Dining philosophers:
An exercise in message passing and state

CS21b: Structure and Interpretation of Computer Programs

Spring Term, 2015

Raphael, The School of Athens
Dining philosophers:

Seat \( n \) philosophers around a table. One fork between each philosopher. Philosophers either think (away from table)...

...or eat (arrive at preassigned seat, pick up fork to left and right, start eating).

Shared resource: the forks.

Parable: how do independent, asynchronous parallel processes share resources?
Deadlock!
How to make a philosopher
(define (make-philosopher name)
  (let ((left-fork '())
          (right-fork '())
          (what-i-am-doing 'thinking))
    (define (eating?) (eq? what-i-am-doing 'eating))
    (define (thinking?) (eq? what-i-am-doing 'thinking))
    (define (thinker m)
      (cond
       ((eq? m 'name) (list 'philosopher name))
       ((eq? m 'thinking?) (thinking?))
       ((eq? m 'eating?) (eating?))
       ((eq? m 'status)
        (list 'philosopher name
              (list '(left fork) (left-fork 'name))
              (list '(right fork) (right-fork 'name))
              (list 'what-i-am-doing? what-i-am-doing)))
       ((eq? m 'load-left-fork)
        (lambda (fork)
          (set! left-fork fork)
          'left-fork-loaded))
       ((eq? m 'load-right-fork)
        (lambda (fork)
          (set! right-fork fork)
          'right-fork-loaded))))
  ;; to be continued
((eq? m 'think!)
 ; You can only start thinking if you are currently eating
 (if (eating?)
  (begin
   ((left-fork 'put-down!) thinker)
   ((right-fork 'put-down!) thinker)
   (set! what-i-am-doing 'thinking)
   (list 'philosopher name 'thinking))
 ; If you are not eating, you are already thinking
 (cons (list 'philosopher name)
         '(already thinking!)))

 ;; to be continued
((eq? m 'eat!)
 ; You can only start eating if you are currently thinking
 (if (thinking?)
  (if ((left-fork 'grab!) thinker)
   (if ((right-fork 'grab!) thinker)
    ; Both forks successfully grabbed
    (begin
      (set! what-i-am-doing 'eating)
      (list 'philosopher name 'eating))
    ; Grabbed left OK, but right fork already
    ; taken...
    ; So you failed:
    ; put left fork down, keep thinking...
    (begin
      ((left-fork 'put-down!) thinker)
      'i-am-hungry-but-still-thinking))
  ; Failed to grab left fork...
  'i-am-hungry-but-still-thinking)
 ; If you are not thinking, you are already eating
 (cons (list 'philosopher name) '(already eating!)))
 (else (error "What ?")))
 thinker)
How to make a fork
(define (make-fork name)
  (let ((left-philosopher '())
         (right-philosopher '())
         (fork-held-by '()))
    (define (fork-raised?) (not (null? fork-held-by)))
    (define (fork m)
      (cond
        ((eq? m 'name)  (list 'fork name) )
        ((eq? m 'status)  (list
          ((eq? m 'status)  (list
            (fork 'name)
            (list '(left-philosopher)
              (left-philosopher 'name))
            (list '(right-philosopher)
              (right-philosopher 'name))
            (list '(fork raised?) (fork-raised?)))))
        ((eq? m 'load-left-philosopher)
          (lambda (thinker)
            (set! left-philosopher thinker)
            'left-philosopher-loaded))
        ((eq? m 'load-right-philosopher)
          (lambda (thinker)
            (set! right-philosopher thinker)
            'right-philosopher-loaded))
        ;; to be continued
(define (make-table n)
 (let ((count (integers-from 1 n)))
   (let ((thinkers
     (map (lambda (x) (make-philosopher x)) count))
   (forks
     (map (lambda (x) (make-fork x)) count)) )
   (linkup thinkers forks)
   (cons thinkers forks))))

(define (linkup thinkers forks)
 (define (link t-list f-list)
   (let ((first-thinker (car t-list))
     (left-fork (car f-list))
     (right-fork (cadr f-list))
     ((first-thinker 'load-left-fork) left-fork)
     ((first-thinker 'load-right-fork) right-fork)
     ((left-fork 'load-right-philosopher) first-thinker)
     ((right-fork 'load-left-philosopher) first-thinker)
   (if (not (null? (cdr t-list)))
     (link (cdr t-list) (cdr f-list)))
   (link thinkers (cons (last forks) forks))))
Claim: Four philosophers will not deadlock.
Q: What if one philosopher keeps eating and thinking and eating and thinking, real fast?

A: The neighboring philosophers get locked out of eating...