};
    int n = sizeof(requests) / sizeof(requests[0]);
    int head;
    printf("Enter the starting head position: ");
    scanf("%d", &head);
    sstf(requests, n, head);
    return 0;
}
```

Slip no 15 repeated please refer to previous slips

Q.1 Write a program to simulate Linked file allocation method. Assume disk with n number of blocks. Give value of n as input. Randomly mark some block as allocated and accordingly maintain the list of free blocks Write menu driver program with menu options as mentioned below and implement each option.

- Show Bit Vector
- Create New File
- Show Directory

• Exit

Q.2 Write a simulation program for disk scheduling using C-SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.. 80, 150, 60,135, 40, 35, 170 Starting Head Position: 70 Direction: Right

Slip no 16 repeated please refer to previous slips

Q.1 Write a program to simulate Sequential (Contiguous) file allocation method. Assume disk with n number of blocks. Give value of n as input. Randomly mark some block as allocated and accordingly maintain the list of free blocks Write menu driver program with menu options as mentioned below and implement each option

• Show Bit Vector
• Create New File
• Show Directory
• Exit

Q.2 Write an MPI program to find the min number from randomly generated 1000 numbers (stored in array) on a cluster (Hint: Use MPI_Reduce)

Slip no 17 repeated please refer to previous slips

Q.1 Write a program to simulate Indexed file allocation method. Assume disk with n number of blocks. Give value of n as input. Randomly mark some block as allocated and accordingly maintain the list of free blocks Write menu driver program with menu options as mentioned above and implement each option.

• Show Bit Vector
• Show Directory
• Delete Already File
• Exit

Q.2 Write a simulation program for disk scheduling using LOOK algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 23, 89, 132, 42, 187, 69, 36, 55
Start Head Position: 40
Direction: Left

Slip no 18 repeated please refer to previous slips

Q.1 Write a program to simulate Indexed file allocation method. Assume disk with n number of blocks. Give value of n as input. Randomly mark some block as allocated and accordingly maintain the list of free blocks Write menu driver program with menu options as mentioned above and implement each option.

- Show Bit Vector
- Create New File
- Show Directory
- Delete File
- Exit

Q.2 Write a simulation program for disk scheduling using SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 33, 99, 142, 52, 197, 79, 46, 65
Start Head Position: 72
Direction: Right

Slip no 19 repeated please refer to previous slips

Q.1 Write a C program to simulate Banker's algorithm for the purpose of deadlock avoidance.Consider the following snapshot of system, A, B, C and D is the resource type. Proces s Allocation Max Available
A B C D A B C D A B C D
P0 0 3 2 4 6 5 4 4 3 4 4 2
P1 1 2 0 1 4 4 4 4
P2 0 0 0 0 0 0 1 2

P3 3 3 2 2 3 9 3 4
P4 1 4 3 2 2 5 3 3
P5 2 4 1 4 4 6 3 4
a) Calculate and display the content of need matrix?
b) Is the system in safe state? If display the safe sequence.

Q.2 Write a simulation program for disk scheduling using C-SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 23, 89, 132, 42, 187, 69, 36, 55 Start Head Position: 40 Direction: Left

Slip no 20 repeated please refer to previous slips

Q.1 Write a simulation program for disk scheduling using SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 33, 99, 142, 52, 197, 79, 46, 65 Start Head Position: 72 Direction: User defined

Q.2 Write an MPI program to find the max number from randomly generated 1000 numbers (stored in array) on a cluster (Hint: Use MPI_Reduce)

Slip no 21

Q.1 Write a simulation program for disk scheduling using FCFS algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 55, 58, 39, 18, 90, 160, 150, 38, 184 Start Head Position: 50

```
#include <stdio.h>
#include <stdlib.h>
```

```
void fcfs(int requests[], int n, int head) {
    int total_head_movements = 0;

    printf("Disk Requests Order: ");
    for (int i = 0; i < n; i++) {
        total_head_movements += abs(head - requests[i]);
        head = requests[i];
        printf("%d ", head);
    }
    printf("\nTotal head movements: %d\n", total_head_movements);
}
int main() {
    int requests[] = {55, 58, 39, 18, 90, 160, 150, 38, 184};
    int n = sizeof(requests) / sizeof(requests[0]);
    int head;
    printf("Enter the starting head position: ");
    scanf("%d", &head);
    fcfs(requests, n, head);
    return 0;
}
```

Q.2 Write an MPI program to calculate sum of all even randomly generated 1000 numbers (stored in array) on a cluster

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
int main(int argc, char *argv[]) {
    int rank, size, n = 1000;
    int numbers[n], local_sum = 0, global_sum = 0;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (rank == 0) {
        for (int i = 0; i < n; i++) {
```

```
            numbers[i] = rand() % 1000;
        }
    }
    MPI_Bcast(numbers, n, MPI_INT, 0, MPI_COMM_WORLD);
    for (int i = rank; i < n; i += size) {
        if (numbers[i] % 2 == 0) {
            local_sum += numbers[i];
        }
    }
    MPI_Reduce(&local_sum, &global_sum, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);
    if (rank == 0) {
        printf("Sum of all even numbers: %d\n", global_sum);
    }
    MPI_Finalize();
    return 0;
}
```

Slip no 22

Q.1 Write an MPI program to calculate sum of all odd randomly generated 1000 numbers (stored in array) on a cluster.

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
int main(int argc, char *argv[]) {
    int rank, size, n = 1000;
    int numbers[n], local_sum = 0, global_sum = 0;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (rank == 0) {
        for (int i = 0; i < n; i++) {
            numbers[i] = rand() % 1000;
```

```
        }
    }
    MPI_Bcast(numbers, n, MPI_INT, 0, MPI_COMM_WORLD);
    for (int i = rank; i < n; i += size) {
        if (numbers[i] % 2 != 0) {
            local_sum += numbers[i];
        }
    }
    MPI_Reduce(&local_sum, &global_sum, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);
    if (rank == 0) {
        printf("Sum of all odd numbers: %d\n", global_sum);
    }
    MPI_Finalize();
    return 0;
}
```

Q.2 Write a program to simulate Sequential (Contiguous) file allocation method. Assume disk with n number of blocks. Give value of n as input. Randomly mark some block as allocated and accordingly maintain the list of free blocks Write menu driver program with menu options as mentioned below and implement each option • Show Bit Vector • Delete already created file • Exit

ANS == Please refer previous slips

Slip no 23

Q.1 Consider a system with ‘m’ processes and ‘n’ resource types. Accept number of instances for every resource type. For each process accept the allocation and maximum requirement matrices. Write a program to display the contents of need matrix and to check if the given request of a process can be granted immediately or not

```
#include <stdio.h>
#include <stdlib.h>
```

```
void displayMatrix(int matrix[][10], int m, int n) {
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}
void calculateNeedMatrix(int allocation[][10], int max[][10], int need[][10], int m, int n) {
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
}
int canGrantRequest(int request[], int available[], int need[], int n) {
    for (int i = 0; i < n; i++) {
        if (request[i] > need[i]) {
            return 0;
        }
        if (request[i] > available[i]) {
            return 0;
        }
    }
    return 1;
}
int main() {
    int m, n;
    printf("Enter the number of processes: ");
    scanf("%d", &m);
    printf("Enter the number of resource types: ");
    scanf("%d", &n);
    int allocation[m][n], max[m][n], available[n], need[m][n], request[n];
    printf("Enter the Allocation Matrix:\n");
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
```

```
            scanf("%d", &allocation[i][j]);
        }
    }
    printf("Enter the Maximum Matrix:\n");
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &max[i][j]);
        }
    }
    printf("Enter the Available resources:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &available[i]);
    }
    calculateNeedMatrix(allocation, max, need, m, n);
    printf("Need Matrix:\n");
    displayMatrix(need, m, n);
    printf("Enter the request from process: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &request[i]);
    }
    if (canGrantRequest(request, available, need[0], n)) {
        printf("The request can be granted immediately.\n");
    } else {
        printf("The request cannot be granted immediately.\n");
    }
    return 0;
}
```

Q.2 Write a simulation program for disk scheduling using SSTF algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 24, 90, 133, 43, 188, 70, 37, 55 Start Head Position: 58

Please refer previous slips

Slip no 24 repeated please refer to previous slips

Q.1 Write an MPI program to calculate sum of all odd randomly generated 1000 numbers (stored in array) on a cluster.

Q.2 Write a C program to simulate Banker's algorithm for the purpose of deadlock avoidance.The following snapshot of system, A, B, C and D are the resource type.
Proces s Allocation Max Available
A B C A B C A B C
P0 0 1 0 0 0 0 0 0
P1 2 0 0 2 0 2
P2 3 0 3 0 0 0
P3 2 1 1 1 0 0
P4 0 0 2 0 0 2

a)Calculate and display the content of need matrix?
b)Is the system in safe state? If display the safe sequence.

Slip no 25

Q.1 Write a simulation program for disk scheduling using LOOK algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 86, 147, 91, 170, 95, 130, 102, 70 Starting Head position= 125 Direction: User Defined

```
#include <stdio.h>
#include <stdlib.h>
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
void sortRequests(int requests[], int n) {
    for (int i = 0; i < n - 1; i++) {
```

```
        for (int j = 0; j < n - i - 1; j++) {
            if (requests[j] > requests[j + 1]) {
                swap(&requests[j], &requests[j + 1]);
            }
        }
    }
}
void look(int requests[], int n, int head, int direction) {
    int total_head_movements = 0;
    int served[n];
    for (int i = 0; i < n; i++) {
        served[i] = 0;
    }
    sortRequests(requests, n);
    int start = (direction == 1) ? 0 : n - 1;
    int end = (direction == 1) ? n - 1 : 0;
    int step = (direction == 1) ? 1 : -1;
    printf("Disk Requests Order: ");
    for (int i = start; (direction == 1 && i <= end) || (direction == 0 && i >= end); i += step) {
        total_head_movements += abs(head - requests[i]);
        head = requests[i];
        printf("%d ", head);
    }
    printf("\nTotal head movements: %d\n", total_head_movements);
}
int main() {
    int requests[] = {86, 147, 91, 170, 95, 130, 102, 70};
    int n = sizeof(requests) / sizeof(requests[0]);
    int head, direction;
    printf("Enter the starting head position: ");
    scanf("%d", &head);
    printf("Enter the direction (1 for right, 0 for left): ");
    scanf("%d", &direction);
    look(requests, n, head, direction);
    return 0;
}
```

Q.2 Write a program to simulate Linked file allocation method. Assume disk with n number of blocks. Give value of n as input. Randomly mark some block as allocated and accordingly maintain the list of free blocks Write menu driver program with menu options as mentioned below and implement each option.

• Show Bit Vector
• Create New File
• Show Directory
• Exit

repeated please refer to previous slips

Slip no 26 repeated please refer to previous slips

Q.1 Write a C program to simulate Banker's algorithm for the purpose of deadlock avoidance.Consider the following snapshot of system, A, B, C and D is the resource type. Proces s Allocation Max Available

A B C D A B C D A B C D

P0 0 0 1 2 0 0 1 2 1 5 2 0

P1 1 0 0 0 1 7 5 0

P2 1 3 5 4 2 3 5 6

P3 0 6 3 2 0 6 5 2

P4 0 0 1 4 0 6 5 6

a)Calculate and display the content of need matrix?
b)Is the system in safe state? If display the safe sequence.

Q.2 Write a simulation program for disk scheduling using FCFS algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 56, 59, 40, 19, 91, 161, 151, 39, 185 Start Head Position: 48

Slip no 27 repeated please refer to previous slips

Q.1 Write a simulation program for disk scheduling using LOOK algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 176, 79, 34, 60, 92, 11, 41, 114 Starting Head Position: 65 Direction: Right

Q.2 Write an MPI program to find the min number from randomly generated 1000 numbers (stored in array) on a cluster (Hint: Use MPI_Reduce)

Slip no 28 repeated please refer to previous slips

Q.1 Write a simulation program for disk scheduling using C-LOOK algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 56, 59, 40, 19, 91, 161, 151, 39, 185 Start Head Position: 48 Direction: User Defined

Q.2 Write an MPI program to calculate sum of randomly generated 1000 numbers (stored in array) on a cluster

Slip no 29 repeated please refer to previous slips

Q.1 Write an MPI program to calculate sum of all even randomly generated 1000 numbers (stored in array) on a cluster.

Q.2 Write a simulation program for disk scheduling using C-LOOK algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.. [15] 80, 150, 60,135, 40, 35, 170 Starting Head Posi

Slip no 30 repeated please refer to previous slips

Q.1 Write an MPI program to find the min number from randomly generated 1000 numbers (stored in array) on a cluster (Hint: Use MPI_Reduce)

Q.2 Write a simulation program for disk scheduling using FCFS algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. 65, 95, 30, 91, 18, 116, 142, 44, 168 Start Head Position: 52