**Assignment-3**

Name: Aastha Jain

Roll No: 39

PRN:12211635

Class: CS-B-2

**Reader Writer Problem**

**Code:**

#!/bin/bash

# Create named pipes for communication between reader and writer processes

mkfifo data\_fifo

mkfifo read\_signal\_fifo

# Writer process

writer() {

while true; do

echo "Writing data..."

echo "Data written" > data\_fifo

sleep 1

done

}

# Reader process

reader() {

while true; do

if read line < data\_fifo; then

echo "Reading data: $line"

echo "Data read" > read\_signal\_fifo

fi

done

}

# Main program

writer &

writer\_pid=$!

reader &

reader\_pid=$!

# Cleanup function to remove FIFOs when script exits

cleanup() {

kill $writer\_pid $reader\_pid

rm -f data\_fifo read\_signal\_fifo

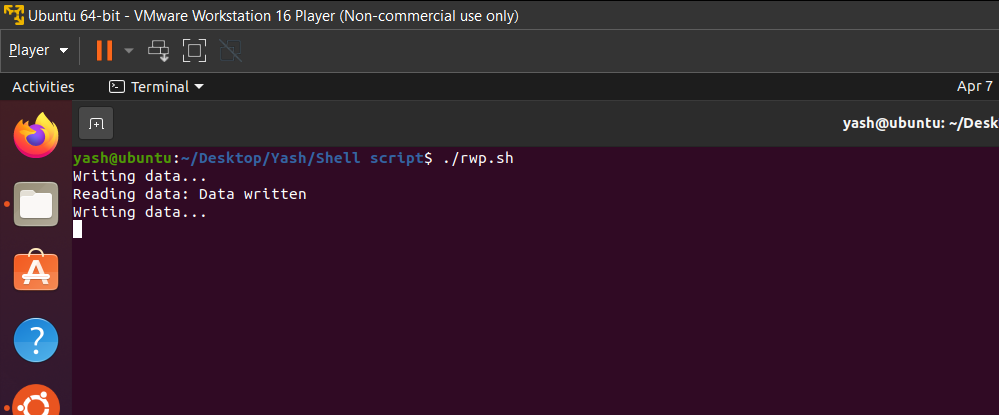
}

# Trap cleanup function on script exit

trap cleanup EXIT

# Wait for child processes to finish

wait



#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#include <unistd.h>

#define N 5a

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N

int state[N];

int phil[N] = { 0, 1, 2, 3, 4 };

sem\_t mutex;

sem\_t S[N];

void test(int phnum)

{

if (state[phnum] == HUNGRY

&& state[LEFT] != EATING

&& state[RIGHT] != EATING) {

// state that eating

state[phnum] = EATING;

sleep(2);

printf("Philosopher %d takes fork %d and %d\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is Eating\n", phnum + 1);

sem\_post(&S[phnum]);

}

}

void take\_fork(int phnum)

{

sem\_wait(&mutex);

state[phnum] = HUNGRY;

printf("Philosopher %d is Hungry\n", phnum + 1);

test(phnum);

sem\_post(&mutex);

sem\_wait(&S[phnum]);

sleep(1);

}

void put\_fork(int phnum)

{

sem\_wait(&mutex);

state[phnum] = THINKING;

printf("Philosopher %d putting fork %d and %d down\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is thinking\n", phnum + 1);

test(LEFT);

test(RIGHT);

sem\_post(&mutex);

}

void\* philosopher(void\* num)

{

while (1) {

int\* i = num;

sleep(1);

take\_fork(\*i);

sleep(0);

put\_fork(\*i);

}

}

int main()

{

int i;

pthread\_t thread\_id[N];

sem\_init(&mutex, 0, 1);

for (i = 0; i < N; i++)

sem\_init(&S[i], 0, 0);

for (i = 0; i < N; i++) {

pthread\_create(&thread\_id[i], NULL,

philosopher, &phil[i]);

printf("Philosopher %d is thinking\n", i + 1);

}

for (i = 0; i < N; i++)

pthread\_join(thread\_id[i], NULL);

}