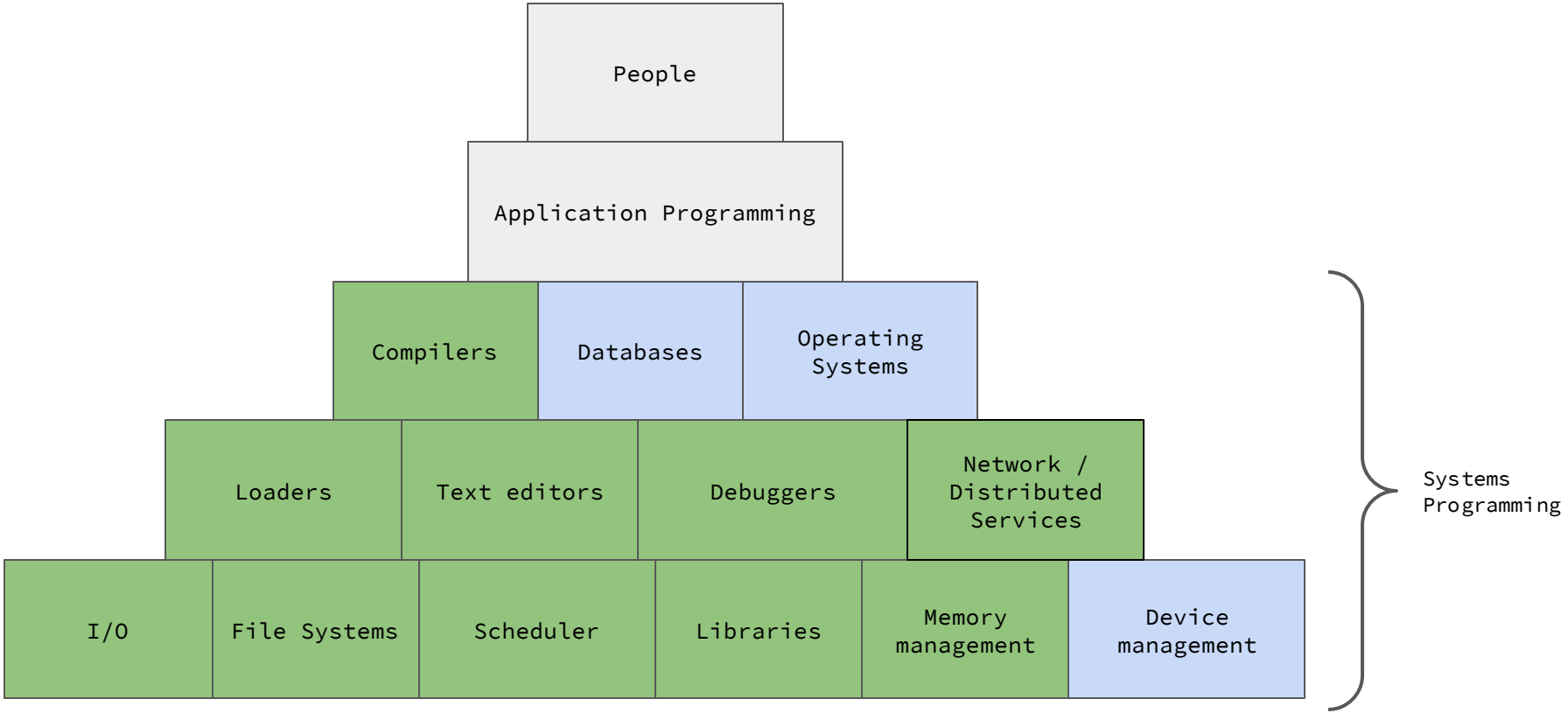
CSC / CPE 357

Systems Programming

Topics

- Integrating Concepts
- Shell

These contents are based on materials from
Professors Aaron Keen, Phillip Nico and Andrew Migler


People
Application Programming
Compilers
Databases
Operating Systems
Loaders
Text editors
Debuggers
Network / Distributed Services
I/O
File Systems
Scheduler
Libraries
Memory management
Device management
Systems Programming

Integrating Concepts

What does "Integrating Concepts" mean in system programming?

- Combining multiple fundamental ideas to create efficient and functional programs.
- Bridging the gap between theory and practical applications.
- Example: Integrating file systems, processes, memory, and networking.

Why is it Important?

- Real-world problems require a holistic approach.
- Encourages modular and scalable design.

Key Areas of Integration

1. Process and Thread Management

- Example: A server handling multiple client requests using threads.
- Integration of concurrency, synchronization, and inter-process communication (IPC).

2. File System and I/O

- Example: Logging system that writes data to files while processing inputs in real-time.

3. Memory Management

- Example: Dynamic memory allocation in large applications.
- Combining stack, heap, and memory mapping techniques.

4. Networking

- Example: A chat application integrating sockets, threads, and message parsing.

Example: A Simple HTTP Server

Concepts Integrated:

- **File System:** Serve static files from a directory.
- **Networking:** Listen for and respond to client requests over TCP.
- **Concurrency:** Handle multiple client requests simultaneously using threads or non-blocking I/O.

*Programming Assignment 5

Challenges in Integration

1. Debugging and Testing
- Issues often arise at the boundaries of integrated components.

2. Performance Trade-offs
- Example: Using threads improves concurrency but increases resource usage.

3. Scalability
- Ensuring integrated solutions work under high load.

4. Security
- Example: Avoiding buffer overflows in systems handling external inputs.

- In-class activity at the end of the lecture

The UNIX Shell

What is Unix Shell?

- A command-line interface for interacting with the operating system.
- Acts as an intermediary between the user and the OS kernel.

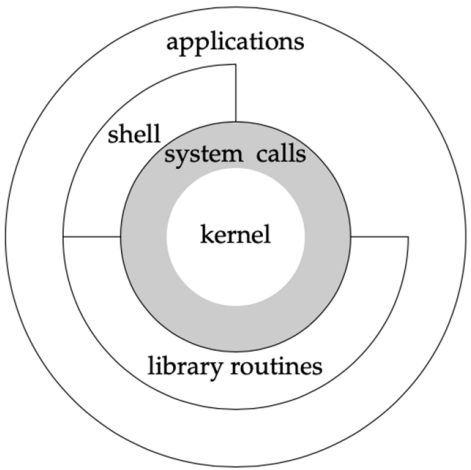


Figure 1.1 Architecture of the UNIX operating system

Source: Advanced Programming in the UNIX Environment, 3rd Ed.

The UNIX Shell

Key Features:

- Command execution (Run programs, Provide input, Inspect output)
- Shell scripting for task automation.
- Program and process control.

Common Shells:

- Bourne Shell (sh): The original Unix shell.
- Bourne-Again Shell (bash): An enhanced version of sh with additional features.
- C Shell (csh): Incorporates C-like syntax.
- Korn Shell (ksh): Combines features from sh and csh.
- Z Shell (zsh): Extended Bourne Shell

Navigating the Shell Environment

Accessing the Shell: Using terminal emulators or console interfaces.

Command Structure: command [options] [arguments]

Getting Help:

man [command]: Opens the manual page for a command.

--help: Displays help information for a command.

Basic UNIX Shell Commands

File and Directory Management:
- ls – List files and directories.
- cd – Change directory.
- mkdir – Create a new directory.
- rm – Remove files/directories.

File Operations:
- cat – Concatenate and display files.
- cp – Copy files/directories.
- mv – Move/rename files.
- touch – Create an empty file.

System Information:
- pwd – Print the current directory.
- whoami – Display current user.
- uname -a – Show system information.

Working with Processes

Key Commands:

- ps – View active processes.
- top/htop – Monitor processes in real-time.
- kill – Terminate a process.
- bg/fg – Manage background and foreground processes.

Job Control:

- jobs – List background jobs.
- & – Run a process in the background.
- Ctrl+Z – Suspend a foreground process.

Unix Shell Scripting

Shell Scripting:

- Writing a series of shell commands in a file to automate tasks.

Shell Script:

- A file containing a sequence of shell commands.
- *.sh

```
#!/bin/bash
echo "Enter your name:"
read name
echo "Hello, $name!"
```

Shell Scripting Basics

Creating a Script:

- Use a text editor to write commands.
- Start with a shebang (#!/bin/sh or #!/bin/bash) to specify the interpreter.

Making Scripts Executable:

- Change permissions using chmod +x scriptname.sh.

Running Scripts:

- Execute with ./scriptname.sh.

Variables and Control Structures

Variables:

Assign values using variable_name=value.

Access values with $variable_name.

Control Structures:

- Conditional Statements:

  ```
  if [ condition ]; then ... fi
  ```

- Loops:

  ```
  for var in list; do ... done
  ```

  ```
  while [ condition ]; do ... done
  ```

Practical Applications of Shell Scripting

- **Automation:** Automate repetitive tasks like backups and system monitoring.
- **System Administration:** Manage user accounts, processes, and services.
- **Text Processing:** Use tools like grep, sed, etc. for data manipulation.
- **Networking:** Automate network configurations and monitor network status.

Best Practices

- Use comments to explain scripts.
- Handle errors gracefully.
- Follow consistent naming conventions.
- Test scripts in a safe environment.
- Familiarize yourself with man pages (e.g., man ls).
- Hands-on practice on Unix-based systems.

- In-class activity