Slip 16

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

Ans:-

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX_BLOCKS 1000
int bit_vector[MAX_BLOCKS];
void initialize(int n) {
    srand(time(NULL)); // Seed the random number generator with the current time
    for (int i = 0; i < n; i++) {
        if (rand() % 2 == 0) {
            bit_vector[i] = 1; // Mark block as allocated
        } else {
            bit_vector[i] = 0; // Mark block as free
        }
    }
}

void show_bit_vector(int n) {
    printf("Block Number\tStatus\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t", i);
        if (bit_vector[i] == 1) {
            printf("Allocated\n");
        } else {
            printf("Free\n");
        }
    }
}

void create_new_file(int n) {
    int start_block, num_blocks;
    printf("Enter the starting block number: ");
    scanf("%d", &start_block);
```

```
    printf("Enter the number of blocks needed: ");
    scanf("%d", &num_blocks);
    int i;
    for (i = start_block; i < start_block + num_blocks; i++) {
        if (i >= n || bit_vector[i] == 1) {
            printf("Error: Cannot allocate file in the specified blocks.\n");
            break;
        }
    }
    if (i == start_block + num_blocks) {
        printf("File created successfully.\n");
        for (i = start_block; i < start_block + num_blocks; i++) {
            bit_vector[i] = 1; // Mark blocks as allocated
        }
    }
}

void show_directory(int n) {
    printf("File Name\tStarting Block\tNumber of Blocks\n");
    // TODO: Implement directory functionality here
}

int main() {
    int n, choice;
    printf("Enter the number of blocks in the disk: ");
    scanf("%d", &n);
    initialize(n);
    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:
                show_bit_vector(n);
                break;
            case 2:
                create_new_file(n);
                break;
            case 3:
                show_directory(n);
```

```
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Error: Invalid choice.\n");
            break;
    }
  } while (choice != 4);
  return 0;
}
```

Q2. Write an MPI program to find the min number from randomly generated 1000 numbers (stored in array) on a cluster (Hint: Use MPI_Reduce)

Ans:-

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

#define N 1000

int main(int argc, char** argv) {
  int rank, size;
  int i, min = 0;
  int arr[N];
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  srand(rank + 1); // Seed the random number generator with the rank to generate different
numbers on each process
  for (i = 0; i < N; i++) {
    arr[i] = rand();
  }
  for (i = 0; i < N; i += size) {
    int j, end = i + size;
    if (end > N) {
      end = N;
    }
    int local_min = arr[i];
    for (j = i + 1; j < end; j++) {
      if (arr[j] < local_min) {
```

```
            local_min = arr[j];
        }
      }
      MPI_Reduce(&local_min, &min, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);
   }
   if (rank == 0) {
      printf("The minimum number is %d\n", min);
   }
   MPI_Finalize();
   return 0;
}
```

Slip No 17

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

Ans:-

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define BLOCK_SIZE 1024

int *disk;
int *free_blocks;
int *index_table;
int n_blocks;

void initialize_disk(int n) {
   /*
    * Initializes a disk with n number of blocks and randomly marks some blocks as allocated.
    */
   int i;
   disk = (int *) malloc(n * sizeof(int));
   free_blocks = (int *) malloc(n * sizeof(int));
   index_table = (int *) calloc(10, sizeof(int)); // Assume we have 10 files
```

```
    n_blocks = n;
    srand(time(NULL));
    for (i = 0; i < n; i++) {
        if ((double) rand() / RAND_MAX < 0.5) {
            disk[i] = 1;
        } else {
            disk[i] = 0;
            free_blocks[free_blocks[0] + 1] = i;
            free_blocks[0]++;
        }
    }
}

void show_bit_vector() {
    /*
     * Shows the bit vector for the disk.
     */
    int i, j;
    printf("Bit Vector:\n");
    for (i = 0; i < n_blocks; i += 10) {
        for (j = i; j < i + 10; j++) {
            if (j >= n_blocks) {
                break;
            }
            printf("%d ", disk[j]);
        }
        printf("\n");
    }
}

void show_directory() {
    /*
     * Shows the directory for the disk.
     */
    int i, j;
    printf("Directory:\n");
    for (i = 0; i < 10; i++) {
        printf("%d: ", i);
        for (j = 0; j < index_table[i]; j++) {
            printf("%d ", index_table[i * BLOCK_SIZE + j]);
        }
        printf("\n");
    }
}
```

```
void delete_file(int file_index) {
    /*
     * Deletes a file from the disk by setting its blocks to free.
     */
    int i, block_index;
    for (i = 0; i < index_table[file_index]; i++) {
        block_index = index_table[file_index * BLOCK_SIZE + i];
        disk[block_index] = 0;
        free_blocks[free_blocks[0] + 1] = block_index;
        free_blocks[0]++;
    }
    index_table[file_index] = 0;
}

int main() {
    int choice, file_index;
    printf("Enter the number of blocks on the disk: ");
    scanf("%d", &n_blocks);
    initialize_disk(n_blocks);

    while (1) {
        printf("\nMenu:\n");
        printf("1. Show Bit Vector\n");
        printf("2. Show Directory\n");
        printf("3. Delete a File\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                show_bit_vector();
                break;
            case 2:
                show_directory();
                break;
            case 3:
                printf("Enter the index of the file to delete: ");
                scanf("%d", &file_index);
                delete_file(file_index);
                printf("File %d deleted.\n", file_index);
                break;
            case 4:
```

```
            return 0;
        default:
            printf("Invalid choice. Please try again.\n");
            break;
    }
  }
}
```

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 Direction: Left
Start Head Position: 40

Ans:-

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

#define DIRECTION_LEFT 0
#define DIRECTION_RIGHT 1

int abs(int x) {
    return x >= 0 ? x : -x;
}

int compare(const void *a, const void *b) {
    return *(int *) a - *(int *) b;
}

void print_request_order(int *request_order, int n) {
    /*
     * Prints the order in which the disk requests are served.
     */
    int i;
    printf("Order of requests served: ");
    for (i = 0; i < n; i++) {
        printf("%d ", request_order[i]);
    }
    printf("\n");
}

void print_head_movements(int head_movements) {
    /*
     * Prints the total number of head movements.
```

```
    */
    printf("Total number of head movements: %d\n", head_movements);
}

int main() {
    int n_blocks, head_pos, direction, n_requests, i, j, k, head_movements = 0, min_request,
max_request, *requests, *request_order;
    printf("Enter the total number of disk blocks: ");
    scanf("%d", &n_blocks);
    printf("Enter the current head position: ");
    scanf("%d", &head_pos);
    printf("Enter the disk request string (separated by spaces): ");
    scanf("%d", &n_requests);
    requests = (int *) malloc(n_requests * sizeof(int));
    request_order = (int *) malloc(n_requests * sizeof(int));
    for (i = 0; i < n_requests; i++) {
        scanf("%d", &requests[i]);
    }
    printf("Enter the direction (0 for left, 1 for right): ");
    scanf("%d", &direction);

    qsort(requests, n_requests, sizeof(int), compare);

    for (i = 0; i < n_requests; i++) {
        if (requests[i] >= head_pos) {
            break;
        }
    }
    k = i;
    if (direction == DIRECTION_LEFT) {
        min_request = 0;
        max_request = k - 1;
        for (i = k - 1; i >= 0; i--) {
            request_order[max_request - i] = requests[i];
        }
        for (i = k; i < n_requests; i++) {
            request_order[i] = requests[i];
        }
    } else {
        min_request = k;
        max_request = n_requests - 1;
        for (i = k; i < n_requests; i++) {
            request_order[i - k] = requests[i];
        }
```

```
        for (i = k - 1; i >= 0; i--) {
            request_order[n_requests - 1 - i] = requests[i];
        }
    }

    for (i = 0; i < n_requests; i++) {
        head_movements += abs(request_order[i] - head_pos);
        head_pos = request_order[i];
    }

    print_request_order(request_order, n_requests);
    print_head_movements(head_movements);

    return 0;
}
```

Slip No 18

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

Ans:-

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

// function to display the bit vector
void showBitVector(int* bitVector, int n) {
    printf("Bit Vector: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", bitVector[i]);
    }
    printf("\n");
}

// function to create a new file
```

```
void createNewFile(int* bitVector, int n, int* directory) {
    int fileSize, numBlocks;
    printf("Enter the file size: ");
    scanf("%d", &fileSize);
    numBlocks = (fileSize + 511) / 512; // 512 bytes per block
    int freeBlocks[numBlocks];
    int freeBlockCount = 0;
    for (int i = 0; i < n; i++) {
        if (bitVector[i] == 0) {
            freeBlocks[freeBlockCount++] = i;
            if (freeBlockCount == numBlocks) {
                break;
            }
        }
    }
    if (freeBlockCount != numBlocks) {
        printf("Not enough free space!\n");
        return;
    }
    int fileId;
    for (fileId = 0; directory[fileId] != -1; fileId++);
    directory[fileId] = freeBlocks[0];
    for (int i = 0; i < numBlocks; i++) {
        bitVector[freeBlocks[i]] = 1;
        if (i < numBlocks - 1) {
            bitVector[freeBlocks[i]] = freeBlocks[i + 1];
        } else {
            bitVector[freeBlocks[i]] = -1;
        }
    }
    printf("File created with ID %d\n", fileId);
}

// function to display the directory
void showDirectory(int* directory) {
    printf("Directory:\n");
    for (int i = 0; directory[i] != -1; i++) {
        printf("%d: block %d\n", i, directory[i]);
    }
}

// function to delete a file
void deleteFile(int* bitVector, int* directory) {
    int fileId;
```

```
    printf("Enter the ID of the file to delete: ");
    scanf("%d", &fileId);
    if (directory[fileId] == -1) {
        printf("File not found!\n");
        return;
    }
    int blockId = directory[fileId];
    while (blockId != -1) {
        bitVector[blockId] = 0;
        blockId = bitVector[blockId + 1];
    }
    directory[fileId] = -1;
    printf("File deleted\n");
}

// main function
int main() {
    srand(time(NULL));
    int n;
    printf("Enter the number of blocks on the disk: ");
    scanf("%d", &n);
    int bitVector[n];
    for (int i = 0; i < n; i++) {
        bitVector[i] = rand() % 2; // randomly mark some blocks as allocated
    }
    int directory[n];
    for (int i = 0; i < n; i++) {
        directory[i] = -1; // initialize directory with -1 (empty)
    }
    int choice;
    do {
        printf("\n1. Show Bit Vector\n2. Create New File\n3. Show Directory\n4. Delete File\n5.
Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
              showBitVector(bitVector, n);
                break;
            case 2:
                createNewFile(bitVector, n, directory);
                break;
            case 3:
                showDirectory(directory);
```

```
            break;
        case 4:
            deleteFile(bitVector, directory);
            break;
        case 5:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
    }
  } while (choice != 5);
  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. 33, 99, 142, 52, 197, 79, 46, 65 Start Head Position: 72
Direction: Right

Ans:-

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

#define MAX_SIZE 100

void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}

void sort(int arr[], int size) {
  for (int i = 0; i < size - 1; i++) {
    for (int j = i + 1; j < size; j++) {
      if (arr[i] > arr[j]) {
        swap(&arr[i], &arr[j]);
      }
    }
  }
}

int main() {
```

```
    int total_blocks, head_pos, num_requests, requests[MAX_SIZE], i, j, k, pos, direction,
head_moments;

    // Accept inputs from user
    printf("Enter total number of disk blocks: ");
    scanf("%d", &total_blocks);

    printf("Enter the disk request string (up to %d requests): ", MAX_SIZE);
    for (i = 0; i < MAX_SIZE; i++) {
        scanf("%d", &requests[i]);
        if (requests[i] == -1) {
            break;
        }
    }
    num_requests = i;

    printf("Enter current head position: ");
    scanf("%d", &head_pos);

    printf("Enter direction (0 for left, 1 for right): ");
    scanf("%d", &direction);

    // Add current head position to requests
    requests[num_requests++] = head_pos;

    // Sort requests in ascending order
    sort(requests, num_requests);

    // Find index of head position
    for (i = 0; i < num_requests; i++) {
        if (requests[i] == head_pos) {
            break;
        }
    }
    pos = i;

    // Serve requests in SCAN order
    head_moments = 0;
    printf("Order of request served:\n");
    if (direction == 1) {  // Right
        for (i = pos; i < num_requests; i++) {
            printf("%d ", requests[i]);
            head_moments += abs(requests[i] - head_pos);
            head_pos = requests[i];
```

```
        }
        for (i = num_requests - 2; i >= 0; i--) {
            printf("%d ", requests[i]);
            head_moments += abs(requests[i] - head_pos);
            head_pos = requests[i];
        }
    } else {  // Left
        for (i = pos; i >= 0; i--) {
            printf("%d ", requests[i]);
            head_moments += abs(requests[i] - head_pos);
            head_pos = requests[i];
        }
        for (i = 1; i < num_requests; i++) {
            printf("%d ", requests[i]);
            head_moments += abs(requests[i] - head_pos);
            head_pos = requests[i];
        }
    }

    printf("\nTotal number of head movements: %d\n", head_moments);

    return 0;
}
```

Slip No:-20

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

Ans:-

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

#define MAX 1000

int main()
{
    int n, head, i, j, temp, total_movement = 0;
    int queue[MAX], sequence[MAX];
```

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

printf("Enter the disk request string:\n");
for (i = 0; i < n; i++)
   scanf("%d", &queue[i]);

printf("Enter the current head position: ");
scanf("%d", &head);

printf("Enter the direction (1 for right, 0 for left): ");
int direction;
scanf("%d", &direction);

// sort the request queue
for (i = 0; i < n - 1; i++)
{
   for (j = i + 1; j < n; j++)
   {
      if (queue[i] > queue[j])
      {
         temp = queue[i];
         queue[i] = queue[j];
         queue[j] = temp;
      }
   }
}

// find the head position in the request queue
int start = 0;
for (i = 0; i < n; i++)
{
   if (queue[i] >= head)
   {
      start = i;
      break;
   }
}

// if direction is right
if (direction == 1)
{
   // add right end boundary
```

```
        sequence[0] = n - 1;
        for (i = 1, j = start; j < n; i++, j++)
        {
            sequence[i] = queue[j];
        }
        // add left end boundary
        sequence[i] = 0;
        for (j = start - 1; j >= 0; i++, j--)
        {
            sequence[i] = queue[j];
        }
    }
    // if direction is left
    else
    {
        // add left end boundary
        sequence[0] = 0;
        for (i = 1, j = start - 1; j >= 0; i++, j--)
        {
            sequence[i] = queue[j];
        }
        // add right end boundary
        sequence[i] = n - 1;
        for (j = start; j < n; i++, j++)
        {
            sequence[i] = queue[j];
        }
    }

    // print the sequence and calculate the total head movements
    printf("Sequence: ");
    for (i = 0; i <= n; i++)
    {
        printf("%d ", sequence[i]);
        if (i > 0)
        {
            total_movement += abs(sequence[i] - sequence[i - 1]);
        }
    }
    printf("\nTotal head movements: %d\n", total_movement);

    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)

Ans:-

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

#define ARRAY_SIZE 1000

int main(int argc, char** argv) {
    int world_rank, world_size;
    int array[ARRAY_SIZE];
    int i, max;

    // Initialize MPI environment
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Generate random array in root process (rank 0)
    if (world_rank == 0) {
        srand(time(NULL));
        printf("Generated Array: ");
        for (i = 0; i < ARRAY_SIZE; i++) {
            array[i] = rand() % 100;
            printf("%d ", array[i]);
        }
        printf("\n");
    }

    // Scatter the array to all processes
    MPI_Scatter(array, ARRAY_SIZE / world_size, MPI_INT, array, ARRAY_SIZE / world_size,
MPI_INT, 0, MPI_COMM_WORLD);

    // Find the maximum element in each process
    max = array[0];
    for (i = 1; i < ARRAY_SIZE / world_size; i++) {
        if (array[i] > max) {
            max = array[i];
        }
    }

    // Find the maximum element across all processes using MPI_Reduce
```

```
    int global_max;
    MPI_Reduce(&max, &global_max, 1, MPI_INT, MPI_MAX, 0, MPI_COMM_WORLD);

    // Print the maximum element in root process
    if (world_rank == 0) {
        printf("Maximum Element: %d\n", global_max);
    }

    // Finalize MPI environment
    MPI_Finalize();

    return 0;
}
```

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
Ans:-

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

int main()
{
    int i, n, head, total_head_movements = 0;
    printf("Enter the total number of disk blocks: ");
    scanf("%d", &n);

    int disk_queue[n];

    printf("Enter the disk request string: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &disk_queue[i]);
    }

    printf("Enter the current head position: ");
    scanf("%d", &head);

    printf("The list of requests in the order in which it is served:\n");
```

```
    printf("%d ", head);

    for (i = 0; i < n; i++) {
        total_head_movements += abs(disk_queue[i] - head);
        head = disk_queue[i];
        printf("%d ", head);
    }

    printf("\nTotal number of head movements: %d\n", total_head_movements);

    return 0;
}
```

Sample OP:-
Enter the total number of disk blocks: 9
Enter the disk request string: 55 58 39 18 90 160 150 38 184
Enter the current head position: 50
The list of requests in the order in which it is served:
50 55 58 39 18 90 160 150 38 184
Total number of head movements: 561

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

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

#define ARRAY_SIZE 1000

int main(int argc, char** argv)
{
    int rank, size;
    int array[ARRAY_SIZE];
    int i, sum = 0, global_sum = 0;

    // Initialize MPI
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```
    // Initialize random number generator
    srand(time(NULL) + rank);

    // Fill array with random numbers
    for (i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 1000;
    }

    // Calculate local sum of even numbers
    for (i = 0; i < ARRAY_SIZE; i++) {
        if (array[i] % 2 == 0) {
            sum += array[i];
        }
    }

    // Reduce sum across all processes
    MPI_Reduce(&sum, &global_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    if (rank == 0) {
        printf("The sum of all even numbers in the array is: %d\n", global_sum);
    }

    // Finalize MPI
    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.

Ans:-

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

#define ARRAY_SIZE 1000

int main(int argc, char** argv)
{
    int rank, size;
```

```
    int array[ARRAY_SIZE];
    int i, sum = 0, global_sum = 0;

    // Initialize MPI
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Initialize random number generator
    srand(time(NULL) + rank);

    // Fill array with random numbers
    for (i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 1000;
    }

    // Calculate local sum of odd numbers
    for (i = 0; i < ARRAY_SIZE; i++) {
        if (array[i] % 2 != 0) {
            sum += array[i];
        }
    }

    // Reduce sum across all processes
    MPI_Reduce(&sum, &global_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    if (rank == 0) {
        printf("The sum of all odd numbers in the array is: %d\n", global_sum);
    }

    // Finalize MPI
    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:-

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

int *bitVector;
int totalBlocks;

void initializeBitVector() {
    bitVector = (int*) malloc(totalBlocks * sizeof(int));
    for(int i = 0; i < totalBlocks; i++) {
        bitVector[i] = rand() % 2;
    }
}

void showBitVector() {
    printf("Bit Vector:\n");
    for(int i = 0; i < totalBlocks; i++) {
        printf("%d ", bitVector[i]);
    }
    printf("\n");
}

void deleteFile() {
    int startBlock, fileSize;
    printf("Enter starting block of file to be deleted: ");
    scanf("%d", &startBlock);
    printf("Enter file size: ");
    scanf("%d", &fileSize);

    if(startBlock < 0 || startBlock >= totalBlocks || startBlock + fileSize > totalBlocks) {
        printf("Invalid input\n");
        return;
    }

    for(int i = startBlock; i < startBlock + fileSize; i++) {
        if(bitVector[i] == 0) {
            printf("Error: Block %d is not allocated\n", i);
            return;
        }
```

```
    }

    for(int i = startBlock; i < startBlock + fileSize; i++) {
        bitVector[i] = 0;
    }

    printf("File deleted successfully\n");
}

int main() {
    printf("Enter total number of blocks in disk: ");
    scanf("%d", &totalBlocks);
    initializeBitVector();

    while(true) {
        printf("\nMenu:\n1. Show Bit Vector\n2. Delete File\n3. Exit\n");
        int choice;
        scanf("%d", &choice);
        switch(choice) {
            case 1:
                showBitVector();
                break;
            case 2:
                deleteFile();
                break;
            case 3:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice\n");
        }
    }
}
```

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

Ans:-

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

```
#define MAX_PROCESS 100
#define MAX_RESOURCES 100

int allocation[MAX_PROCESS][MAX_RESOURCES];
int maximum[MAX_PROCESS][MAX_RESOURCES];
int need[MAX_PROCESS][MAX_RESOURCES];
int available[MAX_RESOURCES];
int work[MAX_RESOURCES];
int finish[MAX_PROCESS];

int m, n;

void displayNeedMatrix()
{
    printf("\nNeed Matrix:\n");

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            need[i][j] = maximum[i][j] - allocation[i][j];
            printf("%d ", need[i][j]);
        }
        printf("\n");
    }
}

int isSafeState()
{
    for (int i = 0; i < n; i++)
        work[i] = available[i];

    for (int i = 0; i < m; i++)
        finish[i] = 0;

    int count = 0;
    while (count < m) {
        int found = 0;
        for (int i = 0; i < m; i++) {
            if (!finish[i]) {
                int j;
                for (j = 0; j < n; j++) {
                    if (need[i][j] > work[j])
                        break;
                }
```

```
            if (j == n) {
                for (j = 0; j < n; j++)
                    work[j] += allocation[i][j];
                finish[i] = 1;
                found = 1;
                count++;
            }
        }
    }
    if (!found)
        break;
  }

  if (count == m)
    return 1;
  else
    return 0;
}

int requestResources(int pid, int request[])
{
  for (int i = 0; i < n; i++) {
    if (request[i] > need[pid][i])
      return 0;

    if (request[i] > available[i])
      return 0;
  }

  for (int i = 0; i < n; i++) {
    allocation[pid][i] += request[i];
    available[i] -= request[i];
  }

  if (isSafeState())
    return 1;
  else {
    for (int i = 0; i < n; i++) {
      allocation[pid][i] -= request[i];
      available[i] += request[i];
    }
    return 0;
  }
}
```

```
int main()
{
    printf("Enter the number of processes: ");
    scanf("%d", &m);

    printf("Enter the number of resource types: ");
    scanf("%d", &n);

    printf("Enter the number of instances for each resource type:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &available[i]);

    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 requirement matrix:\n");
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &maximum[i][j]);

    displayNeedMatrix();

    int pid;
    printf("Enter the process ID for which
```

*******Incomplete answer********

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

Ans:-

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

```
void sstf(int queue[], int n, int head)
{
    int i, j, min, pos, count = 0;
    int visited[n];

    for (i = 0; i < n; i++) {
        visited[i] = 0;
    }

    printf("%d -> ", head);
    visited[head] = 1;
    count++;

    while (count < n) {
        min = INT_MAX;
        for (i = 0; i < n; i++) {
            if (!visited[i]) {
                if (abs(queue[i] - head) < min) {
                    min = abs(queue[i] - head);
                    pos = i;
                }
            }
        }

        visited[pos] = 1;
        count++;
        head = queue[pos];
        printf("%d -> ", head);
    }
}

int main()
{
    int n, i, head;
    printf("Enter the total number of disk blocks: ");
    scanf("%d", &n);
    int queue[n];

    printf("Enter the disk request string:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &queue[i]);
    }

    printf("Enter the current head position: ");
```

```
    scanf("%d", &head);

    sstf(queue, n, head);

    printf("\nTotal number of head movements: %d", n);

    return 0;
}
```

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
Ans:-

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

int main() {
    int n, head, i, j, temp, total_head_movements = 0;
    printf("Enter the total number of disk blocks: ");
    scanf("%d", &n);
    int disk_queue[n];
    printf("Enter the disk request string: ");
    for(i = 0; i < n; i++) {
        scanf("%d", &disk_queue[i]);
    }
    printf("Enter the current head position: ");
    scanf("%d", &head);
    int direction;
    printf("Enter the direction (0 for left, 1 for right): ");
    scanf("%d", &direction);
    for(i = 0; i < n-1; i++) {
        for(j = i+1; j < n; j++) {
            if(disk_queue[i] > disk_queue[j]) {
                temp = disk_queue[i];
                disk_queue[i] = disk_queue[j];
                disk_queue[j] = temp;
```

```
            }
        }
    }
    int index;
    for(i = 0; i < n; i++) {
        if(disk_queue[i] >= head) {
            index = i;
            break;
        }
    }
    if(direction == 0) {
        for(i = index-1; i >= 0; i--) {
            printf("%d ", disk_queue[i]);
            total_head_movements += abs(disk_queue[i] - head);
            head = disk_queue[i];
        }
        for(i = index; i < n; i++) {
            printf("%d ", disk_queue[i]);
            total_head_movements += abs(disk_queue[i] - head);
            head = disk_queue[i];
        }
    }
    else {
        for(i = index; i < n; i++) {
            printf("%d ", disk_queue[i]);
            total_head_movements += abs(disk_queue[i] - head);
            head = disk_queue[i];
        }
        for(i = index-1; i >= 0; i--) {
            printf("%d ", disk_queue[i]);
            total_head_movements += abs(disk_queue[i] - head);
            head = disk_queue[i];
        }
    }
    printf("\nTotal number of head movements: %d", total_head_movements);
    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.

1.Show Bit Vector
2.Create New File
3.Show Directory
4.Exit
Ans:-

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

#define MAX_BLOCKS 100

typedef struct block {
    int index;
    struct block *next;
} Block;

void initialize_disk(int num_blocks, Block **free_blocks) {
    int i, num_allocated = rand() % (num_blocks / 2);
    int *allocated = malloc(sizeof(int) * num_allocated);
    for (i = 0; i < num_allocated; i++) {
        allocated[i] = rand() % num_blocks;
    }
    for (i = 0; i < num_blocks; i++) {
        Block *new_block = malloc(sizeof(Block));
        new_block->index = i;
        if (i == 0) {
            *free_blocks = new_block;
        } else {
            new_block->next = *free_blocks;
            *free_blocks = new_block;
        }
        if (i < num_allocated && allocated[i]) {
            free(new_block);
        }
    }
    free(allocated);
}

void show_bit_vector(int num_blocks, Block *free_blocks) {
    int i;
    printf("Bit Vector:\n");
    for (i = 0; i < num_blocks; i++) {
        if (free_blocks == NULL || free_blocks->index != i) {
            printf("1 ");
```

```
        } else {
            printf("0 ");
            free_blocks = free_blocks->next;
        }
    }
    printf("\n");
}

void create_new_file(int *num_files, char **file_names, int *file_sizes, Block **allocated_blocks,
Block **free_blocks) {
    char file_name[50];
    int file_size, num_blocks, i;
    printf("Enter file name: ");
    scanf("%s", file_name);
    printf("Enter file size (in blocks): ");
    scanf("%d", &num_blocks);
    file_sizes[*num_files] = num_blocks;
    file_names[*num_files] = malloc(sizeof(char) * (strlen(file_name) + 1));
    strcpy(file_names[*num_files], file_name);
    Block *prev_block = NULL;
    for (i = 0; i < num_blocks; i++) {
        if (*free_blocks == NULL) {
            printf("Error: Not enough free blocks to allocate space for file.\n");
            return;
        }
        Block *new_block = *free_blocks;
        if (prev_block == NULL) {
            *allocated_blocks = new_block;
        } else {
            prev_block->next = new_block;
        }
        *free_blocks = new_block->next;
        new_block->next = NULL;
        prev_block = new_block;
    }
    (*num_files)++;
    printf("File created successfully.\n");
}

void show_directory(int num_files, char **file_names, int *file_sizes) {
    int i;
    printf("Directory:\n");
    for (i = 0; i < num_files; i++) {
        printf("%s (%d blocks)\n", file_names[i], file_sizes[i]);
```

```
    }
}

void deallocate_blocks(Block **allocated_blocks, Block **free_blocks) {
    Block *curr_block = *allocated_blocks;
    while (curr_block != NULL) {
        Block *next_block = curr_block->next;
        curr_block->next = *
```

******Incomplete answer*******

Slip No 27

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

Ans:-

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

int main()
{
    int total_blocks, head_pos, requests[100], n, i, j, temp, head_moves = 0;

    // Accepting user input
    printf("Enter the total number of disk blocks: ");
    scanf("%d", &total_blocks);

    printf("Enter the disk request string (end with -1):\n");
    n = 0;
    while (1) {
        scanf("%d", &requests[n]);
        if (requests[n] == -1) {
            break;
        }
        n++;
    }
```

```
printf("Enter the current head position: ");
scanf("%d", &head_pos);

printf("Enter the direction (L for Left, R for Right): ");
char dir;
scanf(" %c", &dir);

// Sorting the requests in ascending order
for (i = 0; i < n-1; i++) {
    for (j = 0; j < n-i-1; j++) {
        if (requests[j] > requests[j+1]) {
            temp = requests[j];
            requests[j] = requests[j+1];
            requests[j+1] = temp;
        }
    }
}

// Finding the index of the request closest to the head position
int index;
for (i = 0; i < n; i++) {
    if (head_pos <= requests[i]) {
        index = i;
        break;
    }
}

// Traversing right and then left
if (dir == 'R') {
    for (i = index; i < n; i++) {
        printf("%d ", requests[i]);
        head_moves += abs(requests[i] - head_pos);
        head_pos = requests[i];
    }
    for (i = index-1; i >= 0; i--) {
        printf("%d ", requests[i]);
        head_moves += abs(requests[i] - head_pos);
        head_pos = requests[i];
    }
}

// Traversing left and then right
else {
    for (i = index-1; i >= 0; i--) {
```

```
            printf("%d ", requests[i]);
            head_moves += abs(requests[i] - head_pos);
            head_pos = requests[i];
        }
        for (i = index; i < n; i++) {
            printf("%d ", requests[i]);
            head_moves += abs(requests[i] - head_pos);
            head_pos = requests[i];
        }
    }

    // Displaying the total number of head movements
    printf("\nTotal number of head movements: %d\n", head_moves);

    return 0;
}
```

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)

Ans:-

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

#define ARRAY_SIZE 1000

int main(int argc, char **argv)
{
    int rank, size;
    int array[ARRAY_SIZE], min, local_min;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Generate random numbers
    srand(rank);
    for (int i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 100;
    }

    // Find local minimum
    local_min = array[0];
    for (int i = 1; i < ARRAY_SIZE; i++) {
```

```
        if (array[i] < local_min) {
            local_min = array[i];
        }
    }

    // Reduce to find global minimum
    MPI_Reduce(&local_min, &min, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);

    // Print results
    if (rank == 0) {
        printf("The minimum number is %d.\n", min);
    }

    MPI_Finalize();
    return 0;
}
```

Slip No 28

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

Ans:-

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

int main() {
    int queue[100], n, head, i, j, k, seek_time = 0, diff;
    float avg_seek_time;

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

    printf("Enter the disk request string: ");
```

```
for(i = 0; i < n; i++)
    scanf("%d", &queue[i]);

printf("Enter the current head position: ");
scanf("%d", &head);

printf("Enter the direction (0 for left, 1 for right): ");
scanf("%d", &k);

// Sorting the request queue
for(i = 0; i < n-1; i++) {
    for(j = i+1; j < n; j++) {
        if(queue[i] > queue[j]) {
            int temp = queue[i];
            queue[i] = queue[j];
            queue[j] = temp;
        }
    }
}

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

if(k == 0) {  // Head moving towards left direction
    for(i = index-1; i >= 0; i--) {
        seek_time += abs(head - queue[i]);
        head = queue[i];
    }
    seek_time += head;
    head = 0;
    for(i = n-1; i >= index; i--) {
        seek_time += abs(head - queue[i]);
        head = queue[i];
    }
}
else {  // Head moving towards right direction
    for(i = index; i < n; i++) {
        seek_time += abs(head - queue[i]);
        head = queue[i];
```

```
        }
        seek_time += abs(head - (n-1));
        head = n-1;
        for(i = index-1; i >= 0; i--) {
            seek_time += abs(head - queue[i]);
            head = queue[i];
        }
    }

    avg_seek_time = (float)seek_time / n;

    printf("\nOrder of disk requests served:\n");
    for(i = 0; i < n; i++)
        printf("%d ", queue[i]);

    printf("\n\nTotal number of head movements: %d", seek_time);
    printf("\nAverage seek time: %f", avg_seek_time);

    return 0;
}
```

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

Ans:-

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

#define ARRAY_SIZE 1000

int main(int argc, char *argv[]) {
    int rank, size, i;
    int array[ARRAY_SIZE];
    int 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) {
        // Generate random numbers in array
        srand(12345);
        for (i = 0; i < ARRAY_SIZE; i++) {
```

```
            array[i] = rand() % 100;
        }
    }

    // Distribute the array to all processes
    MPI_Scatter(array, ARRAY_SIZE/size, MPI_INT, array, ARRAY_SIZE/size, MPI_INT, 0,
MPI_COMM_WORLD);

    // Calculate local sum
    for (i = 0; i < ARRAY_SIZE/size; i++) {
        local_sum += array[i];
    }

    // Calculate global sum using MPI_Reduce
    MPI_Reduce(&local_sum, &global_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    if (rank == 0) {
        printf("The sum of the %d numbers is %d\n", ARRAY_SIZE, global_sum);
    }

    MPI_Finalize();
    return 0;
}
```

Slip No 29

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

Ans:-

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

#define ARRAY_SIZE 1000

int main(int argc, char *argv[]) {
    int rank, size, i;
    int array[ARRAY_SIZE];
    int 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) {
        // Generate random numbers in array
        srand(12345);
        for (i = 0; i < ARRAY_SIZE; i++) {
            array[i] = rand() % 100;
        }
    }

    // Distribute the array to all processes
    MPI_Scatter(array, ARRAY_SIZE/size, MPI_INT, array, ARRAY_SIZE/size, MPI_INT, 0,
MPI_COMM_WORLD);

    // Calculate local sum
    for (i = 0; i < ARRAY_SIZE/size; i++) {
        local_sum += array[i];
    }

    // Calculate global sum using MPI_Reduce
    MPI_Reduce(&local_sum, &global_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    if (rank == 0) {
        printf("The sum of the %d numbers is %d\n", ARRAY_SIZE, global_sum);
    }

    MPI_Finalize();
    return 0;
}
```

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..
80, 150, 60,135, 40, 35, 170
Starting Head Position: 70
Direction: Right
Ans:-

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

int compare(const void *a, const void *b) {
    return (*(int*)a - *(int*)b);
}
```

```
int main() {
    int n, head, i, j, moves = 0;
    printf("Enter the total number of disk blocks: ");
    scanf("%d", &n);
    int requests[n];
    printf("Enter the disk request string:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }
    printf("Enter the starting head position: ");
    scanf("%d", &head);
    qsort(requests, n, sizeof(int), compare);
    int index;
    for (i = 0; i < n; i++) {
        if (requests[i] >= head) {
            index = i;
            break;
        }
    }
    printf("Enter the direction (0 for left, 1 for right): ");
    int direction;
    scanf("%d", &direction);
    if (direction == 0) {
        for (i = index - 1; i >= 0; i--) {
            printf("%d ", requests[i]);
            moves += abs(head - requests[i]);
            head = requests[i];
        }
        moves += head;
        for (i = n - 1; i >= index; i--) {
            printf("%d ", requests[i]);
            moves += abs(head - requests[i]);
            head = requests[i];
        }
    }
    else if (direction == 1) {
        for (i = index; i < n; i++) {
            printf("%d ", requests[i]);
            moves += abs(head - requests[i]);
            head = requests[i];
        }
        moves += abs(requests[n - 1] - requests[index - 1]);
        head = requests[n - 1];
        for (i = index - 1; i >= 0; i--) {
```

```
            printf("%d ", requests[i]);
            moves += abs(head - requests[i]);
            head = requests[i];
        }
    }
    printf("\nTotal number of head movements: %d\n", moves);
    return 0;
}
```

Slip No 30

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)

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

#define ARRAY_SIZE 1000

int main(int argc, char** argv) {
    int rank, size, i;
    int* array = (int*)malloc(ARRAY_SIZE * sizeof(int));
    int min = 0, global_min;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    srand(time(NULL) + rank); // seed the random number generator with rank-dependent seed

    for (i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand();
    }

    MPI_Reduce(&array, &min, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);

    if (rank == 0) {
        printf("The minimum number is %d\n", min);
    }

    MPI_Finalize();
```

```
    free(array);
    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.
65, 95, 30, 91, 18, 116, 142, 44, 168
Start Head Position: 52

Ans:-

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

#define MAX_REQUESTS 100

int main()
{
    int requests[MAX_REQUESTS];
    int num_requests;
    int current_head_position;
    int total_head_movements = 0;

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

    printf("Enter the disk request string: ");
    for(int i = 0; i < num_requests; i++)
    {
        scanf("%d", &requests[i]);
    }

    printf("Enter the current head position: ");
    scanf("%d", &current_head_position);

    printf("Order of request served:\n");
    for(int i = 0; i < num_requests; i++)
    {
        printf("%d ", requests[i]);
        total_head_movements += abs(current_head_position - requests[i]);
        current_head_position = requests[i];
```

```
    }

    printf("\nTotal number of head movements: %d\n", total_head_movements);

    return 0;
}
```