JIT Instrumentation - A Novel Approach to Dynamically Instrument Operating Systems

Marek Olszewski, Keir Mierle, Adam Czajkowski, and Angela Demke Brown

presented by Harald Röck

University of Salzburg, Department of Computer Sciences

3. May 2007

Introduction

- Operating Systems are growing in complexity
- Kernel instrumentation can help
- Dynamic instrumentation
    - No recompilation and no reboot
    - Debugging systemic problems
    - Feasible in production settings

Current Approach: Probe-Based

- Dynamic instrumentation tools for OSs are probe based
- Efficient on fixed lenght architectures
- Slow on variable length architectures
    - Not safe to overwrite multiple instructions
    - Must use trap instruction

Trap-based Instrumentation

Original code:

```
sub  $6c,esp
mov  $ffffe000,edx
and  esp,edx
inc  14(edx)
mov  28(edi),eax
mov  2c(edi),ebx
mov  30(edi),ebp
add  $1,eax
and  $3,eax
or  $c, eax
mov  eax,(ebx)
add  $2,ebp
or  $f, ebp
mov  ebp,4(ebx)
```

Instrumentation:

```
add  $1,count_l
adc  $0,count_h
```

Trap-based Instrumentation

Original code:

```
sub $6c,esp
mov $ffffe000,edx
and esp,edx
int314(edx)
mov 28(edi),eax
mov 2c(edi),ebx
mov 30(edi),ebp
add $1,eax
and $3,eax
or $c, eax
mov eax,(ebx)
add $2,ebp
or $f, ebp
mov ebp,4(ebx)
```

Instrumentation:

```
add $1,count_l
adc $0,count_h
```

Trap-based Instrumentation

Original code:

```
sub $6c,esp
mov $ffffe000,edx
and esp,edx
int314(edx)
mov 28(edi),eax
mov 2c(edi),ebx
mov 30(edi),ebp
add $1,eax
and $3,eax
or $c, eax
mov eax,(ebx)
add $2,ebp
or $f, ebp
mov ebp,4(ebx)
```

Trap Handler:

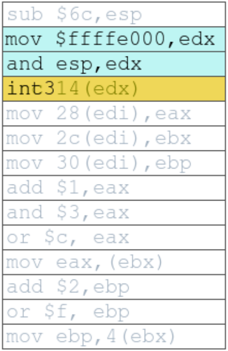
1. Save Processor state
2. Lookup which instrumentation to call
3. Call instrumentation
4. Emulate overwritten instruction
5. Restore processor state

Instrumentation:

```
add $1,count_l
adc $0,count_h
```

Trap-based Instrumentation

Original code:

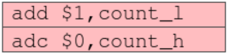


```
sub  $6c,esp
mov  $ffffe000,edx
and  esp,edx
int3 14(edx)
mov  28(edi),eax
mov  2c(edi),ebx
mov  30(edi),ebp
add  $1,eax
and  $3,eax
or  $c,  eax
mov  eax,(ebx)
add  $2,ebp
or  $f,  ebp
mov  ebp,4(ebx)
```

Trap Handler:

1. Save Processor state
2. Lookup which instrumentation to call
3. Call instrumentation
4. Emulate overwritten instruction
5. Restore processor state

Instrumentation:

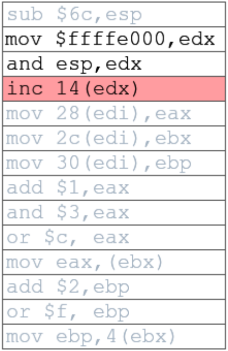


```
add  $1,count_l
adc  $0,count_h
```

Very Expensive!

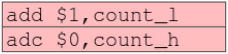
JIT Instrumentation

Original code:



```
sub  $6c,esp
mov  $ffffe000,edx
and  esp,edx
inc  14(edx)
mov  28(edi),eax
mov  2c(edi),ebx
mov  30(edi),ebp
add  $1,eax
and  $3,eax
or  $c, eax
mov  eax,(ebx)
add  $2,ebp
or  $f, ebp
mov  ebp,4(ebx)
```

Instrumentation:



```
add  $1,count_l
adc  $0,count_h
```

JIT Instrumentation

Original code:

```
sub  $6c,esp
mov  $ffffe000,edx
and  esp,edx
inc  14(edx)
mov  28(edi),eax
mov  2c(edi),ebx
mov  30(edi),ebp
add  $1,eax
and  $3,eax
or  $c,  eax
mov  eax,(ebx)
add  $2,ebp
or  $f,  ebp
mov  ebp,4(ebx)
```

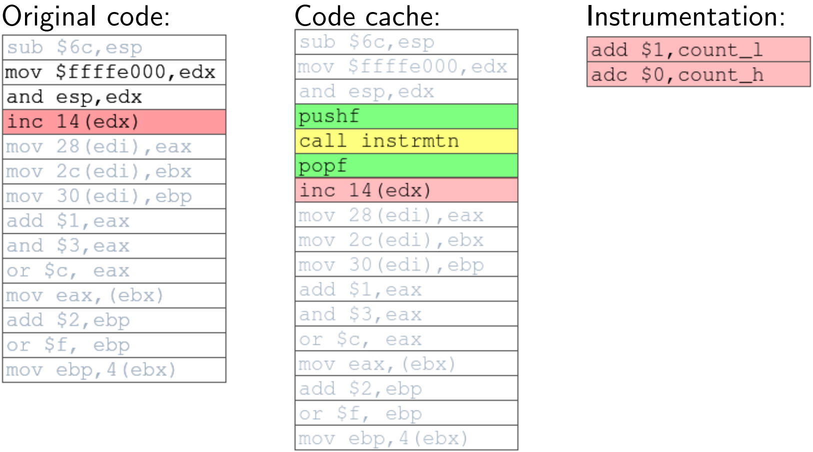
Code cache:

```
sub  $6c,esp
mov  $ffffe000,edx
and  esp,edx
inc  14(edx)
mov  28(edi),eax
mov  2c(edi),ebx
mov  30(edi),ebp
add  $1,eax
and  $3,eax
or  $c,  eax
mov  eax,(ebx)
add  $2,ebp
or  $f,  ebp
mov  ebp,4(ebx)
```

Instrumentation:

```
add  $1,count_l
adc  $0,count_h
```

JIT Instrumentation



Original code:

```
sub $6c,esp
mov $ffffe000,edx
and esp,edx
inc 14(edx)
mov 28(edi),eax
mov 2c(edi),ebx
mov 30(edi),ebp
add $1,eax
and $3,eax
or $c, eax
mov eax,(ebx)
add $2,ebp
or $f, ebp
mov ebp,4(ebx)
```

Code cache:

```
sub $6c,esp
mov $ffffe000,edx
and esp,edx
pushf
call instrmtn
popf
inc 14(edx)
mov 28(edi),eax
mov 2c(edi),ebx
mov 30(edi),ebp
add $1,eax
and $3,eax
or $c, eax
mov eax,(ebx)
add $2,ebp
or $f, ebp
mov ebp,4(ebx)
```

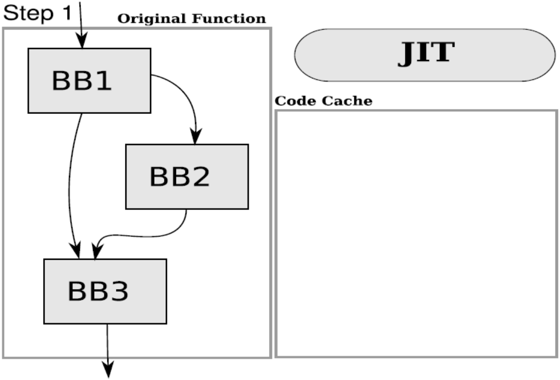
Instrumentation:

```
add $1,count_l
adc $0,count_h
```

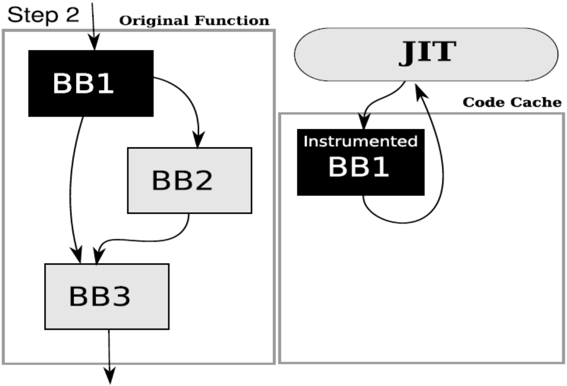
Dynamic Binary Rewriting

- Use binary rewriting to insert the new instructions
- Interleaves binary rewriting with execution
  - Performed by a runtime system
  - Typically at basic block granularity
- Code is rewritten into a *code cache*
- Rewritten code must be:
  - Efficient
  - Unaware of its new location

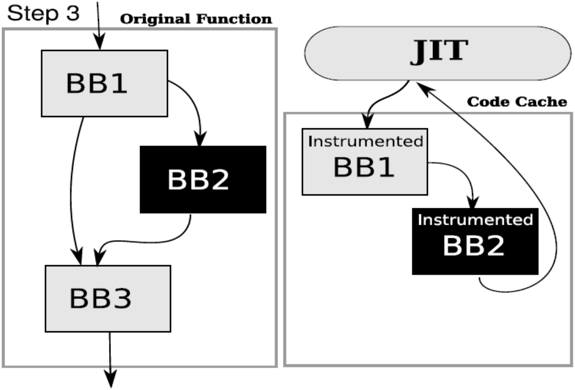
Dynamic Binary Rewriting



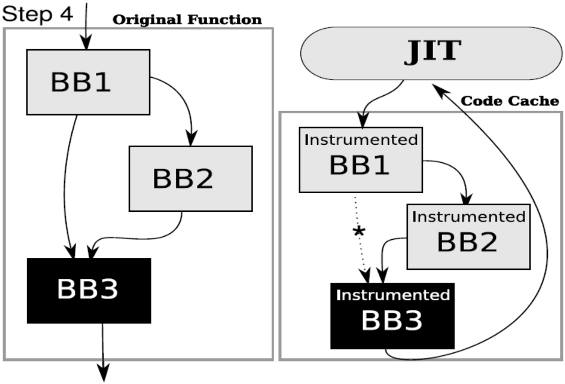
Dynamic Binary Rewriting



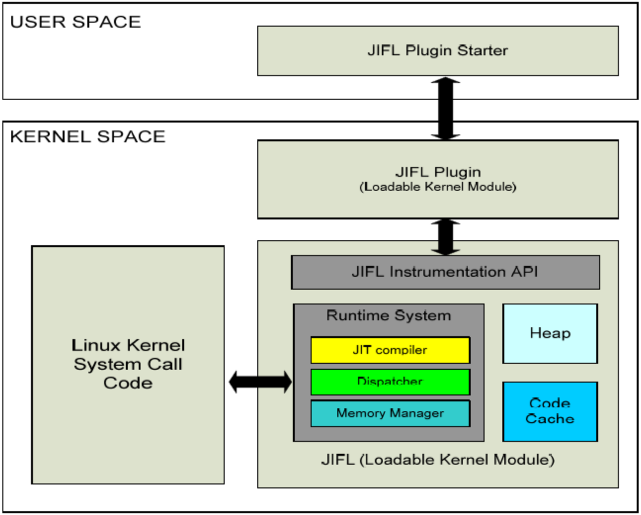
Dynamic Binary Rewriting



Dynamic Binary Rewriting



Design of the JIFL Prototype



Gaining Control

- Runtime System must gain control before it can start rewriting/instrumenting OS
- Update system call table entry to point to a dynamically emitted *entry stub*
  - Calls per-system call instrumentation
  - Calls dispatcher and passes original system call pointer

Dispatcher

- Saves registers and condition code states
- Dispatcher checks if target basic block is in code cache
  - If so it jumps to its basic block
  - Otherwise it invokes the JIT to compile and instrument the new basic block

JIT Compiler

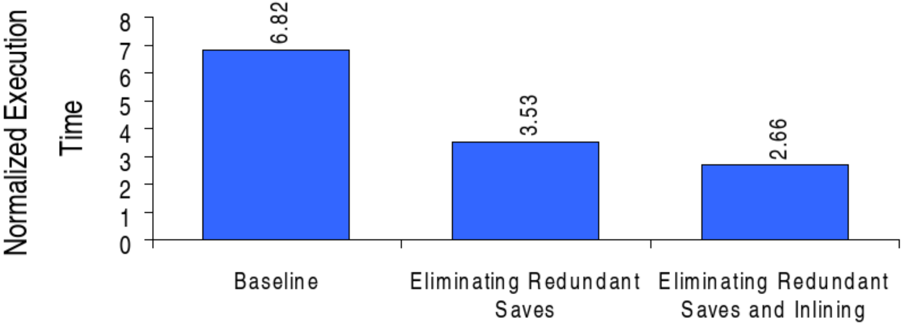
- Like convential JIT compiler, except its intput/output is x86 machine code
- Compiles at a dynamic basic block granularity
  - All but the last control flow instruction are copied directly into the code cache
  - Control flow instructions are modified to account for the new location of the code
- Communicates with the JIFL plugin to determine what instrumentation to insert
  - Insert call instruction
  - Push/Pop instrumentation parameters
  - Save/Restore volatile registers
  - Save/Restore condition code register

Optimizations

- Eliminating Redundant State Saving
- Inlining Instrumentation

Optimizations

- Eliminating Redundant State Saving
- Inlining Instrumentation



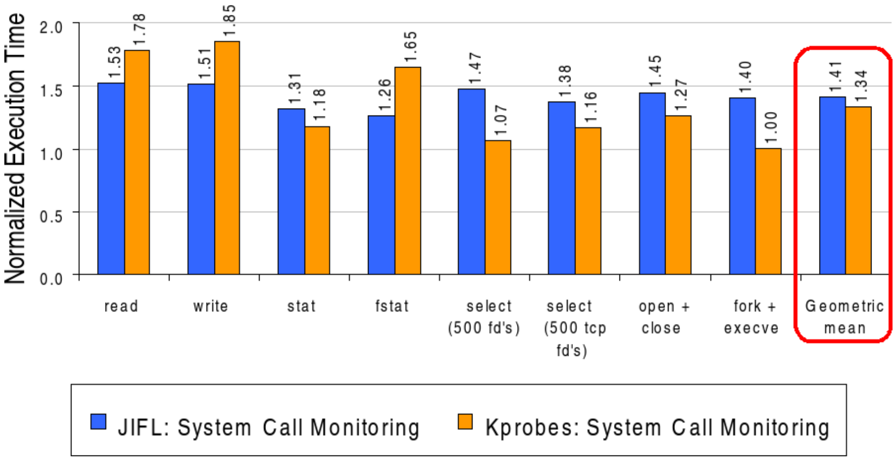
OS Issues

- Memory Allocator
    - JIT needs dynamic allocate memory
    - Linux allocator is not reentrant
    - Use own allocator on a preallocated head
- Release control
    - Calls to schedule() have to be redirected

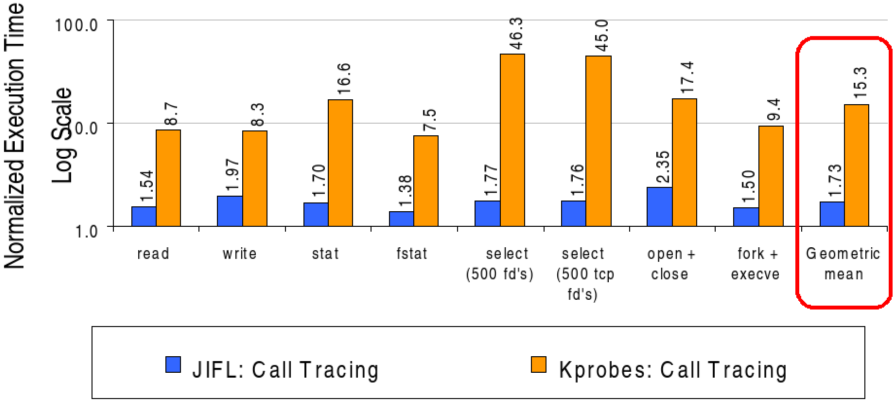
Evaluation

- JIFL vs KProbes
- Instrument every system call with three types of instrumentation
    - System Call Monitoring
    - Call Tracing
    - Basic Block Counting
- LMBench and ApacheBench2 benchmarks
- Test setup
    - 4-way Intel Pentium 4 Xeon SMP 2.8GHz
    - Linux 2.6.17.13 with SMP support and no preemption

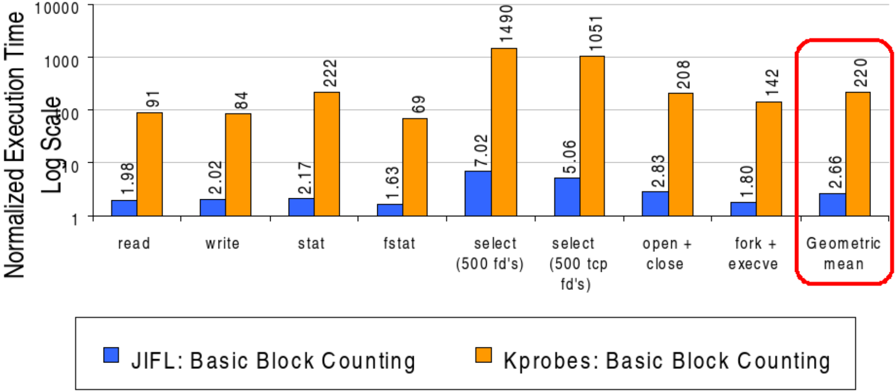
Evaluation - System Call Monitoring



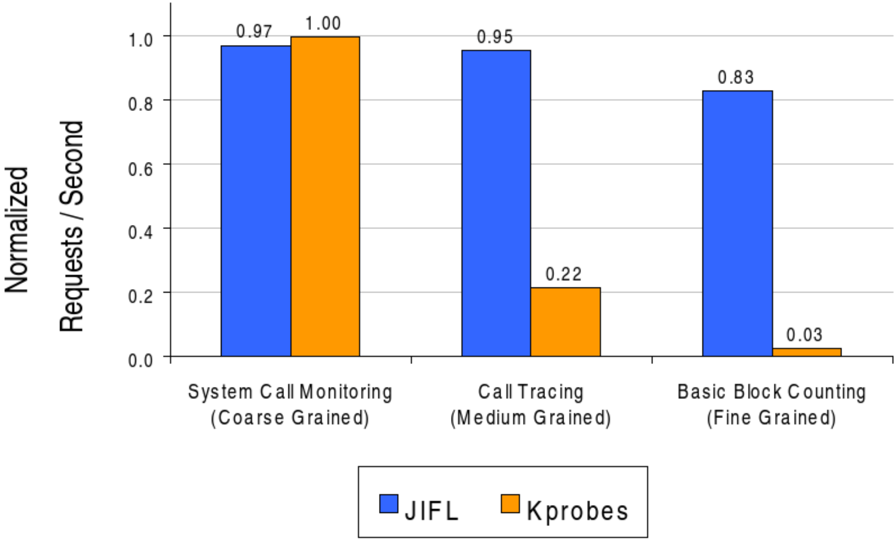
Evaluation - Call Tracing



Evaluation - Basic Block Counting



Evaluation - Apache Throughput



Conclusion

- JIT instrumentation viable for operating systems
- Results are very competitive
- Enables more powerful instrumentation