ECE353 Lectures

Hanhee Lee

January 6, 2025

Contents

1 Review **3**

1.1 Converting Between Binary, Hexadecimal, and Decimal . . . 3

1.2 Little-endian and Big-endian . . . 3

1.3 Memory . . . 4

2 Why Systems Software? **5**

2.1 Three OS Concepts . . . 5

2.2 OS Manages Resources . . . 5

2.3 Program . . . 5

2.4 Process (Abstraction) . . . 5

2.4.1 Basic Requirements for a Process . . . 5

2.5 Process (Abstraction) . . . 5

2.5.1 Static . . . 5

2.5.2 Motivation for Virtualization . . . 5

2.5.3 Does the OS allocate different stacks for each process? . . . 6

2.5.4 What about global variables? . . . 6

2.5.5 Potential Memory Layout for Multiple Processes . . . 6

3 Kernels **7**

3.1 File Descriptor (Abstraction) . . . 7

3.1.1 System Calls Make Requests to the Operating System . . . 7

3.1.2 API Tells You What and ABI Tells You How . . . 7

3.2 Programs on Linux Use the ELF Filer Format . . . 8

3.2.1 Bytes Represent an ELF File . . . 8

3.3 Kernels . . . 8

3.4 System Calls Transition Between User and Kernel Mode . . . 8

3.5 System Calls Are Traceable . . . 8

Definition:

Process:

Motivation:

Derivation:

Warning:

Summary:

Algorithm:

Example:

FAQ:

1 Review

1.1 Converting Between Binary, Hexadecimal, and Decimal

Process:

1. **Binary to Decimal:**
   (a) Write down the binary number.
   (b) Assign place values, starting from 2^0 on the rightmost digit.
   (c) Multiply each binary digit by its corresponding power of 2.
   (d) Add all the results together to get the decimal equivalent.
2. **Decimal to Binary:**
   (a) Divide the decimal number by 2.
   (b) Record the remainder (0 or 1).
   (c) Repeat the division process with the quotient until the quotient is 0.
   (d) Write the remainders in reverse order to obtain the binary equivalent.
3. **Binary to Hexadecimal:**
   (a) Group the binary number into groups of 4 digits, starting from the right. Add leading zeros if necessary.
   (b) Convert each 4-digit binary group to its hexadecimal equivalent using the binary-to-hex mapping (e.g., $0000 = 0$, $0001 = 1$, $1110 = E$).
   (c) Combine the hexadecimal digits to get the hexadecimal equivalent.
4. **Hexadecimal to Binary:**
   (a) Write down each hexadecimal digit.
   (b) Replace each hexadecimal digit with its 4-bit binary equivalent.
   (c) Combine the binary groups to get the binary equivalent.
5. **Decimal to Hexadecimal:**
   (a) Divide the decimal number by 16.
   (b) Record the remainder as a hexadecimal digit (0–9 or A–F).
   (c) Repeat the division process with the quotient until the quotient is 0.
   (d) Write the remainders in reverse order to obtain the hexadecimal equivalent.
6. **Hexadecimal to Decimal:**
   (a) Write down the hexadecimal number.
   (b) Assign place values, starting from 16^0 on the rightmost digit.
   (c) Multiply each hexadecimal digit by its corresponding power of 16, converting any letters (A–F) to decimal values (A=10, B=11, etc.).
   (d) Add all the results together to get the decimal equivalent.

1.2 Little-endian and Big-endian

Definition:

- **Little-endian:** In the little-endian format, the least significant byte (LSB) of a multi-byte data value is stored at the lowest memory address, and the most significant byte (MSB) is stored at the highest memory address.
- **Big-endian:** In the big-endian format, the most significant byte (MSB) of a multi-byte data value is stored at the lowest memory address, and the least significant byte (LSB) is stored at the highest memory address.

Example:

- For example, the hexadecimal value `0x12345678` would be stored in memory as:

  `78 56 34 12`

- For example, the hexadecimal value `0x12345678` would be stored in memory as:

  `12 34 56 78`

1.3 Memory

Summary: Table, int*, &a, int**a, *a, int[5], etc.

2 Why Systems Software?

Summary:
-

2.1 Three OS Concepts

Definition:
1. **Virtualization:** Share one resource by mimicking multiple independent copies.
2. **Concurrency:** Handle multiple things happening at the same time.
3. **Persistence:** Retain data consistency even without power.

2.2 OS Manages Resources

Definition: Insert picture.

2.3 Program

Definition: A file containing all the instructions and data required to run.

2.4 Process (Abstraction)

Definition: An instance of running a program.

2.4.1 Basic Requirements for a Process

Definition: Insert picture w/ virtual memory.

2.5 Process (Abstraction)

2.5.1 Static

Definition: Only able to use the global variable in the current C file.

2.5.2 Motivation for Virtualization

Motivation: How to run two different programs at the same time? Insert code.
- Was the address of local the same b/w 2 processes? Different address in physical memory b/w different processes.
- Was the address of global the same b/w 2 processes? Same address in physical memory b/w different processes, but uses virtual memory.
- What else may be needed for a process?

Warning: Local variables are stored on the stack.

2.5.3 Does the OS allocate different stacks for each process?

Definition: The stacks for each process need to be in physical memory. One option is the operating system just allocates any unused memory for the stack.

-

2.5.4 What about global variables?

Definition: The compiler needs to pick an address (random) for each variable when you compile.

- What if we had a global registry of addresses? Impossible (too much space and know memory addresses ahead of time).

2.5.5 Potential Memory Layout for Multiple Processes

Definition: Insert picture.

Warning: Process 1 wants to use more memory than its allocated.

3 Kernels

Summary:
- The kernel is the part of the operating system (OS) that interacts with hardware (it runs in kernel mode).
- System calls are the interface between user and kernel mode:
  - Every program must use this interface!
- File format and instructions to define a simple "Hello world" (in 168 bytes):
  - Difference between API and ABI.
  - How to explore system calls.
- Different kernel architectures shift how much code runs in kernel mode.

FAQ:
- What is difference b/w printf and write?

3.1 File Descriptor (Abstraction)

Motivation: Since our processes are independent, we need an explicit way to transfer data.

Definition:
1. **IPC:** Inter-process communication is transferring data b/w two processes.
2. **File Descriptor:** A resource that users may either read bytes from or write bytes to (identified by an index stored in a process).
   - e.g. File or terminal.

3.1.1 System Calls Make Requests to the Operating System

Definition:
```
ssize_t write(int fd, const void *buf, size_t count);
Description: writes bytes from a byte array to a file descriptor
    fd    - the file descriptor
    buf   - the address of the start of the byte array (called a buffer)
    count - how many bytes to write from the buffer

void exit_group(int status);
Description: exits the current process and sets an exit status code
    status - the exit status code (0-255)
```

Example: Hypothetical "Hello World" Program
```c
void _start(void) {
    write(1, "Hello world\n", 12);
    exit_group(0);
}
```

Warning: System calls uses registers, while C is stack based.

3.1.2 API Tells You What and ABI Tells You How

Definition:
- Application Programming Interface (API) abstracts the details and describes the arguments and return value of a function.
  - e.g. A function takes 2 integer arguments
- Application Binary Interface (ABI) specifies the details, specifically how to pass arguments and where the

return value is.
   - e.g. The same function using the C calling convention (arguments on the stack)

3.2 Programs on Linux Use the ELF Filer Format

Definition: Executable and Linkable Format (ELF) specifies both executables and libraries
- Always starts with the 4 bytes: 0x7F 0x45 0x4C 0x46 or with ASCII encoding: DEL 'E' 'L' 'F'
- These 4 bytes are called "magic", and that's how you know what kind of file this is (other file formats may have a different number of bytes)

3.2.1 Bytes Represent an ELF File

Definition:

3.3 Kernels

Definition:
- **Kernel mode** is a privilege level on your CPU that gives access to more instructions.
- The kernel is the part of your operating system that runs in kernel mode.
- These instructions allow only trusted software to interact with hardware:
   - e.g., only the kernel can manage virtual memory for processes.

3.4 System Calls Transition Between User and Kernel Mode

Definition:

3.5 System Calls Are Traceable