```
/*
 * Here is the dininig philosopher's problem. It is described in great
 * detail in the book.  Each philosopher will die if they try to pick up a
 * fork that isn't there (mutual exclusion), and will die if they don't eat
 * before too long -- 20 seconds (bounded wait, progress).  Your job is to
 * keep the philosophers alive using concurrency primitives: locks,
 * mutexes, semaphores in the pthread library.  You can put them in the
 * thread function (Philosopher), or you can put them in the fork pickup
 * function.
 *
 * Each philosopher (0-4) will try to take her fork and the fork above her
 * (philosopher 3 uses forks 3 and 4, philosopher 4 uses forks 4 and 0,
 * etc).
 */
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <unistd.h> // added by amaus

// Global array to keep track of which forks are available -- 1 means
// available, 0 means not. Philosopher i will eat with fork i, and i+1
// (addition done modulo 5, so philosopher 4 will use 4 and 0.
int forks[5] = {1,1,1,1,1};
pthread_mutex_t mutex[5];

/*
 * Function to pick up the forks -- it takes the philosopher number that is
 * picking up the forks, and the variable that represents the last time the
 * philosopher ate, so it can reset it. Philosophers are very grumpy
 * people, so if they want to eat and there are no forks they will just
 * starve themselves to death and return 1, which means their thread will
 * die.
*/
int pickup_forks(int philosopher_number, time_t *last_meal) {
  printf("Philosopher %d is attempting to eat.\n",philosopher_number);

  if (philosopher_number % 2 != 0){
    pthread_mutex_lock(&mutex[(philosopher_number+1)%5]);

    if(forks[(philosopher_number+1)%5] < 1)
    {
      printf("Philosopher %d wanted to eat but there was no fork to her right.\
n",philosopher_number);
      printf("She is furious and starved herself in protest\n");
      return 1;
    }
    else {
      forks[(philosopher_number+1)%5]--;
    }

    sleep(2);

    pthread_mutex_lock(&mutex[(philosopher_number)]);

    if(forks[philosopher_number] < 1)
    {
      printf("Philosopher %d wanted to eat but there was no fork to her left.\
n",philosopher_number);
```

```
      printf("She is furious and starved herself in protest\n");
      return 1;
    }
    else {
      forks[philosopher_number]--;
    }
  }

  if (philosopher_number % 2 == 0){
    pthread_mutex_lock(&mutex[(philosopher_number)]);

    if(forks[philosopher_number] < 1)
    {
      printf("Philosopher %d wanted to eat but there was no fork to her left.\
n",philosopher_number);
      printf("She is furious and starved herself in protest\n");
      return 1;
    }
    else {
      forks[philosopher_number]--;
    }

    sleep(2);

    pthread_mutex_lock(&mutex[(philosopher_number+1)%5]);

    if(forks[(philosopher_number+1)%5] < 1)
    {
      printf("Philosopher %d wanted to eat but there was no fork to her right.\
n",philosopher_number);
      printf("She is furious and starved herself in protest\n");
      return 1;
    }
    else {
      forks[(philosopher_number+1)%5]--;
    }
  }

  // If we just ate successfully, reset that philosopher's last_meal time
  time(last_meal);
  return 0;
}

// Function to return the forks
void return_forks(int philosopher_number)
{
  forks[(philosopher_number + 1)%5]++;
  forks[philosopher_number]++;

  pthread_mutex_unlock(&mutex[(philosopher_number+1)%5]);
  pthread_mutex_unlock(&mutex[(philosopher_number)]);


  printf("Philosopher %d has politely returned her forks. \n",philosopher_number);
}

// Our thread function -- seed the random number generator, and while
// (true), wait for 0-5 seconds (thinking), pickup forks (if it returns
// true or 1, we exit), then wait again for 0-5 seconds (eating) and return
```

```
// them
void *Philosopher(void *num)
{
  // Here we cast the argument as an integer and save it, as before
  int philosopher_number = *((int *)num);

  // Create variables to keep track of time, and set the last_meal time
  // to the start of the program
  time_t last_meal,current_time;
  time(&last_meal);


  // Seed the random number generator
  srand(time(NULL)+philosopher_number);

  while(1)
  {
    // If the time elapsed since the last meal of this philosopher is
    // more than 20 seconds, she dies
    time(&current_time);
    if(current_time - last_meal > 20 )
    {
      printf("Philosopher %d didn't eat for 20 seconds, so she died\
n",philosopher_number);
      return NULL;
    }

    // This represents thinking
    sleep(rand()%5);

    // If that function returned 1, then the philosopher couldn't pick
    // up a fork.
    if(pickup_forks(philosopher_number,&last_meal))
      continue;

    printf("Philosopher %d has started her meal!\n",philosopher_number);
    sleep(rand()%5);
    return_forks(philosopher_number);
  }
}

int main()
{
  for(int i=0;i<5;i++){
    pthread_mutex_init(&mutex[i],NULL);
  }
  pthread_t philosophers[5];

  int i=0;
  int args[5];
  for(i=0;i<5;i++)
  {
    args[i] = i;
    pthread_create(&philosophers[i],NULL,Philosopher,(void*)&args[i]);
  }

  for(i=0;i<5;i++)
  {
  pthread_join(philosophers[i],NULL);
```

```
  }
  return 0;
}
```