Ircfs and wm/irc

Mechiel Lukkien

mechiel@xs4all.nl

ABSTRACT

Irc, internet relay chat, is a popular chat protocol. *Ircfs*(4) is an irc client that maintains a connection to an irc server, and exports irc functionality through the styx/9p protocol. *Wm/irc* is a Tk program that provides a user interface to the file system interface. *Ircfs* and *wm/irc* have been used from early stages of development with only a few remaining features to be implemented and bugs to be fixed.

Introduction

Ircfs is typically run on a computer with a stable internet connection, with its files exported over styx. Another machine then mounts the file tree and accesses it using *wm/irc*. *Ircfs+wm/irc* was intended to replace the common *screen+irssi* set up. It has already done that for me and turned out to be an improvement: *Ircfs* is a more generic and reusable irc client, with no user interface logic in it. *Wm/irc* is a simple user interface program, not (very) specific to irc. Because the two programs are separate, they have been written and can be debugged and started independently. The user interface is run locally and thus always responsive.

Ircfs and *wm/irc* are both written in Limbo. More information including the code is available on the *ircfs* home page* and in the manual pages *ircfs*(4) and *wm−irc*(1).

This report continues with an overview of *ircfs*, followed by a description of *wm/irc*'s features and concludes with an explanation of design decisions and ideas for improvements.

Ircfs

Let's start with an example:

```
% mount {ircfs freenode} /mnt/irc
% cd /mnt/irc
% echo 'connect net!irc.freenode.net!6667 mjl0' >ctl
% ls -l
d-r-xr-xr-x M 2 ircfs ircfs 0 Sep 28 15:45 0
---w--w--w- M 2 ircfs ircfs 0 Sep 28 15:45 ctl
--r--r--r-- M 2 ircfs ircfs 0 Sep 28 15:45 event
--r--r--r-- M 2 ircfs ircfs 0 Sep 28 15:45 nick
--r--r--r-- M 2 ircfs ircfs 0 Sep 28 15:45 pong
--rw-rw-rw- M 2 ircfs ircfs 0 Sep 28 15:45 raw
% cat nick
mjl0
```

The `ctl` file accepts plain text commands ranging from connecting and disconnecting

* *Ircfs*, `http://www.ueber.net/code/r/ircfs`

to joining channels. `Nick` returns the current nick of the user. Raw allows reading and writing of raw irc commands (not normally used). `Pong` is a file that returns the delay of irc server responses to irc ping commands periodically sent by *ircfs*. *Ircfs* users can use this to detect disruption of the irc connection between *ircfs* and the irc server, and the styx connection between *wm/irc* and *ircfs*. `Event` returns a line for each change of the user's nick, and added and removed channels/users (due to `join` or `part` commands, or queries from other users). Reads on `raw`, `pong` and `event` block until data becomes available.

Each irc channel and user is represented by a directory. These directories directly map to windows in *wm/irc*. Again an example:

```
% echo 'join #inferno' >ctl
% cat 2/name
#inferno

% ls -l 2/
---w--w--w- M 2 ircfs ircfs 0 Sep 28 15:45 2/ctl
--rw-rw-rw- M 2 ircfs ircfs 0 Sep 28 15:45 2/data
--r--r--r-- M 2 ircfs ircfs 0 Sep 28 15:45 2/name
--r--r--r-- M 2 ircfs ircfs 0 Sep 28 15:45 2/users
% echo hi! >2/data
```

The example above shows how to join a channel and say something. Note that this is normally done in *wm/irc*, with the command `/join #inferno`, after which the message `new 2` would be returned on the `event` file, and a new window for directory 2 would be opened by *wm/irc*.

A special directory 0 always exists, e.g. the irc server message of the day. Each directory has a `ctl` file for writing commands (all those accepted by the top-level `ctl` file, plus some only for channels/users), a `data` file for reading and writing text, a `name` file for reading the name of the channel/user, and a `users` file from which changes of user presence can be read, i.e. users joining, leaving or renaming. Reads of the files `data` and `users` block until data becomes available. Lines written to the data file are sent to the channel/user as text. Lines other users write, or meta messages such as users joining/leaving or a change of the topic, are read from the data file with a character prefixed to indicate the type of the line (text, meta information). The `users` file returns lines when users join/leave the channel. This is used by *wm/irc* to provide tab completion for names.

Wm/irc

Wm/irc is a user interface consisting of three parts: On the left a list of names of channels/users, for each of which *wm/irc* maintains a window†. In the middle/right a text area that holds text from the data file, showing the text of the currently selected *window*. At the bottom is a text field for typing text and commands.

Wm/irc can handle multiple *ircfs* file trees, connections to multiple irc servers. Multiple file trees are typically exported using one *styxlisten*(1) and mounted on the machine running *wm/irc*. Note that Inferno's *styxlisten* and *mount*(1) can authenticate and encrypt the connection.

Wm/irc can be started with multiple paths of file trees as arguments. Paths can be added and removed during operation. For each file tree, the status window (special directory 0) is opened for executing commands on, e.g. `connect` to get started. New windows are created for new channels/users, as indicated by the continuously read `event` file.

The term *window* is misleading, it is just a Tk text widget.

Wm/irc stays informed about the user's own name for each irc file tree, and will highlight windows with lines containing the user's name. Additional patterns to highlight can be specified on the command-line. Unread windows with highlighted text have an = before their name in the list on the left, unread non-highlighted text a +, meta messages a − and delayed status windows are marked with a ~. In the text area itself, highlighted text has a yellow background.

Wm/irc has keyboard shortcuts for navigating among windows, e.g. to the next window with a highlight or the previously selected window. Clicking on the name in the list switches to that window.

Text can be copy-pasted with acme-like chording. A plumb button allows selected text to be plumbed. Searching in the text area (reverse by default, from most to least recent) is done from a dedicated text field. Matches are marked by an orange background.

A line typed in the text field is written to that window's `data` file when *return* is pressed, unless it starts with a single slash. A single slash makes the line a command. If the first word of the line (minus slash) is `win`, the remainder is interpreted by *wm/irc*. Otherwise, the line minus slash is written to the window's `ctl` file. For example, `/win quit` causes *wm/irc* to quit while `/quit` causes `quit` to be written to the *ircfs* `ctl` file, causing it to disconnect from the irc server.

Implementation details

Line counts:

```
        1656 ./appl/cmd/ircfs.b
        1446 ./appl/wm/irc.b
         397 ./appl/lib/irc.b
         125 ./module/irc.m
        3624 total
```

The library parses and packs irc messages, converting between strings and adt's representing a message. *Ircfs* uses this library and otherwise just maintains the irc connection and the state of all channels/users, continuously handling irc and styx messages. The functions for handling a styx message, handling an irc message and handling writes to `ctl` files are the largest, followed by the code maintaining the data structures, including history for all channels/users and accounting for blocked styx reads.

Discussion and future work

Ircfs and *wm/irc* are separate programs with distinct functionality, but together provide an easy to use irc client. *Wm/irc* has practically no irc-specific code and the *ircfs* styx file tree is not very irc-specific either. *Wm/irc* could be reused for other instant message protocols, perhaps requiring small modifications to the styx interface. The styx interface has evolved during development of *ircfs* and *wm/irc*, each time adjusting the behaviour of one program to the needs of the other while keeping the code and mechanisms simple.

Ircfs only maintains a single irc connection. To connect to multiple servers, just start multiple *ircfs*'es. Of course, *wm/irc* does support multiple *ircfs*'es, and multiple *ircfs'es* can be exported on a single styx connection.

Most irc clients reconnect to the irc server when disconnected. *Ircfs* does not. First, I haven't needed that feature since the machine I run *ircfs* on has a very stable internet connection (the machine I run *wm/irc* on does not however). Second, it can be tricky to determine whether a disconnect should be followed by a reconnect. Constantly reconnecting is rarely appreciated by server operators. Perhaps a third reason is that such a feature would require quite some code, especially if channels must be joined automatically too.

Channels and users are represented by a directory in *ircfs*' styx interface. The names of those directories are unique numbers, not the names of the channel/user. I cannot recall the original reason for this choice, but there are problems with directories named after the channel/user. First, irc is case insensitive for channel and user names (with quirks for special characters), so there is no unique or even a canonical name for a channel. Second, users can change their name, requiring a change of the directory name, making the path of open file descriptors unusable for e.g. reopens or opens of the `ctl` file given a previously opened `data` file. In short, the semantics of such directories are tricky while the semantics of numbered directories are not.

Not all irc commands have been implemented, e.g. those used by irc server operators. I have not needed those commands, and because documentation of irc commands is often incomplete and/or does not match practice, it makes little sense to write code them.

Wm/irc could be made to start up faster. Many files are opened and the Tk interface is updated a lot while starting. More files could be opened concurrently at start up, and Tk updates batched. However, *ircfs* does not distinguish existing state from new state to users of its files, so *wm/irc* cannot know when the start up phase ends. It would also require quite some code, while recently added more concurrent file opening has improved start up time already.

A larger issue that will require some redesign: *Ircfs* and *wm/irc* have no good way to know whether some lines from `data` files have been read by the user. This sometimes causes messages to be overlooked. I have not yet found a satisfactory solution to this problem. Currently *ircfs* keeps track of which data has been read from a data file. Data that has been read before is returned in multiple lines per styx read request, new data one line at a time. When *wm/irc* receives multiple lines in one read, it knows those lines have been read before and will never highlight these lines. This mechanism is inaccurate for two reasons. First, some irc directories contain only one line and thus will always appear unread and potentially cause highlights. Second, and more serious, any running *wm/irc*, or any other user of the *ircfs*, will cause data to be marked as read. There is currently no direct relation to human interaction with e.g. `wm/irc` and data being marked as read.

Wm/irc opens all windows by default. In same cases, e.g. when on a low-bandwidth connection, it is not desirable to open all windows or real all history. *Wm/irc* has an option to keep all windows closed by default. The *wm/irc* commands to open, close and list (unopened) windows have not been used a lot and might need improvement. *Ircfs* also has a mechanism to limit the amount of history to send: by a *wstat*(2) that only sets the `length` field. This is not a very clean mechanism, and at least too bothersome to tool-users of the `data` file, when history must often be ignored.

Connecting to an irc server is done by the *connect* `ctl` command (or *reconnect* to reuse the last parameters). This styx write for these commands does not return until the connection succeeds or fails. This can take some time since some irc servers stall the connection while verifying it. *Wm/irc* can handle this now (before this was fixed it would block the entire user interface), but other users of *ircfs* might find this behaviour troublesome.

Wm/irc adds colors and underlines to the editable text areas. Editing sometimes causes the mark up to be lost or mangled due to how these are configured with Tk. I doubt this problem has an elegant solution.

Wm/irc allows acme-like chording (in both *text*(9) area and *entry*(9) fields, each requiring different code), and plumbing by clicking a button. It is a pity this has to be implemented by *wm/irc*. It would be nice if this code would be provided as generic Tk functionality.