Episode Book

EPISODE #6452

Why Recursive File Watching Strains Linux Memory

PUBLISHED	4 November 2026 at 13:03 UTC
DURATION	25:18
LISTEN	myweirdprompts.com/episode/recursive-file-watching-linux-inotify-53332/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	z-ai/glm-5.3
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.26
SOURCES	20 cited by the research (listed at the end)
BOOK GENERATED	5 November 2026 at 07:03 UTC

SYNOPSIS

Linux file monitors like inotify make instant automation possible, but watching entire trees reveals an expensive kernel secret. The kernel provides no native recursive watches, forcing tools to spawn individual monitors per subdirectory. Each watch consumes unswappable kernel memory, creating hidden memory pressure on large directories while write paths silently pay the cost of constant checks. Exploring the system calls, memory footprint, and event-queue overflow behaviors reveals how background file indexers quietly fail.

TRANSCRIPT



Corn

There's a particular kind of small delight on a Linux desktop when the machine seems to notice what you just did. You clone a repo, and before you've typed a single command, something else on the box already knows. Daniel's prompt today is about how that actually works. His scenario: you create a new GitHub repository, clone it locally, and you want that folder to land in zoxide's index automatically, without ever visiting it first. The foundation underneath that kind of automation is inotify, the kernel's file-change notification subsystem, usually reached through the inotify-tools package and a command called inotifywait. And he's clear it matters on servers as much as desktops.



Herman

Daniel being Daniel, the convenience question has a load question hiding inside it. Real-time notifications without choking the machine.



Corn

So that's the shape of it — what is inotify actually doing down there, what does it cost, and then the zoxide pattern itself. Start with the idea. The whole trick is an inversion: the kernel tells a process when something changes on the filesystem, instead of the process constantly asking.



Herman

Polling is the old world. You'd sleep for thirty seconds, stat everything, diff against your last snapshot, sleep again. It's wasteful at any scale, and it's always slightly stale — you find out about a change up to thirty seconds late. inotify inverts it. The kernel already knows the moment a file changes — it's the one doing the changing. So it just tells you.



Corn

Push instead of poll.



Herman

That one inversion is why desktop search indexers, sync clients, build watchers, all of it are viable. And the history explains the design. inotify was merged in kernel 2.6.13, released August twenty-fifteen… no, August two thousand five. Created by John



Herman (continued)

McCutchan with Robert Love, replacing dnotify.



Corn

What was wrong with dnotify?



Herman

Robert Love's design document is wonderful on this. Dnotify kept the watched file open — a file descriptor held on the thing you were watching. And holding the file open pins it, which worse, pins the mount. Love's line is that dnotify is "therefore infeasible for use on a desktop system with removable media as the media cannot be unmounted." You couldn't eject your own USB stick because a search indexer was looking at it.



Corn

"Watching a file should not require that it be open." He actually wrote that?



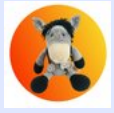
Herman

Verbatim. And there's a nice modern footnote — since kernel 4.3, around twenty-fifteen, inotify and fanotify are both frontends sitting on the same fsnotify backend in the kernel. When people say inotify today, they're describing one access path to a shared mechanism.



Corn

Let's open the hood. Four system calls, you said.



Herman

That's all of it. Inotify_init creates an instance and hands you back a file descriptor. Inotify_add_watch takes a pathname plus a bitmask of the events you care about, and returns a watch descriptor, an integer. Inotify_rm_watch removes one. And closing all the file descriptors referring to the instance tears the whole thing down — every watch on it goes automatically.



Corn

One fd per instance, not one per watch. There's a reason, presumably.



Herman

Love's rationale again. An fd per watch burns file descriptors fast, and there's the ordering argument, which I think is the good one. With a single fd you get a single queue of events, and a single queue trivially gives you ordering. His example: imagine rename a to b, then rename b back to a, with the two events arriving out of order. You'd reconstruct a filesystem state that never existed.



Corn

That would be a fun bug report.



Herman

So events come off that fd as a stream of little structs. Watch descriptor, event mask, a cookie field that links the two halves of a rename together, and a name — the name is only present for events on objects inside a watched directory, not the directory itself. This is why parsing inotifywait's output is a different skill from reading the raw API, and we'll come back to that.



Corn

Now the fact I want planted before anything else, because everything about load follows from it.



Herman

Watching is per-directory, not per-tree. There is no recursive watch in the kernel. None. When a tool offers you recursive watching — and inotifywait does, with a flag — recursion is a userspace fiction: the tool walks the tree and adds one watch per subdirectory.



Corn

So watching my whole home directory "recursively" is really hundreds of separate kernel watches wearing a trench coat.



Herman

Hundreds is underselling it, but yes. Now the event vocabulary, because there's a trap in it that catches almost everyone. IN_CREATE is what the man page calls a plus event — it fires for objects created inside the watched directory, not for the directory itself. And here's the trap: IN_CREATE alone doesn't tell you whether the new thing is a directory or a file. You have to check for the accompanying IN_ISDIR bit in the mask.



Corn

And people don't.



Herman

Constantly. There's a Unix Stack Exchange thread where someone's script was watching for close_write and wondering why new folders were invisible. The answer: "you're looking for CLOSE_WRITE instead of CREATE with IS_DIR." The demonstration is two terminals — you run inotifywait on a directory in one, make a folder in the other, and out comes the event with the ISDIR flag on it. Folders and files announce themselves differently, and if you listen for the wrong announcement, folders simply don't exist as far as your script is concerned.



Corn

Which is exactly the event Daniel needs, so it's worth being precise. Now, inotify-tools itself. I went down the rabbit hole on the package.



Herman

You did warn me you would. It ships inotifywait, which waits for or monitors events, and inotifywatch, which gathers usage statistics, plus a small C library the tools share. The detail I love — it's now written in Rust. And the Rust rewrite ships a libinotifytools that is a drop-in, ABI-compatible replacement for the historical C library. Same symbols, same soname, same headers.



Corn

Nobody was supposed to notice, and the noticeability is the impressive part.



Herman

The release cadence is also unusual. Every commit on master that passes CI gets released. So the version numbers look like four point twenty-six point two-seventy — series, then commit count. The latest is from October, and its release note is "fanotify mount watch," which we'll come back to at the end, because it's a tell about where the tool is drifting.



Corn

The default behavior of inotifywait is the bit that makes the classic idiom work. It exits after one event. There's a monitor flag to keep it running forever, but the default is one and done.



Herman

Which sounds lazy until you see the loop people build with it: while inotifywait with the create event filter, do the thing you wanted, done. The tool fires once, your loop body runs, and the loop re-arms the watch. The one-shot default isn't a limitation, it's the mechanism.



Corn

And the output. The default format is hostile.



Herman

Notoriously hard to parse, because special characters in filenames aren't escaped by default. So there's a format option with the usual printf-ish conversions — the watched directory, the filename that caused the event — and that output truncates at around four thousand characters. Obscure until the day you have a path longer than that, and then very not obscure.



Corn

The recursive flag. The man page warning on that one is practically a public health notice.



Herman

It is, and it's worth hearing the shape of. Watch the root of a large tree and it may take a long while before all the watches are established, and events are missed meanwhile — and because one watch is set per subdirectory, you can hit the per-user watch maximum while you're standing there. Default maximum, eight thousand one hundred ninety-two.



Herman

It's the load story in miniature, actually. The recursive flag is where userspace's tidy fiction — one tree, one watch — collides with the kernel's reality — one watch per subdirectory, each costing real memory.



Corn

And that gets us to what Daniel actually asked. How much does all this cost, and what happens when it goes wrong.



Herman

The answer has a counterintuitive shape. inotify is cheap in CPU and expensive in memory. One watch costs five hundred forty bytes of kernel memory on a thirty-two-bit system, and about double that, a thousand eighty, on sixty-four-bit. And kernel memory doesn't swap. It can't be pushed out to disk under pressure — it's pinned.



Corn

A thousand bytes doesn't sound like much per watch.



Herman

Multiply. A million watches — and people really have run millions on their home directories — is roughly a gigabyte of kernel memory that cannot be swapped, ever. And on top of the kernel cost, the application holds its own bookkeeping per watched item:



Herman (continued)

about ten bytes plus the full filesystem path string. The path string is the sneaky part. Deep trees have long paths, and the userspace cost scales with the characters.



Corn

So the people watching all of home aren't spending CPU. They're spending a gigabyte of the one kind of memory the kernel can't take back.



Herman

The CPU side is real but small, and the kernel has optimized it repeatedly, which tells you it mattered at the margins. Mel Gorman did a patch in twenty-twenty rearranging the fsnotify fast path — the code every write hits even when nothing is watching — and reported two to nine percent latency gains on a benchmark, and the fsnotify maintainer replied that he was surprised the win was that big. Gorman's line: "the fsnotify paths are trivial to hit even when there are no watchers and they are surprisingly expensive." There was an earlier patch reporting nearly fourteen percent more writes per second on unwatched files.



Corn

Hold on. That's the price of nobody watching?



Herman

The price of checking whether anybody is watching. It's in the path of every single write, so a few wasted instructions cost everyone, always. That's the CPU story — small, shaved repeatedly, nothing a desktop user ever feels.



Corn

Okay, so what actually goes wrong? Because you said the failure mode isn't slowness.



Herman

It's the queue being dropped. Each instance has a kernel queue of unread events, capped by max_queued_events, default sixteen thousand and change. When the queue overflows, the kernel does not do the polite thing, which would be ignoring further events. It drops the whole inotify instance and signals an overflow event to whatever was reading it.



Corn

The whole instance.



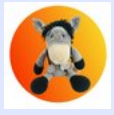
Herman

The watcher is evicted, watching nothing. Watchexec's documentation calls this "usually a fatal error for an application, and may be hard to recover from," and says applications will want to zealously avoid that happening. And there's a real case: crowdsec, the security tool, has an issue open since late twenty-twenty-three where its Go watcher library hardcodes the event mask — it can't narrow what it asks for — so it gets flooded and overflows, and the tool just stops acquiring files. Still open, last I looked.



Corn

A security tool that quietly stops watching, and reports nothing. Herman, you used to read error logs the way you read worried parents. What does that one look like from outside?



Herman

From outside it looks like nothing. That's the nastiest part. The process is running, the dashboard is green, the filesystem is churning, and the queue was dropped an hour ago. A system that reports itself fine is exactly the system I was trained not to trust.



Corn

Which is a lovely design, if your goal is plausible deniability.



Herman

The configuration answer, then. Three knobs, all under proc sys fs inotify. max_user_instances, default one hundred twenty-eight — how many separate inotify instances a user can hold. max_user_watches, default eight thousand one hundred ninety-two — total watched items across all your applications. max_queued_events, default sixteen thousand three hundred eighty-four — the queue depth. And here is the coupling Daniel's question is really about: these limits are per real user ID. Not per application.



Corn

So my zoxide watcher, my editor, my sync client, my file manager and my browser's download monitor are all drawing from the same pool of eight thousand.



Herman

All of them, one pool, and you can't set different limits for different users, let alone different apps. So "without choking the whole machine" really means without starving your other watchers. And the tell when you hit the ceiling is beautiful and horrible: ENOSPC. No space left on device. Tail reports it, Ruby watcher libraries throw it, on a filesystem that is nowhere near full. Generations of administrators have gone looking for a full disk and found a watch limit instead.



Corn

I'd have enjoyed that error more if it were honest. What does raising the limits look like?



Herman

Watchexec calls the default "much too low" and suggests sixty-five thousand as a starting point; Syncovery goes to about a million watches and a thousand instances. You set it permanently with a small file under the sysctl dot d directory, or temporarily with the sysctl command. Corn, can I ask you something — would you raise it on your own machine, or leave the eight thousand as a tripwire?



Corn

Leave it. You know my position: I distrust anything that changes while I'm asleep, which is most of the day, and a limit that complains loudly is the only part of a filesystem that ever tells me the truth.



Herman

I suspected as much. I raised mine within the hour, obviously.



Corn

Obviously.



Herman

What falls out of all this, as consequences rather than advice: watch the narrowest directory that solves your problem — not home, not recursive on a big tree. Filter events so the queue never fills — create with the directory bit, not everything. Keep your consumer fast, because the queue drop is fatal, not lossy. Those aren't tuning tips, they're just what the cost model implies.



Corn

Let's go to the voicemail line. Yara, you're on.
Yara: Hi Corn, hi Herman, this is Yara calling from the back room of the copy shop on Patan Dhoka road in Kathmandu, where I work the evening shift and the machines hum so loud I have to press my phone right against my ear. I wanted to say I loved the episode about the file watching thing, the inotify one, though I kept hearing it as notify and thinking of my cousin who never tells anyone anything until it is too late. My question is small. When I save a scanned document and the folder makes that little new file, does the kernel remember every single one of those forever, or does it forget them when I close the laptop and sleep?



Herman

Yara, the kernel is more like your cousin's opposite here. It doesn't remember the files at all — it remembers the watch, the folder itself, and that survives closing the laptop fine. The individual events only exist in a queue until your application reads them, and then they're gone. Sleep suspends everything, but the watch comes back with the machine.



Corn

And if the application is slow about reading, that queue is the part with the ceiling — which is the next thing on our list anyway.



Herman

The zoxide pattern itself, then, because this is where everything we've said becomes one small script. zoxide, as we've covered before, is the smarter cd — it ranks directories by how often and how recently you visit them. Normally it scores a directory when you cd into it, via a shell hook. The key that makes the watcher pattern work is zoxide add, which writes to the database directly, bypassing the hook entirely.



Corn

That's the whole trick. You don't fake a visit — you just tell zoxide the directory exists.



Herman

And the maintainer's own sanctioned bulk path, from an issue this year: zoxide add handles multiple paths, and most shells support globs. Which is also your backfill — if your watcher was down for a week, one glob picks up everything it missed.



Corn

Now the caveat that breaks the obvious one-liner, because I tripped over this one in my head last night. git clone is not atomic.



Herman

Not remotely. Git creates the target directory first, then populates it. So the moment the directory exists — the instant your create-with-ISDIR event fires — the repo is empty or half-written. A watcher that adds the path immediately indexes a directory that's still being born. The robust version waits for a file inside .git to close — the index or the head file — or just debounces a few seconds and adds whatever survived.



Corn

So the naive script isn't wrong, it's early.



Herman

Early by milliseconds, which for a frecency database is a rounding error, but it's the difference between "z repo works" and "z repo drops you into an empty folder." And one more caveat with teeth: zoxide ten, from this summer, added an exclude-dirs variable, so imports skip directories the user deliberately excluded. A watcher that ignores that setting will cheerfully re-add everything the user told zoxide to forget.



Corn

Undoing the user's own decisions on their behalf. Software's favorite hobby.



Herman

Here's the thing though, and I checked this properly: nobody has packaged this. There is no zoxide-plus-inotify project, no tool, no gist that does exactly this. The closest cousins are the chezmoi loop we mentioned and the maintainer's glob advice — same shape, neither packaged.



Corn

Hm.



Herman

So the caveats are the substance. If you build it, everything that breaks is yours to find. Which, given the queue drop is silent and the early-fire is invisible, is a slightly bracing thought.



Corn

You'd build it anyway.



Herman

I already started it, and I've already broken it once, but that's a story for when it works.



Hilbert

The queue drop being fatal, I agree.



Hilbert

Though the number that bites ordinary people isn't the watches. It's the instances. One hundred twenty-eight of those per user, and I'll tell you how I know. I spent an evening in the early nineties... no, that was something else, that was FAM, that polled. This was a few years back. The machine started refusing things with "no space left on device" and I checked the disk three times before I worked out the disk had nothing to do with it. Every terminal I'd left open had its own inotify instance. I had fourteen running. Some of those terminals I had not touched in days, and one of them — Herman will enjoy this — was still holding a watcher on a folder I'd deleted the contents of.



Herman

Fourteen terminals, and you were near the limit at a hundred twenty-eight?



Hilbert

The neighbour had logged me round to his box to look at something, that used a few, and a sync tool and an editor each take their own. It adds up quietly. I keep a written tally now. Paper, in the drawer by the desk. Fourteen terminals open, so the tally said fourteen, and I check it before I open the fifteenth. My neighbour asked why I don't just raise the limit and be done. Because then I'd stop noticing, Herman. The limit is the only thing keeping me honest about how many of these things I've left lying around. Raising it is the same as not counting.



Herman

That's the tell from the other direction, though — you could close the terminals.



Hilbert

I could. I don't. The tally works, the fingers on it are the problem, not the paper. Anyway, that's the danger for your listener — it doesn't announce itself as a limit, it announces itself as a full disk, and by the time you've ruled out the disk you've wasted the evening. Your crowdsec thing was the quiet death. This one's the disguised death. Both of them, the machine sits there looking healthy.



Corn

The disguised one costs you an evening. The quiet one costs you the security tool.



Hilbert

Yes. Well. The paper's in the drawer either way.



Herman

The instance limit being the one that bites desktop users is a good correction, because every guide on the internet leads with the watch count, and a hundred twenty-eight instances fills up on a machine with a busy terminal habit long before eight thousand watches do.



Herman

And it points at the thing we haven't solved. That whole budget — watches, instances, queue — is a shared resource with no accounting. No tool tells you it's holding four hundred of your eight thousand. As more desktop software adopts filesystem watching, that pool becomes a coordination problem, and nobody is coordinating it. The kernel just quietly divides it and lets the ENOSPC fall where it may.



Corn

So the open question for whoever's listening and irritated by that: the limits are per user because that was simple, but nothing stops a desktop from exposing per-application accounting, and nothing has. That's the thread I'd pull on.



Herman

The other thread is where the tooling is drifting. Inotify-tools' last three releases are fanotify work — the latest one is literally called a fanotify mount watch. Same fsnotify backend underneath, and fanotify is more powerful and potentially far more efficient, but it still requires root, because it's geared toward access control rather than monitoring. Whether the desktop watcher of the future is an inotify watcher at all is unsettled.



Corn

Which is a good note to leave open. If you've enjoyed this — and apparently some of you listen in copy shops in Kathmandu, which is the best thing I've learned all week — leave us a review wherever you get the show. It's how people find us.



Herman

Producer Hilbert Flumingtop at the desk, as ever, tallying things on paper we'll never see.



Corn

For more along these lines, there's episode two, Local STT For AMD GPU Owners; episode sixteen, On Deepfakes, SynthID, And AI Watermarking; and episode eleven, How Does Fine Tuning Work Anyway. This has been My Weird Prompts. We'll be back soon.



Herman

See you tomorrow.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **inotify(7) [primary]**
   Linux manual page, man7.org, retrieved 2026-11-04
   https://man7.org/linux/man-pages/man7/inotify.7.html

2. **inotifywait(1) [primary]**
   Linux manual page, man7.org, retrieved 2026-11-04
   https://man7.org/linux/man-pages/man1/inotifywait.1.html

3. **Inotify [primary]**
   A Powerful yet Simple File Change Notification System, Linux kernel documentation (Robert Love, started 2005-03-15; updated 2015-01-04 by Zhang Zhen), retrieved 2026-11-04
   https://docs.kernel.org/filesystems/inotify.html

4. **inotify-tools/inotify-tools, GitHub README and releases (latest 4.26.270, 2026-10-07) [primary]**
   retrieved 2026-11-04
   https://github.com/inotify-tools/inotify-tools

5. **ajeetdsouza/zoxide, GitHub README and releases (latest v0.10.0, 2026-07-04) [primary]**
   retrieved 2026-11-04
   https://github.com/ajeetdsouza/zoxide

6. **zoxide issue #1223, add multiple sub folders using their parent (opened 2026-04-20, closed 2026-05-15) [primary]**
retrieved 2026-11-04
https://github.com/ajeetdsouza/zoxide/issues/1223

7. **zoxide `src/cmd/cmd.rs` [primary]**
retrieved 2026-11-04
https://github.com/ajeetdsouza/zoxide/blob/6324b4e3/src/cmd/cmd.rs

8. **Linux inotify limits, Watchexec docs**
retrieved 2026-11-04
https://watchexec.github.io/docs/inotify-limits.html

9. **Raising Inotify Limits, Syncovery documentation**
retrieved 2026-11-04
https://www.syncovery.com/documentation/raising-inotify-limits/

10. **What are the costs of increasing /proc/sys/fs/inotify/max_user_watches value?, Unix & Linux Stack Exchange, asked 2017-05-21, modified 2020-01-09**
retrieved 2026-11-04
https://unix.stackexchange.com/questions/366357/

11. **What is exactly difference between INotify max_user_instances and max_user_watches?, Unix & Linux Stack Exchange, answered 2019-07-30**
retrieved 2026-11-04
https://unix.stackexchange.com/questions/532709/

12. **inotifywait -e close_write ignores new folders in watch directory, Unix & Linux Stack Exchange, answer by Chris Davies**
retrieved 2026-11-04
https://unix.stackexchange.com/questions/624619/

13. **inotifywait ignoring new folders in watch directory, Unix & Linux Stack Exchange**
retrieved 2026-11-04
https://unix.stackexchange.com/questions/696128/

14. **chezmoi issue #3680, Possibility to watch/scan a folder for new files? (opened 2024-04-02, closed 2024-04-03)**
retrieved 2026-11-04
https://github.com/twpayne/chezmoi/issues/3680

15. **crowdsec issue #2605, Inotify queue overflow in file acquisition (opened 2023-11-20, updated 2025-07-25)**
retrieved 2026-11-04
https://github.com/crowdsecurity/crowdsec/issues/2605

16. **[PATCH] fsnotify: Rearrange fast path to minimise overhead when there is no watcher, LKML, Mel Gorman and Jan Kara, 2020-06-08**
retrieved 2026-11-04
https://lkml.iu.edu/2006.1/00526.html

17. **fs: optimize inotify/fsnotify code for unwatched files, LWN.net, Dave Hansen, 2015-06-24**
retrieved 2026-11-04
https://lwn.net/Articles/649318/

18. **Wikipedia, Inotify (last edited 2026-07-07)**
retrieved 2026-11-04
https://en.wikipedia.org/wiki/Inotify

19. **Wikipedia, Robert Love (last edited 2026-10-03)**
retrieved 2026-11-04
https://en.wikipedia.org/wiki/Robert_Love

20. **zoxide cheat sheet (covers zoxide 0.9.9), linuxkernel.com**
retrieved 2026-11-04
https://linuxkernel.com/cheatsheets/cli/zoxide