
Parallelism and Concurrency

Exam Solution

Wednesday, May 30, 2018

Exercise 1: Spark (20 points)

Question 1

```
items.filter(_.price > 100000).map(_.name).collect()
```

Question 2

```
orders.filter(isLastBlackFriday(_.date)).map(_.itemIds.size).reduce(_ + _)
```

Question 3

```
orders.flatMap((order: Order) => order.itemIds.map((itemId: Int) => (itemId, order.id)))  
  .join(items.map((item: ItemKind) => (item.id, item.price)))  
  .values  
  .reduceByKey(_ + _)  
  .max(Ordering.by(_._2))
```

Exercise 2: ??? (20 points)

```
val average: Future[Int] =  
  for {  
    g1 <- getGrade(s1)  
    g2 <- getGrade(s2)  
    g3 <- getGrade(s3)  
  } yield (g1 + g2 + g3)
```

```
val f1 = getGrade(s1)  
val f2 = getGrade(s2)  
val f3 = getGrade(s3)
```

```
val average: Future[Int] =  
  for {  
    g1 <- f1  
    g2 <- f2  
    g3 <- f3  
  } yield (g1 + g2 + g3)
```

Exercise 3: Producer-consumer Problem (20 points)

Question 1

The implementation doesn't obey the specification. We should change the first line in `put` and `take` to `while`. This is because when a thread is woken up, the test condition cannot be guaranteed, thus a `while` is needed.

Question 2

```
def snapshot: List[Job] = synchronized {  
  for (i <- front until (front + size) ) yield buffer(i % maxSize)  
}
```

Question 3

```
class MyQueue(maxSize: Int) extends JobQueue(maxSize) {  
  var front = 0  
  var size = 0  
  val buffer = new Array[Job](maxSize)  
  var watchers = 0  
  
  def put(job: Job): Unit = synchronized {  
    while (size == maxSize || watchers > 0) wait()  
  
    // the same  
  }  
  
  def take: Job = synchronized {  
    while (size == 0 || watchers > 0) wait()  
  
    // the same  
  }  
  
  def snapshot: List[Job] = {  
    synchronized {  
      watchers += 1  
    }  
  
    val res = for (i <- front until (front + size) ) yield buffer(i % maxSize)  
  
    synchronized {  
      watchers -= 1  
      if (watchers == 0) notifyAll()  
    }  
  
    res  
  }  
}
```