

Google OS (aka Chrome)

Eric Blair

3 September 2008

[OK, Ms ABR of Washington, Columbia asked me to write about Google's new browser, so here goes. I'm typing fast, editing lightly, and posting on an odd-numbered day.]

Google's browser is an attempt to shift the position of a long-running search for balance, over where work is to be done. So this discussion of the browser has to start with a brief history of networked computing.

We begin with your mainframes of old, like before we were born old, which often had terminals attached. Terminals, like the terminus of a railroad station, were the end of a line out of the central system, where the end in this case has a screen and a keyboard attached. You would send requests from your little end of the line, they would go to the mainframe, and then it would send results back down the line. Thus, these terminals were called dummy terminals, because they did no thinking, just relaying keyboard presses and displaying the output.

This is why the personal computer revolution was so interesting: you now had terminals that looked like dummy terminals (like the TRS-80) that were capable of doing things on their end of the line. So home users, who had no mainframe to attach to, were increasingly using these little terminals to do independent work that the dummies could never do.

Now, put a mainframe capable of math on one end, and a terminal capable of doing math on the other. The key question for the rest of the essay: who does the processing?

To make this more concrete, jump forward to the Internet age. You type in a web address, and the server sends back a big block of text. That's dummy terminal mode, where your computer is doing minimal thinking. Now say you go to a site with silly Flash or Java games. You go to the site, you get a bar that says 'loading' on the screen for a minute, and then you play your game on your screen, without really talking to the server. Now things are reversed: the server just read your request and dumped back data, and your PC does all the work.

Or say you go to Gmail. It has a 'loading' bar like a Flash game. But the server is active, because it's trying to find your new mail, starred mail, spam count, and so on. But your PC is active, because it's opening and closing window bits without talking to the server, autocompleting and highlighting things when your mouse is in just the right place, and so on. There's a sweet spot between work on the server side and work on the client side; a lot of people think Google has hit it. [No citations today. But try typing 'Google sweet spot' into, uh, a search engine.] Me, I think Google has missed it: my email should not need a 'loading' bar, but that's just opinion.

Virtual machines Why not have the client do everything? That's the clear trend, but it's been tried before, and past efforts were not as victorious as hoped. Recall Java, which emerged with much hype the mid-1990s as the way to get networked computing onto our increasingly smart client PCs. In retrospect, we can see Java's failings pretty clearly. First and of least importance, it emerged in the middle of the object-oriented fad of computing, and the language itself went way overboard.

Second, it relied on a virtual machine (VM) that never ran as well as we'd have hoped. Java promised to write a VM for any device (telephone, Windows box, Linux box) that would handle the guts and details, and then you'd write a program in one language—Java—that runs on all these machines. But the VMs were all a little different: at the least, your telephone has buttons that your PC doesn't and vice versa, so how do you write something that works in both places? But the big virtual machine difference was between Sun's virtual machine and Microsoft's. Microsoft's Java machine was designed to be incompatible, as recorded in a ton of court documents. You'll recall the press about the Microsoft antitrust case, which was mostly about Microsoft killing the Netscape browser, but the real crux of the case was about how the browser carried a Java VM, and Microsoft felt it important to kill the VM.

So once you download a Java program, it might not run. Running from a virtual machine instead of native to the hardware, it might run slowly. And finally, there was the downloading issue: a Java program is too much for a guy living in 1995 with spotty AOL dialup to use without frustration.

But the virtual machine idea was a good one. It's a fabulously attractive idea to have a code-running box that manages all the low-level work, so programmers can do the high-level stuff. It's so fabulous that Microsoft does it: their .NET framework basically allows you to write in any language, then translate it to their .NET machinery to run on a Windows box. This is exactly the abstraction Java did, but .NET is written around Windows machines. [The virtual machine idea predates Java. Infocom games, like Zork or the Hitchhiker's Guide to the Galaxy, were data files for a virtual machine. The Infocom VM is easy to rewrite; I could get one for my telephone.]

Your browser is a virtual machine. Every browser can read JavaScript (whose code has no discernable relationship to Java—the naming similarity is pure advertising), and can run Flash, and load Java programs. That's why Google's mail program can run on basically any machine as long as you have a browser to interpret its Javascript.

The family tree One of my favorite things about how modern computers work is the fork/exec model. I won't bother with details, but programs can start other programs. Every process has a parent process (unless the parent died, in which case it's an orphan), and no program can spawn out of nowhere: it needs a parent. This is how the entire thing works, from boot to shutdown: you start with `init`, then `init` forks off a new program, say the bash shell. Then the bash shell forks off a browser when you type `firefox` at the command prompt. Then you open a lot of tabs in Firefox.

The process model gives you stability, because the children are only vaguely related to their parents (mostly via carefully-controlled interprocess communication), and if the parent has issues, then they won't affect the child, and vice versa. It's the operating system's job to make sure that this is the case, and to make sure that the processor gives

fair time to every process running, where by ‘fair time’ I mean access to the hard drive, the processor, and other physical resources the OS is taking care of.

So back to Firefox, which does not spawn child processes (to speak of). It’s one monolithic blob to the operating system, not a family, so, e.g., if one blob of Javascript fails in one place, then all the others will also be stuck.

Google Chrome is prolific: it is designed to spawn lots of children. For every web page you have open, you should have a separate process. So let’s review: you have a Javascript program (aka a web page) in one tab, and that tab is its own process that the operating system treats equally to every other program. Yup, sounds like a standalone virtual machine to me, exactly like the Java VM or Microsoft’s .NET.

So Google has taken those last steps to make our typical programming languages of the Web exactly the languages you need to write standalone programs for any operating system. With a few lines of Javascript and HTML, you can write and distribute a standalone Windows program.

Or to put it more directly: the operating system now gives equal treatment to Google Docs and Microsoft Office.

Critique and politics The Google VM will definitely benefit Google: they’ve got the lead in programmers who speak the language that their VM speaks. Does that make their browser evil? Maybe, but as evil goes, this is pretty beneficial to everybody (except Microsoft), because another VM choice may allow some fun new applications.

In fact, Google has made their code available under a relatively corporate-friendly member of the family of free software licenses (BSD). Why? Because they don’t care about vending VMs, they want to make sure that absolutely everybody has such a VM, so that it’s feasible to write for the Google VM rather than for .NET or whatever other toolkits might be hanging around. How getting people to choose Javascript over .NET will turn into \$\$\$ for Google is left as an exercise for the reader.

Oh, here’s one hint (along one of several threads): go back to the problem of balancing work on the client and server ends of the cable. If Google gives you software that grabs more processor time on your PC for Google Docs, then it can redesign things so that its servers in California don’t have to think so much. Google doesn’t have to spend cash on new servers—they just use more processor time on your PC. Google is thinking maybe you can pay the darn electricity bill for once.

Further, mainframes are not particularly smart. From my own experience buying servers for research, the big boxes are designed to push lots of data through a pipe, hold a big database, mesh together into an army of servers, and otherwise handle lots of little requests. But the processor on some servers is identical to the processor on a high-end PC, and ten cheap PCs would easily run circles around one blade of a server. So the only way that Google could feasibly make a million instances of Docs smarter is to push work out to the clients.

As a digression, all this processor-seeking touches upon one of my personal pet peeves: VMs are slow. As I type, I’m waiting for Amarok to add an album to my playlist. This is not something that should require waiting for (op—it’s done), but Amarok is written in Ruby, which allowed for all sorts of nifty widgets that would take longer to write from scratch. Hey, just click a performer and pull up their Wikipedia

page in your music player, all while you're waiting for their music to actually play. So I'm not sure if we can expect *too* much richness from Google's new virtual machine, though maybe for once the promises that it'll be better with next year's faster processors will actually come true.

But that's all the critique I've got. Google has taken that last step to turn the web pages of the type in which they specialize into bona fide applications that the operating system treats as such. That's nifty, and means that we can expect our web pages to turn increasingly complex and to increasingly take advantage of the processing power on our end of the cable.