A Tour of Computer Systems

CSE 220: Systems Programming

Ethan Blanton & Carl Alphonce

Department of Computer Science and Engineering

University at Buffalo

Concept versus Implementation

The C language and POSIX are implementations of systems.

There are many possible implementations.

Certain conceptual considerations are presented by underlying architecture.

We will look at some of those concepts.

Understanding How Things Work

"Why do I need to know this stuff?"

Abstraction is good, but don't forget reality!

Most CS courses emphasize abstraction

- Abstract data types
- Asymptotic analysis

These abstractions have limits

- Sometimes you need to understand the underlying implementation
- Sometimes the abstract interfaces are not as flexible or performant as you need
- Sometimes there are bugs

Effective Questions

Answering questions is easy.

Asking the right question is hard!

For conceptual problems, ask:

- What do I have?
- What do I need?
- Can I figure out how to get from here to there?
- What about partway there?

If you need help, tell us (in your words!) what you have and what you need!

Numeric Representations

`int`s are not integers, `float`s are not real numbers!

Example 1: Is $x^2 \geq 0$?

- `float`: yes!
- `int`: well …
  - `40000 * 40000` $\rightarrow$ `1600000000`
  - `50000 * 50000` $\rightarrow$ `???`

Example 2: Is $(x + y) + z = x + (y + z)$?

- `int`: yes!
- `float`:
  - `(1e20 + -1e20)+ 3.14` $\rightarrow$ `3.14`
  - `1e20 + (-1e20 + 3.14)` $\rightarrow$ `???`

Computer Arithmetic

Computer operations do have mathematical properties.

However, you cannot assume all usual mathematical properties!

- Finite representations cause various effects
- Integer operations satisfy ring properties:
  - Commutativity, associativity, distributivity
- Floating point operations satisfy ordering properties:
  - Monotonicity, sign values

You must understand which abstractions apply where.

These are important issues for compiler writers, systems programmers, serious application programmers.

Assembly Language

You need to know assembly.
You'll see it next in CSE 341!

You'll probably never write programs in assembly.
(Compilers are better at it and much more patient than you are!)

Understanding assembly is key to understanding the machine.

Where Will I Use Assembly?

Understanding the behavior of programs in the presence of bugs

- High-level language models break down

Tuning program performance

- Understand optimizations the compiler can and cannot do
- Understand sources of program inefficiency

Implementing system software

- Compilers target assembly
- Operating systems manage hardware state

Creating and fighting malware

- Most malware is in x86 assembly!

Memory Management and Layout

Memory matters.

Memory is not unbounded!

- It must be allocated and managed
- Many applications are memory-dominated

Memory referencing bugs are especially pernicious

- Their effects may be distant in both time and space

Memory performance is not uniform

- Cache and virtual memory effects can affect program performance
- Adapting programs to the memory system can have major speed implications

Why Memory Performance Matters

```
void copyij(int src[2048][2048],
            int dst[2048][2048]) {
    for (int i = 0; i < 2048; i++) {
        for (int j = 0; j < 2048; j++)
           {
            dst[i][j] = src[i][j];
        }
    }
}
```

3.8 ms

```
void copyji(int src[2048][2048],
            int dst[2048][2048]) {
    for (int j = 0; j < 2048; j++) {
        for (int i = 0; i < 2048; i++)
           {
            dst[i][j] = src[i][j];
        }
    }
}
```

72.2 ms

All that changed is the order of the loops!

Therac-25

An infamous accident in software engineering: Therac-25

https://medium.com/swlh/software-architecture-therac-25-the-killer-radiation-machine-8a05e0705d5b

- People died.
- Arithmetic bugs were involved.
- Poorly understood copied code was involved.

  (Stack Overflow kills!)

Toyota Acceleration

Some Toyota vehicles experienced unintended acceleration in the late 2000s.

- Toyota was fined 1.2 billion dollars
- ~9 million vehicles were recalled

Expert analysis identified:

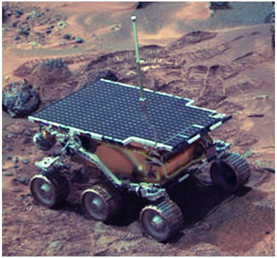
- Memory corruption from software bugs
- Copied code ("Stack overflow …bugs led to memory corruption")

`https://users.ece.cmu.edu/~koopman/pubs/koopman14_toyota_ua_slides.pdf`

Mars Pathfinder

The Pathfinder Mars rover frequently stopped responding.

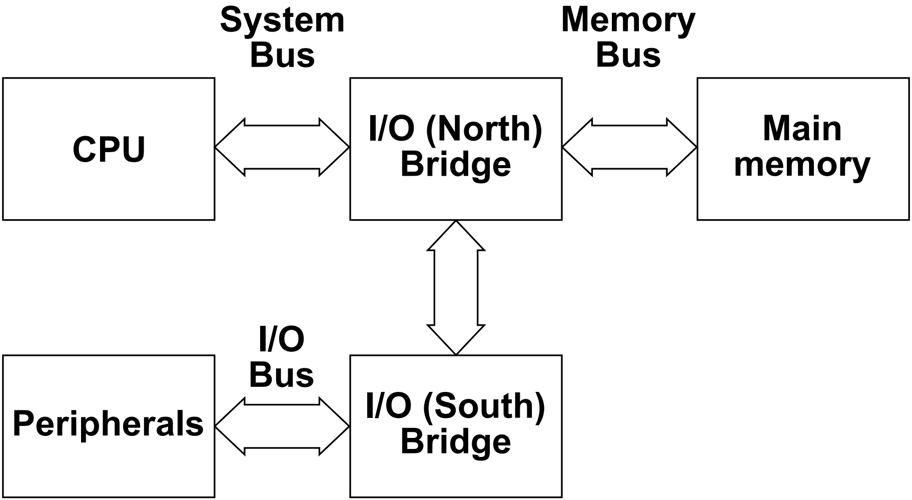
- The problem was system scheduling
- Low-level debugging identified the issue
- Testing could have identified the problem on the ground



(Credit: NASA)

https://www.rapitasystems.com/blog/what-really-happened-to-the-software-on-the-mars-pathfinder-spacecraft

A Bit About Architecture



Buses

A bus has a width, which is literally the number of wires it has.¶

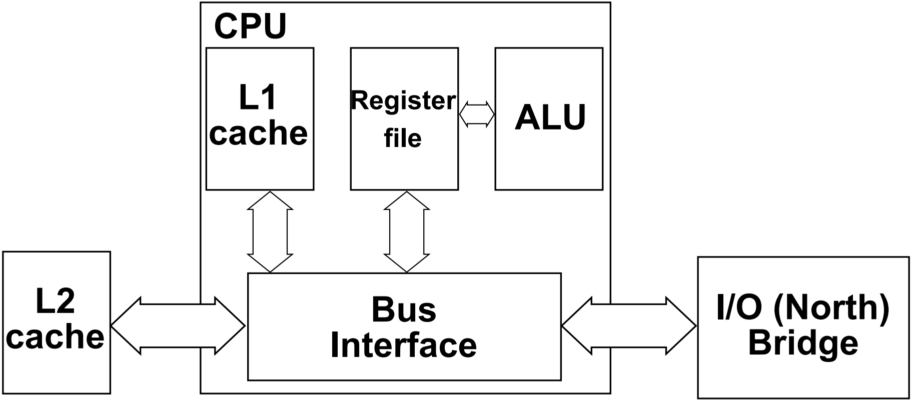
(This is a little less clear on a serial bus, where the width is a protocol convention.)

Each wire transmits one bit per transfer.

Every bus transfer is of that width, though some bits may be ignored.

Therefore, memory has a word size from the view of the CPU: the number of wires on that bus.

A Modern CPU


CPU
L1 cache
Register file
ALU
L2 cache
Bus Interface
I/O (North) Bridge

CPU Properties

Both internal and external busses have fixed widths.

A small number of storage locations called registers:

- Have very fast access time¶
- Have a fixed width
- Are fixed in number

The ALU performs computation.

- It may be able to access only registers
- It may be able to access memory
- It may have arbitrary restrictions

CPU ↔ Memory Transfer

The CPU fetches data from memory in words the width of the memory bus.

It places those words in registers the width of a cpu word.

This register width is the native integer size.[1]

These word widths may or may not be the same.

If they're not, a transfer may require:

- multiple registers, or
- multiple memory transfers.

[1] Some CPUs (including x86-64) can manipulate more than one size of integer in a single register.

Imposing Structure on Memory

That said, programming languages expose things like:

- Booleans
- classes
- strings
- structures

How is that?

We impose meaning on words in memory by convention.

E.g., as we saw before, a C string is a sequence of bytes that happen to be adjacent in memory.

Summary

- Architectural details matter
  - Bus widths
  - Numeric properties
  - Performance details
- C and POSIX are just one possible system
- All systems have those details
- Software correctness can be critically important

Next Time ...

- Memory allocation
- The program heap

References I

Required Readings

[1] Randal E. Bryant and David R. O'Hallaron. *Computer Science: A Programmer's Perspective*. Third Edition. Chapter 1: Intro, 1.1–1.7. Pearson, 2016.

License