Systemd and resource management in GNOME

Benjamin Berg

>> So, the next talk is titled Systemd and resource management in GNOME by Benjamin Berg. Are you here, Benjamin?

[Working on audio issues for the speaker!]

[Still standing by -- looking forward to the talk!]

[Captioner trying to reconnect-it's not liking me at the moment...] [Got kicked out, but I'm back in now and see everyone else coming in too! Hi.]

>> All right. It's working now. Okay. I still need permissions. Sorry, I'm just waiting to be able to upload the slides, in case anyone is wondering. There we go.

Presentation. All right. How does this work? So, I was going to talk a bit about the resource management stuff and we can do it systemd now and the basis of this technology.

As what we can do with this. So, in general, the problem that we have is that we have limited resources on the CPU, laptop. So, we have limited CPU time, limited memory which needs to be distributed across caches, data. We need to also do IO scheduling and ensure that everything receives its fair share of resources. And historically, what we have been doing is that we have distributed those between processes. So, if you run a lot of processes, then they can kill the system. And if you fork -- like if you want run a makeJ or something like that. And you have some possibilities like siting a nice value or ulimits. But it's generally restricted to always managing a single process.

So, we end up treating most processes as equals. However, that means that, like -- not just 8 jobs, 16 jobs, however many in the course you have, might drown out other processes. Or the browser uses separate processes. But then renders one page in one of these processes. And then you have make running taking up the rest of the system and the browser doesn't get all the resources it's requesting. Even with your make job, your compile job, is already taking more resources than the browser itself. But when it's running on one core only and other one is running on multiple cores.

And then we ended up with an unfair distribution of resources between these kind of applications. Which is obviously not something that is desirable to us. So, in general, what we would like to have here? We would like to treat users equally. So, if I have multiple users logged in, it would make sense to allocate the same amount of resources to each user. So, if one user runs and compiles with lots of jobs and the other is just doing a single process task. That single process task should run at full speed while the compile is down as it's running on multiple cores and actually gets a higher share of resources overall.

Similar, between applications. The example from before. And in general, we also want to keep the desktop responsive. So, if gnome-shell needs resources or accesses or CPU time, it's probably a good idea to give it a higher share of the overall resources than the applications are getting. Which then also once we do that, and once we are actually able to treat them equally, we can also go one step further and disrupt them. So, we will pick out and gnome-shell should be more responsive, right? So, we will give it an advantage overall. Give the actor user a better, higher share, or we could also make a difference whether an application is focused or not.

So, David Edmundson actually created the demo for that. I just wanted to share a video. Wait a minute. Entire screen. Share. Ah.

So, we can see here is that two processes are going to be structured. And each one is running a spinner. And the speed at which the spinner is updated depends on how much CPU resources that program gets. And you'll notice that the currently focused application is running very nicely and smoothly. While the other application is running really laggy and lagging behind. So, the one with the dark bar at the top is basically smooth. And the one with the white bar at the top is running slowly. And this is basically a demonstration of what you could do if you discriminate between -- based on whether an application is focused or not.

Right. So, how can we do this? And the solution for this is on a low level and in Linux, are cgroups. So, what cgroups basically -- what cgroups, what you do is order the whole system as the system is running it in a hierarchical structure. And the systemd will pilled build this away and systemd does a few things special in the sense that processes are only going to be running in all the lib notes and in between notes are not going to run any processes. But that's not requiring a limitation. And then, based on top of this hierarchy that we have, we can then schedule on -- sorry. There are controls available. We can enable CPU, IO or memory controllers and there are a number of other controllers that I'm not going into because they're not as relevant for us.

One interesting command might be systemd-cgls. It's system group ls. And this is the hierarchy of the system. This is my system running a few days ago. I have dash dot slice, the root. And the users and all the processes in there. And user 1000. That's in there. And this is the systemd user instance which runs the -- my session. Because my session is launched using systemd. And then I have reordered my sessions so that all the really important processes are in session.slice. And then you see that gsd is here, and GNOME-shell is here. And gnome-shell, it launches ibus, for example, right? And XWayland. So, because gnome-shell launches these, they are grouped together as one and you can consider them as one entity.

So, then we have the CPU controller. And what we get there is that we can read out certain information from the cgroup system. We can look at the CPU statistics per slice, per cgroup. We can look at the pressure information. So, how often the processes are waiting on actually receiving CPU time and are unable to run because of that. We can change the CPU weight so we can change the relevance, how relevant one cgroup is compared to another sibling. This is different from before, one of the spinners was running faster than the other. What we did there was change the CPU weight to be really high for one of them and very low for the other. And meaning all the CPU time was going to one of the processes. And you can configure maximum CPU time limits and stuff like that. That's not that interesting for us I think. But might be interesting eventually.

Then we have the IO controller. So, this is, again, statistics and pressure information. If we are waiting on IO. We have IO, weight, again, the same thing for CPU, just for IO. And then there's latency, IO latency which allows you to configure latency guarantees. And then if these latency guarantees are not met, basically the IO weight, the cgroup is going to get a higher priority.

It's not very useful for us. And it's not relevant. Because setting IO.weight and saying gnome-shell -- the important session purposes have the same right to receive IO as all the applications. If the share is 50-50 -- and gnome-shell doesn't need much IO. It will be drowned out. It's okay to just enable the controller for us.

And the other problem with IO controller is that a requires more configuration work that systemd doesn't do yet. So, you actual need to put files in this to enable it properly. And a further limitation right now is an works real well on a plain partition. So, unfortunately, it doesn't allow us for -- it doesn't help us right now. But once it does, it will allow us for proper separation. Because then, even if one application is thrashing and waiting on IO and stuff like that, and the disks, then the other application will still receive resources and can keep running. Even though the system is overloaded in theory. So, we can actually properly separate everything.

And finally, we have the memory controller. Which, again, same thing. We can get the statistic, we can pressure -- we can get the pressure information. How often are we waiting for memory to be loaded? This is either swap backend or page folds. It might be page folds for cachers. So, if they have been evicted from the cache. And then we have other controls like memory.low and memory.high, which allow us to protect this part of the system from memory reclaim. It's not going to be swapped out. Or the other way around, to basically say this shouldn't grow larger. And it will be very slowed down if it tries to.

The most useful one here is memory.low for us. Because then we can say, basically, it means this part of the system treated as if it's using less memory by this amount. So, if you say, I don't know -- memory.low is 200 megabytes and I have a 200 megabyte and I have a 400 megabyte process. It will compete as if it only has 200 megabytes in memory. And lastly, we have the memory.oom.group, which is all together instead of killing a single process. Makes sense in some situations. But doesn't make sense to kill your entire browser just because one of the processes needs too much memory and needs too much room. It would be sufficient to kill the oom.group. In other work scenarios, it could make sense to kill it all in one go.

Let's do a quick demonstration of this so you can see what's going on. Share my screen again. Share. So, I'm just going to use two you know journal tabs. It's very simple. Journal already puts all of these -- it separates cgroups. I'm going to stress one in one terminal tab and pin it to CPU 0 so they are competing on the same CPU. I'm going to start to in another. What we are seeing now is that these three processes are all getting about one-third of the CPU time. This is exactly what we would expect from a system not running cgroups. All of these processes are treated as equals and they receive one-third of CPU zero.

So, now what I can do is that GPL added four. So, I added the unit that we first showed. Oh, I can -- in -- I can now read out that cgroup information. Which is all this one path. This is -- to this -- and the cgroup. I'm in the cgroup where one of the processes is running. If I now go in the top one cgroup up and look at the controllers, you can see that here it's only memory and it's listed. What's not listed is CPU. So, CPU currently is disabled. So, if I now set a CPU weight on those of these tabs, basically, in systemd and I just store that. What happens is that it enables the CPU controller so it's now listed here.

If I now look at top, what you will see is that test one, so, the first tab, is getting about 50% of the CPU. And test two, the two together are also receiving together half of the CPU. So, now we are scheduling both applications, both tabs, fairly against each other. Rather advantaging the one that is running two tasks. Let's kill all of this again and go back.

All right. This is cool. We can already use this. You just need to set it up correctly in systemd. It does everything for us. It will group everything nicely in a hierarchy for us. We can control the relevant knobs which is mostly these three. MemoryLow, CPUWeight, and IOWeight. However, it would make sense to be able to tell which application is running, for example. So, we want to encode some more information into structure it in a way that is more useful to us.

So, this is why we developed it. This was with David Edmondson worked with us and the systemd people. We discussed this. And the graph is actually committed into the systemd repository. You can look it up there. But basically, the idea is that we'll split the user into three parts

We have a session.slice, that's the important stuff that you want to keep running. And then you have app.slice, the normal applications, and background.slice, background tasks like running a back ground or something. Which really shouldn't compete with -- slow down your applications if it's running.

Another thing that we want to do is encode the application ID in the systemd unit name so we can figure out which application program belongs to. So, this allows us to do everything. And we can also, for example, create the task manager -- improve the task manager that way and manage per-application resources if we actually containerize the applications in that way.

This is an example of the one where you can see that Spotify is grouped together and you can see the different processes belonging to it. This is something that I think David Edmondson did this. Might be committed even. I'm not sure about that one. Correct me.

And then the other day I tried this out in GNOME-usage. So, you'll notice here is this is the before. I was using it at the time. It's clocking in at 567 megabytes, but the system is using 3.4 gigabytes. But if you start making the application ID from the cgroups application there, the picture changes a lot. The system only uses 1.8 gigabytes, all of the stuff that can't be grouped. And web is using 2.1 gigabyte.

And you can see that the heuristic that we used there was not able to pick up the fact when it was actually launched, spawning separate processes for the rendering tasks that used up most of the memory. Right.

So, next thing was -- the idea we separate was -- the idea was to separate everything out and create this session slice and stuff like that. And we also had discussions in Fedora about early -- and protecting all the system. Again, thrashing situations. What came up there was with all this infrastructure being able to set memory low, we can actually be smart about this. We could give an allocation of memory to the session, to the core processes, to the important ones, and then try to ensure that if something starts thrashing, it's going to mostly hit the processes that are not as relevant.

And it's everything that's there. We have only the level tasks done basically. We basically just need to drop in a lot of configuration files in. So, for GNOME, this means making it more conformant to the systemd. This is not as conformant. This is simple to do with con fission rations. And what I had up, the delegate, tracks which session is active. And based on that, it will delegate 250 megabytes. That's the default configuration to the active user. And this is then also further delegated to session.slice. So, basically, all the important session processes like gnome-shell, gnome-settings, daemon, they get a memory guarantee of 250 megabytes way and are treated as if they use 250 megs less overall. And it also means that they should get better behavior in low-level memory situations.

They will not lose their file caches if someone else needs that memory. So, they should be a lot smoother if it comes -- if it comes into weird situations where there's not resources -- not enough resources available. It also, because it's easy to do, it will just give a higher priority to the active user. So, right now in my system, if I logged in through SSH as a root and ran the same test, it would get 20% of the CPU. And my stress test would get the rest. Yeah. So, the fact it that by any ballooning cgroup controller, the applications are going to compete equally. The active user will receive a greater share of the CPU. And the session is protected from thrashing. Because thrashing situations is usually memory waiting for memory. And the task memory of its own, it's not going to be as effective as the applications. In the extreme case, it might be all of your applications are completely frozen, but gnome-shell is not that responsive. It's probably not going to be that good in reality. But that's it.

So, yeah, this is pretty neat overall. What's a bit of a shame is, it's hard to test this. I would have loved to actually demonstrate this working very well. But creating artificial situations that thrash doesn't seem to be that easy. I don't have very good tests to try this out, to be honest. Yeah.

So, yeah. And you can fork-bombed. If I ran a fork-bomb in one of my journals, it wouldn't do anything. It would hang. If you want to try this on Fedora, it's that simple. And get the service and reboot. I've proposed this as a change request for Fedora32, and there's configurations and not much that can go wrong there. It could be fun trying it out.

In the future, I would like all of this to die, to be honest. The daemon, most of the stuff in there should be upstream elsewhere. So, the systems to GNOME need to be updated. Some configurations will need to be changed on the systemd level so the whole delegation system works. And the policy itself, to give resources to the active user, we will probably need to find a new place for that because it doesn't really make sense to just have the daemon running just for that purpose. It might be systemd log in as an obvious choice. But it doesn't feel quite like the right place. I'm not sure yet. Open to suggestions, obviously.

And, yeah. So, just as a quick overview of what there is to do on the GNOME side and systemd-related. So, we should become more compliant with the systemd conventions we discussed. At the task manager, so that's also clear. One of the biggest issues at this point other than finally merging some of the systemd cleanups that's going on is that we need some new APIs. So, if I have a chat program, start launching my browser and the browser wasn't running and shouldn't end up in the same cgroup as the chat application. It should get its own cgroup. The API is basically just launching the process. And then you either need to move it into a new scope. Or the alternative is probably to just go through the next DG portal and delegate the launching to someone else, another service, which can ensure that the systemd hierarchy is correct.

And there are other things going on. So, systemd-oomd will be merged very soon and we should look into using that. So,s is systemd-oomd as an information, basically, the approach is there. What you do is measure whether the system is thrashing and if you detect it thrashing, then you'll start killing processes. You don't wait until you're out of memory. But you wait until the point until you notice everything is actually slowing down.

The problem -- a bit of a problem with that is is that it's always slower because it will -- because you need to measure for a while. So, it will measure for like 60 seconds. In those 60 seconds more than the certain amount of time was spent waiting on the disk. At that point you go, well, this is probably bad. I'm just going to kill some process. That's the general idea here. And then we also need to figure out systemd to become a bit smarter by cost models. There are some rough edges still.

And for people using LUX, you want support for the IO scheduler. All right. That's it from me for now. I'm going to -- as I understand it, the questions are in the Etherpad, right? Where's the Etherpad? There.

>> If you hit the button, the questions is in the shared notes on the left.

Benjamin Berg: Oh, sure. Yeah, yeah. I don't think I had it open yet. Questions. In a full screen game is running, could this have extra weight given to it? Yes.

>> Read out the questions aloud for everybody.

>> The first is a if a fullscreen game or unredirected window is running, could this have extra weight given to it? This is actually the exact example basically of the demonstration from David Edmondson. What was going on is the window manager detected which window was active and then gave a higher CPU share, CPU weight to that application. And obviously, as long as you are able to detect which application is actually focused at the time, you can do that. It's not a problem at all. You can also do other things like freeze the application completely or something like that.

All right. Second question. Once apps and services are in a systemd tree, what would -- what -- what work would need to be done to the systemd to be part of the stack to allow for restarting of the session without losing running apps and services? Interesting.

So, right now what we do is actually ensure that everything just shuts down, mostly. So, what will happen currently is that we'll have a ritual target called graphical session.target. When we look out, we ensure this is stopped. Everything that's part of this target will be stopped automatically. And in GNOME, what we actually also do is that we restart the daemon when you log out. Just to make sure that evolution is dead. Which is a work around for evolution not having the graphical session. But the target in the systemd init. You can basically -- you could already do that if the application is smart enough. But the problem is that you won't be able to connect. The environment there is you're getting the structure to connect to Wayland, will be wrong. For a DBus service, it would work. Except we would restart the DBus daemon right now as a work around. Yeah. So, not great, the situation.

Also, remember that 10 seconds after you log out, last session is closed. systemd will generally stop your user instance and everything is lost anyway. Whether a laptop is on battery mode, can we be more aggressive to non-focused apps to give them less CPU time to save power? This is interesting. I think from a kernel perspective, we can. Because we have the -- let me go back to the slide.

We have the CPU.max and stuff like that. And there you can say something like in the seconds, it's only allowed to get 10 milliseconds of CPU time or something like that. I think. So, CPU.max. And there are some other controls you would need to check. It could be that systemd does not allow you to set these yet. So, you might need to -- it's either in systemd to do it properly, or you may need to run into the cgroup hierarchy directly. Which wouldn't be as nice. It would probably a lot better to, yeah, not do that. To -- first integrate into systemd and then implement something on top of that. But I'm not sure what is actually sensible there. You probably want something else like monitoring how much CPU it's using and then telling the user that this application is doing something bad.

How does unmerged systemd-oomd fit into the picture? All right. This is a bit more complicated. So, as I see it, the problem that we have is, in general, this is -- this is about the situation where the system is linked -- starting to thrash. And you run into those memory situations where you need to kill a process to get the system back to working. And in general, as I see it, there are two ways of handling it.

One is being proactive about it. We protect the sessions, we give it memory. The other way of being proactive about it would be a system like ROUM, which simply -- basically, in my view, is effectively ensures that you have enough memory for file caches which are essential because your applications are nothing else but binaries on disk when they are running. And then you have the systemd-oomd which is more of a reactive approach which you are actually able to measure that the system is thrashing. And you only start to kill if things are going bad. And the thing here is that you need something proactive. If you don't want the session to be completely locked for 1 minute because you're measuring over a minute here. So, you need to protect the session. And so, you need some proactive measure. Right now in Fedora, this is a proactive one. If we start to protect the session by allocating memory to it and allocating guaranteed resources to it, we might be able to instead of running and switching to systemd-oomd and then we are going to be a lot smarter about which applications we are killing and when are we killing than with the heuristics. So, basically, doing all of this should allow us to move to systemd-oomd and be smarter about how and when we are going to kill applications to keep the system running smoothly. All right. Next question.

Is there something background daemons can do now to opt into the improved resource management? Can they already specify background.slice somehow? Yes. All you need to do it -- I can write here, right? So, all you need to do is just do slice -- slice = background.slice in your systemd image. And that's it. And if you look at what -- what your resource does -- wait -- share your screen again. Look at your resource D, all it's doing is just dropping in configurations. Here gnome-shell service is going to get service. And here, it's moving to session slice. So, really that is all there is to it. So, if you want to -- we moved into a background.slice. Background.slice currently doesn't have any configuration yet. But we can add that.

All you need to do is -- this is wrong, actually. Is to write the slice unit there. Whatever. Okay. Next question. Can we have something -- can this -- can we have this enabled in GNOME OS? Well, sure. The easiest thing is to just install your resourced because it does the configurations and does their -- it's important because it needs to assign the memory limits to the -- on the system level. So, you need a small system daemon doing it. But, yeah. You just are installing your resourceD should do it. Why is GNOME calendar daemon 2.2 gigabytes. Let me check my slide.

Where was this? This is Evolution. I'm just using Evolution. That's not evolution Calendar. It's evolution as a whole. It's explaining why it's 2.2 gigabytes because it's in there. Okay. Next question. Could you expand on what needs to be done if someone creates their own application launcher so new applications are grouped correctly?

Basically, I think the best thing is if we provide a low-level API that everyone should use. Because otherwise we'll -- well. There are two ways. Either you go to the systemd page and you basically do exactly what it says there, which would be very hard. It's not that hard. It's just a way of encoding the application ID into the systemd unit name. But you will need to either launch the application using systemd or launch normally and move into a systemd scope. It would be a lot easier for everybody if there was a good API and everything will be done correctly. And the same thing is true for Flatpak in my view because Flatpak basically needs the same -- has the same issues, right if had because you might need to launch an external link or something like that outside of a Flatpak.

Okay. Next question is, Flatpak starts units, scopes, for each app in the user session. Can these be auto-moved to pack ground if it matches some setting or such? This is interesting. So, what you can do is you could -- there are different ways of doing this. One thing is that Flatpak already needs to generate a unit configuration. So, if Flatpak knows this should be running in the background, then it could just explicitly move it into the background right away. The other thing is, if you as a user want to do it around Flatpak, you could configure it through systemd directly. So, you can create a file that basically says, okay, this Flatpak with this application ID should be moved into background.slice. And then you could set your own limits and tell it can only use 1 gigabyte of memory or something like that. I've actually did that for fun for my Evolution so it could not eat more than 1.5 gigabyte or something like that. And it worked. Like at some point the kernel would just go and kill the process and it would just restart can have evolution and I wouldn't need quite as much RAM. It's better now with the new version.

Okay. I think that's it for now. I'm happy to say more.

>> You can have 5 more minutes if you want.

Benjamin Berg: If someone asks another question, I'm happy to answer it. But I'm not sure what I should talk about. We have people typing. The whole systemd boot. Of course. We want to do everything with systemd, obviously.

GMemoryMonitor. Which is that? Oh, GMemoryMonitor looks at information, right? The question was is how did does this interact with GMemoryMonitor? GMemoryMonitor looks at -- let me just share my screen. I took a look at pressure memory, I believe. You can see here, it's how often some applications and this whole system is waiting on memory. So, these statistics will still be correct. However, if we are protecting the slice now, then full, like the -- some parts of the system will still be running smoothly and other parts might not be running smoothly. So, I do expect that it might skew these statistics slightly so they're full statistics. Because if gnome-shell, the session is actually doing work, and it has a protection. So, actual getting work done, other parts are not get anything work done, then the sum line will say, everything is out of whack. We're not getting anything done because there's some processes that really doesn't get anything done. But the full system overall is still mixing up the CPUs. It's not waiting on the disk because the processes -- there are some processes that are getting their work done and they are getting enough IO resources.

So, I imagine that in some situations it could skew these statistics a bit more. It probably shouldn't be that bad. Like it could already get into the same situation today if there's a small process just crunching away on the CPU and then, yeah. You're not going -- it's not going to run into memory pressure. So, the full system is always doing something useful. So, the full line will -- not show anything. But the some line will show something. It can also happen now.

Okay. What else? Are those changes for resource management also compatible with older systemd versions/systems without systemd. Systems without systemd, absolutely not. There's no way to do this without cgroups and something that manages the cgroups for us. Other Systems are problematic. I just proposed the patch today to make systemd 2 and make this work better. So, yeah, so, even systemd versions, we might need to create new versions for all of this to work quite well. Yeah.

So, I think that's it, then. What will be -- again, quickly, what will be in GNOME 3.38? I'm planning to get the -- like the restructuring of the session to GNOME 3.38, a few cleanups, maybe. But you'll need stuff from externally. Like resourceD or something like that to actually get any benefits. I don't think it makes sense to enable the secret controller yet from GNOME side.

All right. I'm done, then. Sorry for not answering the last question.

>> Thank you very much. You can still apply to answer the question over the chat. There is a link in the shared notes. You can continue the discussion over there.

Benjamin Berg: All right.

>> Thank you very much.

[Break!]