210 Systems Programming
Introduction to Linux

Fall 2005

Week 1

Overview

- What is an Operating System?
- The Linux OS
- Why do we learn Linux and C?
- Alternative ways to access a Linux system, i.e., a `bash` terminal
- File system navigation and manipulation commands

Learning Outcomes for Today

- Learn about the components of an Operating System
- Explore the Linux Kernel
- Understand the relationship between Linux and C
- Learn about ssh/scp, Virtual Machines, WSL, Dual Boot
- Learn and use your first Linux commands to create files, navigate the file system effectively

What is an OS?

- An Operating System is a program that sits between the hardware and the application programs.
- It manages storage, processes, network, and provides an interface to programs or users.
- An OS is a program (or a collection of programs) just like other programs you write. The difference is that it is the first program that is run when you boot your machine.

The Functions of a Modern Operating System

- Multitasking – allow multiple programs to run simultaneously in the same computer (much more than the number of processors or cores).
- Multiuser – allow multiple users to use simultaneously in the same computer.
- File system – manage multiple disks, allow storage of files in a secure way
- Networking – provide connection to/from other computers
- GUI/Window management – provide a graphical interface for users
- Utility programs/standard libraries – provide a suite of tools and programs for users or application developers

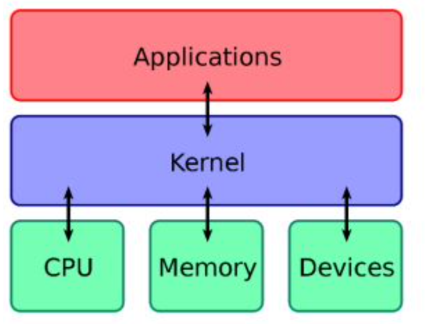
Discussion

- What are some Operating Systems you use?
- What do you use an Operating System for?
- Pros/cons?
    - from a user perspective
    - from a developer perspective

History of Linux

- In 1969, Ken Thompson and Dennis Ritchie of AT&T Bell Laboratories created UNIX (UNICS) - UNiplexed Information Computing System
- Several iterations and versions were developed between 1970-1990. Most of these were commercial software releases and there for only usable by large universities and corporations.
- In 1983 Richard Stallman created the GNU General Public License - setting the stage for shareware versions of software.
- In 1991 Linus Torvalds created a new monolithic kernel under the GNU license titled Linux.

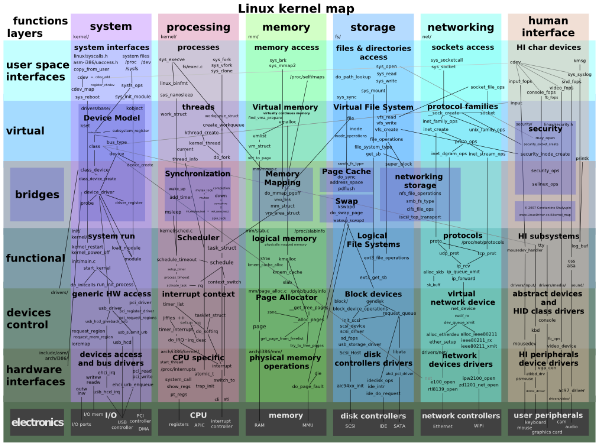
Kernel



- A kernel is simply the layer between the applications and the hardware
- "Monolithic" refers to the kernel being one program (as opposed to a microkernel)

The Linux Kernel

The Kernel



Why Linux?

The top three reasons Linux is running the world's supercomputers

- Linux is Open Source
- Linux offers security and stability
- Linux is flexible and scalable

Linux is full of small, easily used building blocks for common tasks, and has easy ways to connect things to make a bigger application from little pieces.

Productivity rises because you often don't need to build new code – you can just use these existing standard programs in flexible ways.

Why C?

- Linux lets you design applications that correspond closely to the hardware. But then we need a programming language that lets us talk directly to the operating system and the hardware.
- In today's world performance of your program matters especially on the cloud, because compute time means $$$$.
- Although Python and Java provides productivity benefits, they are inefficient. C/C++ programs are compiled and optimized for efficiency.

Unix and C

- At the time Unix was developed, other OSs were written in Assembly – porting to a different CPU was very difficult
- The C programming language was designed from the beginning to be a High-Level Assembly Language. This means that in one side it contained the high-level programming structures such as if/for/while, typed variables, and functions but on the other side it had memory pointers and arrays that allowed manipulating memory locations and their content directly.

Linux and C

In this class, we are going to learn how to write efficient C programs that can

- manage memory effectively
- make system calls using C libraries to manipulate files, processes
- execute tasks in parallel
- communicate with other programs

C Programming

You will learn many tools in this class to help you create efficient, bug free C programs.

- The gcc C compiler
- make and Makefiles
- gdb
- Valgrind
- git

SSH (Secure Socket Shell)

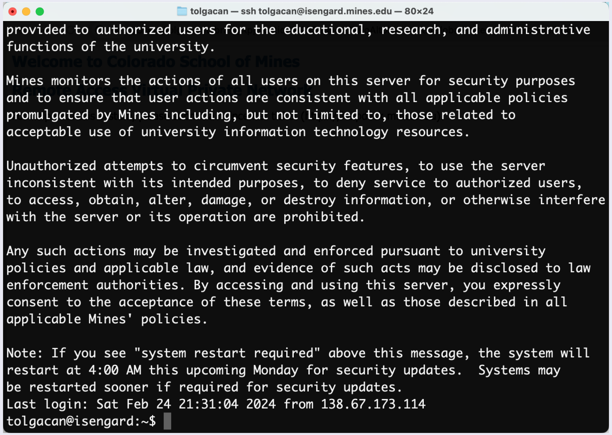
- SSH, also known as Secure Shell or Secure Socket Shell, is a network protocol that gives users, particularly system administrators, a secure way to access a computer over an unsecured network.
- SSH provides strong password authentication and public key authentication, as well as encrypted data communications between two computers connecting over an open network, such as the internet.

SSH Client and Server

- An SSH client is a program that allows establishing secure and authenticated SSH connections to SSH servers. ▸ Link
- SSH client software is available in many desktop operating systems such as Windows and MacOS.
- `sshd` is a process running on a server and listens for incoming connection requests and responds to them. ▸ Link

Let's connect to isengard

```
ssh <username>@isengard.mines.edu
```



tolgacan — ssh tolgacan@isengard.mines.edu — 80×24

```
provided to authorized users for the educational, research, and administrative
functions of the university.

Mines monitors the actions of all users on this server for security purposes
and to ensure that user actions are consistent with all applicable policies
promulgated by Mines including, but not limited to, those related to
acceptable use of university information technology resources.

Unauthorized attempts to circumvent security features, to use the server
inconsistent with its intended purposes, to deny service to authorized users,
to access, obtain, alter, damage, or destroy information, or otherwise interfere
with the server or its operation are prohibited.

Any such actions may be investigated and enforced pursuant to university
policies and applicable law, and evidence of such acts may be disclosed to law
enforcement authorities. By accessing and using this server, you expressly
consent to the acceptance of these terms, as well as those described in all
applicable Mines' policies.

Note: If you see "system restart required" above this message, the system will
restart at 4:00 AM this upcoming Monday for security updates.  Systems may
be restarted sooner if required for security updates.
Last login: Sat Feb 24 21:31:04 2024 from 138.67.173.114
tolgacan@isengard:~$
```

bash

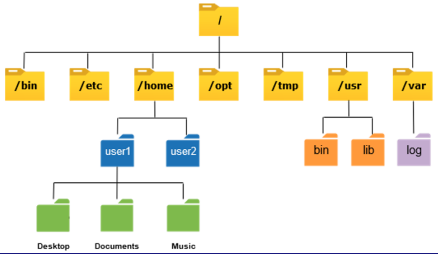
- Installed as the default shell in most Linux distributions
- There are others (Zsh, Korn, Tcsh, Fish)
- This class will focus on bash
- A shell interprets command line operations typed by the user

Alternatives to command line ssh

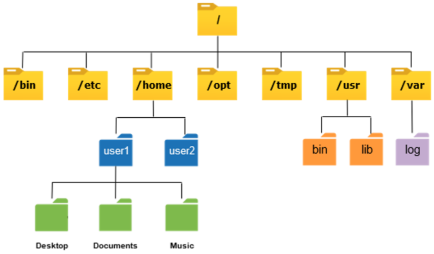
- WSL – Windows Subsystem for Linux ▸ Link
- Virtual Machines
  - Virtual Box ▸ Link
  - VMWare ▸ Link
  - Parallels Desktop ▸ Link
- Installing a Linux distribution on your own machine ▸ Link
- VSCode ssh extension ▸ Link

The Filesystem

- Tools, packages, user documents, configuration files – almost everything – is a file in an operating system
- Some are hidden, some are “accessible by” only certain “super” users
- Files are organized as a set of nested directories



File Path



- A file is referred to by a *path*
- A string of characters used to uniquely identify a file in a file system.
- Example: `/home/user1/Desktop`

Different ways to write paths

- / is the *root* directory, which is the outermost directory that holds all the other directories and files
- An *absolute* path is a path that starts with / and lists all directories leading to a file, i.e., the “complete” address. Example: `/home/user1/Desktop`
- A *relative* path is a path that refers to a file location *relative* to the *current* directory. Example: `Desktop/file1.txt`

Path aliases

- `.` is the relative current directory
- `..` is the relative parent directory. Example: `../Documents`
- ~ is the home directory of the user. Example: `~/Desktop`

Important Tips

- IMPORTANT: Linux is case sensitive. You need to know the relative/absolute location of a file/directory “exactly” to work on them.
- IMPORTANT: Using the `TAB` key is very helpful both to autocomplete and to verify a file’s location.

File system navigation and manipulation

In a file system, we should be able to:

- see where we are
- move somewhere else
- interact with files
- create new files and new directories

Navigation Commands

pwd and ls

`pwd`

Print **W**orking **D**irectory

Use `pwd` to figure out where you are in the filesystem

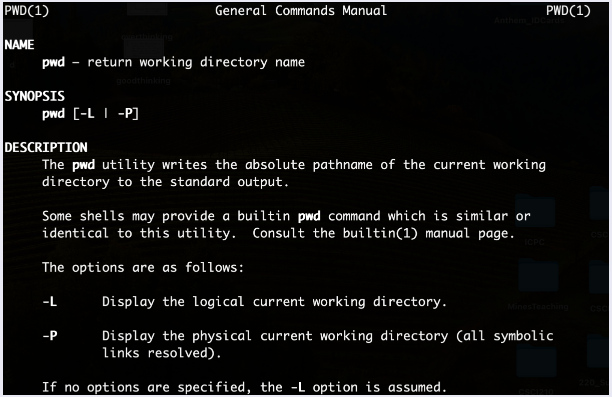
`ls [-a | -l | -h | -r | -S | -t | -1] [file ...]`

List directory contents

Use `ls` to see what's in a directory

Let's learn more about these commands

man command_name



```
PWD(1)                  General Commands Manual                  PWD(1)

NAME
     pwd – return working directory name

SYNOPSIS
     pwd [-L | -P]

DESCRIPTION
     The pwd utility writes the absolute pathname of the current working
     directory to the standard output.

     Some shells may provide a builtin pwd command which is similar or
     identical to this utility.  Consult the builtin(1) manual page.

     The options are as follows:

     -L      Display the logical current working directory.

     -P      Display the physical current working directory (all symbolic
             links resolved).

     If no options are specified, the -L option is assumed.
```

Command anatomy

```
command [options] parameter1 parameter2 ...
```

- Commands are case sensitive
- Options are “optional” and they are modifications to the default command behavior. They are preceded by `-` or `--`
  - Options are specified using *single characters* after a single dash
  - They are specified using *words* after two dashes
  - Multiple single character options can be combined after a single dash, e.g., `ls -al`
- Required parameters are listed outside the square brackets and their number can vary from command to command (most commands have 1 or 2 parameters).

Navigation Commands

ls

Try out different options of `ls`.

```
ls [-a | -l | -h | -r | -S | -t | -1] [file ...]
```

The file parameter is optional and multiple files can be given as parameters. Give directory names or file names as the parameter. See what happens.

Navigation Commands

cd

```
cd [directory]
```

Change current **D**irectory

Use `cd` to move through the filesystem

If `directory` is omitted change directory to the user's home directory, i.e., "`cd`" and "`cd ~`" have the same effect.

File Manipulation

mkdir

```
mkdir [-v | -p] directory
```

Make **Dir**ectories

Use `mkdir` to create new directories

- Default behavior: `directory` should be a file path to a non-existent directory and the parent directories should be created beforehand.
- `-p` option: Create parent/ancestor directories as needed and do not give an error if the directory already exists.

File Manipulation

rmdir

```
rmdir [-v | -p] directory
```

Remove **Dir**ectories

Use `rmdir` to remove **empty** directories

- IMPORTANT: Removal is permanent!

File Manipulation

touch

```
touch [options] file ...
```

Updates/modifies the access time of a file. If `file` does not exist, creates a new empty file

Use `touch` to create new files

File Manipulation

rm

```
rm [-d | -f | -i | -r | -v] file ...
```

Removes files

Use `rm` to remove files (or directories)

- Default behavior: `file` should be a file path to a file (not to a directory).
- `-r` or `-R` option: `file` can refer to a directory and all the directories contents are deleted recursively.
- PERMANENT: again, no undo as in `rmdir`

File Manipulation

mv

```
mv [-b | -f | -i | -n] source target
```

Move `source` file or directory to `target` path

Use `mv` to move files (or directories)

- Note: `-r` option is not required to move directories.
- IMPORTANT: If the target destination is an existing file/directory, it is overwritten with no warning by default. Use "`-i`" to issue a warning.

File Manipulation

cp

`cp [-i | -R] source target`

Copy `source` file or directory to `target` path

Use cp to copy files (or directories)

- Note: `-R` option is required to recursively copy directories.
- IMPORTANT: If the target destination is an existing file/directory, it is overwritten with no warning by default. Use "`-i`" to issue a warning.

Transferring Files

scp

`scp source target`

Use `scp` to securely copy (i.e., transfer) files (or directories with the `-r` option)

- For `scp` to work, the remote host should be able to accept connections like in `ssh`
- The file location on a remote host is specified using the following format: `user@host:path`, where `path` is given relative to the user's home directory (or an absolute path needs to be given). Example:
  `scp local.txt tolgacan@isengard.mines.edu:./remote.txt`

File Manipulation

less

`less file`

Reads the contents of a file one page at a time (fast)

Use `less` to quickly look at the contents of text files

`man` or `help`

- All of the commands above except `cd` have manual pages, because they are individual commands that are executed as separate processes (more on this when we talk about processes later).
- `cd` is a *built-in* function of the shell, which means the `bash` process handles the command.
- If you cannot remember the command name to search for with `man`, you can use the `apropos` command to do a keyword search of the manual pages.