CI4251 - Programación Funcional Avanzada
Tarea 4

Ernesto Hernández-Novich
86-17791
<emhn@usb.ve>

Junio 21, 2013

1. Concurrencia (5 puntos)

El tipo de datos `Chan` está implantado aprovechando `MVar`. Use `MVar` para desarrollar una librería que provea el tipo de datos abstracto `BoundedChan` junto con un API idéntico al ofrecido por `Chan`.

La única diferencia entre ambos APIs es que la función `newBoundedChan` definida en su módulo recibe un parámetro adicional `Int` que indica el número máximo de items sin leer que pueden estar en el `BoundedChan`.

Si se alcanza dicho límite, las llamadas a `writeBoundedChan` deben bloquearse hasta que algún hilo use `readBoundedChan` para consumir algún valor.

Escriba una función principal

```
bounded :: IO ()
```

que demuestre la correcta operación de su librería combinando dos hilos: un productor "rápido" que utilice un `BoundedChan` para suministrar valores hacia un consumidor "lento".

2. Paralelismo (6 puntos)

El **colapso** de un número entero corresponde a la suma de sus dígitos, proceso que puede seguirse recursivamente hasta llegar a un dígito final. Por ejemplo, el colapso de 134957 es 29, que a su vez colapsa en 11 que finalmente colapsa en 2.

Definiremos el colapso de un arreglo de enteros como el colapso de la suma de los colapsos de los enteros que le componen.

Escriba una función no paralela

```
collapse :: [Int] -> Int
```

que calcule el colapso de un arreglo de enteros. Evalúe su desempeño trabajando sobre arreglos que tomen al menos dos minutos en ser procesados hasta obtener el resultado (¡aproveche `QuickCheck`!).

Ahora escriba

```
pcollapse :: [Int] -> Int
```

que efectúe el mismo cálculo pero empleando paralelismo implícito.

Compare el desempeño de las funciones `collapse` y `pcollapse` operando sobre un sólo núcleo. Luego, ajuste `pcollapse` para obtener el máximo desempeño posible sobre **dos** núcleos.

3. La cena mexicana (9 puntos)

Una flauta requiere tres ingredientes:

- Una tortilla.
- Relleno de carne.
- Guacamole.

Alrededor de una mesa se sientan tres mariachis, cada uno de los cuales tiene un suministro *infinito* de los ingredientes – uno tiene tortillas, el segundo tiene relleno y el tercero tiene guacamole.

Como están un poco pasados de tragos, un cuarto mariachi está tratando de que coman algo hasta pasar la "cruda" para irse a amenizar alguna fiesta. Este mariachi selecciona al azar, a **dos** de los tres que están sentados, les pide los ingredientes que proveen y los pone en la mesa. Luego, le notifica al tercer mariachi, que toma los ingredientes de la mesa, los combina con el propio para preparar una flauta, y se la come.

Sin esperar a que termine de comer, el cuarto mariachi vuelve a seleccionar arbitrariamente a **dos** de los tres que están sentados, obtiene los ingredientes, y notifica al tercero para que coma. Si el tercero se encontraba comiendo, cuando termine de comer se prepara la siguiente flauta. Los mariachis comen a velocidad diferente cada vez. El cuarto mariachi, curiosamente, nunca come.

Este proceso continúa indefinidamente.

Proponga una solución para esta cena empleando `STM`, de manera tal que ninguno de los tres mariachis sentados a la mesa se quede sin comer. El programa debe mostrar la actividad de los mariachis indicando para cada flauta preparada quienes ponen los ingredientes, quien la prepara y cuando la come.

Provea la función

```
mariachis :: IO ()
```

que sirva para iniciar la simulación.