GPars - Groovy Parallel Systems

Jon Kerridge

Version 1.0, 2015-10-29

Table of Contents

Accessing Shared Resources: CREW

In this chapter and the next, techniques are described that were developed for, and are used most often in, shared memory multi-processing systems. In such systems, great care has to be taken to ensure that processes running on the same processor do not access an area of shared memory in an uncontrolled manner.

Up to now the solutions have simply ignored this problem because all data has been local to, and encapsulated within, a process. One process has communicated data to another as required by the needs of the solution. The process and channel mechanisms have implicitly provided two capabilities, namely `synchronisation between processes` and `mutual exclusion of data areas`. In shared memory environments, the programmer has to be fully aware of both these aspects to ensure that neither is violated.

`Mutual exclusion` ensures that while one process is accessing a piece of shared data, no other process will be allowed access regardless of the interleaving of the processes on the processor. **Synchronisation** ensures that processes gain access to such shared data areas in a manner that enables them to undertake useful work.

The simplest solution to both these problems is to use a pattern named **CREW**, `Concurrent Read Exclusive Write`, which, as its names suggests, allows any number of reader processes to access a piece of shared data at the same time but only one writer to process to access the same piece of data at one time.

The **CREW** mechanism manages this requirement and in sensible implementations also imposes some concept of fairness. If access is by multiple instances of reader and writer processes then one could envisage a situation where the readers could exclude writers and vice versa and this should be ameliorated as far as is possible. The **JCSP** implementation of a **CREW** does exhibit this capability of fairness, as shall be demonstrated.

At the simplest level, the **CREW** has to be able to protect accesses to the shared data and the easiest way of doing this is to surround each access, be it a read or write, with a call to a method that allows the start of an operation and subsequently when the operation is finished to indicate that it has ended. Between such pairs of method calls, the operation of the `CREW` is guaranteed. Thus, the programmer has to surround access to shared data with the required start and end method calls, be they a read or write to the shared data. It is up to the programmer to ensure that all such accesses to the shared data are suitably protected.

In the **JCSP** implementation of **CREW**, we extend an existing storage collection with a **Crew** class. Then we ensure that each **put** access into the collection is surrounded by a `startWrite()` and `endWrite()` pair of method calls on the **Crew**. Similarly, that each **get** access is surrounded by a `startRead()` and `endRead()` method call. Internally, the **Crew** then ensures that access to the shared storage collection is undertaken in accordance with the required behaviour.

Further, fairness can be implemented quite simply by ensuring that, if the shared data is currently being accessed by one or more reader processes, then as soon as a writer process indicates that it wishes to put some data into the shared collection then no further reader processes are permitted to start reading until the write has finished. Similarly, a sequence of write processes, each of which requires exclusive access, will be interposed by reader process accesses as necessary.