21. Write a program to simulate the Dining Philosopher problem and verify your output with the following test case:

No of Philosophers: 5

THINKING – When philosopher doesn’t want to gain access to either fork.

HUNGRY – When philosopher wants to enter the critical section.

EATING – When philosopher has got both the forks, i.e., he has entered the section.

Philosopher i can set the variable state[i] = EATING only if her two neighbors are not eating

(state[(i+4) % 5] != EATING) and (state[(i+1) % 5] != EATING).

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define N 5 // number of philosophers

#define THINKING 0

#define HUNGRY 1

#define EATING 2

sem\_t forks[N]; // forks are represented by semaphores

sem\_t mutex; // semaphore to protect critical sections

int state[N]; // state of each philosopher

void test(int i) {

// check if philosopher i can start eating

if (state[i] == HUNGRY &&

state[(i + 4) % N] != EATING &&

state[(i + 1) % N] != EATING) {

state[i] = EATING;

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

sem\_post(&forks[i]); // signal the philosopher can start eating

}

}

void take\_forks(int i) {

sem\_wait(&mutex); // enter critical section

state[i] = HUNGRY;

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

test(i); // try to start eating

sem\_post(&mutex); // exit critical section

sem\_wait(&forks[i]); // wait for forks

}

void put\_forks(int i) {

sem\_wait(&mutex); // enter critical section

state[i] = THINKING;

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

test((i + 4) % N); // try to let neighbor start eating

test((i + 1) % N); // try to let neighbor start eating

sem\_post(&mutex); // exit critical section

}

void\* philosopher(void\* arg) {

int i = \*(int\*) arg;

while (1) {

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

sleep(rand() % 10); // thinking

take\_forks(i);

sleep(rand() % 10); // eating

put\_forks(i);

}

}

int main() {

int i;

pthread\_t philosophers[N];

sem\_init(&mutex, 0, 1); // initialize semaphore to 1

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

sem\_init(&forks[i], 0, 1); // initialize each fork to 1

state[i] = THINKING; // initialize state to thinking

}

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

pthread\_create(&philosophers[i], NULL, philosopher, &i);

}

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

pthread\_join(philosophers[i], NULL);

}

return 0;

}

