

Advanced Programming Lab

CS6150

Week 3

(Linux & Debugging Basics)

Sanket Tarafder (cs24s018@{cse|smail})

Learning Objectives

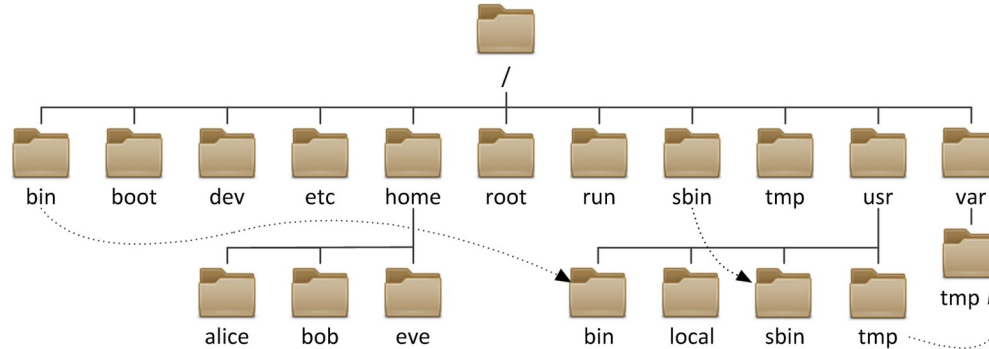
- Linux Basics
- Identifying Types of Errors
- Debugging Techniques
- GDB
- Valgrind (If time permits)

Linux Basics

Why Linux for Programming?

- Developer-Centric Design
- Powerful Command Line
- Development Ready Out-of-the-Box
- Widely Used in Industry
- Free, Open Source, Lightweight

Linux File System



- / : The starting point for all other directories and files within a Linux system
- /bin : Essential softwares to keep the system running
- /home : Contains *home* directories for each user
- /usr : Essentially, subdirectories containing most software used on the system, including system libraries and documentation

Scenario

- You just received a USB from a teammate containing hundreds of files — documents, images, code, logs, etc. — all dumped in a single folder.
- You want to:
 - Sort files into separate folders by extension.
 - Convert all `.c` files to `.cpp` because the project migrated to C++.
 - Find all files mentioning the word `TODO` (case-insensitive) and store them in a separate folder.
 - Generate a report of how many files are in each category.

Linux Demo

Essential Commands

- `pwd` : Show the current working directory
- `cd` : Change to another directory
- `ls` : List files and folders in the current directory
- `mkdir` : Create a new directory
- `touch` : Create a new empty file
- `rm` : Remove files or directories
- `cp` : Copy files or directories

Essential Commands

- `mv` : Move or rename files and directories
- `cat` : Display the contents of a file
- `nano` : Simple terminal-based text editor
- `clear` : Clear the terminal screen
- `man` : Show the manual page for a command
- `wc` : Counts lines, words, and characters in files
- `diff` : Compare two files line by line

find: Locate Files/Dirs by Criteria

```
> find [PATH] [OPTIONS] [ACTIONS]
```

- `-name "*.c"` : Match filenames
- `-type f / d` : File or directory
- `-size +1M` : Bigger than 1MB
- `-mtime -7` : Modified in last 7 days
- `-exec CMD {} \;` : Run CMD on each file
- `-delete` : Remove matched files

Example:

```
> find . -type f -name "*.log" -delete
```

grep: Pattern Matching in Files

```
> grep [OPTIONS] PATTERN [FILE...]
```

- `-n` : Show line numbers
- `-i` : Case-insensitive
- `-r` : Recursive search
- `-v` : Invert match (lines that don't match)
- `-o` : Only matched part

Example:

```
> grep -rin "main" src/
```

Redirection & Pipes

- Redirections

Command	Explanation	Example
>	Redirect stdout (overwrite file)	<code>ls > files.txt</code>
>>	Redirect stdout (append to file)	<code>echo "Hi" >> log.txt</code>
<	Redirect stdin from a file	<code>./sort < data.txt</code>
2>	Redirect stderr	<code>gcc code.c 2> errors.txt</code>
&>	Redirect stdout + stderr	<code>command &> output.txt</code>

- Pipes

- Pass the output of one command as the input to another.
- Example:

```
> ls -l | grep ".cpp"
```

Compilation with g++

```
> g++ -Wall -g main.cpp -o main
```

- `-Wall` : all warnings
- `-g` : enables debugging symbols
- `-o <output>` : names the output binary

Bugs and Errors

What is a Bug?

- A bug is an error or flaw in a program that causes it to behave unexpectedly or incorrectly.
- Bugs can lead to:
 - Crashes (e.g., segmentation faults)
 - Incorrect output
 - Security vulnerabilities
- Bugs are common even in well-tested software.
- Debugging is the process of finding and fixing bugs.

Types of Errors

Type	Example	Detected By
Syntax Error	<code>int x = ;</code>	Compiler
Runtime Error	Segfault, divide by 0	OS
Logic Error	<code>if (marks > 40) std::cout << "Fail";</code>	Only you!

Debugging Techniques

Print-based Debugging (cout / printf)

- Approach
 - Most basic form of debugging
 - Insert print statements to trace values and flow
 - Commonly used for quick checks
- Limitation:
 - Doesn't scale with large codebases
 - Needs recompilation and cleanup afterward

Rubber Debugging

- Approach
 - Explain your code line-by-line to someone — or a duck
 - Forces you to slow down and think clearly
- How it helps:
 - Identifies assumptions
 - Reveals flaws in logic
 - Encourages deliberate review

GDB: Step Through Your Code

- Why GDB?
 - Trace code without inserting print statements
 - Inspect values, set breakpoints
 - Works with segmentation faults, logic bugs, loops
- Key Advantages:
 - No need to modify code repeatedly
 - Powerful for large projects

Debugging Demo

GDB: Commands

Command	Explanation
<code>gdb ./a.out</code>	Start GDB with the compiled executable
<code>break <line func></code>	Sets the breakpoint to the specified line or function
<code>run or r</code>	Start running the program under GDB
<code>next or n</code>	Execute next line (without stepping into functions)
<code>step or s</code>	Step into function calls
<code>continue or c</code>	Continue running the program until next breakpoint
<code>print <var> or p</code>	Print the current value of a variable
<code>display <var></code>	Automatically print variable after each step

GDB: Commands

Command	Explanation
<code>backtrace</code> or <code>bt</code>	Show function call stack trace
<code>list</code> or <code>l</code>	Show source code lines around current line
<code>info locals</code>	Show local variables in the current stack frame
<code>info breakpoints</code>	List all breakpoints set
<code>delete <bp#></code>	Delete a specific breakpoint by number
<code>disable <bp#></code>	Temporarily disable a breakpoint
<code>enable <bp#></code>	Re-enable a disabled breakpoint
<code>quit</code> or <code>q</code>	Exit GDB

What is a Memory Leak?

- A memory leak occurs when a program allocates memory but fails to release it, causing wasted memory over time.
- Downsides of Memory Leaks:
 - Gradually consumes RAM
 - Slows down system or crashes apps
 - Very hard to detect manually!
- Why it matters:
 - Critical for long-running programs (servers, daemons, etc.)
- Common Causes:
 - `new` / `malloc` without `delete` / `free`
 - Forgetting to `free` after early return

What is Valgrind?

- Valgrind is a tool that detects memory leaks, uninitialized memory use, and invalid memory access in C/C++ programs.
- How it works:
 - Runs your program in a simulated environment
 - Tracks all memory allocations and releases
- When to use:
 - After writing/modifying pointer-heavy code
 - When facing strange crashes or slowdowns

Valgrind Demo

Some Important Links

- Setting Up Linux
 - Windows Subsystem for Linux (WSL) Setup. [Link](#)
 - Dual Boot Ubuntu with Windows. [Link](#)
 - VirtualBox Downloads (for running Linux in a VM). [Link](#)
- Installing Development Tools
 - Install [GDB](#) (GNU Debugger)
 - Ubuntu/Debian: `sudo apt install gdb`
 - Install [Valgrind](#)
 - Ubuntu/Debian: `sudo apt install valgrind`
 - Install build-essential (gcc, g++, make, glibc)
 - Ubuntu/Debian: `sudo apt install build-essential`

Some Important Links

- Debugging & Visualization
 - [PythonTutor](#) (visualize code execution, supports C/C++)
 - [Explainshell](#) (breaks down any Linux command)
 - GDB Cheat [Sheet](#)
- Extra Learning
 - [Linux Journey](#) (interactive basics)
 - Valgrind Memcheck [Guide](#)
 - Video [Tutorial](#) on Valgrind

Thank You