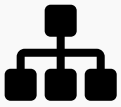
UNIX IN YOUR BROWSER TAB

Run C, C++, Go and Node.js programs as **processes** in browsers, including LaTeX, GNU Make, Go HTTP servers, and POSIX shell scripts.

ABOUT

Programs written to run on conventional operating systems typically depend on OS abstractions like processes, pipes, signals, sockets, and a shared file system. Compiling programs into JavaScript, asm.js (http://asmjs.org/), or WebAssembly (http://webassembly.org/) with tools like Emscripten (https://kripken.github.io/emscripten-site/) or GopherJS (https://github.com/gopherjs/gopherjs) isn't enough to successfully run many programs client-side, as browsers present a non-traditional runtime environment that lacks OS functionality. Porting these applications to the web currently requires extensive rewriting or paying to host significant portions of code in the cloud.

Browsix is our answer to these challenges, featuring:



PROCESSES

Unmodified C, C++, Go, and Node.js programs run as processes on Web Workers, executing in-parallel with the main browser thread – no need to worry about long-running computations blocking event-handling or page rendering.

KERNEL + SYSTEM CALLS

By working at the lowest levels of abstraction, Browsix provides shared resources to multiple language runtimes, just as traditional operating systems enable running programs written in a host of languages.

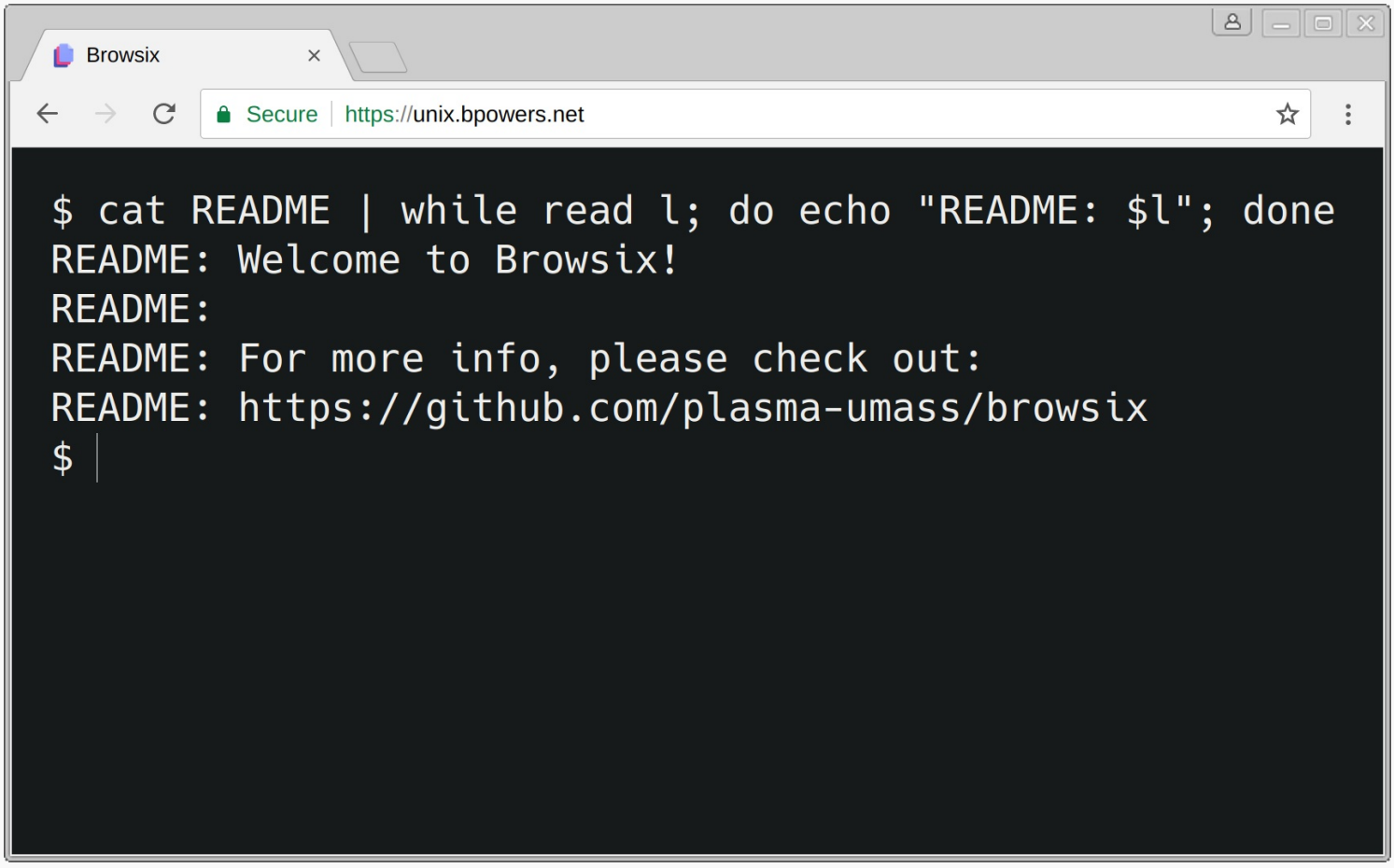
SCALABILITY

By enabling a large class of programs (including legacy codebases) to run in-browser, Browsix can free you from the chore of sandboxing and load-balancing programs server-side.

EXAMPLES

TERMINAL (HTTPS://UNIX.BPOWERS.NET/)

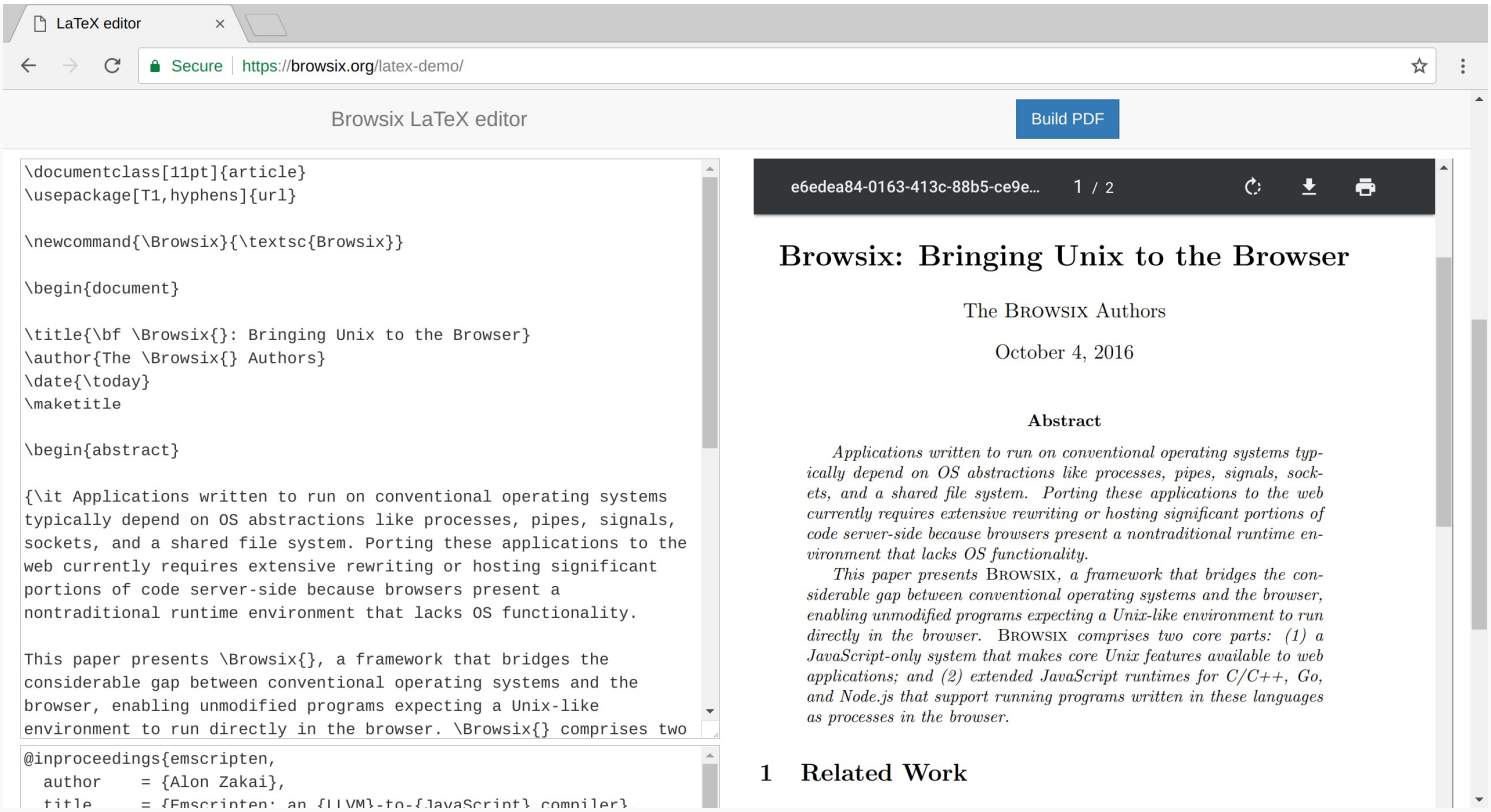
A Unix terminal exposing the dash POSIX shell lets developers compose functionality and inspect Browsix state in a familiar way. (view source) (https://github.com/plasma-umass/browsix/tree/master/app/elements/browsix-terminal)



(https://unix.bpowers.net/)

LATEX EDITOR (HTTPS://BROWSIX.ORG/LATEX-DEMO/)

In-browser editor that runs pdflatex and bibtex to generate PDFs. Required < 150 LoC to orchestrate these applications. (view source) (https://github.com/plasma-umass/browsix/tree/master/examples/latex-editor)



(https://browsix.org/latex-demo/)

MEME GENERATOR (HTTPS://MEME.BPOWERS.NET/)

Client/server web application written in JavaScript and Go. The Go server blits text over images using off-the-shelf libraries and runs unmodified under Browsix. 30 LoC policy in client chooses to route requests to cloud or to in-browser server process. (view source) (https://github.com/plasma-umass/browsix/tree/master/examples/meme-service)

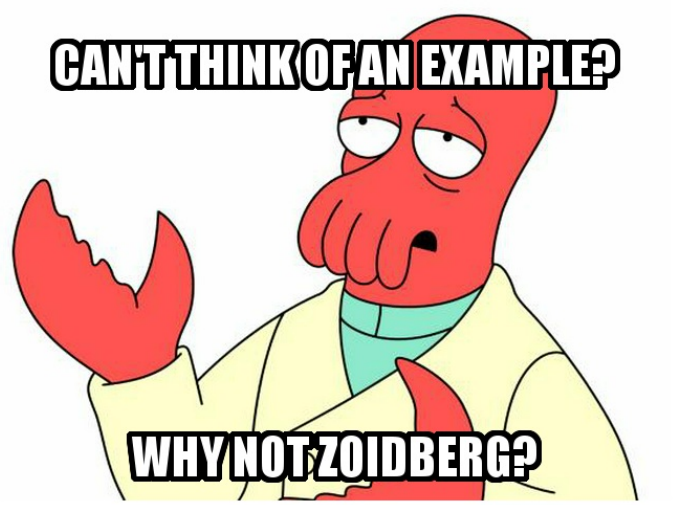
Background Image:

zoidberg

top | Can't think of an example?

bottom | Why not Zoidberg?

Create



(https://meme.bpowers.net/)

HOW IT WORKS

Browsix is a framework that bridges the considerable gap between conventional operating systems and the browser, enabling unmodified programs expecting a Unix-like environment to run directly in the browser. Browsix does this by mapping low-level Unix primitives, like processes and system calls, onto existing browser APIs, like Web Workers (https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers) and postMessage (https://developer.mozilla.org/en-US/docs/Web/API/Worker/postMessage).

Browsix brings all of these abstractions into unmodified browsers, and is isolated and secured to the same extent any normal web page is: at the level of the browser tab.

To use Browsix, client-side JavaScript code creates an instance of a Browsix kernel (which involves telling it how to initialize the filesystem), and then asks the kernel to start or kill processes. Users can also perform HTTP requests to a given Browsix TCP port, and register callbacks a number of events, like when a process has written to standard out or standard error, for when processes exit, and for when ports are ready.

OS ABSTRACTIONS

- **Processes** are built on top of Web Workers, letting applications run in parallel and spawn subprocesses. System calls include `fork`, `spawn`, `exec`, and `wait`.
- **Signals** with `kill(2)` and signal handlers.
- **Shared Filesystem** accessible from multiple processes.
- **Pipes** are supported with `pipe(2)` enabling developers to compose processes into pipelines.
- **Sockets** include support for TCP socket servers and clients, making it possible to run applications like databases and HTTP servers together with their clients in the browser.

INIT

Thanks to a successful Summer of Code project (https://summerofcode.withgoogle.com/archive/2016/projects/5164267334533120/), Browsix has an init system (https://github.com/plasma-umass/systemgo) that supports managing and supervising in-browser services specified by systemd (https://www.freedesktop.org/wiki/Software/systemd/) unit files (https://coreos.com/docs/launching-containers/launching/getting-started-with-systemd/).

PAPER

Browsix is described in a peer-reviewed paper accepted at the upcoming ASPLOS 2017 (http://novel.ict.ac.cn/ASPLOS2017/) conference.

BROWSIX: Bridging the Gap Between Unix and the Browser

Bobby Powers, John Vilk, Emery D. Berger
University of Massachusetts Amherst

Abstract

Applications written to run on conventional operating systems typically depend on OS abstractions like processes, pipes, signals, sockets, and a shared file system. Porting these applications to the web currently requires extensive rewriting or hosting significant portions of code server-side because browsers present a nontraditional runtime environment that lacks OS functionality.

This paper presents BROWSIX*, a framework that bridges the considerable gap between conventional operating systems and the browser, enabling unmodified programs expecting a Unix-like environment to run directly in the browser.* BROWSIX *comprises two core parts: (1) a JavaScript-only system that makes core Unix features (including pipes, concurrent processes, signals, sockets, and a shared file system) available to web applications; and (2) extended JavaScript runtimes for C, C++, Go, and Node.js that support running programs written in these languages as processes in the browser.* BROWSIX *supports running a POSIX shell, making it straightforward to connect applications together via pipes.*

We illustrate BROWSIX*'s capabilities via case studies that demonstrate how it eases porting legacy applications to the browser and enables new functionality. We demonstrate a* BROWSIX*-enabled LaTeX editor that operates by executing unmodified versions of pdfLaTeX and BibTeX. This browser-only LaTeX editor can render documents in seconds, making it fast enough to be practical. We further demonstrate how* BROWSIX *lets us port a client-server application to run entirely in the browser for disconnected operation. Creating these applications required less than 50 lines of glue code and no code modifications, demonstrating how easily* BROWSIX *can be used to build sophisticated web applications from existing parts without modification.*

1. Introduction

Web browsers make it straightforward to build user interfaces, but they can be difficult to use as a platform to build sophisticated applications. Code must generally be written from scratch or heavily modified; compiling existing code or libraries to JavaScript is not sufficient because these applications depend on standard OS abstractions like processes and a shared file system, which browsers do not support. Many web applications are thus divided between a browser front-end and a server backend. The server runs on a traditional operating system, where the application can take advantage of familiar OS abstractions and run a wide variety of off-the-shelf libraries and programs.

As a representative example, websites like ShareLaTeX[1] and Overleaf[2] let users write and edit LaTeX documents in the browser without the need to install a TeX distribution locally. This workflow lowers the barrier for students and first-time LaTeX authors and enables real-time collaboration, eliminating some of the complexity of creating multi-author documents. These applications achieve this functionality by providing a browser-based frontend for editing; user input is sent to the server for persistence and collaboration purposes. When the user requests a generated PDF, the website runs `pdflatex` and `bibtex` server-side on the user's behalf, with the resulting PDF sent to the browser when complete.

These web applications generate PDFs server-side out of necessity because browsers lack the operating system services and execution environment that Unix programs expect. Creating PDFs from LaTeX requires spawning multiple processes to run `pdflatex` and `bibtex`, which need to read from and write to a shared file system. If PDF generation takes too long and the user cancels the request, the application needs to send a `SIGTERM` or `SIGKILL` signal to clean up any running processes. If PDF generation encounters an error, the application needs to pipe the output of the relevant process back to the client over the network. Since browsers do not support processes, signals, pipes, sockets, or a shared filesystem, they cannot perform any of these steps without program modification.

Previous attempts to cope with this impedance mismatch between conventional applications and the browser fall short of providing the environment needed by many programs (see Section 7). Emscripten and Doppio provide a POSIX-like runtime system for single processes, including a single-process file system, limited support for threads, synchronous I/O, and proxying support for TCP/IP sockets [9, 12]. While these single-process runtimes are useful for some applications, they are severely limited because they are unable to provide the range of operating system functionality that many legacy applications demand.

To overcome these limitations, we introduce BROWSIX, a framework that brings Unix abstractions to the browser through a shared kernel and common system-call conventions, bridging the gap between conventional operating systems and the browser. BROWSIX consists of two core components: (1) a JavaScript-only operating system that exposes a wide array of OS services that applications expect (including pipes, concurrent processes, signals, sockets, and a shared file system); and (2) extended JavaScript runtimes for C, C++, Go, and Node.js

[1] https://www.sharelatex.com/
[2] https://www.overleaf.com/

(https://arxiv.org/abs/1611.07862)

Check out a pre-print of the paper on the arxiv here (https://arxiv.org/abs/1611.07862).

DOWNLOAD

Browsix comprises two core parts:

1. A kernel (https://github.com/plasma-umass/browsix) written in TypeScript that makes core Unix features (including pipes, concurrent processes, signals, sockets, and a shared file system (https://github.com/jvilk/BrowserFS)) available to web applications.
2. Extended JavaScript runtimes for C, C++ (https://github.com/bpowers/emscripten), Go (https://github.com/bpowers/browsix-gopherjs), and Node.js (https://github.com/plasma-umass/browsix/tree/master/src/browser-node) that support running programs written in these languages as processes in the browser.

We will soon have more push-button instructions for integrating Browsix into your project. For now, check out GitHub (https://github.com/plasma-umass/browsix) for more details.

CREDITS

(https://plasma.cs.umass.edu/)

Browsix is a research project from the PLASMA lab (https://plasma.cs.umass.edu/) at the University of Massachusetts, Amherst.

Browsix is primarily the work of Bobby Powers (https://bpowers.net/) with significant contributions from John Vilk (https://jvilk.com/) and with guidance from their advisor, Emery Berger (https://emeryberger.com/). Browsix is open source, and a number of folks (https://github.com/plasma-umass/browsix/graphs/contributors) have made valuable contributions.