

Lecture 10: User Space

Kernel to Userspace Transition

- The Kernel code is tightly controlled by Linus Torvolds. He decides* what is included in new releases.
- The Kernel code has to be efficient and stable. Kernel code runs as a single monolithic process.
- All the other code in the world that executes on Linux is in "User Space" under the control of the Kernel.



Linus Torvolds
Finnish Computer Scientist

*This is why he wrote git. Probably named that because he has to be a "ruthless git" when deciding what kernel changes to include.

The init program (system for example)

- Systemd is a collection of programs.
- These programs often run as daemons* (increasingly called services, that's what the "d" stands for)
- Daemons run in the background performing system tasks usually without administrator intervention.
- Systemd daemon programs are in `/sbin` (system binaries)
- Programs that admins and users run to interface with systemd are in `/bin` (general binaries).
- The majority of the systemd code is in `/lib/systemd`. (`/lib` is for library files)



Lennart Poettering[†]
German/Brazilian Computer
Scientist

*In Greek mythology a daemon was a spirit that served the gods. They were below the gods but above mere mortals. They often served as helpers and guardians of humans.

[†]Debate between Linus Torvalds and Lennart Poettering regarding the User Space/Kernel interface <https://www.youtube.com/watch?v=Nn-SGblUhi4>

SystemD paradigm

- Systemd handles the boot process after the Kernel has identified devices and filesystems.
- Systemd is relatively recent and aims to replace a lot of standard Linux tools (such as cron)*
- Systemd is “goal oriented” where each goal is defined as a *unit*.
- Most *units* run as daemons, they start on boot and run as long as the system is up.
- Each *unit* has a configuration file that defines how it works. These config files are in `/usr/lib/systemd/system/`

SystemD paradigm

- Each *unit* has a configuration file that defines how it works. These config files are in `/usr/lib/systemd/system/`

```
[matthew@moonshine ~]$ ls /usr/lib/systemd/system | head -n 10
```

```
arp-ethers.service
```

```
auditd.service
```

```
auth-rpcgss-module.service
```

```
autovt@.service
```

```
basic.target
```

```
basic.target.wants
```

```
blk-availability.service
```

```
blockdev@.target
```

```
bluetooth.target
```

```
boot-complete.target
```

```
[matthew@moonshine ~]$ cat /usr/lib/systemd/system/systemd-udev.service
```

```
# SPDX-License-Identifier: LGPL-2.1-or-later
```

```
#
```

```
# This file is part of systemd.
```

```
#
```

```
# systemd is free software; you can redistribute it and/or modify it
```

```
# under the terms of the GNU Lesser General Public License as published by
```

```
# the Free Software Foundation; either version 2.1 of the License, or
```

```
# (at your option) any later version.
```

```
[Unit]
```

```
Description=Rule-based Manager for Device Events and Files
```

```
Documentation=man:systemd-udev.service(8) man:udev(7)
```

```
DefaultDependencies=no
```

```
After=systemd-sysusers.service systemd-hwdb-update.service
```

```
Before=sysinit.target
```

```
ConditionPathIsReadWrite=/sys
```

```
[Service]
```

```
CapabilityBoundingSet=~CAP_SYS_TIME CAP_WAKE_ALARM
```

```
Delegate=pids
```

```
Type=notify
```

```
# Note that udev will reset the value internally for its workers
```

```
OOMScoreAdjust=-1000
```

```
Sockets=systemd-udev-control.socket systemd-udev-kernel.socket
```

```
Restart=always
```

Recall from the
devices lecture that
the udev daemon
handles device
events reported by
the Kernel.

```
[matthew@moonshine ~]$ cat /usr/lib/systemd/system/systemd-udev.service
```

```
# SPDX-License-Identifier: LGPL-2.1-or-later
#
# This file is part of systemd.
#
# systemd is free software; you can redistribute it and/or modify it
# under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation; either version 2.1 of the License, or
# (at your option) any later version.
```

[Unit]

```
Description=Rule-based Manager for Device Events and Files
Documentation=man:systemd-udev.service(8) man:udev(7)
DefaultDependencies=no
```

```
After=sysusers.service systemd-hwdb-update.service
```

```
Before=sysinit.target
```

```
ConditionPathIsReadWrite=/sys
```

[Service]

```
CapabilityBoundingSet=~CAP_SYS_TIME CAP_WAKE_ALARM
Delegate=pids
Type=notify
# Note that udev will reset the value internally for its workers
OOMScoreAdjust=-1000
Sockets=systemd-udev-control.socket systemd-udev-kernel.socket
Restart=always
```

Systemd units start as soon as they are ready, the order is constrained by the “before” and “after” keywords.

Unit Types

- Services

Manage Linux daemons

- Targets

Manage other units (starting groups of units for example)

- Sockets

Handles socket communications (see devices lecture)

- Mounts

For accessing filesystems

Systemd Daemon Example

- This example has three parts.
 1. A little python program that prints to *standard out* whatever it receives on *standard in* (recall *standard out* and *standard in* from the devices lecture)
 2. A *systemd* socket unit that opens a TCP network *port* (4444) and listens for data (recall TCP *ports* from the networking lecture)
 3. A *systemd* service unit that creates a new daemon process to handle connections to *port* 4444. A new daemon is created for each new connection so it can hold multiple conversations at once.
- We will also need to install a little program to easily write data to port 4444.
- SELinux is a security daemon. We will have to temporarily make it not care about port 4444.

Git Clone the Files

```
[matthew@moonshine ~]$ git clone https://github.com/gmfricke/echo daemon.git
```

```
Cloning into 'echo_daemon'...
```

```
remote: Enumerating objects: 6, done.
```

```
remote: Counting objects: 100% (6/6), done.
```

```
remote: Compressing objects: 100% (5/5), done.
```

```
remote: Total 6 (delta 0), reused 6 (delta 0), pack-reused 0
```

```
Receiving objects: 100% (6/6), done.
```

Install socat (socket concatenator – it’s like the “cat” command you have been using, but for network sockets)

```
[matthew@moonshine ~]$ sudo yum install socat
```

```
Last metadata expiration check: 1:45:44 ago on Sat 17 Feb 2024 09:56:45 PM CST.  
Dependencies resolved.
```

```
=====
```

Package	Architecture	Version	Repository	Size
Installing:				
socat	x86_64	1.7.4.1-5.el9	appstream	300 k

```
=====
```

```
Transaction Summary
```

```
=====
```

Install	1 Package
---------	-----------

```
Total download size: 300 k
```

```
Installed size: 1.1 M
```

```
Is this ok [y/N]: y
```

```
Downloading Packages:
```

socat-1.7.4.1-5.el9.x86_64.rpm	695 kB/s	300 kB	00:00
--------------------------------	----------	--------	-------

```
-----
```

Total	347 kB/s	300 kB	00:00
-------	----------	--------	-------

Only one person per server team needs to do this

Make SELinux “Permissive” so we can use the port

```
[matthew@moonshine ~]$ sestatus
```

```
SELinux status:          enabled
SELinuxfs mount:         /sys/fs/selinux
SELinux root directory:  /etc/selinux
Loaded policy name:       targeted
Current mode:             enforcing
Mode from config file:    enforcing
Policy MLS status:        enabled
Policy deny_unknown status: allowed
Memory protection checking: actual (secure)
Max kernel policy version: 33
```

Make SELinux “Permissive” so we can use the port

```
[matthew@moonshine ~]$ sudo setenforce Permissive
```

```
[matthew@moonshine ~]$ sestatus
```

```
SELinux status:                enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:              targeted
Current mode:                    permissive
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Memory protection checking:     actual (secure)
Max kernel policy version:      33
```

Read through the systemd and python code.

```
[matthew@moonshine echo_daemon]$ cd echo_daemon/
```

```
[matthew@moonshine echo_daemon]$ ls
```

```
echod@.service echod.socket echo.py README.md
```

Read through the systemd and python code.

```
[matthew@moonshine echo_daemon]$ cat echod.socket
```

```
# Create a Socket to Listen to
```

```
[Unit]
```

```
Description = Echo server
```

```
[Socket]
```

```
ListenStream = 4444
```

```
Accept = yes
```

```
[Install]
```

```
WantedBy = sockets.target
```

Socket units automatically look for a service unit with the same name + @ and activate that service. In this case echod@.service.

Read through the systemd and python code.

```
[matthew@moonshine echo_daemon]$ cat echod@.service
```

```
# echod@.service
```

```
[Unit]
```

```
Description=Echo server service
```

```
[Service]
```

```
User=root
```

```
ExecStart=/sbin/echo.py
```

```
StandardInput=socket
```



The @ means this is a “template service”. Template services take an argument – in this case it will be an identifier to uniquely label the conversation this service will handle. (More than one program might connect to port 4444 at a time).

Read through the systemd and python code.

```
[matthew@moonshine echo_daemon]$ cat echod@.service
```

```
# echod@.service
```

```
[Unit]
```

```
Description=Echo server service
```

```
[Service]
```

```
User=root
```

```
ExecStart=/sbin/echo.py
```

```
StandardInput=socket
```



Read through the systemd and python code.

```
[matthew@moonshine echo_daemon]$ cat echod@.service
```

```
# echod@.service
```

```
[Unit]
```

```
Description=Echo server service
```

```
[Service]
```

```
User=root
```

```
ExecStart=/sbin/echo.py
```

```
StandardInput=socket
```

This the path to our program that will handle the data and write a reply.

Python Program to Read from Standard in and write to standard out.

```
[matthew@moonshine echo_daemon]$ cat echo.py
```

```
#!/usr/bin/python
```

Python is a scripting language.

```
# Program that reads
```

```
import sys
```

The first line tells the system what program to use to execute the script.

```
# Incoming Message
```

```
message = sys.stdin.readline().strip()
```

```
# Print the message
```

```
sys.stdout.write( "Echo Server Received: " + message + "\n")
```

Guido van Rossum

Dutch Computer Scientist and
“benevolent dictator for life”
(BDFL)

- Guido wrote Python to be a system scripting language. It’s designed to make automating Linux tasks easy.
- It became so popular that now it is used for all sorts of tasks and has become a general purpose language.



Let's move these files into place (coordinate with your team mate)

```
[matthew@moonshine echo_daemon]$ sudo cp echo.py /sbin/
```

```
[matthew@moonshine echo_daemon]$ sudo cp echod@.service /usr/lib/systemd/system/
```

```
[matthew@moonshine echo_daemon]$ sudo cp echod.socket /usr/lib/systemd/system/
```

Now activate the echod.socket systemd unit

We are going to start the daemon that listens for data on socket 4444. It sits between the “Kernel god” and the “mortal user” programs.

```
[matthew@moonshine echo_daemon]$ sudo systemctl start echod.socket
```

We use the systemctl program to control system units.

Now check the status of the echod socket handler daemon.

```
[matthew@moonshine echo_daemon]$ sudo systemctl start echod.socket
```

```
[matthew@moonshine echo_daemon]$ systemctl status echod.socket
```

- echod.socket - Echo server

Loaded: loaded (/usr/lib/systemd/system/echod.socket; disabled; preset: disabled)

Active: active (listening) since Sun 2024-02-18 00:28:46 CST; 11s ago

Until: Sun 2024-02-18 00:28:46 CST; 11s ago

Listen: [::]:4444 (Stream)

Accepted: 0; Connected: 0;

Tasks: 0 (limit: 407887)

Memory: 8.0K

CPU: 595us

CGroup: /system.slice/echod.socket

```
[matthew@moonshine echo_daemon]$
```

Journalctl -u {unit name} displays the system unit log

```
[matthew@moonshine echo_daemon]$ sudo journalctl -u echod.socket  
Feb 18 00:28:46 moonshine systemd[1]: Listening on Echo server.
```

Any error messages will show up here

Journalctl -xe will show all messages from systemd

```
[matthew@moonshine echo_daemon]$ sudo journalctl -xe
```

```
Feb 18 00:37:58 moonshine sshd[191411]: Received disconnect from 111.230.20.37 port 48942:11: Bye Bye [preauth]
```

```
Feb 18 00:37:58 moonshine sshd[191411]: Disconnected from invalid user xmj 111.230.20.37 port 48942 [preauth]
```

```
Feb 18 00:38:22 moonshine sshd[191480]: Invalid user nichengzhuo from 186.117.143.206 port 35074
```

```
Feb 18 00:38:22 moonshine sshd[191480]: pam_unix(sshd:auth): check pass; user unknown
```

```
Feb 18 00:38:22 moonshine sshd[191480]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=186.117.143.206
```

```
Feb 18 00:38:24 moonshine sshd[191480]: Failed password for invalid user nichengzhuo from 186.117.143.206 port 35074 ssh2
```

```
Feb 18 00:38:25 moonshine sshd[191480]: Received disconnect from 186.117.143.206 port 35074:11: Bye Bye [preauth]
```

```
Feb 18 00:38:25 moonshine sshd[191480]: Disconnected from invalid user nichengzhuo 186.117.143.206 port 35074 [preauth]
```

```
Feb 18 00:37:22 moonshine sshd[191341]: Failed password for invalid user yaowz from 186.117.143.206
```

```
Feb 18 00:37:23 moonshine sshd[191341]: Received disconnect from 186.117.143.206 port 44992:11: Bye
```

```
Feb 18 00:37:23 moonshine sshd[191341]: Disconnected from invalid user yaowz 186.117.143.206 port 44
```

```
Feb 18 00:37:35 moonshine sudo[191137]: pam_unix(sudo:session): session closed for user root
```

```
Feb 18 00:37
```

Journalctl -f will display the log “live”

```
[matthew@moonshine echo_daemon]$ sudo journalctl -f
```

```
Feb 18 00:41:38 moonshine sshd[191876]: Failed password for root from 103.124.191.67 port 51242 ssh2
```

```
Feb 18 00:41:40 moonshine sshd[191876]: Connection closed by authenticating user root 103.124.191.67 port 51242 [preauth]
```

```
Feb 18 00:41:59 moonshine sshd[191941]: Invalid user forest from 111.230.20.37 port 53846
```

```
Feb 18 00:41:59 moonshine sshd[191941]: pam_unix(sshd:auth): check pass; user unknown
```

```
Feb 18 00:41:59 moonshine sshd[191941]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=111.230.20.37
```

```
Feb 18 00:42:01 moonshine sshd[191941]: Failed password for invalid user forest from 111.230.20.37 port 53846 ssh2
```

```
Feb 18 00:42:01 moonshine sshd[191941]: Received disconnect from 111.230.20.37 port 53846:11: Bye Bye [preauth]
```

```
Feb 18 00:42:01 moonshine sshd[191941]: Disconnected from invalid user forest 111.230.20.37 port 53846 [preauth]
```

```
Feb 18 00:42:15 moonshine sudo[191980]: matthew : TTY=pts/1 ; PWD=/home/matthew/echo_daemon ; USER=root ; COMMAND=/bin/journalctl -f
```

```
Feb 18 00:42:15 moonshine sudo[191980]: pam_unix(sudo:session): session opened for user root(uid=0) by matthew(uid=1000)
```

```
Feb 18 00:42:20 moonshine sshd[191736]: fatal: Timeout before authentication for 140.246.225.169 port 47158
```

Journalctl -f will display the log “live”

```
[matthew@moonshine echo_daemon]$ sudo journalctl -f
```

```
Feb 18 00:41:38 moonshine sshd[191876]: Failed password for root from 103.124.191.67 port 51242 ssh2
```

```
Feb 18 00:41:40 moonshine sshd[191876]: Connection closed by authenticating user root 103.124.191.67 port 51242 [preauth]
```

```
Feb 18 00:41:59 moonshine sshd[191941]: Invalid user forest from 111.230.20.37 port 53846
```

```
Feb 18 00:41:59 moonshine sshd[191941]: pam_unix(sshd:auth): check pass; user unknown
```

```
Feb 18 00:41:59 moonshine sshd[191941]: pam_unix(sshd:auth): authentication failure: logname= uid=0 euid=0 tty=ssh ruser= rhost=111.230.20.37
```

```
Feb 18 00:42:01 moonshine sshd[191941]: Failed password for invalid user forest from 111.230.20.37 port 53846 ssh2
```

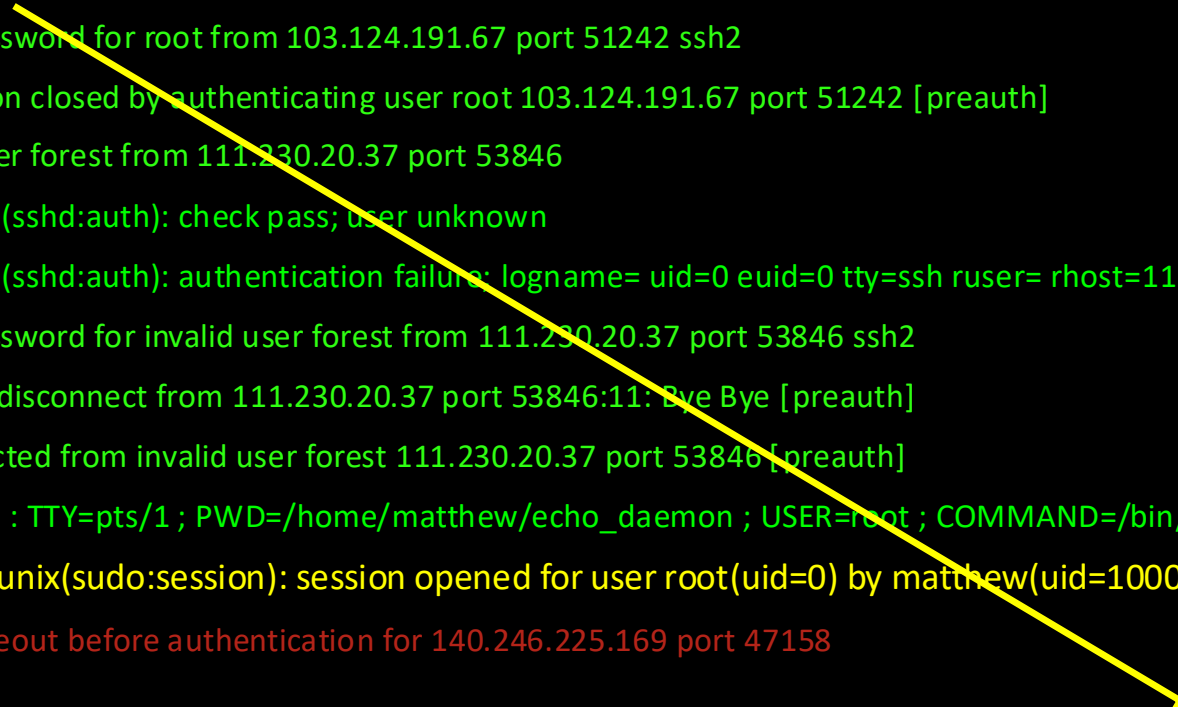
```
Feb 18 00:42:01 moonshine sshd[191941]: Received disconnect from 111.230.20.37 port 53846:11: Bye Bye [preauth]
```

```
Feb 18 00:42:01 moonshine sshd[191941]: Disconnected from invalid user forest 111.230.20.37 port 53846 [preauth]
```

```
Feb 18 00:42:15 moonshine sudo[191980]: matthew : TTY=pts/1 ; PWD=/home/matthew/echo_daemon ; USER=root ; COMMAND=/bin/journalctl -f
```

```
Feb 18 00:42:15 moonshine sudo[191980]: pam_unix(sudo:session): session opened for user root(uid=0) by matthew(uid=1000)
```

```
Feb 18 00:42:20 moonshine sshd[191736]: fatal: Timeout before authentication for 140.246.225.169 port 47158
```



You or your teammate open a terminal to monitor your echod@ service log

```
[matthew@moonshine ~]$ sudo journalctl -f -u "echod@*"
```

Write data to port 4444 with socat.

```
$ socat - TCP:moonshine:4444  
test  
Echo Server Received: test
```

```
Feb 17 23:18:39 moonshine systemd[1]: Started Echo  
server service (127.0.0.1:52724).  
Feb 17 23:18:42 moonshine systemd[1]: echo@25-  
127.0.0.1:4444-127.0.0.1:52724.service: Deactivated  
successfully.
```

Can you write to another team's echo server? (See HW3 Q10)

Clean up our insecure deamon

```
[matthew@moonshine echo_daemon]$ sudo systemctl stop echod.socket
```

```
[matthew@moonshine echo_daemon]$ systemctl status echod.socket
```

```
o echod.socket - Echo server
```

```
Loaded: loaded (/usr/lib/systemd/system/echod.socket; disabled; preset: disabled)
```

```
Active: inactive (dead)
```

```
Listen: [::]:4444 (Stream)
```

```
Accepted: 0; Connected: 0;
```

Reenable “Secure ”Linux

```
[matthew@moonshine echo_daemon]$ sudo setenforce Enforcing
```

```
[matthew@moonshine echo_daemon]$ sestatus
```

```
SELinux status:      enabled
```

```
SELinuxfs mount:     /sys/fs/selinux
```

```
SELinux root directory: /etc/selinux
```

```
Loaded policy name:   targeted
```

```
Current mode:         enforcing
```

```
Mode from config file: enforcing
```

```
Policy MLS status:    enabled
```

```
Policy deny_unknown status: allowed
```

```
Memory protection checking: actual (secure)
```

```
Max kernel policy version: 33
```

The Default Target

```
[matthew@moonshine echo_daemon]$ systemctl status default.target
```

- multi-user.target - Multi-User System

Loaded: loaded (/usr/lib/systemd/system/multi-user.target; indirect; preset: disable

Active: active since Tue 2024-02-06 20:21:25 CST; 1 week 4 days ago

Until: Tue 2024-02-06 20:21:25 CST; 1 week 4 days ago

Docs: man:systemd.special(7)

The Default Target – Defines a set of daemons to start on boot.

```
[matthew@moonshine echo_daemon]$ systemctl list-dependencies default.target
```

```
default.target
```

- └─auditd.service
- └─chronyd.service
- └─crond.service
- └─firewalld.service
- └─irqbalance.service
- └─kdump.service
- └─mdmonitor.service
- └─NetworkManager.service
- └─rpcbind.service
- └─rsyslog.service
- └─sshd.service
- └─sssd.service
- └─systemd-ask-password-wall.path
- └─systemd-logind.service
- └─systemd-update-utmp-runlevel.service
- └─systemd-user-sessions.service

List all units...

```
[matthew@moonshine echo_daemon]$ systemctl list-units | head -n 10
```

UNIT	LOAD	ACTIVE	SUB	DESCRIPTION
proc-sys-fs-binfmt_misc.automount	loaded	active	waiting	Arbitrary Executable File Formats File System Automount Point
sys-devices-pci0000:00-0000:00:01.0-0000:02:00.0-net-eno3.device	loaded	active	plugged	NetXtreme BCM5720 Gigabit Ethernet PCIe
sys-devices-pci0000:00-0000:00:01.0-0000:02:00.1-net-eno4.device	loaded	active	plugged	NetXtreme BCM5720 Gigabit Ethernet PCIe
sys-devices-pci0000:00-0000:00:01.1-0000:01:00.0-net-eno1.device	loaded	active	plugged	NetXtreme BCM5720 Gigabit Ethernet PCIe
sys-devices-pci0000:00-0000:00:01.1-0000:01:00.1-net-eno2.device	loaded	active	plugged	NetXtreme BCM5720 Gigabit Ethernet PCIe
sys-devices-pci0000:00-0000:00:02.2-0000:03:00.0-host1-target1:2:0-1:2:0:0-block-sda-sda1.device	loaded	active	plugged	PERC_H310 EFI\x20System\x20Partition
sys-devices-pci0000:00-0000:00:02.2-0000:03:00.0-host1-target1:2:0-1:2:0:0-block-sda-sda2.device	loaded	active	plugged	PERC_H310 2
sys-devices-pci0000:00-0000:00:02.2-0000:03:00.0-host1-target1:2:0-1:2:0:0-block-sda-sda3.device	loaded	active	plugged	PERC_H310 3
sys-devices-pci0000:00-0000:00:02.2-0000:03:00.0-host1-target1:2:0-1:2:0:0-block-sda.device	loaded	active	plugged	PERC_H310

System V

- System V is a very old way to startup User Space programs when the system boots up.
 - It is still used today even though the newer Systemd also starts up daemons.
- System V has “runlevels”. Different startup/shutdown scripts are run during each runlevel.
- There are 6 runlevels
 - 0 — Halt
 - 1 — Single-user text mode
 - 2 — Not used (user-definable)
 - 3 — Full multi-user text mode
 - 4 — Not used (user-definable)
 - 5 — Full multi-user graphical mode (with an X-based login screen)
 - 6 — Reboot

Who -r shows the current runlevel and when it started

```
[matthew@moonshine ~]$ who -r  
run-level 3 2024-02-06 20:21
```

```
[matthew@moonshine ~]$ ssh wheeler  
mfricke@wheeler:~ $ ls /etc/rc.d/  
init.d rc0.d rc1.d rc2.d rc3.d rc4.d rc5.d rc6.d rc.local
```

Wheeler's OS is old enough that it still has System V scripts. Each of the subdirectories correspond to a runlevel. For example, rc6.d scripts execute on shutdown.

When troubleshooting a system we often want to reboot into the Single User Text Mode runlevel. This mode has almost nothing extra running which makes debugging easier. (Like ‘safe mode’ in Windows – but even more barebones.) The option to boot to single-user mode is currently “rescue.target” in systemd

```
Red Hat Enterprise Linux Server (3.10.0-862.2.3.el7.x86_64) 7.5 (Maipo)
Red Hat Enterprise Linux Server (3.10.0-862.el7.x86_64) 7.5 (Maipo)
```

```
Use the ^ and v keys to change the selection.
Press 'e' to edit the selected item, or 'c' for a command prompt.
I
```