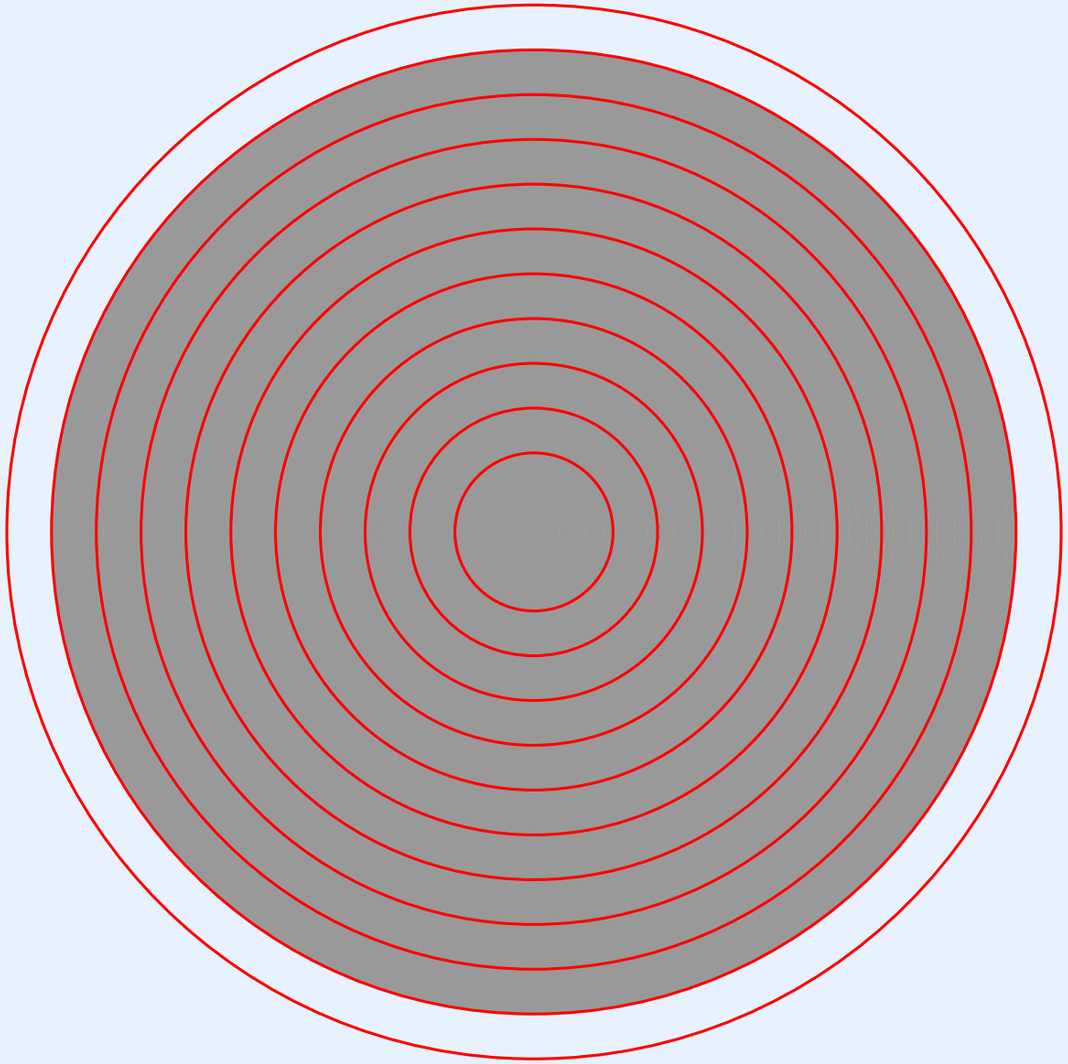
Module XXII

System Initialization

Copyright © 2025 by Douglas Comer and CRC Press, Inc. All rights reserved.

Relation of Initialization To The Hierarchy



Copyright © 2025 by Douglas Comer and CRC Press, Inc. All rights reserved.

Starting A Computer From Scratch

- General idea: the OS is *not* the first piece of software that runs
- A typical boot scenario at power-up
  - The hardware performs some basic initialization
  - The fetch-execute cycle begins executing code in flash memory or ROM
  - ROM code completes hardware checks and hardware initialization
  - ROM code identifies a boot device, finds an executable image, and loads a copy into memory
  - ROM code sets the hardware registers for kernel mode and physical address space
  - ROM code branches to entry point of the image

Copyright © 2025 by Douglas Comer and CRC Press, Inc. All rights reserved.

The Initial Image

- Is *not* usually an operating system
- Is known as a *bootstrap* program and has the following capabilities
  - Knows about some devices, and contains code to use them
  - Is configured with a set of devices to try
- An example bootstrap strategy
  - If the hard disk contains an executable image, load and run it
  - If an external USB drive contains a bootable image, load and run it
  - If the Ethernet card can boot over the network, try a network boot
  - If the above fails, display a message for the user and halt

Copyright © 2025 by Douglas Comer and CRC Press, Inc. All rights reserved.

An Example That Illustrates Bootstrap Complexity

- The Xinu lab contains Galileo Boards
- When a Galileo board boots, it must download a Xinu image over the network
- Bad news: the Galileo bootstrap system does not perform network bootstrap automatically
- Apparently good news: the hardware includes a configurable bootstrap program named *grub*
- Bad news: the version of grub on the board cannot be reconfigured to boot over a network
- Good news: the Galileo bootstrap *can* be configured to boot a file from the internal flash storage

Copyright © 2025 by Douglas Comer and CRC Press, Inc. All rights reserved.

Our Solution

- Place a more advanced version of *grub* on the internal flash that uses *multiboot* to check several devices
- Configure the Galileo bootstrap to run multiboot grub from the internal flash
- Bad news: even multiboot *grub* cannot use the network adapter
- Good news: multiboot grub can boot a program from an SD card plugged into the board
- The solution
  - Write our own network download program
  - Put the program on the SD card
  - Configure multiboot grub to run the program

Copyright © 2025 by Douglas Comer and CRC Press, Inc. All rights reserved.

Our Solution
(continued)

- We named our download program *xboot*)
- *Xboot*
  - Contacts a server in the lab
  - Uses TFTP to download a file and place it in memory
  - Branches to the entry point of the program

Copyright © 2025 by Douglas Comer and CRC Press, Inc. All rights reserved.

The Cute Twist

- The xboot program requires
  - A device driver for the Ethernet device
  - The code to handle basic network protocols: ARP, IP, and UDP
  - A way to handle network packets asynchronously (e.g., a network input process)
- We already had all the necessary pieces in Xinu
- The twist: *xboot* is actually a version of Xinu in which the main program performs the download (i.e., we reused working protocol software)

Copyright © 2025 by Douglas Comer and CRC Press, Inc. All rights reserved.

Memory Occupancy During Bootstrap

- Where should a bootstrap program reside?
- If the goal is to boot an operating system, the bootstrap program cannot occupy the locations that the OS will occupy
- Two approaches have been used
  - Self-relocating code: the bootstrap starts in the standard location, but moves a copy of itself to high memory and then branches to the copy
  - The bootstrap is bound to high memory addresses: the bootstrap program is compiled and linked to run at a high memory address (beyond the memory into which the operating system is loaded)
- Note: self-relocating code requires address constants to be relocated
- *Xboot* uses the second approach

Copyright © 2025 by Douglas Comer and CRC Press, Inc. All rights reserved.

Operating System Startup After Initial Bootstrap

Steps An Operating System Takes When It Starts

- Perform initialization required by the hardware platform
- Initialize the memory management hardware and the free memory list
- Initialize each operating system module
- Load (if not present) and initialize a device driver for each device
- Start (or reset) each I/O device
- Transform from a sequential program to a concurrent system
- Create a null process
- Create a process to execute user code (e.g., a desktop)

Copyright © 2025 by Douglas Comer and CRC Press, Inc. All rights reserved.

When Xinu Has Been Loaded Into Memory And Begins Running

- Startup code written in assembly language performs the following
  - Initializes the processor hardware
  - Initializes the bus
  - Zeroes the bss segment
  - Initializes the co-processor, if one is present
  - Creates an environment suitable for C (e.g., sets the stack pointer to a valid location)
  - Invokes the C initialization function, *nulluser*
  - *Nulluser* invokes function *sysinit* (also written in C)

Copyright © 2025 by Douglas Comer and CRC Press, Inc. All rights reserved.

The Sysinit Function

- Performs any remaining platform initialization
- Initializes the memory management hardware and the free list
- Initializes each I/O device and its driver
- Initializes operating system modules
- Transforms from a sequential program to a concurrent system
- Arranges for the current computation to become the null process
- Enables interrupts
- Creates a new process to execute *main*

Copyright © 2025 by Douglas Comer and CRC Press, Inc. All rights reserved.

Transforming From A Program To A Concurrent System

- Is the most significant aspect of initialization
- Occurs in Xinu function *sysinit*
- Is surprisingly simple and elegant
- Allows fetch-execute to continue
- Does not involve any change in code or special instructions
- Really only changes the way we view the system

Copyright © 2025 by Douglas Comer and CRC Press, Inc. All rights reserved.

Transforming From A Program To A Concurrent System (continued)

- Steps required
  - Fill in the process table entry for process 0
  - Make the state *PRCURR* and the priority zero (the lowest in the system)
  - Set *currpid* to zero
  - Create and resume a process for main program
- When *resume* is called
  - Resched will choose the new process
  - A context switch will proceed as usual
- Suddenly, a concurrent system is executing!

Copyright © 2025 by Douglas Comer and CRC Press, Inc. All rights reserved.

Transforming From A Program To A Concurrent System (continued)

- To summarize

After it fills in the process table entry for process zero, the code sets variable currpid to zero, which transforms the sequential program into a process.

Copyright © 2025 by Douglas Comer and CRC Press, Inc. All rights reserved.

Perspective

- Startup involves *many* low-level, boring hardware details, and getting the startup code to work can be frustrating
- When an operating system starts, it runs as a sequential program
- Instead of creating a separate concurrent system and then jumping to it, the code declares that the null process is running and invokes *create* to create a process running the main function
- The point is that if operating systems functions have been designed carefully, changing from a sequential code to a concurrent system is both surprisingly straightforward and elegant

Copyright © 2025 by Douglas Comer and CRC Press, Inc. All rights reserved.

Questions?