Quiz 14a

1. (a) (2 points) Suppose we say

> (define x 7)

> (define s1 (make-serializer))

> (define s2 (make-serializer))

> (parallel-execute (s1 (lambda () (set! x (\* x 2))))

(s2 (lambda () (set! x (+ x 3)))))

What are all the possible values of x after this finishes? If there

can be a deadlock, write DEADLOCK.

17, 20, 14, 10

(b) (1 point) Which of the values you wrote above are correct values of x?

17, 20

2. (4 points) Ping is a utility program used to find out if another computer is on the

network; the basic idea is that one computer sends the other a PING message and

the other sends the first one back a PONG message. If the first computer

receives the PONG, it knows the second is running and prints out a message

saying so. Here's a simplified example:

ping star.cs.berkeley.edu

star.cs.berkeley.edu is online.

The following procedure is meant to be used as a callback for any network

activity. Fill in the blanks to make it handle PING and PONG

messages correctly.

(define (receive-message message other-computer)

(cond ((eq? message ‘ping)

(send-message other-computer ‘pong)

((eq? message ‘pong)

(display other-computer)

(display " is online.")

(newline))

(else

(error "Invalid message received: " name))))

; you may assume this procedure is already defined

(define (send-message message other-computer) ...)

3. (3 points) Here is a proposed solution to the dining philosophers problem, supposing

there are five philosophers and five chopsticks. Chopstick 0 is to the

left of philosopher 0, etc.

(define chopstick-serializers (list (make-serializer)

(make-serializer)

(make-serializer)

(make-serializer)

(make-serializer))

(define chopsticks (list #f #f #f #f #f)) ; will be #T if chopstick busy

(define (philosopher num)

(think)

(get-chopstick num) ; left chopstick

(get-chopstick (remainder (+ num 1) 5)) ; right chopstick

(eat)

(release-chopstick num)

(release-chopstick (remainder (+ num 1) 5))

(philosopher num))

(define (get-chopstick num)

(if (((list-ref chopstick-serializers num) ; find the serializer

(lambda () ; serialize this procedure

(let ((ch ((repeated cdr num) chopsticks)))

(if (car ch) ; if chopstick is in use...

#t ; return #T to the IF

(begin

(set-car! ch #t) ; otherwise mark it busy

#f)))))) ; and return #F to the IF

(get-chopstick num)))

(define (release-chopstick num)

(set-car! ((repeated cdr num) chopsticks) #f))

(parallel-execute (lambda () (philosopher 0))

(lambda () (philosopher 1))

(lambda () (philosopher 2))

(lambda () (philosopher 3))

(lambda () (philosopher 4))

(a) Suppose that get-chopstick is correct (and note that it provides

serialization). Does philosopher exhibit (pick the correct answer):

(A) possible incorrect results

***(B) possible deadlock (2 points all or nothing)***

***(C) inefficiency (reduced parallelism)***

(D) none of the above

**B, C or B and C**

(b) Suppose that philosopher is correct. Does get-chopstick

exhibit (pick the correct answer):

(A) possible incorrect results

(B) possible deadlock

(C) inefficiency (reduced parallelism)

***(D) none of the above (1 point all or nothing)***