

IEN-007 - Stage 6 — PID 1: The Beginning of User Space

At the end of the previous stage, the Linux kernel has successfully mounted the real root filesystem.

The temporary `initrd` environment has completed its job and has been discarded through the `switch_root` operation.

For the first time since the system was powered on, the kernel has access to the complete operating system installed on disk.

Yet one question remains:

“ Who starts everything else?

The kernel is responsible for managing hardware, memory, scheduling, and system calls, but it does not start services, launch user sessions, or manage applications directly.

Instead, it hands control to the first userspace process.

That process is known as **PID 1**.

Everything that happens after this point ultimately begins with PID 1.

What Is PID 1?

In Linux, every running program is represented as a **process**.

Each process receives a unique **Process ID (PID)** assigned by the kernel.

For example:

PID	Process
-----	---------

1	init or systemd
247	sshd
831	nginx
1524	mysqld
2765	bash

The very first process created by the kernel is always assigned:

PID = 1

No exceptions.

This process becomes the **ancestor of almost every other userspace process** running on the system.

The Transition from Kernel Space to User Space

Until now, everything has been executed in **kernel space**.

This includes:

- Memory management
- Interrupt handling
- Driver initialization
- Storage discovery
- Root filesystem mounting

Applications do not run here.

Instead, once the kernel has finished its initialization, it starts the first userspace program.

The transition looks like this:

Kernel Space

|



Execute PID 1

|



User Space Begins

This is one of the most important transitions during the entire boot sequence.

How Does the Kernel Find PID 1?

The kernel looks for a userspace initialization program in a predefined order.

Traditionally, it searches for:

`/sbin/init`

If that does not exist, it may try:

`/etc/init`

or

`/bin/init`

or

```
/bin/sh
```

If none of these programs can be executed, the kernel has nowhere to transfer control.

The result is typically a panic similar to:

```
Kernel panic - not syncing:  
No working init found.
```

This is one of the few kernel panic messages where the kernel itself is functioning correctly, but it cannot continue because the first userspace program is missing or unusable.

Engineering Insight

Many administrators immediately suspect a damaged kernel when they see a kernel panic.

However, in the case of "**No working init found**", the kernel has already completed most of its initialization successfully.

The actual problem is that it cannot execute the first userspace process.

Replacing the kernel would not solve the issue if `/sbin/init` is missing or the root filesystem is corrupted.

The Evolution of Linux Initialization Systems

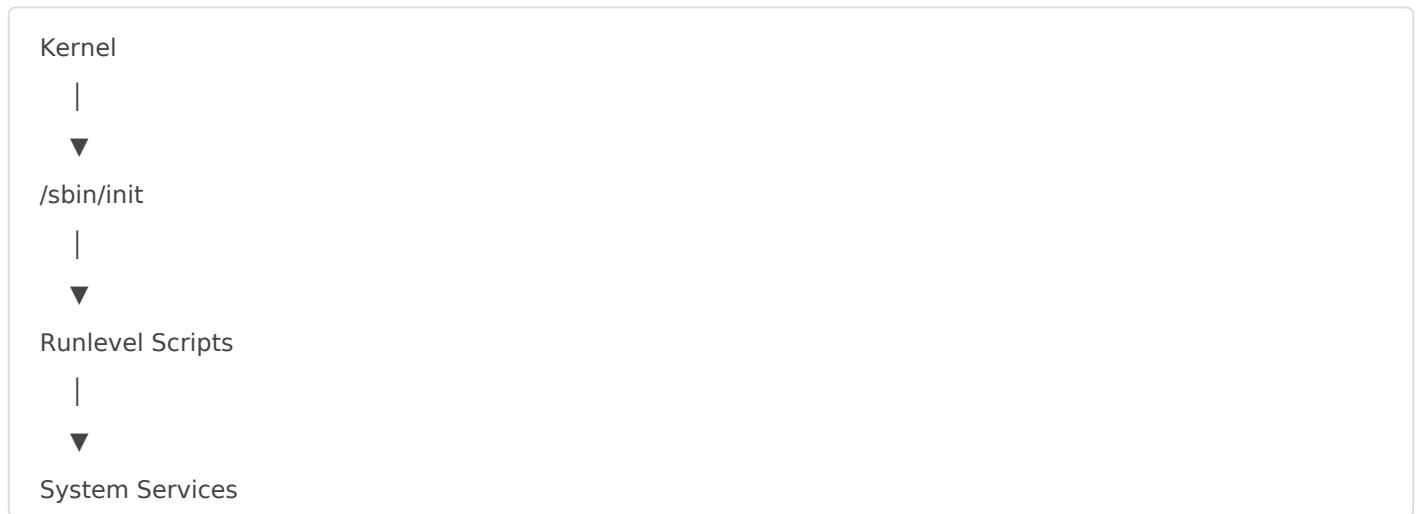
The role of PID 1 has remained the same throughout Linux history, but the software implementing it has evolved significantly.

SysV init

For many years, Linux systems used **System V init**.

This implementation was simple and relied on shell scripts.

The boot process followed a sequence similar to:



Service startup was largely sequential, meaning each service waited for the previous one to complete.

Upstart

As Linux systems became more dynamic, **Upstart** introduced an event-driven model.

Instead of relying solely on predefined runlevels, services could start in response to events such as:

- Filesystem mounted
- Network available
- Device detected

Although innovative, Upstart was eventually replaced by systemd in most major distributions.

systemd

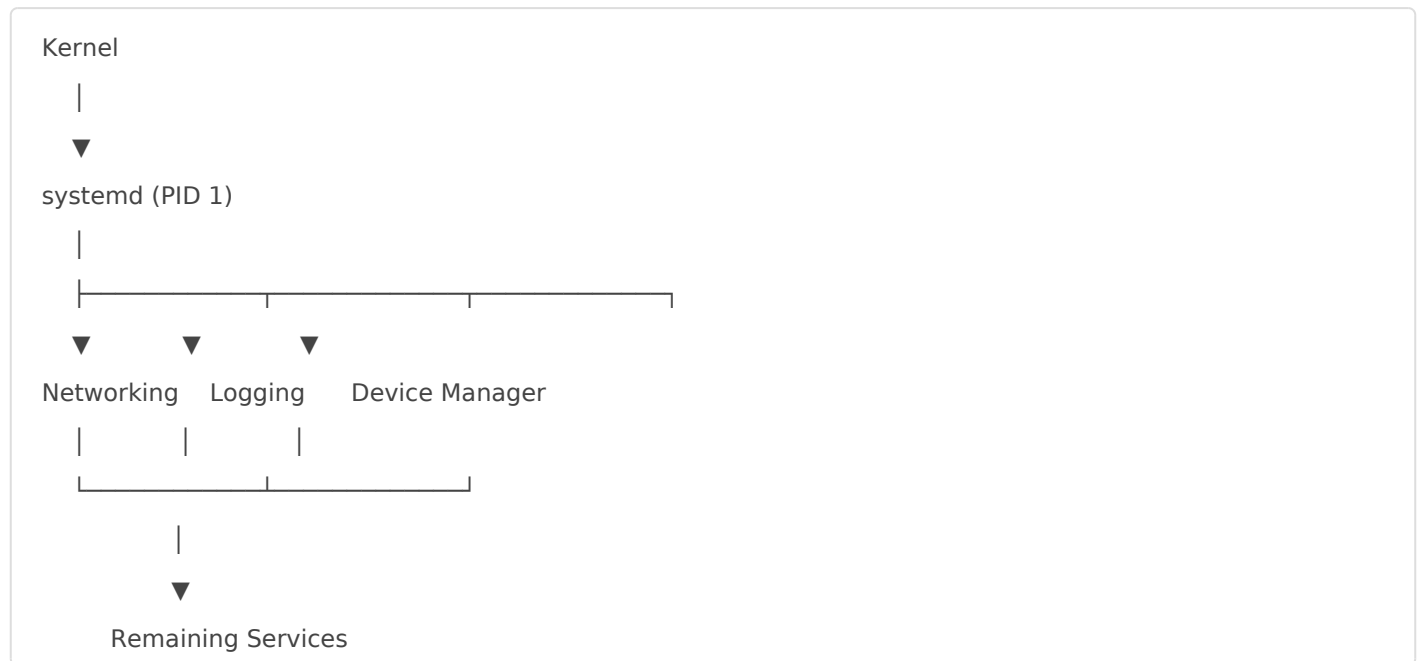
Today, the majority of Linux distributions use **systemd** as PID 1.

Unlike earlier initialization systems, systemd introduces:

- Parallel service startup

- Dependency management
- Socket activation
- Service supervision
- Integrated logging (journal)
- Timers
- Mount unit management
- Target-based booting

A simplified flow looks like this:



This architecture enables modern Linux systems to boot faster and manage services more efficiently than traditional SysV init.

Runlevels vs Targets

Older Linux distributions, including RHEL 5, organize the boot process using **runlevels**.

Common runlevels include:

Runlevel	Purpose
0	Halt
1	Single-user mode
3	Multi-user (text mode)

Runlevel	Purpose
5	Multi-user with graphical interface
6	Reboot

Modern distributions using systemd replace runlevels with **targets**.

For example:

SysV Runlevel	systemd Target
1	rescue.target
3	multi-user.target
5	graphical.target

Although the terminology differs, the underlying goal is the same: define the desired operating state of the system.

The Importance of PID 1

PID 1 has several unique responsibilities that no other process performs.

These include:

- Starting essential system services.
- Managing service dependencies.
- Reaping orphaned child processes.
- Coordinating system shutdown and reboot.
- Handling service failures.

Because of these responsibilities, PID 1 is special.

If a normal process crashes, only that application is affected.

If PID 1 exits unexpectedly, the operating system can no longer manage its processes correctly.

Historically, this often resulted in a kernel panic or an unusable system.

Engineering Insight

PID 1 is more than "the first process."

It acts as the **orchestrator** of the entire userspace environment.

Every service you rely on—networking, SSH, databases, logging, web servers—ultimately exists because PID 1 started or supervised it.

Understanding this role helps explain why issues affecting `init` or `systemd` can have system-wide consequences.

Common PID 1 Failures

Although failures at this stage are less common than storage-related issues, they can still occur.

Symptom	Likely Cause
No working init found	Missing or corrupted <code>/sbin/init</code>
Immediate reboot after mounting root	PID 1 crashed
Rescue shell instead of normal boot	Essential services failed or rescue target entered
system hangs after switching root	Initialization process could not continue

When troubleshooting, it is important to distinguish between:

- **The kernel failing to start PID 1**, and
- **PID 1 starting successfully but encountering problems later while launching services.**

The second scenario belongs to the next stage of the boot process.

Verifying PID 1

On a running system, confirming the identity of PID 1 is straightforward.

```
ps -p 1
```

Example output:

PID	TTY	TIME	CMD
1	?	00:00:05	systemd

or on legacy systems:

PID	TTY	TIME	CMD
1	?	00:00:01	init

You can also inspect the executable directly:

```
ls -l /proc/1/exe
```

This symbolic link reveals the actual program running as PID 1.

Summary

The execution of PID 1 marks the beginning of **userspace**.

Until this point, Linux has been concerned primarily with preparing hardware and mounting the operating system.

Now the focus shifts to bringing the operating system to life.

Whether implemented as SysV init, Upstart, or systemd, PID 1 serves as the parent of nearly every userspace process and coordinates the initialization of the entire system.

Without it, Linux cannot progress beyond a mounted root filesystem into a fully operational environment.

Revision #1

Created 4 July 2026 04:20:31 by Sulistio Priatmojo

Updated 4 July 2026 04:22:11 by Sulistio Priatmojo