Debugging with `strace`, `lsof`, and `gdb` for System-Level Errors

When programs fail silently or behave unexpectedly at the system level—such as crashing, hanging, or failing to open files—**system-level debugging tools** become invaluable. Tools like `strace`, `lsof`, and `gdb` let you inspect program behavior from the kernel interaction level up to memory and symbol resolution.

🔗 2. `lsof`: List Open Files

Every file, socket, pipe, and device in Unix is a file descriptor. `lsof` shows which files a process has open.

🧪 Use Case: Check if a File or Port is in Use

```
lsof /path/to/file
```

```
lsof -i :8080
```

Shows which process is using port 8080.

🧪 Use Case: See What Files a Process Has Open

```
lsof -p <pid>
```

Helpful for debugging leaks, unclosed descriptors, or resource locks.

⍰ Combine Tools for Power Debugging

Problem	Tool(s)	Example Command
	`strace`	`strace ./myapp`

Problem	Tool(s)	Example Command
File not found / permission issue		
Process hangs	`strace`, `gdb`	`strace -p <pid>`, `gdb -p <pid>`
Port/file already in use	`lsof`	`lsof -i :8080`, `lsof /tmp/some.lock`
Inspect call stack / variables	`gdb`	`gdb ./app`, `(gdb) bt`
Memory access violation	`gdb`	Run with debug symbols, trigger crash

📚 Summary

Tool	Best For	One-Liner Example
`strace`	Tracing system calls and errors	`strace ./myapp`
`lsof`	Listing open files/ports by process	`lsof -p <pid>`
`gdb`	Full debugger: crashes, memory, stack	`gdb ./myapp`

These tools together help uncover system-level issues fast—whether it's a missing file, a permission error, a deadlock, or a segmentation fault.