Unwind Stack Frame in Real Time

Mu Lin

Feb, 2011

Agenda

Objective

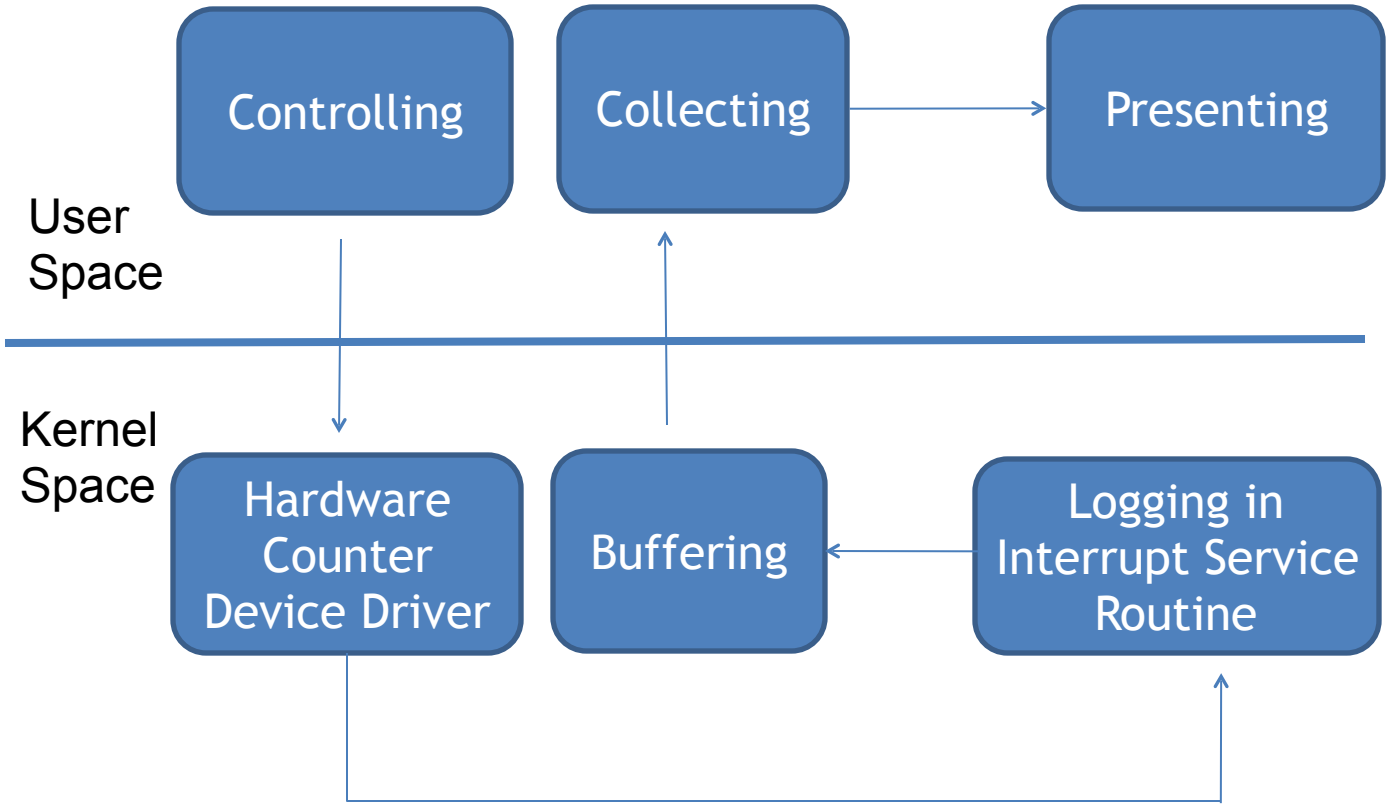
challenges

GDB Solution --- scan backward from PC

Real time solution --- pre-processing and PC lookup

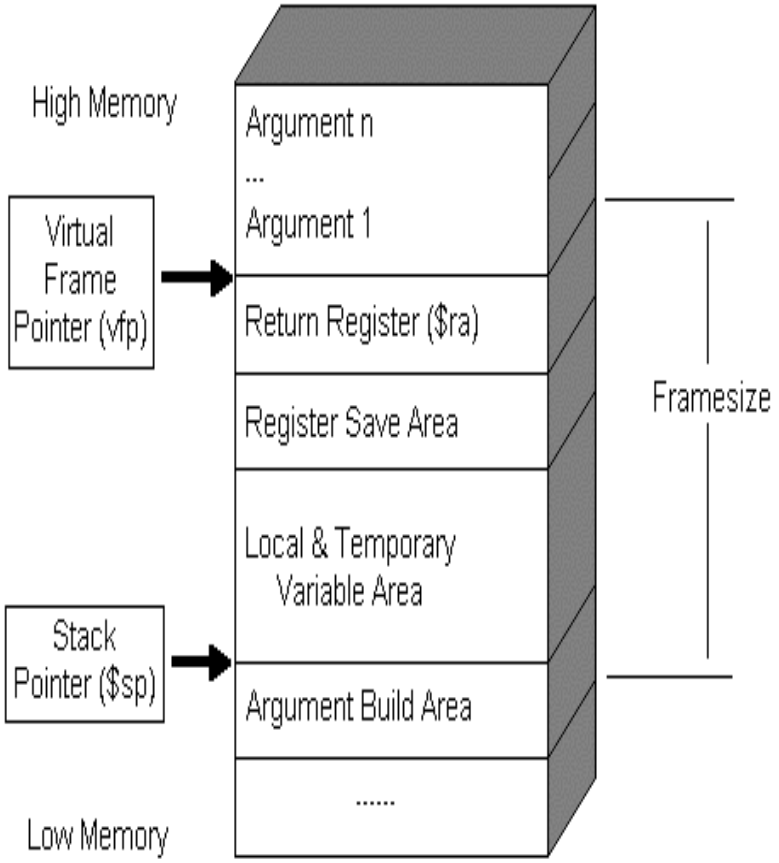
Q&A

Components

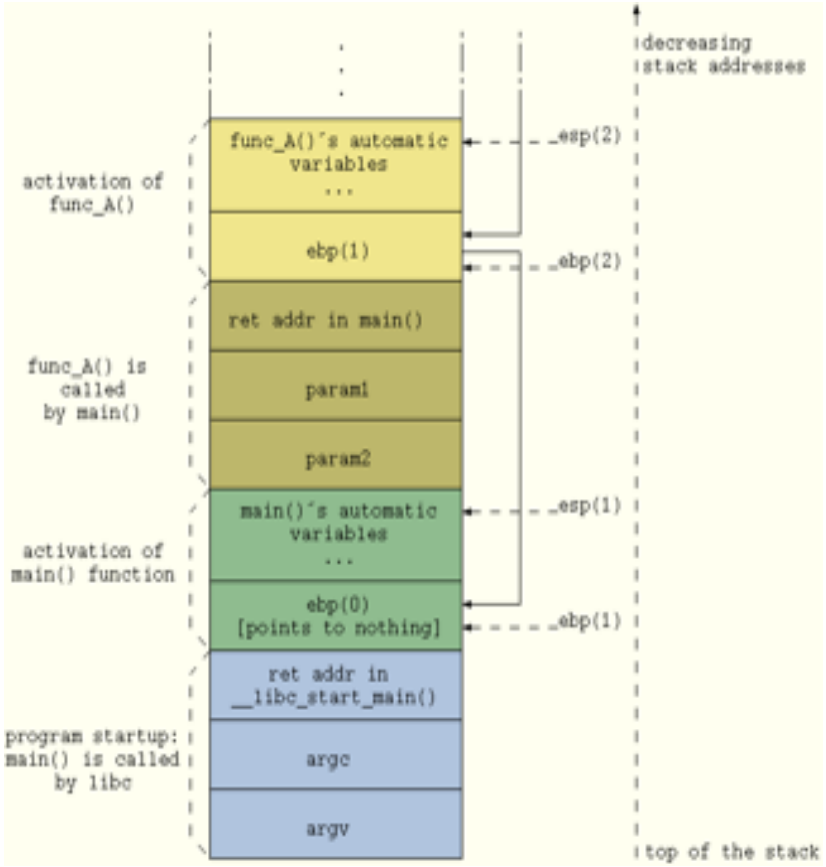

Controlling
Collecting
Presenting
User Space
Kernel Space
Hardware Counter Device Driver
Buffering
Logging in Interrupt Service Routine

Sampling

- Periodically obtain call graph (stack trace) on the fly on clock_tick **when profiling**.
- The algorithm used in trap.c and gdb for MIPS will be used as the first step.
- The method/algorithm needs to be as light-weighted as possible, the "probe effect" or "top syndrome" (but there are limits and costs associated with optimization).


High Memory
Argument n
...
Argument 1
Virtual Frame Pointer (vfp)
Return Register ($ra)
Register Save Area
Framesize
Local & Temporary Variable Area
Stack Pointer ($sp)
Argument Build Area
.....
Low Memory

MIPS


decreasing stack addresses
activation of func_A()
func_A()'s automatic variables
...
esp(2)
ebp(1)
ebp(2)
ret addr in main()
func_A() is called by main()
param1
param2
activation of main() function
main()'s automatic variables
...
esp(1)
ebp(0) [points to nothing]
ebp(1)
ret addr in __libc_start_main()
program startup: main() is called by libc
argc
argv
top of the stack

i386

GDB's Solution

```
MIPS_Back_Trace() {
    Obtain SP, PC;
    Walk back from PC to find start_of_routine;
    Walk forward and decode instructions to find ra and
        stack_size;
    SP + stack_size; PC = ra; depth--;
    Repeat till ra==0 or depth == 0;
}
```

The prologue

- **00000174 <foo>:**
- **174: 27bdffd8 addiu sp,sp,-40**
- **178: afbf0024 sw ra,36(sp)**
- **17c: afb00020 sw s0,32(sp)**
- **180: 3c1c0000 lui gp,0x0**
- **184: 279c0000 addiu gp,gp,0**
- **188: afbc0010 sw gp,16(sp)**
- **18c: 00a08021 move s0,a1**
- **190: 8f990000 lw t9,0(gp)**
- **194: 0320f809 jalr t9**
- **198: 27a60018 addiu a2,sp,24**
- **19c: 10400002 beqz v0, 1a8 <foo+0x34>**
- **1a0: 8fbc0010 lw gp,16(sp)**
- **1a4: ae000000 sw zero,0(s0)**
- **1a8: 8fbf0024 lw ra,36(sp)**
- **1ac: 8fb00020 lw s0,32(sp)**
- **1b0: 03e00008 jr ra**
- **1b4: 27bd0028 addiu sp,sp,40**

Solution

- MIPS's function call overhead is quite expensive,
- we have more and more inline functions and longer functions in order to minimize function call overhead.

So, we may have a problem by linearly walking back from PC.

- Good news is We have a O(1) algorithm to find the start_of_routine from the given PC.

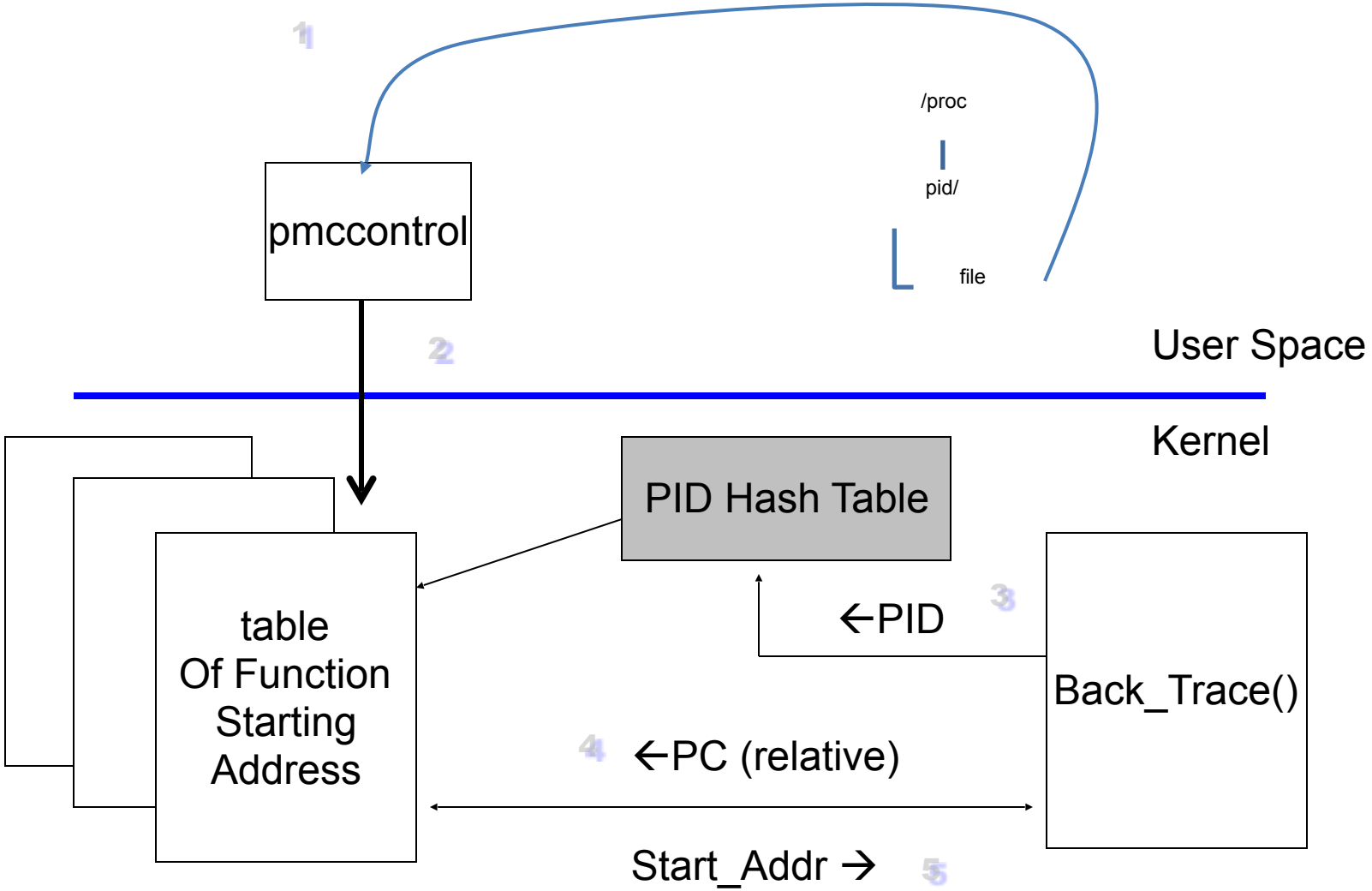
Real Time Solution

- Kernel keeps a table of starting_addr of routines for each targeted app.
- Do a indexed lookup to find the starting_addr for the given PC.

Take a typical daemon as example:

- Examine the symbol table or .pdr of the daemon, sort the function addresses and generate a compliant table to store the array.
- pmccontrol insert the table into kernel through ioctl() of /dev/pmc.
- Kernel keeps a hash-table to associate PID with the sorted arrays.

Solution Diagram



Multi-thread

- MIPS64 has two perf counters per core, but these counters are not virtualized.
- Thus for 4 VCPU per core, only two of them at one time can use the perf counters.
- Work around this using static thread pool.