Paradigmas de Programação – prova II (hipotética)

Ex. 1 — Explique o que são:

a)Trava para leitores e escritores

b)Memória transacional por software

Ex. 2 — É possível ocorrer *deadlock* em um programa sem variáveis compartilhadas, usando apenas trocas assíncronas de mensagens (como no modelo actor)? Diga porque não, ou mostre porque sim.

Ex. 3 — Em um jogo multiusuário distribuído um jogador C, que tinha no mundo virtual a quantia de $400, uma lanterna e um livro, morreu. Os jogadores A e B (que nada tinham) aproximaram-se dos pertences do jogador C tentando obtê-los. O jogador A saiu dali com uma lanterna e $400; o jogador C ficou com um livro, uma lanterna e $400. Explique um possível erro do programador, mostrando sua hipótese de como o programa se comportou, passo a passo.

Ex. 4 — Nossa implementação de rendezvous usando semáforos é mostrada a seguir:

```
(define make-sema-rendezvous
  (lambda ()
    (let ((a-ok (make-semaphore 0))
          (b-ok (make-semaphore 0)))
      (list (lambda ()
              (semaphore-signal-by! a-ok 1)
              (semaphore-wait! b-ok))
            (lambda ()
              (semaphore-signal-by! b-ok 1)
              (semaphore-wait! a-ok))))))
```

Ao traduzí-la diretamente para usar mensagens assíncronas, obtemos o seguinte código:

```
(define make-msg-rendezvous
  (lambda ()
    (let ((m (make-mutex))
          (a-ok (make-condition-variable))
          (b-ok (make-condition-variable)))
```

```
(mutex-lock! m)
(list (lambda ()
        (condition-variable-signal! a-ok)
        (mutex-unlock m b-ok))
      (lambda ()
        (condition-variable-signal! b-ok)
        (mutex-unlock m a-ok))))))
```

a)Explique porque esta versão não funciona.

b)Se trocarmos, em cada procediento interno, a ordem das linhas que fazem condition-variable-signal! e mutex-unlock!, resolveríamos o problema? Porque?