Webterm — An HTML5 Desktop Environment

Eli Cohen
echoline@gmail.com

ABSTRACT

Webterm is an HTML5 remote desktop for use with Plan 9[1] which re-imagines drawterm[2] inside of a browser. The sourcecode is available at `https://github.com/echoline/webterm`

1. Introduction

Webterm is an HTML5 remote desktop. Like drawterm, it is a remote desktop login to a Plan 9 CPU server. It is mostly backwards-compatible with the `draw` device for graphics, so it can run most existing graphical Plan 9 programs. It also has a `dom` device for HTML5 programs.

2. Drawterm

Drawterm is difficult to explain without some experience with Plan 9. It's basically a lot of the functionality of a Plan 9 kernel implemented in portable C for non-Plan 9 systems. It's a remote desktop program, but it's also more. On Plan 9, the GUI is an integral part of the system, unlike on most Unix-like operating systems. Graphics are handled through the `draw` device, usually expected at /dev/draw. Typically the kernel sets up a screen framebuffer. After that, connecting to the draw device allows programs to draw graphics primitives to the framebuffer instead of accessing the screen directly. For example, there is a specific message to the draw device for drawing an ellipse. Drawterm is a draw device and a few other components of a Plan 9 terminal used for logging into a Plan 9 CPU server from a different system. Drawterm differs from a Plan 9 terminal in that it cannot directly run Plan 9 programs. All of the programs run on the CPU server. For people familiar with SSH, drawterm does everything SSH does for a system where graphics, a mouse, and networking were always part of the design.

3. Webterm

Webterm is a spinoff of drawterm for browsers. Webterm was an exploration in synthesizing Plan 9 ideas with a browser. Webterm cannot do everything drawterm does for security reasons. For example, drawterm exposes the shell and storage of the host it runs on, and obviously this cannot (and should not) be done from a webpage. Overall, webterm can do much less than drawterm from a technical viewpoint, but it was easy to implement a filesystem around the Javascript Document Object Model[3], available at /dev/dom, and utilize features built into the browser. For example, alpha transparency was already available as simple CSS. Backwards compatibility with /dev/draw was harder. Webterm follows aiju's way of doing things with her jsdrawterm[4] and compiled the draw code to Javascript with emscripten.

4. Login

The way logins are handled on Plan 9 is a simple and interesting method. On Plan 9, not only is everything a file, the 9P[5] filesystem protocol is a simple bytestream. On Plan9front, the authentication is done, then after logging in, the client sends a few shell commands. This script sets things up on the server to mount the network connection at `/mnt/term`, with the client exporting a 9P namespace over the connection. In this way the filesystem of the client, in this case webterm, becomes available in the namespace of the login to the server at `/mnt/term`. The code for all of this and the corresponding TLS encryption was entirely rewritten in Javascript, though not particularly well, which was a very enlightening exercise in learning about Plan 9's minimal, well-designed, and unique login process. Plan9front introduced the `dp9ik` authentication protocol and associated programs for logging into CPU servers. Before dp9ik, Plan 9-to-Plan 9 CPU and other-to-Plan 9 drawterm connections were authenticated with the older DES-based `p9sk1` protocol.

5. 9P

9P is a bytestream filesystem protocol. It can be transported over basically any link, even inside another 9P file. It's a simple protocol with messages for filesystem operations. A file in 9P is a generalized system object that can be opened, read, written, closed, etc. A file isn't necessarily data on a disk; it can represent other things. For example, writing raw PCM samples to /dev/audio causes them to come out of the speakers. This is part of what makes Unix elegant, and Plan 9 extends that concept to its natural conclusion.

Plan 9 is simply a 9P multiplexer. The kernel has some of the hardware drivers, and the system has some Unix-like programs and (sometimes) the ability to run them. It's a minimal but powerful system where components can be pieced together easily. After that, it gets a little foreign from a Unix viewpoint. Drawterm really is a Plan 9 kernel, but can't run programs. In some alternate universe where PC's never took off, a draw terminal is the logical completion of the idea of a dumb terminal. It just logs in and shares a screen, keyboard, mouse, and maybe some storage to a server over almost any link. 9P is the filesystem protocol which allows everything, not only to be a file, but to be transported easily across any medium. In this case the browser shares a Javascript 9P filesystem to the CPU which is transported over a websocket.

6. DOM Filesystem

Before completing devdraw backwards-compatibility support as mentioned above, I implemented a filesystem for the Javascript Document Object Model. Most people who know web programming are familiar with this. XML documents are a tree, like a filesystem. Exposing the XML document of the webterm page through 9P was extremely easy. There is a /dev/dom directory, which is the top-level tag for the desktop div. The tags nest naturally, mapping very well to a hierarchical filesystem view. Writing to the files in /dev/dom can change the innerHTML, value, or attributes of tags. Some simple programs such as hclock utilize the dom device to render HTML to windows.

7. Kludges for the Future

Plan 9 is in a unique position to exemplify good software engineering, even for modern applications. The Plan 9 community knows very well the history of grid computing and terminals, an almost-forgotten precursor to cloud computing and browsers. The browser was never really meant for any of the things we see today, and a real desktop inside of a modern browser is simply reaching too far.

More than ever, the community viewpoints of the many drawbacks of browsers were made clear to me. First of all, nothing in web browsers was ever standardized at all. The first step was an innerHTML file to "draw" html to the window, and soon it became apparent that textarea and input tags don't work that way. They were never standardized. There are probably all sorts of other edge-cases which do not work, but for which it would be very easy to add functionality.

Over long distances it doesn't work as quickly as drawterm through the same connection. Not only does it have the websocket overhead, TLS had to be implemented again entirely in Javascript running within the browser. The 9P files local to the browser are all written in Javascript as well. It ends up very slow. One idea that could speed up browser technology in general is "bttp://," Bittorrent Transfer Protocol. There are many details to work out for this to happen, but if it were eventually done, it would mean that the more users online at the same time, the faster it would be. Instead of getting slower under load, a distributed web protocol would get faster.

Cross-site scripting attacks are a huge concern with webterm. For a desktop in a browser, what can you do if you want to include things from other sites? This could be handled better with something like an <sframe/> tag, a secure iframe which is basically all the same functionality as a separate tab. It would be a container as much as possible, the same as a tab. It would ideally have absolutely no interaction with the page it was on, aside from being an opaque HTML element on it.

A final idea that came out of doing this project is draw support for common image formats. Draw supports the Plan 9 image format, both raw and compressed, but it could also directly handle JPEGs, PNGs, etc.

8. Conclusion

This project gave fascinating insights into what makes Plan 9 such a refined system, especially compared with the modern web. Making webterm also led to ideas for improving browser technology. It was an illuminating exercise in discovering more about what makes Plan 9 exceptional.

9. References

1. https://en.wikipedia.org/wiki/Plan_9_from_Bell_Labs
2. https://man.cat-v.org/plan_9/8/drawterm
3. https://www.w3schools.com/js/js_htmldom.asp
4. https://github.com/aiju/jsdrawterm
5. https://9p.cat-v.org