SYSTEMS ENGINEERING · WEEK 3 · STATION III

Emulation

Mipster as the universal machine, the price of interpretation, and self-execution.

THE LOOP

Fetch, decode, execute, repeat. A page of C* per instruction group — and it runs every program for the machine.

```
// mipster's core, in spirit
while (1) {
fetch(); // ir = memory[pc]
decode(); // opcode, registers, immediate
execute(); // one of fourteen cases
// exceptions and timeouts stop the loop
}
```

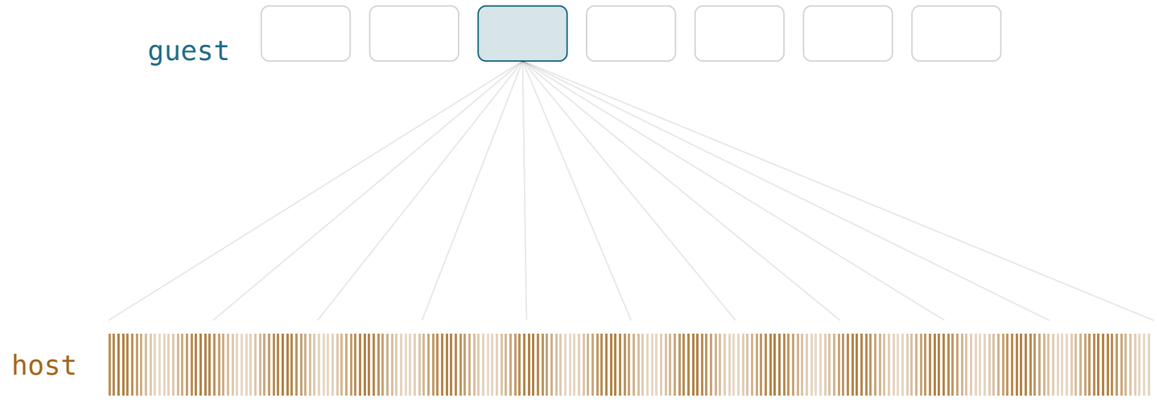
The machine chapter's cycle as a program. The guest's code is data in the host's memory, read one word at a time and never executed by the host's processor. Fourteen cases in `execute`, each a few lines of arithmetic on the context's registers.

This is Turing's universal machine: one program that becomes any program when handed its description. The introduction class proved it universal; this class runs it.

And it is an operating system: the guest cannot touch anything the emulator does not hand it. Isolation for free. No self-reference anywhere.

IN THE TERMINAL

Self-execution: the emulator runs its own machine code.



one guest instruction, about 2,600 on the host

```
$ make emu
./selfie: summary: 85754 executed instructions in total
$ make emu-emu
> selfie.m: summary: 86380 executed instructions in total
./selfie: summary: 222410917 executed instructions in total
```

Selfie printing its synopsis costs 85,754 instructions bare-metal. Hosted by mipster inside mipster it costs the guest 86,380 and the host 222 million: a factor of 2,593, one per level.

The guest cannot tell. Same output, same instruction count give or take the boot level, which selfie tells it deliberately. Emulation is semantically transparent and computationally expensive, and that sentence is the whole motivation for the next three weeks.

WHAT THE BILL BUYS

Emulation is isolation. Everything the guest does is a read in the host.

SPACE

The guest's memory is an array in the host's heap. An address the guest computes is an index the emulator checks. Out of range: an exception, handled by the emulator, never by the hardware. Spatial isolation by construction.

TIME

The emulator counts instructions and can stop after any number of them. A guest that loops forever costs the host exactly the instructions the host chooses to spend. Temporal isolation by construction.

TRUST

Nothing the guest does reaches the host's processor. The host's own isolation is never at stake. And that is the property that virtualization gives up, on purpose, for a factor of twelve.

Emulation as the *executable specification* of virtualization: whatever hypster does must be indistinguishable to the guest from what mipster does. Week 6 shows that it is.

STATION III

A universal machine can run any machine — so nothing general about the machines it runs can be decided.

The gift and the limit are the same fact. Because mipster runs every program, it cannot know whether the program it is running will stop, or divide by zero, or read outside its memory, before it happens. Rice's theorem, on the emulator.

So it does not try to know. It counts instructions and stops after a timeout; it checks each address as it is used; it catches the division when it occurs. Bounding, not deciding — the third sentence of week 1, and the design principle of every kernel.

NEXT WEEKS

Space: give each guest its own addresses and translate them, week 4. Time: the timer and the scheduler, week 5. Then run the guest on the real processor and see what breaks, week 6.

ASSIGNMENT

`self-assembler`: generate code, and assemble selfie's own assembly.

1. Extend last week's parser with code generation: for each parsed instruction, encode it with selfie's own `encode_X` procedures and emit the word into the code binary, as the compiler does.
2. Handle the data segment as the assembly file presents it, and the entry point, so that the result is a loadable binary.
3. Self-assemble. `./selfie -c selfie.c -s selfie.s`, then `./selfie -a selfie.s -o selfie.m`, and compare with `./selfie -c selfie.c -o selfie.m`. Identical bytes: a fixed point of your own.
4. `./grader/self.py self-assembler`.

WHAT THE FIXED POINT PROVES

That the assembler agrees with the compiler's backend and the disassembler, on the one input that is all of selfie. Not that it is correct — the compiler class spends a week on why. Keep the assembler; week 10 hands it to a model checker.