```
/*
 *
 * dinphil.c
 *
 * Author: Michael Kepple (modifications from Tanenbaum code)
 * Date:   02 Apr 2013
 * Description: Robust solution to dining philosopher problem as outlined in
 *   Fig 2-20 of Tanenbaum OS textbook.
 *
*/
#include "dinphil.h"
int state[N];
int idNum[N] = {0,1,2,3,4};
sem_t mutex;
sem_t philosophers[N];

int main()
{
    int iter;
    pthread_t threads[N];
    sem_init(&mutex,0,1);
    for (iter=0; iter<N; iter++)
        sem_init(&philosophers[iter],0,0);
    for (iter=0; iter<N; iter++)
        pthread_create(&threads[iter], NULL, philosopher, &idNum[iter]);
    for (iter=0; iter<N; iter++)
        pthread_join(threads[iter],NULL);
}

/*
 * Function: philosopher
 * Description: main runnable thread for each philosopher. Will attempt to take
 *   forks, eat for five seconds, then put the forks down and wait it's turn to
 *   eat once more.
 * Params: philosopher ID number
 * Returns: Nothing (should never return).
 * Modifies: Nothing.
*/
void *philosopher(void *num)
{
    int *philosopher = num;
    printf("Philosopher %d has joined the table\n", *philosopher);
    for (;;)
    {
        take_forks(*philosopher);
        sleep(5);
        put_forks(*philosopher);
    }
}

/*
 * Function: take_forks
 * Description: philosophers will attempt to take both forks from their left and
 *   right. If they fail, they will block until another philosopher wakes them up
 *   by testing if they're now ready to eat.
 * Params: philosopher ID number
 * Returns: Nothing.
 * Modifies: Mutex, specific philosopher semaphore, philosopher state.
*/
void take_forks(int philosopher)
{
    // Enter critical section - lock other philosophers out
    sem_wait(&mutex);
```

```
        // Regardless of availability, this philosopher is now hungry
        state[philosopher] = HUNGRY;
        // Check if forks are available
        test_forks(philosopher);
        // Exit critical section
        sem_post(&mutex);
        // Wait here until forks available (if test_forks indicated so).
        sem_wait(&philosophers[philosopher]);
    }

    /*
     * Function: put_forks
     * Description: after getting their fill of spaghetti, indicated philosopher will
     *   put his forks back on the table and test to see whether any of his neighbors
     *   are now free to eat.
     * Params: philosopher ID number
     * Return: Nothing.
     * Modifies: mutex, philosopher state.
    */
    void put_forks(int philosopher)
    {
        // Enter critical section
        sem_wait(&mutex);
        // Philosopher has finished eating
        state[philosopher] = THINKING;
        printf("Philosopher %d is now thinking\n", philosopher);
        // See if left neighbor can now eat (now that I'm no longer using fork).
        test_forks(LEFT);
        // See if right neighbor can now eat
        test_forks(RIGHT);
        // Exit critical region
        sem_post(&mutex);
    }

    /*
     * Function: test_forks
     * Description: tests to see whether indicated philosopher has the requisite
     *   fork availability to commence eating (i.e. both his neighbors aren't).
     * Params: philosopher ID number
     * Returns: Nothing.
     * Modifies: philosopher state, specific philosopher semaphore.
    */
    void test_forks(int philosopher)
    {
        if (state[philosopher] == HUNGRY && state[LEFT] != EATING &&
                state[RIGHT] != EATING)
        {
            printf("Philosopher %d is now eating\n", philosopher);
            state[philosopher] = EATING;
            sem_post(&philosophers[philosopher]);
        }
    }
    /*
     * File:   dinphil.h
     * Author: Michael Kepple
     * Date:   02 Apr 2013
     *
    */
    #ifndef DINPHIL_H
    #define DINPHIL_H
    #include<stdio.h>
    #include<semaphore.h>
    #include<pthread.h>
```

```
#define N 5
#define LEFT (philosopher+N-1)%N
#define RIGHT (philosopher+1)%N
#define THINKING 0
#define HUNGRY 1
#define EATING 2
void * philosopher(void *);
void take_forks(int);
void put_forks(int);
void test_forks(int);
#endif
#
# Makefile for dinphil
# Author: Michael Kepple
# Date:   02 Apr 2013
#
all: dinphil.c dinphil.h
        gcc dinphil.c dinphil.h -lpthread -o dinphil

clean:
        rm -rf *.o dinphil
```