Assignment NO:01 PROGRAM:

```
fileName="MyAddressBook"

# Function to create a new address book create_address_book()
{    if [ -e  "$fileName" ];
 then
rm
"$fileName"
  fi
 echo -e "NAME\tNUMBER\t\tADDRESS\n======================================"
>
"$fileName"

echo "Address book
created." }

# Function to view the address book view_address_book()
{
   if [ -e "$fileName" ];
 then
 cat "$fileName"
 else        echo "Address book is
empty."     fi }

# Function to insert a new
record insert_record() {
echo -e "\nEnter Name:"
read -r name
echo "Enter
Phone Number of $name:"
read -r number
echo "Enter Address of
$name:"     read -r address
echo -e
"$name\t$number\t$address" >>
"$fileName"     echo "Record added." }

# Function to delete a record delete_record()
{
echo -e "\nDelete record\nEnter Name/Phone Number:"
read -r pattern
if [ -e "$fileName" ]; then
grep -v "$pattern" "$fileName" > temp && mv temp
"$fileName"         echo "Record deleted if it existed."
else
echo "Address book does not exist."
fi }

# Function to modify a record modify_record()
{
echo -e "\nModify record\nEnter Name/Phone
Number:"
```

```
read -r pattern
 if [ -e "$fileName" ];
then
grep -v "$pattern" "$fileName" >
temp        mv temp "$fileName"
echo "Enter Name:"
read -r name
echo "Enter Phone Number of $name:"
read -r number
echo "Enter Address of
$name:"
 read -r address
 echo -e "$name\t$number\t$address" >>
"$fileName"                    echo "Record
modified."
 else
echo "Address book does not
exist."     fi  }

# Main program loop while
true; do
   echo -e
"\n*************MENU*************"      echo
"Choose one of the Following:"
echo "1. Create Address Book"
echo "2. View Address
Book"

 echo "3. Insert a Record"

  echo "4. Delete a Record"

  echo "5. Modify a Record"

  echo "6. Exit"

  echo -n "Enter your choice (1-6): "     read -r
opt

   case $opt in
1)create_address_book ;;
2)view_address_book ;;
3)insert_record ;;
4)delete_record ;;
5)modify_record ;;
6)echo "Exiting..."; break ;;
        *) echo "Invalid option. Please choose between 1-6." ;;
esac done
```

OUTPUT

Assignment NO:02 PROGRAM:

******** Assign2.c**********

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h> #include <sys/
wait.h> int main() { int n;  printf("Enter the
number of elements: "); scanf("%d",
&n); char *args[n + 2]; args[0] =
"./2Bchild"; // Program to execute args[n + 1] =
NULL; // Null terminate the arguments
printf("Enter the elements of the array:\n"); for
(int i = 1; i <= n; i++) { args[i] = (char
*)malloc(10 * sizeof(char)); scanf("%s", args[i]);
  } pid_t pid = fork(); if (pid < 0) { perror("Fork
failed");
exit(EXIT_FAILURE)
;  }

if (pid == 0) { // Child process
execve(args[0], args, NULL);
perror("Execve failed"); exit(EXIT_FAILURE);  }
else { // Parent process wait(NULL);
printf("Parent process: Child process has
completed.\n");  for (int i = 1; i <= n; i++)
{  free(args[i]); } } return 0;
}
```

**********2bchild.c**********

```
#include <stdio.h> int main(int
argc, char *argv[]) { printf("Array in
reverse order:\n"); for (int i = argc -
1; i > 0; i--)
{  printf("%s ",
argv[i]);  }
printf("\n"); return
0;
}
```

OUTPUT

Assignment NO:03

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h> #define
N 100

struct process {    int
process_id;    int
arrival_time;    int
burst_time;    int
waiting_time;    int
turn_around_time;    int
remaining_time;
};

int queue[N]; int front
= 0, rear = 0; struct
process proc[N];

void push(int process_id) {    queue[rear]
= process_id;    rear
= (rear + 1) % N;
}

int pop() {    if (front
== rear)       return -
1;
   int return_position =
queue[front];    front = (front +
```

```
1) % N;     return
return_position;
}

int main() {     float wait_time_total
= 0.0, tat = 0.0;     int n,
time_quantum;

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

   for (int i = 0; i < n; i++) {         printf("Enter the
arrival time for process %d: ", i + 1);         scanf("%d",
&proc[i].arrival_time);         printf("Enter the burst time
for process %d: ", i + 1);         scanf("%d",
&proc[i].burst_time);         proc[i].process_id = i + 1;
proc[i].remaining_time = proc[i].burst_time;
   }

   printf("Enter time quantum: ");     scanf("%d",
&time_quantum);

   int time = 0;     int
processes_left = n;     int
position = -1;     int
local_time
= 0;
   for (int j = 0; j < n; j++)         if
(proc[j].arrival_time == time)             push(j);

   while (processes_left)
{         if (local_time == 0)
{            if (position !=
-1)
push(position);
position = pop();
      }

      for (int i = 0; i < n; i+
+) {             if
(proc[i].arrival_time >
time)
continue;             if (i ==
position)
continue;
         if (proc[i].remaining_time == 0)
continue;           proc[i].waiting_time++;
proc[i].turn_around_time++;
      }

      if (position != -1) {
proc[position].remaining_time--;           proc[position].turn_around_time+
+;            if
(proc[position].remaining_time == 0) {
            processes_left--;
```

```
local_time = -1;
position = -1;
        }
    } else {
        local_time = -1;
    }

    time++;       local_time = (local_time + 1) %
time_quantum;

    for (int j = 0; j < n; j++)           if
(proc[j].arrival_time == time)
            push(j);
  }

  printf("\nProcess\t\tArrival Time\tBurst Time\tWaiting time\tTurn around
time\n");     for (int i = 0; i < n; i++) {
    printf("%d\t\t%d\t\t", proc[i].process_id, proc[i].arrival_time);
printf("%d\t\t%d\t\t%d\n", proc[i].burst_time, proc[i].waiting_time,
proc[i].turn_around_time);
tat += proc[i].turn_around_time;       wait_time_total += proc[i].waiting_time;
  }    tat = tat / (1.0 * n);

wait_time_total =

wait_time_total / (1.0 * n);    printf("\nAverage
waiting time: %f", wait_time_total);
printf("\nAverage turn around time: %f\n", tat);

  return 0; // Ensure to return 0 at the end of main }
```

**********SJFP**********

PROGRAM

```
#include <stdio.h>
struct Process { int
id; int arrivalTime;
int burstTime; int
waitingTime; int
turnAroundTime;
}; void calculateTimes(struct Process proc[], int n) { int
totalWaitingTime = 0, totalTurnAroundTime =
0; int completionTime[n];// Sort the processes by Arrival
Time and Burst Time for (int i = 0; i < n - 1; i++) { for (int j = i
+ 1; j < n; j++) { if (proc[i].arrivalTime > proc[j].arrivalTime ||
(proc[i].arrivalTime == proc[j].arrivalTime && proc[i].burstTime > proc[j].burstTime))
{ struct Process temp = proc[i]; proc[i] = proc[j];
```

```
proc[j] = temp; }}}
// Initialize the completion time of the first process completionTime[0]
= proc[0].arrivalTime + proc[0].burstTime; proc[0].turnAroundTime
= proc[0].burstTime; proc[0].waitingTime =
0;
// Calculate waiting time and turn-around time for each process for
(int i = 1; i < n; i++) {
// Calculate completion time for this process completionTime[i] =
completionTime[i-1] + proc[i].burstTime; // Turn Around Time = Completion
Time - Arrival Time proc[i].turnAroundTime
=  completionTime[i] - proc[i].arrivalTime; //
Waiting Time = Turn Around Time - Burst
Time
proc[i].waitingTime = proc[i].turnAroundTime - proc[i].burstTime;
}
// Display results printf("Process\tBurst Time\tArrival Time\tWaiting
Time\tTurnAround Time\n");
for (int i = 0; i < n; i++) {
printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\n", proc[i].id, proc[i].burstTime,
proc[i].arrivalTime, proc[i].waitingTime, proc[i].turnAroundTime); totalWaitingTime +=
proc[i].waitingTime; totalTurnAroundTime += proc[i].turnAroundTime;
}
printf("Average waiting time: %.2f\n", (float)totalWaitingTime / n);
printf("Average turn around time: %.2f\n",
(float)totalTurnAroundTime / n);  } int main()
{
int n;
printf("Enter number of
processes: ");  scanf("%d", &n);
struct Process proc[n];
// Input arrival time and burst time for each process
for (int i = 0; i < n; i++) { proc[i].id = i + 1;printf("Enter arrival
time for process %d: ", proc[i].id);  scanf("%d",
&proc[i].arrivalTime);
printf("Enter burst time for process %d: ", proc[i].id);
scanf("%d", &proc[i].burstTime);
}
// Calculate and display the scheduling results
calculateTimes(proc, n); return 0; }
```

OUTPUT:

Assignment NO:04 PROGRAM:

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

```
#include <unistd.h> #include
<stdlib.h> void *producer(void
*thread); void *consumer(void
*thread); int count = 0, in = 0,
out  = 0, a[5]; sem_t full; sem_t
empty;pthread_mutex_t mutex;
int main() { int i, p, c; pthread_t
pid[10], cid[10]; pthread_mutex_init(&mutex,
NULL);
sem_init (&full, 0, 0); sem_init (&empty, 0, 5); printf("\nEnter
number of producers: "); scanf("%d", &p); printf("\nEnter
number of consumers: "); scanf("%d", &c); int
producer_indices[p]; int consumer_indices[c]; for (i = 0; i < p;
i++) { producer_indices[i] = i; pthread_create(&pid[i], NULL,
producer, &producer_indices[i]);
}
for (i = 0; i < c; i++) { consumer_indices[i] = i;
pthread_create(&cid[i], NULL, consumer, &consumer_indices[i]);
}  for (i = 0; i < p; i++)
{ pthread_join(pid[i],
NULL);
}
for (i = 0; i < c; i++) { pthread_join(cid[i],
NULL); } sem_destroy(&full);
sem_destroy(&empty);
pthread_mutex_destroy(&mutex); return
0;
} void *producer(void *thread)
{
int t = *(int *)thread;  while
(1) { sem_wait(&empty);
pthread_mutex_lock(&mutex); if (count >=
5) { printf("\nBuffer is full"); } else { a[in] =
rand() % 100; printf("\nProducer %d
produced:
%d", t, a[in]); in = (in + 1) % 5;count+
+;  } pthread_mutex_unlock(&mutex);
sem_post(&full);  sleep(1); }
pthread_exit(0);
} void *consumer(void *thread)
{
int t = *(int *)thread; while (1)
{ sem_wait(&full);
pthread_mutex_lock(&mutex);
if (count <= 0)
{ printf("\nBuffer is empty");
} else { printf("\nConsumer %d consumed:
%d", t, a[out]);  out = (out + 1) % 5;  count--; }
pthread_mutex_unlock(&mutex);
sem_post(&empty); sleep(1); }
pthread_exit(0);
}
```

**********Reader and Writer**********

```
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include <unistd.h> #define

BUFFER_SIZE 16

int buffer[BUFFER_SIZE]; sem_t database,  mutex;
int counter, readerCount; pthread_t
readerThread[50], writerThread[50];
```

```
void init()
{     sem_init(&mutex, 0, 1);
sem_init(&database, 0, 1);
counter
= 0;
readerCount =
0;
}

void *writer(void *param)
{    sem_wait(&database);    int item = rand() %
5;    buffer[counter] = item;
   printf("Data written by the writer %d is %d\n", (*(int *)param),
buffer[counter]);    counter++;    sleep(1);    sem_post(&database);
return  NULL; }

void *reader(void *param)
{     sem_wait(&mutex);
readerCount++;     if
(readerCount
== 1) {
sem_wait(&database);
   }
   sem_post(&mutex);

   // Read from the buffer
   int dataRead = buffer[counter - 1]; // Read the last written item
printf("Data read by the reader %d is %d\n", (*(int *)param),
dataRead);

   sleep(1);
   sem_wait(&mutex);
readerCount--;     if (readerCount
== 0) {
sem_post(&database);
   }
   sem_post(&mutex);
return NULL;
} int main()
{     ini
t();
   int no_of_writers, no_of_readers;

   printf("Enter number of readers: ");
scanf("%d", &no_of_readers);     printf("Enter
number of writers: ");     scanf("%d",
&no_of_writers);

   int
writerIndices[no_of_writers];     int
readerIndices[no_of_readers];

   for (int i = 0; i < no_of_writers; i++) {
      writerIndices[i] = i;  // Store indices to pass to threads
```

```
        pthread_create(&writerThread[i], NULL, writer, &writerIndices[i]);
    }

    for (int i = 0; i < no_of_readers; i++) {
        readerIndices[i] = i;  // Store indices to pass to threads
        pthread_create(&readerThread[i], NULL, reader, &readerIndices[i]);
    }

    for (int i = 0; i < no_of_writers; i++) {
pthread_join(writerThread[i], NULL);
    }
    for (int i = 0; i < no_of_readers; i++)
{        pthread_join(readerThread[i], NULL);
    }

sem_destroy(&database);
 sem_destroy(&mutex);
return 0; }
```

Assignment NO:05

PROGRAM:

```
#include <stdio.h> int max[100][100];
int alloc[100][100]; int need[100][100];
int avail[100]; int n, r; void input(); void
show(); void cal(); int main()
{ printf("**** Banker's Algorithm
****\n");
input();
show();
cal();
getchar();
return 0;  }
void input()
{
int i, j;
printf("Enter the number of processes: ");
scanf("%d", &n); printf("Enter the number of
resource instances: "); scanf("%d", &r);
printf("Enter the Max Matrix\n"); for (i = 0; i < n;
i++) { for (j = 0; j < r; j++) { scanf("%d",
&max[i][j]);
} }
printf("Enter the Allocation
Matrix\n");  for (i = 0; i < n; i++) {  for
(j = 0; j < r; j++) { scanf("%d",
&alloc[i][j]);
} }
printf("Enter the available resources\n");
for (j = 0; j < r; j++) { scanf("%d",
&avail[j]);
} } void show()
{  int
i, j;
printf("Process\t Allocation\t Max\t
Available\n");
for (i = 0; i < n; i++)
{ printf("P%d\t ", i + 1);
for (j = 0; j < r; j++)
{ printf("%d ", alloc[i][j]);
}  printf("\t"); for (j = 0; j
< r; j++) { printf("%d ",
max[i][j]); } printf("\t"); if
(i == 0) { for (j = 0; j
< r; j++)
{ printf("%d ", avail[j]);
} } printf("\n"); }} void cal() { int finish[100],
temp, flag = 1, k, c1 = 0; int safe[100]; int i,
j; for (i = 0; i < n; i++) { finish[i] = 0;
}  for (i = 0; i < n; i++) { for (j =
0; j
< r; j++) { need[i][j] = max[i][j]
```

```
- alloc[i][j];
} } printf("\n"); while (flag) { flag = 0; for (i =
0; i < n; i++) { int c = 0; for (j = 0; j < r; j++)
{ if (finish[i] == 0 && need[i][j] <=  avail[j])
{ c++;  } if (c == r) { for (k = 0; k < r; k++)
{ avail[k] += alloc[i][k];  } finish[i] = 1; flag
=
1; printf("P%d -> ",
i); } } } } for (i = 0; i
< n; i++) { if (finish[i]
== 1) { c1++; } else
{ printf("P%d -> ",
i); } } if (c1 == n) {
printf("\nThe system is in a safe state\n");
 } else { printf("\nProcesses are in
deadlock\n"); printf("System is in an
unsafe state\n");
} }
```

Assignment NO:06

PROGRAM:

```
#include <stdio.h> void printFrames(int
frames[], int frameSize) { for (int i = 0; i <
frameSize; i++)
{ if (frames[i] == -1)
{  printf(" - "); } else { printf("
%d ", frames[i]);
} }
printf("\n");
} void fcfs(int refString[], int refSize, int
frameSize)
{ int frames[frameSize]; for (int i = 0; i <
frameSize; i++) frames[i] = -1; int pageFaults = 0,
nextReplace = 0; printf("\nFCFS Page
Replacement:\n"); for (int i = 0; i < refSize; i++)
{ int found = 0; for (int j = 0; j < frameSize; j++)
{ if
(frames[j] == refString[i]) { found = 1; break; }
}
if (!found) { frames[nextReplace] =
refString[i]; nextReplace = (nextReplace + 1)
% frameSize; pageFaults++; }
printFrames(frames, frameSize);
}
printf("Total Page Faults: %d\n", pageFaults);  } int
main() { int refSize, frameSize; printf("Enter the
number of pages in the reference string: "); scanf("%d",
&refSize); int refString[refSize]; printf("Enter the
reference string:\n"); for (int i = 0; i < refSize; i++)
{ scanf("%d",
&refString[i]);
}
printf("Enter the number of frames (minimum
3): "); scanf("%d", &frameSize); if (frameSize <
3) { printf("Frame size should be at least 3.\n");
return 1;
} fcfs(refString, refSize,
frameSize); return 0; }
```

**********LFS**********

PROGRAM

```
#include <stdio.h>
void printFrames(int frames[], int frameSize) {
for (int i = 0; i < frameSize; i++)
{  if (frames[i] == -1) { printf(" -
");
} else { printf(" %d ", frames[i
]);
} }
printf("\n");
} int findOptimal(int frames[], int frameSize, int refString[], int refSize, int currentIndex)
{
int farthest = currentIndex; int
index = -1;  for (int i = 0; i <
```

```
frameSize; i++) { int j;
for (j = currentIndex; j < refSize; j++)
{  if (frames[i] == refString[j]) {
if (j > farthest)
{  farthest = j; index
= i; } break; }
}
if (j == refSize) return i; // If not found in future, replace this
}return index == -1 ? 0 : index;
}  void optimal(int refString[], int refSize, int
frameSize) { int frames[frameSize]; for (int i =
0; i < frameSize; i++) frames[i] = -1;  int
pageFaults = 0;
printf("\nOptimal Page
Replacement:\n");  for (int i = 0; i <
refSize; i++) {  int found = 0; for (int
j = 0; j < frameSize; j++) { if
(frames[j] ==  refString[i]) {  found
= 1;  break; }  } if (!found)
{ int
replaceIn
de x = (i <
frameSiz
e)
? i :
findOptimal
(frames,
frameSize,
refString,
refSize, i +
1);
frames[replaceIndex] = refString[i];
pageFaults++;
}
printFrames(frames, frameSize);
}  printf("Total Page Faults: %d\n",
pageFaults);
}  int main()
{ int  refSize,
frameSize;
printf("Enter the number of pages in the reference string: ");
scanf("%d", &refSize); int refString[refSize]; printf("Enter the
reference string:\n"); for (int i = 0; i < refSize; i++) {
scanf("%d", &refString[i]);
}
printf("Enter the number of frames
(minimum 3): ");  scanf("%d", &frameSize);
if (frameSize < 3) {  printf("Frame size
should be at least 3.\n");  return 1; }
optimal(refString, refSize, frameSize);
return 0;
}
```

Assignment NO:07

PROGRAM:

**********7a.c**********

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#define PIPE1 "/tmp/pipe1" #define
PIPE2 "/tmp/pipe2"  int main() { int fd1,
fd2;  char input[1000], output[1000]; //
Create the named pipes mkfifo(PIPE1,
0666); mkfifo(PIPE2, 0666); // Get  user
input printf("Enter a sentence (type 'exit'
to quit): "); fgets(input, sizeof(input),
stdin); if (strncmp(input, "exit", 4) == 0)
{ return 0; } //
Open PIPE1 for writing fd1 = open(PIPE1,
O_WRONLY); write(fd1, input, strlen(input)
+ 1); close(fd1); // Open PIPE2 for reading
fd2 = open(PIPE2, O_RDONLY); read(fd2,
output, sizeof(output)); close(fd2);
// Print the output received from the second process
printf("Output from second process:\n%s\n", output);  //
Remove the named pipes unlink(PIPE1); unlink(PIPE2);
return 0;}
```

**********7b.c**********

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#define PIPE1 "/tmp/pipe1" #define PIPE2 "/tmp/
pipe2" int countWords(char *str) { int count = 0;
char *token = strtok(str, " \n");while (token != NULL)
```

```
{  count++; token = strtok(NULL, " \n");
}  return
count;  } int
countLines(
char *str)
{  int count =
 0;
for (int i = 0; str[i] != '\0'; i++)
{  if (str[i] == '\n') {  count+
+; } }  return count;
} int main() { int fd1, fd2;
char
input[1000], output[1000]; int charCount,
wordCount, lineCount;
FILE *file;
// Open PIPE1 for reading fd1 = open(PIPE1,
O_RDONLY);
read(fd1, input, sizeof(input));
close(fd1);
// Count characters, words, and
lines charCount = strlen(input);
wordCount = countWords(input);
lineCount = countLines(input); //
Write the results to a file file =
fopen("output.txt",
"w");
fprintf(file, "Characters: %d\nWords: %d\nLines: %d\n", charCount, wordCount, lineCount);
fclose(file);
// Read the content of the file and send it through PIPE2 file
= fopen("output.txt", "r");
fread(output, sizeof(char), sizeof(output), file);
fclose(file);
// Open PIPE2 for writing  fd2 =
open(PIPE2, O_WRONLY);
write(fd2, output, strlen(output) +
1);  close(fd2);  return 0;
}
```

2 Program

**********7b1.c**********

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define SHM_KEY 12345
#define SHM_SIZE 1024
int main() { int shmid; char
*shmaddr;
// Locate the shared memory segment created by the server
shmid = shmget(SHM_KEY, SHM_SIZE,
0666); if (shmid < 0) { perror("shmget"); exit(1);
}
// Attach the shared memory segment to the client's address space
shmaddr = shmat(shmid, NULL,
0); if (shmaddr == (char *) -1)
{ perror("shmat");  exit(1);  }
// Read the message from the shared memory segment
printf("Reading from shared memory...\n"); printf("Message
from shared memory: %s\n", shmaddr);
// Detach the shared memory segment if
(shmdt(shmaddr) == -1) {perror("shmdt");
exit(1);
} return
0;
}
```

**********7b.c**********

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>
#define SHM_KEY 12345
#define SHM_SIZE 1024
int main() { int shmid; char
*shmaddr;
// Create a shared memory segment
shmid = shmget(SHM_KEY, SHM_SIZE, IPC_CREAT |
0666); if (shmid < 0) { perror("shmget"); exit(1);
}
// Attach the shared memory segment to the server's address space
shmaddr = shmat(shmid, NULL,
0); if (shmaddr == (char *) -1)
{ perror("shmat");  exit(1);  }
// Write a message to the shared memory segment
printf("Writing to shared memory...\n"); char *message =
"Hello from DVVPCOE,Ahmednagar Server!";
strncpy(shmaddr, message, SHM_SIZE); // Detach the
shared memory segment if
(shmdt(shmaddr) == -1)
{ perror("shmdt");  exit(1);  }
printf("Message written to shared memory: %s\n", message);
return 0; }
```

OUTPUT:

Assignment NO:08 PROGRAM:

```
#include <stdio.h> #include <stdlib.h>
int main() { int n, i, j, head,
total_movement = 0; printf("Enter the
number of requests:
");
scanf("%d", &n); int requests[n],  completed[n];
printf("Enter the request sequence: ");  for (i = 0; i < n; i+
+) {  scanf("%d", &requests[i]);  completed[i] = 0; // Mark
all requests as uncompleted initially
}
printf("Enter the initial head position: ");
scanf("%d", &head); for (i = 0; i < n; i++)
{ int min =  100000, min_index = -1; for (j =
0; j < n; j++) { if (!completed[j] && abs(head
- requests[j]) < min) {
min = abs(head - requests[j]); min_index = j;} }
completed[min_index] = 1; // Mark the request as completed
total_movement += abs(head - requests[min_index]); head
= requests[min_index]; printf("Serviced
request: %d\n", head);
} printf("Total head movement: %d\n",
total_movement);
return 0;
}
```

**********8b.c********** PROGRAM:

```
#include <stdio.h> #include <stdlib.h> int
main() { int n, i, head, total_movement = 0,
direction; printf("Enter the number of
requests:
");
scanf("%d", &n); int requests[n];
printf("Enter the request sequence: "); for
(i = 0; i < n; i++) { scanf("%d",
&requests[i]);
}
printf("Enter the initial head position: ");
scanf("%d", &head); printf("Enter the disk size
(last cylinder number): "); int disk_size;
scanf("%d", &disk_size); printf("Enter the
direction (1 for high, 0 for low): "); scanf("%d",
&direction);// Sort the request array  for (i = 0; i
< n - 1; i++) { for (int j = i + 1; j < n; j++) { if
(requests[i] > requests[j]) { int
temp = requests[i]; requests[i]
= requests[j]; requests[j] =
temp;
}
}
} // SCAN algorithm if (direction == 1) { //
Move towards higher end for (i = 0; i < n &&
requests[i] < head; i++); for (; i < n; i++)
{ printf("Serviced request: %d\n",
requests[i]); total_movement += abs(head -
requests[i]); head = requests[i]; } if (head <
disk_size - 1)
{ total_movement += abs(head - (disk_size -
1)); head = disk_size -
1;
}
for (i--; i >= 0; i--) { printf("Serviced
request: %d\n", requests[i]);
total_movement
+= abs(head - requests[i]);
head = requests[i];
}
} else { // Move towards lower end for (i = n -
1; i >= 0 && requests[i] > head; i--); for (; i >=
0; i--) { printf("Serviced request: %d\n",
requests[i]); total_movement += abs(head -
```

```
requests[i]);  head = requests[i]; }
if (head > 0)
{ total_movement += head;
head
=
0;  }
for (i++; i < n; i++) { printf("Serviced request: %d\n",
requests[i]); total_movement += abs(head -
requests[i]); head = requests[i]; }} printf("Total head
movement: %d\n", total_movement); return 0;
}
```

**********8c.c********** PROGRAM:

```
#include <stdio.h>
#include <stdlib.h> int
main() {
int n, i, head, total_movement = 0;
printf("Enter the number of requests:
");  scanf("%d", &n); int requests[n];
printf("Enter the request sequence:
"); for (i = 0; i < n; i++) { scanf("%d",
&requests[i]); }printf("Enter the initial
head position: ");
scanf("%d", &head); // Sort
the request array for (i = 0; i
< n - 1; i++) { for (int j = i +
1; j < n; j++) { if (requests[i] >
requests[j]) { int temp =
requests[i]; requests[i] =
requests[j];  requests[j] =
temp;
}
}
}
// C-LOOK algorithm for (i = 0; i < n &&
requests[i] < head; i++); for (; i < n; i++)
{ printf("Serviced request: %d\n", requests[i]);
total_movement += abs(head - requests[i]);
head = requests[i];
} if (i > 0)
{
total_movement += abs(head - requests[0]);
head = requests[0]; for (i = 1; i < n; i++)
{ printf("Serviced request: %d\n", requests[i]);
total_movement += abs(head - requests[i]);
head = requests[i];
}
} printf("Total head movement: %d\n",
total_movement); return 0;
}
```

OUTPUT

Assignment NO:09 PROGRAM:

```
#include <linux/kernel.h> asmlinkage
long sys_hello(void)
{
printk("Hello world\n");
return 0;
}
//Makefile objy
:= hello.o
//userspace.c
#include
<stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h> int
main()
{ long int amma =
syscall(354);
printf("System call sys_hello returned %ld\n", amma);
return 0;
}
```

OUTPUT :

```
Hello world
```