**Aim:** To implement Dinning Philosophers Problem with Semaphore.

**Algorithm:**

The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pickup the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.

Pseudocode:

*process P[i]*

*while true do*

*{ THINK;*

*PICKUP(CHOPSTICK[i], CHOPSTICK[i+1 mod 5]);*

*EAT;*

*PUTDOWN(CHOPSTICK[i], CHOPSTICK[i+1 mod 5])*

*}*

**Code:**

#include <stdio.h>

#include <semaphore.h>

#include <pthread.h>

#define N 5

#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]) has no effect during takefork

//used to wake up hungry philosophers during putfork

sem\_post(&S[phnum]);

}

}

//take up chopsticks

void take\_fork(int phnum)

{

//access critical section

sem\_wait(&mutex);

//state that hungry

state[phnum] = HUNGRY;

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

//eat if neighbours are not eating

test(phnum);

//leave critical section

sem\_post(&mutex);

//if unable to eat wait to be signalled

sem\_wait(&S[phnum]);

sleep(1);

}

//put down chopsticks

void put\_fork(int phnum)

{

//access critical section

sem\_wait(&mutex);

// state that thinking

state[phnum] = THINKING;

printf("Philosopher %d putting", phnum + 1);

printf(" fork %d and %d down\n", LEFT + 1, phnum + 1);

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

test(LEFT);

test(RIGHT);

//leave critical section

sem\_post(&mutex);

}

void \*philospher(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];

// initialize the semaphores

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

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

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

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

{

// create philosopher processes

pthread\_create(&thread\_id[i], NULL, philospher, &phil[i]);

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

}

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

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

return 0;

}

**Output(screenshots):**

// gcc Readers-writers.c -lpthread