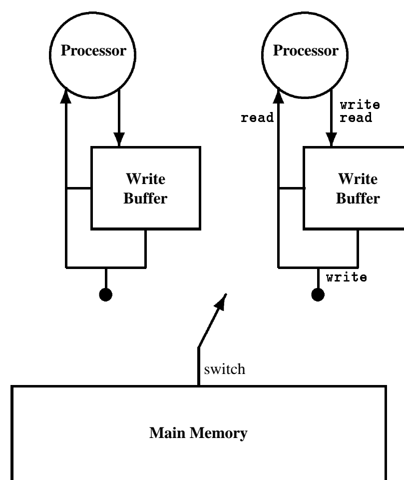
Programming Critical Sections on SPARC

Jalal Kawash

An example SPARC architecture [2, 3] is shown below. There is one write buffer associated with each processor[1]. A buffer is local to its processor. The main memory is single ported with a non-deterministic switch providing one memory access at a time.



Each buffer operates in parallel with the processor. When a processor performs a write, it does not wait for it to be committed to main memory. Instead, the write is sent to the write buffer which will be responsible for committing the pending writes to main memory in a First-In-First-Out (FIFO) order.[2]

When a read is issued by a processor, the associated write buffer is checked for any pending writes to the same location or variable. If there is any such writes, the value "to be written" by the last such write is returned. In this case, the read completes without accessing main memory. The read accesses main memory in the normal manner when there are no pending writes to the same location in the associated buffer.

The buffer mechanism implies that a write by a processor, say P, becomes immediately visible to P, but may take some time to become visible to others. That is, a write will not be visible to other processors until it is committed to main memory.

For this reason, a solution to the critical section problem without hardware synchronization is impossible (a formal theorem is elsewhere [1]). For instance, recall Peterson's Algorithm for two processes (the code for process P_i is given below).

[1]The term processor is used to mean a process as well.

[2]FIFO buffers apply to one version of SPARC; one other buffer arrangement exists.

```
Two Processes

    shared objects
    flag[0 .. 1] in {true, false}, initialized to false
    turn in {0,1}

    <entry>
(1) flag[i] ← true
(2) turn ← j
(3) while (flag[j] and turn = j) do nothing

    <critical section>

    <exit>
    flag[i] ← false
```

Here is a scenario where Mutual Exclusion is violated between P_0 and P_1 on SPARC. The scenario is based on the possibility of both P_0 and P_1 entering their critical sections before the contents of their buffers are emptied.

- In shared memory, `flag[0]` $=$ *false*, `flag[1]` $=$ *false*, and `turn` $= 0$.
- P_0 executes lines `(1)` and `(2)`. These writes are sent to the buffer. P_0's buffer contains "`flag[0]` $\leftarrow$ *true*" and "`turn` $\leftarrow 1$". P_0 executes line `(3)` reading `flag[1]` from shared memory. Since `flag[1]` is *false*, P_0 enters its critical section.

  The values in shared memory are still intact (`flag[0]` $=$ *false*, `flag[1]` $=$ *false*, and `turn` $= 0$.)
- P_1 executes lines `(1)` and `(2)`. These writes are sent to the buffer. P_1's buffer contains "`flag[1]` $\leftarrow$ *true*" and "`turn` $\leftarrow 0$".

  The values in shared memory are still intact (`flag[0]` $=$ *false*, `flag[1]` $=$ *false*, and `turn` $= 0$.)

  P_1 executes line `(3)` reading `flag[0]` from shared memory. Since `flag[0]` is *false*, P_1 enters its critical section.

Hardware syhcnronization is supported in SPARC through the `SwapAtomic()` instruction[3] which can be described as follows:

```
SwapAtomic(x,v):
    begin[atomic]
        t ← x
        x ← v
        return t
    end[atomic]
```

[3] Also refered to as `AtomicLoadStore`.

This instruction is executed atomically, meaning that its execution is performed in a critical section. As the name suggests, this instruction is both a load and a store.[4] Therefore, it is sent to the buffer like any other store. Since loads are blocking a SwapAtomic is also blocking. When buffers are FIFO, a SwapAtomic flushes the write-buffer before the processor is allowed to proceed. The processor proceeds when the SwapAtomic changes the value of x in memory and returns the old value to the processor.

Exercises

1. Modify Peterson's two process algorithm so that it works on SPARC. One SwapAtomic per process is necessary and sufficient.
2. Argue that when buffers are not FIFO the SwapAtomic instruction is no longer sufficient to solve the problem. That is, when the buffers are not guaranteed to be FIFO, more hardware support is required.

References

[1] J. Kawash. *Limitations and Capabilities of Weak Memory Consistency Systems*. Ph.D. dissertation, Department of Computer Science, The University of Calgary, January 2000.

[2] SPARC International, Inc. *The SPARC Architecture Manual version 8*. Prentice-Hall, 1992.

[3] D. L. Weaver and T. Germond, editors. *The SPARC Architecture Manual version 9*. Prentice-Hall, 1994.

[4] A store is a write and a load is a read.