LibreOffice On-Line server

Initial implementation details

Tor Lillqvist
Collabora Productivity
@TorLillqvist
www.facebook.com/TorLillqvist

“For Thursday's child is Sunday's clown
For whom none will go mourning”

Goals

- Client: any modern web browser
- Server: Linux, but don't be more platform-dependent than necessary
- Simple protocol over WebSocket
- Layered security to guard against vulnerabilities in LibreOffice or 3rd-party code

Basics

- WebSocket: A simple message-oriented full-duplex protocol over TCP
- Session starts as normal HTTP but switches immediately after handshake to WebSocket

Basics

- LOOL server process(es) isolated from rest of system
- One process per client session
- Isolation between client sessions

Basics

- LOOL server code uses LibreOffice functionality through LibreOfficeKit
- No separate LibreOffice process(es)
- No LibreOffice APIs used directly in the server
- No UNO
- Which is good

Basics

- Tiles sent to client are kept cached for a while
- In case same parts of document viewed later, no LibreOfficeKit instance needed

LibreOfficeKit

- A very simple C & C++ API for LibreOffice
- Exposes the core value of LibreOffice
  - File format filters
  - Tiled rendering (converting documents to pixels)
  - Editing, selections etc
- A very simple ~header-only API – no linking
  - Fully abstract: fn pointers, opaque structs etc
  - No sockets opened, no plugins / simple init
  - Global error messages
- Used also by Android app and loconv

But

- Most of the LibreOfficeKit functionality used is “unstable”
- Whenever new features are added to client-side LOOL, it likely requires bleeding-edge LO on the server
- Not really ideal, but unavoidable

POCO

- Looked for suitable WebSocket implementation
- Found POCO: http://pocoproject.org
- Relatively clean C++ code
  - (Not that I am any connoisseur)
- Lots of utility classes for various commonly needed functionality

POCO

- Availability in popular distros lagging behind by several versions. Looking at you, Debian
- Significant overlap with functionality already in the standard C++11 library
  - Presumably POCO intends to be usable also with older C++ implementations
  - But we require C++11 for LibreOffice anyway
- Obviously, prefer to use std:: and not POCO when possible

POCO

- In addition to WebSocket, for instance also classes for HTTP server and client functionality, easy to use
- Using POCO omehow makes your code look a bit like Java, in a good sense
- In general I have been quite happy with it

LOOL protocol

- Not strict request-response, but asynchronous, full-duplex. Initially planned to be as stateless as possible
- Mostly human-readable and verbose
- First (and usually only) line of WebSocket messages is completely ASCII
- First line can be followed by more (perhaps binary) data

LOOL protocol

- One document open per client session
- New session required to switch to another document
- Tiles returned as PNG-compressed pixmaps

Security

- Layered security
- Chroot jail for each session
- Chroot requires privileges: Use Linux capabilities, not setuid root
- Drop capability immediately when no longer needed
- But anyway, for production, probably want to use some container technology

FIN

“People respected one if one didn’t talk. They believed that one knew a great many things and led a very exciting life.”

git://anongit.freedesktop.org/libreoffice/online

Thanks to IceWarp for funding this work

Technical questions welcome

Collabora

- Collabora Ltd.
  - Leading Open Source Consultancy
  - 10 years of experience. 90+ People.
- Collabora Productivity Ltd.
  - Dedicated to Enterprise LibreOffice
  - Provides Level-3 support (code issues) to all Novell / SUSE LibreOffice clients
  - Architects of Microsoft OpenXML filters