Write a C Program to solve Dining philosophers problem.  
[Dining philosophers problem](http://msdn.microsoft.com/en-us/magazine/dd882512.aspx)is a classic [synchronization](http://www.cs.cf.ac.uk/Dave/C/node31.html) problem.A problem introduced by [Dijkstra](http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm) concerning resource allocation between processes. Five silent philosophers sit  around table with a bowl of spaghetti. A fork is placed between each pair of adjacent philosophers.   
  
Each philosopher must alternately think and eat.  
Eating is not limited by the amount of spaghetti left: assume an infinite supply.  
However, a philosopher can only eat while holding both the fork to the left and the fork to the right  
(an alternative problem formulation uses rice and chopsticks instead of spaghetti and forks).  
  
Each philosopher can pick up an adjacent fork, when available, and put it down, when holding it.  
These are separate actions: forks must be picked up and put down one by one.  
The problem is how to design a discipline of behavior (a concurrent algorithm) such that each philosopher won't starve, i.e. can forever continue to alternate between eating and thinking.

#include<stdio.h>

#include<semaphore.h>

#include<pthread.h>

#define N 5

#define THINKING 0

#define HUNGRY 1

#define EATING 2

#define LEFT (ph\_num+4)%N

#define RIGHT (ph\_num+1)%N

sem\_t mutex;

sem\_t S[N];

void \* philospher(void \*num);

void take\_fork(int);

void put\_fork(int);

void test(int);

int state[N];

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

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,philospher,&phil\_num[i]);

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

    }

    for(i=0;i<N;i++)

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

}

void \*philospher(void \*num)

{

    while(1)

    {

        int \*i = num;

        sleep(1);

        take\_fork(\*i);

        sleep(0);

        put\_fork(\*i);

    }

}

void take\_fork(int ph\_num)

{

    sem\_wait(&mutex);

    state[ph\_num] = HUNGRY;

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

    test(ph\_num);

    sem\_post(&mutex);

    sem\_wait(&S[ph\_num]);

    sleep(1);

}

void test(int ph\_num)

{

    if (state[ph\_num] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)

    {

        state[ph\_num] = EATING;

        sleep(2);

        printf("Philosopher %d takes fork %d and %d\n",ph\_num+1,LEFT+1,ph\_num+1);

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

        sem\_post(&S[ph\_num]);

    }

}

void put\_fork(int ph\_num)

{

    sem\_wait(&mutex);

    state[ph\_num] = THINKING;

    printf("Philosopher %d putting fork %d and %d down\n",ph\_num+1,LEFT+1,ph\_num+1);

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

    test(LEFT);

    test(RIGHT);

    sem\_post(&mutex);

}