Contribution ID: **241** Type: **not specified**

Dependency ordering in the Linux kernel

Thursday, 27 August 2020 07:05 (30 minutes)

For better or worse, the Linux kernel relies heavily on hardware ordering guarantees concerning dependencies between memory access instructions as a way to provide efficient, portable implementations of concurrent algorithms. In spite of the lack of C language support, preserving source-level dependencies through to the generated CPU instructions is achieved through a delicate balance of volatile casts, magic compiler flags and sheer luck.

Wouldn't it be nice if we could do better?

This talk will briefly introduce the problem space (and aim to define some basic terminology to avoid people talking past each other) before opening up to discussion. Some questions to start us off:

- What does Linux currently rely on?
- How can we enforce dependencies at the source level?
- How can we detect broken dependencies and/or insert memory barriers?
- Are annotations a non-starter?
- Does LTO make things worse and why?
- Just how expensive are memory barriers?
- Can we strike a balance between "optimising compiler" and "portable assembler"?

I agree to abide by the anti-harassment policy

I agree

Primary authors: DEACON, Will; DEACON, Will; ZIJLSTRA, Peter (Intel OTC); MCKENNEY, Paul (Facebook)

Presenters: DEACON, Will; ZIJLSTRA, Peter (Intel OTC); MCKENNEY, Paul (Facebook)

Session Classification: LLVM MC

Track Classification: LLVM MC