CI4251 - Programación Funcional Avanzada
Tarea 3

Stefano De Colli
09-10203

Junio 6, 2015

Clásico - MVars

En primera instancia tenemos a Rafita, para que ella empiece a cocinar se deben cumplir dos condiciones, primero que el colador donde sirve las empanadas esté vacío y que algún parroquiano esté hambriento, si esto se cumple empezará a cocinar m empanadas, y luego de 3 a 5 segundos las pone en el colador.

```
rafitaC m bowl total stall buffer =forever $ do
  b ← takeMVar bowl
  s ← takeMVar stall
  if b >0 || s then
    do
      putMVar stall s
      putMVar bowl b
  else
    do
      t ← takeMVar total
      buff ←takeMVar buffer
      putMVar buffer (buff |>"Rafita esta cocinando")
      randomDelay 3000000 5000000
      buff2 ←takeMVar buffer
      putMVar buffer (buff |>"Rafita sirvio las empanadas")
      putMVar total $ t +m
      putMVar stall True
      putMVar bowl m
```

Luego tenemos al parroquiano, en principio el sólo dirá que tiene hambre una sola vez, luego de eso se quedará esperando hasta que hayan empanadas (Para eso es el último parámetro). Sí logró agarrar una empanada, entonces va a beber, tardando entre 1 a 7 segundos antes de que le dé hambre nuevamente. Si no logra agarrar empanada, intentará comer después.

```
parroquiano i counter bowl buffer stall bool =do
  if bool then
    do
      buff ←takeMVar buffer
      putMVar buffer (buff |>("Parroquiano " ++(show i) ++" tiene hambre"))
      comer
  else
    comer

  where
    agarrarEmpanada =do
      b ← takeMVar bowl
      if b >0 then
        do
          c ← takeMVar counter
          putMVar counter $ c +1
          buff ←takeMVar buffer
          putMVar buffer (buff |>("Parroquiano " ++(show i) ++" come empanada"))
          putMVar bowl $ b - 1
      else
        putMVar bowl 0

    comer =do
      pre ←readMVar counter
      agarrarEmpanada
      post ←readMVar counter
      if pre ==post then
        do
          takeMVar stall
          putMVar stall False
          parroquiano i counter bowl buffer stall False
      else
        do
          randomDelay 1000000 7000000
          parroquiano i counter bowl buffer stall True
```

Esta función imprimirá lo que exista en el buffer, si es que hay.

```
outputC buffer =do
    str ←takeMVar buffer
    case viewl str of
      EmptyL       → do
                   putMVar buffer str
                   threadDelay 1
                   outputC buffer
      item :< rest →do
                   putStrLn item
                   putMVar buffer rest
                   outputC buffer
```

Y la función para inciar la simulación clasica, generamos los MVars para guardar la información, obtenemos los ThreadsID de Rafita y de cada parroquiano, y instalamos una función que se ejecutará al presionar Crtl-C

```
classic m n =do
      setStdGen $ mkStdGen randomSeed
      counters ←replicateM n newMVar 0
      bowl ←newMVar 0
      total ←newMVar 0
      buffer ←newMVar DS.empty
      stall ←newMVar True
      mainTID ←myThreadId
      parroquianosTID ←forM [1..n] $ (λi →
        forkIO $ parroquiano i (counters !! (i - 1)) bowl buffer stall True)
      rafitaTID ←forkIO $ rafitaC m bowl total stall buffer

      installHandler sigINT (Catch (
        printInfoC total counters rafitaTID parroquianosTID mainTID)) Nothing
      outputC buffer
```

La función que atrapa el Crtl-C hace lo siguiente, saca de los MVars el total de empanadas hechas por Radita y el contador de cada Parroquiano, una vez hecho esto, todos los hilos deberían estar bloqueados, así que matamos a todos los hilos, esperamos unos segundos[1] y luego imprimimos las estadisticas. Finalmente matamos al hilo principal (Que está imprimiendo del buffer compartido)[2].

```
printInfoC total counters rTID pTID mTID =do
  t ← readMVar total
  cs ← mapM readMVar counters
  forM_ pTID (λt →killThread t)
  killThread rTID
  threadDelay 200000
  putStrLn $ "λnλnRafita preparo " ++(show t) ++" empanadas"
  let ps =zip cs [1..]
  forM ps (λ(c, i)→
     putStrLn $ "Parroquiano " ++(show i) ++":λt" ++(show c))
  putStrLn $ "Total: " ++(show $ sum cs)

  killThread mTID
```

[1] Esto se debe a que el hilo principal todavía está intentando imprimir, así que esperamos para que varios hilos no intenten imprimir en la consola simultaneamente

[2] Si el programa fuese compilado y no ejecutado por GHCI, matar al hilo principal no sería necesario, pero esto es un problema ya que GHCI no mata los hilos al terminar la función, para más información, ver acá

Transaccional - TVars

Similar al modelo clásico, sólo que en el transaccional no tenemos que reservar las variables en un orden particular y no usamos forever, mas bien recursión

```
rafitaT m bowl total stall buffer =do
  b ← readTVarIO bowl
  s ← readTVarIO stall
  if b >0 || s then
    rafitaT m bowl total stall buffer
  else
    do
      atomically $ writeTVar stall True »put buffer "Rafita esta cocinando"
      randomDelay 3000000 5000000
      atomically $ do
           modifyTVar' total (incM m)
           writeTVar bowl m
           put buffer "Rafita sirvio las empanadas"
           writeTVar stall True
      rafitaT m bowl total stall buffer
```

No hay mucho que explicar, sigue el mismo sentido que en el esquema clásico.

```
parroquiano i counter bowl buffer stall bool =do
  if bool then
    do
      atomically $ put buffer $ "Parroquiano " ++(show i) ++" tiene hambre"
      comer
  else
    do
      comer
      threadDelay 1

  where
    agarrarEmpanada =do
      b ← readTVar bowl
      if b >0 then
        do
          writeTVar bowl $ b - 1
          modifyTVar' counter inc
          put buffer $ "Parroquiano " ++(show i) ++" come empanada"
      else
        return ()
    comer =do
      pre ←readTVarIO counter
      atomically $ agarrarEmpanada
      post ←readTVarIO counter
      if pre ==post then
        do
          atomically $ writeTVar stall False
          parroquiano i counter bowl buffer stall False
      else
        do
          randomDelay 1000000 7000000
          parroquiano i counter bowl buffer stall True
```

Para imprimir el buffer compartido, es muy sencillo.

```
outputT buffer =
    do str ←atomically $ get buffer
       putStrLn str
       outputT buffer
```

Y para iniciar la simulación es casi que lo mismo.

```
transactional m n =do
  setStdGen $ mkStdGen randomSeed
  counters ←replicateM n newTVarIO 0
  bowl ←newTVarIO 0
  total ←newTVarIO 0
  stall ←newTVarIO True
  buffer ←newTVarIO DS.empty
  mainTID ←myThreadId
  parroquianosTID ←forM [1..n] $ (λi →
    forkIO $ parroquiano i (counters !! (i - 1)) bowl buffer stall True)
  rafitaTID ←forkIO $ rafitaT m bowl total stall buffer
  installHandler sigINT (Catch (
    printInfoT total counters rafitaTID parroquianosTID mainTID)) Nothing
  outputT buffer
```

Por último tenemos a la función encargada de atrapar al Crtl-C. El único cambio importante es que en el transaccional no hay que esperar por los otros hilos, esto se debe por el concepto de una transacción, si el hilo se destruye antes de que finalice, nada de los cambios tendrán efecto.

```
printInfoT total counters rTID pTID mTID =do
  forM_ pTID (λt →killThread t)
  killThread rTID
  buffer ←newTVarIO DS.empty
  t ← readTVarIO total
  cs ← mapM readTVarIO counters
  putStrLn $ "λnλnRafita preparo " ++(show t) ++" empanadas"
  let ps =zip cs [1..]
  forM ps (λ(c, i)→
     putStrLn $ "Parroquiano " ++(show i) ++":λt" ++(show c)
   )
  putStrLn $ "Total: " ++(show $ sum cs)

  killThread mTID
```