

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 ☐

delete

Exit

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 200
```

```
typedef struct dir {
    char fname[20];
    int start;
    struct dir *next;
} NODE;
```

```
NODE *first, *last;
int n, fb, bit[MAX];
```

```
void init() {
    int i;
    printf("Enter total no. of disk blocks:");
    scanf("%d", &n);
    fb = n;

    for (i = 0; i < 10; i++) {
        int k = rand() % n;
        if (bit[k] != -2) {
            bit[k] = -2;
            fb--;
        }
    }
}
```

```
void show_bitvector() {
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", bit[i]);
    printf("\n");
}
```

```
void show_dir() {
    NODE *p;
    int i;
    printf("File\tChain\n");
    p = first;
    while (p != NULL) {
        printf("%s\t", p->fname);
        i = p->start;
```

```
        while (i != -1) {
            printf("%d->", i);
            i = bit[i];
        }
        printf("NULL\n");
        p = p->next;
    }
}
```

```
void create() {
    NODE *p;
    char fname[20];
    int i, j, nob;

    printf("Enter file name:");
    scanf("%s", fname);
    printf("Enter no. of blocks:");
    scanf("%d", &nob);

    if (nob > fb) {
        printf("Failed to create file %s\n", fname);
        return;
    }
```

```
    for (i = 0; i < n; i++) {
        if (bit[i] == 0)
            break;
    }
```

```
    p = (NODE*)malloc(sizeof(NODE));
    strcpy(p->fname, fname);
    p->start = i;
    p->next = NULL;
```

```
    if (first == NULL)
        first = p;
    else
        last->next = p;
```

```
    last = p;
    fb -= nob;
    j = i + 1;
    nob--;
```

```
    while (nob > 0) {
        if (bit[j] == 0) {
            bit[i] = j;
            i = j;
            nob--;
        }
        j++;
    }
    bit[i] = -1;
    printf("File %s created successfully.\n",
    fname);
}
```

```

void delete() {
    char fname[20];
    NODE *p, *q;
    int nob = 0, i, j;

    printf("Enter file name to be deleted:");
    scanf("%s", fname);

    p = q = first;
    while (p != NULL) {
        if (strcmp(p->fname, fname) == 0)
            break;
        q = p;
        p = p->next;
    }

    if (p == NULL) {
        printf("File %s not found.\n", fname);
        return;
    }

    i = p->start;
    while (i != -1) {
        nob++;
        j = i;
        i = bit[j];
        bit[j] = 0;
    }

    fb += nob;
    if (p == first)
        first = first->next;
    else if (p == last) {
        last = q;
        last->next = NULL;
    } else {
        q->next = p->next;
    }

    free(p);
    printf("File %s deleted successfully.\n",
fname);
}

int main() {
    int ch;
    init();
    while (1) {
        printf("1. Show bit vector\n");
        printf("2. Create new file\n");
        printf("3. Show directory\n");
        printf("4. Delete file\n");
        printf("5. Exit\n");
        printf("Enter your choice (1-5):");
        scanf("%d", &ch);

        switch (ch) {
            case 1:
                show_bitvector();
                break;
            case 2:
                create();
                break;
            case 3:
                show_dir();
                break;
            case 4:
                delete();
                break;
            case 5:
                exit(0);
        }
    }
    return 0;
}

```

Q. Write an OS program to implement C-SCAN algorithm Disk Scheduling algorithm.

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

int main() {
    int RQ[100], i, j, n, TotalHeadMoment = 0,
    initial, size, move;

    printf("Enter the number of Requests\n");
    scanf("%d", &n);

    printf("Enter the Requests sequence\n");
    for (i = 0; i < n; i++)
        scanf("%d", &RQ[i]);

    printf("Enter initial head position\n");
    scanf("%d", &initial);

    printf("Enter total disk size\n");
    scanf("%d", &size);

    printf("Enter the head movement direction
    for high 1 and for low 0\n");
    scanf("%d", &move);

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

    int index;
    for (i = 0; i < n; i++) {
        if (initial < RQ[i]) {
            index = i;
            break;
        }
    }

    // If movement is towards high value
    if (move == 1) {
        for (i = index; i < n; i++) {
            TotalHeadMoment += abs(RQ[i] - initial);
            initial = RQ[i];
        }

        // Last movement for max size
        TotalHeadMoment += abs(size - RQ[i - 1] -
1);
```

```
TotalHeadMoment += abs(size - 1 - 0);
initial = 0;

    for (i = 0; i < index; i++) {
        TotalHeadMoment += abs(RQ[i] - initial);
        initial = RQ[i];
    }

    // If movement is towards low value
    else {
        for (i = index - 1; i >= 0; i--) {
            TotalHeadMoment += abs(RQ[i] - initial);
            initial = RQ[i];
        }

        // Last movement for min size
        TotalHeadMoment += abs(RQ[i + 1] - 0);

        // Movement from min to max disk
        TotalHeadMoment += abs(size - 1 - 0);
        initial = size - 1;

        for (i = n - 1; i >= index; i--) {
            TotalHeadMoment += abs(RQ[i] - initial);
            initial = RQ[i];
        }
    }

    printf("Total head movement is %d",
    TotalHeadMoment);

    return 0;
}
```

Write an OS program to implement FCFS Disk Scheduling algorithm.

```
#include <stdio.h>

int main()
{
    int i, j, sum = 0, n;
    int ar[20], tm[20];
    int disk;

    printf("Enter number of locations: ");
    scanf("%d", &n);

    printf("Enter position of head: ");
    scanf("%d", &disk);

    printf("Enter elements of disk queue:\n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &ar[i]);
        tm[i] = disk - ar[i];

        if (tm[i] < 0)
        {
            tm[i] = ar[i] - disk;
        }

        disk = ar[i];
        sum = sum + tm[i];
    }

    printf("\nMovement of total cylinders: %d\n",
sum);

    return 0;
}
```

Write a C Program to simulate Banker's algorithm

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

#define MAX_PROCESSES 10
#define MAX_RESOURCES 10

int
allocation[MAX_PROCESSES][MAX_RESOURCES];
int max[MAX_PROCESSES][MAX_RESOURCES];
int need[MAX_PROCESSES][MAX_RESOURCES];
int available[MAX_RESOURCES];
int work[MAX_RESOURCES];
bool finish[MAX_PROCESSES];

int num_processes, num_resources;

void acceptAvailable() {
    printf("Enter the available resources:\n");
    for (int i = 0; i < num_resources; i++) {
        scanf("%d", &available[i]);
    }
}

void displayAllocationMax() {
    printf("Allocation Matrix:\n");
    for (int i = 0; i < num_processes; i++) {
        for (int j = 0; j < num_resources; j++) {
            printf("%d ", allocation[i][j]);
        }
        printf("\n");
    }
    printf("\nMax Matrix:\n");
    for (int i = 0; i < num_processes; i++) {
        for (int j = 0; j < num_resources; j++) {
            printf("%d ", max[i][j]);
        }
        printf("\n");
    }
}

void displayNeedMatrix() {
    printf("Need Matrix:\n");
    for (int i = 0; i < num_processes; i++) {
        for (int j = 0; j < num_resources; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
            printf("%d ", need[i][j]);
        }
        printf("\n");
    }
}

void displayAvailable() {
    printf("Available resources:\n");
    for (int i = 0; i < num_resources; i++) {
        printf("%d ", available[i]);
    }
}
```

```

    }
    printf("\n");
}

bool isSafeState() {
    for (int i = 0; i < num_resources; i++) {
        work[i] = available[i];
    }
    for (int i = 0; i < num_processes; i++) {
        finish[i] = false;
    }
    int count = 0;
    int safeSeq[num_processes];
    while (count < num_processes) {
        bool found = false;
        for (int i = 0; i < num_processes; i++) {
            if (!finish[i]) {
                bool canAllocate = true;
                for (int j = 0; j < num_resources; j++) {
                    if (need[i][j] > work[j]) {
                        canAllocate = false;
                        break;
                    }
                }
                if (canAllocate) {
                    for (int j = 0; j < num_resources; j++) {
                        work[j] += allocation[i][j];
                    }
                    safeSeq[count++] = i;
                    finish[i] = true;
                    found = true;
                }
            }
        }
        if (!found) {
            printf("System is not in a safe state.\n");
            return false;
        }
    }
    printf("System is in a safe state. Safe sequence: ");
    for (int i = 0; i < num_processes; i++) {
        printf("%d ", safeSeq[i]);
    }
    printf("\n");
    return true;
}

bool checkRequest(int process, int request[]) {
    for (int i = 0; i < num_resources; i++) {
        if (request[i] > need[process][i] ||
            request[i] > available[i])
            return false;
    }
    return true;
}

```

```

void processRequest(int process, int request[]) {
    if (checkRequest(process, request)) {
        for (int i = 0; i < num_resources; i++) {
            available[i] -= request[i];
            allocation[process][i] += request[i];
            need[process][i] -= request[i];
        }
        if (isSafeState()) {
            printf("Request can be granted immediately.\n");
        } else {
            printf("Request cannot be granted immediately.\n");
            // Revert changes
            for (int i = 0; i < num_resources; i++) {
                available[i] += request[i];
                allocation[process][i] -= request[i];
                need[process][i] += request[i];
            }
        } else {
            printf("Request cannot be granted as it exceeds maximum need or available resources.\n");
        }
    }
}

int main() {
    printf("Enter the number of processes: ");
    scanf("%d", &num_processes);
    printf("Enter the number of resources: ");
    scanf("%d", &num_resources);

    printf("Enter the Allocation Matrix:\n");
    for (int i = 0; i < num_processes; i++) {
        for (int j = 0; j < num_resources; j++) {
            scanf("%d", &allocation[i][j]);
        }
    }

    printf("Enter the Max Matrix:\n");
    for (int i = 0; i < num_processes; i++) {
        for (int j = 0; j < num_resources; j++) {
            scanf("%d", &max[i][j]);
        }
    }

    acceptAvailable();

    int choice;
    do {
        printf("\nMenu:\n");
        printf("1. Accept Available\n");
        printf("2. Display Allocation, Max\n");
        printf("3. Display Need Matrix\n");
        printf("4. Display Available\n");
    } while (choice != 4);
}

```

```

printf("5. Check Safe State\n");
printf("6. Process Resource Request\n");
printf("0. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        acceptAvailable();
        break;
    case 2:
        displayAllocationMax();
        break;
    case 3:
        displayNeedMatrix();
        break;
    case 4:
        displayAvailable();
        break;
    case 5:
        isSafeState();
        break;
    case 6: {
        int process;
        printf("Enter the process number (0-
indexed): ");
        scanf("%d", &process);
        int request[num_resources];
        printf("Enter the resource request for
process %d: ", process);
        for (int i = 0; i < num_resources; i++) {
            scanf("%d", &request[i]);
        }
        processRequest(process, request);
        break;
    }
    case 0:
        printf("Exiting...\n");
        break;
    default:
        printf("Invalid choice. Please try
again.\n");
}
} while (choice != 0);

return 0;
}

```

SSTF algorithm.

```

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

int main()
{
    int RQ[100], i, n, TotalHeadMoment = 0, initial,
    count = 0;

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

    printf("Enter the Requests sequence:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &RQ[i]);

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

    printf("%d", initial);

    // Logic for SSTF disk scheduling
    /* Loop will execute until all requests are
    processed */
    while (count != n)
    {
        int min = 1000, d, index = -1;

        for (i = 0; i < n; i++)
        {
            d = abs(RQ[i] - initial);
            if (min > d)
            {
                min = d;
                index = i;
            }
        }

        TotalHeadMoment += min;
        initial = RQ[index];
        printf(" --> %d", RQ[index]);

        // Mark the processed request as visited by
        setting it to a high value
        RQ[index] = 1000;
        count++;
    }

    printf("\nTotal head movement is %d\n",
    TotalHeadMoment);

    return 0;
}

```

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>
```

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

```
int allocation[20][20], max[20][20], available[20],
need[20][20], safe[10], s = 0;
```

```
int finish[10], work[10], cnt = 0, flag = 0, temp = 0;
```

```
int p, r, i, j, ch, proc_index, req[10];
```

```
void display_matrices() {
    printf("\nAllocation Table:\n");
    for (i = 0; i < p; i++) {
        for (j = 0; j < r; j++) {
            printf("%d\t", allocation[i][j]);
        }
        printf("\n");
    }
}
```

```
printf("\nMax Table:\n");
for (i = 0; i < p; i++) {
    for (j = 0; j < r; j++) {
        printf("%d\t", max[i][j]);
    }
    printf("\n");
}
```

```
printf("\nNeed Table:\n");
for (i = 0; i < p; i++) {
    for (j = 0; j < r; j++) {
        printf("%d\t", need[i][j]);
    }
    printf("\n");
}
```

```
printf("\nAvailable Resources:\n");
for (i = 0; i < r; i++) {
    printf("%d\t", available[i]);
}
printf("\n");
}
```

```
void check() {
    temp = 0;
    s = 0;

    // Calculate need = max - allocation
    for (i = 0; i < p; i++) {
        for (j = 0; j < r; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
}
```

```
display_matrices();
```

```
for (i = 0; i < p; i++) {
    finish[i] = 0;
}
```

```
for (i = 0; i < r; i++) {
    work[i] = available[i];
}
```

```
while (temp < 2) {
    for (i = 0; i < p; i++) {
        flag = 1;
        for (j = 0; j < r; j++) {
            if (finish[i] == 0 && need[i][j] > work[j])
            {
                flag = 0;
                break;
            }
        }
        if (flag == 1) {
            for (j = 0; j < r; j++) {
                work[j] += allocation[i][j];
            }
            finish[i] = 1;
            safe[s++] = i;
        }
        temp++;
    }
}
```

```
flag = 0;
for (i = 0; i < p; i++) {
    if (finish[i] == 0) {
        flag = 1;
        break;
    }
}
```

```
if (flag == 1) {
    printf("\nSystem is in Deadlock state\n");
} else {
    printf("\nSystem is in Safe state\n");
    printf("Safe Sequence is: ");
    for (i = 0; i < p; i++) {
        printf("P%d\t", safe[i]);
    }
    printf("\n");
}
}
```

```
int main() {

    printf("\n~~~~~BANKER'S
ALGORITHM~~~~~\n");
}
```

```

printf("\nEnter the number of resources and
processes: ");
scanf("%d%d", &r, &p);

printf("\nEnter the Allocation Table:\n");
for (i = 0; i < p; i++) {
    for (j = 0; j < r; j++) {
        scanf("%d", &allocation[i][j]);
    }
}

printf("\nEnter the Max Table:\n");
for (i = 0; i < p; i++) {
    for (j = 0; j < r; j++) {
        scanf("%d", &max[i][j]);
    }
}

printf("\nEnter the Available Resources:\n");
for (i = 0; i < r; i++) {
    scanf("%d", &available[i]);
}

check();

printf("\nDo you want to add a new request?
(0/1): ");
scanf("%d", &ch);
if (ch == 0) {
    exit(1);
}

printf("\nEnter the process number: ");
scanf("%d", &proc_index);
printf("\nEnter the request: ");
for (i = 0; i < r; i++) {
    scanf("%d", &req[i]);
}

flag = 1;
for (i = 0; i < r; i++) {
    if (req[i] > need[proc_index][i]) {
        flag = 0;
        break;
    }
}

if (flag == 0) {
    printf("\nRequest cannot be satisfied...\n");
    exit(1);
}

flag = 1;
for (i = 0; i < r; i++) {
    if (req[i] > available[i]) {
        flag = 0;
        break;
    }
}

```


Sequential(Contiguous) allocation method

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

#define MAX 200

typedef struct dir {
    char fname[20];
    int start;
    struct dir *next;
} NODE;

NODE *first, *last;
int n, fb, bit[MAX];

void init() {
    int i;
    printf("Enter total no.of disk blocks: ");
    scanf("%d", &n);
    fb = n;
    for (i = 0; i < 10; i++) {
        int k = rand() % n;
        if (bit[k] != -2) {
            bit[k] = -2;
        }
    }
}

void show_bitvector() {
    int i;
    for (i = 0; i < n; i++) {
        printf("%d ", bit[i]);
    }
    printf("\n");
}

void delete() {
    char fname[20];
    NODE *p, *q;
    int nob = 0, i, j;

    printf("Enter file name to be deleted: ");
    scanf("%s", fname);

    p = q = first;
    while (p != NULL) {
        if (strcmp(p->fname, fname) == 0)
            break;
        q = p;
        p = p->next;
    }

    if (p == NULL) {
        printf("File %s not found.\n", fname);
        return;
    }
```

```
    i = p->start;
    while (i != -1) {
        nob++;
        j = i;
        i = bit[i];
        bit[j] = 0;
    }

    fb -= nob;

    if (p == first)
        first = first->next;
    else if (p == last) {
        last = q;
        last->next = NULL;
    } else
        q->next = p->next;

    free(p);
    printf("File %s deleted successfully.\n",
    fname);
}

int main() {
    int ch;
    init();
    while (1) {
        printf("1. Show bit vector\n");
        printf("2. Delete file\n");
        printf("3. Exit\n");
        printf("Enter your choice (1-3): ");
        scanf("%d", &ch);

        switch (ch) {
            case 1:
                show_bitvector();
                break;
            case 2:
                delete();
                break;
            case 3:
                exit(0);
        }
    }
    return 0;
}
```
