```
module Waiter[t = 0 to 4]
  op getforks(int), relforks(int);  # for philosophers
  op needL(), needR(),              # for waiters
     passL(), passR();
  op forks(bool,bool,bool,bool);  # for initialization
body
  op hungry(), eat();      # local operations
  bool haveL, dirtyL, haveR, dirtyR;  # status of forks
  int left = (t-1) % 5 ;       # left neighbor
  int right = (t+1) % 5;       # right neighbor

  proc getforks() {
    send hungry();  # tell waiter philosopher is hungry
    receive eat();  # wait for permission to eat
  }

  process the_waiter {
    receive forks(haveL, dirtyL, haveR, dirtyR);
    while (true) {
      in hungry() ->
          # ask for forks I don't have
          if (!haveR) send Waiter[right].needL();
          if (!haveL) send Waiter[left].needR();
          # wait until I have both forks
          while (!haveL or !haveR)
            in passR() ->
                haveR = true; dirtyR = false;
            [] passL() ->
                haveL = true; dirtyL = false;
            [] needR() st dirtyR ->
                haveR = false; dirtyR = false;
                send Waiter[right].passL();
                send Waiter[right].needL()
            [] needL() st dirtyL ->
                haveL = false; dirtyL = false;
                send Waiter[left].passR();
                send Waiter[left].needR();
            ni
          # let philosopher eat, then wait for release
          send eat(); dirtyL = true; dirtyR = true;
          receive relforks();
      [] needR() ->
          # neighbor needs my right fork (its left)
          haveR = false; dirtyR = false;
          send Waiter[right].passL();
      [] needL() ->
          # neighbor needs my left fork (its right)
          haveL = false; dirtyL = false;
          send Waiter[left].passR();
      ni
    }
  }
end Waiter
```

```
process Philosopher[i = 0 to 4] {
  while (true) {
    call Waiter[i].getforks();
    eat;
    call Waiter[i].relforks();
    think;
  }
}

process Main {  # initialize the forks held by waiters
  send Waiter[0].forks(true, true, true, true);
  send Waiter[1].forks(false, false, true, true);
  send Waiter[2].forks(false, false, true, true);
  send Waiter[3].forks(false, false, true, true);
  send Waiter[4].forks(false, false, false, false);
}
```

Figure 9.21 Decentralized dining philosophers.