Atomic operations and the transputer

Atomic operations are relevant in systems where there is state that can be observed and/or modified by more than one agent. Commonly atomic operations concern the state of a memory system, although the atomicity of operations on other state could also be an issue.

Transputer memory system

The transputer memory system was much simpler than many modern memory systems. The only caching was the 2-word instruction prefetch buffer. There was no mechanism in the transputer to ensure coherency between the prefetch buffer and the memory and it is possible to construct programs which show the breakdown in coherency and consistency. However, apart from this, the memory system was coherent and had strict consistancy for all observers.

The memory accesses were atomic at the word and sub-word levels. All the bits in a word could be written to at the same time, and all the bytes could be individually written to. This atomicity was not true for larger accesses; for example, the action of the processor writing multiple words to memory (e.g. writing four words to save the stack when performing a call instruction) was performed as four separate writes to memory which could be interleaved with memory accesses from the links. If regions of memory were shared between agents (processor and links) it would be possible to observe this non-atomicity. For the most part, by convention, the agents in the transputer did not share memory regions which might be read and written concurrently. The occam rules ensured that this was true for data manipulated by the processor and links; a link channel would be operating as a result of an input or output operation performed by a process and hence the memory accessed by that process would be accessed in accordance with the occam rules. One perhaps subtle point is that occam permits memory to be disjoint at the byte level; that is, a single word could contain bytes “belonging” to different processes. It was therefore necessary that the memory system supported the writing of individual bytes in a word; there would be an atomicity issue to be addressed if part-word writes had to be implemented as a read-modify-write sequence.

So, in summary, as far as program data access were concerned, the occam sharing rules and the sub-word atomicity of the memory system ensured there were no atomicity issues. The transputer scheduler also ensured that all memory accesses associated with processes running in parallel had completed before a subsequent process started to execute. For example, in

```
SEQ
  PAR
    P
    Q
  R
```

all memory accesses for P and Q will have completed before any memory accesses of R start.

There is genuine sharing of the memory system between the processor and the data-transfer engines of the links, however the occam disjoint rules and the memory's atomicity properties ensure that that these operate without any interference.

Atomicity and the scheduler

The remaining atomicity concerns in the transputer relate to the operation of the scheduler and, in particular, to the data structures used by the scheduler. The data structures comprise:- the process control blocks (the first few locations below a process workspace), the scheduler queues, the timer queues, the channels, and the interrupted process save area.

The atomicity of access to these structures is ensured because there is only one entity which performs scheduling operations - the processor. The processor executes instructions (or part instructions in the case of interruptable instructions) and scheduling operations on behalf of the links.

[]