1) Write a program to implement an address book with options given below: a) Create address book. b) View address book. c) Insert a record. d) Delete a record. e) Modify a record. f) Exit

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

#define MAX 100

struct Address {
   char name[50];
   char phone[15];
   char email[50];
};

struct Address book[MAX];
int count = 0;

void createBook() {
   count = 0;
   printf("\nAddress book created successfully!\n");
}

void viewBook() {
   if (count == 0) {
      printf("\nAddress book is empty.\n");
      return;
   }
   printf("\n------ Address Book ------\n");
   for (int i = 0; i < count; i++) {
      printf("\nRecord %d\n", i + 1);
      printf("Name: %s\n", book[i].name);
      printf("Phone: %s\n", book[i].phone);
      printf("Email: %s\n", book[i].email);
   }
}

void insertRecord() {
   if (count >= MAX) {
      printf("\nAddress book is full!\n");
      return;
   }
   printf("\nEnter name: ");
   scanf(" %[^\n]", book[count].name);
   printf("Enter phone: ");
   scanf(" %[^\n]", book[count].phone);
   printf("Enter email: ");
   scanf(" %[^\n]", book[count].email);
   count++;
   printf("\nRecord inserted successfully!\n");
}

void deleteRecord() {
   char name[50];
   printf("\nEnter name to delete: ");
   scanf(" %[^\n]", name);

   int found = 0;
   for (int i = 0; i < count; i++) {
      if (strcmp(book[i].name, name) == 0) {
         found = 1;
         for (int j = i; j < count - 1; j++) {
            book[j] = book[j + 1];
         }
         count--;
         printf("\nRecord deleted successfully!\n");
         break;
      }
   }
   if (!found)
      printf("\nRecord not found!\n");
}
```

```
void modifyRecord() {
    char name[50];
    printf("\nEnter name to modify: ");
    scanf(" %[^\n]", name);

    int found = 0;
    for (int i = 0; i < count; i++) {
        if (strcmp(book[i].name, name) == 0) {
            found = 1;
            printf("\nEnter new name: ");
            scanf(" %[^\n]", book[i].name);
            printf("Enter new phone: ");
            scanf(" %[^\n]", book[i].phone);
            printf("Enter new email: ");
            scanf(" %[^\n]", book[i].email);
            printf("\nRecord modified successfully!\n");
            break;
        }
    }
    if (!found)
        printf("\nRecord not found!\n");
}

int main() {
    int choice;
    while (1) {
        printf("\n===== ADDRESS BOOK MENU =====");
        printf("\n1. Create Address Book");
        printf("\n2. View Address Book");
        printf("\n3. Insert Record");
        printf("\n4. Delete Record");
        printf("\n5. Modify Record");
        printf("\n6. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: createBook(); break;
            case 2: viewBook(); break;
            case 3: insertRecord(); break;
            case 4: deleteRecord(); break;
            case 5: modifyRecord(); break;
            case 6: exit(0);
            default: printf("\nInvalid choice! Try again.\n");
        }
    }
    return 0;
}
```

Run

Create file touch filename.c
gcc address_book.c
./a.out
or
gcc address_book.c -o address_book
./address_book / ./a.out

2) A. Implement the C program in which main program accepts the integers to be sorted. Main program uses the FORK system call to create a new process called a child process. Parent process sorts the integers using sorting algorithm and waits for child process using WAIT system call to sort the integers using any sorting algorithm. Also demonstrate zombie and orphan states.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>   // for fork(), getpid(), getppid()
#include <sys/wait.h> // for wait()

// Function for Bubble Sort
void bubbleSort(int arr[], int n) {
   for (int i = 0; i < n - 1; i++) {
      for (int j = 0; j < n - i - 1; j++) {
         if (arr[j] > arr[j + 1]) {
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}

// Function to print an array
void printArray(int arr[], int n) {
   for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");
}

int main() {
   int n, i;
   printf("Enter number of integers: ");
   scanf("%d", &n);

   int arr[n];
   printf("Enter %d integers:\n", n);
   for (i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }

   pid_t pid = fork(); // Create a new process

   if (pid < 0) {
      printf("Fork failed!\n");
      exit(1);
   }
   else if (pid == 0) {
      // Child process
      printf("\n[Child] PID: %d, PPID: %d\n", getpid(), getppid());
      printf("[Child] Sorting numbers (using Selection Sort)...\n");

      // Selection Sort
      for (i = 0; i < n - 1; i++) {
         int min = i;
         for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[min])
               min = j;
         }
         int temp = arr[min];
         arr[min] = arr[i];
         arr[i] = temp;
      }

      printf("[Child] Sorted array: ");
      printArray(arr, n);

      printf("[Child] Sleeping for 5 seconds to create ORPHAN state...\n");
      sleep(5);
      printf("[Child] Parent PID after 5 sec: %d (should be 1 -> orphan adopted by init)\n", getppid());
      printf("[Child] Exiting now.\n\n");
      exit(0);
```

```
    }
    else {
        // Parent process
        printf("\n[Parent] PID: %d, Child PID: %d\n", getpid(), pid);
        printf("[Parent] Sorting numbers (using Bubble Sort)...\n");
        bubbleSort(arr, n);
        printf("[Parent] Sorted array: ");
        printArray(arr, n);

        printf("[Parent] Sleeping for 10 seconds to create ZOMBIE state...\n");
        sleep(10);

        int status;
        wait(&status); // Wait for child to finish
        printf("[Parent] Child process finished. (Zombie cleared)\n");
        printf("[Parent] Exiting now.\n");
    }

    return 0;
}
```

Run

Touch filename.c

gcc fork_sort.c -o fork_sort

./fork_sort

3) B. Implement the C program in which main program accepts an array. Main program uses the FORK system call to create a new process called a child process. Parent process sorts an array and passes the sorted array to child process through the command line arguments of EXECVE system call. The child process uses EXECVE system call to load new program which display array in reverse order.

Program 1 touch parent_sort.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>

// Bubble sort function
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    bubbleSort(arr, n);

    printf("\n[Parent] Sorted Array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    pid_t pid = fork();

    if (pid < 0) {
```

```
        perror("Fork failed");
        exit(1);
    }

    if (pid == 0) {
        // CHILD PROCESS
        printf("\n[Child] Executing new program using execve()...\n");

        char *args[n + 2];
        args[0] = "./reverse_display";  // Program to execute
        for (int i = 0; i < n; i++) {
            char *num = malloc(10);
            sprintf(num, "%d", arr[i]);  // Convert int → string
            args[i + 1] = num;
        }
        args[n + 1] = NULL;  // Null-terminate argument list

        execve(args[0], args, NULL);

        perror("execve failed");
        exit(1);
    } else {
        // PARENT PROCESS
        wait(NULL);
        printf("\n[Parent] Child process completed successfully.\n");
    }

    return 0;
}
```

Program 2 touch reverse_display.c

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

int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("No numbers received!\n");
        return 1;
    }

    printf("\n[Child Program] Displaying array in reverse order:\n");
    for (int i = argc - 1; i > 0; i--) {
        printf("%s ", argv[i]);
    }
    printf("\n\n");

    return 0;
}
```

Run

```
gcc parent_sort.c -o parent_sort
gcc reverse_display.c -o reverse_display
./parent_sort
```

4) Implement the C program for CPU Scheduling Algorithms: Shortest Job First (Preemptive) and Round Robin with different arrival time.

touch cpu_scheduling.c

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

struct Process {
    int pid;
    int arrival;
    int burst;
    int remaining;
    int waiting;
    int turnaround;
    int completed;
};

// Function for Shortest Job First (Preemptive / SRTF)
void SJF_Preemptive(struct Process p[], int n) {
    int complete = 0, t = 0, minm = 1e9;
    int shortest = 0, finish_time;
```

```
    float total_wait = 0, total_tat = 0;

    for (int i = 0; i < n; i++)
        p[i].remaining = p[i].burst;

    printf("\n===== Shortest Job First (Preemptive) =====\n");

    while (complete != n) {
        shortest = -1;
        minm = 1e9;

        for (int i = 0; i < n; i++) {
            if ((p[i].arrival <= t) && (p[i].completed == 0) && (p[i].remaining < minm) && p[i].remaining > 0) {
                minm = p[i].remaining;
                shortest = i;
            }
        }

        if (shortest == -1) {
            t++;
            continue;
        }

        p[shortest].remaining--;

        if (p[shortest].remaining == 0) {
            complete++;
            p[shortest].completed = 1;
            finish_time = t + 1;
            p[shortest].turnaround = finish_time - p[shortest].arrival;
            p[shortest].waiting = p[shortest].turnaround - p[shortest].burst;
            if (p[shortest].waiting < 0)
                p[shortest].waiting = 0;
        }

        t++;
    }

    printf("\nPID\tAT\tBT\tWT\tTAT\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrival, p[i].burst, p[i].waiting, p[i].turnaround);
        total_wait += p[i].waiting;
        total_tat += p[i].turnaround;
    }

    printf("\nAverage Waiting Time: %.2f", total_wait / n);
    printf("\nAverage Turnaround Time: %.2f\n", total_tat / n);
}

// Function for Round Robin Scheduling
void RoundRobin(struct Process p[], int n, int quantum) {
    int t = 0, done;
    float total_wait = 0, total_tat = 0;
    int rem[n];

    for (int i = 0; i < n; i++) {
        rem[i] = p[i].burst;
        p[i].waiting = 0;
        p[i].turnaround = 0;
    }

    printf("\n===== Round Robin Scheduling =====\n");

    while (1) {
        done = 1;
        for (int i = 0; i < n; i++) {
            if (rem[i] > 0 && p[i].arrival <= t) {
                done = 0;
                if (rem[i] > quantum) {
                    t += quantum;
                    rem[i] -= quantum;
```

```
            } else {
                t += rem[i];
                p[i].waiting = t - p[i].burst - p[i].arrival;
                p[i].turnaround = t - p[i].arrival;
                rem[i] = 0;
            }
          }
        }
        if (done == 1)
          break;
        t++;
    }

    printf("\nPID\tAT\tBT\tWT\tTAT\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrival, p[i].burst, p[i].waiting, p[i].turnaround);
        total_wait += p[i].waiting;
        total_tat += p[i].turnaround;
    }

    printf("\nAverage Waiting Time: %.2f", total_wait / n);
    printf("\nAverage Turnaround Time: %.2f\n", total_tat / n);
}

int main() {
    int n, quantum;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process p[n];
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("\nEnter Arrival Time for P%d: ", i + 1);
        scanf("%d", &p[i].arrival);
        printf("Enter Burst Time for P%d: ", i + 1);
        scanf("%d", &p[i].burst);
        p[i].completed = 0;
    }

    // Run SJF Preemptive
    SJF_Preemptive(p, n);

    // Reset completed flags for Round Robin
    for (int i = 0; i < n; i++)
        p[i].completed = 0;

    printf("\nEnter Time Quantum for Round Robin: ");
    scanf("%d", &quantum);

    // Run Round Robin
    RoundRobin(p, n, quantum);

    return 0;
}
gcc cpu_scheduling.c
./a.out
or
gcc cpu_scheduling.c -o cpu_scheduling
./cpu_scheduling
```

5) Thread synchronization using counting semaphores. Application to demonstrate: producerconsumer problem with counting semaphores and mutex.

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

#define BUFFER_SIZE 5

int buffer[BUFFER_SIZE];
int in = 0, out = 0;
```

```
// Semaphores and Mutex
sem_t empty;
sem_t full;
pthread_mutex_t mutex;

// Producer function
void* producer(void* arg) {
  int item;
  for (int i = 1; i <= 10; i++) {
    item = rand() % 100;  // produce item
    sem_wait(&empty);     // decrease empty count
    pthread_mutex_lock(&mutex); // lock critical section

    buffer[in] = item;
    printf("Producer produced: %d\n", item);
    in = (in + 1) % BUFFER_SIZE;

    pthread_mutex_unlock(&mutex); // unlock critical section
    sem_post(&full);              // increase full count
    sleep(1);
  }
  return NULL;
}

// Consumer function
void* consumer(void* arg) {
  int item;
  for (int i = 1; i <= 10; i++) {
    sem_wait(&full);              // wait if buffer empty
    pthread_mutex_lock(&mutex);   // lock critical section

    item = buffer[out];
    printf("Consumer consumed: %d\n", item);
    out = (out + 1) % BUFFER_SIZE;

    pthread_mutex_unlock(&mutex); // unlock
    sem_post(&empty);             // increase empty slots
    sleep(2);
  }
  return NULL;
}

int main() {
  pthread_t prodThread, consThread;

  // Initialize semaphores and mutex
  sem_init(&empty, 0, BUFFER_SIZE);
  sem_init(&full, 0, 0);
  pthread_mutex_init(&mutex, NULL);

  // Create threads
  pthread_create(&prodThread, NULL, producer, NULL);
  pthread_create(&consThread, NULL, consumer, NULL);

  // Wait for threads to finish
  pthread_join(prodThread, NULL);
  pthread_join(consThread, NULL);

  // Destroy semaphores and mutex
  sem_destroy(&empty);
  sem_destroy(&full);
  pthread_mutex_destroy(&mutex);

  printf("\nExecution completed successfully.\n");
  return 0;
}
gcc producer_consumer.c –pthread
./a.out
```

6) **Thread synchronization and mutual exclusion using mutex. Application to demonstrate: ReaderWriter problem with reader priority.**
Touch reader_writer.c

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

pthread_mutex_t mutex;     // For mutual exclusion of readcount
pthread_mutex_t wrt;       // For writer lock
int readcount = 0;         // Number of readers currently reading
int data = 0;              // Shared resource

// Reader function
void* reader(void* arg) {
   int id = *(int*)arg;
   while (1) {
      pthread_mutex_lock(&mutex);
      readcount++;
      if (readcount == 1)
         pthread_mutex_lock(&wrt); // first reader locks writer
      pthread_mutex_unlock(&mutex);

      // Reading section
      printf("Reader %d is reading data = %d\n", id, data);
      sleep(1);

      pthread_mutex_lock(&mutex);
      readcount--;
      if (readcount == 0)
         pthread_mutex_unlock(&wrt); // last reader unlocks writer
      pthread_mutex_unlock(&mutex);

      sleep(1);
   }
}

// Writer function
void* writer(void* arg) {
   int id = *(int*)arg;
   while (1) {
      pthread_mutex_lock(&wrt); // writer gets access
      data++;
      printf("Writer %d modified data to %d\n", id, data);
      pthread_mutex_unlock(&wrt);
      sleep(2);
   }
}

int main() {
   pthread_t rtid[3], wtid[2];
   int rid[3] = {1, 2, 3};
   int wid[2] = {1, 2};

   pthread_mutex_init(&mutex, NULL);
   pthread_mutex_init(&wrt, NULL);

   // Create reader threads
   for (int i = 0; i < 3; i++)
      pthread_create(&rtid[i], NULL, reader, &rid[i]);

   // Create writer threads
   for (int i = 0; i < 2; i++)
      pthread_create(&wtid[i], NULL, writer, &wid[i]);

   // Wait (join) threads — this program runs continuously, but join keeps main alive
   for (int i = 0; i < 3; i++)
      pthread_join(rtid[i], NULL);
   for (int i = 0; i < 2; i++)
      pthread_join(wtid[i], NULL);

   pthread_mutex_destroy(&mutex);
   pthread_mutex_destroy(&wrt);
```

```
    return 0;
}
gcc reader_writer.c –pthread
./reader_writer
./a.out
```

7) Implement the C program for Deadlock Avoidance Algorithm: Banke Algorithmrs

```
#include <stdio.h>

int main() {
    int n, m; // n = number of processes, m = number of resources
    printf("Enter number of processes: ");
    scanf("%d", &n);
    printf("Enter number of resources: ");
    scanf("%d", &m);

    int alloc[n][m], max[n][m], avail[m];
    int need[n][m], finish[n], safeSeq[n];
    int count = 0;

    printf("\nEnter Allocation Matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &alloc[i][j]);

    printf("\nEnter Max Matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &max[i][j]);

    printf("\nEnter Available Resources:\n");
    for (int i = 0; i < m; i++)
        scanf("%d", &avail[i]);

    // Calculate Need Matrix
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];

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

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

                if (j == m) {
                    for (int k = 0; k < m; k++)
                        avail[k] += alloc[i][k];
                    safeSeq[count++] = i;
                    finish[i] = 1;
                    found = 1;
                }
            }
        }
        if (found == 0) {
            printf("\nSystem is not in a safe state (DEADLOCK may occur).\n");
            return 0;
        }
    }

    printf("\nSystem is in a SAFE state.\nSafe Sequence: ");
    for (int i = 0; i < n; i++)
        printf("P%d ", safeSeq[i]);
    printf("\n");
```

```
    return 0;
}
```

```
gcc bankers_algorithm.c -o bankers
./bankers.c
```

8) Implement the C program for Page Replacement Algorithms: FCFS, LRU, andOptimal for frame size as minimum three.

Touch page_replacement.c

```
#include <stdio.h>

#define MAX 20

// Function to check if page is present in frame
int isPresent(int frames[], int n, int page) {
    for (int i = 0; i < n; i++) {
        if (frames[i] == page)
            return 1;
    }
    return 0;
}

// Function to print current frame contents
void printFrames(int frames[], int n) {
    for (int i = 0; i < n; i++) {
        if (frames[i] != -1)
            printf("%d ", frames[i]);
    }
}

// FIFO Page Replacement
void fifo(int pages[], int n, int frameSize) {
    int frames[MAX], faults = 0, index = 0;

    for (int i = 0; i < frameSize; i++) frames[i] = -1; // initialize empty frames

    printf("\nFIFO Page Replacement\n");
    printf("Page\tFrames\t\tFault\n");

    for (int i = 0; i < n; i++) {
        if (!isPresent(frames, frameSize, pages[i])) { // page fault
            frames[index] = pages[i];
            index = (index + 1) % frameSize; // circular replacement
            faults++;
            printf("%d\t", pages[i]);
            printFrames(frames, frameSize);
            printf("\tF\n");
        } else {
            printf("%d\t", pages[i]);
            printFrames(frames, frameSize);
            printf("\tH\n");
        }
    }
    printf("Total Page Faults = %d\n", faults);
}

// Optimal Page Replacement
void optimal(int pages[], int n, int frameSize) {
    int frames[MAX], faults = 0;

    for (int i = 0; i < frameSize; i++) frames[i] = -1;

    printf("\nOptimal Page Replacement\n");
    printf("Page\tFrames\t\tFault\n");

    for (int i = 0; i < n; i++) {
        if (!isPresent(frames, frameSize, pages[i])) {
            int replaceIndex = -1, farthest = i;
            for (int j = 0; j < frameSize; j++) {
                if (frames[j] == -1) { replaceIndex = j; break; }
                int k;
                for (k = i + 1; k < n; k++) {
                    if (frames[j] == pages[k]) break;
                }
```

```
            if (k > farthest) {
              farthest = k;
              replaceIndex = j;
            }
          }
          frames[replaceIndex] = pages[i];
          faults++;
          printf("%d\t", pages[i]);
          printFrames(frames, frameSize);
          printf("\tF\n");
        } else {
          printf("%d\t", pages[i]);
          printFrames(frames, frameSize);
          printf("\tH\n");
        }
    }
    printf("Total Page Faults = %d\n", faults);
}

// LRU Page Replacement
void lru(int pages[], int n, int frameSize) {
    int frames[MAX], faults = 0, recent[MAX];

    for (int i = 0; i < frameSize; i++) {
        frames[i] = -1;
        recent[i] = -1;
    }

    printf("\nLRU Page Replacement\n");
    printf("Page\tFrames\t\tFault\n");

    for (int i = 0; i < n; i++) {
        if (!isPresent(frames, frameSize, pages[i])) {
            int replaceIndex = -1, leastRecent = i;
            for (int j = 0; j < frameSize; j++) {
                if (frames[j] == -1) { replaceIndex = j; break; }
                int k;
                for (k = i - 1; k >= 0; k--) {
                    if (frames[j] == pages[k]) break;
                }
                if (k < leastRecent) {
                    leastRecent = k;
                    replaceIndex = j;
                }
            }
            frames[replaceIndex] = pages[i];
            faults++;
            printf("%d\t", pages[i]);
            printFrames(frames, frameSize);
            printf("\tF\n");
        } else {
            printf("%d\t", pages[i]);
            printFrames(frames, frameSize);
            printf("\tH\n");
        }
    }
    printf("Total Page Faults = %d\n", faults);
}

int main() {
    int n, frameSize, choice;
    int pages[MAX];

    printf("Enter number of pages in reference string: ");
    scanf("%d", &n);

    printf("Enter the reference string (space separated): ");
    for (int i = 0; i < n; i++)
        scanf("%d", &pages[i]);

    printf("Enter frame size: ");
```

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

    do {
        printf("\nMENU:\n1. FIFO\n2. LRU\n3. Optimal\n4. Exit\nEnter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: fifo(pages, n, frameSize); break;
            case 2: lru(pages, n, frameSize); break;
            case 3: optimal(pages, n, frameSize); break;
            case 4: printf("Exiting...\n"); break;
            default: printf("Invalid choice!\n");
        }
    } while (choice != 4);

    return 0;
}
```

Run

```
gcc page_replacement.c -o page_replacement
./a.out
```

9) Inter process communication in Linux using following. A. FIFOS: Full duplex communication between two independent processes. First process accepts sentences and writes on one pipe to be read by second process and second process counts number of characters, number of words and number of lines in accepted sentences, writes this output in a text file and writes the contents of the file on second pipe to be read by first process and displays onstandard output.

Process1 touch process1.c

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

#define FIFO1 "fifo1"
#define FIFO2 "fifo2"

int main() {
    char sentence[200], result[200];
    int fd1, fd2;

    mkfifo(FIFO1, 0666);
    mkfifo(FIFO2, 0666);

    while (1) {
        printf("\nEnter a sentence (or type 'exit' to quit): ");
        fgets(sentence, sizeof(sentence), stdin);

        if (strncmp(sentence, "exit", 4) == 0) {
            printf("Exiting...\n");
            break;
        }

        // Write the sentence to FIFO1
        fd1 = open(FIFO1, O_WRONLY);
        write(fd1, sentence, strlen(sentence) + 1);
        close(fd1);

        // Read processed result from FIFO2
        fd2 = open(FIFO2, O_RDONLY);
        read(fd2, result, sizeof(result));
        close(fd2);

        printf("Received from Process2:\n%s\n", result);
    }

    unlink(FIFO1);
    unlink(FIFO2);
    return 0;
}
```

Process 2 touch process2.c

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

```
#include <fcntl.h>

#define FIFO1 "fifo1"
#define FIFO2 "fifo2"

int main() {
   char sentence[200], result[200];
   FILE *fp;
   int fd1, fd2;

   mkfifo(FIFO1, 0666);
   mkfifo(FIFO2, 0666);

   while (1) {
      // Read sentence from FIFO1
      fd1 = open(FIFO1, O_RDONLY);
      read(fd1, sentence, sizeof(sentence));
      close(fd1);

      if (strncmp(sentence, "exit", 4) == 0) break;

      int chars = 0, words = 0, lines = 0;
      int inWord = 0;
      for (int i = 0; sentence[i] != '\0'; i++) {
         char c = sentence[i];
         chars++;
         if (c == '\n')
            lines++;
         if (c == ' ' || c == '\n' || c == '\t')
            inWord = 0;
         else if (!inWord) {
            inWord = 1;
            words++;
         }
      }
      if (lines == 0) lines = 1; // Single line input

      // Write results to a file
      fp = fopen("output.txt", "w");
      fprintf(fp, "Characters: %d\nWords: %d\nLines: %d\n", chars, words, lines);
      fclose(fp);

      // Read from file and send through FIFO2
      fp = fopen("output.txt", "r");
      fread(result, sizeof(char), sizeof(result), fp);
      fclose(fp);

      fd2 = open(FIFO2, O_WRONLY);
      write(fd2, result, strlen(result) + 1);
      close(fd2);
   }

   unlink(FIFO1);
   unlink(FIFO2);
   return 0;
}

gcc process2.c -o process2
./process1
./process2
```

10) Inter-process Communication using Shared Memory using System V. Application to demonstrate: Client and Server Programs in which server process creates a shared memory segment and writes the message to the shared memory segment. Client process reads the message from the shared memory segment and displays it to the screen

Touch server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>
```

```
#define SHM_KEY 1234  // Unique key for shared memory

int main() {
   int shmid;
   char *shared_memory;

   // Create shared memory segment of 1024 bytes
   shmid = shmget(SHM_KEY, 1024, 0666 | IPC_CREAT);
   if (shmid == -1) {
      perror("shmget");
      exit(1);
   }

   // Attach to the shared memory
   shared_memory = (char*) shmat(shmid, NULL, 0);
   if (shared_memory == (char*)(-1)) {
      perror("shmat");
      exit(1);
   }

   // Write message
   printf("Enter message to write to shared memory: ");
   fgets(shared_memory, 1024, stdin);

   printf("Server: Message written to shared memory.\n");

   // Detach from shared memory
   shmdt(shared_memory);

   return 0;
}
```

Touch client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHM_KEY 1234  // Must be same as in server

int main() {
   int shmid;
   char *shared_memory;

   // Get the shared memory segment created by server
   shmid = shmget(SHM_KEY, 1024, 0666);
   if (shmid == -1) {
      perror("shmget");
      exit(1);
   }

   // Attach to the shared memory
   shared_memory = (char*) shmat(shmid, NULL, 0);
   if (shared_memory == (char*)(-1)) {
      perror("shmat");
      exit(1);
   }

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

   // Detach from shared memory
   shmdt(shared_memory);

   // Delete the shared memory segment
   shmctl(shmid, IPC_RMID, NULL);

   return 0;
}
gcc server.c -o server
gcc client.c -o client
./server.c
```

./client.c

11 Disk Scheduling

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

int SSTF();
int SCAN();
int CLOOK();
int main()
{


   int ch, YN = 1, i;
   //char F[10], s[25];

   do{
      //system("clear");
      printf("\n\n\t***** MENU *****");
      printf("\n\n\t1:SSTF\n\n\t2:SCAN\n\n\t3:CLOOK\n\n\t4:EXIT");
      printf("\n\n\tEnter your choice: ");
      scanf("%d", &ch);

      switch (ch)
      {
         case 1:

         SSTF();
         break;
         case 2:

         SCAN();
         break;
         case 3:

         CLOOK();
         break;
         case 4:
         exit(0);
      }
      printf("\n\n\tDo u want to continue IF YES PRESS 1\n\n\tIF NO PRESS 0: ");


      scanf("%d", &YN);
   }
   while (YN == 1);
   return (0);
}

//SSTF Algorithm

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

   while (count != n)
   {
      int min = 1000, d, index;
      for (i = 0; i < n; i++)
      {
         d = abs(RQ[i] - initial);
         if (min > d)
```

```
        {
            min = d;
            index = i;
        }
    }
    TotalHeadMoment = TotalHeadMoment + min;
    initial = RQ[index];


    RQ[index] = 1000;
    count++;
  }
  printf("Total head movement is %d", TotalHeadMoment);
  return 0;
}


//SCAN Algorithm

int SCAN(){
    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);
    for (i = 0; i < n; i++){
      for (j = 0; j < n - i - 1; j++){
        if (RQ[j] > RQ[j + 1]){
          int temp;
          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 (move == 1){
      for (i = index; i < n; i++){
        TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
        initial = RQ[i];
      }
      TotalHeadMoment = TotalHeadMoment + abs(size - RQ[i - 1] - 1);
      initial = size - 1;
      for (i = index - 1; i >= 0; i--){
        TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
        initial = RQ[i];
      }
    }
    else{
      for (i = index - 1; i >= 0; i--) {
        TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
        initial = RQ[i];
      }
      TotalHeadMoment = TotalHeadMoment + abs(RQ[i + 1] - 0);
      initial = 0;
      for (i = index; i < n; i++){
        TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
```

```
            initial = RQ[i];
          }
      }
      printf("Total head movement is %d", TotalHeadMoment);
      return 0;
   }


//C-LOOK Algorithm

int CLOOK(){
   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);

   for (i = 0; i < n; i++){
      for (j = 0; j < n - i - 1; j++){
         if (RQ[j] > RQ[j + 1]){
            int temp;
            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 (move == 1){
      for (i = index; i < n; i++){
      TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
      initial = RQ[i];
      }
      for (i = 0; i < index; i++){
         TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
         initial = RQ[i];
      }
   }
   else{
      for (i = index - 1; i >= 0; i--){
         TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
         initial = RQ[i];
      }
      for (i = n - 1; i >= index; i--){
         TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
         initial = RQ[i];
      }
   }
   printf("Total head movement is %d", TotalHeadMoment);
   return 0;
}
```

Run touch filemname.c

```
gcc filename.c
./a.out
```