OS & NETWORK LAB ASSIGNMENT

Department of
Computer Science & Engineering
NIT JAMSHEDPUR

Name: Suryakant

Reg no. : 2021PGCACA063

Course : MCA(5^{TH} Sem)

FACULTY-IN-CHARGE: Dr. A. K. Mishra

Assignment-04

Q. 1 Write a shell script to input a filename via command line arguments and print the type of file (such as c prog, c++, text file, pdf file, doc file etc.)

Command –

Open ▾ Q1.sh ~/Documents/assign04

```
#!/bin/bash

if [ $# -eq 0 ]; then
    echo "Usage: $0 <filename>"
    exit 1
fi

filename="$1"
extension="${filename##*.}"

case "$extension" in
   "c" | "h" | "c++" |  "hpp")
      echo "C/C++ Source code"
      ;;
    "txt")
      echo "Text file"
      ;;
     "pdf")
      echo "PDF file"
      ;;
    *)

     echo "Unknown file type"
     ;;
 esac
```

Output:

```
surya@ubuntu:~/Documents/assign04$ touch Q1.sh
surya@ubuntu:~/Documents/assign04$ chmod +x Q1.sh
```

```
surya@ubuntu:~/Documents/assign04$ ./Q1.sh help.txt
Text file
surya@ubuntu:~/Documents/assign04$ ./Q1.sh helper.pdf
PDF file
surya@ubuntu:~/Documents/assign04$ ./Q1.sh palindrome.cpp
Unknown file type
surya@ubuntu:~/Documents/assign04$ ./Q1.sh palindrome.c
C/C++ Source code
surya@ubuntu:~/Documents/assign04$
```

Q2 . Write a shell script to input three numbers via command line arguments and print the largest among them

Command –

Open ~/Desktop/assign_04 Q2.sh

```
#!/bin/bash

#verifying the number of argument passed
if [ "$#" -ne 3 ]; then
   echo "Usage: $0 <number1> <number2> <number3>"
   exit 1
fi

#Assign to variables
n1=$1
n2=$2
n3=$3

#check which number is largest
if [ "$n1" -ge "$n2" ] && ["$1" -ge "$n3"]; then
   echo "The largest number is : $n1"
elif [ "$n2" -ge "$n1" ] && ["$2" -ge "$n1"]; then
   echo "The largest number is : $n2"
else
   echo "The largest number is : $n3"
fi
```

Output:

```
surya@ubuntu:~/Desktop/assign_04$ touch Q2.sh
surya@ubuntu:~/Desktop/assign_04$ chmod +x Q2.sh
```

```
surya@ubuntu:~/Desktop/assign_04$ ./Q2.sh 12 2 5
The largest number is :12
surya@ubuntu:~/Desktop/assign_04$
```

Q3. Write a shell script to input two filenames in the current directory as arguments and print whether both files has equal number of lines.

Command –

Open ▾ Q3.sh ~/Desktop/assign_04

```
#!/bin/bash

#verifying the number of argument passed
if [ "$#" -ne 2 ]; then
   echo "Usage: $0 <file1> <file2>"
   exit 1
fi

#Assign to variables
file1=$1
file2=$2

#check if both file exist
if [ ! -e "$file1" ] || [ ! -e "$file2" ]; then
   echo "Error : One or both of the files do not exist."
   exit 1
fi
#Get the line count for each line
lines_file1=$(wc -l < "$file1")
lines_file2=$(wc -l < "$file2")
#compare the line counts and prints the result
if [ "$lines_file1" -eq "$lines_file2" ]; then
   echo "Both the files have an equal number of lines : $lines_file1"
else
   echo "Both the files have a different number of lines."
   echo "$file1 : $lines_file1 lines"
   echo "$file2 : $lines_file2 lines"
fi
```

Output:

```
surya@ubuntu:~/Desktop/assign_04$ touch Q3.sh
surya@ubuntu:~/Desktop/assign_04$ chmod +x Q3.sh
```

```
surya@ubuntu:~/Desktop/assign_04$ touch file1.txt
surya@ubuntu:~/Desktop/assign_04$ touch file2.txt
surya@ubuntu:~/Desktop/assign_04$ ./Q3.sh file1.txt file2.txt
Both the files have a different number of lines.
file1.txt : 11 lines
file2.txt : 8 lines
surya@ubuntu:~/Desktop/assign_04$
```

Q 4: Write a shell script to input the registration number (2021UGA011) of a student print its branch.

Command -

```
#!/bin/bash

#input the registration number
read -p "Enter the registration numebr(e.g. 2021PGCACA063): " reg_number
#check if the registration number has the correct format
if [[ $reg_number =~ ^[0-9]{4}[A-Z]{6}[0-9]{3}$ ]]; then
  #Extract the branch
  branch=${reg_number:4:6}

  #print the branch
  echo "Branch : $branch"
else
  echo "Invalid registration number format , please use the format 'YYYYBBBBBBNNN'"
fi
```

Output:

```
surya@ubuntu:~/Desktop/assign_04$ touch Q4.sh
surya@ubuntu:~/Desktop/assign_04$ chmod +x Q4.sh
```

```
surya@ubuntu:~/Desktop/assign_04$ ./Q4.sh
Enter the registration numebr(e.g. 2021PGCACA063): 2021PGCACA063
Branch : PGCACA
surya@ubuntu:~/Desktop/assign_04$
```

Assignment-05

a) Write a shell script that receives any number of file names as arguments checks if every argument supplied is a file or a directory and reports accordingly.whenever the argument is a file or directory.

Command -

```
#!/bin/bash

for arg in "$@"; do
    if [ -f "$arg" ]; then
        echo "$arg is a file."
    elif [ -d "$arg" ]; then
        echo "$arg is a directory."
    else
        echo "$arg is neither a file nor a
directory."
    fi
done
```

```
Surya : :~$  sh as51.sh output1.txt margesort.c
output1.txt is a file.
margesort.c is a file.
```

b) Write a shell script to input a number of random integers as arguments and print only the prime numbers.

Command –

```
#!/bin/bash

is_prime() {
    num=$1
    if [ "$num" -le 1 ]; then
        return 1
    fi
    for ((i = 2; i * i <= num; i++)); do


        if [ "$((num % i))" -eq 0 ]; then
            return 1
        fi
    done
    return 0
}

for num in "$@"; do
    if is_prime "$num"; then
        echo "$num is a prime number."
    fi
done
```

```
Surya : :~$  sh as52.sh 67 13
Prime numbers in the input are:
67
13
```

c) Write a shell script to input a file name and print the lines that contains numeric values in it.

Command -

```
#!/bin/bash

if [ "$#" -ne 1 ]; then
    echo "Usage: $0 <file_name>"
    exit 1
fi

file="$1"

if [ ! -f "$file" ]; then
    echo "File $file does not exist."
    exit 1
fi

grep -E '[0-9]+' "$file"
```

```
Surya : :~$  sh as53.sh clerk.txt
7369 SMITH CLERK 7902 17-DEC-1980 800 20
```

Assignment-06

a) Demonstrate the insertion, delection, and replacement of string using sed Editor.

Command –

```
Surya : :~$  cat input.txt
hello
Surya : :~$ sed -i '1ihello world' input.txt
Surya : :~$      cat input.txt
hello world
hello
Surya : :~$ sed '2d' input.txt
hello world
Surya : :~$      cat input.txt
hello world
hello
Surya : :~$ Sed 's/hello/hello surya/g' input.txt
hello surya world
hello surya
```

b) Write a awk script to print the details of all employees in employee.txt whose salary is greater than $2000.

Command –

```
Surya : :~$ awk '{if($3 > 2000) print $1, $2, $3}' employee.txt
7499 ALLEN SALESMAN
7521 WARD SALESMAN
7566 JONES MANAGER
7654 MARTIN SALESMAN
7698 BLAKE MANAGER
7782 CLARK MANAGER
7788 SCOTT ANALYST
7839 KING PRESIDENT
7844 TURNER SALESMAN
7902 FORD ANALYST
```

```
+ 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610
- 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610
r 1.84471 2 1 cbr 210 ------- 1 3.0 1.0 195 600
r 1.84566 2 0 ack 40 ------- 2 3.2 0.1 82 602
+ 1.84566 0 2 tcp 1000 ------- 2 0.1 3.2 102 611
- 1.84566 0 2 tcp 1000 ------- 2 0.1 3.2 102 611
r 1.84609 0 2 cbr 210 ------- 0 0.0 3.1 225 610
+ 1.84609 2 3 cbr 210 ------- 0 0.0 3.1 225 610
d 1.84609 2 3 cbr 210 ------- 0 0.0 3.1 225 610
- 1.8461 2 3 cbr 210 ------- 0 0.0 3.1 192 511
r 1.84612 3 2 cbr 210 ------- 1 3.0 1.0 196 603
+ 1.84612 2 1 cbr 210 ------- 1 3.0 1.0 196 603
- 1.84612 2 1 cbr 210 ------- 1 3.0 1.0 196 603
+ 1.84625 3 2 cbr 210 ------- 1 3.0 1.0 199 612
```

c) write an awk script to read the above trace file and print the number of messages sent by each node. (here column 3 an 4 indicates the source and destination node id of the packet)

Command –

```
Surya : :~$ cat employee.txt
```

```
7499 ALLEN SALESMAN 7698 20-FEB-1981 1600 30
7521 WARD SALESMAN 7698 22-FEB-1981 1250 30
7566 JONES MANAGER 7839 2-APR-1981 2975 20
7654 MARTIN SALESMAN 7698 28-SEP-1981 1250 30
7698 BLAKE MANAGER 7839 1-MAY-1981 2850 30
7782 CLARK MANAGER 7839 9-JUN-1981 2450 10
7788 SCOTT ANALYST 7566 09-DEC-1982 3000 20
7839 KING PRESIDENT 1 17-NOV-1981 5000 10
7844 TURNER SALESMAN 7698 8-SEP-1981 1500 30
7902 FORD ANALYST 7566 3-DEC-1981 3000 20
```

```
Surya : :~$ awk '{ source_node = $3 ; dest_node = $4;
sent_msg[source_node]++;}
END {
for (node in sent_msg){
printf("Node %d sent %d message\n", node, sent_msg[node]);
}
}' employee.txt
Node 0 sent 1 message
Node 0 sent 4 message
Node 0 sent 2 message
Node 0 sent 1 message
Node 0 sent 3 message
```

d) Write an awk script to read the above trace file and print the total amount of bytes exchanged during network communications. (the column 6 represents the size of a message).

Command –

```
Surya : :~$ awk '{total_bytes += $6} END {print "Total bytes exchanged:
total_bytes}' employee.txt
Total bytes exchanged: 24875
```

Assignment-07

a) Write a C Program that makes a copy of a file using standard I/O and system calls.

Command –

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

int main() {
    FILE *source, *destination;
    char sourceFileName[] = "output1.txt";
    char destinationFileName[] = "output2.txt";

    source = fopen(sourceFileName, "rb");
    if (source == NULL) {
        perror("Error opening source file");
        return 1;
    }

    destination = fopen(destinationFileName, "wb");
    if (destination == NULL) {
        perror("Error creating destination file");
        fclose(source);
        return 1;
    }

    char ch;
    while ((ch = fgetc(source)) != EOF) {
        fputc(ch, destination);
    }
```

```
    fclose(source);
    fclose(destination);
    printf("File copied successfully.\n");

    return 0;
}
```

```
Surya : :~$ gcc as71.c -o as71

Surya : :~$ ./as71 output1.txt output2.txt

File copied successfully.

Surya : :~$ cat output2.txt

hello world I am surya
```

b) Write in C the following Unix commands using system calls
i) cat ii) cp

Command –

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

int main() {
    FILE *source, *destination;
    char sourceFileName[] = "output1.txt";
    char destinationFileName[] = "output2.txt";

    source = fopen(sourceFileName, "rb");
    if (source == NULL) {
        perror("Error opening source file");
        return 1;
    }

    destination = fopen(destinationFileName, "wb");
    if (destination == NULL) {
        perror("Error creating destination file");
        fclose(source);
        return 1;
    }

    char ch;
    while ((ch = fgetc(source)) != EOF) {
        fputc(ch, destination);
    }

    fclose(source);
    fclose(destination);
    printf("File copied successfully.\n");

    return 0;
}
```

c) Write a C program to list files in a directory

Command –

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

int main() {
    // Emulate "cat" command
    int file_descriptor = open("output1.txt", O_RDONLY);
    if (file_descriptor < 0) {
        perror("Error opening source file");
        return 1;
    }

    char buffer[1024];
    ssize_t bytes_read;

    while ((bytes_read = read(file_descriptor, buffer, sizeof(buffer))) > 0)
        write(STDOUT_FILENO, buffer, bytes_read);
    }

    close(file_descriptor);

    // Emulate "cp" command
    int source_fd = open("output1.txt", O_RDONLY);
    int dest_fd = open("output2.txt", O_WRONLY | O_CREAT, 0666);
    if (source_fd < 0 || dest_fd < 0) {
        perror("Error opening source or destination file");
        return 1;
    }

    while ((bytes_read = read(source_fd, buffer, sizeof(buffer))) > 0) {
        write(dest_fd, buffer, bytes_read);
    }
```

```
    close(file_descriptor);

    // Emulate "cp" command
    int source_fd = open("output1.txt", O_RDONLY);
    int dest_fd = open("output2.txt", O_WRONLY | O_CREAT, 0666);
    if (source_fd < 0 || dest_fd < 0) {
        perror("Error opening source or destination file");
        return 1;
    }

    while ((bytes_read = read(source_fd, buffer, sizeof(buffer))) > 0) {
        write(dest_fd, buffer, bytes_read);
    }

    close(source_fd);
    close(dest_fd);

    return 0;
}
```

```
Surya : :~$  gcc as73.c -o as73

Surya : :~$  ./as73  /home/ Surya/
```

Files in directory '/home/surya/'

```
newemp.txt
margesort.c
.bash_logout
task.bat
output_file.txt
a.out
employee2nd.txt
as73
a3q3.sh
output2.txt
input.txt
as71.c
as72
```

(d) Write a C program to list for every file in a directory, its inode number and file name

Command –

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

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

    int fd = open(argv[1], O_RDONLY);

    if (fd < 0) {
        perror("Error opening the file");
        exit(1);
    }

    char buffer[4096];
    ssize_t n;

    while ((n = read(fd, buffer, sizeof(buffer))) > 0) {
        if (write(STDOUT_FILENO, buffer, n) < 0) {
            perror("Error writing to stdout");
            exit(1);
        }
    }

    close(fd);

    return 0;
}
```

```
Surya : :~$ gcc as74.c -o as74

Surya : :~$ ./as74 /home/ Surya/

Surya : :~$
```

Assignment-08

(a) Write a C program to create a child process and allow the parent to display ‘parent‘ and the child to display ‚child‘ on the screen.

C Program –

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

int main() {

        pid_t pid;
                // Create a child process
        pid = fork();
        if (pid < 0) {
                // Fork failed
        fprintf(stderr, "Fork failed\n");
        exit(1);
        } else if (pid == 0) {
                // Child process
        printf("child\n");
        } else {
                // Parent process
        printf("parent\n");
        }
return 0;
}
```

Command :

```
Surya : :~$ gcc a8q1.c -o a
Surya : :~$ ./a
parent
child
```

(b) Write a C program to create a Zombie and orphan process.

C Program –

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

int main() {

pid_t child_pid;
                                                    // Create a child process
child_pid = fork();
if (child_pid < 0) {
                                                    // Fork failed
fprintf(stderr, "Fork failed\n");
exit(1);
}
else if (child_pid == 0) {
                                                    // Child process
                                                    // Orphan process
Printf("Child (PID=%d) is now an orphan. Parent PID=%d\n", getpid(), getppid());
// Sleep to become a zombie
sleep(2);
                                                    // The child process is now a zombie
printf("Child process (PID=%d) is now a zombie.\n", getpid());
}
else {
                                                    // Parent process
                                                    // Sleep to let child become an orphan
sleep(1);
                                                    // Display the status of the child process
printf("Parent: Child (PID=%d) is created.\n", child_pid);
                                                        // Wait for the child to exit
wait(NULL);
                                                    // Parent process is done
printf("Parent process is done.\n");
}
return 0;
}
```

Command :

```
Surya : :~$ gcc a8q2.c -o a

Surya : :~$ ./a

Child (PID=3627) is now an orphan. Parent PID=3626
Parent: Child (PID=3627) is created.
Child process (PID=3627) is now a zombie.
Parent process is done.
```

(c) Write a program that illustrates how to execute two commands concurrently

C Program –

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

int main() {

pid_t pid1 = fork();

if (pid1 == -1) {

perror("Fork failed");
exit(EXIT_FAILURE);
}

if (pid1 == 0) {
                                                    // Child process for the first command
execlp("echo", "echo", "Command 1 executed", NULL);
perror("execlp");
exit(EXIT_FAILURE);
}
pid_t pid2 = fork();

if (pid2 == -1) {
perror("Fork failed");
exit(EXIT_FAILURE);
}

if (pid2 == 0) {
```

```
        // Child process for the second command
execlp("echo", "echo", "Command 2 executed", NULL);
perror("execlp");
exit(EXIT_FAILURE);
}
        // Wait for both child processes to finish
waitpid(pid1, NULL, 0);
waitpid(pid2, NULL, 0);
printf("Both commands executed concurrently.\n");
return 0;
}
```

Command :

```
Surya : :~$ gcc a8q3.c -o a
Surya : :~$ ./a
Command 2 executed
Command 1 executed
Both commands executed concurrently.
```

(d) Write a C program that illustrates suspending and resuming processes using signals

C Program –

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h> // Include the necessary header

void signal_handler(int signo) {
        // Signal handler function
printf("Received signal %d\n", signo);
}
int main() {

pid_t child_pid;
                // Register signal_handler for SIGSTOP and SIGCONT
```

```
signal(SIGSTOP, signal_handler);
signal(SIGCONT, signal_handler);
                        // Create a child process
child_pid = fork();

if (child_pid < 0) {
fprintf(stderr, "Fork failed\n");
exit(1);
}
 else if (child_pid == 0) {
                // Child process
while (1) {
printf("Child process running...\n");
sleep(1);
}
}
else {
                // Parent process
                // Let the child run for a while
sleep(3);
                // Suspend the child process
printf("Parent: Suspending child process...\n");
kill(child_pid, SIGSTOP);
                // Let the child be suspended for a while
sleep(3);
                // Resume the child process
printf("Parent: Resuming child process...\n");
kill(child_pid, SIGCONT);
                // Let the child run again
sleep(3);

// Terminate the child process
printf("Parent: Terminating child process...\n");
kill(child_pid, SIGTERM);

                // Wait for the child to exit
wait(NULL);
printf("Parent process is done.\n");

}
return 0;
}
```

Command :

```
Surya : :~$ gcc a8q4.c -o a

Surya : :~$ ./a

Child process running...
Child process running...
Child process running...
Parent: Suspending child process...
Parent: Resuming child process...
Received signal 18
Child process running...
Child process running...
Child process running...
Parent: Terminating child process...
Parent process is done.
```

Assignment-09

a) Write a C program that illustrates inter process communication using shared memory.

C Program –

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

#define SHM_SIZE 1024

struct shared_data {
   int value;
   char message[100];
};

int main() {
   int shmid;
   key_t key = 1234; // Unique key for shared memory segment
   struct shared_data *shared_memory;

   // Create a shared memory segment
   if ((shmid = shmget(key, sizeof(struct shared_data), IPC_CREAT | 0666)) < 0) {
      perror("shmget");
      exit(1);
   }

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

   // Initialize the shared memory data
   shared_memory->value = 0;
   strcpy(shared_memory->message, "Hello, shared memory!");

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

```
    if (pid < 0) {
        perror("fork");
        exit(1);
    } else if (pid == 0) {
        // Child process
        printf("Child process reading from shared memory:\n");
        printf("Initial value: %d\n", shared_memory->value);
        printf("Initial message: %s\n", shared_memory->message);

        // Modify the shared memory data
        shared_memory->value = 42;
        strcpy(shared_memory->message, "Modified by child process");

        printf("Child process updated shared memory:\n");
        printf("New value: %d\n", shared_memory->value);
        printf("New message: %s\n", shared_memory->message);

        // Detach the shared memory segment from the process
        shmdt(shared_memory);
    } else {
        // Parent process
        sleep(2); // Wait for the child process to finish

        printf("Parent process reading from shared memory after child process update:\n");
        printf("Updated value: %d\n", shared_memory->value);
        printf("Updated message: %s\n", shared_memory->message);

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

    return 0;
}
```

Command :

```
toki@Toki : :~/Desktop$ ./a.out
Child process reading from shared memory:
Initial value: 0
Initial message: Hello, shared memory!
Child process updated shared memory:
New value: 42
New message: Modified by child process
Parent process reading from shared memory after child process update:
Updated value: 42
Updated message: Modified by child process
toki@Toki : :~/Desktop$
```

b) Write C programs that illustrate communication between two unrelated processes using named pipe(FIFO file).

C Program –

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

#define FIFO_FILE "myfifo"

int main() {
    pid_t pid;
    int fd;
    char buf[100];

    // Create the FIFO (named pipe)
    mkfifo(FIFO_FILE, 0666);

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

    if (pid < 0) {
        perror("fork");
        exit(1);
    } else if (pid == 0) {
        // Child process (writer)
        printf("Child process (writer) opening FIFO...\n");

        // Open the FIFO for writing
        fd = open(FIFO_FILE, O_WRONLY);

        printf("Child process (writer) writing to FIFO...\n");

        // Write data to the FIFO
        write(fd, "Hello from writer process!", sizeof("Hello from writer process!"));

        printf("Child process (writer) finished writing.\n");
```

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

        exit(0);
    } else {
        // Parent process (reader)
        printf("Parent process (reader) opening FIFO...\n");

        // Open the FIFO for reading
        fd = open(FIFO_FILE, O_RDONLY);

        printf("Parent process (reader) reading from FIFO...\n");

        // Read data from the FIFO
        read(fd, buf, sizeof(buf));

        printf("Parent process (reader) received: %s\n", buf);

        // Close the FIFO
        close(fd);

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

        // Remove the FIFO
        unlink(FIFO_FILE);

        exit(0);
    }

    return 0;
}
```

Command :

```
toki@Toki : :~/Desktop$ ./a.out
Parent process (reader) opening FIFO...
Child process (writer) opening FIFO...
Parent process (reader) reading from FIFO...
Child process (writer) writing to FIFO...
Parent process (reader) received: Hello from writer process!
Child process (writer) finished writing.
toki@Toki : :~/Desktop$
```

c) Write a C program to demonstrate multithreading execution.

C Program –

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

// Function to be executed by the threads
void *threadFunction(void *arg) {
    int thread_number = *(int *)arg;

    printf("Thread %d is running!\n", thread_number);

    // Perform some work (for demonstration purposes)
    for (int i = 0; i < 3; ++i) {
        printf("Thread %d is working... (%d)\n", thread_number, i);
    }

    printf("Thread %d is done.\n", thread_number);

    // Exit the thread
    pthread_exit(NULL);
}

int main() {
    pthread_t thread1, thread2;
    int thread1_number = 1, thread2_number = 2;

    // Create two threads
    if (pthread_create(&thread1, NULL, threadFunction, (void *)&thread1_number) != 0) {
        fprintf(stderr, "Error creating thread 1.\n");
        exit(EXIT_FAILURE);
    }

    if (pthread_create(&thread2, NULL, threadFunction, (void *)&thread2_number) != 0) {
        fprintf(stderr, "Error creating thread 2.\n");
        exit(EXIT_FAILURE);
    }

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

```
    printf("Both threads have completed.\n");

    return 0;
}
```

Command :

```
toki@Toki : :~/Desktop$ gcc test.c -pthread
toki@Toki : :~/Desktop$ ./a.out
Thread 1 is running!
Thread 1 is working... (0)
Thread 1 is working... (1)
Thread 1 is working... (2)
Thread 1 is done.
Thread 2 is running!
Thread 2 is working... (0)
Thread 2 is working... (1)
Thread 2 is working... (2)
Thread 2 is done.
Both threads have completed.
toki@Toki : :~/Desktop$
```

Assignment-10

a) Write a client and server programs for interaction between server and client processes using TCP sockets. On successful connection, client sends an IPv4 address to the server. Sever identifies the network id of the IP based on its class and replies the network back to the client.

C Program –

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

#define PORT 8080

void identifyNetworkId(char *ipAddress, char *networkId) {
   // Parse the IP address to identify the class
   unsigned int ip;
   sscanf(ipAddress, "%u", &ip);

   if ((ip & 0x80000000) == 0) {
      strcpy(networkId, "Class A");
   } else if ((ip & 0xC0000000) == 0x80000000) {
      strcpy(networkId, "Class B");
   } else if ((ip & 0xE0000000) == 0xC0000000) {
      strcpy(networkId, "Class C");
   } else {
      strcpy(networkId, "Unknown Class");
   }
}

int main() {
   int serverSocket, clientSocket;
   struct sockaddr_in serverAddr, clientAddr;
   char buffer[INET_ADDRSTRLEN];

   // Create socket
   if ((serverSocket = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
      perror("Socket creation failed");
      exit(EXIT_FAILURE);
   }
```

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

    // Bind the socket to the specified port
    if (bind(serverSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) == -1)
{
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(serverSocket, 5) == -1) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

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

    // Accept incoming connection
    socklen_t clientAddrLen = sizeof(clientAddr);
    if ((clientSocket = accept(serverSocket, (struct sockaddr *)&clientAddr,
&clientAddrLen)) == -1) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }

    // Read the IP address from the client
    recv(clientSocket, buffer, sizeof(buffer), 0);
    printf("Received IP address from client: %s\n", buffer);

    // Identify the network ID
    char networkId[20];
    identifyNetworkId(buffer, networkId);

    // Send the network ID back to the client
    send(clientSocket, networkId, sizeof(networkId), 0);
    printf("Sent network ID to client: %s\n", networkId);

    // Close sockets
    close(clientSocket);
    close(serverSocket);
```

```
    return 0;
}
```

Client Program –

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

#define PORT 8080
#define SERVER_IP "127.0.0.1"  // Use the actual IP address of your server

int main() {
    int clientSocket;
    struct sockaddr_in serverAddr;
    char ipAddress[INET_ADDRSTRLEN];

    // Create socket
    if ((clientSocket = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

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

    // Convert IPv4 address from text to binary form
    if (inet_pton(AF_INET, SERVER_IP, &serverAddr.sin_addr) <= 0) {
        perror("Invalid address/ Address not supported");
        exit(EXIT_FAILURE);
    }

    // Connect to the server
    if (connect(clientSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) == -
1) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    // Get and send the local IPv4 address to the server
    inet_ntop(AF_INET, &serverAddr.sin_addr, ipAddress, INET_ADDRSTRLEN);
    printf("Connected to server. Sending IP address: %s\n", ipAddress);
```

```
    send(clientSocket, ipAddress, sizeof(ipAddress), 0);

    // Receive the network ID from the server
    char networkId[20];
    recv(clientSocket, networkId, sizeof(networkId), 0);
    printf("Received network ID from server: %s\n", networkId);

    // Close socket
    close(clientSocket);

    return 0;
}
```

Command :

```
Surya : :~$ gcc as101server.c -o server

Surya : :~$ ./server

Server is listening on port 8080...
```

```
Surya : :~$ gcc as101client.c -o client

Surya : :~$ ./client

Connected to server. Sending IP address: 127.0.0.1
Received network ID from server: Class A
```

```
Surya : :~$ gcc as101server.c -o server

Surya : :~$ ./server

Connected to server...
Sent IP address to server: 192.168.1.1
Received network ID from server: Public Network
```

b) Write a client and server programs using TCP sockets, where a server maintains a list of countrycapital pair in its system. On successful connection, client sends the name of a country and in response serve replies the name of the capital of the country if it is present in the list; otherwise, sends 'NOT FOUND' back to the client.

Server Program –

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

#define PORT 8080
#define MAX_COUNTRIES 5

// Structure to store country-capital pairs
struct CountryCapital {
    char country[50];
    char capital[50];
};

// Function to search for the capital of a given country
const char *getCapital(const char *country, struct CountryCapital *countries, int
numCountries) {
    for (int i = 0; i < numCountries; ++i) {
        if (strcmp(country, countries[i].country) == 0) {
            return countries[i].capital;
        }
    }
    return "NOT FOUND";
}

int main() {
    int serverSocket, clientSocket;
    struct sockaddr_in serverAddr, clientAddr;

    // List of country-capital pairs
    struct CountryCapital countries[MAX_COUNTRIES] = {
        {"USA", "Washington, D.C."},
        {"India", "New Delhi"},
        {"France", "Paris"},
        {"Japan", "Tokyo"},
        {"Brazil", "Brasília"}
    };

    // Create socket
    if ((serverSocket = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }
```

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

    // Bind the socket to the specified port
    if (bind(serverSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) == -1)
{
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(serverSocket, 5) == -1) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

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

    // Accept incoming connection
    socklen_t clientAddrLen = sizeof(clientAddr);
    if ((clientSocket = accept(serverSocket, (struct sockaddr *)&clientAddr,
&clientAddrLen)) == -1) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }

    // Read the country name from the client
    char country[50];
    recv(clientSocket, country, sizeof(country), 0);
    printf("Received country name from client: %s\n", country);

    // Get the capital for the given country
    const char *capital = getCapital(country, countries, MAX_COUNTRIES);

    // Send the capital (or "NOT FOUND") back to the client
    send(clientSocket, capital, strlen(capital), 0);
    printf("Sent capital to client: %s\n", capital);

    // Close sockets
    close(clientSocket);
    close(serverSocket);
```

```
    return 0;
}
```

Client Program –

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

#define PORT 8080
#define SERVER_IP "127.0.0.1"  // Use the actual IP address of your server

int main() {
    int clientSocket;
    struct sockaddr_in serverAddr;
    char country[50], capital[50];

    // Create socket
    if ((clientSocket = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

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

    // Convert IPv4 address from text to binary form
    if (inet_pton(AF_INET, SERVER_IP, &serverAddr.sin_addr) <= 0) {
        perror("Invalid address/ Address not supported");
        exit(EXIT_FAILURE);
    }

    // Connect to the server
    if (connect(clientSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) == -
1) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    // Get the country name from the user
    printf("Enter the name of the country: ");
    scanf("%s", country);
```

```
    // Send the country name to the server
    send(clientSocket, country, strlen(country), 0);
    printf("Sent country name to server: %s\n", country);

    // Receive the capital from the server
    recv(clientSocket, capital, sizeof(capital), 0);
    printf("Received capital from server: %s\n", capital);

    // Close socket
    close(clientSocket);

    return 0;
}
```

Command :

```
Surya : :~$ gcc as101server.c -o server

Surya : :~$ ./server

Server is listening on port 8080...


Surya : :~$ gcc as101client.c -o client

Surya : :~$ ./client

Enter the name of the country: India
Sent country name to server: India
Received capital from server: New Delhi

Surya : :~$ gcc as101server.c -o server

Surya : :~$ ./server

Server is listening on port 8080...

Received country name from client: India
Sent capital to client: New Delhi
```

c) **Write a client and server programs using TCP sockets, where a concurrent TCP server maintains user credentials at its end. On successful connection, client enters username, and then password. Server verifies the username and password and replies either 'Success', 'Invalid user', or 'Invalid password' depending on the result of verification.**

Server Program –

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

#define PORT 8080
#define MAX_USERS 3

// Structure to store user credentials
struct User {
    char username[50];
    char password[50];
};

// Function to verify user credentials
const char *verifyCredentials(const char *username, const char *password, struct
User *users, int numUsers) {
    for (int i = 0; i < numUsers; ++i) {
        if (strcmp(username, users[i].username) == 0) {
            if (strcmp(password, users[i].password) == 0) {
                return "Success";
            } else {
                return "Invalid password";
            }
        }
    }
    return "Invalid user";
}

int main() {
    int serverSocket, clientSocket;
    struct sockaddr_in serverAddr, clientAddr;

    // List of user credentials
    struct User users[MAX_USERS] = {
        {"user1", "password1"},
        {"user2", "password2"},
```

```
        {"user3", "password3"}
    };

    // Create socket
    if ((serverSocket = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

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

    // Bind the socket to the specified port
    if (bind(serverSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) == -1)
{
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(serverSocket, 5) == -1) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

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

    while (1) {
        // Accept incoming connection
        socklen_t clientAddrLen = sizeof(clientAddr);
        if ((clientSocket = accept(serverSocket, (struct sockaddr *)&clientAddr,
&clientAddrLen)) == -1) {
            perror("Accept failed");
            exit(EXIT_FAILURE);
        }

        // Read the username from the client
        char username[50];
        recv(clientSocket, username, sizeof(username), 0);
        printf("Received username from client: %s\n", username);

        // Read the password from the client
        char password[50];
```

```
        recv(clientSocket, password, sizeof(password), 0);
        printf("Received password from client: %s\n", password);

        // Verify user credentials and send the result back to the client
        const char *result = verifyCredentials(username, password, users,
MAX_USERS);
        send(clientSocket, result, strlen(result), 0);
        printf("Sent verification result to client: %s\n", result);

        // Close the client socket
        close(clientSocket);
    }

    // Close the server socket (this part is not reachable in this example)
    close(serverSocket);

    return 0;
}
```

Client Program –

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

#define PORT 8080
#define SERVER_IP "127.0.0.1"  // Use the actual IP address of your server

int main() {
    int clientSocket;
    struct sockaddr_in serverAddr;
    char username[50], password[50], result[50];

    // Create socket
    if ((clientSocket = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Initialize server address structure
    memset(&serverAddr, 0, sizeof(serverAddr));
```

```
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(PORT);

    // Convert IPv4 address from text to binary form
    if (inet_pton(AF_INET, SERVER_IP, &serverAddr.sin_addr) <= 0) {
        perror("Invalid address/ Address not supported");
        exit(EXIT_FAILURE);
    }

    // Connect to the server
    if (connect(clientSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) == -
1) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    // Get the username from the user
    printf("Enter the username: ");
    scanf("%s", username);

    // Send the username to the server
    send(clientSocket, username, strlen(username), 0);
    printf("Sent username to server: %s\n", username);

    // Get the password from the user
    printf("Enter the password: ");
    scanf("%s", password);

    // Send the password to the server
    send(clientSocket, password, strlen(password), 0);
    printf("Sent password to server: %s\n", password);

    // Receive the verification result from the server
    recv(clientSocket, result, sizeof(result), 0);
    printf("Received verification result from server: %s\n", result);

    // Close socket
    close(clientSocket);

    return 0;
}
```

Command :

```
Surya : :~$ gcc as101server.c -o server
Surya : :~$ ./server
Server is listening on port 8080...
Surya : :~$ gcc as101client.c -o client
Surya : :~$ ./client
Enter the username: surya
Sent username to server: surya
Enter the password: 1234
Sent password to server: 1234
Received verification result from server: Success
```

```
Surya : :~$ gcc as101server.c -o server
Surya : :~$ ./server
Server is listening on port 8080...
Received username from client: surya
Received password from client: 1234
Sent verification result to client: Success
```

d) Write a client and server programs(using c)for interaction between server and client processes using UDP sockets. Perform the country-capital pair assignment. for UDP Socket.

Server Program –

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

```
#define PORT 8080
#define MAX_COUNTRIES 5

// Structure to store country-capital pairs
struct CountryCapital {
    char country[50];
    char capital[50];
};

// Function to search for the capital of a given country
const char *getCapital(const char *country, struct CountryCapital *countries, int
numCountries) {
    for (int i = 0; i < numCountries; ++i) {
        if (strcmp(country, countries[i].country) == 0) {
            return countries[i].capital;
        }
    }
    return "NOT FOUND";
}

int main() {
    int serverSocket;
    struct sockaddr_in serverAddr, clientAddr;
    socklen_t clientAddrLen = sizeof(clientAddr);

    // List of country-capital pairs
    struct CountryCapital countries[MAX_COUNTRIES] = {
        {"USA", "Washington, D.C."},
        {"India", "New Delhi"},
        {"France", "Paris"},
        {"Japan", "Tokyo"},
        {"Brazil", "Brasília"}
    };

    // Create socket
    if ((serverSocket = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

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

```
    // Bind the socket to the specified port
    if (bind(serverSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) == -1)
{
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

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

    while (1) {
        // Receive the country name from the client
        char country[50];
        ssize_t receivedBytes = recvfrom(serverSocket, country, sizeof(country), 0,
(struct sockaddr *)&clientAddr, &clientAddrLen);
        if (receivedBytes == -1) {
            perror("recvfrom failed");
            exit(EXIT_FAILURE);
        }

        // Get the capital for the given country
        const char *capital = getCapital(country, countries, MAX_COUNTRIES);

        // Send the capital (or "NOT FOUND") back to the client
        ssize_t sentBytes = sendto(serverSocket, capital, strlen(capital), 0, (struct
sockaddr *)&clientAddr, clientAddrLen);
        if (sentBytes == -1) {
            perror("sendto failed");
            exit(EXIT_FAILURE);
        }

        printf("Sent capital to client: %s\n", capital);
    }

    // Close the server socket (this part is not reachable in this example)
    close(serverSocket);

    return 0;
}
```

Client Program –

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

```
#define PORT 8080
#define SERVER_IP "127.0.0.1"  // Use the actual IP address of your server

int main() {
    int clientSocket;
    struct sockaddr_in serverAddr;

    // Create socket
    if ((clientSocket = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

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

    // Convert IPv4 address from text to binary form
    if (inet_pton(AF_INET, SERVER_IP, &serverAddr.sin_addr) <= 0) {
        perror("Invalid address/ Address not supported");
        exit(EXIT_FAILURE);
    }

    // Get the country name from the user
    char country[50];
    printf("Enter the name of the country: ");
    scanf("%s", country);

    // Send the country name to the server
    ssize_t sentBytes = sendto(clientSocket, country, strlen(country), 0, (struct
sockaddr *)&serverAddr, sizeof(serverAddr));
    if (sentBytes == -1) {
        perror("sendto failed");
        exit(EXIT_FAILURE);
    }

    // Receive the capital from the server
    char capital[50];
    ssize_t receivedBytes = recvfrom(clientSocket, capital, sizeof(capital), 0, NULL,
NULL);
    if (receivedBytes == -1) {
        perror("recvfrom failed");
        exit(EXIT_FAILURE);
    }
```

```
    printf("Received capital from server: %s\n", capital);

    // Close socket
    close(clientSocket);

    return 0;
}
```

Output –

```
Surya : :~$ gcc as101server.c -o server
Surya : :~$ ./server
Server is listening on port 8080...
Sent capital to client: New Delhi
```