19)

#include <stdio.h>

#include <pthread.h>

#define NUM\_THREADS 5

pthread\_mutex\_t lock;

void\* threadFunction(void\* arg) {

pthread\_mutex\_lock(&lock);

printf("Thread %d is in the critical section.\n", \*(int\*)arg);

pthread\_mutex\_unlock(&lock);

return NULL;

}

int main() {

pthread\_t threads[NUM\_THREADS];

int threadIds[NUM\_THREADS];

pthread\_mutex\_init(&lock, NULL);

for (int i = 0; i < NUM\_THREADS; i++) {

threadIds[i] = i;

pthread\_create(&threads[i], NULL, threadFunction, &threadIds[i]);

}

for (int i = 0; i < NUM\_THREADS; i++) {

pthread\_join(threads[i], NULL);

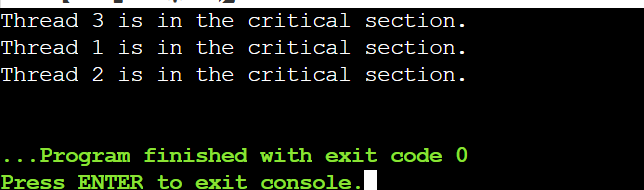
}

pthread\_mutex\_destroy(&lock);

return 0;

}

**OUTPUT:**

****

20)

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define READERS 5

#define WRITERS 2

sem\_t mutex, writeBlock;

int readCount = 0;

void\* reader(void\* arg) {

int id = \*(int\*)arg;

while (1) {

sem\_wait(&mutex);

readCount++;

if (readCount == 1) sem\_wait(&writeBlock);

sem\_post(&mutex);

printf("Reader %d is reading.\n", id);

sleep(1);

sem\_wait(&mutex);

readCount--;

if (readCount == 0) sem\_post(&writeBlock);

sem\_post(&mutex);

sleep(1);

}

return NULL;

}

void\* writer(void\* arg) {

int id = \*(int\*)arg;

while (1) {

sem\_wait(&writeBlock);

printf("Writer %d is writing.\n", id);

sleep(2);

sem\_post(&writeBlock);

sleep(1);

}

return NULL;

}

int main() {

pthread\_t r[READERS], w[WRITERS];

int ids[READERS + WRITERS];

sem\_init(&mutex, 0, 1);

sem\_init(&writeBlock, 0, 1);

for (int i = 0; i < READERS; i++) {

ids[i] = i + 1;

pthread\_create(&r[i], NULL, reader, &ids[i]);

}

for (int i = 0; i < WRITERS; i++) {

ids[READERS + i] = i + 1;

pthread\_create(&w[i], NULL, writer, &ids[READERS + i]);

}

for (int i = 0; i < READERS; i++) pthread\_join(r[i], NULL);

for (int i = 0; i < WRITERS; i++) pthread\_join(w[i], NULL);

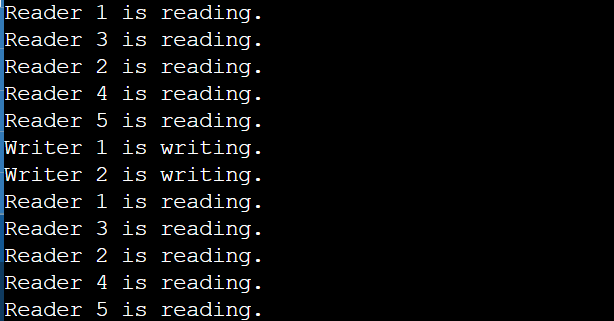
sem\_destroy(&mutex);

sem\_destroy(&writeBlock);

return 0;

}

**OUT PUT:**



21)

#include <stdio.h>

int main() {

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

int n = sizeof(blockSize) / sizeof(blockSize[0]);

int m = sizeof(processSize) / sizeof(processSize[0]);

int allocation[m];

for (int i = 0; i < m; i++) {

int worstIdx = -1;

for (int j = 0; j < n; j++) {

if (blockSize[j] >= processSize[i]) {

if (worstIdx == -1 || blockSize[j] > blockSize[worstIdx])

worstIdx = j;

}

}

if (worstIdx != -1) {

allocation[i] = worstIdx;

blockSize[worstIdx] -= processSize[i];

} else {

allocation[i] = -1;

}

}

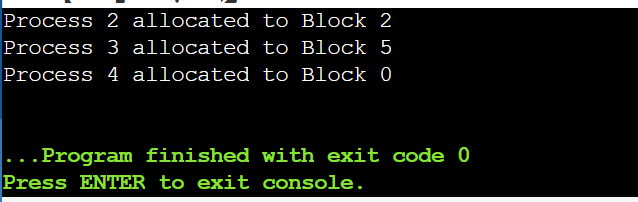
for (int i = 0; i < m; i++)

printf("Process %d allocated to Block %d\n", i + 1, allocation[i] + 1);

return 0;

}

**OUT PUT:**

****

22)

#include <stdio.h>

int main() {

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

int n = sizeof(processSize) / sizeof(processSize[0]);

int m = sizeof(blockSize) / sizeof(blockSize[0]);

int allocation[n];

for (int i = 0; i < n; i++) {

int bestIdx = -1;

for (int j = 0; j < m; j++) {

if (blockSize[j] >= processSize[i]) {

if (bestIdx == -1 || blockSize[bestIdx] > blockSize[j])

bestIdx = j;

}

}

if (bestIdx != -1) {

allocation[i] = bestIdx;

blockSize[bestIdx] -= processSize[i];

} else {

allocation[i] = -1;

}

}

printf("Process No.\tProcess Size\tBlock No.\n");

for (int i = 0; i < n; i++) {

printf("%d\t\t%d\t\t", i + 1, processSize[i]);

if (allocation[i] != -1)

printf("%d\n", allocation[i] + 1);

else

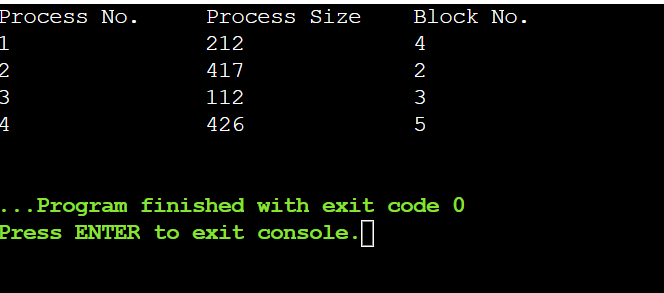
printf("Not Allocated\n");

}

return 0;

}

**OUTP UT:**

****

23)

#include <stdio.h>

int main() {

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

int n = sizeof(processSize) / sizeof(processSize[0]);

int m = sizeof(blockSize) / sizeof(blockSize[0]);

int allocation[n];

for (int i = 0; i < n; i++) {

allocation[i] = -1;

for (int j = 0; j < m; j++) {

if (blockSize[j] >= processSize[i]) {

allocation[i] = j;

blockSize[j] -= processSize[i];

break;

}

}

}

printf("Process No.\tBlock No.\n");

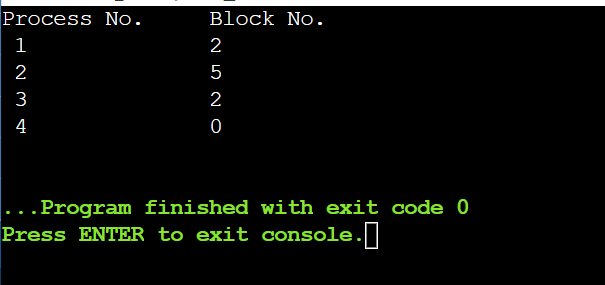
for (int i = 0; i < n; i++)

printf(" %d\t\t%d\n", i + 1, allocation[i] + 1);

return 0;

}

**OUT PUT:**

****

24)

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

int main() {

int fd;

char \*text = "Hello, UNIX System Calls!\n";

char buffer[50];

fd = open("example.txt", O\_CREAT | O\_WRONLY, 0644);

write(fd, text, 30);

close(fd);

fd = open("example.txt", O\_RDONLY);

read(fd, buffer, 30);

printf("%s", buffer);

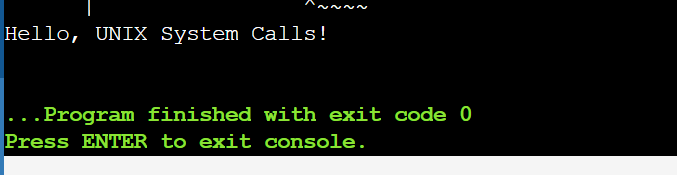
close(fd);

unlink("example.txt");

return 0;

}

**OUT PUT:**

****

25)

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

#include <dirent.h>

int main() {

int fd = open("example.txt", O\_RDWR | O\_CREAT, 0644);

fcntl(fd, F\_SETFL, O\_NONBLOCK);

lseek(fd, 0, SEEK\_SET);

struct stat fileStat;

stat("example.txt", &fileStat);

printf("File size: %ld bytes\n", fileStat.st\_size);

DIR \*dir = opendir(".");

struct dirent \*entry;

while ((entry = readdir(dir)) != NULL) {

printf("Found: %s\n", entry->d\_name);

}

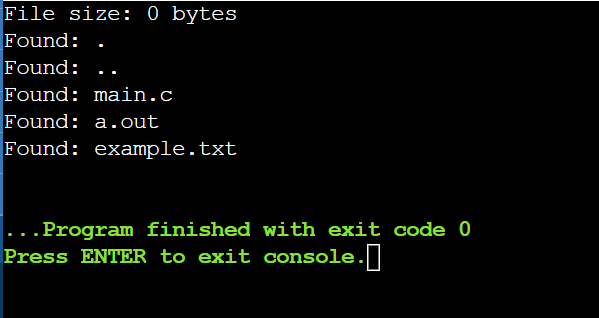
closedir(dir);

close(fd);

return 0;

}

**OUT PUT:**

****

**26)**

#include <stdio.h>

#include <stdlib.h>

void createFile(const char \*filename) {

FILE \*file = fopen(filename, "w");

if (file) fclose(file);

}

void writeFile(const char \*filename, const char \*content) {

FILE \*file = fopen(filename, "a");

if (file) {

fputs(content, file);

fclose(file);

}

}

void readFile(const char \*filename) {

char buffer[100];

FILE \*file = fopen(filename, "r");

if (file) {

while (fgets(buffer, sizeof(buffer), file)) {

printf("%s", buffer);

}

fclose(file);

}

}

void deleteFile(const char \*filename) {

remove(filename);

}

int main() {

const char \*filename = "example.txt";

createFile(filename);

writeFile(filename, "Hello, World!\n");

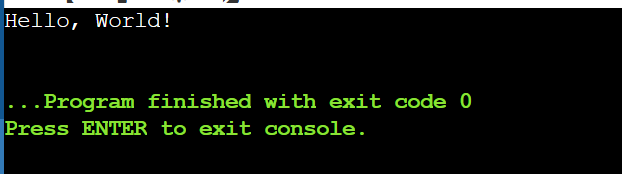
readFile(filename);

deleteFile(filename);

return 0;

}

**OUT PUT:**

****

**27)**

#include <stdio.h>

#include <dirent.h>

int main() {

struct dirent \*entry;

DIR \*dp = opendir(".");

if (dp == NULL) {

perror("opendir");

return 1;

}

while ((entry = readdir(dp)) != NULL) {

printf("%s\n", entry->d\_name);

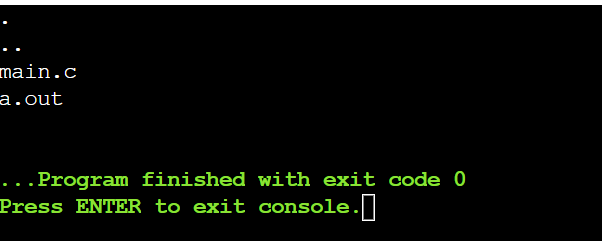
}

closedir(dp);

return 0;

}

OUT PUT:

****

28)

#include <stdio.h>

#include <string.h>

int main(int argc, char \*argv[]) {

if (argc != 3) {

printf("Usage: %s <pattern> <file>\n", argv[0]);

return 1;

}

FILE \*file = fopen(argv[2], "r");

if (!file) {

perror("File opening failed");

return 1;

}

char line[256];

while (fgets(line, sizeof(line), file)) {

if (strstr(line, argv[1])) {

printf("%s", line);

}

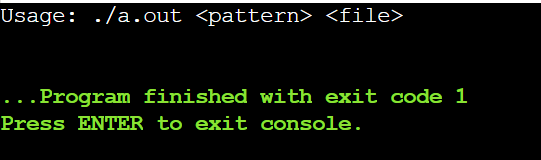
}

fclose(file);

return 0;

}

OUT PUT:



29)

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define NUM\_THREADS 5

sem\_t semaphore;

void\* threadFunction(void\* arg) {

int id = \*((int\*)arg);

sem\_wait(&semaphore);

printf("Thread %d is in the critical section.\n", id);

sleep(1);

printf("Thread %d is leaving the critical section.\n", id);

sem\_post(&semaphore);

return NULL;

}

int main() {

pthread\_t threads[NUM\_THREADS];

int thread\_ids[NUM\_THREADS];

sem\_init(&semaphore, 0, 1);

for (int i = 0; i < NUM\_THREADS; i++) {

thread\_ids[i] = i + 1;

pthread\_create(&threads[i], NULL, threadFunction, &thread\_ids[i]);

}

for (int i = 0; i < NUM\_THREADS; i++) {

pthread\_join(threads[i], NULL);

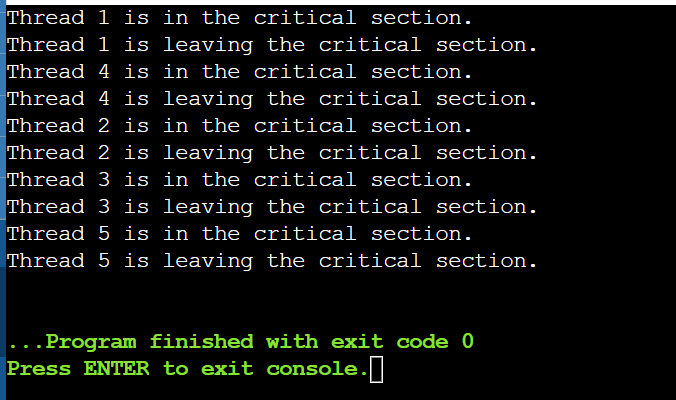
}

sem\_destroy(&semaphore);

return 0;

}

**OUT PUT:**

****

**30)**

1.

#include <stdio.h>

#include <pthread.h>

void\* printMessage(void\* msg) {

printf("%s\n", (char\*)msg);

return NULL;

}

int main() {

pthread\_t thread;

char\* message = "Hello from the thread!";

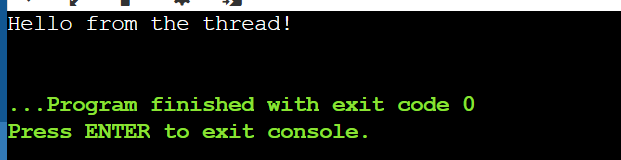
pthread\_create(&thread, NULL, printMessage, (void\*)message);

pthread\_join(thread, NULL);

return 0;

}

OUT PUT:



2)

#include <stdio.h>

#include <pthread.h>

int counter = 0;

pthread\_mutex\_t lock;

void\* incrementCounter(void\* arg) {

pthread\_mutex\_lock(&lock);

counter++;

pthread\_mutex\_unlock(&lock);

return NULL;

}

int main() {

pthread\_t threads[5];

pthread\_mutex\_init(&lock, NULL);

for (int i = 0; i < 5; i++) pthread\_create(&threads[i], NULL, incrementCounter, NULL);

for (int i = 0; i < 5; i++) pthread\_join(threads[i], NULL);

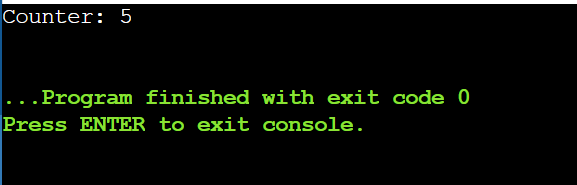
printf("Counter: %d\n", counter);

pthread\_mutex\_destroy(&lock);

return 0;

}

OUT PUT:



3)

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

void\* run(void\* arg) {

while (1) {

printf("Thread running...\n");

sleep(1);

}

return NULL;

}

int main() {

pthread\_t thread;

pthread\_create(&thread, NULL, run, NULL);

sleep(3);

pthread\_cancel(thread);

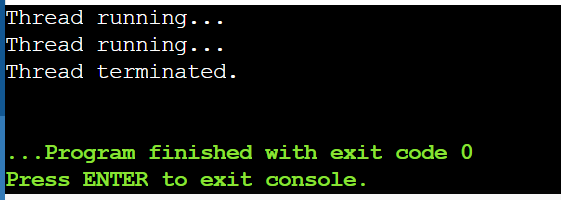
pthread\_join(thread, NULL);

printf("Thread terminated.\n");

return 0;

}

OUT PUT:



31)

#include <stdio.h>

#define FRAME\_SIZE 3

#define PAGE\_COUNT 5

int main() {

int pages[PAGE\_COUNT] = {0, 1, 2, 3, 4};

int frames[FRAME\_SIZE] = {-1, -1, -1};

int pageFaults = 0, i, j, k;

for (i = 0; i < PAGE\_COUNT; i++) {

int found = 0;

for (j = 0; j < FRAME\_SIZE; j++) {

if (frames[j] == pages[i]) {

found = 1; break;

}

}

if (!found) {

frames[pageFaults % FRAME\_SIZE] = pages[i];

pageFaults++;

}

printf("Page: %d | Frames: ", pages[i]);

for (k = 0; k < FRAME\_SIZE; k++) printf("%d ", frames[k]);

printf("\n");

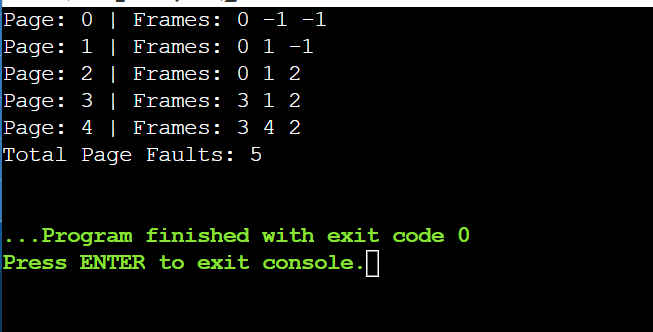
}

printf("Total Page Faults: %d\n", pageFaults);

return 0;

}

OUT PUT:



32)

#include <stdio.h>

int main() {

int pages[10], frame[3], n, m, i, j, k, pos, flag, pageFaults = 0;

printf("Enter number of pages: ");

scanf("%d", &n);

printf("Enter pages: ");

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

for (i = 0; i < 3; i++) frame[i] = -1;

for (i = 0; i < n; i++) {

flag = 0;

for (j = 0; j < 3; j++) if (frame[j] == pages[i]) flag = 1;

if (!flag) {

pageFaults++;

pos = -1;

for (j = 0; j < 3; j++) {

if (frame[j] == -1) { pos = j; break; }

for (k = i - 1; k >= 0; k--) if (frame[j] == pages[k]) { pos = j; break; }

}

if (pos == -1) pos = 0;

frame[pos] = pages[i];

}

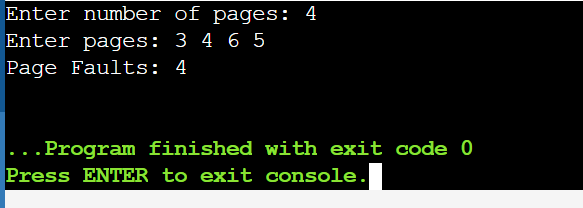
}

printf("Page Faults: %d\n", pageFaults);

return 0;

}

OUT PUT:



33)

#include <stdio.h>

int main() {

int pages[] = {0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

int frames[3] = {-1, -1, -1}, pageFaults = 0, n = 12, f = 3;

for (int i = 0; i < n; i++) {

int j, found = 0;

for (j = 0; j < f; j++) if (frames[j] == pages[i]) found = 1;

if (!found) {

int replace = -1, farthest = -1;

for (j = 0; j < f; j++) {

int k;

for (k = i + 1; k < n; k++) {

if (frames[j] == pages[k]) break;

}

if (k > farthest) { farthest = k; replace = j; }

}

frames[replace] = pages[i];

pageFaults++;

}

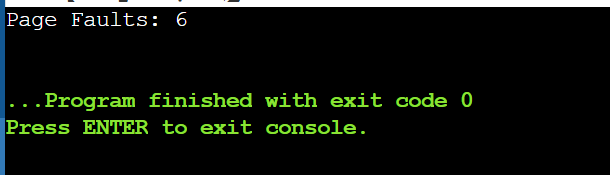
}

printf("Page Faults: %d\n", pageFaults);

return 0;

}

OUT PUT:



34)

#include <stdio.h>

#define MAX\_RECORDS 100

void readRecords(char records[MAX\_RECORDS][50], int count) {

for (int i = 0; i < count; i++) {

printf("Record %d: %s\n", i + 1, records[i]);

}

}

int main() {

char records[MAX\_RECORDS][50] = {

"Record 1: Data A",

"Record 2: Data B",

"Record 3: Data C"

};

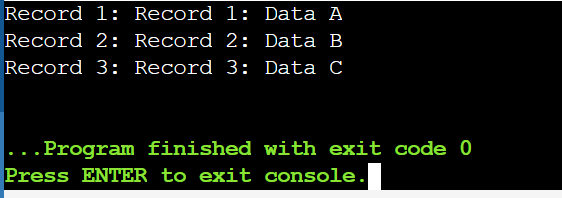
int count = 3;

readRecords(records, count);

return 0;

}

OUT PUT:



35)

#include <stdio.h>

#include <stdlib.h>

#define BLOCKS 5

typedef struct {

int block[BLOCKS];

} IndexBlock;

void allocateFile(IndexBlock \*index) {

for (int i = 0; i < BLOCKS; i++) {

index->block[i] = i + 1;

}

}

void displayAllocation(IndexBlock \*index) {

printf("File Allocation:\n");

for (int i = 0; i < BLOCKS; i++) {

printf("Index %d -> Block %d\n", i, index->block[i]);

}

}

int main() {

IndexBlock index;

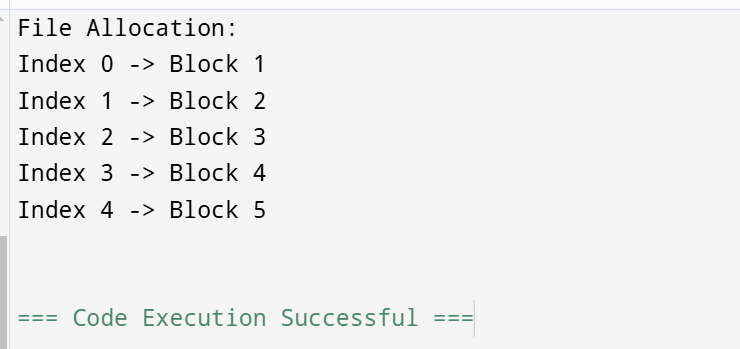
allocateFile(&index);

displayAllocation(&index);

return 0;

}

**OUT PUT:**

****

36)

#include <stdio.h>

#include <stdlib.h>

typedef struct Block {

int data;

struct Block\* next;

} Block;

typedef struct File {

Block\* head;

Block\* tail;

} File;

File\* createFile() {

File\* f = malloc(sizeof(File));

f->head = f->tail = NULL;

return f;

}

void addBlock(File\* f, int data) {

Block\* b = malloc(sizeof(Block));

b->data = data;

b->next = NULL;

if (!f->head) f->head = f->tail = b;

else f->tail->next = b, f->tail = b;

}

void displayFile(File\* f) {

for (Block\* b = f->head; b; b = b->next)

printf("%d -> ", b->data);

printf("NULL\n");

}

int main() {

File\* myFile = createFile();

addBlock(myFile, 1);

addBlock(myFile, 2);

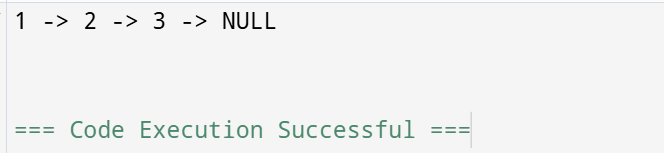
addBlock(myFile, 3);

displayFile(myFile);

return 0;

}

**OUT PUT:**

****

37)

#include <stdio.h>

#include <stdlib.h>

void FCFS(int arr[], int n, int head) {

int total\_head\_movement = 0;

for (int i = 0; i < n; i++) {

total\_head\_movement += abs(arr[i] - head);

head = arr[i];

}

printf("Total head movement: %d\n", total\_head\_movement);

}

int main() {

int requests[] = {176, 79, 34, 60, 92, 11, 41, 114};

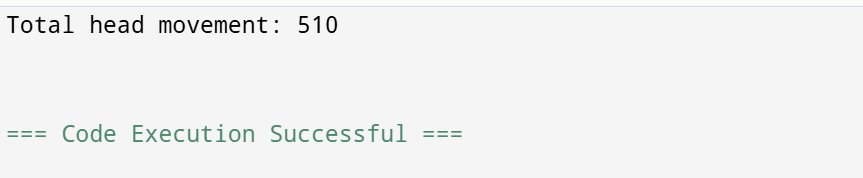
int n = sizeof(requests) / sizeof(requests[0]);

int initial\_head = 50;

FCFS(requests, n, initial\_head);

return 0;

}



38)

#include <stdio.h>

#include <stdlib.h>

int main() {

int requests[] = {176, 79, 34, 60, 92, 11, 41, 114};

int head = 50, total\_head\_movement = 0, size = sizeof(requests) / sizeof(requests[0]);

int i, j, temp, seek\_sequence[100], index = 0;

for (i = 0; i < size - 1; i++)

for (j = 0; j < size - i - 1; j++)

if (requests[j] > requests[j + 1]) {

temp = requests[j];

requests[j] = requests[j + 1];

requests[j + 1] = temp;

}

for (i = 0; i < size; i++)

if (requests[i] >= head) break;

for (j = i; j < size; j++) {

total\_head\_movement += abs(requests[j] - head);

head = requests[j];

seek\_sequence[index++] = head;

}

total\_head\_movement += abs(head - 199);

head = 199;

for (j = i - 1; j >= 0; j--) {

total\_head\_movement += abs(requests[j] - head);

head = requests[j];

seek\_sequence[index++] = head;

}

printf("Total head movement: %d\n", total\_head\_movement);

return 0;

}

**OUT PUT:**



39)

#include <stdio.h>

#include <stdlib.h>

int main() {

int requests[] = {176, 79, 34, 60, 92, 11, 41, 114};

int head = 50, total\_head\_movement = 0, size = sizeof(requests) / sizeof(requests[0]);

int i, j, temp, seek\_sequence[100], index = 0;

for (i = 0; i < size - 1; i++)

for (j = 0; j < size - i - 1; j++)

if (requests[j] > requests[j + 1]) {

temp = requests[j];

requests[j] = requests[j + 1];

requests[j + 1] = temp;

}

for (i = 0; i < size; i++)

if (requests[i] >= head) break;

for (j = i; j < size; j++) {

total\_head\_movement += abs(requests[j] - head);

head = requests[j];

seek\_sequence[index++] = head;

}

total\_head\_movement += abs(head - 199);

head = 0;

total\_head\_movement += abs(199 - head);

for (j = 0; j < i; j++) {

total\_head\_movement += abs(requests[j] - head);

head = requests[j];

seek\_sequence[index++] = head;

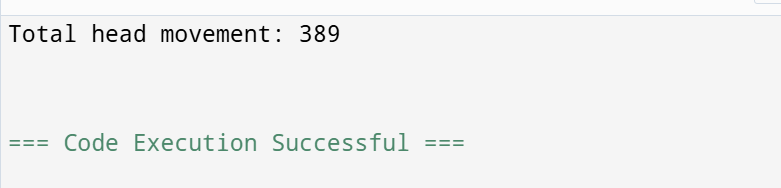
}

printf("Total head movement: %d\n", total\_head\_movement);

return 0;

}

**OUT PUT:**

****

40)

#include <stdio.h>

int main() {

char \*users[] = {"Owner", "Group", "Others"};

char \*permissions[] = {"r", "w", "x"};

for (int i = 0; i < 3; i++) {

printf("%s: %s%s%s\n", users[i],

(i == 0 ? permissions[0] : "-"),

(i == 1 ? permissions[1] : "-"),

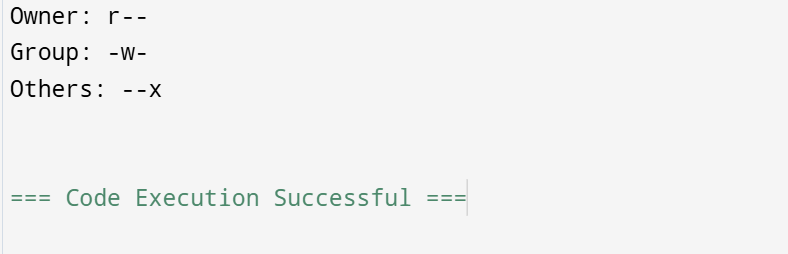
(i == 2 ? permissions[2] : "-"));

}

return 0;

}

**OUT PUT:**

****