

CSC / CPE 357

Systems Programming

Course Learning Objectives

After completion of this course, you will be able to:

- Read and write complex C programs
- Understand UNIX architecture, commands and development environment
- Write programs using operating system services (system calls)
- Distinguish between language features and operating system features

Lecture Topic

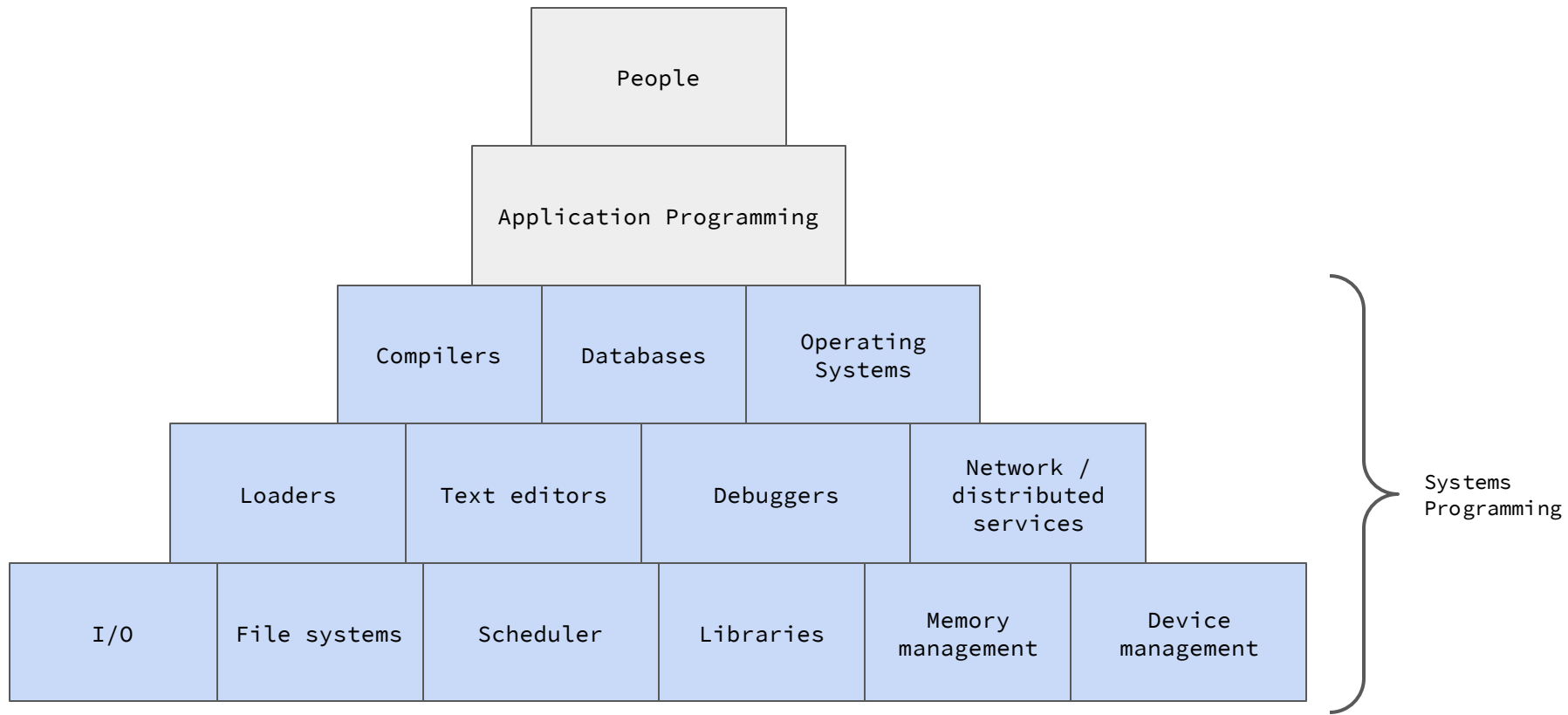
- Introduction
- What is Systems Programming?
- Why Systems Programming?

What is Systems Programming?

- The term "Systems Programming" is used and interpreted in many different ways
- Here we will sketch a shared definition for this course, drawing on these (and other) references:
 - [Systems Programming Languages](#) (Bergeron et al. 1972)
 - [BLISS: A Language for Systems Programming](#) (Wulf et al. 1972)
 - [Systems Programming](#) (Donovan 1972)
 - Panel: [Systems Programming in 2014 and Beyond](#)

Systems vs Application Programming

- Systems Programs serve other programs. For example:
 - Operating systems
 - Database systems
 - HTTP servers
- Applications serve users



Systems Programming Languages (Bergeron et al. 1972)

System Program:

- Integrated set of subprograms
- Forms a whole greater than the sum of its parts
- Exceeds some threshold of size and/or complexity

Systems Programming Languages (Bergeron et al. 1972)

Goals of a systems programming language:

- Can be used without undue concern about hardware details
- Readability of high level languages
- Space and time efficiency
- Ability to “get at” machine and operating system facilities that are accessible via assembly language.

Systems Programming Languages (Bergeron et al. 1972)

1. The problem to be solved is of a broad nature consisting of many, and usually quite varied, sub-problems.

-
2. The system program is likely to be used to support other software and applications programs, but may also be a complete applications package itself.

3. It is designed for continued “production” use rather than a one-shot solution to a single applications problem.

4. It is likely to be continuously evolving in the number and types of features it supports.

5. A system program requires a certain discipline or structure, both within and between modules (i.e. , “communication”), and is usually designed and implemented by more than one person.

Systems Programs – Summary

1. Broad problem, numerous subproblems
2. Support other systems and applications
3. Continuous use
4. Continuously evolving features
5. Disciplined communication between modules

Recent Definitions of Systems Programming

From the panel: [Systems Programming in 2014 and Beyond](#)

Bjarne Stroustrup (creator of C++): Systems programming came out of the field where you had to deal with **hardware**, and then the applications became more complicated. You need to deal with **complexity**. If you have any issues of significant **resource constraints**, you're in the systems programming domain. If you need **finer grained control**, then you're also in the systems programming domain. It's the constraints that determine whether it's systems programming. Are you running out of **memory**? Are you running out of **time**?

Recent Definitions of Systems Programming

From the panel: [Systems Programming in 2014 and Beyond](#)


Andrei Alexandrescu (D developer): I have a few litmus tests for checking whether something is a systems programming language. A systems programming language must allow you to **write your own memory allocator**. You should be able to forge a number into a pointer, since that's **how hardware works**.

Recent Definitions of Systems Programming

From the panel: [Systems Programming in 2014 and Beyond](#)

Rob Pike (creator of Go): When we first announced Go, we called it a systems programming language, and I slightly regret that because a lot of people assumed it was an operating systems writing language. What we should have called it is a server writing language, which is what we really thought of it as. Now I understand that what we have is a cloud infrastructure language. Another definition of systems programming is **the stuff that runs in the cloud**.

Low-Level / Systems Programming

1. Broad problem, numerous subproblems
 2. Support other systems and applications
 3. Continuous use
 4. Continuously evolving features
 5. Disciplined communication between modules
 6. Significant resource constraints
 7. Need to leverage hardware details (preferably without assembly language)
- 
- (AKA
Software
Development)

Low-Level Software Systems

Many interesting, fundamental low-level software systems:

- Database systems
- Networking/distributed systems
- Operating systems
- Compilers

Low-Level / Systems Programming

- How to design and construct systems that have significant resource constraints?
- How can we design the system around the machine?

Low-Level / Systems Programming Languages

Distinguishing features of systems programming languages:

- Details of the underlying hardware are exposed
 - Memory allocation/layout
 - Registers
- Fine-grained resource management
 - Possible to eliminate bottlenecks with careful control over machine details

Contrast with JIT / Scripting Languages

- "Scripting Languages" (Python, JavaScript, etc.)
 - Glue together components built using systems programming languages
- Competitive performance from Just-in-Time compilers for Java, JavaScript, Python ([2011](#))
 - "Dropbox is a big user of Python. It's our most widely used language both for backend services" ([2019](#))

Low-Level / Systems Programming in 357

Handling low-level constraints (memory, time) requires:

- Understanding of the environment (in 357: UNIX/Linux)
- Programming language support (in 357: C)
- Creative design
- Carefully-applied abstraction

Abstraction

- Abstractions arise as we filter out details to focus on key ideas.
- Edsger Dijkstra: "...think of a complex system as a layered structure, in which each layer transforms layers below it into a higher-level abstraction"
 - At every layer we have data types, and abstract operations
 - Each hides its implementation and introduces properties and guarantees

Benefits of Abstraction

- Dijkstra: The real power of layered abstractions centers on specifications and proofs.
- If we can fully describe the interfaces to our systems (APIs), and their behaviors, we can rigorously verify implementations.

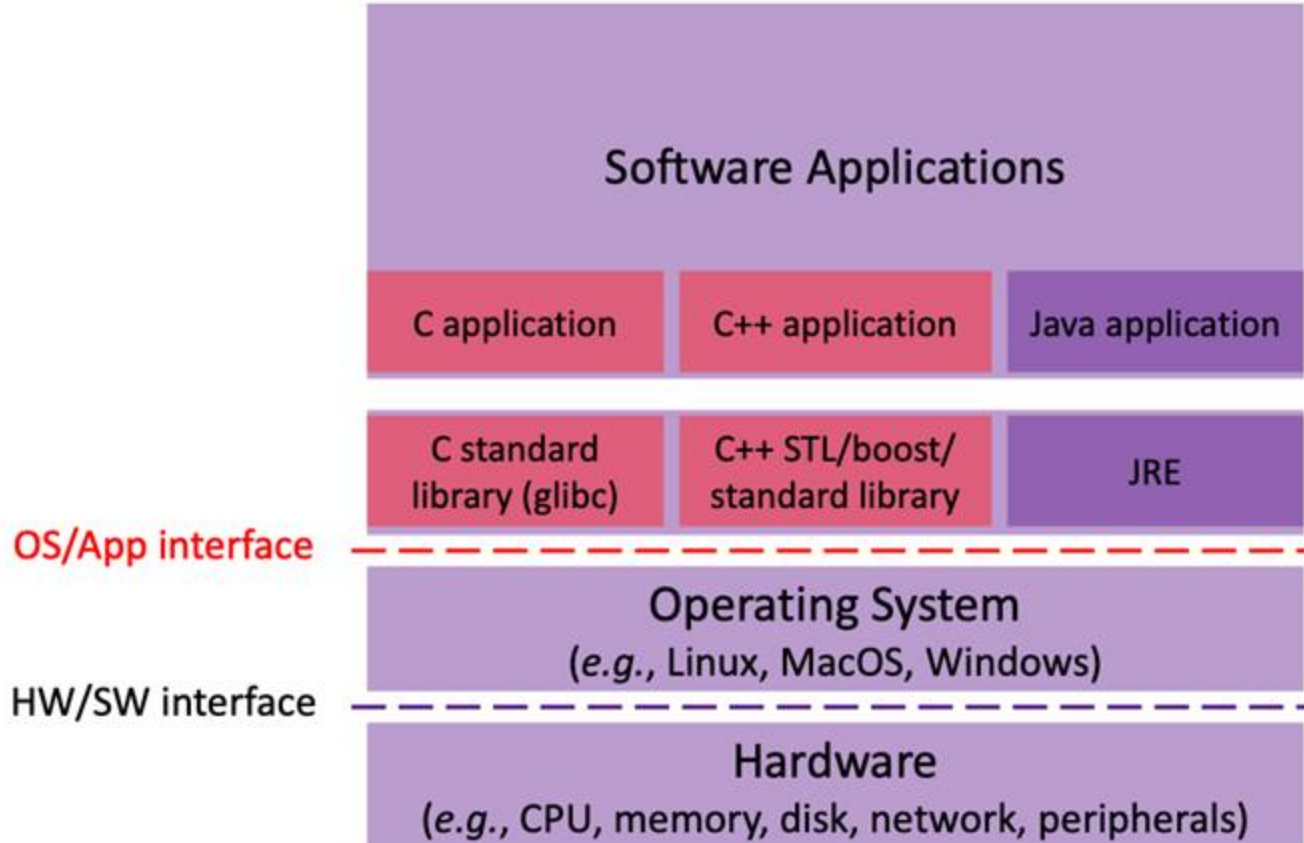
Abstraction Considerations

- We want simple, elegant abstractions. However, abstractions can hide costs.
- Need to leverage high-performance hardware that may be very complex to use directly.
- We want security and protection... without sacrificing speed.

Abstraction

- Abstraction is a powerful tool, for example:
 - A file system abstracts storage: The device just hold bytes
 - We can abstract an entire system as a VM/container, use this to move applications to a new environment like a cloud.
- Excessive abstraction can be harmful or overly limiting
 - The C programming language errs on the side of less abstraction

Layers of Abstraction



Definition: Operating System

- **Operating System:** software that controls the hardware resources of the computer and provides an environment under which programs can run.
- A **kernel** resides at the core of the environment.
 - $\text{Linux} \cong \text{kernel} + \text{system utilities} + \text{applications} + \text{shells} / \text{command interpreters} + \text{libraries of common functions}$

UNIX/Linux Operating System

- Manage hardware and devices and file system. Be minimal, performant, correct, secure. Use energy efficiently.
- Offer a process abstraction, segmented memory, threads.
- Many computers are shared. Create a “virtual private computer” for each user.

UNIX/Linux Operating System (continued)

- Ease of management (“administration”), easy to diagnose problems when they occur.
- Portability: the same kernel should run on many kinds of computers, and it should be easy to move a user and her processes from machine to machine.

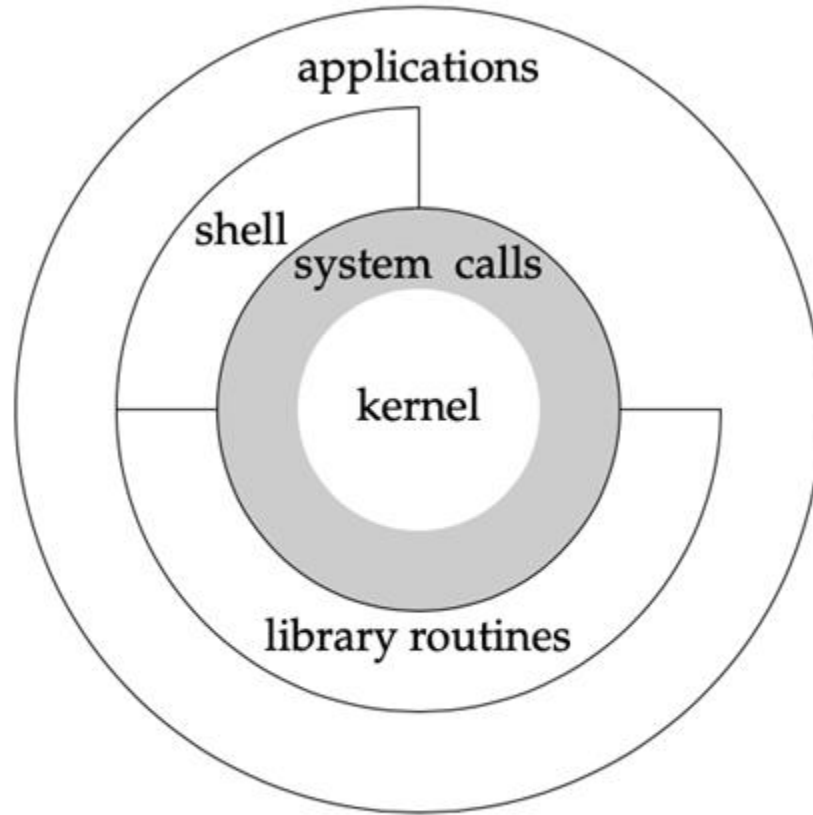


Figure 1.1 Architecture of the UNIX operating system

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