



FUNDAMENTALS

CPE / CSC 159: OPERATING SYSTEM PRAGMATICS

GREG CRIST (CRIST@CSUS.EDU)



FUNDAMENTALS

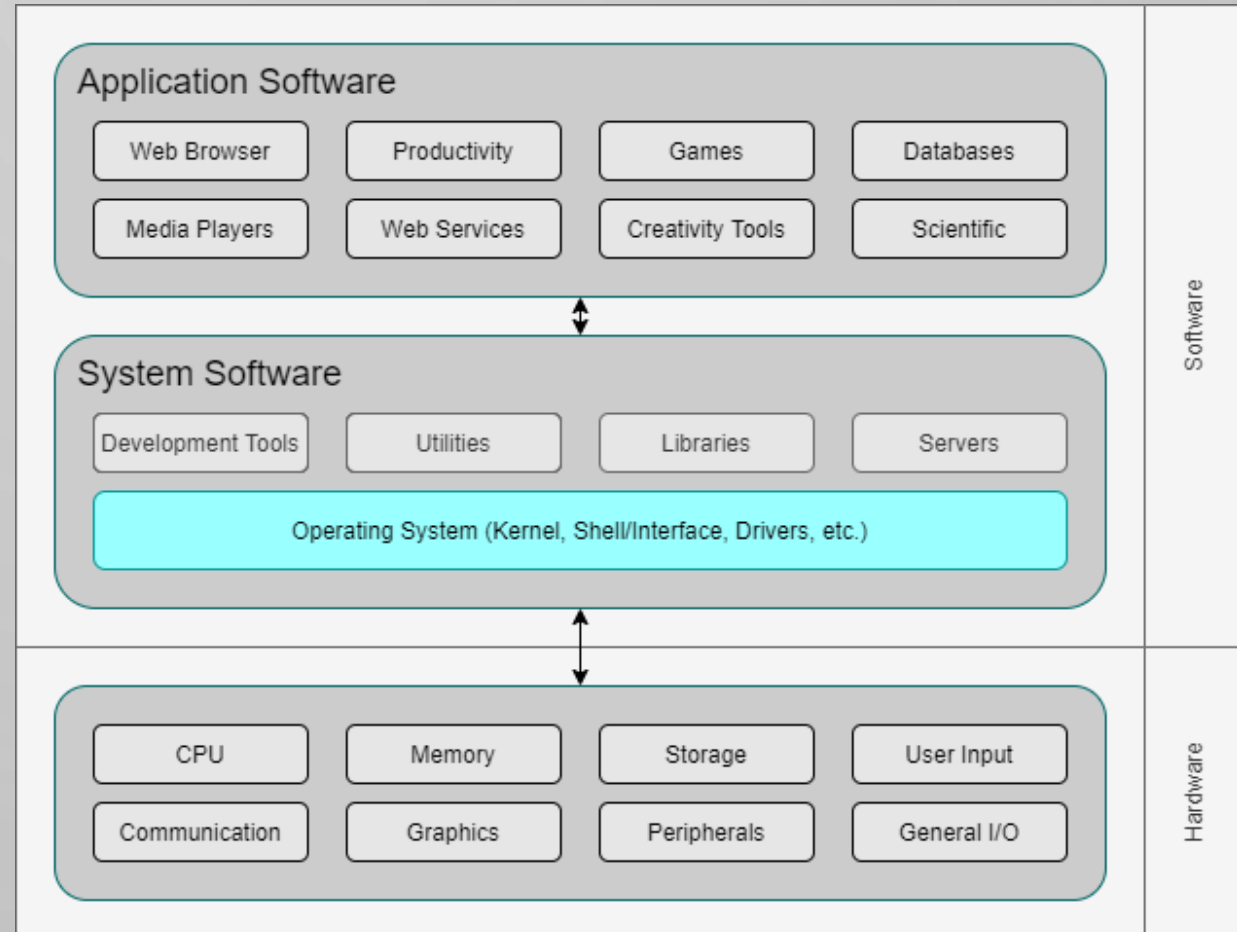
- What is an Operating System?
 - Intel x86 Architecture
 - Project Development Environment (SPEDE)
- 
- 
- 

A decorative graphic on the left side of the slide consisting of a network of light blue lines and small circles, resembling a circuit board or a neural network, extending from the top and bottom edges towards the center.

WHAT IS AN OPERATING SYSTEM?

WHAT IS AN OPERATING SYSTEM?

- The operating system (OS) is the most important software in a computer system.
- It is the software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals.
- It manages all hardware and software resources, provides services to software that runs within the operating system, and typically provides a user interface to facilitate the use of the system.





OPERATING SYSTEM SCOPE

- When referring to an operating system, there are many perspectives as to what it means:
 1. The Kernel
 2. The Kernel... and Drivers, Interfaces, and System Services
 3. The Kernel, Drivers, Interfaces, System Services... and Utilities, Programs, Tools, and more
 4. The above, and ...
- In our class, we will largely be focusing on the first two perspectives:
 1. The Kernel
 2. The supporting drivers, interfaces, and system services that make use of the Kernel



THE ROLE OF AN OPERATING SYSTEM

- Operating systems take on various roles:
 - Managing resources
 - Hardware Abstraction
 - Providing system services
 - Providing user interfaces
- 
- 

MANAGING RESOURCES?

- Resources may be:
 - CPU Time
 - Memory
 - Data / File Storage
 - Hardware and peripherals
 - Networking and communication
 - Security
 - Programs / Processes
 - More?
- The operating system controls the allocation, operation, isolation, and arbitration of access to different resources

HARDWARE ABSTRACTIONS



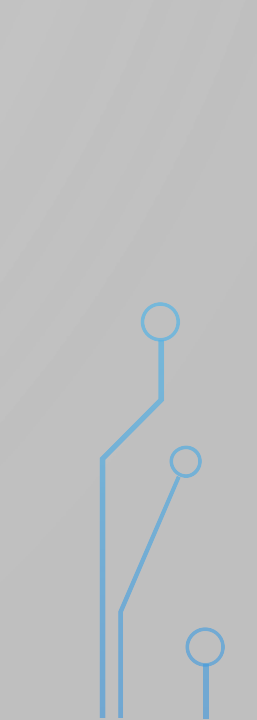
- Portions of the operating system must have intimate knowledge and understanding of the underlying hardware
 - Specifically: system architecture, underlying hardware layout
- Portions of the operating system may have hardware abstracted
 - Examples: process scheduling algorithms, inter process communication methods, user interfaces
- User software and applications typically do not need to be aware of the underlying hardware due to this abstraction
 - All programs are compiled for a specific instruction set
 - Programs *can be* programmed to be platform independent (see: “hello world”)
- As hardware becomes abstracted, more complex interactions can take place:
 - CPUs/CPU Cores -> Parallelization / increased multitasking
 - Physical RAM -> paged memory -> virtual memory
 - Physical storage -> complex file systems and structures
 - Physical (or wireless) means of connecting systems -> network protocols and communication

SYSTEM SERVICES

- Provide a common set of mechanisms for processes to interact
 - With the kernel
 - With hardware
 - With other processes
 - With other/external systems
- User programs may make use of the operating system environment to
 - Interact with various resources
 - Interact with other programs/processes



KEY COMPONENTS

- Kernel
 - Programs/Processes
 - Drivers/Hardware Interfaces
 - Memory Management
 - Data Storage/Transfer
 - User Interfaces
 - Bootloader*
- 
- 
- 

THE KERNEL

- The kernel: “the most important piece of something”
- It is the core of the operating system
- It has complete control over everything within the computer system
- Provides services and interfaces to user processes

PROGRAMS/PROCESSES

- Processes are instances of a computer program that are running in memory
 - A set of computer instructions that are executed by the CPU
- Processes may operate in different contexts
 - User context or user processes (think: visible to the user in some way)
 - Kernel context or kernel processes (think: internal to the kernel/operating system)
 - “part of the operating system”

DRIVERS / HARDWARE INTERFACES

- Drivers are a piece of software that manages and controls some piece of hardware
 - Can be simple or complex
- Drivers may be built-in to the operating system/kernel, or may be separate
 - Depends on operating system design/implementation
- Drivers may provide an abstraction of, or interface to the underlying hardware
 - Programs making use of the underlying hardware may not need to be aware of how it actually functions

MEMORY MANAGEMENT

- All programs need to allocate and use memory to function
- The operating system controls the allocation and use of memory
- Memory is physically limited, but can be abstracted to seem limitless
 - paged memory -> virtual memory

DATA STORAGE/TRANSFER

- Programs and data are stored in some physical medium
- The operating system can abstract how this data is accessed
 - Read/write operations
 - File systems
 - Loading from non-volatile storage to RAM
- The operating system facilitates the exchange of data
 - Within the system (interprocess communication)
 - Externally to the system (data busses, networking, etc.)

USER INTERFACES


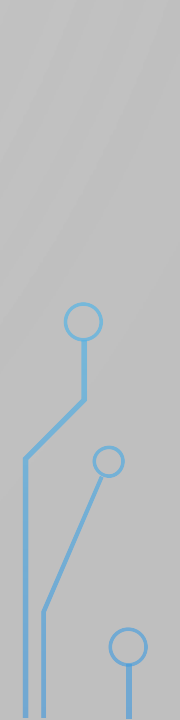
- Most operating systems provide some mechanism for the user to interact with it
- The term “user” can be subjective
 - We typically think of this as a person
 - Consider systems or devices that may not be directly controlled by a person – interfaces can be to other systems/computers
- Examples of user / human interfaces:
 - text console / shell
 - graphical interfaces (point-and-click/drag-and-drop/touch)
- Examples of non-human interfaces
 - client/server architectures
 - system interconnects/busses

BOOTLOADER

- The bootloader is a piece of software that actually runs before an operating system is running
- It is usually tiny, focused, and performs the task of loading the operating system kernel from disk/persistent storage into RAM and then passing control to the kernel to take over
- It may be bundled with the operating system, but is generally not considered to be part of the operating system



OUR OPERATING SYSTEM

- We will be developing a simple operating system with these concepts in mind
 - It will run on top of the Intel x86 CPU architecture
- 
- 

A stylized graphic of a circuit board or network diagram is positioned on the left side of the slide. It features a central vertical line with numerous horizontal and diagonal branches extending to the left. Each branch terminates in a small circle, resembling a component or a node in a network. The lines are a light blue color, and the circles are white with blue outlines.

INTEL X86 ARCHITECTURE

A (VERY) BRIEF INTRODUCTION

WHY INTEL X86?

- With so many different CPU and system architectures available, why the Intel x86?
 - Well documented and very well understood
 - Should be somewhat familiar for most students
 - Instruction set has been future-compatible

BRIEF HISTORY OF THE INTEL X86 ARCHITECTURES

- Originated in 1978 with the Intel 8086 processor
- Successors included the 80186, 80286, 80386, 80486 processors
 - Hence, the nomenclature of “x86”
- Key aspect of the x86 architectures
 - Memory segmentation (8086 was 16-bit, but allowed access to more memory than could be addressed by 16-bit addresses)
 - Mostly full backwards compatibility (still to this day!)
- CISC (complex instruction set computer) architecture
 - Many instructions, variable instruction length
 - New processor architectures add new instructions

BACKWARDS COMPATIBILITY

- Backwards compatibility due to different operating modes
 - Real mode – 16-bit instruction set, limited memory can be addressed
 - Protected mode – 32-bit instruction set, segmentation, addressable virtual memory, retained 16-bit instructions
 - Long mode – 64-bit instruction set, larger addressable memory, dropped some 16-bit and 32-bit instructions
- As the system boots, transitions from real -> protected -> long mode
 - This occurs via software, typically as part of the bootloader

BOOT PROCESS

- CPU contains small ROM with instructions sufficient to load BIOS
 - UEFI in modern systems
- BIOS detects and initializes basic hardware
 - Enumerates boot devices
 - Loads boot records and starts running
- Bootloader
 - Run from the boot device
 - Loads operating system kernel and passes control to kernel

OUR OPERATING SYSTEM

- We will be developing our operating system using the intel x86 architecture
 - Specifically: targeting the Intel 80386 instruction set (protected mode)
- To facilitate this, we will be using a special development operating environment called SPEDE

An abstract graphic on the left side of the slide, consisting of a network of light blue lines and small circles, resembling a circuit board or a neural network diagram. The lines are vertical and horizontal, with some diagonal connections, and the circles are placed at various points along these lines.

SPEDE

OUR DEVELOPMENT / TEST ENVIRONMENT

SPEDE

- SPEDE: System Programmers Educational Development Environment
- Developed at CSUS (Dr. John Clevenger; Brian Witt, and others)
- Originally developed for Motorola CPUs
- Ported to the Intel 80386 CPU where it has matured for many years
- Has been the standard environment for 159 over the years

INTRODUCTION TO SPEDE

- SPEDE: System Programmers Educational Development Environment
- Two Virtual Machines:
 - SPEDEHost
 - Our development, compilation, and debugging environment
 - Modern ubuntu Linux with various tools, editors, and more
 - SPEDETarget
 - MS DOS based environment with a bootloader to launch our “operating system” kernel

WHY SPEDE?

- SPEDE provides a framework that consists of:
 - a set of basic functionalities to get up and running *fast*
 - c libraries, compiler, debugger
 - a bootloader to load our operating system
- SPEDE traditionally ran on physical hardware but also runs within a Virtual Machine environment
 - Keeps it portable and accessible

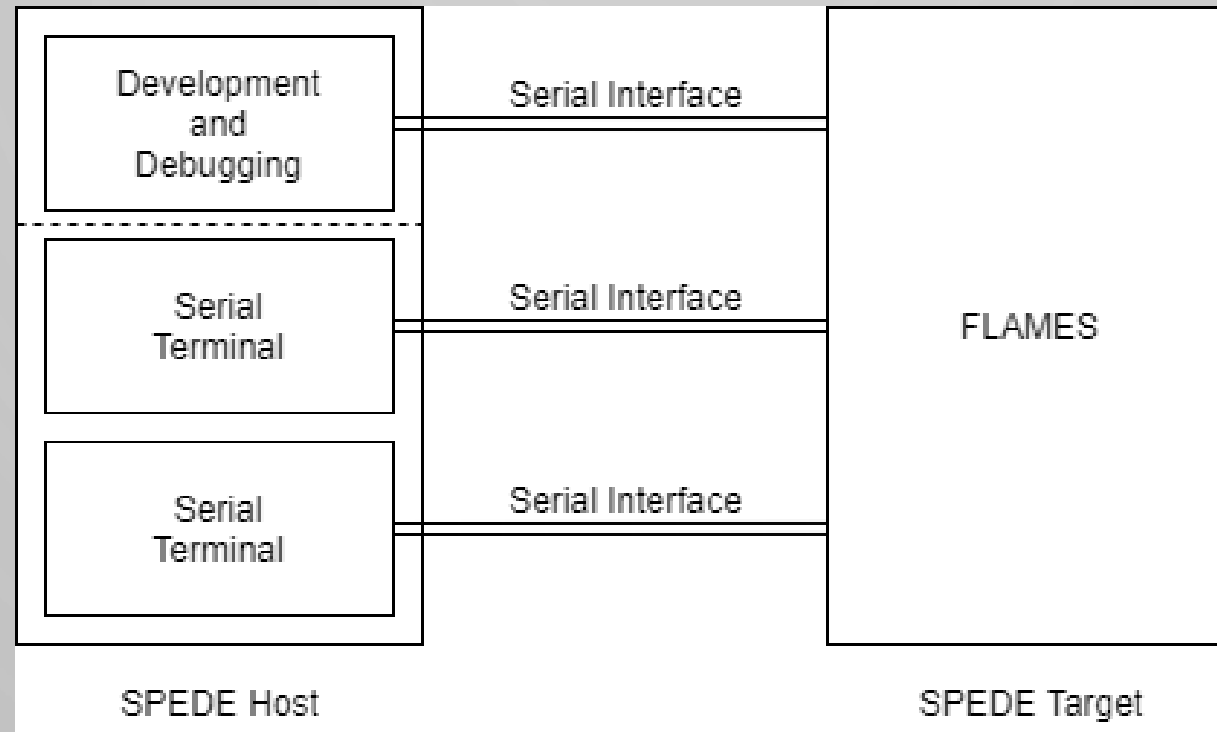
A PHYSICAL DEVELOPMENT ENVIRONMENT



THE SPEDE VIRTUAL MACHINES

- Since we are in a fully-remote environment, we will be using a set of virtual machines exclusively to develop our operating system
- SPEDEHost
 - Our development, compilation, and debugging environment
 - Modern ubuntu Linux with various tools, editors, and more
- SPEDETarget
 - MS DOS based environment with a bootloader to launch our “operating system” kernel

OUR VIRTUAL ENVIRONMENT



THE SPEDEHOST SYSTEM

- Includes general tools and utilities
- Includes a compiler to build an image for our target system
- Includes a debugger (GDB) to perform live debug on our target system
- Includes the SPEDE/FLAMES header files, libraries, and source code
- Includes the SPEDE/FLAMES “flash programmer” to download our operating system image

THE SPEDETARGET SYSTEM

- Includes the FLAMES bootloader
- Includes the FLAMES runtime environment
- Otherwise, full “hardware” access is available!

BASIC REQUIREMENTS

- Computer with the ability to run VirtualBox Virtual Machines
 - Generally: “modern” x86 (Intel or AMD) CPU; 2GB+ RAM; 10GB+ Disk Space
- Please communicate ASAP if this is NOT possible for you!

GETTING STARTED

- Your first assignment will be to get up and running with the SPEDE environment
 - Downloading and installing the SPEDE virtual machines
 - Loading a pre-built demo image from the SPEDE Host to the SPEDE Target
 - Modifying and compiling code to be run on the SPEDE Target
 - Walk through the process of debugging code that is running on the SPEDE Target
- This is an individual assignment!
 - You need to know how to develop, run, and debug to support your team