**Assignment 4**

**Q.Implement Reader-Writer Problem using:**

a. using threads and semaphores.

b. using threads and mutex.

1. **using threads and semaphores:**

#!/bin/bash

# Create a temporary file to act as a lock

LOCK\_FILE="/tmp/reader\_writer.lock"

DATA\_FILE="/tmp/shared\_data.txt"

MAX\_READS=5

MAX\_WRITES=3

# Initialize shared data file

echo "0" > "$DATA\_FILE"

# Function for reader

reader() {

local id=$1

for ((i = 1; i <= MAX\_READS; i++)); do

(

flock -s 200

local data

data=$(<"$DATA\_FILE")

echo "Reader $id: read data = $data (iteration $i)"

) 200<$LOCK\_FILE

sleep 1

done

}

# Function for writer

writer() {

local id=$1

for ((i = 1; i <= MAX\_WRITES; i++)); do

(

flock -x 200

local data

data=$(<"$DATA\_FILE")

data=$((data + 1))

echo "$data" > "$DATA\_FILE"

echo "Writer $id: updated data to $data (iteration $i)"

) 200>$LOCK\_FILE

sleep 2

done

}

# Start readers and writers

for i in {1..3}; do

reader $i &

done

for i in {1..2}; do

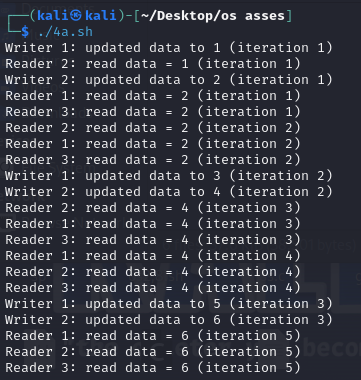
writer $i &

done

# Wait for all background processes to finish

wait

**OUTPUT:**



**b. using threads and mutex:**

#!/bin/bash

# Create a temporary lock file

MUTEX="/tmp/mutex.lock"

DATA\_FILE="/tmp/shared\_data.txt"

MAX\_READS=5

MAX\_WRITES=3

# Initialize shared data file

echo "0" > "$DATA\_FILE"

# Function for reader

reader() {

local id=$1

for ((i = 1; i <= MAX\_READS; i++)); do

# Readers only acquire a shared lock

flock -s 200

local data

data=$(<"$DATA\_FILE")

echo "Reader $id: read data = $data (iteration $i)"

flock -u 200

sleep 1

done 200<"$MUTEX"

}

# Function for writer

writer() {

local id=$1

for ((i = 1; i <= MAX\_WRITES; i++)); do

# Writers need exclusive access

flock -x 200

local data

data=$(<"$DATA\_FILE")

data=$((data + 1))

echo "$data" > "$DATA\_FILE"

echo "Writer $id: updated data to $data (iteration $i)"

flock -u 200

sleep 2

done 200>"$MUTEX"

}

# Start readers and writers

for i in {1..3}; do

reader $i &

done

for i in {1..2}; do

writer $i &

done

# Wait for all background processes to finish

wait

**OUTPUT:**

