Creating a **custom init system with PID 1 and syscall filtering using seccomp** is a niche but powerful exercise in systems programming and Linux security. It gives insight into how init systems like `systemd`, `SysVinit`, and others orchestrate process lifecycle and apply security hardening.

🔧 Basic Responsibilities of an Init Process

A minimal init process in Linux must:

1. Be assigned PID 1.
2. Reap zombie child processes.
3. Spawn and supervise key daemons or shells.
4. Optionally handle signals like `SIGTERM`, `SIGCHLD`.

🧪 Running Your Init in a Container

You can test it inside a minimal Docker container:

```
FROM alpine
COPY myinit /sbin/init
ENTRYPOINT ["/sbin/init"]
```

```
docker build -t custom-init .
docker run --rm -it --init=false custom-init
```

Note: `--init=false` disables Docker's own tini init process.

📌 Security Hardening Additions

- Use `prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0)` before applying filters.
- Use `capset` to drop capabilities.
- Mount `/proc`, `/sys` manually if you're in a custom namespace.
- Apply seccomp at a per-process level to children.

⬅ END Summary

A custom init with syscall filtering:

- Helps understand low-level Linux process management.
- Provides security isolation for minimal environments.
- Can serve as a lightweight replacement for full init systems in containers.