TY(IT)
Unix Operating System
Probable Assignment List for External Practical Exam
1. Write a program to use fork system call to create 5 child processes and assign 5 operations to Childs.

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


void performOP(int childId){
   switch(childId){
          case 0:
                 printf("ChildID square %d\n",childId*childId);
                 break;
          case 1:
                 printf("ChildID cube %d\n", childId*childId*childId);
                 break;
          case 2:
                 printf("ChildID double %d\n", childId*2);
                 break;
          case 3:
                 printf("ChildID triple %d\n", childId*3);
                 break;
          case 4:
                 printf("ChildID four time %d\n", childId*4);
                 break;
          case 5:
                 printf("ChildID five time %d\n", childId*5);
                 break;
          default:
                 break;
   }
}


int main(){
   int childId;
   pid_t pid;

   for(childId = 0; childId < 5; childId++){
          pid = fork();

          if(pid<0){
```

```
                printf("Fork failed\n");
                return 1;
        } else if (pid == 0){
                performOP(childId);
                return 0;
        }
    }

    for(childId = 0; childId < 5; childId++){
        wait(NULL);
    }
}
```

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>

int main(){

    pid_t pid;
    char login[20];
    char password[20];

    pid = vfork();

    if(pid < 0){
        printf("Fork failed");
        return 1;
    }else if(pid == 0){
        printf("child process\n");
        printf("Enter login name: ");
        scanf("%s",login);
        printf("Login name: %s\n",login);
        exit(0);
    }else{
        printf("Parent process\n");
        printf("Enter password: ");
        scanf("%s",password);
        printf("Password: %s\n",password);
    }

    return 0;
```

```
}
```
3. Write a program to open any application using a fork system call.

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

int main(){

   pid_t pid;

   pid = fork();

   if(pid==0){
           char *arg[] = {"/usr/bin/gedit", "ramu", NULL};
           execvp(arg[0], arg);
   }else if(pid>0){
           printf("Parent Process");
   }else{
           printf("Error in child creation");
   }

   return 0;
}
```
4. Write a program to open any application using the vfork system call.

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

int main(){

   pid_t pid;

   pid = vfork();

   if(pid==0){
           char *arg[] = {"/usr/bin/gedit","ramu", NULL};
           execvp(arg[0], arg);
   }else if(pid > 0){
           printf("Parent Process");
   }else{
           printf("Error in child creation");
   }
```

```
    return 0;
}
```
5. Write a program to demonstrate the wait use with fork sysem call.
```
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>

int main(){
    pid_t pid;
    int status;

    pid = fork();

    if(pid < 0){
            printf("Fork failed");
            return 1;
    }else if (pid ==0){
            printf("Parent Id %d and Id %d\n",getppid(),getpid());
            sleep(3);
            printf("child process: Exiting\n");
            return 42;
    }else{
            printf("Process Id %d and child Id %d\n", getpid(),pid);
            wait(&status);
            if(WIFEXITED(status)){
                    int exit_status = WEXITSTATUS(status);
                    printf("Parent process: child process exit normally\n");
            }else{
                    printf("Parent process: child process did not exit normally\n");
            }
    }

    return 0;
}
```
6. Write a program to demonstrate the variations exec system call.
```
#include <stdio.h>
#include <unistd.h>


int main(){
    //using execv()
    char* args[] = {"ls", "-l",NULL};
    execv("/bin/ls", args);
    printf("This line will not be executed if execv() is succesfull");
```

```
    //execvp()
    char* cmd = "ls";
    char* argv[] = {"ls", "-l", NULL};
    execvp(cmd, argv);
    printf("This line will not be executed if execvp() is successfull");

    //execve();
    char* envp[] = {"HOME=/home/user","PATH=/usr/bin",NULL};
    execve("/bin/ls",argv,envp);
    printf("This line will not be executed if execve() is successful.\n");

    perror("exec failed");

    return 1;
}
```

The full forms of the mentioned exec() functions are as follows:
execl: Execute a file with a specified path using a list of arguments. The 'l' in execl stands for "list".
execv: Execute a file with a specified path using an array of arguments. The 'v' in execv stands for "vector".
execle: Execute a file with a specified path using a list of arguments and environment variables. The 'le' in execle stands for "list with the environment".
execvp: Execute a file with a specified name, searching for the file in the directories listed in the PATH environment variable. The 'vp' in execvp stands for "vector with path".
These functions are part of the exec() function family in C, and their names provide a hint about their behavior and usage.

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

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

int main(){
    pid_t pid;
    int status;

    pid = fork();

    if(pid < 0){
            printf("Fork failed");
            return 1;
    }else if (pid ==0){
```

```
            printf("Parent Id %d and Id %d\n",getppid(),getpid());
            sleep(3);
            printf("child process: Exiting\n");
            exit(5);
    }else{
            printf("Process Id %d and child Id %d\n", getpid(),pid);
            wait(&status);
            if(WIFEXITED(status)){
                    int exit_status = WEXITSTATUS(status);
                    printf("Parent process: child process exit normally\n");
            }else{
                    printf("Parent process: child process did not exit normally\n");
            }
    }

    return 0;
}
```

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

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

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

int main()
{
        pid_t pid1, pid2;

        // Fork the first child process
        pid1 = fork();

        if (pid1 < 0) {
        // Fork failed
        printf("Forking first child process failed\n");
        return 1;
        } else if (pid1 == 0) {
        // First child process
        printf("First child process with PID: %d\n", getpid());

        // Register signal handler
```

```
        signal(SIGUSR1, signal_handler);

        while (1) {
        sleep(1);
        }
        } else {
        // Parent process
        printf("Parent process with PID: %d\n", getpid());

        // Fork the second child process
        pid2 = fork();

        if (pid2 < 0) {
        // Fork failed
        printf("Forking second child process failed\n");
        return 1;
        } else if (pid2 == 0) {
        // Second child process
        printf("Second child process with PID: %d\n", getpid());

        // Register signal handler
        signal(SIGUSR1, signal_handler);

        // Wait for a moment to ensure first child process is set up
        sleep(1);

        // Send a signal from the second child process to the first child process
        printf("Second child process with PID %d sending SIGTERM signal to the first child process
with PID %d\n", getpid(), pid1);
        kill(pid1, SIGTERM);
        } else {
        // Parent process

        // Wait for a moment to ensure child processes are set up
        sleep(1);

        // Wait for the second child process to terminate
        //waitpid(pid2, NULL, 0);

        // Terminate the first child process
        printf("Terminating the first child process with PID %d\n", pid1);
        kill(pid1, SIGTERM);
        }
        }
```

```
	return 0;
}
```

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

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

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

int main()
{
	pid_t pid;

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

	if (pid < 0) {
	// Fork failed
	printf("Forking child process failed\n");
	return 1;
	} else if (pid == 0) {
	// Child process
	printf("Child process with PID: %d\n", getpid());

	// Register signal handler
	signal(SIGUSR1, signal_handler);

	while (1) {
	sleep(1);
	}
	} else {
	// Parent process
	// Wait for a moment to ensure child process and unrelated process are set up
	sleep(1);

	// Send a signal to the child process
```

```
        printf("Sending SIGUSR1 signal to child process with PID: %d\n", pid);
        kill(pid, SIGUSR1);

        // Wait for a moment to allow child process to handle the signal
        sleep(1);

        // Terminate the child process
        printf("Terminating child process\n");
        kill(pid, SIGTERM);


        }
        return 0;
        }
```

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

```
#include <signal.h> // library for signal handling
#include <stdio.h>  // library for input and output functions
#include <unistd.h> // library for sleep function
#include <stdbool.h>// library for boolean datatype
#include <stdlib.h> // library for exit function

bool flag = false; // boolean variable to be used later

// Signal handler function to be called when SIGALRM is triggered
void alarmhandle(int sig){
        printf("Input time expired\n"); // print the message to the console
        exit(1); // exit program with status code 1
}

// Main function
int main()
{
        int a = 0; // variable to store user input
        printf("Input now in 10 seconds\n"); // print the message to the console
        sleep(1); // sleep for 1 second
        alarm(10); // set an alarm that will trigger in 10 seconds
        signal(SIGALRM, alarmhandle); // register the signal handler function with SIGALR
        scanf("%d", &a); // read integer input from the console and store in variable a
        printf("You entered %d\n", a); // print the user input to the console
        return 0; // exit program with status code 0
}
```

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

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

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

int main()
{
	pid_t pid;

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

	if (pid < 0) {
	// Fork failed
	printf("Forking child process failed\n");
	return 1;
	} else if (pid == 0) {
	// Child process
	printf("Child process with PID: %d\n", getpid());

	// Register signal handler
	signal(SIGUSR1, signal_handler);

	while (1) {
	sleep(1);
	}
	} else {
	// Parent process
	// Wait for a moment to ensure child process and unrelated process are set up
	sleep(1);

	// Send a signal to the child process
	printf("Sending SIGUSR1 signal to child process with PID: %d\n", pid);
	kill(pid, SIGUSR1);

	// Wait for a moment to allow child process to handle the signal
```

```
        sleep(1);

        // Terminate the child process
        printf("Terminating child process\n");
        kill(pid, SIGTERM);


        }
        return 0;
        }
```

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

```
#include <signal.h> // library for signal handling
#include <stdio.h>  // library for input and output functions
#include <unistd.h> // library for sleep function
#include <stdbool.h>// library for boolean datatype
#include <stdlib.h> // library for exit function

bool flag = false; // boolean variable to be used later

// Signal handler function to be called when SIGALRM is triggered
void alarmhandle(int sig){
        printf("Input time expired\n"); // print the message to the console
        exit(1); // exit program with status code 1
}

// Main function
int main()
{
        int a = 0; // variable to store user input
        printf("Input now in 10 seconds\n"); // print the message to the console
        sleep(1); // sleep for 1 second
        alarm(10); // set an alarm that will trigger in 10 seconds
        signal(SIGALRM, alarmhandle); // register the signal handler function with SIGALR
        scanf("%d", &a); // read integer input from the console and store in variable a
        printf("You entered %d\n", a); // print the user input to the console
        return 0; // exit program with status code 0
}
```

12. Write a program to give statistics of a given file using stat system call.

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

```
int main(int argc, char **argv)
{
   // Declare a struct to store file information
   struct stat fileStat;

   // Call the stat function to get information about the file
   // specified by the path "/home/sumit/Documents/UOS/abc.txt"
   if(stat("/home/lac-it/Documents/UOS/abc.txt", &fileStat) < 0) {
      printf("Failed to get file information\n");
      return 1;
   }

   // Print file information
   printf("--------------------------\n");
   printf("File Size: \t\t%ld bytes\n", (long)fileStat.st_size);
   printf("Number of Links: \t%ld\n", (long)fileStat.st_nlink);
   printf("File inode: \t\t%ld\n", (long)fileStat.st_ino);

   // Print file permissions
   printf("File Permissions: \t");
   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" : "-");
   printf("\n\n");

   // Check if the file is a symbolic link
   printf("The file %s a symbolic link\n", (S_ISLNK(fileStat.st_mode)) ? "is" : "is not");

   return 0;
}
```

13. Write a program to give statistics of a given file using fstat system call.

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

```
#include <unistd.h>

int main(int argc, char *argv[])
{
   int fd;              // file descriptor for the file to be opened
   struct stat st;      // struct to store information about the file

   if (argc != 2) {
      fprintf(stderr, "Usage: %s <file>\n", argv[0]);
      return 1;
   }

   // Open the file
   fd = open(argv[1], O_RDONLY);
   if (fd == -1) {
      perror("open");  // print an error message if the file could not be opened
      return 1;
   }

   // Retrieve information about the file using fstat
   if (fstat(fd, &st) == -1) {
      perror("fstat"); // print an error message if fstat call fails
      return 1;
   }

   // Print information about the file
   printf("File type: ");
   switch (st.st_mode & S_IFMT) {    // extract file type from st_mode field
      case S_IFREG:  printf("regular file\n");    break;   // regular file
      case S_IFDIR:  printf("directory\n");       break;   // directory
      case S_IFLNK:  printf("symbolic link\n");   break;   // symbolic link
      default:       printf("unknown\n");         break;   // unknown file type
   }
   printf("Size: %ld bytes\n", st.st_size);                // print file size
   printf("Permissions: %o\n", st.st_mode & 07777);        // print file permissions
   printf("Last modified: %s", ctime(&st.st_mtime));       // print last modified time
   close(fd); // Close the file
   return 0;
}
```

14. Write a program to convert pathname to Inode using stat system call

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

```
#include <sys/types.h>
#include <sys/stat.h>

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <pathname>\n", argv[0]);
        return 1;
    }

    const char *pathname = argv[1];
    struct stat fileStat;

    if (stat(pathname, &fileStat) == -1) {
        perror("stat");
        return 1;
    }

    printf("Inode number of %s: %ld\n", pathname, (long) fileStat.st_ino);

    return 0;
}
```

15. Write a program to convert pathname to Inode using ‘ls’ command.

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

#define MAX_BUFFER_SIZE 1024

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <pathname>\n", argv[0]);
        return 1;
    }

    const char *pathname = argv[1];
    char command[MAX_BUFFER_SIZE];
    sprintf(command, "ls -i \"%s\"", pathname);

    FILE *fp = popen(command, "r");
    if (fp == NULL) {
        perror("popen");
        return 1;
```

```
    }

    char output[MAX_BUFFER_SIZE];
    if (fgets(output, sizeof(output), fp) != NULL) {
        char *token = strtok(output, " ");
        if (token != NULL) {
            printf("Inode number of %s: %s\n", pathname, token);
        } else {
            fprintf(stderr, "Unable to retrieve inode number.\n");
        }
    } else {
        fprintf(stderr, "Error reading output from command.\n");
    }

    pclose(fp);

    return 0;
}
```

16. Write a multithreaded program in JAVA for chatting.

ChatServer.java

```
// Common Serve Code


import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

public class ChatServer {

    private static List<ClientThread> clients = new ArrayList<>();

    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(8000);
        System.out.println("Chat server is listening on port 8000...");

        while (true) {
            Socket socket = serverSocket.accept();
```

```
            System.out.println("A new client has connected: " + socket);
            ClientThread client = new ClientThread(socket);
            client.start();
            clients.add(client);
        }
    }

    private static class ClientThread extends Thread {
        private Socket socket;
        private DataInputStream input;
        private DataOutputStream output;

        ClientThread(Socket socket) {
            this.socket = socket;
        }

        @Override
        public void run() {
            try {
                input = new DataInputStream(socket.getInputStream());
                output = new DataOutputStream(socket.getOutputStream());
                String name = input.readUTF();
                broadcast("[" + name + " has joined the chat room]");

                while (true) {
                    String message = input.readUTF();
                    broadcast(name + ": " + message);
                }
            } catch (IOException e) {
                System.out.println("A client has disconnected: " + socket);
                clients.remove(this);
                broadcast("[" + socket + " has left the chat room]");
            }
        }

        private void broadcast(String message) {
            for (ClientThread client : clients) {
                try {
                    client.output.writeUTF(message);
                } catch (IOException e) {
                    System.out.println("Error broadcasting message to " + client.socket);
                }
            }
        }
```

```
    }
}
```

ChatClient.java

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.Scanner;

public class ChatClient {

    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 8000);
            System.out.println("Connected to the chat server.");

            DataInputStream input = new DataInputStream(socket.getInputStream());
            DataOutputStream output = new DataOutputStream(socket.getOutputStream());

            // Read user's name
            Scanner scanner = new Scanner(System.in);
            System.out.print("Enter your name: ");
            String name = scanner.nextLine();

            // Send name to the server
            output.writeUTF(name);

            // Start a thread to receive messages from the server
            Thread receiveThread = new Thread(() -> {
                try {
                    while (true) {
                        String message = input.readUTF();
                        System.out.println(message);
                    }
                } catch (IOException e) {
                    System.out.println("Lost connection to the chat server.");
                    System.exit(0);
                }
            });
            receiveThread.start();

            // Send messages to the server
            while (true) {
```

```
            String message = scanner.nextLine();
            output.writeUTF(message);
        }
    } catch (IOException e) {
        System.out.println("Could not connect to the chat server.");
    }
  }
}
```

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

ThreadDemo.java

```
public class ThreadDemo {

   public static void main(String[] args) {

   // Creating three threads

      Thread evenThread = new Thread(new EvenThread());
      Thread oddThread = new Thread(new OddThread());
      Thread primeThread = new Thread(new PrimeThread());

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

  // Thread to print Even Numbers
   static class EvenThread implements Runnable {

      @Override
      public void run() {
         for (int i = 0; i <= 25; i += 2) {
            System.out.println("EvenThread: " + i);
         }
      }
   }

  // Thread to print Odd Numbers
   static class OddThread implements Runnable {

      @Override
      public void run() {
```

```
            for (int i = 1; i <= 25; i += 2) {
                System.out.println("OddThread: " + i);
            }
        }
    }

    // Thread to print Prime Numbers
    static class PrimeThread implements Runnable {

        @Override
        public void run() {
            for (int i = 2; i <= 25; i++) {


                boolean isPrime = true;
                for (int j = 2; j < i; j++) {
                    if (i % j == 0) {
                        isPrime = false;
                        break;
                    }
                }
                if (isPrime) {
                    System.out.println("PrimeThread: " + i);
                }
            }
        }
    }
}
```

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

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

#define NUM_THREADS 5

void *threadFunction(void *threadId) {
  long tid = (long)threadId;
```

```
  printf("Hello from thread %ld\n", tid);
  pthread_exit(NULL);
}

int main() {
  pthread_t threads[NUM_THREADS];
  int i;

  for (i = 0; i < NUM_THREADS; i++) {
    int result = pthread_create(&threads[i], NULL, threadFunction, (void *)i);
    if (result) {
      printf("Error creating thread. Return code: %d\n", result);
      return -1;
    }
  }

  pthread_exit(NULL);
}
```

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

```
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

class Producer implements Runnable {
    private BlockingQueue<Integer> buffer; // Shared buffer
    private int maxSize; // Maximum size of the buffer

    public Producer(BlockingQueue<Integer> buffer, int maxSize) {
        this.buffer = buffer;
        this.maxSize = maxSize;
    }

    public void run() {
        try {
            for (int i = 0; i < maxSize; i++) {
                System.out.println("Producing: " + i);
                buffer.put(i); // Add item to the buffer
                Thread.sleep(1000); // Simulating some work
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
```

```
        }
    }
}

class Consumer implements Runnable {
    private BlockingQueue<Integer> buffer; // Shared buffer

    public Consumer(BlockingQueue<Integer> buffer) {
        this.buffer = buffer;
    }

    public void run() {
        try {
            while (true) {
                int value = buffer.take(); // Take item from the buffer
                System.out.println("Consuming: " + value);
                Thread.sleep(2000); // Simulating some work
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}

public class ProducerConsumerExample {
    public static void main(String[] args) {
        BlockingQueue<Integer> buffer = new ArrayBlockingQueue<>(5); // Shared buffer with a
maximum size of 5
        int maxSize = 10; // Maximum number of items to produce

        // Create producer and consumer threads
        Thread producerThread = new Thread(new Producer(buffer, maxSize));
        Thread consumerThread = new Thread(new Consumer(buffer));

        // Start producer and consumer threads
        producerThread.start();
        consumerThread.start();
    }
}
```

20. Write a program to implement a shell script for calculator.

```
#!/bin/bash

# Function to perform addition
addition() {
   echo "Enter the first number: "
   read num1
   echo "Enter the second number: "
   read num2
   sum=$((num1 + num2))
   echo "The sum of $num1 and $num2 is $sum."
}


# Function to perform subtraction
subtraction() {
   echo "Enter the first number: "
   read num1
   echo "Enter the second number: "
   read num2
   diff=$((num1 - num2))
   echo "The difference between $num1 and $num2 is $diff."
}

# Function to perform multiplication
multiplication() {
   echo "Enter the first number: "
   read num1
   echo "Enter the second number: "
   read num2
   product=$((num1 * num2))
   echo "The product of $num1 and $num2 is $product."
}

# Function to perform division
division() {
   echo "Enter the numerator: "
   read num1
   echo "Enter the denominator: "
   read num2
   if [ $num2 -eq 0 ]
   then
      echo "Error: Division by zero."
   else
```

```
        quotient=$((num1 / num2))
        echo "The quotient of $num1 and $num2 is $quotient."
    fi
}



# Main program
echo "Calculator Menu:"
echo "1. Addition"
echo "2. Subtraction"
echo "3. Multiplication"
echo "4. Division"
echo "Enter your choice (1-4): "
read choice

case $choice in
    1) addition ;;
    2) subtraction ;;
    3) multiplication ;;
    4) division ;;
   *) echo "Error: Invalid choice." ;;
esac
```

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

```
#!/bin/bash

while true; do
    clear  # Clear the screen
    echo "Digital Clock"
    echo "----------------"
    echo $(date +"%T")  # Display the current time in HH:MM:SS format
    sleep 1  # Pause for 1 second
done
```

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

```
#!/bin/bash

# Define the target host or IP address to ping
target="www.google.com"
```

```
# Ping the target with a single packet and a timeout of 1 second
ping -c 1 -W 1 $target > /dev/null

# 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
```

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

```
#!/bin/bash
echo "enter size of array"
read n;
declare -a a;
for((i=0;i<n;i++))
 do
   read a[$i];
 done
for((i=0;i<n;i++))
do
 for((j=i+1;j<n;j++))
  do
    {
      if((a[i]>a[j]))
      then
      temp=${a[i]};
       a[$i]=${a[j]};
       a[$j]=$temp;
      fi
    }
done
done
for((i=0;i<n;i++))
do
echo ${a[i]}
done
```

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

```
#!/bin/bash

print_colored() {
  # Prints the given text in the specified color
  local text=$1
  local color_code=$2
  echo -e "\033[${color_code}m${text}\033[0m"
}

# Print "Hello World" in bold
echo -e "\033[1mHello World\033[0m"

# Print "Hello World" in blink effect
echo -e "\033[5mHello World\033[0m"

# Print "Hello World" in different colors
print_colored "Hello World" "31"  # Red color
print_colored "Hello World" "34"  # Blue color
```

25. Write a shell script to find whether given file exist or not.

```
#!/bin/bash

filename="$1"  # Get the filename as the first argument

if [ -e "$filename" ]; then
   echo "File '$filename' exists."
else
   echo "File '$filename' does not exist."
fi
```

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

```
#!/bin/bash

echo "Disk Partition Information:"
echo "---------------------------"

# Run the 'df' command to get disk partition information
df_output=$(df -h)

# Print the column headers
echo "$df_output" | awk 'NR==1 {print $1 "\t" $2 "\t" $3 "\t" $4 "\t" $5}'
```

```
# Print the disk partition details
echo "$df_output" | awk 'NR>1 {print $1 "\t" $2 "\t" $3 "\t" $4 "\t" $5}'

echo "---------------------------"
```

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

```
#!/bin/bash

filename="$1"  # Get the filename as the first argument

echo "Searching for file '$filename'..."

# Use the find command to search for the file
find_results=$(find / -name "$filename" 2>/dev/null)

if [ -n "$find_results" ]; then
   echo "File '$filename' found at the following locations:"
   echo "$find_results"
else
   echo "File '$filename' not found."
fi
```

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

```
#!/bin/bash

url="$1"  # Get the URL as the first argument
output_dir="$2"  # Get the output directory as the second argument

echo "Downloading webpage from: $url"

# Create the output directory if it doesn't exist
mkdir -p "$output_dir"

# Use the wget command to download the webpage
wget -P "$output_dir" "$url"

echo "Webpage downloaded successfully."



# "https://www.goolge.com" "/home/pandors"
```

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

```
#!/bin/bash

url="$1"  # Get the URL as the first argument
output_dir="$2"  # Get the output directory as the second argument

echo "Downloading webpage from: $url"

# Create the output directory if it doesn't exist
mkdir -p "$output_dir"

# Use the wget command to download the webpage
wget -P "$output_dir" "$url"

echo "Webpage downloaded successfully."



# "https://www.goolge.com" "/home/pandors"
```

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

```
#!/bin/bash

echo "Users on the System:"
echo "---------------------"

# Use the who command to get the list of users
who_output=$(who)

# Print the user information
echo "$who_output"

echo "---------------------"
```

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

```
def is_prime(number, divisor=2):
    # Base cases
    if number <= 1:
        return False
    if number == 2:
        return True
    if number % divisor == 0:
        return False
```

```
    if divisor * divisor > number:
        return True

    # Recursive case
    return is_prime(number, divisor + 1)


def find_prime_numbers(limit):
    prime_numbers = []
    for num in range(limit + 1):
        if is_prime(num):
            prime_numbers.append(num)
    return prime_numbers


# Example usage
input_limit = 50
primes = find_prime_numbers(input_limit)
print("Prime numbers up to", input_limit, ":", primes)
```

32. 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).

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

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

 ProducerConsumerSemaphore.java
class Producer implements Runnable {
    private Buffer buffer;
    private Semaphore mutex;
    private Semaphore empty;
    private Semaphore full;
    private int data;

    public Producer(Buffer buffer, Semaphore mutex, Semaphore empty, Semaphore full) {
        this.buffer = buffer;
        this.mutex = mutex;
        this.empty = empty;
        this.full = full;
        this.data = 1;
    }
```

```
    @Override
    public void run() {
        try {
            while (true) {
                Thread.sleep((long) (Math.random() * 5000)); // Simulate producing time
                empty.acquire(); // Wait for an empty slot in the buffer
                mutex.acquire(); // Obtain exclusive access to the buffer
                buffer.add(data); // Add item to the buffer
                System.out.println("Producer produced: " + data);
                data++;
                mutex.release(); // Release exclusive access to the buffer
                full.release(); // Signal that a new item is available in the buffer
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

class Consumer implements Runnable {
    private Buffer buffer;
    private Semaphore mutex;
    private Semaphore empty;
    private Semaphore full;

    public Consumer(Buffer buffer, Semaphore mutex, Semaphore empty, Semaphore full) {
        this.buffer = buffer;
        this.mutex = mutex;
        this.empty = empty;
        this.full = full;
    }

    @Override
    public void run() {
        try {
            while (true) {
                Thread.sleep((long) (Math.random() * 5000)); // Simulate consuming time
                full.acquire(); // Wait for a filled slot in the buffer
                mutex.acquire(); // Obtain exclusive access to the buffer
                int data = buffer.remove(); // Remove item from the buffer
                System.out.println("Consumer consumed: " + data);
                mutex.release(); // Release exclusive access to the buffer
                empty.release(); // Signal that an empty slot is available in the buffer
            }
```

```
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

class Buffer {
    private int[] buffer;
    private int size;
    private int in;
    private int out;

    public Buffer(int size) {
        this.size = size;
        this.buffer = new int[size];
        this.in = 0;
        this.out = 0;
    }

    public void add(int data) {
        buffer[in] = data;
        in = (in + 1) % size;
    }

    public int remove() {
        int data = buffer[out];
        out = (out + 1) % size;
        return data;
    }
}

public class ProducerConsumerSemaphore {
    public static void main(String[] args) {
        int bufferSize = 5;
        Buffer buffer = new Buffer(bufferSize);
        Semaphore mutex = new Semaphore(1); // Mutex for buffer access
        Semaphore empty = new Semaphore(bufferSize); // Empty slots in buffer
        Semaphore full = new Semaphore(0); // Filled slots in buffer

        // Create producer and consumer threads
        Thread producerThread = new Thread(new Producer(buffer, mutex, empty, full));
        Thread consumerThread = new Thread(new Consumer(buffer, mutex, empty, full));

        // Start the threads
```

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

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

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

class Reader implements Runnable {
    private Semaphore mutex;
    private Semaphore wrt;
    private int readerId;

    public Reader(Semaphore mutex, Semaphore wrt, int readerId) {
        this.mutex = mutex;
        this.wrt = wrt;
        this.readerId = readerId;
    }

    @Override
    public void run() {
        try {
            while (true) {
                Thread.sleep((long) (Math.random() * 5000)); // Simulate reading time
                mutex.acquire(); // Acquire mutex to ensure mutual exclusion between readers
                System.out.println("Reader " + readerId + " is reading");
                mutex.release(); // Release mutex

                // Reading is happening concurrently, so multiple readers can read at the same time

                Thread.sleep((long) (Math.random() * 5000)); // Simulate processing time
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

class Writer implements Runnable {
    private Semaphore mutex;
    private Semaphore wrt;
    private int writerId;
```

```
    public Writer(Semaphore mutex, Semaphore wrt, int writerId) {
        this.mutex = mutex;
        this.wrt = wrt;
        this.writerId = writerId;
    }

    @Override
    public void run() {
        try {
            while (true) {
                Thread.sleep((long) (Math.random() * 5000)); // Simulate writing time
                wrt.acquire(); // Acquire write lock
                System.out.println("Writer " + writerId + " is writing");

                Thread.sleep((long) (Math.random() * 5000)); // Simulate processing time

                wrt.release(); // Release write lock
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class ReaderWriterSemaphore {
    public static void main(String[] args) {
        int numReaders = 3;
        int numWriters = 2;
        Semaphore mutex = new Semaphore(1); // Mutex for reader access
        Semaphore wrt = new Semaphore(1); // Semaphore for write lock

        // Create reader threads
        for (int i = 1; i <= numReaders; i++) {
            Thread readerThread = new Thread(new Reader(mutex, wrt, i));
            readerThread.start();
        }

        // Create writer threads
        for (int i = 1; i <= numWriters; i++) {
            Thread writerThread = new Thread(new Writer(mutex, wrt, i));
            writerThread.start();
        }
    }
}
```

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

36. Write a program to demonstrate IPC using shared memory (shmget, shmat, shmdt). In this, one process will take numbers as input from user and another process will sort the numbers.

37. Write a program in which different processes will perform different operation on shared memory. (using shmget, shmat, shmdt).

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

#define SHARED_MEMORY_KEY 1234
#define MAX_SIZE 100

typedef struct {
    int data[MAX_SIZE];
    int count;
} SharedData;

void writeData(SharedData* sharedData) {
    printf("Writing data to shared memory...\n");

    // Write data to the shared memory
    for (int i = 0; i < sharedData->count; i++) {
        sharedData->data[i] = i + 1;
    }

    printf("Data written to shared memory successfully.\n");
}

void readData(SharedData* sharedData) {
    printf("Reading data from shared memory...\n");

    // Read and display the data from the shared memory
    for (int i = 0; i < sharedData->count; i++) {
        printf("%d ", sharedData->data[i]);
    }
```

```
    printf("\n");
}

void modifyData(SharedData* sharedData) {
    printf("Modifying data in shared memory...\n");

    // Modify the data in the shared memory
    for (int i = 0; i < sharedData->count; i++) {
        sharedData->data[i] += 10;
    }

    printf("Data modified successfully.\n");
}

int main() {
    int shmid;
    SharedData* sharedData;

    // Create the shared memory segment
    shmid = shmget(SHARED_MEMORY_KEY, sizeof(SharedData), IPC_CREAT | 0666);

    if (shmid == -1) {
        perror("shmget");
        exit(1);
    }

    // Attach the shared memory segment to the process's address space
    sharedData = (SharedData*)shmat(shmid, NULL, 0);

    if (sharedData == (SharedData*)-1) {
        perror("shmat");
        exit(1);
    }

    // Create child processes
    pid_t childPid1 = fork();

    if (childPid1 == -1) {
        perror("fork");
        exit(1);
    }

    if (childPid1 == 0) {
```

```
    // Child process 1 (write data to shared memory)
    writeData(sharedData);

    // Detach the shared memory segment
    shmdt(sharedData);
    exit(0);
}

pid_t childPid2 = fork();

if (childPid2 == -1) {
    perror("fork");
    exit(1);
}

if (childPid2 == 0) {
    // Child process 2 (read data from shared memory)
    sleep(1); // Wait for the write operation to complete

    readData(sharedData);

    // Detach the shared memory segment
    shmdt(sharedData);
    exit(0);
}

pid_t childPid3 = fork();

if (childPid3 == -1) {
    perror("fork");
    exit(1);
}

if (childPid3 == 0) {
    // Child process 3 (modify data in shared memory)
    sleep(2); // Wait for the read operation to complete

    modifyData(sharedData);

    // Detach the shared memory segment
    shmdt(sharedData);
    exit(0);
}
```

```
    // Wait for all child processes to complete
    wait(NULL);
    wait(NULL);
    wait(NULL);

    // Detach and remove the shared memory segment
    shmdt(sharedData);
    shmctl(shmid, IPC_RMID, NULL);

    return 0;
}
```

38. Write programs to simulate linux commands cat, ls, cp, mv, head etc.

39. 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>

// Semaphore for function f1
sem_t semaphore_f1;

void *f1(void *arg) {
    // Code for function f1
    char username[100];
    printf("Enter username: ");
    scanf("%s", username);
    // Perform necessary operations with the username
    printf("Function f1 executed successfully.\n");
    // Release the semaphore to allow function f2 to execute
    sem_post(&semaphore_f1);
    pthread_exit(NULL);
}

void *f2(void *arg) {
    // Wait for the semaphore to be released by function f1
    sem_wait(&semaphore_f1);
    // Code for function f2
    char password[100];
```

```
    printf("Enter password: ");
    scanf("%s", password);
    // Perform necessary operations with the password
    printf("Function f2 executed successfully.\n");
    pthread_exit(NULL);
}

int main() {
    pthread_t thread1, thread2;

    // Initialize the semaphore
    sem_init(&semaphore_f1, 0, 0);

    // Create and start the threads
    pthread_create(&thread1, NULL, f1, NULL);
    pthread_create(&thread2, NULL, f2, NULL);

    // Wait for the threads to complete
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    // Destroy the semaphore
    sem_destroy(&semaphore_f1);

    return 0;
}
```

40. 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 number) {
    if (number <= 1) {
        return 0;
    }

    for (int i = 2; i * i <= number; i++) {
        if (number % i == 0) {
            return 0;
        }
    }
```

```
    return 1;
}

int main() {
    int start = 1;
    int end = 100;

    #pragma omp parallel for
    for (int i = start; i <= end; i++) {
        if (is_prime(i)) {
            printf("%d is prime.\n", i);
        }
    }

    return 0;
}

// gcc -fopenmp -o code.c
```

41. 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 threadCount = 0;

    #pragma omp parallel
    {
        #pragma omp master
        {
            threadCount = omp_get_num_threads();
            printf("Total threads: %d\n", threadCount);
        }

        int threadNum = omp_get_thread_num();
        printf("Thread %d\n", threadNum);
    }

    return 0;
}
```

42. 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;

   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);

   printf("Hello, world! From process %d of %d\n", rank, size);

   MPI_Finalize();

   return 0;
}
```

43. Write 2 programs that will both send 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

Sender.c

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

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

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

  // Generate a unique key
```

```
  key = ftok("message_queue", 65);

  // Create a message queue
  msgid = msgget(key, 0666 | IPC_CREAT);

  // Send the message
  msg.mtype = 1;
  sprintf(msg.mtext, "Are you hearing me?");
  msgsnd(msgid, &msg, sizeof(msg), 0);

  // Wait for the reply
  msgrcv(msgid, &msg, sizeof(msg), 2, 0);
  printf("Process 2: %s\n", msg.mtext);

  // Send the response
  msg.mtype = 1;
  sprintf(msg.mtext, "I can hear you too");
  msgsnd(msgid, &msg, sizeof(msg), 0);

  // Remove the message queue
  msgctl(msgid, IPC_RMID, NULL);

  return 0;
}
```

Receiver.c

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

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

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

  // Generate a unique key
```

```
  key = ftok("message_queue", 65);

  // Access the message queue
  msgid = msgget(key, 0666 | IPC_CREAT);

  // Receive the message
  msgrcv(msgid, &msg, sizeof(msg), 1, 0);
  printf("Process 1: %s\n", msg.mtext);

  // Send the reply
  msg.mtype = 2;
  sprintf(msg.mtext, "Loud and Clear");
  msgsnd(msgid, &msg, sizeof(msg), 0);

  // Receive the response
  msgrcv(msgid, &msg, sizeof(msg), 1, 0);
  printf("Process 1: %s\n", msg.mtext);

  return 0;
}
```

44. Write a program for TCP to demonstrate the socket system calls

TCP_Server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[BUFFER_SIZE] = {0};
    char* message = "Hello from server";

    // Create a socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
```

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

    // Attach socket to the port
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt))) {
        perror("setsockopt failed");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    if (bind(server_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

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

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

    // Read data from the client
    valread = read(new_socket, buffer, BUFFER_SIZE);
    printf("Client: %s\n", buffer);

    // Send a response to the client
    send(new_socket, message, strlen(message), 0);
    printf("Server: %s\n", message);

    return 0;
}
```

TCP_Client.c

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

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char* message = "Hello from client";
    char buffer[BUFFER_SIZE] = {0};

    // Create a socket
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket failed");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IP address from string to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        perror("inet_pton failed");
        return -1;
    }

    // Connect to the server
    if (connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0) {
        perror("connect failed");
        return -1;
    }

    // Send a message to the server
    send(sock, message, strlen(message), 0);
    printf("Client: %s\n", message);

    // Read data from the server
    valread = read(sock, buffer, BUFFER_SIZE);
    printf("Server: %s\n", buffer);
```

```
    return 0;
}
```

45. Write a program for UDP to demonstrate the socket system calls
46. Implement echo server using TCP in iterative/concurrent logic.
47. Implement echo server using UDP in iterative/concurrent logic.
48. Write a program using PIPE, to Send data from parent to child over a pipe. (unnamed pipe)

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

#define BUFFER_SIZE 1024

int main() {
  int pipefd[2];
  pid_t pid;
  char buffer[BUFFER_SIZE];
  const char *message = "Hello from parent";

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

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

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

  if (pid > 0) {
    // Parent process

    // Close the read end of the pipe
    close(pipefd[0]);

    // Write data to the pipe
    write(pipefd[1], message, strlen(message) + 1);
```

```
    printf("Parent: Sent message to child\n");

    // Close the write end of the pipe
    close(pipefd[1]);

    // Wait for the child process to exit
    wait(NULL);
  } else {
    // Child process

    // Close the write end of the pipe
    close(pipefd[1]);

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

    // Close the read end of the pipe
    close(pipefd[0]);

    exit(EXIT_SUCCESS);
  }

  return 0;
}
```

49. 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>
#include <string.h>

#define BUFFER_SIZE 1024

int main() {
    int fd;
    pid_t pid;
    char buffer[BUFFER_SIZE];
    const char* fifoPath = "/tmp/myfifo";
    const char* message = "Hello from parent";
```

```
// Create the named pipe (FIFO)
if (mkfifo(fifoPath, 0666) == -1) {
    perror("mkfifo failed");
    exit(EXIT_FAILURE);
}

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

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

if (pid > 0) {
    // Parent process

    // Open the named pipe for writing
    fd = open(fifoPath, O_WRONLY);
    if (fd == -1) {
        perror("open failed");
        exit(EXIT_FAILURE);
    }

    // Write data to the named pipe
    write(fd, message, strlen(message) + 1);
    printf("Parent: Sent message to child\n");

    // Close the named pipe
    close(fd);

    // Wait for the child process to exit
    wait(NULL);
} else {
    // Child process

    // Open the named pipe for reading
    fd = open(fifoPath, O_RDONLY);
    if (fd == -1) {
        perror("open failed");
        exit(EXIT_FAILURE);
    }
```

```
        // Read data from the named pipe
        read(fd, buffer, BUFFER_SIZE);
        printf("Child: Received message from parent: %s\n", buffer);

        // Close the named pipe
        close(fd);

        exit(EXIT_SUCCESS);
    }

    // Remove the named pipe (FIFO)
    unlink(fifoPath);

    return 0;
}
```

50. 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>

int main() {
  int fd[2];
  pid_t pid;
  char file_path[] = "/home/pandoras/ramu/output.txt";

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

  // Fork the process
  pid = fork();

  if (pid > 0) {
    // Parent process
    close(fd[0]); // Close the read end of the pipe
    FILE *file = fopen(file_path, "rb");

    if (file == NULL) {
      perror("fopen");
      exit(EXIT_FAILURE);
```

```
    }

    // Send the file size to the child process
    fseek(file, 0, SEEK_END);
    long file_size = ftell(file);
    fseek(file, 0, SEEK_SET);
    write(fd[1], &file_size, sizeof(long));

    // Send the file contents to the child process
    char buffer[1024];
    size_t bytesRead;

    while ((bytesRead = fread(buffer, 1, sizeof(buffer), file)) > 0) {
      write(fd[1], buffer, bytesRead);
    }

    close(fd[1]); // Close the write end of the pipe

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

    printf("File sent successfully!\n");
    fclose(file);
  } else if (pid == 0) {
    // Child process
    close(fd[1]); // Close the write end of the pipe

    // Receive the file size from the parent process
    long file_size;
    read(fd[0], &file_size, sizeof(long));

    // Open a new file to save the received data
    char received_file_path[] = "received_file.txt";
    FILE *received_file = fopen(received_file_path, "wb");

    if (received_file == NULL) {
      perror("fopen");
      exit(EXIT_FAILURE);
    }

    // Receive the file contents from the parent process
    char buffer[1024];
    size_t totalReceived = 0;
    ssize_t bytesRead;
```

```
    while (totalReceived < file_size) {
      bytesRead = read(fd[0], buffer, sizeof(buffer));
      fwrite(buffer, 1, bytesRead, received_file);
      totalReceived += bytesRead;
    }

    close(fd[0]); // Close the read end of the pipe

    printf("File received and saved successfully!\n");
    fclose(received_file);
  } else {
    perror("fork");
    exit(EXIT_FAILURE);
  }

  return 0;
}
```

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

52. 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 <ctype.h>
#include <sys/wait.h>
#include <string.h>

int main() {
    int pipefd[2];
    pid_t pid;

    if (pipe(pipefd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }
    pid = fork();
    if (pid == -1) {
```

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

    if (pid == 0) { // child process (lowercase filter)
        close(pipefd[1]); // close write end of pipe
        char c;
        while (read(pipefd[0], &c, sizeof(c)) > 0) { // read data from read end of pipe
            c = tolower(c); // convert character to lowercase
            write(STDOUT_FILENO, &c, sizeof(c)); // write data to stdout
        }
        close(pipefd[0]); // close read end of pipe
        exit(EXIT_SUCCESS);
    } else { // parent process
        close(pipefd[0]); // close read end of pipe
        char *data = "ThIs Is A StRiNg To Be ConVertEd.";
        write(pipefd[1], data, strlen(data) + 1); // write data to write end of pipe
        close(pipefd[1]); // close write end of pipe
        wait(NULL); // wait for child process to finish
        exit(EXIT_SUCCESS);
    }
    return 0;
}
```

53. Write a program to illustrate the semaphore concept. Use fork so that 2 process running simultaneously and communicate via semaphore.

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

int main() {
    int sem_id;
    key_t key;
    pid_t pid;

    // Generate a key for the semaphore
    if ((key = ftok(".", 'S')) == -1) {
        perror("ftok");
        exit(EXIT_FAILURE);
    }
```

```
// Create a semaphore
if ((sem_id = semget(key, 1, IPC_CREAT | 0666)) == -1) {
    perror("semget");
    exit(EXIT_FAILURE);
}

// Fork the process
pid = fork();

if (pid == -1) {
    perror("fork");
    exit(EXIT_FAILURE);
} else if (pid == 0) {
    // Child process

    printf("Child process is waiting...\n");

    // Wait for the semaphore to be available
    semop(sem_id, NULL, 0);

    printf("Child process has acquired the semaphore.\n");
    printf("Child process is releasing the semaphore.\n");

    // Release the semaphore
    semop(sem_id, NULL, 1);
} else {
    // Parent process

    printf("Parent process is waiting...\n");

    // Wait for the semaphore to be available
    semop(sem_id, NULL, 0);

    printf("Parent process has acquired the semaphore.\n");
    printf("Parent process is releasing the semaphore.\n");

    // Release the semaphore
    semop(sem_id, NULL, 1);
}

// Remove the semaphore
if (semctl(sem_id, 0, IPC_RMID) == -1) {
    perror("semctl");
```

```
        exit(EXIT_FAILURE);
    }

    return 0;
}
```

54. 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.

```
#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 key for the semaphore
    if ((key = ftok(".", 'S')) == -1) {
        perror("ftok");
        exit(EXIT_FAILURE);
    }

    // Create a semaphore
    if ((sem_id = semget(key, 1, IPC_CREAT | IPC_EXCL | 0666)) == -1) {
        perror("semget");
        exit(EXIT_FAILURE);
    }

    printf("Semaphore created with ID: %d\n", sem_id);

    return 0;
}
```

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

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

```
void perform_P_operation(int sem_id) {
   struct sembuf sem_op;
   sem_op.sem_num = 0;
   sem_op.sem_op = -1;
   sem_op.sem_flg = 0;
   semop(sem_id, &sem_op, 1);
}

int main() {
   key_t key;
   int sem_id;

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

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

   printf("Performing P operation on Semaphore\n");

   // Perform P operation
   perform_P_operation(sem_id);

   printf("P operation completed\n");

   return 0;
}
```

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

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

void perform_V_operation(int sem_id) {
```

```
    struct sembuf sem_op;
    sem_op.sem_num = 0;
    sem_op.sem_op = 1;
    sem_op.sem_flg = 0;
    semop(sem_id, &sem_op, 1);
}

int main() {
    key_t key;
    int sem_id;

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

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

    printf("Performing V operation on Semaphore\n");

    // Perform V operation
    perform_V_operation(sem_id);

    printf("V operation completed\n");

    return 0;
}
```

A.J.Umbarkar