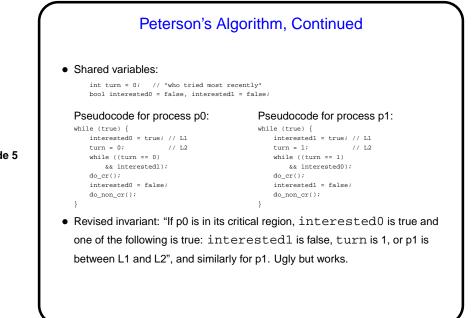
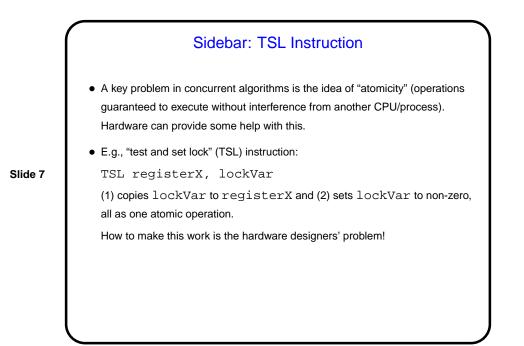


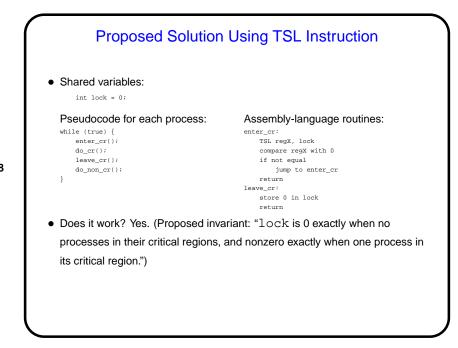
Peterson's Algorithm, Continued
Intuitive idea — p0 can only start do\_cr() if either p1 isn't interested, or p1 is interested but it's p0's turn; turn "breaks ties".
Semi-formal proof using invariants is a bit tricky. Proposed invariant: "If p0 is in its critical region, interested0 is true and either interested1 is false or turn is 1"; similarly for p1.
If we can show this is an invariant, first requirement is met. Others are too.
But a fiddly detail — the invariant can be false if p0 is in its critical region when p1 executes the lines interested1 = true; turn = 1;. See next slide for revision.

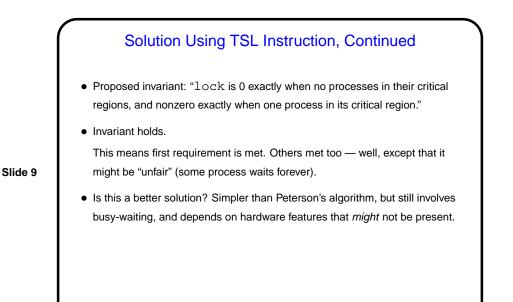


Peterson's Algorithm, Continued
Requires essentially no hardware support (aside from "no two simultaneous writes to memory location X" – pretty much a given). Can be extended to more than two processes.
But complicated and not very efficient.

Slide 5







Mutual Exclusion Solutions So Far
Solutions so far have some problems: inefficient, dependent on whether scheduler/etc. guarantees fairness. (It's worth noting too that for the simple ones needing no special hardware — e.g., Peterson's algorithm — whether they work on real hardware may depend on whether values "written" to memory are actually written right away or cached.)
Also, they're very low-level, so might be hard to use for more complicated problems.
So, people have proposed various "synchronization mechanisms" ... (to be continued).

