```
monitor dining_controller;
enum states {thinking, hungry, eating} state[5];
cond needFork[5]                                      /* condition variable */

void get_forks(int pid)                 /* pid is the philosopher id number */
{
   state[pid] = hungry;                             /* announce that I'm hungry */
   if (state[(pid+1) % 5] == eating || (state[(pid-1) % 5] == eating)
      cwait(needFork[pid]);             /* wait if either neighbor is eating */
   state[pid] = eating;             /* proceed if neither neighbor is eating */
}

void release_forks(int pid)
{
   state[pid] = thinking;
   /* give right (higher) neighbor a chance to eat */
   if (state[(pid+1) % 5] == hungry) && (state[(pid+2) % 5]) != eating)
   csignal(needFork[pid+1]);
   /* give left (lower) neighbor a chance to eat */
   else if (state[(pid–1) % 5] == hungry) && (state[(pid–2) % 5]) != eating)
   csignal(needFork[pid–1]);
}
```

```
void philosopher[k=0 to 4]          /* the five philosopher clients */
{
   while (true) {
      <think>;
      get_forks(k);         /* client requests two forks via monitor */
      <eat spaghetti>;
      release_forks(k);     /* client releases forks via the monitor */
   }
}
```

Figure 6.18 Another Solution to the Dining Philosophers Problem Using a Monitor