Revised: 3/24/2020

System Initialization

Michael C. Hackett

Computer Science Department

Community College of Philadelphia

Lecture Topics

- Boot Processes
  - Boot Loaders
  - GRUB Legacy
  - GRUB2
- Initialization
  - SysV
  - Systemd
- X Windows
  - Window Managers
  - Desktop Environments
  - Configuring X Windows
  - Assistive Technologies
- Localization
  - Time Localization
  - Format Localization

Boot Processes

- The system BIOS performs a series of tests and checks called Power On Self Test or POST when the computer first initializes.

- The BIOS typically checks the MBR (or GPT) of the first hard disk in the system
  - Depending on the order of boot devices set in the BIOS, it may first look at the MBR/GPT on other hard drives or devices like CDs, DVDs, or USB flash drives.

Boot Processes

- Computers with network interface cards that support **Preboot Execution Environment** (PXE) can have their BIOS configured to boot an operating system from an NFS, HTTP, or FTP server
  - *Netbooting*

Boot Processes

- A **boot loader** is a program that loads an operating system
- The MBR (or GPT) will contain the boot loader or it will point to the **active partition** that contains a boot loader in its first sector
- Systems with a UEFI BIOS will load a boot loader from the UEFI System Partition
  - The is only one UEFI System Partition per hard disk

Boot Processes

- UEFI BIOS allows for secure booting
  - The boot loader will have a digital signature
  - Checks that the boot loader was not modified by malware

- Regardless of the location of the boot loader (UEFI System Partition or MBR/GPT), the Linux kernel is always stored in /boot
  - **vmlinux-*kernel-version*** (uncompressed)
  - **vmlinuz-*kernel-version*** (compressed)

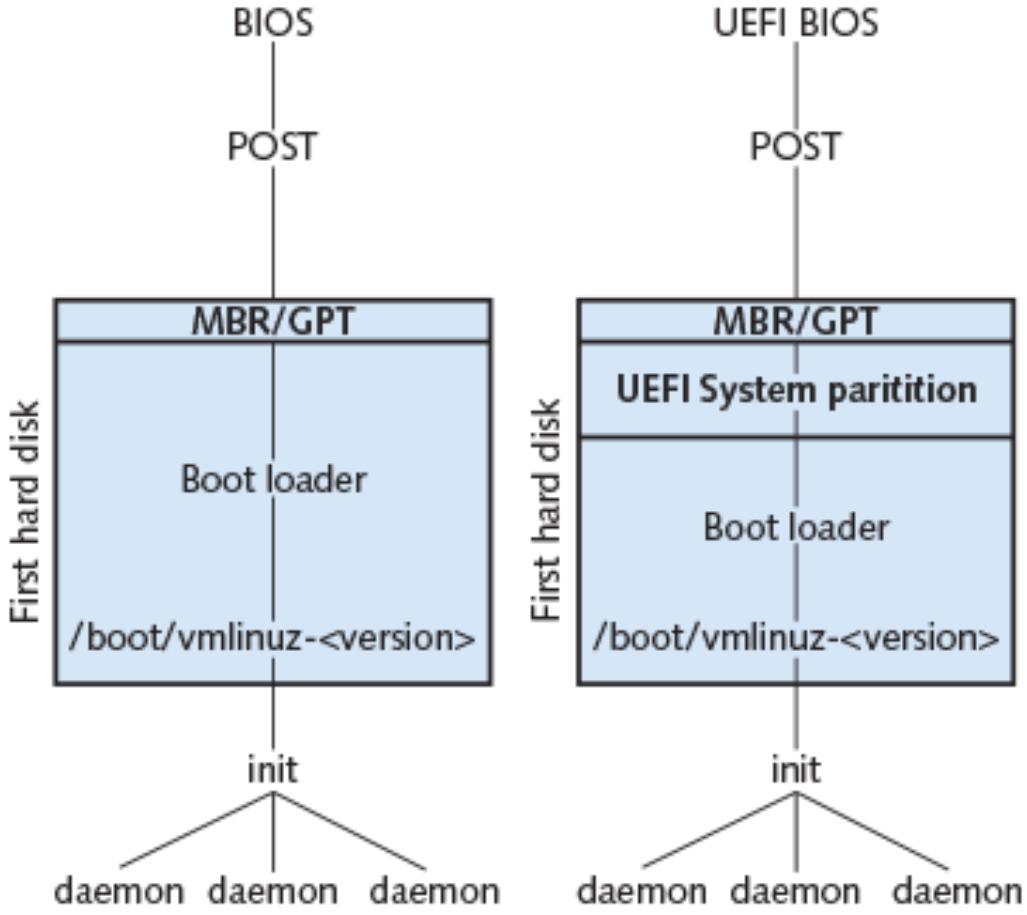
Boot Processes

- After the kernel is loaded into memory, it continues initializing the system by loading daemons

- A **daemon** (“day-mon” or “demon”) is a system process that carries out a specific task

- The **initialize daemon** (or **init daemon**) is the first process loaded
  - It starts all other daemons necessary to bring the system to a usable state

Boot Processes



Boot Loaders

- In addition to loading the Linux kernel into memory, other functions include:
  - Passing information to the kernel during startup
  - Booting other operating systems

- **Multi-booting** allows one boot loader to boot one of several operating systems

- The two most common boot loaders on Linux systems are GRUB and GRUB2
  - The older, original GRUB is now referred to as “GRUB Legacy”

GRUB Legacy

- **Gr**and **U**nified **B**ootloader
- Supports booting Windows, OSX, Linux, Unix operating systems
- GRUB Legacy is not used on modern systems
  - Older systems still in use might still use it

GRUB Legacy

- GRUB Legacy has three stages:
  - Stage 1
    - Resides in the MBR
    - Points to Stage 1.5
  - Stage 1.5
    - Resides in the 30KB of space after the MBR
    - Loads filesystem support before loading Stage 2
  - Stage 2
    - Resides in the /boot/grub directory
    - Performs the actual boot loader functions and displays the graphical boot loader screen

GRUB Legacy

- GRUB Legacy is configured through the **`/boot/grub/grub.conf`** configuration file
  - This configuration file is read by the Stage 2 boot loader
  - Is normally symbolically linked to **`/etc/grub.conf`**

- A damaged GRUB Legacy boot loader can be re-installed using the **`grub-install`** command
  - **`grub-install /dev/sda`** would re-install GRUB Legacy on sda

GRUB2

- Successor to GRUB/GRUB Legacy

- Supports booting Windows, OSX, Linux, Unix operating systems on drives that use an MBR or GPT
  - Also supports newer storage like NVMe

GRUB 2

- Standard BIOS
  - Stage 1
    - Resides in the MBR or GPT
    - Points to Stage 1.5
  - Stage 1.5
    - (MBR) Resides in the 30KB of space after the MBR
    - (GPT) Resides in a BIOS Boot partition
    - Loads filesystem support before loading Stage 2
  - Stage 2
    - Resides in the **`/boot/grub`** or **`/boot/grub2`** directory
    - Performs the actual boot loader functions and displays the graphical boot loader screen

GRUB 2

- UEFI BIOS
  - All stages are on the UEFI System Partition
  - UEFI System Partition is created during installation and is formatted with the FAT filesystem
  - Mounted to **`/boot/efi`** during the boot process

GRUB2

- Configuration file for GRUB2 is named either **`grub.conf`** or **`grub2.conf`**
- Standard BIOS
  - Located in **`/boot/grub/`** (or **`/boot/grub2`** on certain distributions)
- UEFI BIOS
  - Located in **`/boot/efi/EFI/`***`distribution`*

- **The GRUB2 configuration file is not meant to be edited**

GRUB2

- The configuration file for GRUB2 is built automatically based on the contents of **`/etc/default/grub`**
- After making the changes to **`/etc/default/grub`**, run the **`grub2-mkconfig`** command to rebuild the GRUB2 configuration file
  - Be sure to specify where to place the rebuilt configuration file by using the **`-o`** option
- Usage:

  `grub2-mkconfig -o /boot/grub/grub.conf`

GRUB Legacy

- A damaged GRUB Legacy boot loader can be re-installed using the **`grub2-install`** command
  - **`grub2-install /dev/sda`** would re-install GRUB2 on sda

Initialization

- After the kernel is loaded into memory by the boot loader, the init daemon begins system initialization to bring the system to a usable state

- Older Linux systems used the UNIX **SysV** (“System Five”) standard for system initialization
- More modern Linux systems use **Systemd** for system initialization
  - Systemd is completely compatible with SysV

Initialization

- Some Linux daemons have not been revised to work with Systemd

- Some distributions that have adopted Systemd will still use SysV processes to initialize such daemons

SysV

- Modern systems that use SysV will use either the traditional SysV initialization process or a more recent version of SysV called **upstart**

- In both, the init daemon uses scripts to start other daemons to bring the system to a usable state
  - The init daemon is responsible for starting and stopping daemons during the initialization process
  - It is also responsible for stopping daemons when the system is halted or rebooted

SysV

- The init daemon categorizes the system into **runlevels** (also called **initstates**) which define what daemons are started

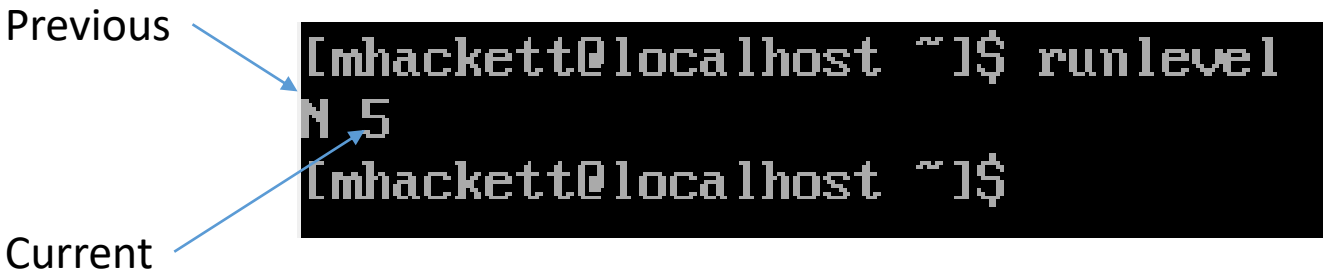
- A Linux system has seven standard runlevels

SysV

Runlevel	Name	Description
0	Halt	No daemons active and ready to be powered off
1	Single-User Mode	Only enough daemons for root to log in and perform maintenance tasks
2	Multi-User Mode	Most daemons are running and multiple users can log in; Basic networking services are started
3	Extended Multi-User Mode	Same as runlevel 2, but with extra network services started
4	Not used	No official use; Customizable runlevel
5	Graphical Mode	Same as runlevel 3, but with a graphical login screen; Typically the default runlevel
6	Reboot	Reboots the system

SysV

- The **`runlevel`** command is used to view the current (and previous) runlevel
  - An N would indicate none



SysV

- The **`init`** command is used to change the current runlevel
- Usage:
  `init` *`runlevel`*
- Example:
  `init 1`
  Would switch the system to runlevel 1

- Requires root/superuser privileges to manually switch runlevels

SysV

- The init daemon will enter the default runlevel specified in the **`/etc/inittab`** file

- While **`/etc/inittab`** used to contain the entire configuration for the init daemon, today it only contains the statement to configure the default run level.
  - The file is not used in the Systemd initialization process

SysV

- The init daemon executes scripts called **runtime configuration scripts** (or “rc scripts”) to start and stop other daemons when entering a runlevel

- The init daemon finds the rc scripts in the **`/etc/rc`***X*`.d` directory
  - *X* indicates the runlevel; **`/etc/rc5.d`** would contain the rc scripts for runlevel 5

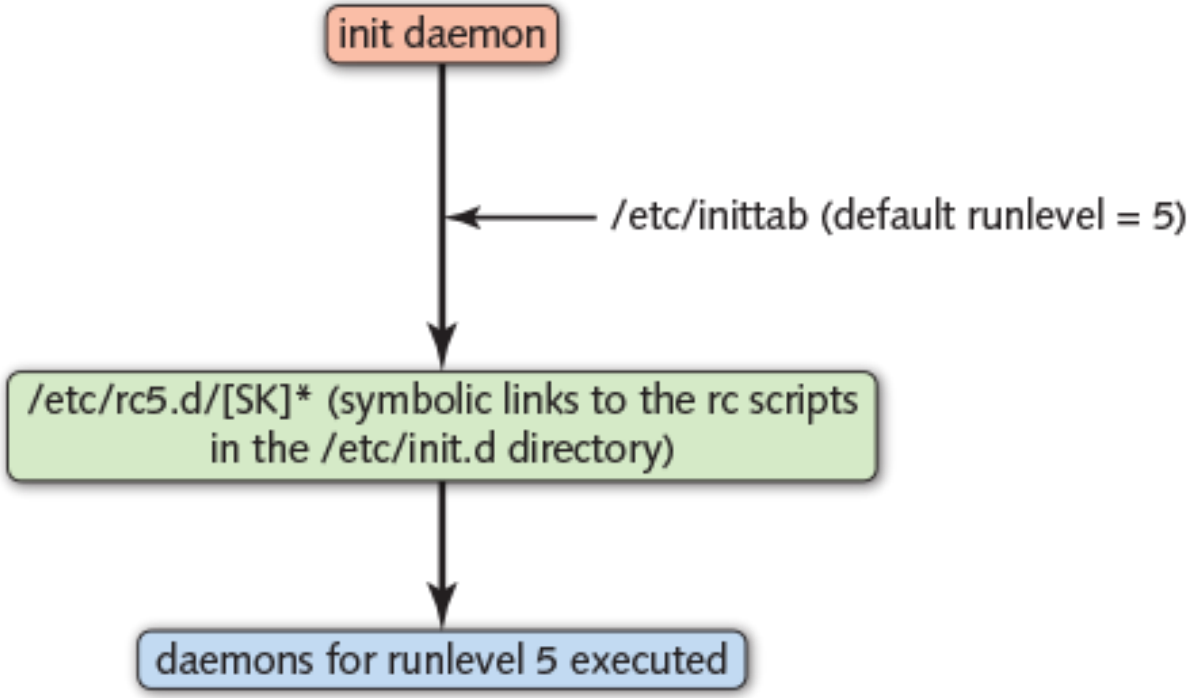
SysV

- Each rc script filename begins with an S or K
  - Scripts that begin with an S are daemons to be **s**tarted when entering this runlevel
  - Scripts that begin with an K are daemons to be **k**illed/stopped when entering this runlevel

- Each rc script in the rcX.d directories are symlinks to rc scripts contained in the /etc/init.d directory
  - This allows multiple rcX.d directories to utilize the same scripts

SysV

SysV Initialization Process



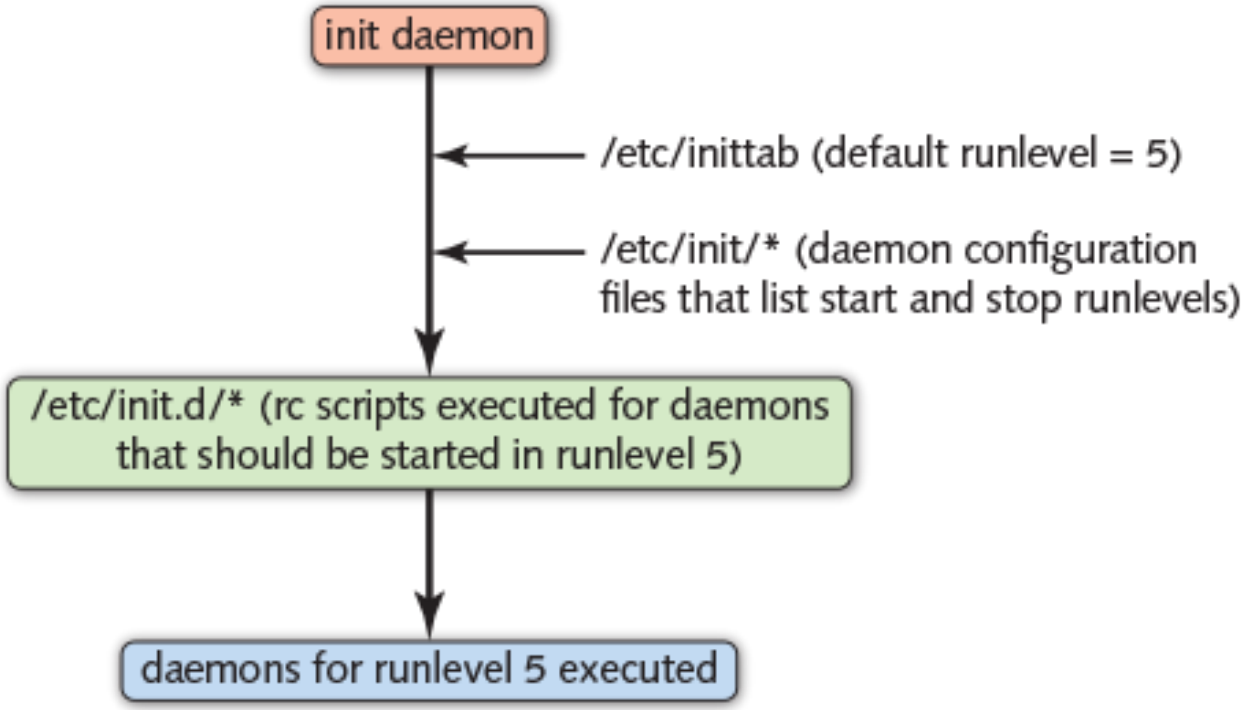
SysV

- The rcX.d directories are not used in the upstart initialization process

- Instead, it directly executes the rc scripts in **`/etc/init.d`** based on information contained in configuration files stored in the **`/etc/init`** directory

SysV

Upstart Initialization Process



SysV

- After system startup, daemons can be started, stopped, and restarted by directly executing their script in **`/etc/init.d`** and providing a **`start`**, **`stop`**, or **`restart`** argument
- To start a daemon:
  `/etc/init.d/`*`script`* **`start`**
- To stop a daemon:
  `/etc/init.d/`*`script`* **`stop`**
- To restart a daemon
  `/etc/init.d/`*`script`* **`restart`**

SysV

- Alternatively, the **`service`** command can be used to start, stop, or restart a daemon in **`/etc/init.d`**
- To start a daemon:

  `service` *`script`* **`start`**
- To stop a daemon:

  `service` *`script`* **`stop`**
- To restart a daemon

  `service` *`script`* **`restart`**

SysV

- If a daemon's configuration is changed, the daemon will usually need to be restarted
- Some daemons can be reloaded, where it can update its configuration without restarting
- A daemon can be reloaded using either:

  `/etc/init.d/`*`script`* **`reload`**

  or

  `service` *`script`* **`reload`**

SysV

- The status of a daemon can be viewed using either:

  /etc/init.d/*script* **status**

  or

  service *script* **status**

SysV

- In the upstart init system, the following commands are available for starting, stopping, restarting, reloading, and viewing the status of a daemon:
- To start a daemon:
  `start` *`script`*
- To stop a daemon:
  `stop` *`script`*
- To restart a daemon
  `restart` *`script`*
- To reload a daemon
  `reload` *`script`*
- To view the status of a daemon
  `status` *`script`*

SysV

- SysV uses the **`chkconfig`** command to easily modify whether or not a daemon is to be started or stopped at a particular runlevel

- Some distributions use the **`update-rc.d`** command in place of **`chkconfig`**

Systemd

- Like SysV, Systemd:
  - Starts daemons during initialization
  - Can start and stop daemons after initialiation

- Unlike SysV, Systemd can start, stop, and configure other components

Systemd

- In Systemd terminology…
- Each operating system component is called a **unit**
- Daemons are called **service units**
- Runlevels are called **target units** (or simply **targets**)

Systemd

SysV Runlevel	Equivalent Systemd Target	Alternative Systemd Targets*
0	**poweroff.target**	**runlevel0.target**
1	**rescue.target**	**runlevel1.target**
2	**multi-user.target**	**runlevel2.target** **runlevel3.target** **runlevel4.target**
3		
4		
5	**graphical.target**	**runlevel5.target**
6	**reboot.target**	**runlevel6.target**

* Some distributions use these target names

Systemd

- graphical.target is the default target when a GUI environment is installed

- The default target is specified as a symbolic link:

  `/etc/systemd/system/default.target`

  - Links to a .target file in **`/lib/systemd/system`**

Systemd

- .service files in the /lib/systemd/system directory comprise most of Systemd's rc scripts

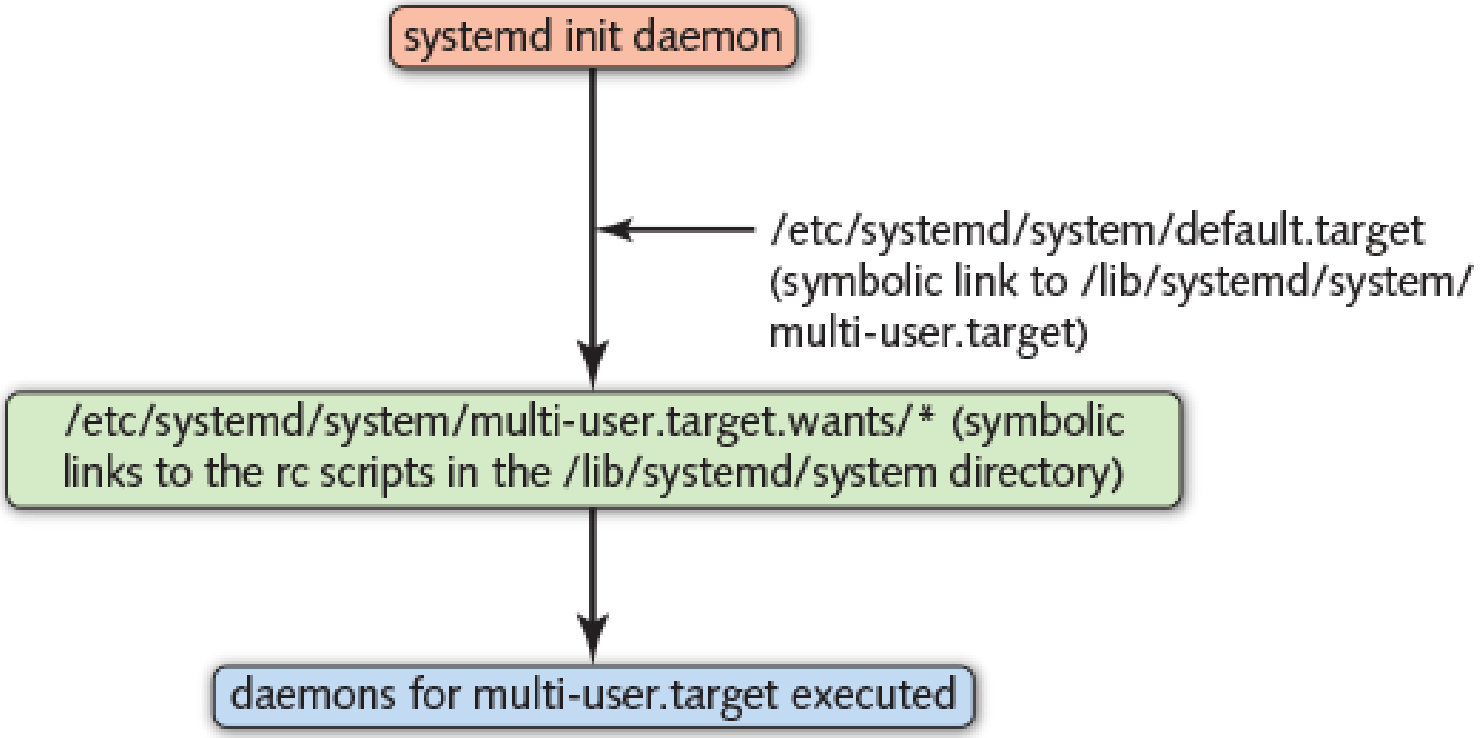
- To ensure a service unit is started when entering a target, a symbolic link is created in:

  `/etc/systemd/system/`*`X`***`.target.wants/`***`daemon`***`.service`**

  - *X* is the name of the target ("graphical", "multi-user", etc.)
  - *daemon* is the name of the service

Systemd

Systemd Initialization Process



Systemd

- Service units are controlled with the **`systemctl`** command
- Starting a service:
  - **`systemctl start`** *daemon*`.service`
- Stopping a service:
  - **`systemctl stop`** *daemon*`.service`
- Restarting a service:
  - **`systemctl restart`** *daemon*`.service`
- Display the current status of a service:
  - **`systemctl status`** *daemon*`.service`

Systemd

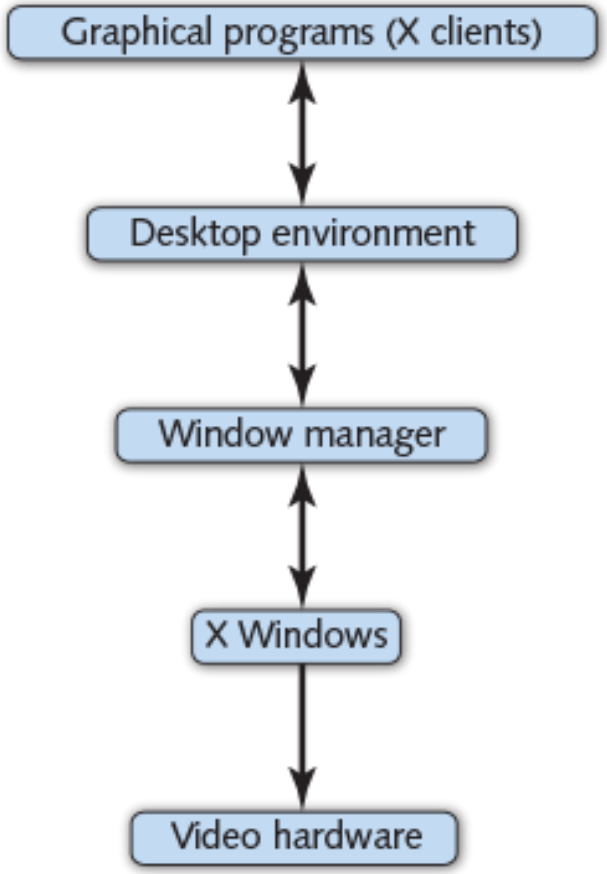
- Reloading a service after a configuration change:
  - **`systemctl reload`** *`daemon`***`.service`**
- Enable a service to start in the default target:
  - **`systemctl enable`** *`daemon`***`.service`**
- Disable a service from starting in the default target:
  - **`systemctl disable`** *`daemon`***`.service`**
- Prevent a service from being started:
  - **`systemctl mask`** *`daemon`***`.service`**

Systemd

- Create/edit environment variables that are loaded when a service is started:
  - **`systemctl edit`** *daemon*`.service`

- Change to a different runlevel:
  - **`systemctl isolate`** *X*`.target`

Linux GUIs


Graphical programs (X clients)
Desktop environment
Window manager
X Windows
Video hardware

X Windows

- **X Windows** is at the core of a Linux GUI
  - It interfaces with the system's video hardware to draw graphics in windows on a terminal screen

- Programs with graphical user interfaces are called **X clients**

- Since X Windows can send graphics to an X client on a different computer, X Windows sometimes called **X server**

X Windows

- X Windows was released in 1985
  - Many Linux distributions used XFree86, the open source version of X Windows
  - Since 2004, **X.org** is the common X Windows implementation on Linux Systems and is maintained by the X.org Foundation

- **Wayland**, a new X server, is intended to replace X.org
  - Still in development; Newer distributions use Wayland as the default X server

Window Managers

- A **Window Manager** provides the appearance (*look and feel*) of windows drawn by X Windows.
  - Windows Managers compatible with Wayland are called **Wayland Compositors**

- Window Managers control things like:
  - The dimensions of windows drawn on a screen
  - The colors of windows drawn on a screen
  - The ability to move, minimize, maximize, and resize windows

Window Managers

- Some common Window Managers are
  - Compiz – Highly customizable with 3D graphics effects
  - kwin – KDE's Window Manager
  - metacity – Window Manager used by older versions of GNOME
  - mutter – Windows Manager used by the latest version of GNOME
  - lxde – A lightweight Window Manager for low-power systems
  - twm – An older and basic Window Manager

- Window Managers can be used along or used with a desktop environment

Desktop Environments

- **Desktop Environments** are sets of GUI tools (window managers and graphical programs) that are bundled together and distributed for Linux systems

- The two most common Desktop Environments are **KDE** and **GNOME**
  - KDE uses kwin (Window Manager) and the Qt toolkit for graphical applications
  - GNOME uses mutter (Window Manager) and the GTK+ toolkit for graphical applications

- XFCE is a lightweight desktop environment that uses few system resources

Desktop Environments

- X Windows is only started by default in runlevel 5 (graphical.target)

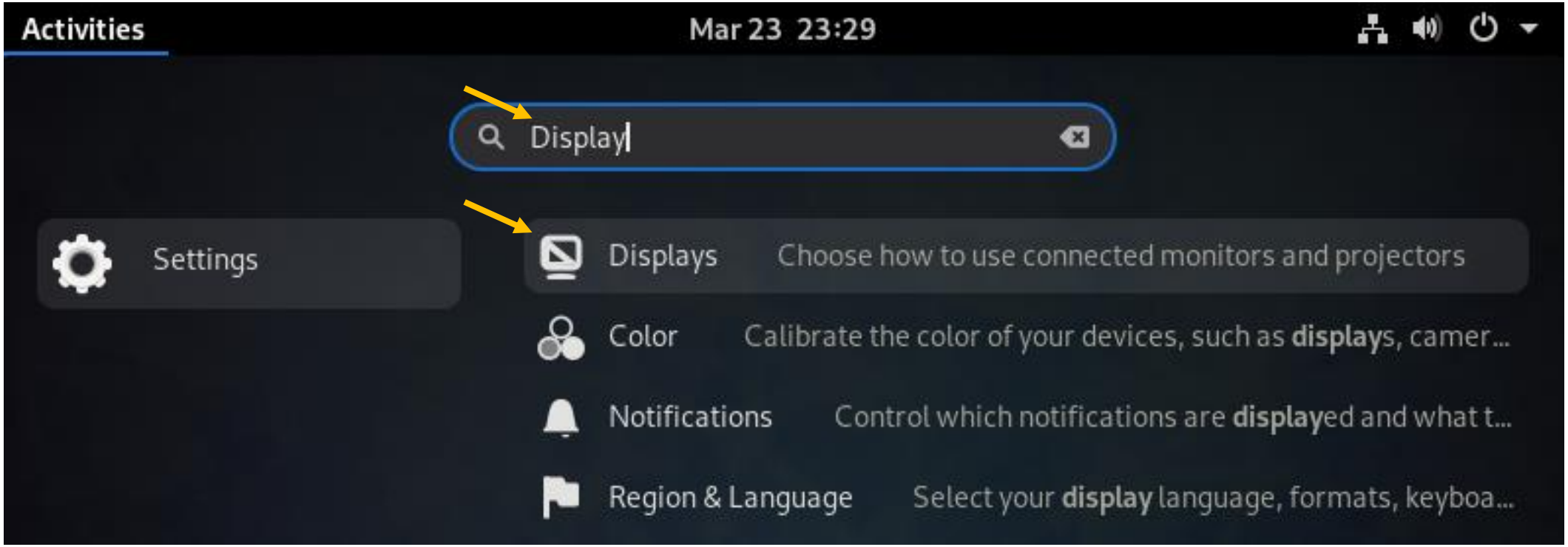
- The **`startx`** command is used to start X Windows at the terminal within a different runlevel (or to restart the graphical environment if it crashes).

Configuring X Windows

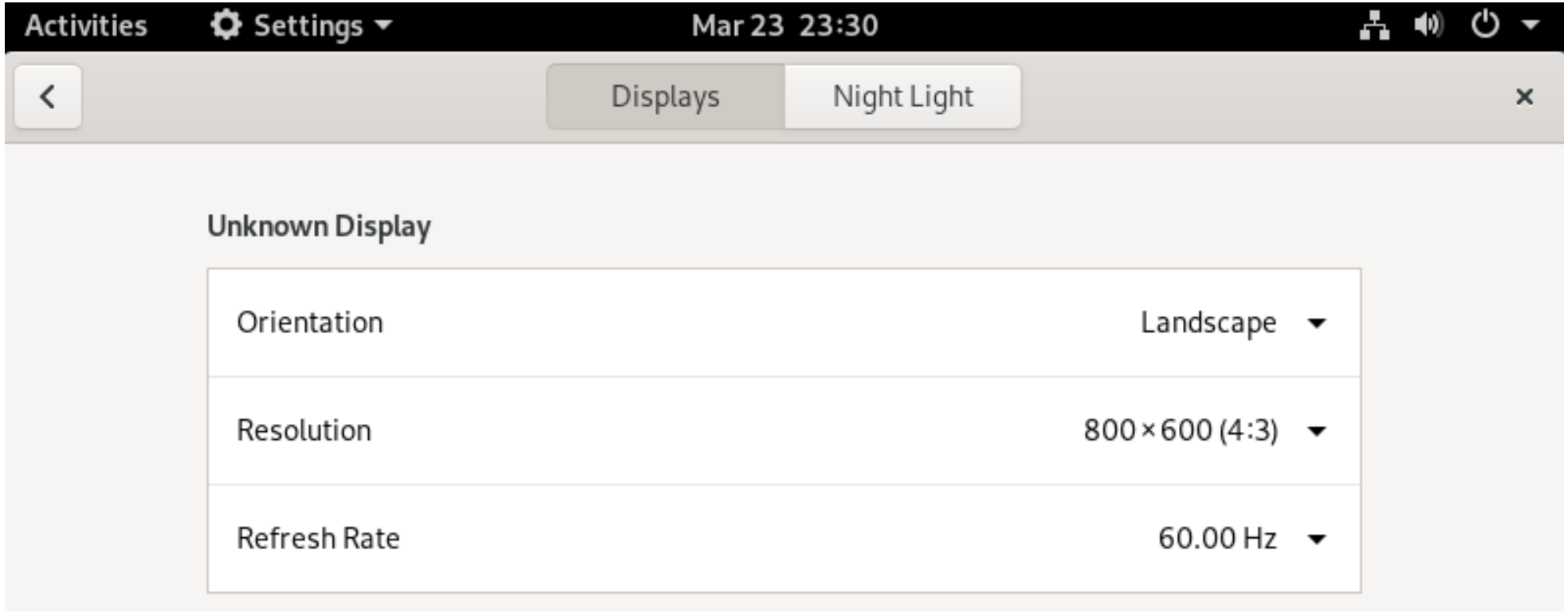
- X Windows needs information about to the mouse, monitor, and video card to work correctly
  - This is usually detected automatically by related kernel modules when the system is booted

- X Windows stores its configuration in /etc/X11/xorg.conf and in files within /etc/X11/xorg.conf.d/
- X Windows can be configured by modifying those files or using a graphical utility in the desktop environment

Configuring X Windows



Configuring X Windows



Assistive Technologies

- **Assistive Technologies** are tools that make a system more accessible for users with disabilities.

- This includes tools like
  - High Contrast – Alternate colors for those with low vision
  - Large Text – Increases size of text for those with low vision
  - Cursor Size – Increases size of the cursor for those with low vision
  - Zoom/Magnification – Magnifies parts of the screen for those with low vision
  - Screen Reader – Narrates the text on the screen in the active window

Assistive Technologies

- (Continued)
  - Sound Keys – Beep when Num Lock or Caps Lock is pressed
  - Visual Alerts – Pop-up messages/alerts in place of beeps or sounds
  - Screen Keyboard – On-screen keyboard that can be used with a mouse
  - Repeat Keys – Simulates multiple key presses when a single key is held down
  - Cursor Blinking – Adds blinking cursors to text fields

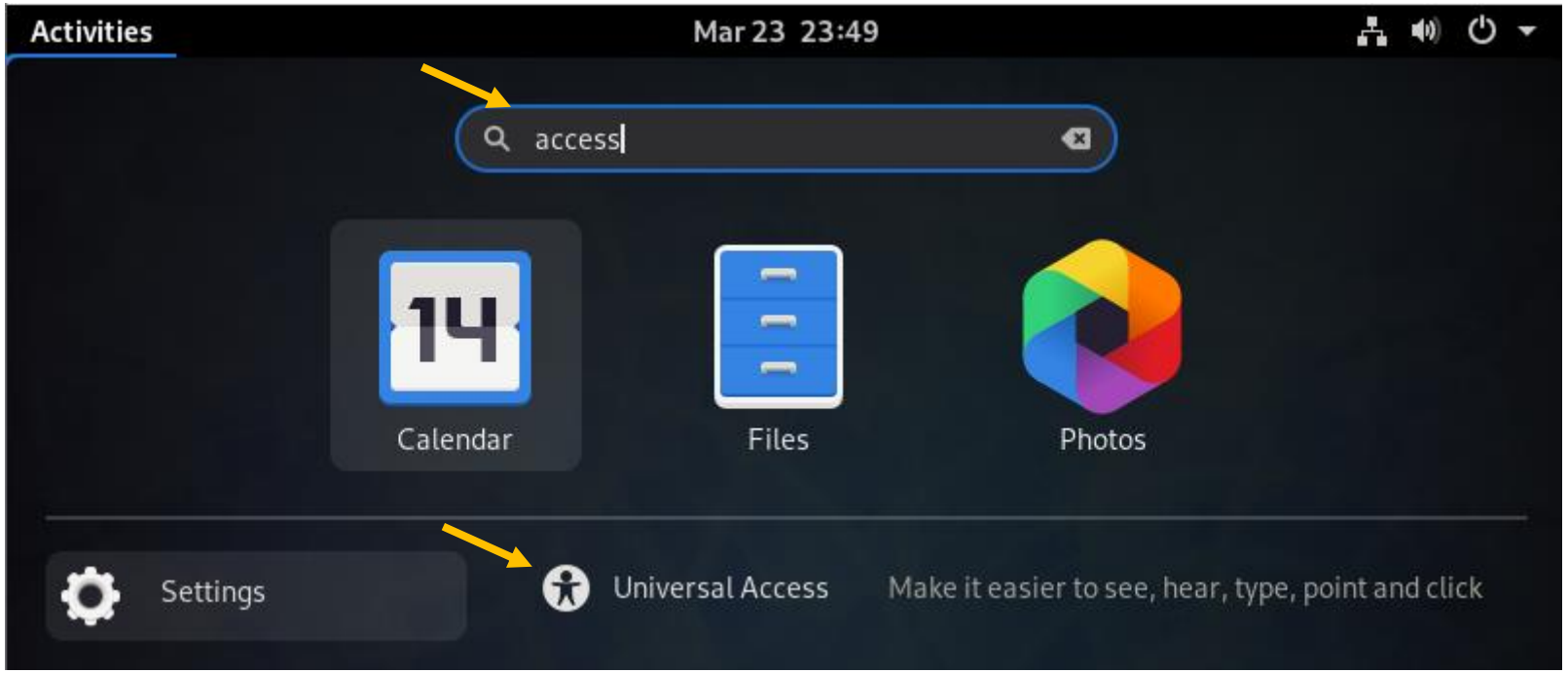
Assistive Technologies

- Assistive technologies for the keyboard:
  - Sticky Keys – Simulate key presses when two keys are pressed in sequence
  - Slow Keys – Adds a delay following each key press
  - Bounce Keys – Ignores fast duplicate key presses

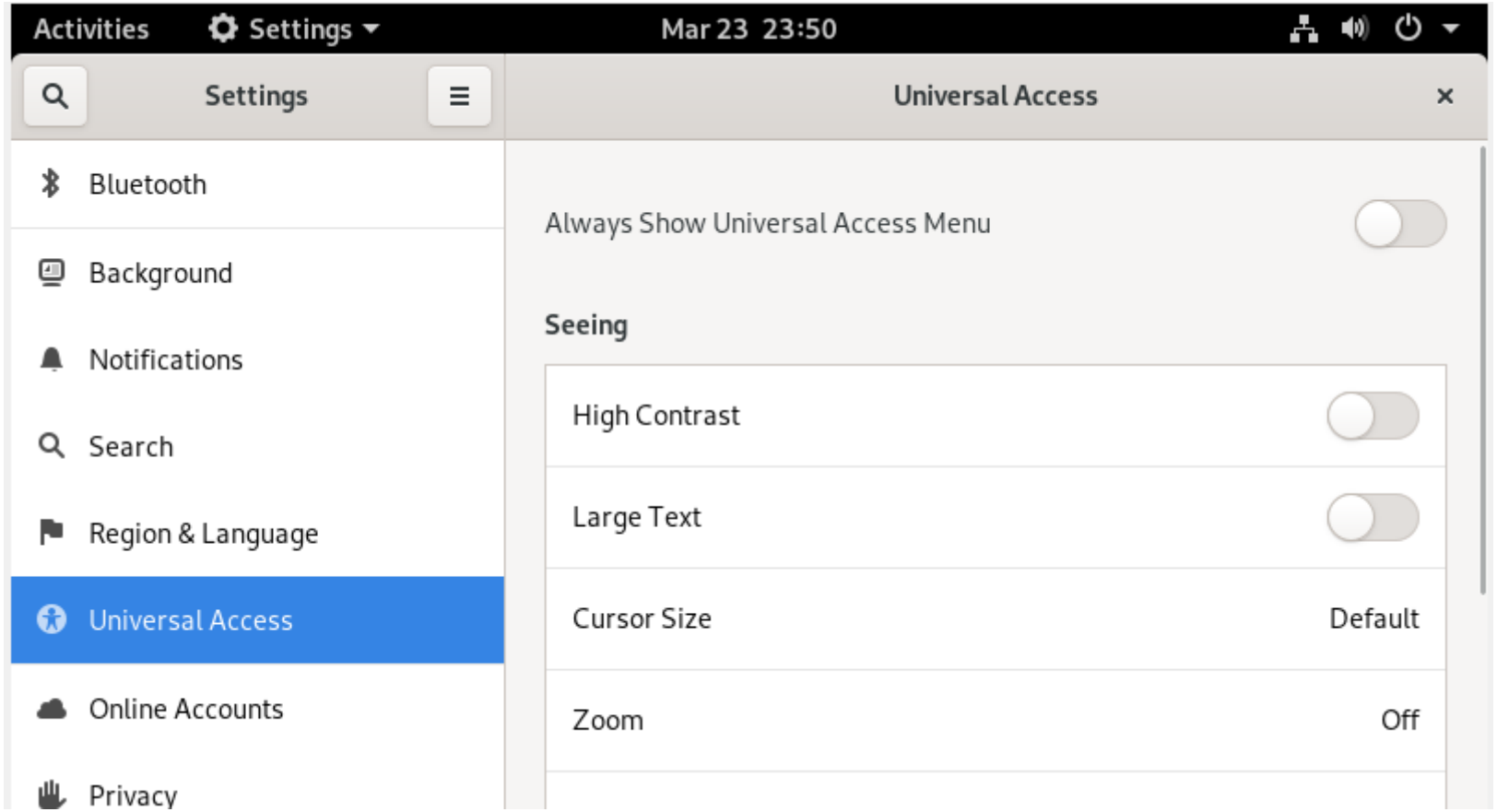
- Assistive technologies for the mouse:
  - Mouse Keys – Controls the mouse with the arrow keys on the keyboard
  - Click Assist – Simulates a right click when holding down the left mouse button

Assistive Technologies

- Assistive technologies can be configured in the desktop environment's Accessibility settings:



Assistive Technologies



Localization

- **Localization** is the system settings related to where, geographically, the system is.

- A system in New York, USA will have different settings from a system in Tokyo, Japan.

- Or perhaps, a system in Paris, France needs to be localized as if it were in Beijing, China

Time Localization

- Linux systems store time as the number of seconds since January 1, 1970
  - The UNIX Epoch
  - This time is referred to as **epoch time**

- Can be viewed in the terminal with the command **`date +%s`**

Time Localization

- A Linux systems obtains its current time from the system BIOS's clock

- The **`hwclock`** (**h**ard**w**are **clock**) command will display the system BIOS's time
  - This command can also be used to set the system BIOS's clock

- The system's date and time can also be set using the **`date`** command with the **`-s`** option.
- Example: **`date -s "1 JUL 2021 14:35:00"`**

Time Localization

- Time zone settings play a factor in the system's date and time

- The system stores time zone information in /etc/localtime
  - This specifies how to calculate the correct time for that time zone, based on the current epoch time
- /etc/localtime is normally a symlink to a file in /usr/share/zoneinfo
  - /usr/share/zoneinfo contains all files related to time zones

```
[root@localhost ~]# ls -l /etc/localtime
lrwxrwxrwx. 1 root root 38 Nov 30 13:57 /etc/localtime -> ../usr/share/zoneinfo/America/New_York
[root@localhost ~]#
```

Time Localization

- Linking a different file from /usr/share/zoneinfo to /etc/localtime will change the system's time zone
  - Some distributions also have a /etc/timezone file that identifies the subfolder/file in /usr/share/zoneinfo

- A utility to change time zones is the **`tzselect`** command
  - This will start a program that prompts you to choose from a list of regions and locations
  - This is useful if you don't know what time zone files are available for use

Time Localization

- Another tool to view and set the system time and its time zone is the **`timedatectl`** command.

```
timedatectl set-time "2022-08-05 16:30:00"
```

```
timedatectl set-timezone 'America/New York'
```

Time Localization

```
[root@localhost ~]# timedatectl
               Local time: Tue 2020-03-24 00:13:52 EDT
           Universal time: Tue 2020-03-24 04:13:52 UTC
                 RTC time: Tue 2020-03-24 04:13:51
                Time zone: America/New_York (EDT, -0400)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
[root@localhost ~]#
```

Format Localization

- Different regions may have different formats for text, character sets, and keyboard layouts
- A Linux system's **locale** is the language and character set the system uses
- When the kernel is loaded, the locale may be set by the LANG option in the GRUB2 configuration

Format Localization

- The locale may also be set by a configuration file
  - Ubuntu: /etc/default/locale
  - Fedora: /etc/locale.conf

```
[root@localhost ~]# cat /etc/locale.conf
LANG="en_US.UTF-8"
[root@localhost ~]#
```

Format Localization

- The **`locale`** command will display the system's locale variables

```
[root@localhost ~]# locale
LANG=en_US.UTF-8
LC_CTYPE="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"
LC_TIME="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"
LC_PAPER="en_US.UTF-8"
LC_NAME="en_US.UTF-8"
LC_ADDRESS="en_US.UTF-8"
LC_TELEPHONE="en_US.UTF-8"
LC_MEASUREMENT="en_US.UTF-8"
LC_IDENTIFICATION="en_US.UTF-8"
LC_ALL=
[root@localhost ~]# _
```

Format Localization

- The **`localectl`** command can be used to view or change the system's locale
- Display the current locale

```
localectl
```

- List available locales

```
localectl list-locales
```

- Change the locale

```
localectl set-locale LANG=en_US.UTF-8
```

Format Localization

- The **`iconv`** command can be used to convert a file in one character set to another
- Usage:

  `iconv -f` *`from`* **`-t`** *`to file`*

- Where
  - *from* is the original encoding/character set
  - *to* is the desired encoding/character set
  - *file* is the file to convert