Продолжения и Сопрограммы (Continuations and CoRoutines)

Многозадачность

Вытесняющая многозадачность

Наиболее популярная реализация многозадачности в современных операционных системах на сегодняшний день. В такой реализации ОС сама определяет когда прервать выполнение приложения. Чаще всего решение принимается по исчерпанию процессом своей квоты по времени или по какому - либо сигналу. Преимуществом такого подхода является то, что можно гарантировать работоспособность системы в целом, в случае сбоев выполняющихся приложений. Кроме того с вытесняющей многозадачностью проще работать на системах с большим количеством процессоров

Т.к. процесс может быть остановлен, практически в любом месте, возникает необходимость очень аккуратно работать с разделяемым состоянием, т.к. просто оказаться в неконсистентном состоянии. Для решения проблем с состоянием применяют мьютексы, семафоры и другие примитивы, которые в свою очередь, порождают ряд новых проблем, таких как dead и live locks, процессорное голодание, гонки и т.д.

Многозадачность

Кооперативная многозадачность

Подход при котором, каждый процесс сам решает, когда отдать ресурсы.Основная проблема реализации больших систем основанных на кооперативной многозадачности в том, что все процессы должны следовать одному протоколу и следить за тем, чтобы ресурс не остался занят ими, даже в случае фатальной проблемы. Тем не менее, в небольших масштабах КА становится очень удобна, т.к. позволяет разработать приложения так, чтобы они никогда не оставались в неконситетном состоянии при передаче ресурса другому процессу. Еще одним преимуществом является то, что при правильном проектировании, уменьшается потребность в разделяемом состоянии.

Многозадачность

Многозадачность, асинхронность, конкурентность.

Перед тем, как двинуться дальше, нужно освежить в памяти в чем заключаются различия между концепция приведенными выше.

- **многозадачность** - как следует из названия, это особенность устройства, ОС и т.д. выполнять много задач. Это наиболее общее понятие, которое включает в себя оба оставшихся
- **асинхронность -** это возможность нескольких процессов (вычислений) выполняться асинхронно, т.е. независимо друг от друга. При это они не обязаны выполняться параллельно. достаточно вспомнить JS, где полно асинхронности, но мало конкурентности
- **конкурентность(параллельность)**. Это та самая возможность, выполняться нескольким вычислениям в одно и тоже время. Очень часто, когда говорят о многозадачности, имеют ввиду именно конкурентность.

Продолжения(continuations) в scala

Продолжения

Последняя концепция, с которой надо быть знакомым, чтобы лучше понять суть сопрограмм - это **продолжения** (continuations). Любой программист на JS знает, что это такое. По сути это лямбда или функция, переданная в явном виде в другую функцию. При этом, **продолжение** обязательно должно быть хвостовым вызовом, то есть вызвано обязательно последним. Это позволит компилятору оптимизировать стек.

Продолжения делают явным порядок вызовов подпрограмм и их интерфейсы. Они часто применяются для приложений с большим количеством асинхронных операций, например в программировании интерфейсов.

Продолжения бывают

- не разделяемые (undelimeted). Такое продоление по сути - это вся программы записанная в виде одной цепочки продолжений. Такая форма не сильно отличается от GOTO и в подавляющем большинстве случаев не применима
- разделяемые (delimeted). Такие продолжения больше похожи на функции, они имеют выходное значение и могут быть скомпозированы.

Продолжения(continuations) в scala

На данный момент, в scala, активно поддерживается только одна библиотека, построенная на принципах delimeted continuations - **scala-async.** Она предназначена для реализации асинхронных параллельных задач и представляет собой альтернативу стандартному подходу работы с Futures.

lectures.concurrent.coroutines.ContinuationsExampleTest

Сопрограммы в scala

Сопрограмма (coroutine) - это обобщение функции. Отличительными чертами всех сопрограмм является

- возможность остановиться в произвольном месте, сохранив свое состояние
- возможность возобновить свою работу, начиная с состояния в котором она была остановлена в предыдущий раз
- сопрограмма может вернуть результат несколько раз

Это один из базовых способов реализовать многозадачность в кооперативном стиле

В scala для реализации сопрограмм, есть библиотека http://storm-enroute.com/coroutines/learn/
К большому сожалению из-за бага в рефлексии в 2.12 эта библиотека пока не доступна. Тем не менее авторы обещают сделать поддержку будущих версий scala, когда проблема будет устранена.

Сопрограммы в scala

Разберем небольшой пример, который можно запустить на scala 2.11.x

```
val range = coroutine { (n: Int) => // объявление сопрограммы
 var i = 0
 while (i < n) {
   yieldval(i)  // сопрограмма останавливается, сохраняя временный результат.
              // она будет остановлена до тех пор, пока не будет вызван resume
   i += 1
 }
}

def extract(c: Int <~> Unit): Seq[Int] = {
 var xs: List[Int] = Nil
 while (c.resume) if (c.hasValue) xs ::= c.value //c.resume - вызов сопрограммы
 xs.reverse
}
val instance = call(range(10)) //создание инстанса сопрограммы
val elems = extract(instance)
assert(elems == 0 until 10)
```