Reworking threading in GNOME Software

Philip Withnall
Endless
philip@tecnocode.co.uk
History of gnome-software

- Project started in 2012
- Plugins added in 2013
- Architecture has always been entirely plugin based
- AppStream support from 2013
Previous architecture

**Main thread**

**Thread pool threads**

![Diagram](image)

**Figure:** Previous architecture of gnome-software
Issues

1. Limited number of threads in thread pool leads to exhaustion (hangs)
Issues

1. Limited number of threads in thread pool leads to exhaustion (hangs)
2. Large number of threads uses lots of resources (memory)
Issues

1. Limited number of threads in thread pool leads to exhaustion (hangs)
2. Large number of threads uses lots of resources (memory)
3. Locking required everywhere causes serialisation (slow)
Issues

1. Limited number of threads in thread pool leads to exhaustion (hangs)
2. Large number of threads uses lots of resources (memory)
3. Locking required everywhere causes serialisation (slow)
4. Threading overhead is large for simple jobs (slow)
GDBus threading

Figure: Threading when making a GDBus call
Previous architecture

Main thread

Thread pool threads

Job 1 | Job 2

Time

Figure: Previous architecture of gnome-software
New architecture

Main thread

Plugin worker threads

Time

Job 1 | Job 2

appstream | flatpak

Figure: New architecture of gnome-software
Different approaches to threading in C

- Synchronous code always run in a worker thread
- Asynchronous code run somewhere
- Threading model determined at a high level vs locally
Approaches for landing big changesets

- Land early
- Keep things working
- Keep adapter wrappers around old code and drop it eventually
Miscellany

Slide source  https://gitlab.com/pwithnall/
guadae-gnome-software-presentation-2022

gnome-software project  https://gitlab.gnome.org/GNOME/gnome-software

Creative Commons Attribution-ShareAlike 4.0 International License

Beamer theme: https://gitlab.gnome.org/GNOME/presentation-templates/tree/master/GUADEC/2022