Binary Instrumentation with Pin

brandon lucia

(with some content adapted from Kim Hazelwood's ASPLOS 2008 tutorial)

What is “instrumentation”?

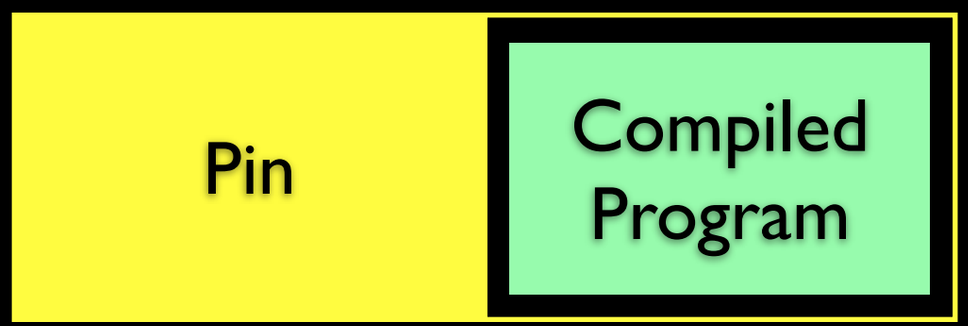
```
void f(){
  printf(“in f()\n”);
  //function body
  ...
}
```

Source Instrumentation

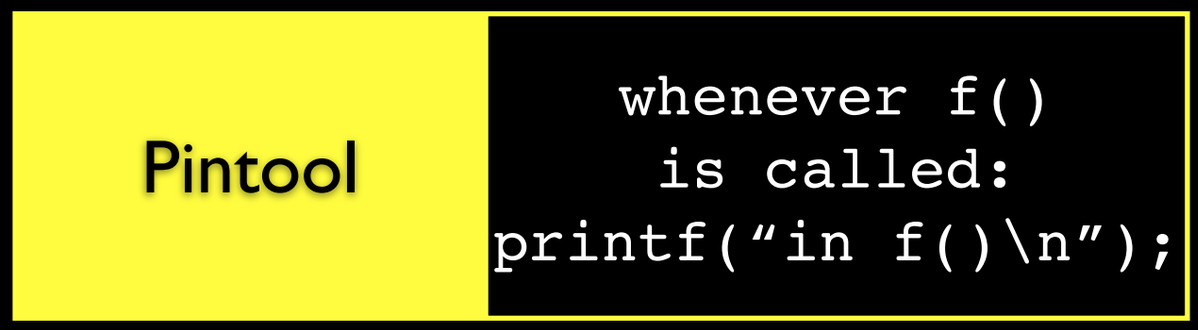
Source Program + Source Instrumentation = “in f()”

Binary Instrumentation

Pin lets us add code like before, but to binaries



+



= "in f()"

- Not necessary to recompile or relink
- Code discovered at runtime
- Can write instrumentation just once

Why should you care?

- Pin gets used a lot in research
  - Trace generation
  - Cache modeling
  - Branch predictor modeling
  - Emulating speculation
  - Modeling new instructions
  - Dynamic analysis
  - Call-graph generation
  - Memory leak detection
  - Profiling
  - Race detection
  - ...

And you have a project to do this quarter.

Using Pin

Download it at http://www.pintool.org/

To use pin, run pin, passing it a "pintool"
and a program

Pin comes with a few pintools to use as examples

These are really helpful when you're building your own

Instrumentation Routine vs. Analysis Routine

- Instrumentation Routine (`Instruction()`)
  - Decorates whatever you're instrumenting (i.e.: instructions) in the way you specify
- Analysis Routine (`docount()`)
  - Is added by instrumentation routine and does the actual work

Passing arguments to Analysis Routines

- IARG_INST_PTR
- IARG_UINT32, <value>
- IARG_REG_VALUE, <register name>
- IARG_BRANCH_TARGET_ADDR
- IARG_FUNCARG_ENTRYPOINT_VALUE
- IARG_FUNCRET_EXITPOINT_VALUE
- ....tons of others (read the manual)

Instrumentation Granularity
(and related data structures)

Basic Block

```
sub eax, edx
test edx,edx
jz 0xb7fb6aa0
```

Instruction

```
mov eax, esp
```

Trace

```
jmp 0xb7fb6aa0
```

```
jmp 0xabcdabcd
```

Image
`blah.c`

Routine
`foo()`

Asking Pin questions

- INS_IsMemoryRead/Write(ins)
- INS_IsStackRead(ins)
- INS_IsAtomicUpdate(ins)
- RTN_Name(rtn)
- ...see the docs....

Being smart about instrumentation

- We want to do the *instrumenting* as infrequently as possible
    - Choose the coarsest instrumentation granularity possible
    - Traces are instrumented and cached (so trace stuff is fast!)

Doing Multi-Threaded Stuff in Pin*

- Your pintool operates *truly concurrently* on each program thread
  - In analysis routines, must use Pin's Locks
    - PIN_LOCK, GetLock(), ReleaseLock()

*Always wear a helmet when using Pin in multi-threaded apps

Really using Pin

- I have used Pin in several of my own projects
  - Deterministic Multi-Processing Simulation
  - Concurrency Error Detection
  - Fancy Cache Simulation