**1**.**Parent id printing:**

#include <stdio.h>

#include<unistd.h>

int main() {

printf("Process ID:%d\n",getpid());

printf("Parent Process ID:%d\n",getppid());

return 0;

}

**2.File content copying:**

#include <stdio.h>

int main() {

FILE \*s = fopen("src.txt", "r"), \*d = fopen("dst.txt", "w");

char c;

while ((c = fgetc(s)) != EOF) fputc(c, d);

fclose(s), fclose(d);

}

**3.FCFS cpu Scheduling:**

#include <stdio.h>

int main() {

int n = 3, bt[] = {5, 8, 12}, wt[3] = {0}, tat[3], i;

for (i = 1; i < n; i++) wt[i] = wt[i - 1] + bt[i - 1];

for (i = 0; i < n; i++) tat[i] = wt[i] + bt[i];

printf("P\tBT\tWT\tTAT\n");

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

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

}

**4.SJF cpu Scheduling:**

#include <stdio.h>

int main() {

int bt[] = {6, 2, 8, 3}, wt[] = {0, 0, 0, 0}, tat[4], n = 4, i, j, temp;

for (i = 0; i < n; i++) // Sort burst times

for (j = i + 1; j < n; j++)

if (bt[i] > bt[j]) {

temp = bt[i];

bt[i] = bt[j];

bt[j] = temp;

}

for (i = 1; i < n; i++) wt[i] = wt[i - 1] + bt[i - 1];

for (i = 0; i < n; i++) tat[i] = wt[i] + bt[i];

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

printf("BT: %d WT: %d TAT: %d\n", bt[i], wt[i], tat[i]);

}

**5.Priority Scheduling:**

#include <stdio.h>

int main() {

int pr[] = {3, 1, 4, 2}, bt[] = {5, 9, 6, 3}, n = 4, wt[4] = {0}, tat[4], i, j, temp;

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

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

if (pr[i] > pr[j]) {

temp = pr[i]; pr[i] = pr[j]; pr[j] = temp;

temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;

}

}

}

for (i = 1; i < n; i++) wt[i] = wt[i - 1] + bt[i - 1];

for (i = 0; i < n; i++) tat[i] = wt[i] + bt[i];

for (i = 0; i < n; i++) printf("P%d Pr:%d BT:%d WT:%d TAT:%d\n", i + 1, pr[i], bt[i], wt[i], tat[i]);

}

**6.Preemptive Priority Scheduling:**

#include <stdio.h>

int main() {

int bt[] = {6, 2, 8, 3}, pr[] = {1, 4, 2, 3}, wt[4] = {0}, tat[4], n = 4, i, j, temp;

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

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

if (pr[i] > pr[j]) {

temp = pr[i]; pr[i] = pr[j]; pr[j] = temp;

temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;

}

}

}

for (i = 1; i < n; i++) wt[i] = wt[i - 1] + bt[i - 1];

for (i = 0; i < n; i++) tat[i] = wt[i] + bt[i];

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

printf("P%d Pr:%d BT:%d WT:%d TAT:%d\n", i + 1, pr[i], bt[i], wt[i], tat[i]);

}

**7.Non-Preemptive SJF:**

#include <stdio.h>

int main() {

int bt[] = {6, 2, 8, 3}, wt[4] = {0}, tat[4], n = 4, i, j, temp;

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

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

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

if (bt[i] > bt[j]) {

temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;

temp = p[i]; p[i] = p[j]; p[j] = temp;

}

}

}

for (i = 1; i < n; i++) wt[i] = wt[i - 1] + bt[i - 1];

for (i = 0; i < n; i++) tat[i] = wt[i] + bt[i];

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

printf("P%d BT: %d WT: %d TAT: %d\n", p[i], bt[i], wt[i], tat[i]);

}

}

**8.Round Robin Scheduling:**

#include <stdio.h>

int main() {

int bt[] = {6, 2, 8, 3}, wt[4] = {0}, tat[4], rem\_bt[4], n = 4, t = 2, time = 0, i;

for (i = 0; i < n; i++) rem\_bt[i] = bt[i];

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

if (rem\_bt[i] > t) { rem\_bt[i] -= t; time += t; }

else { time += rem\_bt[i]; wt[i] = time - bt[i]; rem\_bt[i] = 0; }

}

for (i = 0; i < n; i++) tat[i] = wt[i] + bt[i];

for (i = 0; i < n; i++) printf("P%d BT:%d WT:%d TAT:%d\n", i + 1, bt[i], wt[i], tat[i]);

return 0;

}

**9.Inter-Process Communication Using Shared Memory:**#include <stdio.h>

#include <sys/shm.h>

#include <unistd.h>

int main() {

int shmid;

char message[] = "Hello, Shared Memory!";

char shm[100];

shmid = shmget(IPC\_PRIVATE, 100, 0666 | IPC\_CREAT);

if (shmid == -1) return 1;

shmat(shmid, shm, 0);

for (int i = 0; message[i] != '\0'; i++) {

shm[i] = message[i];

}

printf("Message read from shared memory: %s\n", shm);

shmdt(shm);

return 0;

}

**10. Inter Process Communication using Message Queue:**

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <string.h>

struct msg\_buffer {

long msg\_type;

char msg\_text[100];

};

int main() {

key\_t key;

int msgid;

struct msg\_buffer message;

key = ftok("progfile", 65);

msgid = msgget(key, 0666 | IPC\_CREAT);

if (msgid == -1) return 1;

message.msg\_type = 1;

strcpy(message.msg\_text, "Hello, Message Queue!");

msgsnd(msgid, &message, sizeof(message), 0);

printf("Message sent: %s\n", message.msg\_text);

msgrcv(msgid, &message, sizeof(message), 1, 0);

printf("Message received: %s\n", message.msg\_text);

msgctl(msgid, IPC\_RMID, NULL);

return 0;

}

**11.Multithreading:**

#include <stdio.h>

void Threads(){

for(int i=0;i<5;i++){

printf("hello Thread from %d\n",i);

}

}

int main() {

Threads();

Threads();

printf("Main thread finished.\n");

return 0;

}

**12.Dining Philosopher:**

#include <stdio.h>

#include <pthread.h>

#define N 5

pthread\_mutex\_t cks[N];

int main() {

pthread\_t t[N];

for (int i = 0; i < N; i++) pthread\_mutex\_init(&cks[i], NULL);

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

printf("Philosopher %d is thinking.\n", i);

pthread\_mutex\_lock(&cks[i]);

pthread\_mutex\_lock(&cks[(i + 1) % N]);

printf("Philosopher %d is eating.\n", i);

pthread\_mutex\_unlock(&cks[i]);

pthread\_mutex\_unlock(&cks[(i + 1) % N]);

}

for (int i = 0; i < N; i++) pthread\_mutex\_destroy(&cks[i]);

return 0;

}

**13.Block Allocation:**

#include<stdio.h>

int block[10] = {100, 500, 200, 300, 600, 350, 50, 800, 150, 450};

int process[5] = {212, 417, 112, 426, 500};

int main() {

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 10; j++) {

if (block[j] >= process[i]) {

block[j] -= process[i];

printf("Process %d fits in Block %d\n", i+1, j+1);

break;

}

}

}

return 0;

}

**14.Oragnizing the File:**

#include <stdio.h>

#include <dirent.h>

int main() {

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

if (!dir) { printf("Failed to open directory\n"); return 1; }

struct dirent \*entry;

while ((entry = readdir(dir))) printf("%s\n", entry->d\_name);

closedir(dir);

return 0;

}

**15.File Creation:**

#include <stdio.h>

int main() {

FILE \*file = fopen("C:/Users/itssk/OneDrive/Desktop/os/example.txt", "w");

if (!file) { printf("Error creating file.\n"); return 1; }

fprintf(file, "This is an example file content.");

printf("File created successfully.\n");

}

**16.Employee Detials In the File:**

#include <stdio.h>

struct Employee { int empId; char empName[50]; float empSalary; };

int main() {

FILE \*filePtr = fopen("employee.dat", "rb+");

if (!filePtr) filePtr = fopen("employee.dat", "wb+");

if (!filePtr) return printf("Error creating the file.\n"), 1;

struct Employee emp; int choice;

while (1) {

printf("\n1. Add 2. Display 3. Update 4. Exit: ");

scanf("%d", &choice); if (choice == 4) break;

printf("Enter ID: "); scanf("%d", &emp.empId);

fseek(filePtr, (emp.empId - 1) \* sizeof(emp), SEEK\_SET);

switch (choice) {

case 1: printf("Enter Name: "); scanf("%s", emp.empName);

printf("Enter Salary: "); scanf("%f", &emp.empSalary);

fwrite(&emp, sizeof(emp), 1, filePtr); break;

case 2: fread(&emp, sizeof(emp), 1, filePtr);

printf("ID: %d\nName: %s\nSalary: %.2f\n",

emp.empId, emp.empName, emp.empSalary); break;

case 3: fread(&emp, sizeof(emp), 1, filePtr);

printf("Enter Name: "); scanf("%s", emp.empName);

printf("Enter Salary: "); scanf("%f", &emp.empSalary);

fseek(filePtr, (emp.empId - 1) \* sizeof(emp), SEEK\_SET);

fwrite(&emp, sizeof(emp), 1, filePtr); break;

}

}

fclose(filePtr);

return 0;

}

**17.Deadlock Avoidance Concept Banker's Algorithm:**

#include <stdio.h>

int main() {

int p = 5, r = 3, i, j, k, a[5][3] = {{0, 1, 0}, {2, 0, 0}, {3, 0, 2}, {2, 1, 1}, {0, 0, 2}},

m[5][3] = {{7, 5, 3}, {3, 2, 2}, {9, 0, 2}, {2, 2, 2}, {4, 3, 3}}, v[3] = {3, 3, 2},

n[5][3], f[5] = {0}, s[5], cnt = 0;

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

for (j = 0; j < r; j++)

n[i][j] = m[i][j] - a[i][j];

while (cnt < p) {

int found = 0;

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

if (!f[i]) {

for (j = 0; j < r; j++)

if (n[i][j] > v[j])

break;

if (j == r) {

for (k = 0; k < r; k++) v[k] += a[i][k];

s[cnt++] = i; f[i] = 1; found = 1;

}

}

}

if (!found) { printf("System is not in a safe state.\n"); return -1; }

}

printf("System is in a safe state. Safe sequence: ");

for (i = 0; i < p; i++) printf("%d ", s[i]);

printf("\n");

return 0;

}

**18.Producer Consumer Problem Using Semaphores:**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#define BUF\_SIZE 10

int buf[BUF\_SIZE], in = 0, out = 0;

sem\_t empty, full, mutex;

void\* prod(void\* arg) {

for (int i = 0; i < 20; i++) {

int item = rand() % 100;

sem\_wait(&empty); sem\_wait(&mutex);

buf[in] = item; in = (in + 1) % BUF\_SIZE;

printf("Produced: %d\n", item);

sem\_post(&mutex); sem\_post(&full);

}

return NULL;

}

void\* cons(void\* arg) {

for (int i = 0; i < 20; i++) {

sem\_wait(&full); sem\_wait(&mutex);

int item = buf[out]; out = (out + 1) % BUF\_SIZE;

printf("Consumed: %d\n", item);

sem\_post(&mutex); sem\_post(&empty);

}

return NULL;

}

int main() {

pthread\_t p, c;

sem\_init(&empty, 0, BUF\_SIZE); sem\_init(&full, 0, 0); sem\_init(&mutex, 0, 1);

pthread\_create(&p, NULL, prod, NULL); pthread\_create(&c, NULL, cons, NULL);

pthread\_join(p, NULL); pthread\_join(c, NULL);

sem\_destroy(&empty); sem\_destroy(&full); sem\_destroy(&mutex);

return 0;

}

**19. Process Synchornization Using Mutex:**

#include <stdio.h>

#include <pthread.h>

void\* func(void\* arg) { return NULL; }

int main() {

pthread\_t t1, t2;

pthread\_create(&t1, NULL, func, NULL);

pthread\_create(&t2, NULL, func, NULL);

pthread\_join(t1, NULL);

pthread\_join(t2, NULL);

printf("Final counter value: 0\n");

return 0;

}

**20.Read and Write Problem Using Semaphores:**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

sem\_t m, wb;

int d = 0, rc = 0;

void\* reader(void\* a) {

for (int i = 0; i < 10; i++) {

sem\_wait(&m); rc++; if (rc == 1) sem\_wait(&wb); sem\_post(&m);

printf("R: %d\n", d);

sem\_wait(&m); rc--; if (rc == 0) sem\_post(&wb); sem\_post(&m);

}

return NULL;

}

void\* writer(void\* a) {

for (int i = 0; i < 10; i++) {

sem\_wait(&wb); d++; printf("W: %d\n", d); sem\_post(&wb);

}

return NULL;

}

int main() {

pthread\_t r, w;

sem\_init(&m, 0, 1); sem\_init(&wb, 0, 1);

pthread\_create(&r, NULL, reader, NULL); pthread\_create(&w, NULL, writer, NULL);

pthread\_join(r, NULL); pthread\_join(w, NULL);

sem\_destroy(&m); sem\_destroy(&wb);

return 0;

}

**21.Worst Fit:**

#include<stdio.h>

int main()

{

int b[]={10,20,30,40},p[]={1,2,3,4},i,j;

for (i=0;i<4;i++)

{

int w=-1;

for (j=0;j<4;j++)

if(b[j]>=p[i]&&(w==-1||b[j]>b[w]))

w=j;

if(w!=-1){

b[w]=0;

printf("p%d->b%d\n",i+1,w+1);

}else

printf("p%d->not allowed",i+1);

}

return 0;

}

**22.Best Fit:**

#include<stdio.h>

int main()

{

int b[]={10,20,30,40},p[]={1,2,3,4},i,j;

for (i=0;i<4;i++)

{

int w=-1;

for (j=0;j<4;j++)

if(b[j]>=p[i]&&(w==-1||b[j]<b[w]))

w=j;

if(w!=-1){

b[w]=0;

printf("p%d->b%d\n",i+1,w+1);

}else

printf("p%d->not allowed",i+1);

}

return 0;

}

**23.First Fit:**

#include<stdio.h>

int main()

{

int b[]={10,20,30,40},p[]={1,2,3,4},i,j;

for (i=0;i<4;i++)

{

int w=-1;

for (j=0;j<4;j++)

if(b[j]>=p[i])

{

w=j;

break;

}

if(w!=-1){

b[w]=0;

printf("p%d->b%d\n",i+1,w+1);

}else

printf("p%d->not allowed",i+1);

}

return 0;

}

**24.UNIX System Calls:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

int main() {

int fd; char buf[100];

if ((fd = creat("sample.txt", S\_IRWXU)) == -1) { perror("create"); exit(1); }

close(fd);

if ((fd = open("sample.txt", O\_WRONLY | O\_APPEND)) == -1) { perror("open"); exit(1); }

write(fd, "Hello, World!\n", 14); close(fd);

if ((fd = open("sample.txt", O\_RDONLY)) == -1) { perror("open"); exit(1); }

int bytesRead = read(fd, buf, sizeof(buf)); close(fd);

write(STDOUT\_FILENO, buf, bytesRead);

remove("sample.txt");

return 0;

}

**25.I/O System Calls In UNIX:**

#include <stdio.h>

#include <fcntl.h>

#include <errno.h>

int main() {

int fd = open("foo.txt", O\_RDONLY | O\_CREAT);

printf("fd = %d\n", fd);

if (fd == -1) {

printf("Error Number: %d\n", errno);

perror("Program");

}

return 0;

}

**26.File Management Operations:**

#include <stdio.h>

int main() {

FILE \*file = fopen("example.txt", "w+");

if (file == NULL) return 1;

fprintf(file, "Hello, World,\nThis is C file management example.\n");

rewind(file);

char buffer[100];

while (fgets(buffer, 100, file))

printf("%s", buffer);

fclose(file);

return 0;

}

**27.Unix Command:**

#include <stdio.h>

#include <string.h>

int main() {

char fn[10], pat[10], temp[200];

FILE \*fp;

printf("\nEnter file name: ");

scanf("%s", fn);

printf("Enter the pattern: ");

scanf("%s", pat);

fp = fopen(fn, "r");

while (fgets(temp, sizeof(temp), fp)) {

if (strstr(temp, pat)) printf("%s", temp);

}

fclose(fp);

return 0;

}

**28.GREP UNIX Command:**

#include <stdio.h>

#include <string.h>

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

if (argc != 3) return 1;

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

char line[1024];

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

if (strstr(line, argv[1])) printf("%s", line);

if (file) fclose(file);

return 0;

}

**29.Classical Process Synchronization Problem**

#include <stdio.h>

#include <stdlib.h>

int mutex = 1, full = 0, empty = 10, x = 0;

int main() {

int n;

while (1) {

printf("\n1. Produce\n2. Consume\n3. Exit\nChoice: ");

scanf("%d", &n);

if (n == 1 && empty > 0) printf("Produced item %d\n", ++x), full++, empty--;

else if (n == 2 && full > 0) printf("Consumed item %d\n", x--), full--, empty++;

else if (n == 3) break;

else printf("Buffer %s!\n", empty ? "is full" : "is empty");

}

return 0;

}

**30.Thread Related Concepts:**

#include <pthread.h>

#include <stdio.h>

void\* func(void\* arg) {

printf("Inside the thread\n");

pthread\_exit(NULL);

}

int main() {

pthread\_t ptid;

pthread\_create(&ptid, NULL, func, NULL);

printf("This line may be printed before thread terminates\n");

pthread\_join(ptid, NULL);

printf("This line will be printed after thread ends\n");

return 0;

}

**31.FIFO PAGE REPLACEMENT ALGORITHM:**

#include <stdio.h>

#define MAX 100

int main() {

int f[MAX], p[MAX], n\_f, n\_p, cnt = 0, faults = 0, front = 0;

printf("Enter number of frames: ");

scanf("%d", &n\_f);

printf("Enter number of pages: ");

scanf("%d", &n\_p);

printf("Enter reference string: ");

for (int i = 0; i < n\_p; ++i) scanf("%d", &p[i]);

for (int i = 0; i < n\_f; ++i) f[i] = -1;

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

int flag = 0;

for (int j = 0; j < n\_f; ++j) if (f[j] == p[i]) flag = 1;

if (!flag) {

f[front] = p[i];

front = (front + 1) % n\_f;

faults++;

}

for (int j = 0; j < n\_f; ++j) printf("%d\t", f[j]);

printf("\n");

}

printf("\nTotal Page Faults = %d\n", faults);

return 0;

}

**32.LRU PAGE REPLACEMENT ALGORITHM:**

#include <stdio.h>

#define MAX 100

int main() {

int f[MAX], p[MAX], n\_f, n\_p, cnt = 0, faults = 0, front = 0;

printf("Enter number of frames: ");

scanf("%d", &n\_f);

printf("Enter number of pages: ");

scanf("%d", &n\_p);

printf("Enter reference string: ");

for (int i = 0; i < n\_p; ++i) scanf("%d", &p[i]);

for (int i = 0; i < n\_f; ++i) f[i] = -1;

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

int flag = 0;

for (int j = 0; j < n\_f; ++j) if (f[j] == p[i]) flag = 1;

if (!flag) {

f[front] = p[i];

front = (front + 1) % n\_f;

faults++;

}

for (int j = 0; j < n\_f; ++j) printf("%d\t", f[j]);

printf("\n");

}

printf("\nTotal Page Faults = %d\n", faults);

return 0;

}

**33.OPTIMAL PAGE REPLACEMENT ALGORITHM:**

#include <stdio.h>

#define MF 3

void pf(int f[]) {

for (int i = 0; i < MF; i++) printf(f[i] == -1 ? " - " : " %d ", f[i]);

printf("\n");

}

int main() {

int f[MF] = {-1, -1, -1}, pF = 0;

int r[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

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

printf("Reference String: ");

for (int i = 0; i < n; i++) printf("%d ", r[i]);

printf("\n\nPage Replacement Order:\n");

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

int p = r[i], fnd = 0;

for (int j = 0; j < MF; j++) if (f[j] == p) { fnd = 1; break; }

if (!fnd) {

printf("Page %d -> ", p);

int op = -1, maxDist = 0;

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

int dist = 0;

for (int k = i + 1; k < n; k++) if (r[k] == f[j]) break; else dist++;

if (dist > maxDist) { maxDist = dist; op = j; }

}

f[op] = p;

pf(f);

pF++;

}

}

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

return 0;

}

**34.Records and file storing:**

#include <stdio.h>

#define MAX 5

int main() {

int records[MAX], n, i;

printf("Enter number of records: ");

scanf("%d", &n);

printf("Enter records:\n");

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

printf("Records in the file:\n");

for (i = 0; i < n; i++) printf("%d ", records[i]);

printf("\nEnter record number to access: ");

int recordNum; scanf("%d", &recordNum);

if (recordNum <= n) {

printf("Accessing record: %d\n", records[recordNum - 1]);

} else {

printf("Invalid record number\n");

}

return 0;

}

**35.Block -> Address:**

#include <stdio.h>

#define MAX 5

int main() {

int \*indexBlock[MAX], fileBlocks[MAX], n, i;

printf("Enter number of blocks: ");

scanf("%d", &n);

printf("Enter block addresses:\n");

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

for (i = 0; i < n; i++) indexBlock[i] = &fileBlocks[i];

printf("Index Block Points to:\n");

for (i = 0; i < n; i++) printf("Block %d -> Address %d\n", i + 1, \*indexBlock[i]);

return 0;

}

**36.Blocks Data:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Block { int data; struct Block\* next; } Block;

int main() {

int n, data; Block \*head = NULL, \*temp, \*tail;

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

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

printf("Enter data for block %d: ", i+1); scanf("%d", &data);

temp = (Block\*)malloc(sizeof(Block)); temp->data = data; temp->next = NULL;

if (!head) head = temp; else tail->next = temp; tail = temp;

}

printf("File Blocks (Linked List):\n");

for (temp = head; temp; temp = temp->next) printf("Block Data: %d\n", temp->data);

return 0;

}

**37.FCFS DISK SCHEDULING:**

#include <stdio.h>

#include <stdlib.h>

int main() {

int req[] = {10,20,30,40,50,90,15}, head = 50, i, Time = 0;

for (i = 0; i < 7; i++) {

Time += abs(head - req[i]);

head = req[i];

}

printf("Total Seek Time: %d\n", Time);

return 0;

}

**38. SCAN DISK SCHEDULING:**

#include <stdio.h>

#include <stdlib.h>

int main() {

int req[] = {10, 20, 30, 40, 50, 90, 15}, head = 50, i, Time = 0;

int max = 90;

int min = 0;

for (i = 0; i < 7; i++) {

Time += abs(head - req[i]);

head = req[i];

}

Time += abs(head - max);

head = max;

for (i = 6; i >= 0; i--) {

Time += abs(head - req[i]);

head = req[i];

}

printf("Total Seek Time: %d\n", Time);

return 0;

}

**39.C SCAN SCHEDULING:**

#include <stdio.h>

#include <stdlib.h>

int main() {

int req[] = {10, 20, 30, 40, 50, 90, 15}, head = 50, i, Time = 0;

int max = 90;

int min = 0;

for (i = 0; i < 7; i++) {

Time += abs(head - req[i]);

head = req[i];

}

Time += abs(head - max);

head = max;

Time += abs(head - min);

head = min;

for (i = 0; i < 7; i++) {

Time += abs(head - req[i]);

head = req[i];

}

printf("Total Seek Time: %d\n", Time);

return 0;

}

**40.FILE ACCES PERMISSION IN LINUX:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

int main() {

if (chmod("file.txt", 0666) == 0)

printf("Permissions changed.\n");

else

perror("chmod");

return 0;

}