ECE353 Lectures

Hanhee Lee

January 9, 2025

Contents

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., $0000 = 0$, $0001 = 1$, $1110 = \text{E}$).
   (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 Useful Terminal Commands

Summary:

- `./hello-world-linux-aarch64` to run hello world.
- `readelf -a <FILE>` to see the ELF header.
- `strace <PROGRAM>` to trace all the system calls a process makes on Linux.

2.2 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.

2.3 OS Manages Resources

Definition: Insert picture.

2.4 Program

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

2.5 Process:

Definition: An instance of running a program.

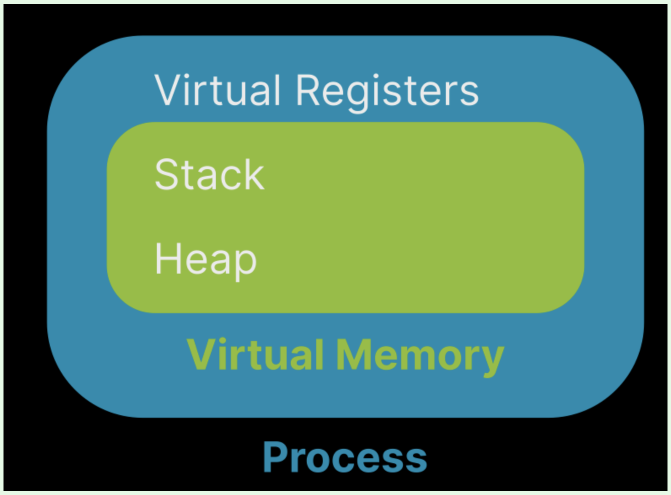


Figure 1: Process

2.5.1 Basic Requirements for a Process

Definition: Insert picture w/ virtual memory.

2.6 Process (Abstraction)

2.6.1 Static

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

2.6.2 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.

2.6.3 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.

-

2.6.4 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).

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.
   - e.g. 0 is standard input, 1 is standard output, and 2 is standard error.

3.2 System Calls

Definition: System calls are the interface b/w user and kernel mode.

3.2.1 System Calls Make Requests to the Operating System

Definition:

```c
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

```c
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.

3.3 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.
- Application Binary Interface (ABI) specifies the details, specifically how to pass arguments and where the return value is.

3.4 Magic

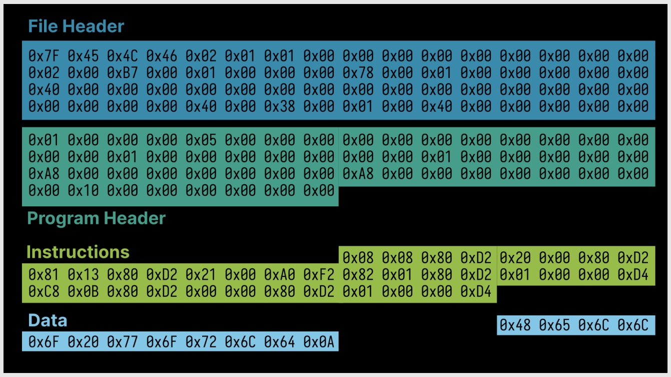
Definition: The "magic bytes" refer to the first 4 bytes of a file that uniquely identify the file format.

3.4.1 Programs on Linux Use the ELF File 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’

Example: Hello World ELF File

1. **168 Byte Program:**
   - Tells the OS to load the entire executable file into memory at address `0x10000`.
   - The file header is 64 bytes, and the “program header” is 56 bytes (120 bytes total).
   - The next 36 bytes are instructions, followed by 12 bytes for the string:
     - `"Hello world\n"`
     - Instructions start at `0x10078` (`0x78` is 120).
     - The string (data) starts at `0x1009C` (`0x9C` is 156).



```
File Header
0x7F 0x45 0x4C 0x46 0x02 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x02 0x00 0xB7 0x00 0x01 0x00 0x00 0x00 0x78 0x00 0x01 0x00 0x00 0x00 0x00 0x00
0x40 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x40 0x00 0x38 0x00 0x01 0x00 0x40 0x00 0x00 0x00 0x00 0x00

0x01 0x00 0x00 0x00 0x05 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00
0xA8 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xA8 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x10 0x00 0x00 0x00 0x00 0x00 0x00
Program Header

Instructions                            0x08 0x08 0x80 0xD2 0x20 0x00 0x80 0xD2
0x81 0x13 0x80 0xD2 0x21 0x00 0xA0 0xF2 0x82 0x01 0x80 0xD2 0x01 0x00 0x00 0xD4
0xC8 0x0B 0x80 0xD2 0x00 0x00 0x80 0xD2 0x01 0x00 0x00 0xD4

Data                                                        0x48 0x65 0x6C 0x6C
0x6F 0x20 0x77 0x6F 0x72 0x6C 0x64 0x0A
```

Figure 2: ELF File Division

2. **C Program:** Takes 500 bytes.
3. **Python Program:** Takes 2000 bytes.
4. **Java Program:** Takes 2000000 bytes.

3.5 Kernel

Definition: Kernel is a core part of the operating system that interacts with hardware that runs in kernel mode.

3.5.1 Kernel Mode

Definition: Kernel mode is a privilege level on your CPU that gives access to more instructions.

3.5.2 Levels of Privelege

Definition:

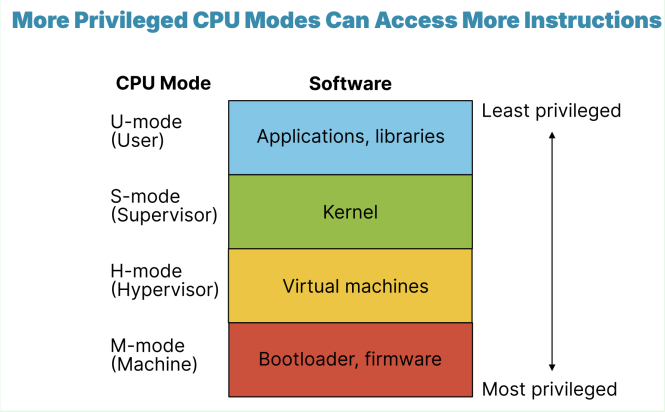
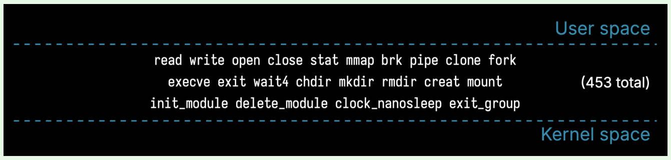


Figure 3: Levels of Privelege

3.5.3 System Calls Transition Between User and Kernel Mode

Definition:



```
read write open close stat mmap brk pipe clone fork
execve exit wait4 chdir mkdir rmdir creat mount
init_module delete_module clock_nanosleep exit_group
```

Figure 4: System Calls Transition

3.5.4 Different Tpyes of Kernel Architectures

Definition:

- **Monolithic Kernel:** All the services are in the kernel.

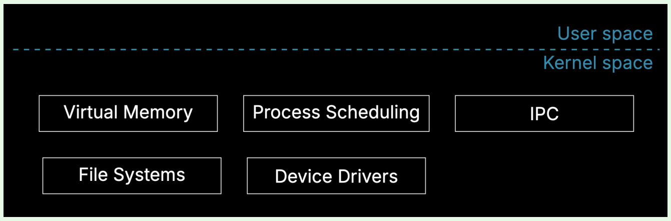


Figure 5: Monolithic Kernel

- **Microkernel:** Only the essential services are in the kernel.

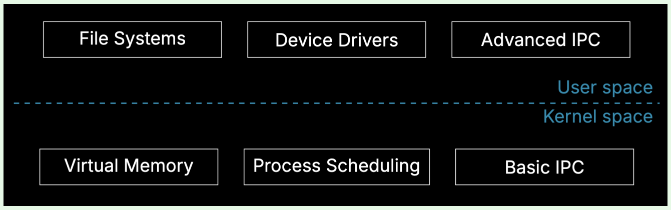


Figure 6: Microkernel

- **Hybrid Kernel:** A mix of monolithic and microkernel.
- **Nanokernel and picokernel:** Even smaller services than microkernel.

Warning: Short answer question.

4 Libraries

4.1 What is an Operating System?

Definition: An operating system consists of a kernel and libraries required for your application.

Linux distributions may be considered GNU/Linux, where GNU distributes the standard C library and common utilities.

Warning: OS have different libraries for different applications.

4.2 Normal Compilation in C

4.3 Static Libraries and Dynamic Libraries

4.3.1 Comparison of Static and Dynamic Libraries