What’s Up With Processes

With Special Guest Darin

Today, we're talking about processes in Chrome with special guest Darin. He was one of the founding members of the Chrome team, and wrote the first implementation of the multi-process architecture.

Process
Process is the container in which applications run on your system. Every process has both its own executing set of threads, also its own memory space. That way, processes have their own independent memory, their own independent data, and their own independent execution. The system is multitasking across all of the processes on the system.

Chrome process vs. OS process
Chrome is made up of multiple processes. That's done for multiple reasons. One is so that they can run independently, and take advantage of multi-core systems so that if you have more computing power, then it's possible for more things to happen concurrently. When we talk about a process in Chrome, we mean an OS process.

Multi-process architecture vs. process model
Multi-process architecture is the embodiment of the whole chrome. And we have these different process types that make up that whole thing. There's the browser process(main one, hosts the whole UI of the app. child processes), a renderer process (runs Blink), a networking process, a GPU process, utility process, in the lifespan of Chrome, other types of processes, such as plugin processes. When we were hosting flash in Chrome. And the native client had its own type of processes as well.

Service worker
Extensions can inject stuff to pages to modify them. And they achieve this by maintaining an invisible page in the background that consumes resources and memory. So we created this concept called event pages, which allowed for these background pages to be a little more transient, that come into being only as needed. At the same time, Service Worker had been created, which was a tool for web pages to be able to do background event processing. Service workers are not web pages. They're just JavaScript. But they can listen to different kinds of events. They have some functions that are given to them on the global scope that lets them talk to the outside world, to the web page that created them. They actually have events they can receive to handle

network requests on behalf of the page. That's one of the main uses for them in the context of the web.

Big changes to the multi-process architecture since launch
The biggest one by far is the per-site isolation, other big changes were the introduction of the GPU process, and the introduction of modular IPC.

What would change if we were to redesign chrome?

- Introduce site isolation earlier. We had no much ownership of Webkit when we first started.
- Introduce Mojo to allow a much more flexible system.
- Build a cross platform product from day 1, instead of only building a browser for Windows XP Service Pack 2 (Chrome was very focused on being a product first, not a browser construction kit)
- Views to allow cross-platform UI

IO and UI threads
The design of the system was there was a UI thread, and that's where all the UI lives. And the IO thread was meant to be a highly responsive thread for processing asynchronous IO, and non-blocking. Because the name IO thread subsequently confused lots of people who wanted to do blocking IO on that thread, we invented something called the file thread; this is the thread where you read files. They're all running in the browser process.

On what process a certain block of code runs?
Go to the content directory. You'll see a browser directory, subdirectory, a renderer subdirectory, and a common directory. And there's some other ones that have these familiar names. We use that structure all throughout the code base for different components. So if you go components, components Foo, you'd see browser, renderer, common, maybe a subset of those. Code that only runs in the renderer lives in the render directory, that only runs in the browser lives in the browser directory. If it's code that could run in either, it lives in the common directory. So you see mojom definitions in common directories because mojom is where you define the Mojo interface that's going to be used in both processes.

Process limits
Process limits exist to have a reasonable number of processes allocated. Early on, it was based on how much RAM you had on your system. More processes use more RAM. There was research done into freezing tabs, i.e. suspending them and not letting them

do any work. But that comes with challenges of what do you do with all the IPCs that are inbound to those processes? If you unfreeze them, now there's a blast of IPCs coming in that they suddenly have to service. You either cannot drop those IPCs on the floor which might lead to weird states. It's not right to keep all the tabs open. We should find a balanced state by being judicious about what we keep open.