The dining philosophers problem

Five philosophers live in a house. The life of each philosopher consists principally of thinking and eating, and the only food that contributed to their thinking efforts was spaghetti. Due to a lack of manual skill, each philosopher requires two forks to eat spaghetti.

The eating arrangements are simple (Figure 6.11): a round table on which is set a large serving bowl of spaghetti, **five plates**, one for each philosopher, **and five forks**. A philosopher wishing to eat goes to his or her assigned place at the table and, using the two forks on either side of the plate, takes and eats some spaghetti.

The problem: Devise an algorithm that will allow the philosophers to eat. The algorithm must satisfy mutual exclusion (no two philosophers can use the same fork at the same time) while avoiding deadlock and starvation (in this case, the term has literal as well as algorithmic meaning!).

This problem illustrates basic problems in **deadlock** and **starvation**

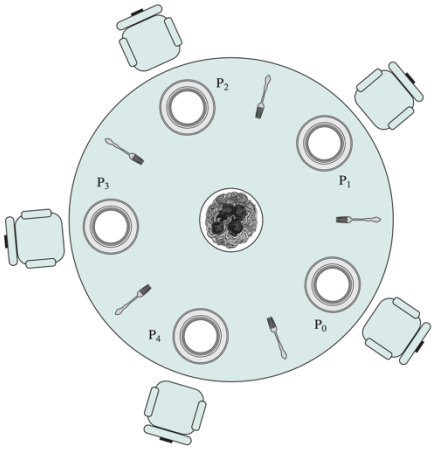

Figure 6.11 Dining Arrangement for Philosophers

```
/* program   diningphilosophers */
semaphore fork [5] = {1};
int i;
void philosopher (int i)
{
     while (true) {
          think();
          wait (fork[i]);
          wait (fork [(i+1) mod 5]);
          eat();
          signal(fork [(i+1) mod 5]);
          signal(fork[i]);
     }
}
void main()
{
     parbegin (philosopher (0), philosopher (1),
          philosopher (2),    philosopher (3),
          philosopher (4));
     }
```

Figure 6.12

SOLUTION #1: Using Semaphores → leads to Deadlock and Starvation

Each philosopher picks up first the fork on the left and then the fork on the right. After the philosopher is finished eating, the two forks are replaced on the table (De-allocation as in bankers algorithm).

To overcome the risk of deadlock, we could buy five additional forks or teach the philosophers to eat spaghetti with just one fork.

As another approach, we could consider adding an attendant who only allows four philosophers at a time into the dining room

Monitor

Code: Learn the code!!

SOLUTION #1: Using Monitor No deadlock

A vector of five condition variables is defined, one condition variable per fork. These condition variables are used to enable a philosopher to wait for the availability of a fork. In addition, there is a Boolean vector that records the availability status of each fork (true means the fork is available). The monitor consists of two procedures.

The get_forks procedure is used by a philosopher to seize his or her left and right forks. If either fork is unavailable, the philosopher process is queued on the appropriate condition variable. This enables another philosopher process to enter the monitor. The release-forks procedure is used to make two forks available.

Note that the structure of this solution is similar to that of the semaphore solution proposed in Figure 6.12 . In both cases, a philosopher seizes first the left fork and then the right fork. Unlike the semaphore solution, this monitor solution does not suffer from deadlock, because only one process at a time may be in the monitor. For example, the first philosopher process to enter the monitor is guaranteed that it can pick up the right fork after it picks up the left fork before the next philosopher to the right has a chance to seize its left fork, which is this philosopher's right fork.

```
/* program   diningphilosophers */
semaphore fork[5] = {1};
semaphore room = {4};
int i;
void philosopher (int i)
{
     while (true) {
          think();
          wait (room);
          wait (fork[i]);
          wait (fork [(i+1) mod 5]);
          eat();
          signal (fork [(i+1) mod 5]);
          signal (fork[i]);
          signal (room);
     }
}
void main()
{
     parbegin (philosopher (0), philosopher (1),
          philosopher (2), philosopher (3),
          philosopher (4));
}
```

Figure 6.13 **A Second Solution to the Dining Philosophers Problem**