1. Write a program to use fork system call to create 5 child processes and assign 5 operations to childs.

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

void child_operation(int index) {
        switch (index) {
        case 1:
        printf("Child %d: I am performing addition.\n", getpid());
        // Perform addition operation
        break;
        case 2:
        printf("Child %d: I am performing subtraction.\n", getpid());
        // Perform subtraction operation
        break;
        case 3:
        printf("Child %d: I am performing multiplication.\n", getpid());
        // Perform multiplication operation
        break;
        case 4:
        printf("Child %d: I am performing division.\n", getpid());
        // Perform division operation
        break;
        case 5:
        printf("Child %d: I am performing modulus.\n", getpid());
        // Perform modulus operation
        break;
        default:
        printf("Invalid index.\n");
        exit(EXIT_FAILURE);
        }
        exit(EXIT_SUCCESS);
}

int main() {
        int num_processes = 5;

        for (int i = 1; i <= num_processes; i++) {
```

```
    pid_t pid = fork();
    if (pid == -1) {
    perror("fork failed");
    exit(EXIT_FAILURE);
    } else if (pid == 0) {
    // This is the child process
    child_operation(i);
    }
    }

    // Parent process waits for all child processes to finish
    for (int i = 0; i < num_processes; i++) {
    wait(NULL);
    }

    return 0;
}
```

2. Write a program to use vfork system call(login name by child and password by parent)

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

int main() {
    pid_t pid;
    int status;

    pid = vfork(); // Creating a child process using vfork

    if (pid == -1) {
    perror("vfork failed");
    exit(EXIT_FAILURE);
    } else if (pid == 0) {
    // Child process
    struct passwd *pw;
    pw = getpwuid(getuid());
    if (pw == NULL) {
    perror("getpwuid failed");
    exit(EXIT_FAILURE);
    }
    printf("Child: Login name: %s\n", pw->pw_name);
    exit(EXIT_SUCCESS);
    } else {
    // Parent process
    wait(&status); // Wait for the child to finish
    printf("Parent: Please enter your password: ");
    char password[100];
    scanf("%s", password);
    printf("Parent: Password entered: %s\n", password);
    }

    return 0;
}
```

3. Write a program to open any application using fork sysem call.

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

int main() {
    pid_t pid;

    pid = fork(); // Creating a child process using fork

    if (pid == -1) {
    perror("fork failed");
    exit(EXIT_FAILURE);
    } else if (pid == 0) {
    // Child process
    printf("Child: Opening application...\n");
    // Use execl to replace the child process with the desired application
    execl("/usr/bin/gedit", "gedit", NULL);
    // If execl returns, it means there was an error
    perror("execl failed");
    exit(EXIT_FAILURE);
    } else {
```

```
    // Parent process
    printf("Parent: Child process ID: %d\n", pid);
    printf("Parent: Application opened.\n");
    }

    return 0;
}
```

4. Write a program to open any application using vfork sysem call.

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

int main() {
    pid_t pid;

    pid = vfork(); // Creating a child process using vfork

    if (pid == -1) {
    perror("vfork failed");
    exit(EXIT_FAILURE);
    } else if (pid == 0) {
    // Child process
    printf("Child: Opening application...\n");
    // Use execl to replace the child process with the desired application
    execl("/usr/bin/gedit", "gedit", NULL);
    // If execl returns, it means there was an error
    perror("execl failed");
    _exit(EXIT_FAILURE);
    } else {
    // Parent process
    printf("Parent: Child process ID: %d\n", pid);
    printf("Parent: Application opened.\n");
    }

    return 0;
}
```

5. Write a program to demonstrate the wait use with fork sysem call.

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

int main() {
        pid_t pid;
        int status;

        printf("Parent: Before forking\n");

        pid = fork(); // Creating a child process using fork

        if (pid == -1) {
        perror("fork failed");
        exit(EXIT_FAILURE);
        } else if (pid == 0) {
        // Child process
        printf("Child: I am the child process (PID: %d)\n", getpid());
        printf("Child: Sleeping for 3 seconds...\n");
        sleep(3);
        printf("Child: Exiting\n");
        exit(EXIT_SUCCESS);
        } else {
        // Parent process
        printf("Parent: I am the parent process (PID: %d)\n", getpid());
        printf("Parent: Waiting for child to finish...\n");
        wait(&status); // Parent waits for the child process to finish
        printf("Parent: Child process finished\n");
        }

        return 0;
}
```

6. Write a program to demonstrate the variations exec system call.

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

int main() {
        pid_t pid;
        int status;

        // Using execl
        pid = fork(); // Create a child process
        if (pid == 0) { // If this is the child process
        printf("Child process using execl:\n");
        execl("/bin/ls", "ls", "-l", NULL); // Execute ls -l command
        perror("execl"); // Print error if execl fails
        _exit(1); // Terminate child process
        }
        waitpid(pid, &status, 0); // Wait for child process to finish

        // Using execlp
        pid = fork(); // Create a child process
        if (pid == 0) { // If this is the child process
        printf("\nChild process using execlp:\n");
        execlp("ls", "ls", "-l", NULL); // Execute ls -l command using PATH
        perror("execlp"); // Print error if execlp fails
        _exit(1); // Terminate child process
        }
        waitpid(pid, &status, 0); // Wait for child process to finish

        // Using execle
        pid = fork(); // Create a child process
        if (pid == 0) { // If this is the child process
        printf("\nChild process using execle:\n");
        char *envp[] = {"PATH=/bin", NULL}; // Define environment variable
        execle("/bin/ls", "ls", "-l", NULL, envp); // Execute ls -l command with custom environment
        perror("execle"); // Print error if execle fails
        _exit(1); // Terminate child process
        }
```

```
    waitpid(pid, &status, 0); // Wait for child process to finish

    // Using execv
    pid = fork(); // Create a child process
    if (pid == 0) { // If this is the child process
    printf("\nChild process using execv:\n");
    char *args[] = {"ls", "-l", NULL}; // Arguments array
    execv("/bin/ls", args); // Execute ls -l command with arguments
    perror("execv"); // Print error if execv fails
    _exit(1); // Terminate child process
    }
    waitpid(pid, &status, 0); // Wait for child process to finish

    // Using execvp
    pid = fork(); // Create a child process
    if (pid == 0) { // If this is the child process
    printf("\nChild process using execvp:\n");
    char *args[] = {"ls", "-l", NULL}; // Arguments array
    execvp("ls", args); // Execute ls -l command using PATH and arguments
    perror("execvp"); // Print error if execvp fails
    _exit(1); // Terminate child process
    }
    waitpid(pid, &status, 0); // Wait for child process to finish

    printf("\nParent process done.\n");

    return 0; // Exit parent process
}
```

7.Write a program to demonstrate the exit system call use with wait & fork sysem call.

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

```
int main() {
    pid_t pid;
    int status;

    printf("Parent: Before forking\n");

    pid = fork(); // Creating a child process using fork

    if (pid == -1) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process
        printf("Child: I am the child process (PID: %d)\n", getpid());
        printf("Child: Exiting with status 42\n");
        exit(42); // Child process exits with status 42
    } else {
        // Parent process
        printf("Parent: I am the parent process (PID: %d)\n", getpid());
        printf("Parent: Waiting for child to finish...\n");
        wait(&status); // Parent waits for the child process to finish
        if (WIFEXITED(status)) {
            printf("Parent: Child process exited with status: %d\n", WEXITSTATUS(status));
        } else {
            printf("Parent: Child process exited abnormally\n");
        }
    }

    return 0;
}
```

8. Write a program to demonstrate the kill system call to send signals between unrelated processes

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

void signal_handler(int signum) {
        printf("Signal %d received\n", signum);
}

int main() {
        pid_t pid;

        pid = fork(); // Creating a child process using fork

        if (pid == -1) {
        perror("fork failed");
        exit(EXIT_FAILURE);
        } else if (pid == 0) {
        // Child process
        printf("Child: I am the child process (PID: %d)\n", getpid());
        // Register signal handler for SIGUSR1
        signal(SIGUSR1, signal_handler);
        printf("Child: Waiting for signal from parent...\n");
        while(1); // Child process waits indefinitely
        } else {
        // Parent process
        printf("Parent: I am the parent process (PID: %d)\n", getpid());
        sleep(1); // Parent process sleeps for a second to ensure child process starts
        printf("Parent: Sending signal to child...\n");
        // Send SIGUSR1 signal to the child process
        kill(pid, SIGUSR1);
        }

        return 0;
}
```

9.Write a program to demonstrate the kill system call to send signals between related processes(fork)

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

void signal_handler(int signum) {
    printf("Child: Signal %d received\n", signum);
}
```

```
int main() {
        pid_t pid;

        pid = fork(); // Creating a child process using fork

        if (pid == -1) {
        perror("fork failed");
        exit(EXIT_FAILURE);
        } else if (pid == 0) {
        // Child process
        printf("Child: I am the child process (PID: %d)\n", getpid());
        // Register signal handler for SIGUSR1
        signal(SIGUSR1, signal_handler);
        printf("Child: Waiting for signal from parent...\n");
        while(1); // Child process waits indefinitely
        } else {
        // Parent process
        printf("Parent: I am the parent process (PID: %d)\n", getpid());
        printf("Parent: Sending signal to child...\n");
        // Send SIGUSR1 signal to the child process
        kill(pid, SIGUSR1);
        wait(NULL); // Wait for child to finish
        printf("Parent: Child process finished\n");
        }

        return 0;
}
```

10. Write a program to use alarm and signal sytem call(check i/p from user within time)

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

// Global variable to track whether the alarm has gone off
volatile sig_atomic_t alarm_triggered = 0;

// Signal handler function for SIGALRM
```

```
void alarm_handler(int signum) {
        alarm_triggered = 1;
}

int main() {
        char input[100];

        // Set up signal handler for SIGALRM
        if (signal(SIGALRM, alarm_handler) == SIG_ERR) {
        perror("signal");
        exit(EXIT_FAILURE);
        }

        printf("Enter something within 5 seconds: ");

        // Set an alarm for 5 seconds
        alarm(5);

        // Read user input
        fgets(input, sizeof(input), stdin);

        // Check if the alarm has been triggered
        if (alarm_triggered) {
        printf("Time's up! You didn't enter anything within 5 seconds.\n");
        } else {
        printf("You entered: %s", input);
        }

        return 0;
}
```

11. Write a program for alarm clock using alarm and signal system call.

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

// Global variable to track whether the alarm has gone off
volatile sig_atomic_t alarm_triggered = 0;

// Signal handler function for SIGALRM
```

```
void alarm_handler(int signum) {
        alarm_triggered = 1;
}

int main() {
        int seconds;

        // Set up signal handler for SIGALRM
        if (signal(SIGALRM, alarm_handler) == SIG_ERR) {
        perror("signal");
        exit(EXIT_FAILURE);
        }

        printf("Enter the number of seconds for the alarm: ");
        scanf("%d", &seconds);

        printf("Setting alarm for %d seconds...\n", seconds);

        // Set an alarm for the specified number of seconds
        alarm(seconds);

        // Wait for the alarm to go off
        while (!alarm_triggered) {
        // Wait for the alarm to trigger
        }

        printf("Alarm triggered! Time's up!\n");

        return 0;
}
```

12. Write a program to give statistics of a given file using stat system call. (few imp field like FAP, file type)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <time.h>
#include <pwd.h>
#include <grp.h>
```

```
int main(int argc, char *argv[]) {
        if (argc != 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        exit(EXIT_FAILURE);
        }

        struct stat fileStat;

        if (stat(argv[1], &fileStat) == -1) {
        perror("stat");
        exit(EXIT_FAILURE);
        }

        printf("File: %s\n", argv[1]);
        printf("Size: %ld bytes\n", fileStat.st_size);
        printf("File Permissions: ");
        printf((S_ISDIR(fileStat.st_mode)) ? "d" : "-");
        printf((fileStat.st_mode & S_IRUSR) ? "r" : "-");
        printf((fileStat.st_mode & S_IWUSR) ? "w" : "-");
        printf((fileStat.st_mode & S_IXUSR) ? "x" : "-");
        printf((fileStat.st_mode & S_IRGRP) ? "r" : "-");
        printf((fileStat.st_mode & S_IWGRP) ? "w" : "-");
        printf((fileStat.st_mode & S_IXGRP) ? "x" : "-");
        printf((fileStat.st_mode & S_IROTH) ? "r" : "-");
        printf((fileStat.st_mode & S_IWOTH) ? "w" : "-");
        printf((fileStat.st_mode & S_IXOTH) ? "x\n" : "-\n");

        printf("File Type: ");
        switch (fileStat.st_mode & S_IFMT) {
        case S_IFREG:
        printf("Regular File\n");
        break;
        case S_IFDIR:
        printf("Directory\n");
        break;
        case S_IFLNK:
        printf("Symbolic Link\n");
        break;
        case S_IFBLK:
```

```
    printf("Block Device\n");
    break;
    case S_IFCHR:
    printf("Character Device\n");
    break;
    case S_IFIFO:
    printf("FIFO/Named Pipe\n");
    break;
    case S_IFSOCK:
    printf("Socket\n");
    break;
    default:
    printf("Unknown\n");
    }

    printf("Inode Number: %ld\n", fileStat.st_ino);
    printf("Number of Hard Links: %ld\n", fileStat.st_nlink);

    struct passwd *pwd = getpwuid(fileStat.st_uid);
    printf("Owner User ID: %d (%s)\n", fileStat.st_uid, pwd ? pwd->pw_name :
"Unknown");

    struct group *grp = getgrgid(fileStat.st_gid);
    printf("Owner Group ID: %d (%s)\n", fileStat.st_gid, grp ? grp->gr_name :
"Unknown");

    printf("Last Access Time: %s", ctime(&fileStat.st_atime));
    printf("Last Modification Time: %s", ctime(&fileStat.st_mtime));
    printf("Last Status Change Time: %s", ctime(&fileStat.st_ctime));

    return 0;
}
```

13. Write a program to give statistics of a given file using fstat system call. (few imp field like FAP, file type)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
```

```
#include <time.h>
#include <pwd.h>
#include <grp.h>
#include <unistd.h> // Include for close function

int main(int argc, char *argv[]) {
        if (argc != 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        exit(EXIT_FAILURE);
        }

        struct stat fileStat;
        int fd;

        fd = open(argv[1], O_RDONLY);
        if (fd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
        }

        if (fstat(fd, &fileStat) == -1) {
        perror("fstat");
        exit(EXIT_FAILURE);
        }

        printf("File: %s\n", argv[1]);
        printf("Size: %ld bytes\n", fileStat.st_size);
        printf("File Permissions: ");
        printf((S_ISDIR(fileStat.st_mode)) ? "d" : "-");
        printf((fileStat.st_mode & S_IRUSR) ? "r" : "-");
        printf((fileStat.st_mode & S_IWUSR) ? "w" : "-");
        printf((fileStat.st_mode & S_IXUSR) ? "x" : "-");
        printf((fileStat.st_mode & S_IRGRP) ? "r" : "-");
        printf((fileStat.st_mode & S_IWGRP) ? "w" : "-");
        printf((fileStat.st_mode & S_IXGRP) ? "x" : "-");
        printf((fileStat.st_mode & S_IROTH) ? "r" : "-");
        printf((fileStat.st_mode & S_IWOTH) ? "w" : "-");
        printf((fileStat.st_mode & S_IXOTH) ? "x\n" : "-\n");

        printf("File Type: ");
```

```
    switch (fileStat.st_mode & S_IFMT) {
    case S_IFREG:
    printf("Regular File\n");
    break;
    case S_IFDIR:
    printf("Directory\n");
    break;
    case S_IFLNK:
    printf("Symbolic Link\n");
    break;
    case S_IFBLK:
    printf("Block Device\n");
    break;
    case S_IFCHR:
    printf("Character Device\n");
    break;
    case S_IFIFO:
    printf("FIFO/Named Pipe\n");
    break;
    case S_IFSOCK:
    printf("Socket\n");
    break;
    default:
    printf("Unknown\n");
    }

    printf("Inode Number: %ld\n", fileStat.st_ino);
    printf("Number of Hard Links: %ld\n", fileStat.st_nlink);

    struct passwd *pwd = getpwuid(fileStat.st_uid);
    printf("Owner User ID: %d (%s)\n", fileStat.st_uid, pwd ? pwd->pw_name :
"Unknown");

    struct group *grp = getgrgid(fileStat.st_gid);
    printf("Owner Group ID: %d (%s)\n", fileStat.st_gid, grp ? grp->gr_name :
"Unknown");

    printf("Last Access Time: %s", ctime(&fileStat.st_atime));
    printf("Last Modification Time: %s", ctime(&fileStat.st_mtime));
    printf("Last Status Change Time: %s", ctime(&fileStat.st_ctime));
    close(fd);

    return 0;
}
```

14. Write a multithreaded program in JAVA for chatting.

```
import java.io.*;
import java.net.*;
import java.util.*;

public class ChatServer {
    private static final int PORT = 12345;
    private static Set<String> usernames = new HashSet<>();
    private static List<ClientHandler> clients = new ArrayList<>();

    public static void main(String[] args) {
    try (ServerSocket serverSocket = new ServerSocket(PORT)) {
    System.out.println("Chat server is running on port " + PORT);

    while (true) {
        Socket clientSocket = serverSocket.accept();
        System.out.println("New client connected: " + clientSocket);

        ClientHandler clientHandler = new ClientHandler(clientSocket);
        clients.add(clientHandler);
        clientHandler.start();
    }
    } catch (IOException e) {
    e.printStackTrace();
    }
    }

    private static class ClientHandler extends Thread {
    private Socket clientSocket;
    private PrintWriter out;
    private BufferedReader in;
    private String username;

    public ClientHandler(Socket socket) {
    this.clientSocket = socket;
    }

    public void run() {
    try {
        out = new PrintWriter(clientSocket.getOutputStream(), true);
        in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

        out.println("Welcome to the chat server! Please enter your username:");
        username = in.readLine();

        synchronized (usernames) {
        while (usernames.contains(username)) {
        out.println("Username already taken. Please choose another one:");
        username = in.readLine();
        }
        usernames.add(username);
        }

        out.println("Welcome, " + username + "!");

        String input;
        while ((input = in.readLine()) != null) {
        if (input.equalsIgnoreCase("/quit")) {
        break;
        }
        broadcast(username + ": " + input);
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
        if (username != null) {
        synchronized (usernames) {
                usernames.remove(username);
        }
        System.out.println(username + " has left the chat.");
        }
        if (clientSocket != null) {
```

```
        clientSocket.close();
        }
        } catch (IOException e) {
        e.printStackTrace();
        }
    }
    }

    private void broadcast(String message) {
    synchronized (clients) {
        for (ClientHandler client : clients) {
        client.out.println(message);
        }
    }
    }
    }
}
```

CLIENT

```
import java.io.*;
import java.net.*;

public class ChatClient {
    private static final String SERVER_ADDRESS = "localhost";
    private static final int SERVER_PORT = 12345;

    public static void main(String[] args) {
    try (
    Socket socket = new Socket(SERVER_ADDRESS, SERVER_PORT);
    PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
    BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in))
    ) {
    System.out.println("Connected to the chat server.");
    System.out.println(in.readLine());

    Thread receiveThread = new Thread(() -> {
        try {
        String message;
        while ((message = in.readLine()) != null) {
        System.out.println(message);
        }
        } catch (IOException e) {
        e.printStackTrace();
        }
    });
    receiveThread.start();

    String input;
    while ((input = stdin.readLine()) != null) {
        out.println(input);
        if (input.equalsIgnoreCase("/quit")) {
        break;
        }
    }

    receiveThread.join(); // Wait for receive thread to finish
    } catch (IOException | InterruptedException e) {
    e.printStackTrace();
    }
    }
}
```

15. Write a program to create 3 threads, first thread printing even no, second thread printing odd no. and third thread printing prime no

```
public class NumberThreads {
    public static void main(String[] args) {
    Thread evenThread = new Thread(new EvenRunnable());
    Thread oddThread = new Thread(new OddRunnable());
    Thread primeThread = new Thread(new PrimeRunnable());

    evenThread.start();
    oddThread.start();
    primeThread.start();
    }
}

class EvenRunnable implements Runnable {
    public void run() {
    for (int i = 0; i <= 10; i += 2) {
```

```
    System.out.println("Even: " + i);
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    }
    }
}

class OddRunnable implements Runnable {
    public void run() {
    for (int i = 1; i <= 10; i += 2) {
    System.out.println("Odd: " + i);
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    }
    }
}

class PrimeRunnable implements Runnable {
    public void run() {
    for (int i = 2; i <= 10; i++) {
    if (isPrime(i)) {
        System.out.println("Prime: " + i);
    }
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    }
    }

    private boolean isPrime(int n) {
    if (n <= 1) {
    return false;
    }
    for (int i = 2; i <= Math.sqrt(n); i++) {
    if (n % i == 0) {
        return false;
    }
    }
    return true;
    }
}
```

16. Write a multithread program in linux to use the pthread library.

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

#define NUM_THREADS 5

void *threadFunction(void *threadId) {
    long tid;
    tid = (long) threadId;
    printf("Hello from thread %ld\n", tid);
    pthread_exit(NULL);
}

int main() {
    pthread_t threads[NUM_THREADS];
    int rc;
    long t;

    for (t = 0; t < NUM_THREADS; t++) {
    printf("Creating thread %ld\n", t);
    rc = pthread_create(&threads[t], NULL, threadFunction, (void *) t);
    if (rc) {
    printf("ERROR: return code from pthread_create() is %d\n", rc);
    exit(-1);
    }
    }

    pthread_exit(NULL);
}
```

17. Write a multithreaded program for producer-consumer problem in JAVA.

```
import java.util.LinkedList;

public class ProducerConsumer {
    public static void main(String[] args) {
    Buffer buffer = new Buffer(5); // Buffer size is 5
    Producer producer = new Producer(buffer);
    Consumer consumer = new Consumer(buffer);

    Thread producerThread = new Thread(producer);
    Thread consumerThread = new Thread(consumer);

    producerThread.start();
    consumerThread.start();
    }
}

class Buffer {
    private LinkedList<Integer> buffer;
    private int capacity;

    public Buffer(int capacity) {
    this.buffer = new LinkedList<>();
    this.capacity = capacity;
    }

    public synchronized void produce(int item) throws InterruptedException {
    while (buffer.size() == capacity) {
    wait(); // Wait if buffer is full
    }
    buffer.add(item);
    System.out.println("Produced: " + item);
    notify(); // Notify consumer thread that a new item is produced
    }

    public synchronized int consume() throws InterruptedException {
    while (buffer.isEmpty()) {
    wait(); // Wait if buffer is empty
    }
    int item = buffer.remove();
    System.out.println("Consumed: " + item);
    notify(); // Notify producer thread that an item is consumed
    return item;
    }
}

class Producer implements Runnable {
    private Buffer buffer;

    public Producer(Buffer buffer) {
    this.buffer = buffer;
    }

    public void run() {
    for (int i = 1; i <= 10; i++) {
    try {
        buffer.produce(i);
        Thread.sleep(1000); // Simulate some time taken to produce an item
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    }
    }
}

class Consumer implements Runnable {
    private Buffer buffer;

    public Consumer(Buffer buffer) {
    this.buffer = buffer;
    }

    public void run() {
    for (int i = 1; i <= 10; i++) {
    try {
        int item = buffer.consume();
        Thread.sleep(2000); // Simulate some time taken to consume an item
    } catch (InterruptedException e) {
        e.printStackTrace();
        }
        }
        }
}
```

18. Write a program to implement shell script for calculator

```
#!/bin/bash

echo "Calculator"

echo "Enter first number:"
read num1

echo "Enter second number:"
read num2

echo "Choose operation:"
echo "1. Addition"
echo "2. Subtraction"
echo "3. Multiplication"
echo "4. Division"
read choice

case $choice in
        1)
        result=$(echo "$num1 + $num2" | bc)
        echo "Result: $result"
        ;;
        2)
        result=$(echo "$num1 - $num2" | bc)
        echo "Result: $result"
        ;;
        3)
        result=$(echo "$num1 * $num2" | bc)
        echo "Result: $result"
        ;;
        4)
        if [ $num2 -eq 0 ]; then
        echo "Error: Division by zero"
        else
        result=$(echo "scale=2; $num1 / $num2" | bc)
        echo "Result: $result"
        fi
        ;;
        *)
        echo "Invalid choice"
        ;;
esac
```

```
snehal@snehal-HP-Notebook:~/UOS$ chmod +x 17th.sh
snehal@snehal-HP-Notebook:~/UOS$ ./17th.sh
```

19. Write a program to implement digital clock using shell script.

```
#!/bin/bash

while true; do
        clear
        date +"%H:%M:%S"
        sleep 1
done
```

20. Write a program to check whether system is in network or not using ’ping’ command using shell script.

```
#!/bin/bash

# Define the IP address or domain name to ping
target="google.com"

# Ping the target with a single packet and wait for 1 second
ping -c 1 -W 1 "$target" > /dev/null 2>&1

# Check the exit status of the ping command
if [ $? -eq 0 ]; then
        echo "System is connected to the network"
else
        echo "System is not connected to the network"
fi
```

21. Write a program to sort 10 the given 10 numbers in ascending order using shell.

```
#!/bin/bash

# Input 10 numbers
echo "Enter 10 numbers:"
read -a numbers

# Sort the numbers in ascending order
sorted_numbers=$(printf "%s\n" "${numbers[@]}" | sort -n)

# Print the sorted numbers
echo "Sorted numbers in ascending order:"
echo "$sorted_numbers"
```

22. Write a program to print “Hello World” message in bold, blink effect, and in different colors like red, blue etc.

```
#!/bin/bash

# Bold text
echo -e "\e[1mHello World\e[0m"

# Blinking text
echo -e "\e[5mHello World\e[0m"

# Red text
echo -e "\e[31mHello World\e[0m"

# Green text
echo -e "\e[32mHello World\e[0m"

# Yellow text
echo -e "\e[33mHello World\e[0m"

# Blue text
echo -e "\e[34mHello World\e[0m"

# Magenta text
echo -e "\e[35mHello World\e[0m"

# Cyan text
echo -e "\e[36mHello World\e[0m"
```

23. Write a shell script to find whether given file exist or not in folder or on drive

```
#!/bin/bash

# Check if correct number of arguments are provided
if [ $# -ne 2 ]; then
        echo "Usage: $0 <folder_path> <file_name>"
        exit 1
fi

folder_path="$1"
file_name="$2"

# Check if the folder exists
if [ ! -d "$folder_path" ]; then
        echo "Error: Folder $folder_path does not exist"
        exit 1
fi

# Check if the file exists in the folder
if [ -e "$folder_path/$file_name" ]; then
        echo "File $file_name exists in $folder_path"
else
        echo "File $file_name does not exist in $folder_path"
fi
```

24. Write a shell script to show the disk partitions and their size and disk usage i.e free space.

```
#!/bin/bash

echo "Disk partitions and their sizes:"

# Use the df command to display disk partitions and their sizes
df -h | awk '{print $1 "\t" $2 "\t" $4}'

echo -e "\nDisk usage (free space) for each partition:"

# Use the df command to display disk usage (free space) for each partition
df -h | awk '{print $1 "\t" $5}'
```

25. Write a shell script to find the given file in the system using find or locate command.

```
#!/bin/bash

# Check if correct number of arguments are provided
if [ $# -ne 1 ]; then
        echo "Usage: $0 <file_name>"
        exit 1
fi

file_name="$1"

# Use the find command to search for the file in the system
found_files=$(find / -name "$file_name" 2>/dev/null)

# Check if any files are found
if [ -n "$found_files" ]; then
        echo "Found the following occurrences of $file_name in the system:"
        echo "$found_files"
else
        echo "File $file_name not found in the system"
fi
```

26. Write a shell script to download webpage at given url using command(wget)

```
#!/bin/bash

# Check if correct number of arguments are provided
if [ $# -ne 1 ]; then
        echo "Usage: $0 <url>"
        exit 1
fi

url="$1"

# Use wget to download the webpage at the given URL
wget "$url" -O downloaded_page.html

if [ $? -eq 0 ]; then
        echo "Webpage downloaded successfully"
else
```

```
	echo "Failed to download the webpage"
fi
```

27. Write a shell script to download a webpage from given URL . (Using wget command)

```
#!/bin/bash

# Check if correct number of arguments are provided
if [ $# -ne 1 ]; then
	echo "Usage: $0 <url>"
	exit 1
fi

url="$1"

# Use wget to download the webpage at the given URL
wget "$url" -O downloaded_page.html

if [ $? -eq 0 ]; then
	echo "Webpage downloaded successfully"
else
	echo "Failed to download the webpage"
fi
```

28. Write a shell script to display the users on the system . (Using finger or who command).

```
#!/bin/bash

echo "Users currently logged in to the system:"
who
```

29. Write a python recursive function for prime number input limit in as parameter to it.

```
def is_prime(n, divisor=2):
	if n <= 1:
	return False
	if n == 2:
	return True
	if n % divisor == 0:
	return False
	if divisor * divisor > n:
    return True
    return is_prime(n, divisor + 1)

def find_primes(limit, num=2):
    if num <= limit:
    if is_prime(num):
    print(num, end=" ")
    find_primes(limit, num + 1)

# Example usage:
limit = int(input("Enter the limit to find prime numbers: "))
print("Prime numbers up to", limit, "are:")
find_primes(limit)
```

30. Write a program to display the following pyramid. The number of lines in the pyramid should not be hard-coded. It should be obtained from the user. The pyramid should appear as close to the center of the screen as possible. (Hint: Basics n loop)

```
import shutil

def display_pyramid(num_lines):
    max_width = num_lines * 2 - 1

    # Get the width of the terminal window
    terminal_width = shutil.get_terminal_size().columns

    for i in range(1, num_lines + 1):
    stars = '*' * (i * 2 - 1)
    print(stars.center(max_width).center(terminal_width))

# Get the number of lines for the pyramid from the user
num_lines = int(input("Enter the number of lines for the pyramid: "))

# Display the pyramid
display_pyramid(num_lines)
```

31. Take any txt file and count word frequencies in a file.(hint : file handling + basics)

```
def count_word_frequencies(file_path):
    word_freq = {}
    with open(file_path, 'r') as file:
    # Read each line from the file
    for line in file:
    # Split the line into words
    words = line.split()

    # Count the frequencies of words
    for word in words:
        # Remove punctuation and convert to lowercase
        word = word.strip().lower().strip('.,?!')

        # Increment the frequency count for the word
        word_freq[word] = word_freq.get(word, 0) + 1

    return word_freq

# Path to the text file
file_path = 'sample.txt'  # Change this to the path of your text file


# Count word frequencies in the file
word_freq = count_word_frequencies(file_path)

# Print word frequencies
for word, freq in word_freq.items():
    print(f'{word}: {freq}')
```

32. Generate frequency list of all the commands you have used, and show the top 5 commands along with their count. (Hint: history command hist will give you a list of all commands used.)

```
# Generate frequency list of all the commands you have used, and show the top 5
# commands along with their count. (Hint: history command hist will give you a list of
# all commands used.
# Get the list of all commands from history and count their frequencies
command_freq=$(history | awk '{print $2}' | sort | uniq -c | sort -nr)

# Display the top 5 commands along with their counts
echo "Top 5 commands:"
echo "$command_freq" | head -n 5
```

33. Write a shell script that will take a filename as input and check if it is executable. 2. Modify the script in the previous question, to remove the execute permissions, if the file is executable.

1.Shell script to check if a file is executable

```
#!/bin/bash

# Check if correct number of arguments are provided
if [ $# -ne 1 ]; then
	echo "Usage: $0 <filename>"
	exit 1
fi

filename="$1"

# Check if the file is executable
if [ -x "$filename" ]; then
	echo "$filename is executable"
else
	echo "$filename is not executable"
fi
```

2.Shell script to remove execute permissions if the file is executable

```
#!/bin/bash

# Check if correct number of arguments are provided
if [ $# -ne 1 ]; then
	echo "Usage: $0 <filename>"
	exit 1
fi
```

```
filename="$1"

# Check if the file is executable
if [ -x "$filename" ]; then
	# Remove execute permissions
	chmod -x "$filename"
	echo "Execute permissions removed from $filename"
else
	echo "$filename is not executable"
fi
```

34. Generate a word frequency list for wonderland.txt. Hint: use grep, tr, sort, uniq (or anything else that you want)

```
#!/bin/bash

# Check if the file exists
if [ ! -f "wonderland.txt" ]; then
	echo "Error: File wonderland.txt not found."
	exit 1
fi

# Extract words from the file, convert to lowercase, and remove punctuation
words=$(grep -oE '\b\w+\b' wonderland.txt | tr '[:upper:]' '[:lower:]' | tr -d '[:punct:]')

# Count the frequency of each word and sort them
word_freq=$(echo "$words" | tr ' ' '\n' | sort | uniq -c)

# Print the word frequency list
echo "$word_freq"
```

35. Write a bash script that takes 2 or more arguments,
i)All arguments are filenames
ii)If fewer than two arguments are given, print an error message
iii)If the files do not exist, print error message
iv)Otherwise concatenate files

```
#!/bin/bash

# Check if the number of arguments is less than 2
if [ "$#" -lt 2 ]; then
    echo "Error: At least two filenames are required."
    exit 1
fi

# Check if all files exist
for filename in "$@"; do
    if [ ! -f "$filename" ]; then
    echo "Error: File $filename does not exist."
    exit 1
    fi
done

# Concatenate all files
cat "$@"
```

36. Write a python function for merge/quick sort for integer list as parameter to it.
Merge Sort:

```
def merge_sort(arr):
    if len(arr) > 1:
    mid = len(arr) // 2
    left_half = arr[:mid]
    right_half = arr[mid:]

    merge_sort(left_half)
    merge_sort(right_half)

    i = j = k = 0

    while i < len(left_half) and j < len(right_half):
    if left_half[i] < right_half[j]:
        arr[k] = left_half[i]
        i += 1
    else:
        arr[k] = right_half[j]
        j += 1
    k += 1
    while i < len(left_half):
    arr[k] = left_half[i]
    i += 1
    k += 1

    while j < len(right_half):
    arr[k] = right_half[j]
    j += 1
    k += 1

def merge_sort_test():
    arr = [12, 11, 13, 5, 6, 7]
    merge_sort(arr)
    print("Merge Sorted array is:", arr)

merge_sort_test()
```

Quick Sort:

```
def quick_sort(arr):
    if len(arr) <= 1:
    return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

def quick_sort_test():
    arr = [12, 11, 13, 5, 6, 7]
    sorted_arr = quick_sort(arr)
    print("Quick Sorted array is:", sorted_arr)

quick_sort_test()
```

37. Write a shell script to download a given file from ftp://10.10.13.16 if it exists on ftp.(use lftp, get and mget commands).

```
#!/bin/bash

# FTP server details
FTP_SERVER="10.10.13.16"
FTP_USERNAME="your_username"
FTP_PASSWORD="your_password"

# File to download
FILE_TO_DOWNLOAD="example_file.txt"

# Check if file exists on the FTP server
lftp -u $FTP_USERNAME,$FTP_PASSWORD -e "ls $FILE_TO_DOWNLOAD; quit"
$FTP_SERVER > /dev/null 2>&1
if [ $? -eq 0 ]; then
        # File exists, download it
        lftp -u $FTP_USERNAME,$FTP_PASSWORD -e "get $FILE_TO_DOWNLOAD;
quit" $FTP_SERVER
        echo "File downloaded successfully."
else
        echo "File does not exist on the FTP server."
fi
```

38. Write program to implement producer consumer problem using semaphore.h in C/JAVA

C code:

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

#define BUFFER_SIZE 5
#define NUM_PRODUCERS 2
#define NUM_CONSUMERS 2

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

sem_t mutex, full, empty;
```

```
void *producer(void *arg) {
    int item;
    for (int i = 0; i < 10; i++) {
    item = rand() % 100; // Generate a random item
    sem_wait(&empty);
    sem_wait(&mutex);
    buffer[in] = item;
    printf("Producer %d produced item: %d\n", *(int *)arg, item);
    in = (in + 1) % BUFFER_SIZE;
    sem_post(&mutex);
    sem_post(&full);
    }
    pthread_exit(NULL);
}

void *consumer(void *arg) {
    int item;
    for (int i = 0; i < 10; i++) {
    sem_wait(&full);
    sem_wait(&mutex);
    item = buffer[out];
    printf("Consumer %d consumed item: %d\n", *(int *)arg, item);
    out = (out + 1) % BUFFER_SIZE;
    sem_post(&mutex);
    sem_post(&empty);
    }
    pthread_exit(NULL);
}

int main() {
    pthread_t producers[NUM_PRODUCERS], consumers[NUM_CONSUMERS];
    int producer_ids[NUM_PRODUCERS], consumer_ids[NUM_CONSUMERS];

    sem_init(&mutex, 0, 1);
    sem_init(&full, 0, 0);
    sem_init(&empty, 0, BUFFER_SIZE);

    for (int i = 0; i < NUM_PRODUCERS; i++) {
    producer_ids[i] = i;
    pthread_create(&producers[i], NULL, producer, &producer_ids[i]);
    }

    for (int i = 0; i < NUM_CONSUMERS; i++) {
    consumer_ids[i] = i;
    pthread_create(&consumers[i], NULL, consumer, &consumer_ids[i]);
    }

    for (int i = 0; i < NUM_PRODUCERS; i++) {
    pthread_join(producers[i], NULL);
    }

    for (int i = 0; i < NUM_CONSUMERS; i++) {
    pthread_join(consumers[i], NULL);
    }

    sem_destroy(&mutex);
    sem_destroy(&full);
    sem_destroy(&empty);

    return 0;
}
```

Java Code:

```
import java.util.concurrent.Semaphore;

public class Main {
    static final int BUFFER_SIZE = 5;
    static final int NUM_PRODUCERS = 2;
    static final int NUM_CONSUMERS = 2;

    static int[] buffer = new int[BUFFER_SIZE];
    static int in = 0, out = 0;

    static Semaphore mutex = new Semaphore(1);
    static Semaphore full = new Semaphore(0);
    static Semaphore empty = new Semaphore(BUFFER_SIZE);

    static class Producer implements Runnable {
    private int id;

    Producer(int id) {
```

```
this.id = id;
}

public void run() {
try {
        for (int i = 0; i < 10; i++) {
        int item = (int) (Math.random() * 100); // Generate a random item
        empty.acquire();
        mutex.acquire();
        buffer[in] = item;
        System.out.println("Producer " + id + " produced item: " + item);
        in = (in + 1) % BUFFER_SIZE;
        mutex.release();
        full.release();
        }
} catch (InterruptedException e) {
        e.printStackTrace();
}
}
}

static class Consumer implements Runnable {
private int id;

Consumer(int id) {
this.id = id;
}

public void run() {
try {
        for (int i = 0; i < 10; i++) {
        full.acquire();
        mutex.acquire();
        int item = buffer[out];
        System.out.println("Consumer " + id + " consumed item: " + item);
        out = (out + 1) % BUFFER_SIZE;
        mutex.release();
        empty.release();
        }
} catch (InterruptedException e) {
        e.printStackTrace();
    }
    }
    }

    public static void main(String[] args) {
    Thread[] producers = new Thread[NUM_PRODUCERS];
    Thread[] consumers = new Thread[NUM_CONSUMERS];

    for (int i = 0; i < NUM_PRODUCERS; i++) {
    producers[i] = new Thread(new Producer(i));
    producers[i].start();
    }

    for (int i = 0; i < NUM_CONSUMERS; i++) {
    consumers[i] = new Thread(new Consumer(i));
    consumers[i].start();
    }
    }
}
```

39. Write a program to implement reader-writers problem using semaphore.

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

#define NUM_READERS 5
#define NUM_WRITERS 2

sem_t mutex, write_mutex;
int readers_count = 0;
int shared_resource = 0;

void *reader(void *arg) {
    int id = *((int *)arg);
    while (1) {
    sem_wait(&mutex);
    readers_count++;
    if (readers_count == 1) {
    sem_wait(&write_mutex);
    }
    sem_post(&mutex);

    // Reading shared resource
    printf("Reader %d read: %d\n", id, shared_resource);

    sem_wait(&mutex);
    readers_count--;
    if (readers_count == 0) {
    sem_post(&write_mutex);
    }
    sem_post(&mutex);

    // Simulating some delay
    usleep(100000);
    }
}

void *writer(void *arg) {
    int id = *((int *)arg);
    while (1) {
    sem_wait(&write_mutex);

    // Writing to shared resource
    shared_resource++;
    printf("Writer %d wrote: %d\n", id, shared_resource);

    sem_post(&write_mutex);

    // Simulating some delay
    usleep(200000);
    }
}

int main() {
    pthread_t readers[NUM_READERS], writers[NUM_WRITERS];
    int reader_ids[NUM_READERS], writer_ids[NUM_WRITERS];

    sem_init(&mutex, 0, 1);
    sem_init(&write_mutex, 0, 1);
    for (int i = 0; i < NUM_READERS; i++) {
    reader_ids[i] = i + 1;
    pthread_create(&readers[i], NULL, reader, &reader_ids[i]);
    }

    for (int i = 0; i < NUM_WRITERS; i++) {
    writer_ids[i] = i + 1;
    pthread_create(&writers[i], NULL, writer, &writer_ids[i]);
    }

    for (int i = 0; i < NUM_READERS; i++) {
    pthread_join(readers[i], NULL);
    }

    for (int i = 0; i < NUM_WRITERS; i++) {
    pthread_join(writers[i], NULL);
    }

    sem_destroy(&mutex);
    sem_destroy(&write_mutex);

    return 0;
}
```

40. Write a program for chatting between two/three users to demonstrate IPC using message passing (msgget, msgsnd, msgrcv).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <unistd.h>
#include <signal.h>

#define MAX_MSG_SIZE 256
#define MSG_KEY 1234

struct message {
    long mtype;
    char mtext[MAX_MSG_SIZE];
};

int main() {
    int msgid;
    struct message msg;
    key_t key;

    // Create a message queue
    key = ftok("/tmp", MSG_KEY);
    if ((msgid = msgget(key, IPC_CREAT | 0666)) == -1) {
    perror("msgget");
    exit(1);
    }

    // Fork a child process for receiving messages
    pid_t pid = fork();

    if (pid == -1) {
    perror("fork");
    exit(1);
    } else if (pid == 0) { // Child process - Receiving messages
    while (1) {
    if (msgrcv(msgid, &msg, MAX_MSG_SIZE, 1, 0) == -1) {
        perror("msgrcv");
        exit(1);
    }
    printf("User 1: %s\n", msg.mtext);
    printf("Enter your message (User 2): ");
    fgets(msg.mtext, MAX_MSG_SIZE, stdin);
    msg.mtype = 1;
    // Remove the newline character from the message
    msg.mtext[strcspn(msg.mtext, "\n")] = 0;
    // Send the message
    if (msgsnd(msgid, &msg, strlen(msg.mtext) + 1, 0) == -1) {
        perror("msgsnd");
        exit(1);
    }
    }
    } else { // Parent process - Sending messages
    while (1) {
    printf("Enter your message (User 1): ");
    fgets(msg.mtext, MAX_MSG_SIZE, stdin);
    msg.mtype = 1;
    // Remove the newline character from the message
    msg.mtext[strcspn(msg.mtext, "\n")] = 0;
    // Send the message
    if (msgsnd(msgid, &msg, strlen(msg.mtext) + 1, 0) == -1) {
        perror("msgsnd");
        exit(1);
    }
    if (msgrcv(msgid, &msg, MAX_MSG_SIZE, 1, 0) == -1) {
        perror("msgrcv");
        exit(1);
    }
    printf("User 2: %s\n", msg.mtext);
    }
    }

    // Clean up: Remove the message queue
    if (msgctl(msgid, IPC_RMID, NULL) == -1) {
    perror("msgctl");
    exit(1);
    }

    return 0;
}
```

41. Write a program to demonstrate IPC using shared memory (shmget, shmat, shmdt). In this, one process will send A to Z/1 to 100 as input from user and another process will receive it.

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

#define SHM_SIZE 1024
```

```
int main() {
    int shmid;
    key_t key;
    char *shm, *s;

    // Generate a key for shared memory
    key = ftok("/tmp", 'S');
    if (key == -1) {
    perror("ftok");
    exit(1);
    }

    // Create a shared memory segment
    shmid = shmget(key, SHM_SIZE, IPC_CREAT | 0666);
    if (shmid == -1) {
    perror("shmget");
    exit(1);
    }

    // Attach the shared memory segment to the process's address space
    shm = shmat(shmid, NULL, 0);
    if (shm == (char *) -1) {
    perror("shmat");
    exit(1);
    }

    // Parent process: Sender
    printf("Enter input (A-Z or 1-100): ");
    fgets(shm, SHM_SIZE, stdin);

    // Detach the shared memory segment from the process's address space
    if (shmdt(shm) == -1) {
    perror("shmdt");
    exit(1);
    }

    // Child process: Receiver
    pid_t pid = fork();
    if (pid == -1) {
```

```
    perror("fork");
    exit(1);
    } else if (pid == 0) {
    // Child process: Receiver
    sleep(1); // Ensure sender process finishes writing to shared memory
    printf("Received input: %s", shm);
    }

    // Wait for the child process to finish
    wait(NULL);

    // Clean up: Remove the shared memory segment
    if (shmctl(shmid, IPC_RMID, NULL) == -1) {
    perror("shmctl");
    exit(1);
    }

    return 0;
}
```

42. Write a program to demonstrate IPC using shared memory (shmget, shmat, shmdt). In this, one process will send from file A to Z/1 to 100 as input from user and another process will receive it in file. (use same directory and different name files)

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

#define SHM_SIZE 1024 // Size of shared memory segment

int main() {
    key_t key = ftok(".", 'A'); // Generate a unique key
    int shmid; // Shared memory ID
    char *shm_ptr; // Pointer to shared memory
    char input[100]; // Input buffer
    int i;

    // Create a shared memory segment
    shmid = shmget(key, SHM_SIZE, IPC_CREAT | 0666);
    if (shmid == -1) {
    perror("shmget");
    exit(1);
    }

    // Attach the shared memory segment to the process's address space
    shm_ptr = shmat(shmid, NULL, 0);
    if (shm_ptr == (char *) -1) {
    perror("shmat");
    exit(1);
    }

    printf("Enter characters from A to Z or numbers from 1 to 100 (separated by
spaces):\n");
    fgets(input, sizeof(input), stdin); // Read input from user

    // Write input data to shared memory
    for (i = 0; input[i] != '\0'; i++) {
    shm_ptr[i] = input[i];
    }
    shm_ptr[i] = '\0'; // Null-terminate the data

    // Detach the shared memory segment
    if (shmdt(shm_ptr) == -1) {
    perror("shmdt");
    exit(1);
    }

    // Fork a child process to read from shared memory and write to file
    pid_t pid = fork();
    if (pid == -1) {
    perror("fork");
    exit(1);
    } else if (pid == 0) { // Child process
    FILE *file = fopen("output.txt", "w"); // Open file for writing
    if (file == NULL) {
```

```
    perror("fopen");
    exit(1);
    }

    // Attach the shared memory segment to the child process's address space
    shm_ptr = shmat(shmid, NULL, 0);
    if (shm_ptr == (char *) -1) {
    perror("shmat");
    exit(1);
    }

    // Write data from shared memory to file
    fprintf(file, "%s", shm_ptr);

    // Detach the shared memory segment from the child process
    if (shmdt(shm_ptr) == -1) {
    perror("shmdt");
    exit(1);
    }

    fclose(file); // Close the file
    exit(0);
    } else { // Parent process
    wait(NULL); // Wait for the child process to finish

    printf("Data has been written to output.txt\n");

    // Remove the shared memory segment
    if (shmctl(shmid, IPC_RMID, NULL) == -1) {
    perror("shmctl");
    exit(1);
    }
    }

    return 0;
}
```

43. Write a program to demonstrate IPC using shared memory (shmget, shmat, shmdt). In this, one process will take numbers as input from user and second process will sort the numbers and put back to shared memory. Third process will display the shared memory.

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

#define SHM_SIZE 1024 // Size of shared memory segment

int main() {
        key_t key = ftok(".", 'A'); // Generate a unique key
        int shmid; // Shared memory ID
        int *shm_ptr; // Pointer to shared memory
        int numbers[100]; // Array to store numbers
        int num_count; // Number of numbers entered by user
        int i;

        // Create a shared memory segment
        shmid = shmget(key, SHM_SIZE, IPC_CREAT | 0666);
        if (shmid == -1) {
        perror("shmget");
        exit(1);
        }

        // Attach the shared memory segment to the process's address space
        shm_ptr = (int *) shmat(shmid, NULL, 0);
        if (shm_ptr == (int *) -1) {
        perror("shmat");
        exit(1);
        }

        // Prompt the user to enter numbers
        printf("Enter the number of numbers (up to 100): ");
        scanf("%d", &num_count);
```

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

// Write input numbers to shared memory
for (i = 0; i < num_count; i++) {
shm_ptr[i] = numbers[i];
}

// Fork a child process to sort numbers in shared memory
pid_t pid = fork();
if (pid == -1) {
perror("fork");
exit(1);
} else if (pid == 0) { // Child process
// Sort numbers in shared memory using bubble sort algorithm
for (i = 0; i < num_count - 1; i++) {
for (int j = 0; j < num_count - i - 1; j++) {
        if (shm_ptr[j] > shm_ptr[j + 1]) {
        // Swap numbers if they are in the wrong order
        int temp = shm_ptr[j];
        shm_ptr[j] = shm_ptr[j + 1];
        shm_ptr[j + 1] = temp;
        }
}
}
exit(0);
} else { // Parent process
wait(NULL); // Wait for the child process to finish sorting

// Fork another child process to display sorted numbers
pid_t pid2 = fork();
if (pid2 == -1) {
perror("fork");
exit(1);
} else if (pid2 == 0) { // Child process
printf("Sorted numbers in shared memory:\n");
for (i = 0; i < num_count; i++) {
        printf("%d ", shm_ptr[i]);
        }
        printf("\n");
        exit(0);
        } else { // Parent process
        wait(NULL); // Wait for the second child process to finish

        // Detach the shared memory segment
        if (shmdt(shm_ptr) == -1) {
                perror("shmdt");
                exit(1);
        }

        // Remove the shared memory segment
        if (shmctl(shmid, IPC_RMID, NULL) == -1) {
                perror("shmctl");
                exit(1);
        }
        }
        }

        return 0;
}
```

44. Write a program in which different processes will perform different operation on shared memory. Operation: create memory, delete, attach/ detach(using shmget, shmat, shmdt).

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

#define SHM_SIZE 1024 // Size of shared memory segment

int main() {
        key_t key = ftok(".", 'A'); // Generate a unique key
        int shmid; // Shared memory ID
        char *shm_ptr; // Pointer to shared memory
```

```
// Fork a child process to create shared memory
pid_t pid_create = fork();
if (pid_create == -1) {
perror("fork");
exit(1);
} else if (pid_create == 0) { // Child process to create memory
// Create a shared memory segment
shmid = shmget(key, SHM_SIZE, IPC_CREAT | 0666);
if (shmid == -1) {
perror("shmget");
exit(1);
}
printf("Shared memory segment created\n");
exit(0);
} else { // Parent process
wait(NULL); // Wait for the child process to finish creating memory

// Fork another child process to attach to shared memory
pid_t pid_attach = fork();
if (pid_attach == -1) {
perror("fork");
exit(1);
} else if (pid_attach == 0) { // Child process to attach memory
// Attach the shared memory segment to the process's address space
shmid = shmget(key, SHM_SIZE, 0666); // Get existing shared memory ID
if (shmid == -1) {
        perror("shmget");
        exit(1);
}
shm_ptr = shmat(shmid, NULL, 0);
if (shm_ptr == (char *) -1) {
        perror("shmat");
        exit(1);
}
printf("Attached to shared memory segment\n");
exit(0);
} else { // Parent process
wait(NULL); // Wait for the child process to finish attaching memory
```

```
// Fork another child process to detach from shared memory
pid_t pid_detach = fork();
if (pid_detach == -1) {
        perror("fork");
        exit(1);
} else if (pid_detach == 0) { // Child process to detach memory
        // Detach the shared memory segment from the process's address space
        shmid = shmget(key, SHM_SIZE, 0666); // Get existing shared memory ID
        if (shmid == -1) {
        perror("shmget");
        exit(1);
        }
        shm_ptr = shmat(shmid, NULL, 0);
        if (shm_ptr == (char *) -1) {
        perror("shmat");
        exit(1);
        }
        if (shmdt(shm_ptr) == -1) {
        perror("shmdt");
        exit(1);
        }
        printf("Detached from shared memory segment\n");
        exit(0);
} else { // Parent process
        wait(NULL); // Wait for the child process to finish detaching memory

        // Fork another child process to delete shared memory
        pid_t pid_delete = fork();
        if (pid_delete == -1) {
        perror("fork");
        exit(1);
        } else if (pid_delete == 0) { // Child process to delete memory
        // Remove the shared memory segment
        shmid = shmget(key, SHM_SIZE, 0666); // Get existing shared memory ID
        if (shmid == -1) {
        perror("shmget");
        exit(1);
        }
        if (shmctl(shmid, IPC_RMID, NULL) == -1) {
        perror("shmctl");
        exit(1);
        }
        printf("Shared memory segment deleted\n");
        exit(0);
        } else { // Parent process
        wait(NULL); // Wait for the child process to finish deleting memory
        }
    }
    }
    }

    return 0;
}
```

45. Write programs to simulate linux commands cat, ls, cp, mv, head etc.
Cat:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    FILE *file;
    char ch;

    if (argc < 2) {
    printf("Usage: %s <filename1> [filename2] ...\n", argv[0]);
    return 1;
    }

    for (int i = 1; i < argc; i++) {
    file = fopen(argv[i], "r");
    if (file == NULL) {
    printf("Cannot open file: %s\n", argv[i]);
    continue;
    }

    while ((ch = fgetc(file)) != EOF) {
    putchar(ch);
    }

    fclose(file);
    }
    return 0;
}
```

Ls:

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

int main() {
    DIR *dir;
    struct dirent *entry;

    dir = opendir(".");
    if (dir == NULL) {
    perror("opendir");
    return 1;
    }

    while ((entry = readdir(dir)) != NULL) {
    printf("%s\n", entry->d_name);
    }

    closedir(dir);
    return 0;
}
```

Cp:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    FILE *src, *dest;
    char ch;

    if (argc != 3) {
    printf("Usage: %s <source> <destination>\n", argv[0]);
    return 1;
    }

    src = fopen(argv[1], "r");
    if (src == NULL) {
```

```
    perror("fopen");
    return 1;
    }

    dest = fopen(argv[2], "w");
    if (dest == NULL) {
    perror("fopen");
    fclose(src);
    return 1;
    }

    while ((ch = fgetc(src)) != EOF) {
    fputc(ch, dest);
    }

    fclose(src);
    fclose(dest);

    return 0;
}
```

Head:

```
#include <stdio.h>

#define DEFAULT_LINES 10

int main(int argc, char *argv[]) {
    FILE *file;
    char ch;
    int lines = DEFAULT_LINES;
    int count = 0;

    if (argc < 2) {
    printf("Usage: %s <filename> [lines]\n", argv[0]);
    return 1;
    }

    if (argc >= 3) {
```

```
    lines = atoi(argv[2]);
    }

    file = fopen(argv[1], "r");
    if (file == NULL) {
    perror("fopen");
    return 1;
    }

    while ((ch = fgetc(file)) != EOF && count < lines) {
    putchar(ch);
    if (ch == '\n') {
    count++;
    }
    }

    fclose(file);
    return 0;
}
```

46. Write a program to ensure that function f1 should executed before executing function f2 using semaphore. (Ex. Program should ask for username before entering password).

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

// Define semaphore
sem_t semaphore;

void f1() {
    printf("Enter username: ");
    char username[100];
    scanf("%s", username);
```

```
        // Simulate processing username
        printf("Processing username: %s\n", username);
}

void f2() {
        printf("Enter password: ");
        char password[100];
        scanf("%s", password);
        // Simulate processing password
        printf("Processing password: %s\n", password);
}

void *thread_function(void *arg) {
        // Wait for semaphore
        sem_wait(&semaphore);
        f1();
        // Post semaphore
        sem_post(&semaphore);
        f2();
        return NULL;
}

int main() {
        // Initialize semaphore
        sem_init(&semaphore, 0, 1);

        pthread_t thread;
        // Create a thread
        if (pthread_create(&thread, NULL, thread_function, NULL)) {
        fprintf(stderr, "Error creating thread\n");
        return 1;
        }

        // Join the thread
        if (pthread_join(thread, NULL)) {
        fprintf(stderr, "Error joining thread\n");
        return 1;
        }

        // Destroy semaphore
```

```
    sem_destroy(&semaphore);

    return 0;
}
```

47. Write a program using OpenMP library to parallelize the for loop in sequential program of finding prime numbers in given range

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

int is_prime(int n) {
    if (n <= 1) return 0;
    for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) return 0;
    }
    return 1;
}

int main() {
    int lower = 2, upper = 100; // Define the range
    printf("Prime numbers between %d and %d are:\n", lower, upper);

    #pragma omp parallel for
    for (int num = lower; num <= upper; num++) {
    if (is_prime(num)) {
    printf("%d\n", num);
    }
    }

    return 0;
}
```

```
gcc -fopenmp -o prime prime.c
```

48. Using OpemnMP library write a program in which master thread count the total no. of threads created, and others will print their thread numbers.

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

int main() {
    int total_threads, thread_id;

    // Start parallel region
    #pragma omp parallel private(thread_id)
    {
    thread_id = omp_get_thread_num();
    // Only master thread counts total number of threads
    #pragma omp master
    {
    total_threads = omp_get_num_threads();
    printf("Total number of threads: %d\n", total_threads);
    }
    printf("Thread number: %d\n", thread_id);
    }

    return 0;
}
```

49. Implement the program for IPC using MPI library (“Hello world” program).

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

int main(int argc, char **argv) {
    int rank, size;

    // Initialize MPI environment
    MPI_Init(&argc, &argv);

    // Get the rank of the current process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Get the total number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Print "Hello world" message from each process
    printf("Hello world from process %d of %d\n", rank, size);

    // Finalize MPI environment
    MPI_Finalize();
```

```
        return 0;
}
```

```
sudo apt install openmpi-bin libopenmpi-dev
 mpicc --version

mpicc -o 49th 49th.c

mpiexec -n 2 ./49th
```

50. Write a 2 programs that will both send and messages and construct the following dialog between them
(Process 1) Sends the message "Are you hearing me?"
(Process 2) Receives the message and replies "Loud and Clear".
(Process 1) Receives the reply and then says "I can hear you too".
IPC:Message Queues:msgget, msgsnd, msgrcv.

Process1:

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

// Define message structure
struct message {
        long mtype;
        char mtext[100];
};

int main() {
        key_t key = ftok(".", 'A'); // Generate a unique key for the message queue

        // Create a message queue
        int msqid = msgget(key, IPC_CREAT | 0666);
        if (msqid == -1) {
        perror("msgget");
        exit(1);
        }
```

```
    // Send message
    struct message msg_send;
    msg_send.mtype = 1; // Message type
    strcpy(msg_send.mtext, "Are you hearing me?");
    if (msgsnd(msqid, &msg_send, sizeof(msg_send.mtext), 0) == -1) {
    perror("msgsnd");
    exit(1);
    }
    printf("Process 1: Sent message - %s\n", msg_send.mtext);

    // Receive reply
    struct message msg_recv;
    if (msgrcv(msqid, &msg_recv, sizeof(msg_recv.mtext), 2, 0) == -1) {
    perror("msgrcv");
    exit(1);
    }
    printf("Process 1: Received reply - %s\n", msg_recv.mtext);

    // Send acknowledgment
    strcpy(msg_send.mtext, "I can hear you too");
    if (msgsnd(msqid, &msg_send, sizeof(msg_send.mtext), 0) == -1) {
    perror("msgsnd");
    exit(1);
    }
    printf("Process 1: Sent acknowledgment - %s\n", msg_send.mtext);

    // Clean up: Remove message queue
    if (msgctl(msqid, IPC_RMID, NULL) == -1) {
    perror("msgctl");
    exit(1);
    }

    return 0;
}
```

Process2:

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

```
#include <string.h>

// Define message structure
struct message {
        long mtype;
        char mtext[100];
};

int main() {
        key_t key = ftok(".", 'A'); // Generate the same key for the message queue

        // Get the message queue ID
        int msqid = msgget(key, 0666);
        if (msqid == -1) {
        perror("msgget");
        exit(1);
        }

        // Receive message
        struct message msg_recv;
        if (msgrcv(msqid, &msg_recv, sizeof(msg_recv.mtext), 1, 0) == -1) {
        perror("msgrcv");
        exit(1);
        }
        printf("Process 2: Received message - %s\n", msg_recv.mtext);

        // Reply
        struct message msg_reply;
        msg_reply.mtype = 2; // Message type
        strcpy(msg_reply.mtext, "Loud and Clear");
        if (msgsnd(msqid, &msg_reply, sizeof(msg_reply.mtext), 0) == -1) {
        perror("msgsnd");
        exit(1);
        }
        printf("Process 2: Replied - %s\n", msg_reply.mtext);

        return 0;
}
```

Make sure to compile each program separately:

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

Then run them in separate terminals:

```
./process1
./process2
```

51. Write a program for TCP to demonstrate the socket system calls in c/python

Server:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[BUFFER_SIZE] = {0};
    const char *message = "Hello from server";

    // Create TCP socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
    perror("socket failed");
    exit(EXIT_FAILURE);
    }

    // Prepare the sockaddr_in structure
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // Bind socket to localhost and port 8080
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
```

```
    perror("bind failed");
    exit(EXIT_FAILURE);
    }

    // Start listening for incoming connections
    if (listen(server_fd, 3) < 0) {
    perror("listen");
    exit(EXIT_FAILURE);
    }

    // Accept an incoming connection
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
(socklen_t*)&addrlen)) < 0) {
    perror("accept");
    exit(EXIT_FAILURE);
    }

    // Send message to client
    send(new_socket, message, strlen(message), 0);
    printf("Message sent to client\n");

    // Close the socket
    close(server_fd);
    return 0;
}
```

CLIENT:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[BUFFER_SIZE] = {0};
    // Create TCP socket
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("socket failed");
    exit(EXIT_FAILURE);
    }

    // Prepare sockaddr_in structure
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
    perror("inet_pton");
    exit(EXIT_FAILURE);
    }

    // Connect to server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
    perror("connect");
    exit(EXIT_FAILURE);
    }

    // Receive message from server
    read(sock, buffer, BUFFER_SIZE);
    printf("Message from server: %s\n", buffer);

    // Close the socket
    close(sock);
    return 0;
}
```

```
gcc -o server server.c
gcc -o client client.c
```

```
./server
./client
```

52. Write a program for UDP to demonstrate the socket system calls in c/python

Server:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
        int sockfd;
        char buffer[BUFFER_SIZE];
        struct sockaddr_in servaddr, cliaddr;

        // Create UDP socket
        if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
        }

        // Initialize server address structure
        memset(&servaddr, 0, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = INADDR_ANY;
        servaddr.sin_port = htons(PORT);

        // Bind socket with server address
        if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
        }

        int len, n;
        len = sizeof(cliaddr); // len is value/result

        // Receive message from client
```

```
    n = recvfrom(sockfd, (char *)buffer, BUFFER_SIZE, MSG_WAITALL, (struct sockaddr *)&cliaddr, &len);
    buffer[n] = '\0';
    printf("Client : %s\n", buffer);

    // Reply to client
    const char *message = "Hello from server";
    sendto(sockfd, (const char *)message, strlen(message), MSG_CONFIRM, (const struct sockaddr *)&cliaddr, len);
    printf("Message sent to client\n");

    // Close the socket
    close(sockfd);
    return 0;
}
```

Client:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int sockfd;
    char buffer[BUFFER_SIZE];
    struct sockaddr_in servaddr;

    // Create UDP socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("socket creation failed");
    exit(EXIT_FAILURE);
    }

    // Initialize server address structure
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
```

```
        servaddr.sin_port = htons(PORT);
        servaddr.sin_addr.s_addr = INADDR_ANY;

        int n, len;
        len = sizeof(servaddr); // len is value/result

        // Send message to server
        const char *message = "Hello from client";
        sendto(sockfd, (const char *)message, strlen(message), MSG_CONFIRM, (const
struct sockaddr *)&servaddr, len);
        printf("Message sent to server\n");

        // Receive message from server
        n = recvfrom(sockfd, (char *)buffer, BUFFER_SIZE, MSG_WAITALL, (struct
sockaddr *)&servaddr, &len);
        buffer[n] = '\0';
        printf("Server : %s\n", buffer);

        // Close the socket
        close(sockfd);
        return 0;
}
```

```
gcc -o udp_server udp_server.c
gcc -o udp_client udp_client.c

./udp_server
./udp_client
```

53. Implement echo server using TCP in iterative/concurrent logic.

CLIENT:

```
package TCP_Concurrent;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.*;

public class Client {

	public static void main(String[] args) {

	try {
	Socket socket = new Socket("localhost",8080);

	BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
	PrintWriter out = new PrintWriter(socket.getOutputStream(),true);

	BufferedReader userInput = new BufferedReader(new
InputStreamReader(System.in));
	String message;

	while((message = userInput.readLine())!= null){

		out.println(message);
		System.out.println("Sent to Server");

	}

	userInput.close();
	in.close();
	out.close();
	socket.close();
	} catch (Exception e) {
	System.out.println(e);
	}

	}
}
```

Server:

```
package TCP_Concurrent;
import java.io.*;
import java.net.*;

public class Server {
    public static void main(String[] args) {
    try {
    ServerSocket serverSocket = new ServerSocket(8080);
    System.out.println("Concurrent TCP Echo Server started. Listening on port
12345...");

    while (true) {
        Socket clientSocket = serverSocket.accept();
        System.out.println("Client connected: " + clientSocket);

        Thread clientHandler = new Thread(new ClientHandler(clientSocket));
        clientHandler.start();
    }
    } catch (IOException e) {
    e.printStackTrace();
    }
    }

    static class ClientHandler implements Runnable {
    private Socket clientSocket;

    public ClientHandler(Socket socket) {
    this.clientSocket = socket;
    }

    @Override
    public void run() {
    try {
        BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
        String message;
        while ((message = in.readLine()) != null) {
        System.out.println("Received from client: " + message);
        out.println("Echo: " + message);
        }

        // Close resources
        in.close();
        out.close();
        clientSocket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    }
    }
}
```

54. Implement echo server using UDP in iterative/concurrent logic
Client

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#define PORT 9999
#define BUFFER_SIZE 4096

int main() {
    struct sockaddr_in server_addr;
    int sockfd;
    char buffer[BUFFER_SIZE];
    socklen_t server_len;

    // Create a UDP socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0) {
    perror("socket creation failed");
    exit(EXIT_FAILURE);
```

```
        }

        // Initialize server address structure
        memset(&server_addr, 0, sizeof(server_addr));
        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = INADDR_ANY;
        server_addr.sin_port = htons(PORT);

        while (1) {
        printf("Enter message to send (type 'exit' to quit): ");
        fgets(buffer, BUFFER_SIZE, stdin);

        // Remove newline character from the input
        buffer[strcspn(buffer, "\n")] = 0;

        if (strcmp(buffer, "exit") == 0) {
        break;
        }

        // Send message to the server
        server_len = sizeof(server_addr);
        if (sendto(sockfd, buffer, strlen(buffer), 0, (struct sockaddr*)&server_addr,
server_len) < 0) {
        perror("sendto failed");
        exit(EXIT_FAILURE);
        }

        // Receive response from server
        if (recvfrom(sockfd, buffer, BUFFER_SIZE, 0, NULL, NULL) < 0) {
        perror("recvfrom failed");
        exit(EXIT_FAILURE);
        }

        printf("Received response: %s\n", buffer);
        }

        // Close the socket
        close(sockfd);

        return 0;
```

```
}
```

Server

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#define PORT 9999
#define BUFFER_SIZE 4096

int main() {
	struct sockaddr_in server_addr, client_addr;
	int sockfd, client_len, recv_len;
	char buffer[BUFFER_SIZE];

	// Create a UDP socket
	if ((sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0) {
	perror("socket creation failed");
	exit(EXIT_FAILURE);
	}

	// Initialize server address structure
	memset(&server_addr, 0, sizeof(server_addr));
	server_addr.sin_family = AF_INET;
	server_addr.sin_addr.s_addr = INADDR_ANY;
	server_addr.sin_port = htons(PORT);

	// Bind the socket
	if (bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
	perror("bind failed");
	exit(EXIT_FAILURE);
	}

	printf("Server is running on port %d\n", PORT);

	while (1) {
	printf("\nWaiting to receive message...\n");
```

```
    // Receive message from client
    client_len = sizeof(client_addr);
    if ((recv_len = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, (struct sockaddr*)&client_addr, &client_len)) < 0) {
    perror("recvfrom failed");
    exit(EXIT_FAILURE);
    }

    printf("Received %d bytes from %s:%d\n", recv_len, inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
    printf("Data received: %s\n", buffer);

    // Echo back the received data
    if (sendto(sockfd, buffer, recv_len, 0, (struct sockaddr*)&client_addr, client_len) < 0) {
    perror("sendto failed");
    exit(EXIT_FAILURE);
    }
    }

    return 0;
}
```

55. Write a program using PIPE, to Send data from parent to child over a pipe. (unnamed pipe)

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

#define BUFFER_SIZE 25

int main() {
    int pipefd[2]; // File descriptors for the pipe
    pid_t pid; // Process ID

    char buffer[BUFFER_SIZE]; // Buffer for reading from and writing to the pipe
    // Create the pipe
    if (pipe(pipefd) == -1) {
    perror("pipe");
    exit(EXIT_FAILURE);
    }

    // Fork a child process
    pid = fork();

    if (pid < 0) {
    perror("fork");
    exit(EXIT_FAILURE);
    }

    if (pid > 0) { // Parent process
    close(pipefd[0]); // Close the reading end of the pipe in the parent

    printf("Parent: Writing data to the pipe...\n");
    // Write data to the pipe
    write(pipefd[1], "Hello, child!", 14);

    close(pipefd[1]); // Close the writing end of the pipe in the parent
    printf("Parent: Data written to the pipe.\n");
    } else { // Child process
    close(pipefd[1]); // Close the writing end of the pipe in the child

    printf("Child: Reading data from the pipe...\n");
    // Read data from the pipe
    read(pipefd[0], buffer, BUFFER_SIZE);
    printf("Child: Received message: %s\n", buffer);

    close(pipefd[0]); // Close the reading end of the pipe in the child
    }

    return 0;
}
```

56. Write a program using FIFO, to Send data from parent to child over a pipe. (named pipe)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define FIFO_PATH "/tmp/myfifo"
#define BUFFER_SIZE 25

int main() {
        int fd; // File descriptor for the FIFO
        pid_t pid; // Process ID

        char buffer[BUFFER_SIZE]; // Buffer for reading from and writing to the FIFO

        // Create the FIFO
        mkfifo(FIFO_PATH, 0666);

        // Fork a child process
        pid = fork();

        if (pid < 0) {
        perror("fork");
        exit(EXIT_FAILURE);
        }

        if (pid > 0) { // Parent process
        printf("Parent: Opening FIFO for writing...\n");
        // Open the FIFO for writing
        fd = open(FIFO_PATH, O_WRONLY);

        printf("Parent: Writing data to the FIFO...\n");
        // Write data to the FIFO
        write(fd, "Hello, child!", 14);

        close(fd); // Close the FIFO
```

```
    printf("Parent: Data written to the FIFO.\n");
    } else { // Child process
    printf("Child: Opening FIFO for reading...\n");
    // Open the FIFO for reading
    fd = open(FIFO_PATH, O_RDONLY);

    printf("Child: Reading data from the FIFO...\n");
    // Read data from the FIFO
    read(fd, buffer, BUFFER_SIZE);
    printf("Child: Received message: %s\n", buffer);

    close(fd); // Close the FIFO
    }

    return 0;
}
```

57. Write a program using PIPE, to Send file from parent to child over a pipe. (unnamed pipe)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define BUFFER_SIZE 1024

int main() {
    int pipefd[2]; // File descriptors for the pipe
    pid_t pid; // Process ID

    char buffer[BUFFER_SIZE]; // Buffer for reading from and writing to the pipe

    // Create the pipe
    if (pipe(pipefd) == -1) {
    perror("pipe");
    exit(EXIT_FAILURE);
    }
```

```
// Fork a child process
pid = fork();

if (pid < 0) {
perror("fork");
exit(EXIT_FAILURE);
}

if (pid > 0) { // Parent process
close(pipefd[0]); // Close the reading end of the pipe in the parent

printf("Parent: Reading file...\n");
FILE *file = fopen("input.txt", "r");
if (file == NULL) {
perror("fopen");
exit(EXIT_FAILURE);
}

printf("Parent: Writing file data to the pipe...\n");
ssize_t bytes_read;
while ((bytes_read = fread(buffer, 1, sizeof(buffer), file)) > 0) {
write(pipefd[1], buffer, bytes_read);
}
close(pipefd[1]); // Close the writing end of the pipe in the parent
fclose(file);
printf("Parent: File data written to the pipe.\n");
} else { // Child process
close(pipefd[1]); // Close the writing end of the pipe in the child

printf("Child: Reading data from the pipe...\n");
FILE *output_file = fopen("output.txt", "w");
if (output_file == NULL) {
perror("fopen");
exit(EXIT_FAILURE);
}

ssize_t bytes_read;
while ((bytes_read = read(pipefd[0], buffer, sizeof(buffer))) > 0) {
fwrite(buffer, 1, bytes_read, output_file);
}
```

```
        fclose(output_file);
        printf("Child: Data written to output.txt.\n");

        close(pipefd[0]); // Close the reading end of the pipe in the child
        }

        return 0;
}
```

58. Write a program using FIFO, to Send file from parent to child over a pipe. (named pipe)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define FIFO_PATH "/tmp/myfifo"
#define BUFFER_SIZE 1024

int main() {
        int fd; // File descriptor for the FIFO
        pid_t pid; // Process ID

        char buffer[BUFFER_SIZE]; // Buffer for reading from and writing to the FIFO

        // Create the FIFO
        mkfifo(FIFO_PATH, 0666);

        // Fork a child process
        pid = fork();

        if (pid < 0) {
        perror("fork");
        exit(EXIT_FAILURE);
        }

        if (pid > 0) { // Parent process
        printf("Parent: Opening file for reading...\n");
```

```
FILE *file = fopen("input.txt", "r");
if (file == NULL) {
perror("fopen");
exit(EXIT_FAILURE);
}

printf("Parent: Opening FIFO for writing...\n");
// Open the FIFO for writing
fd = open(FIFO_PATH, O_WRONLY);
if (fd == -1) {
perror("open");
exit(EXIT_FAILURE);
}

printf("Parent: Writing file data to the FIFO...\n");
ssize_t bytes_read;
while ((bytes_read = fread(buffer, 1, sizeof(buffer), file)) > 0) {
write(fd, buffer, bytes_read);
}
close(fd); // Close the FIFO
fclose(file);
printf("Parent: File data written to the FIFO.\n");
} else { // Child process
printf("Child: Opening FIFO for reading...\n");
// Open the FIFO for reading
fd = open(FIFO_PATH, O_RDONLY);
if (fd == -1) {
perror("open");
exit(EXIT_FAILURE);
}

printf("Child: Creating file for writing...\n");
FILE *output_file = fopen("output.txt", "w");
if (output_file == NULL) {
perror("fopen");
exit(EXIT_FAILURE);
}

printf("Child: Reading data from the FIFO...\n");
ssize_t bytes_read;
```

```
    while ((bytes_read = read(fd, buffer, sizeof(buffer))) > 0) {
    fwrite(buffer, 1, bytes_read, output_file);
    }
    fclose(output_file);
    printf("Child: Data written to output.txt.\n");

    close(fd); // Close the FIFO
    }

    return 0;
}
```

59. Write a program using PIPE, to convert uppercase to lowercase filter to read command/from file

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

#define BUFFER_SIZE 1024

int main() {
    int pipefd[2]; // File descriptors for the pipe
    pid_t pid; // Process ID

    // Create the pipe
    if (pipe(pipefd) == -1) {
    perror("pipe");
    exit(EXIT_FAILURE);
    }

    // Fork a child process
    pid = fork();

    if (pid < 0) {
    perror("fork");
    exit(EXIT_FAILURE);
    }

    if (pid > 0) { // Parent process (reads from file)
    close(pipefd[0]); // Close the reading end of the pipe in the parent

    printf("Parent: Reading from file and writing to pipe...\n");
    FILE *file = fopen("input.txt", "r");
    if (file == NULL) {
    perror("fopen");
    exit(EXIT_FAILURE);
    }

    char buffer[BUFFER_SIZE];
    ssize_t bytes_read;
    while ((bytes_read = fread(buffer, 1, sizeof(buffer), file)) > 0) {
    write(pipefd[1], buffer, bytes_read);
    }
    close(pipefd[1]); // Close the writing end of the pipe in the parent
    fclose(file);
    } else { // Child process (converts uppercase to lowercase)
    close(pipefd[1]); // Close the writing end of the pipe in the child

    printf("Child: Converting uppercase to lowercase...\n");
    char buffer[BUFFER_SIZE];
    ssize_t bytes_read;
    while ((bytes_read = read(pipefd[0], buffer, sizeof(buffer))) > 0) {
    for (int i = 0; i < bytes_read; ++i) {
        if (buffer[i] >= 'A' && buffer[i] <= 'Z') {
        buffer[i] = buffer[i] + 32; // Convert uppercase to lowercase
        }
    }
    write(STDOUT_FILENO, buffer, bytes_read); // Write to standard output
    }
    close(pipefd[0]); // Close the reading end of the pipe in the child
    }

    return 0;
}
```

60. Write a program to illustrate the semaphore concept. Use fork so that 2 process running simultaneously and communicate via semaphore. (give diff between sem.h/semaphore.h)

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

int main() {
	// Create a semaphore and initialize it to 1
	sem_t *sem = sem_open("/my_semaphore", O_CREAT, 0644, 1);
	if (sem == SEM_FAILED) {
	perror("sem_open");
	exit(EXIT_FAILURE);
	}

	// Fork a child process
	pid_t pid = fork();
	if (pid < 0) {
	perror("fork");
	exit(EXIT_FAILURE);
	} else if (pid == 0) { // Child process
	printf("Child process trying to lock the semaphore...\n");
	sem_wait(sem); // Lock the semaphore
	printf("Child process locked the semaphore.\n");
	sleep(2); // Simulate some work
	sem_post(sem); // Unlock the semaphore
	printf("Child process released the semaphore.\n");
	} else { // Parent process
	printf("Parent process trying to lock the semaphore...\n");
	sem_wait(sem); // Lock the semaphore
	printf("Parent process locked the semaphore.\n");
	sleep(2); // Simulate some work
	sem_post(sem); // Unlock the semaphore
	printf("Parent process released the semaphore.\n");
	}

	// Close and unlink the semaphore
	sem_close(sem);
```

```
    sem_unlink("/my_semaphore");

    return 0;
}
```

The difference between `sem.h` and `semaphore.h` lies in their functionality, portability, and underlying implementations. Here's a comparison between the two:

`sem.h` (System V Semaphores):

1. **Functionality:**
   - Provides functions like `semget`, `semop`, and `semctl` for semaphore creation, operation, and control.
   - Allows for the creation of System V semaphores, which are managed using unique semaphore identifiers (semid).
2. **Portability:**
   - `sem.h` is part of the System V IPC (Inter-Process Communication) mechanisms.
   - It may not be available on all systems, especially those that follow POSIX standards exclusively.
3. **Usage:**
   - Commonly used in older UNIX systems and legacy codebases.
   - Not as widely supported across different platforms compared to POSIX semaphores.

`semaphore.h` (POSIX Semaphores):

1. **Functionality:**
   - Provides functions like `sem_init`, `sem_wait`, `sem_post`, and `sem_destroy` for semaphore management.
   - Allows for the creation of POSIX semaphores, which are managed using pointers to sem_t structures.
2. **Portability:**
   - `semaphore.h` is part of the POSIX standard (POSIX.1-2001 and later revisions).
   - It is more widely supported across different UNIX-like systems and modern operating systems.
3. **Usage:**
   - Preferred choice for new projects and modern codebases due to its standardization and portability.

    - Offers a more consistent and intuitive interface compared to System V semaphores.

Summary:

- `sem.h` is associated with S
- ystem V semaphores and provides functions for semaphore management specific to System V IPC mechanisms. It is less portable and not as widely used in modern applications.
- `semaphore.h` is part of the POSIX standard and provides a more modern and portable interface for semaphore management. It is the preferred choice for new projects and is widely supported across different platforms

61. Write 3 programs separately, 1st program will initialize the semaphore and display the semaphore ID. 2nd program will perform the P operation and print message accordingly. 3rd program will perform the V operation print the message accordingly for the same semaphore declared in the 1st program.

1)

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>



int main() {

    key_t key;

    int sem_id;



    // Generate a unique key

    if ((key = ftok(".", 'S')) == -1) {
```

```
    perror("ftok");
    exit(EXIT_FAILURE);
    }

    // Create a semaphore with key and initial value 1
    if ((sem_id = semget(key, 1, IPC_CREAT | IPC_EXCL | 0666)) == -1) {
    perror("semget");
    exit(EXIT_FAILURE);
    }
  printf("Semaphore initialized with ID: %d\n", sem_id);

    return 0;
}
```

2nd:

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

int main() {
    key_t key;
}

int sem_id;
struct sembuf sem_op;

// Get the key
if ((key = ftok(".", 'S')) == -1) {
perror("ftok");
exit(EXIT_FAILURE);
}

// Get the semaphore ID
if ((sem_id = semget(key, 1, 0)) == -1) {
perror("semget");
exit(EXIT_FAILURE);
}

// Perform the P operation
sem_op.sem_num = 0;
sem_op.sem_op = -1;
sem_op.sem_flg = 0;
if (semop(sem_id, &sem_op, 1) == -1) {
perror("semop");
exit(EXIT_FAILURE);
}
```

```
        printf("P operation performed successfully\n");

        return 0;
}
```

3

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

int main() {
        key_t key;
        int sem_id;
        struct sembuf sem_op;

        // Get the key
        if ((key = ftok(".", 'S')) == -1) {
        perror("ftok");
        exit(EXIT_FAILURE);
        }
        // Get the semaphore ID

        if ((sem_id = semget(key, 1, 0)) == -1) {

        perror("semget");

        exit(EXIT_FAILURE);

        }


        // Perform the V operation

        sem_op.sem_num = 0;

        sem_op.sem_op = 1;

        sem_op.sem_flg = 0;

        if (semop(sem_id, &sem_op, 1) == -1) {

        perror("semop");

        exit(EXIT_FAILURE);

        }

    printf("V operation performed successfully\n");

    return 0;

}
```

How to run:

```
snehal@snehal-HP-Notebook:~/UOS$ ./61p1

Semaphore initialized with ID: 13

snehal@snehal-HP-Notebook:~/UOS$ ./61p2

^C
```

```
snehal@snehal-HP-Notebook:~/UOS$ ./61p3

V operation performed successfully

snehal@snehal-HP-Notebook:~/UOS$ ./61p2

P operation performed successfully
```

62. Write a program to demonstrate the lockf system call for locking.

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

int main() {
        int fd;
        char *filename = "example.txt";
        char data[] = "This is a test message.\n";

        // Open or create the file
        fd = open(filename, O_WRONLY | O_CREAT, 0644);
        if (fd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
        }

        // Lock a portion of the file
        printf("Locking file...\n");
        if (lockf(fd, F_LOCK, 0) == -1) {
        perror("lockf");
        exit(EXIT_FAILURE);
        }

        // Write to the file
        printf("Writing to locked file...\n");
        if (write(fd, data, sizeof(data)) == -1) {
        perror("write");
        exit(EXIT_FAILURE);
        }

        // Release the lock
```

```
    printf("Releasing lock...\n");
    if (lockf(fd, F_ULOCK, 0) == -1) {
    perror("lockf");
    exit(EXIT_FAILURE);
    }

    // Close the file
    printf("Closing file...\n");
    if (close(fd) == -1) {
    perror("close");
    exit(EXIT_FAILURE);
    }

    printf("Program completed successfully.\n");

    return 0;
}
```

63. Write a program to demonstrate the flock system call for locking.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/file.h>

int main() {
    int fd;
    char *filename = "example.txt";
    char data[] = "This is a test message.\n";

    // Open or create the file
    fd = open(filename, O_WRONLY | O_CREAT, 0644);
    if (fd == -1) {
    perror("open");
    exit(EXIT_FAILURE);
    }

    // Lock a portion of the file
    printf("Locking file...\n");
    if (flock(fd, LOCK_EX) == -1) {
```

```
    perror("flock");
    exit(EXIT_FAILURE);
    }

    // Write to the file
    printf("Writing to locked file...\n");
    if (write(fd, data, sizeof(data)) == -1) {
    perror("write");
    exit(EXIT_FAILURE);
    }

    // Release the lock
    printf("Releasing lock...\n");
    if (flock(fd, LOCK_UN) == -1) {
    perror("flock");
    exit(EXIT_FAILURE);
    }

    // Close the file
    printf("Closing file...\n");
    if (close(fd) == -1) {
    perror("close");
    exit(EXIT_FAILURE);
    }

    printf("Program completed successfully.\n");

    return 0;
}
```

64. Using FIFO as named pipe use read and write system calls to establish communication (IPC) between two ends.

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
```

```
{
        int fd;

        // FIFO file path
        char * myfifo = "/tmp/myfifo";

        // Creating the named file(FIFO)
        // mkfifo(<pathname>, <permission>)
        mkfifo(myfifo, 0666);

        char arr1[80], arr2[80];
        while (1)
        {
        // Open FIFO for write only
        fd = open(myfifo, O_WRONLY);

        // Take an input arr2ing from user.
        // 80 is maximum length
        fgets(arr2, 80, stdin);

        // Write the input arr2ing on FIFO
        // and close it
        write(fd, arr2, strlen(arr2)+1);
        close(fd);

        // Open FIFO for Read only
        fd = open(myfifo, O_RDONLY);

        // Read from FIFO
        read(fd, arr1, sizeof(arr1));

        // Print the read message
        printf("User2: %s\n", arr1);
        close(fd);
        }
        return 0;
}
```

Writes first

READ

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
        int fd1;

        // FIFO file path
        char * myfifo = "/tmp/myfifo";

        // Creating the named file(FIFO)
        // mkfifo(<pathname>,<permission>)
        mkfifo(myfifo, 0666);

        char str1[80], str2[80];
        while (1)
        {
        // First open in read only and read
        fd1 = open(myfifo,O_RDONLY);
        read(fd1, str1, 80);

        // Print the read string and close
        printf("User1: %s\n", str1);
        close(fd1);

        // Now open in write mode and write
        // string taken from user.
        fd1 = open(myfifo,O_WRONLY);
        fgets(str2, 80, stdin);
        write(fd1, str2, strlen(str2)+1);
        close(fd1);
        }
        return 0;
}
```

```
snehal@snehal-HP-Notebook:~/UOS$ cc -o 64write 64write.c
snehal@snehal-HP-Notebook:~/UOS$ ./64write

nehal@snehal-HP-Notebook:~/UOS$ cc -o 64read 64read.c
snehal@snehal-HP-Notebook:~/UOS$ ./64read
```

65. write shell script with FIFO/mknod (named pipe) us for communication (IPC)

```
#!/bin/bash

# Define the FIFO name
FIFO_NAME="myfifo"

# Create the FIFO (named pipe)
if [ ! -p "$FIFO_NAME" ]; then
        mkfifo "$FIFO_NAME"
fi

# Function to read data from the FIFO
read_from_fifo() {
        echo "Reader process is listening for messages..."
        while read line < "$FIFO_NAME"; do
        echo "Received message: $line"
        done
}

# Function to write data to the FIFO
write_to_fifo() {
        echo "Enter message to send (or 'exit' to quit):"
        while true; do
        read message
        if [ "$message" = "exit" ]; then
        break
        fi
        echo "$message" > "$FIFO_NAME"
        done
}

# Check if the script is run with an argument
```

```
if [ $# -ne 1 ]; then
	echo "Usage: $0 [read|write]"
	exit 1
fi

# Check the argument and run the appropriate function
case "$1" in
	read) read_from_fifo ;;
	write) write_to_fifo ;;
	*) echo "Invalid argument: $1"; exit 1 ;;
esac
```

chmod +x fifo.sh
./fifo.sh write
./fifo.sh read

66.write prog using FIFO/mknod (named pipe)/unmanned pipe for uppercase to lowercase conversion

```
#include <iostream>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <cstring>
#include <cctype>
#include <cstdlib>

using namespace std;

const char *inputFifo = "/tmp/input_fifo";
const char *outputFifo = "/tmp/output_fifo";

int main()
{
  // Create the input FIFO (named pipe) if it doesn't exist
  if (mkfifo(inputFifo, 0666) == -1)
  {
	cerr << "Error creating input FIFO" << endl;
	exit(EXIT_FAILURE);
  }
```

```
// Create the output FIFO (named pipe) if it doesn't exist
if (mkfifo(outputFifo, 0666) == -1)
{
    cerr << "Error creating output FIFO" << endl;
    exit(EXIT_FAILURE);
}

// Open the input FIFO for reading
int inputFd = open(inputFifo, O_RDONLY);
if (inputFd == -1)
{
    cerr << "Error opening input FIFO for reading" << endl;
    exit(EXIT_FAILURE);
}

// Open the output FIFO for writing
int outputFd = open(outputFifo, O_WRONLY);
if (outputFd == -1)
{
    cerr << "Error opening output FIFO for writing" << endl;
    exit(EXIT_FAILURE);
}

char buffer[256];
ssize_t bytesRead;

// Read from the input FIFO and convert to lowercase
while ((bytesRead = read(inputFd, buffer, sizeof(buffer))) > 0)
{
    for (ssize_t i = 0; i < bytesRead; ++i)
    {
    buffer[i] = tolower(buffer[i]);
    }

    // Write the converted text to the output FIFO
    if (write(outputFd, buffer, bytesRead) == -1)
    {
    cerr << "Error writing to output FIFO" << endl;
    exit(EXIT_FAILURE);
    }
```

```
  }

  // Close the FIFOs
  close(inputFd);
  close(outputFd);

  // Remove the FIFOs
  unlink(inputFifo);
  unlink(outputFifo);

  cout << "Conversion complete." << endl;

  return 0;
}
```

How to run:

```
// rm /tmp/input_fifo /tmp/output_fifo
// g++ 66.cpp -o uppercase_to_lowercase
// ./uppercase_to_lowercase
// On New Terminal
// echo "HELLO" >/tmp/input_fifo
// cat /tmp/output_fifo
// rm /tmp/input_fifo /tmp/output_fifo
```