IEN-008 - Stage 7 — System Service Initialization

At the end of the previous stage, the Linux kernel has successfully transferred control to PID 1.

Whether PID 1 is implemented as **SysV init**, **Upstart**, or **systemd**, the operating system has now entered the userspace initialization phase.

For the first time since the power button was pressed, Linux is no longer focused on preparing itself to boot.

Instead, it begins preparing itself to **serve users and applications**.

This stage is responsible for transforming a mounted filesystem into a fully operational operating system.

What Is a System Service?

A Linux system consists of much more than the kernel.

Even after the kernel has initialized hardware and mounted the root filesystem, the system still lacks many essential capabilities.

For example:

- Networking is not yet configured.
- Remote SSH access is unavailable.
- System logs are not being collected.
- Scheduled tasks are inactive.
- Time synchronization has not started.
- Firewalls may not yet be enforced.
- Databases and web servers are still offline.

These capabilities are provided by **system services**, also known as **daemons**.

A daemon is simply a process that runs in the background and performs a specific function.

Examples include:

Service	Purpose
`sshd`	Remote SSH access
`NetworkManager`	Network configuration
`systemd-journald`	System logging
`chronyd`	Time synchronization
`crond`	Scheduled task execution
`firewalld`	Firewall management
`dbus`	Inter-process communication

Without these services, Linux would boot successfully but remain largely unusable in a production environment.

From PID 1 to Running Services

The initialization sequence now looks like this:

```
Kernel
   |
   ▼
PID 1 (systemd/init)
   |
   ▼
Load Unit Files / Init Scripts
   |
   ▼
Resolve Dependencies
   |
   ▼
Start Core Services
   |
   ▼
```

```
Start Optional Services
    |
    ▼
Reach Target State
```

Each step builds upon the previous one.

Unlike the firmware and kernel stages, this phase is highly configurable.

Administrators can decide:

- Which services start automatically.
- The order in which they start.
- Whether a failure should stop the boot process.
- Whether a service should restart automatically.

Service Dependencies

Modern Linux systems rarely start services in a fixed order.

Instead, they rely on **dependencies**.

Consider an SSH server.

It cannot accept remote connections until:

- the network interface exists,
- IP addresses are configured,
- the firewall is initialized,
- DNS (if required) is available.

The dependency graph looks something like this:

```
Hardware Ready
    |
    ▼
udev
    |
    ▼
```

```
Network Stack
      |
      ▼
IP Configuration
      |
      ▼
Firewall
      |
      ▼
SSH Server
```

If one dependency fails, downstream services may also fail or wait indefinitely.

Engineering Insight

A common misunderstanding is that Linux starts services one by one in a hard-coded sequence.

Modern systems using **systemd** actually construct a dependency graph and start many services in parallel whenever possible.

This approach significantly reduces boot time while ensuring that dependent services do not start before their prerequisites are satisfied.

---

systemd Units

Unlike SysV init, which relied primarily on shell scripts, systemd manages resources using **unit files**.

Common unit types include:

Unit Type	Purpose
`.service`	Background service
`.socket`	Socket activation
`.target`	Logical boot target

Unit Type	Purpose
`.mount`	Filesystem mount
`.automount`	Automatic filesystem mounting
`.timer`	Scheduled task
`.device`	Hardware device
`.path`	File or directory monitoring

This modular design allows systemd to manage far more than just services.

For example, mounting a filesystem or waiting for a hardware device can be handled through dedicated unit types rather than custom scripts.

Reaching the Target State

One of systemd's primary goals is to bring the system to a predefined **target**.

Common targets include:

Target	Purpose
`rescue.target`	Single-user recovery mode
`multi-user.target`	Multi-user text environment
`graphical.target`	Graphical desktop environment
`emergency.target`	Minimal emergency shell

Most production servers aim to reach:

```
multi-user.target
```

while desktop systems typically boot into:

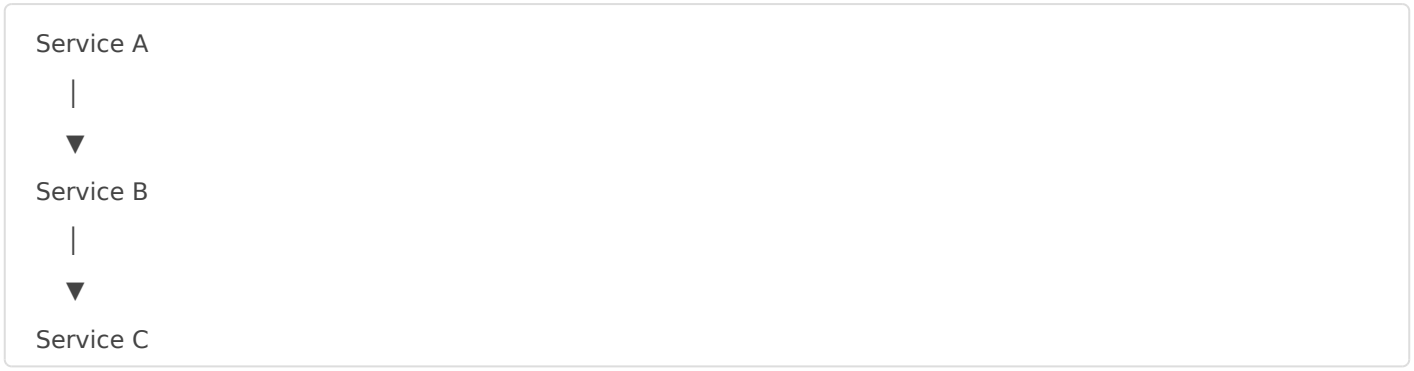
```
graphical.target
```

Once the target is reached, Linux is considered operational.

Parallel Startup

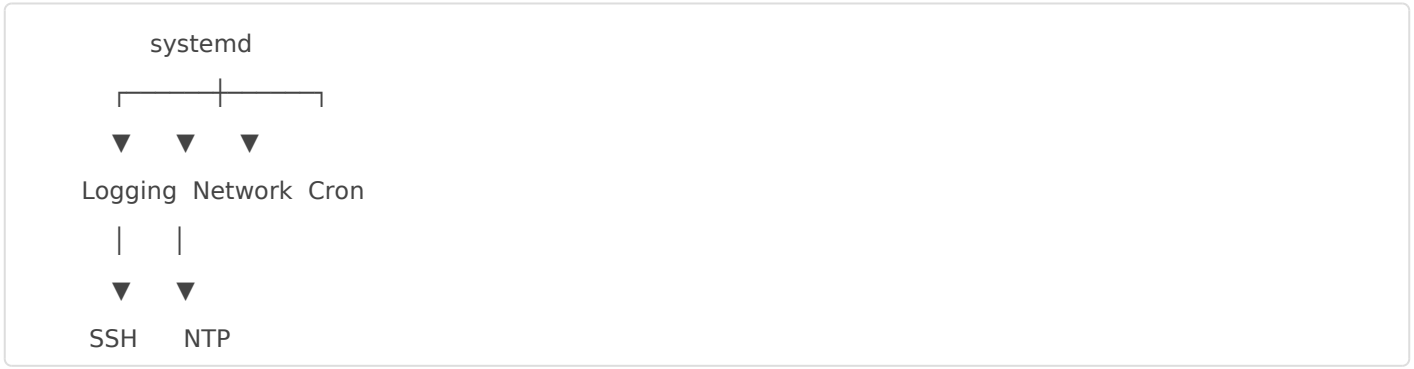
One of the most significant innovations introduced by systemd is **parallel service startup**.

Older SysV init systems generally followed a linear process:



If one service took ten seconds to start, every subsequent service had to wait.

Systemd instead starts independent services simultaneously:



As long as dependency requirements are met, multiple services can initialize concurrently, reducing overall boot time.

Service Failures During Boot

Not every service failure prevents Linux from booting.

For example, if a monitoring agent fails to start, the operating system can usually continue running.

However, failures involving critical services may leave the system in a degraded state.

Examples include:

- Network initialization failure.
- Storage mount failure.
- Authentication service failure.
- Device manager failure.
- Logging subsystem failure.

Systemd records these failures and provides detailed diagnostic information for later analysis.

Investigating Boot Problems

One of systemd's greatest strengths is its diagnostic capabilities.

Several commands are particularly useful when troubleshooting service initialization issues.

```
systemctl --failed
```

Lists services that failed during startup.

```
systemctl status <service>
```

Displays the status, recent log entries, and dependency information for a specific service.

```
journalctl -b
```

Shows all log messages generated since the current boot.

```
journalctl -u sshd
```

Displays logs related to the SSH service.

```
systemd-analyze
```

Reports the total boot time.

Example:

```
Startup finished in 2.432s (kernel)
          + 8.917s (userspace)
          = 11.349s
```

For deeper analysis:

```
systemd-analyze blame
```

lists services sorted by startup duration, helping identify bottlenecks.

Engineering Insight

When a Linux system appears to "boot slowly," the kernel is often blamed.

In reality, the kernel typically completes its work within a few seconds. Delays are more commonly caused by userspace services waiting for storage devices, network configuration, DNS resolution, or external resources.

Tools such as `systemd-analyze blame` and `journalctl -b` are often far more useful than focusing solely on kernel messages.

Common Service Initialization Failures

The table below summarizes several common issues encountered during this stage.

Symptom	Possible Cause
Network unavailable	Network service failed
SSH connection refused	`sshd` did not start
Boot stops waiting for a mount	Filesystem dependency failed
System enters emergency mode	Critical mount or service failure
Long boot time	Slow or blocked service startup

Unlike earlier stages, these failures occur **after the operating system has already mounted the root filesystem and started PID 1**.

This distinction helps narrow the investigation to userspace rather than firmware, bootloader, or kernel components.

Summary

The service initialization stage transforms Linux from a mounted operating system into a fully functional platform capable of supporting users and applications.

PID 1 launches essential services, resolves dependencies, manages failures, and brings the system to its configured operational target.

While earlier boot stages focus on hardware and storage, this stage focuses on functionality—networking, logging, scheduling, security, and every background process required for normal operation.

By the time the target state is reached, Linux is ready to present a login interface and begin accepting user sessions.

Revision #1
Created 4 July 2026 04:22:21 by Kapak Maut
Updated 4 July 2026 04:24:15 by Kapak Maut