User Space Dynamic Instrumentation
based on kprobe

04.06.2009

Samsung Advanced Institute of Technology &
Samsung Research Center in Moscow
Jaehoon Jeong(hoony_jeong@samsung.com)
Ekaterina Gorelkina(e.gorelkina@samsung.com)
Alexey A. Gerenkov(a.gerenkov@samsung.com)

Contents

- Overview of Dynamic Instrumentation
- Overview of Kprobe infra-structures
- Approaches for user space dynamic instrumentation
  - Introduction of SWAP(System-wide Analyzer of Performance)
  - Goals and 3 problems
  - SWAP - Architecture and procedure summary
  - SWAP - Instrumentation result
  - SWAP - Overhead
  - Future work
- Open Source Plan

Classification of Analysis

- Static vs Dynamic Analysis
- Source code vs Binary Analysis

	Static	**Dynamic**
Source code	Static Source Analysis	Dynamic Source Analysis
Binary	Static Binary Analysis	Dynamic Binary Analysis

Dynamic Binary Analysis can instrument software
without modification, re-build and reload.

However, it is not easy!

"Dynamic Binary Analysis and Instrumentation or Building Tools is Easy"
2004, Nichola Nethercote

Why Dynamic Instrumentation?

General Requirements:

- Kernel Developer : I wish I could add a debug statement easily without recompile & rebuild.
- Technical Supporter: How can I get this additional data that is already available in the kernel easily and safely?
- Application Developer: How can I improve performance of my application on Linux?

From : RH2_Systemtap_OLS_2006

CE Specific Requirements:

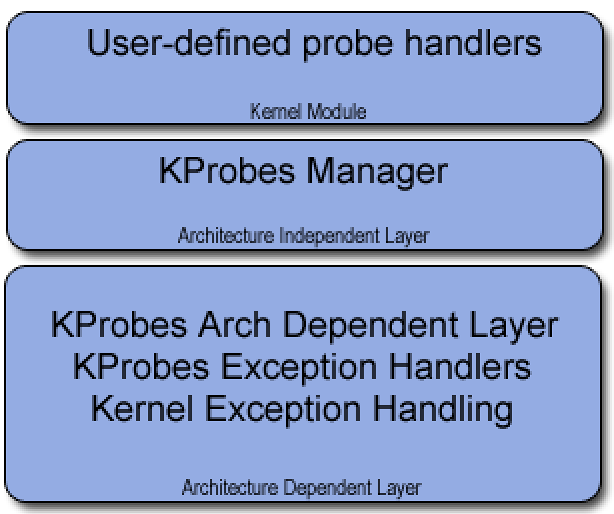
- CE Product Developer:
  - How can I analyze system behaviors in both kernel and application simultaneously?
  - Right before mass-production, how can I trace system events without modification of the current product?

Current Dynamic Instrumentation Tools

- Dtrace
  - Sun Solaris dynamic tracing framework for kernel and application.
  - D programming languages support for tracing program.
  - Instrumentation of both kernel and user space.
  - http://www.sun.com/software/solaris/ds/dtrace.jsp
- Systemtap
  - Linux dynamic instrumentation tool based on kprobe(kernel) and utrace/uprobe(user space)
  - Command line interface and scripting language
  - Open Source project(Red Hat, IBM, Intel, Hitachi, Oracle etc)
  - http://sourceware.org/systemtap/
- Dyninst
  - A set of APIs to permit the insertion of code into a running program.
  - Paradyn - dynamic instrumentation tool based on dyninst API.
  - User space only
  - Developed by University of Maryland
  - http://www.dyninst.org/

Kprobe infra-structures overview

- Kprobe is an infra-structure for dynamic instrumentation in Linux.
  - Designed for kernel space dynamic instrumentation.
  - 3 probing methods :
    - Kprobe : general use for any instruction(before/after execution of instruction)
    - Jprobe : special case for function entry(function argument when function is called)
    - Kretprobe : special case for function exit(function return value when function is returned)



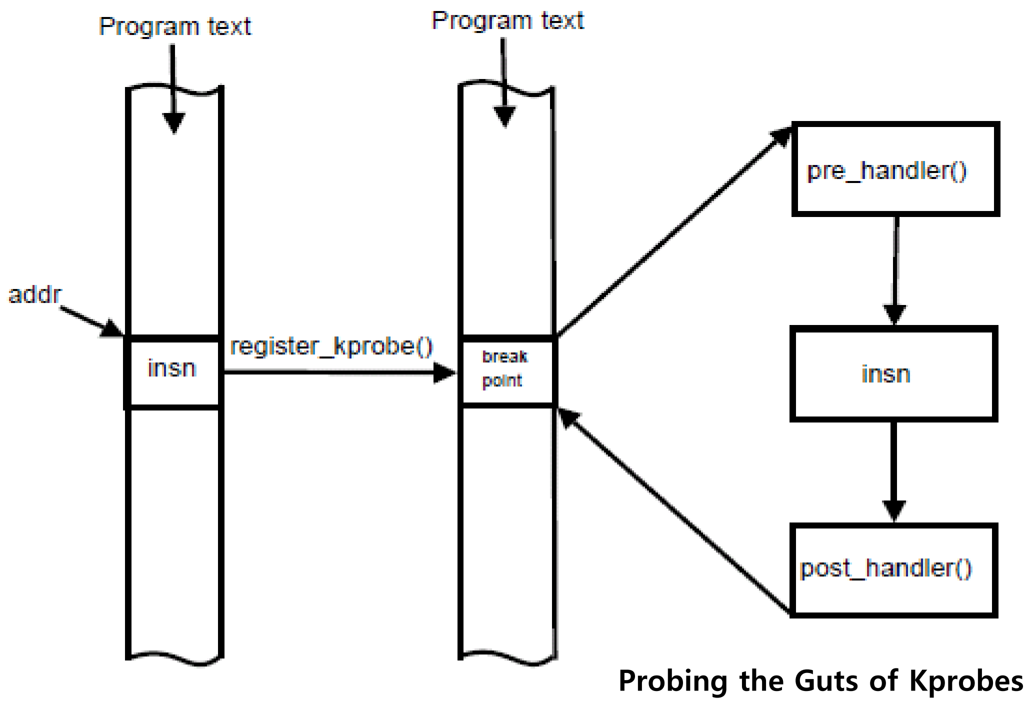
Users can make their own handler for specific probepoints.

- **Register/unregister probepoints**
- **Manage kprobe data structure**

- **Arch. specific break instruction**
- **Exception handling mechanism**

Kprobe

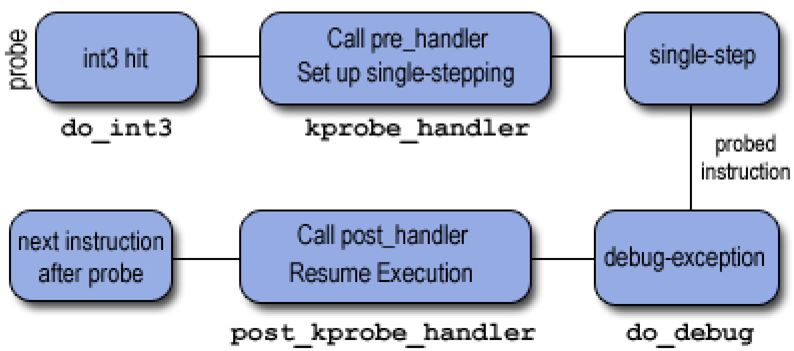
- It replaces the program inst. with a break instruction at the probepoint.
- Pre-handler is a user-defined routine that runs just before the inst.
- Original instruction is executed during exception.
- Post-handler is a user-defined routine that runs just after the inst.
- After return from exception, the next inst. is executed.



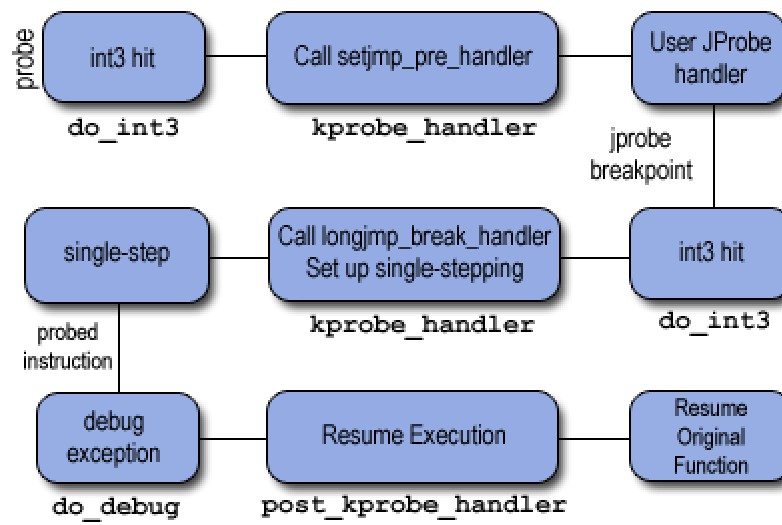
Probing the Guts of Kprobes : 2006 Linux Symposium

Jprobe

- Use the mirror prototype of original functions.
- Use its own pre-handler (setjmp_pre_handler) to save function arguments and CPU registers. After that, invoke one more break.
- Set up single-stepping and execute the original inst..
- Last steps are the same with original kprobe.



Execution of a KProbe

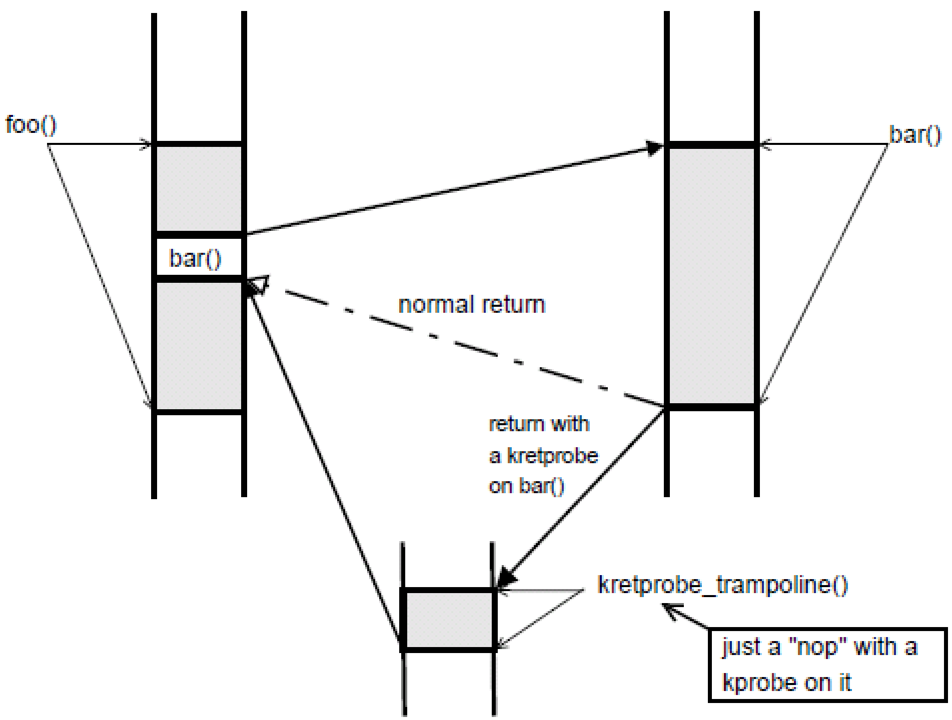


Execution of a JProbe

2005 Kprobe articles : lwn.net/Articles/132196/

kretprobe

- Set a break exception on function entry.
- When an exception is occurred, change the return address of function.
- When function is returned, go to kretprobe_tampoline.
- Save function return value and go to the original return address.



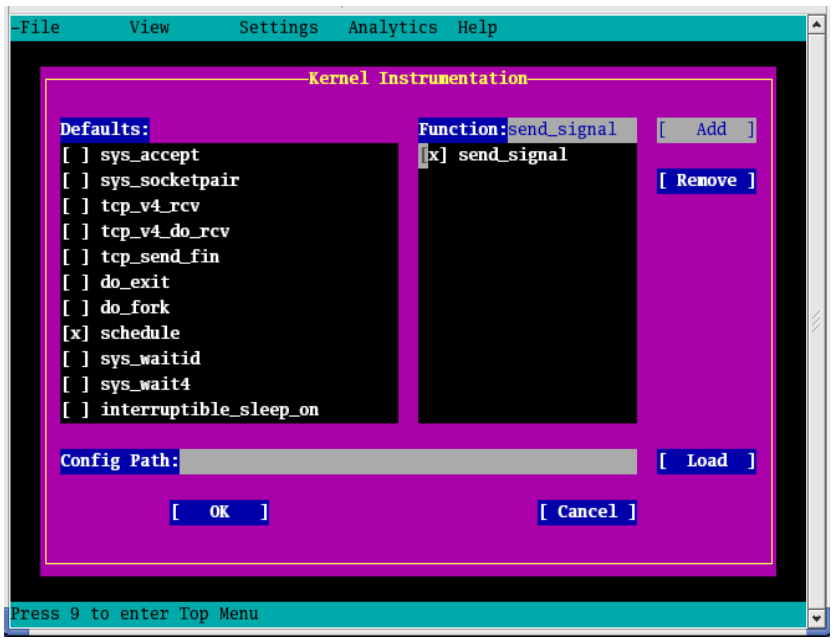
Probing the Guts of Kprobes : 2006 Linux Symposium

Contents

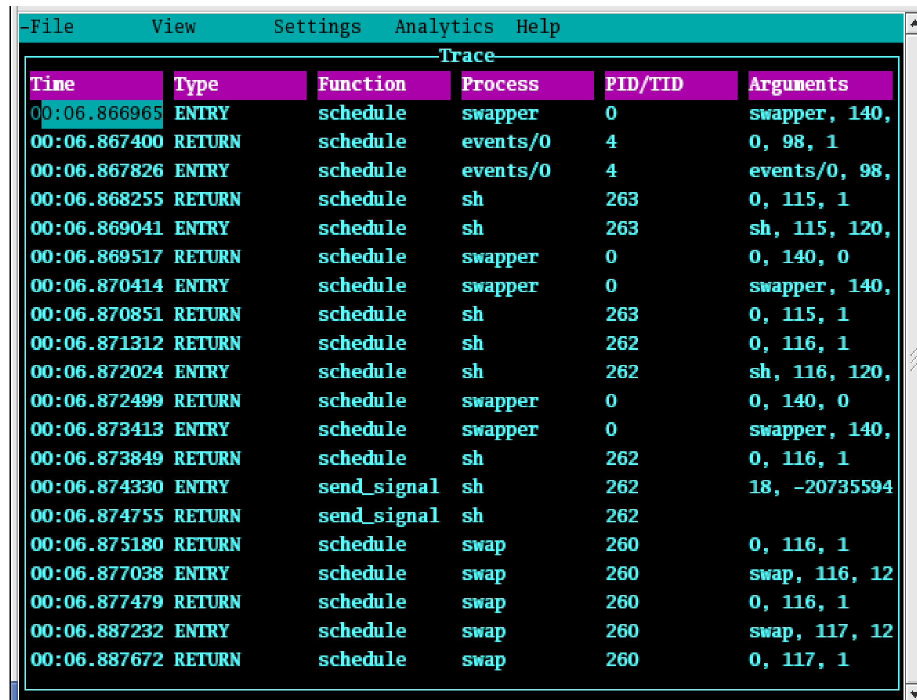
- Overview of Dynamic Instrumentation
- Overview of Kprobe infra-structures
- Approaches for user space dynamic instrumentation
  - Introduction of SWAP(System-wide Analyzer of Performance)
  - Goals and 3 problems
  - SWAP - Architecture and procedure summary
  - SWAP - Instrumentation result
  - SWAP - Overhead
  - Future work
- Open Source Plan

Introduction of SWAP
(System-wide Analyzer of Performance)

- A dynamic instrumentation tool developed by Samsung
- Kprobe ported for ARM and MIPS architecture
- A small agent to control and show dynamic instrumentation on target
  - Based on ncurses UI
- Function level traceability in the kernel space.



Ex) control of dynamic instrumentation



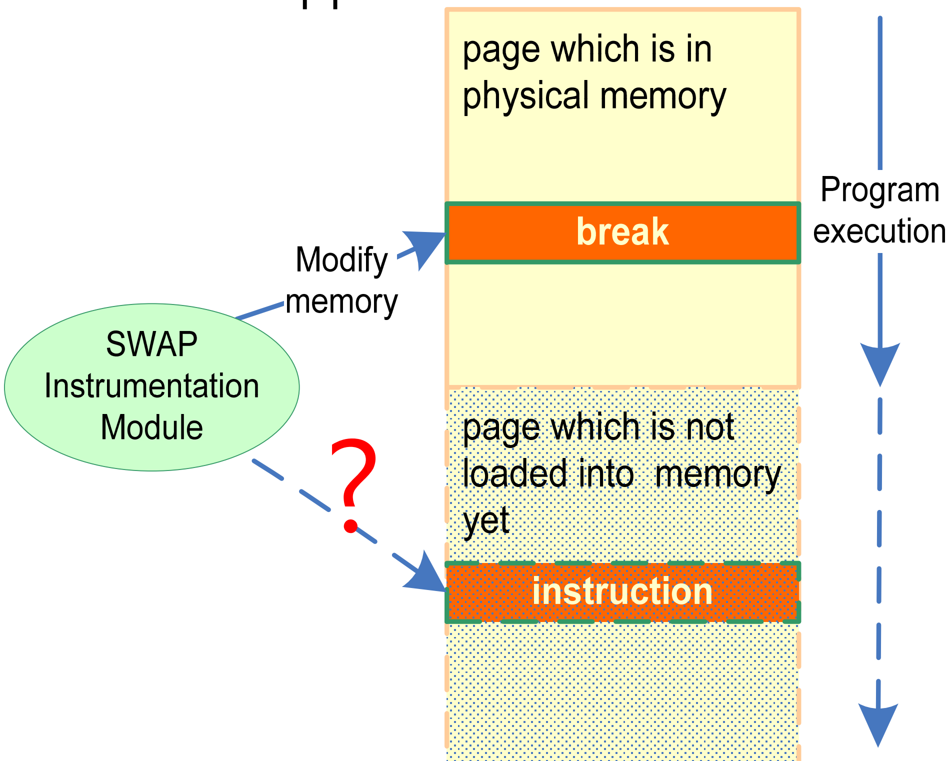
Ex) traced events

Goals and 3 problems

- Goals:
  - No intrusion of user space applications
  - Simultaneous instrumentation of both kernel and user space
  - Use of kprobe infra-structures for user space dynamic instrumentation
- Problems encountered during implementation:
  1. Demand-paging problem
  2. SSIL(single-stepping In Line), SSOL(Single-Stepping Out of Line)
  3. Return handler

1. Demand-Paging(1/2)

- Problem definition :
  - Kernel is always loaded in physical memory.
    - Probepoints can be inserted at any time.
  - Application is loaded in physical memory *on demand.*
    - How can we replace instructions with probepoints for applications?



Possible Solutions:

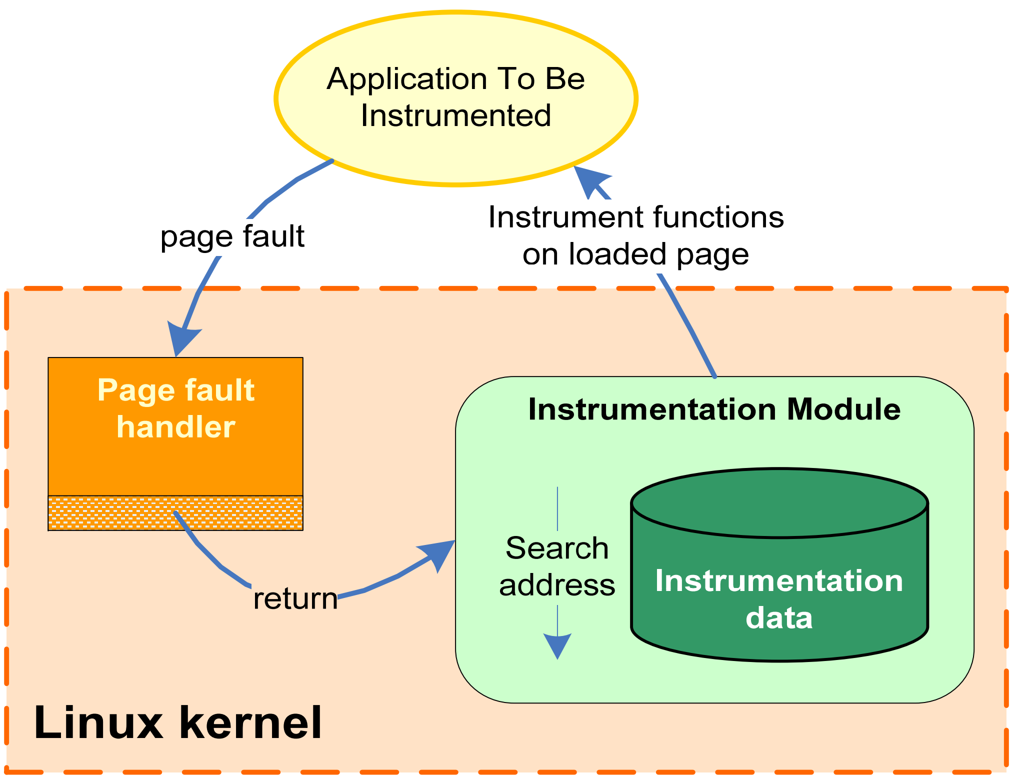
- Load all pages containing probepoints – No more demand-paging!!

But some questions:

- Is there enough physical memory?
- Would applications behave the same?
- Is it ok not to see any page-faults?

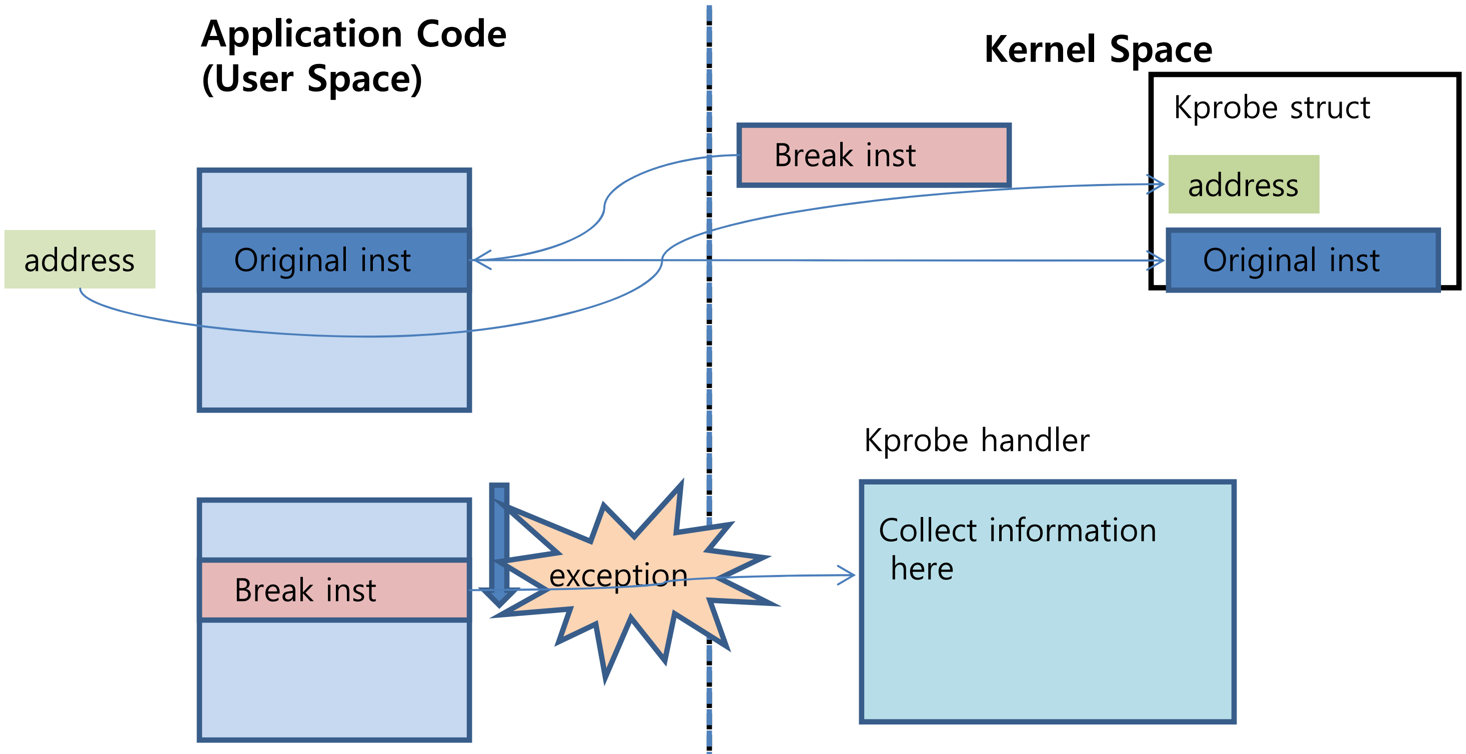
1. Demand-Paging(2/2)

- Our approaches
  - Maintain a list of probepoints for user specified functions
  - Instrument a page-fault handler in kernel with kretprobe in order to get information about loaded pages.
  - Check the probepoint list and insert probepoints on loaded page when page-fault handler returns



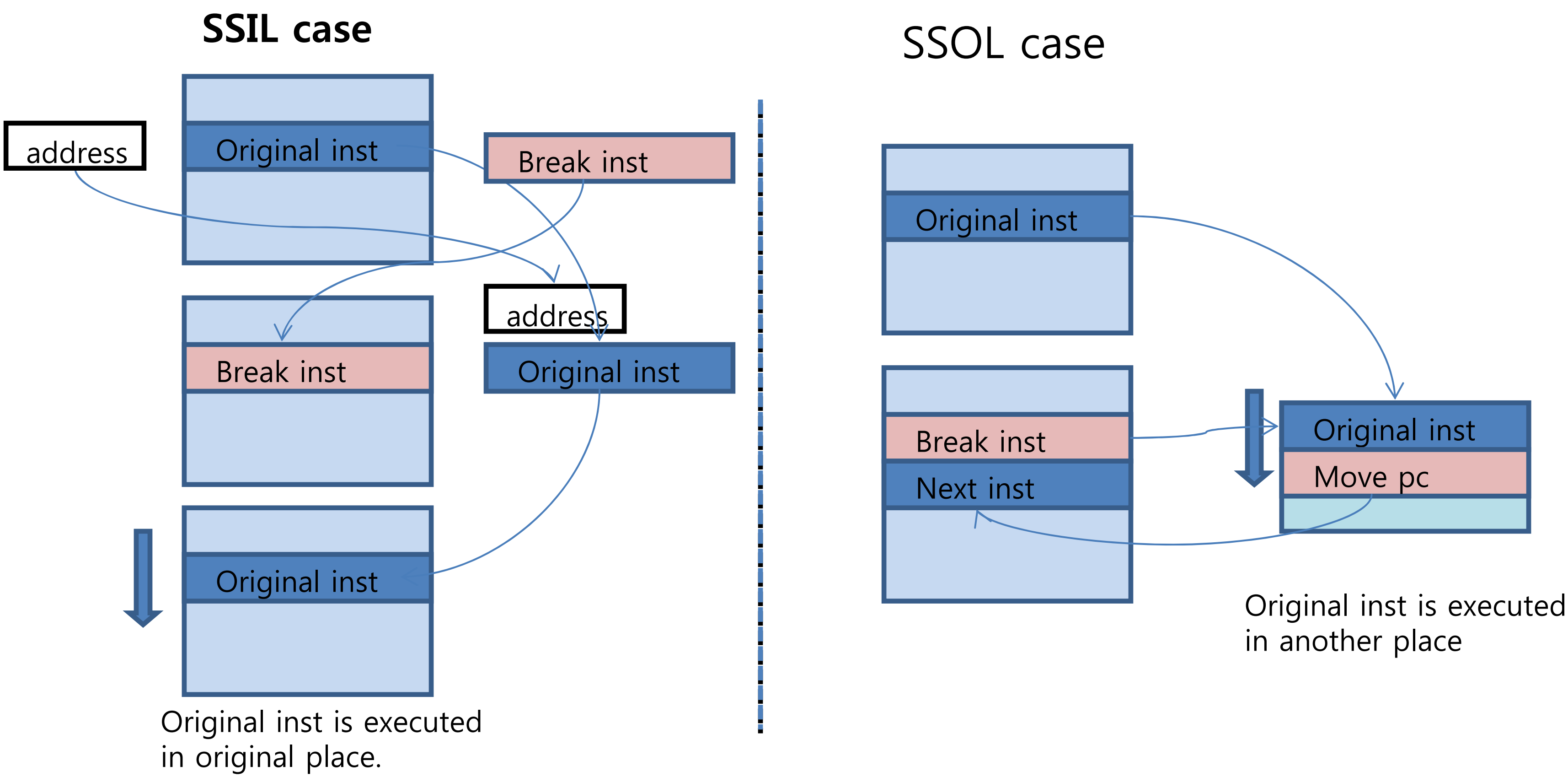
2. SSIL or SSOL problem(1/3)

- General approaches for our goals:
  - Instrument both kernel and user-space simultaneously.
  - Use a break instruction on a probepoint.
  - Use the same mechanism for kernel and user space.
    - Use kprobe infra-structures.
    - Execute handlers for both kernel and user in kernel.
  - This approach make a single stepping problem.



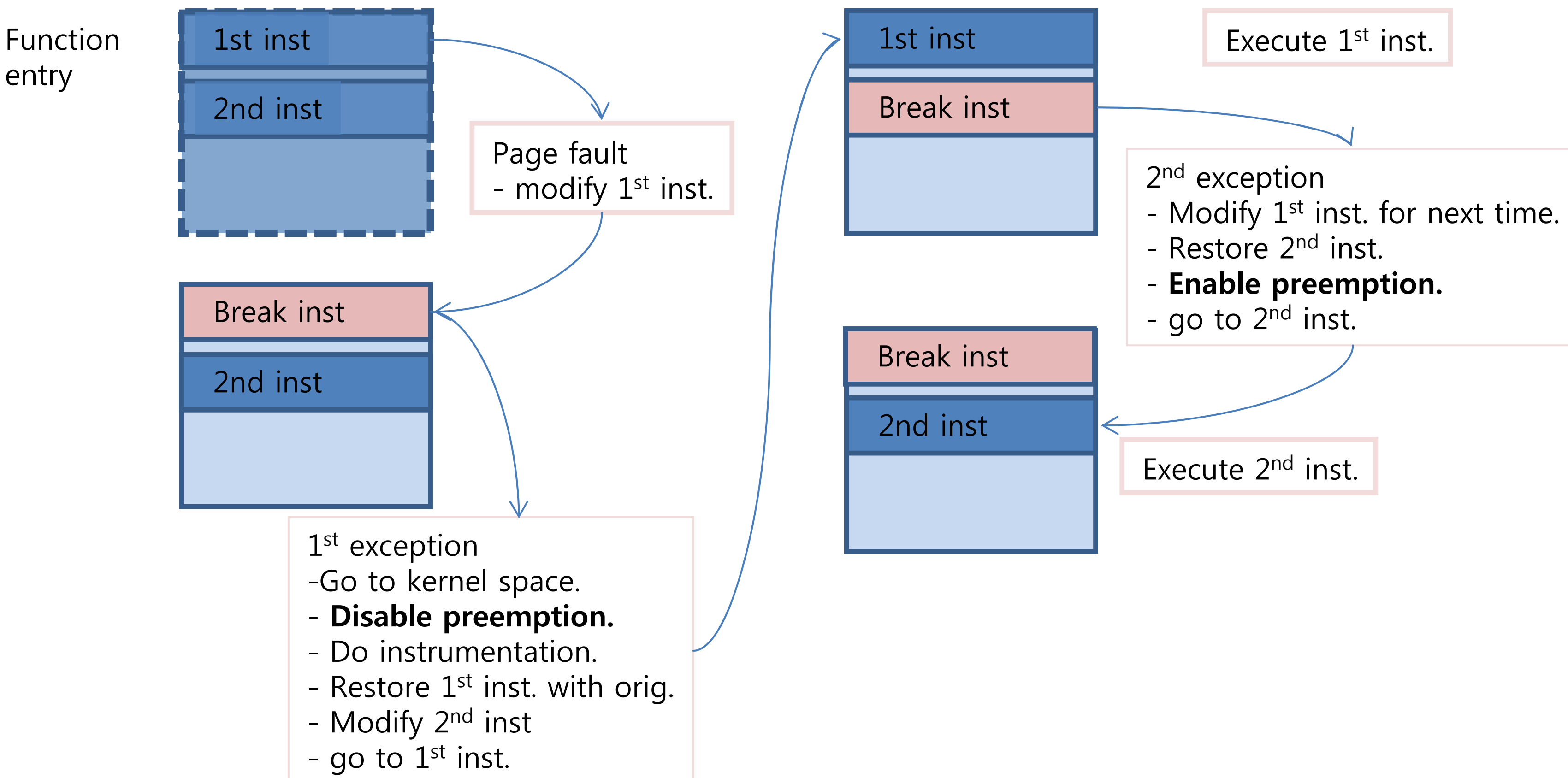
2. SSIL or SSOL problem(2/3)

- Problem definition :
  - SSIL(single-stepping inline) or SSOL(single-stepping out of line)
  - Kprobe uses SSOL not to miss a probepoint.
  - For the instrumentation of user applications in kernel space, SSOL is not acceptable.
    - Application instructions should be executed in user space.



2. SSIL or SSOL problem(3/3)

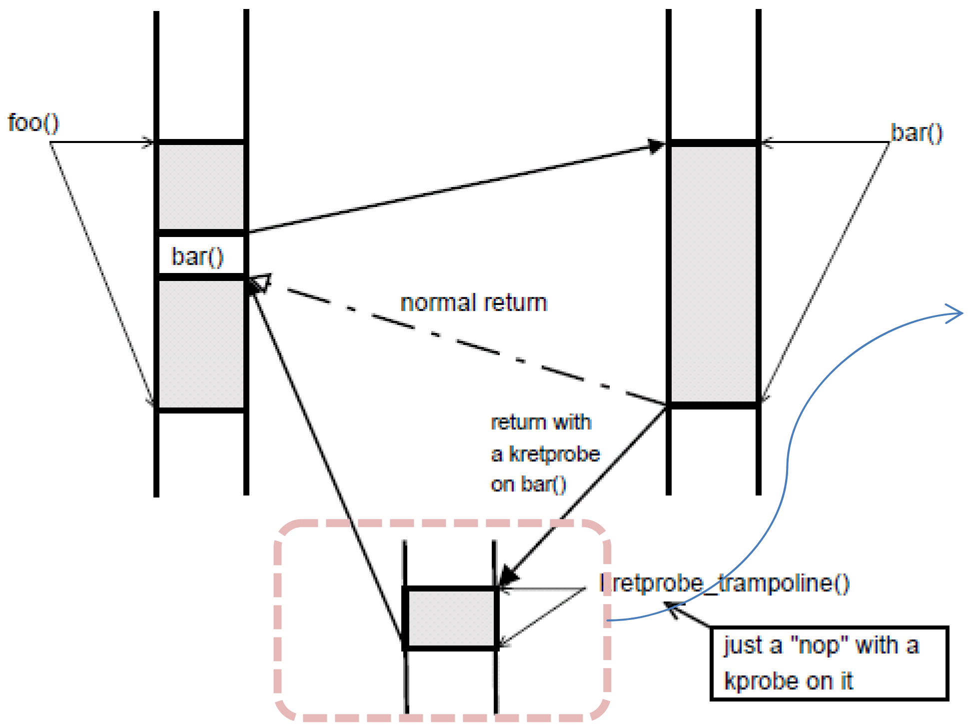
- Our approach :
  - Use SSIL for function-entry instrumentation
  - In order to restore and execute an original inst., use the break-inst. in the location of the 2nd instr..
  - During this sequence, set “preemption disable” for preventing from preemption.



3. Return handler (1/2)

- Problem definition :
  - From a slide no. 9, kretprobe use its trampoline in the same address space.
  - In this mechanism, trampoline will execute in user-space.
  - We want to instrument user-space events in kernel-space.

Application in User-space

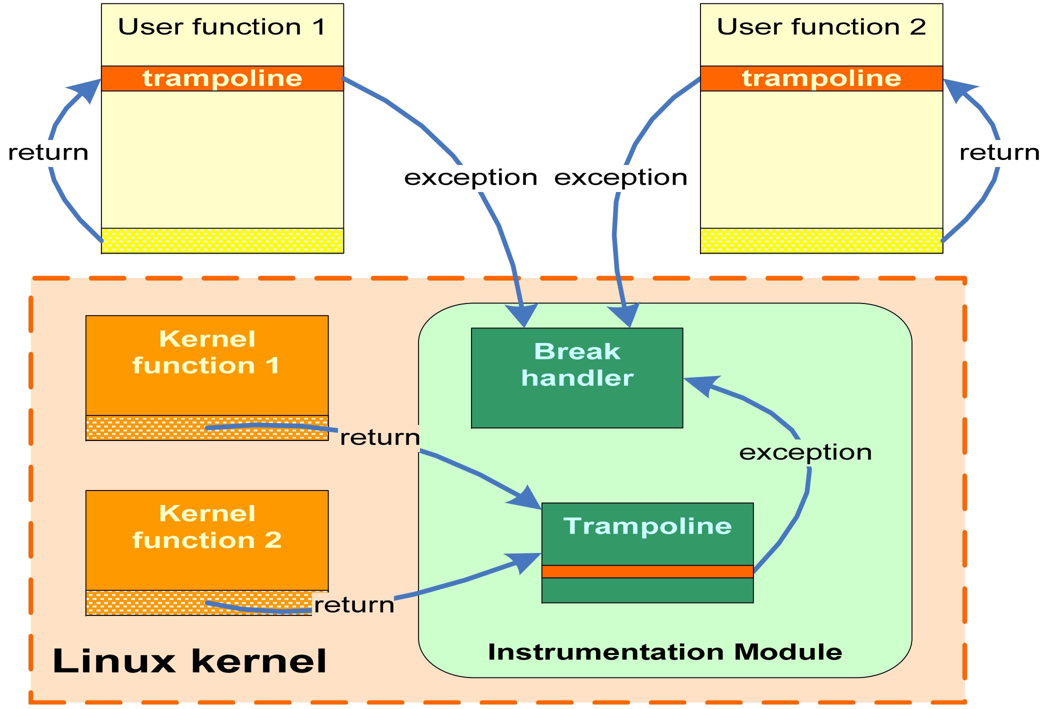


Same address space with application.
How can we move this trampoline
into kernel space.

Kretprobe mechanism

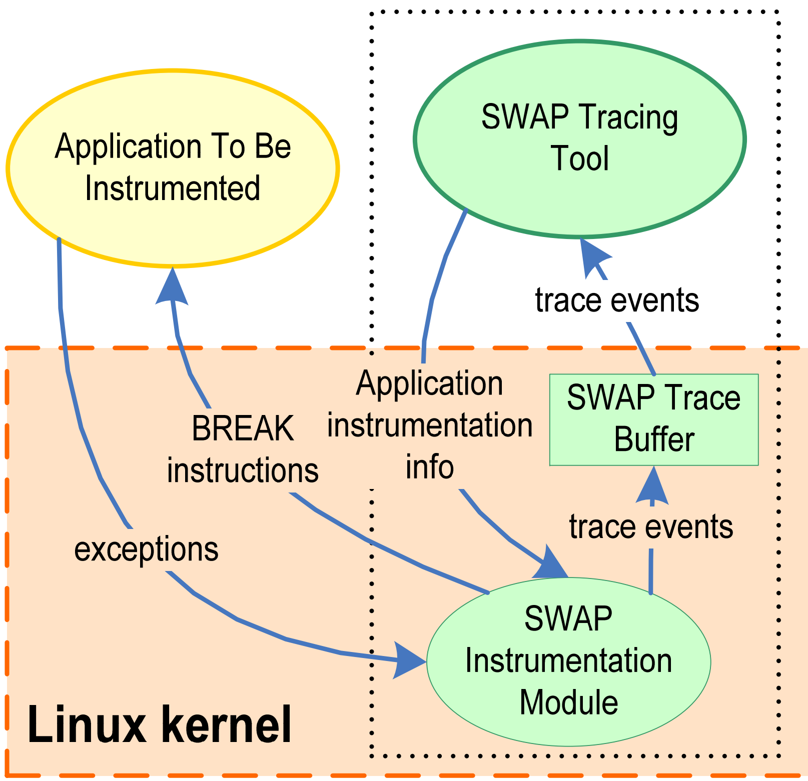
3. Return handler (2/2)

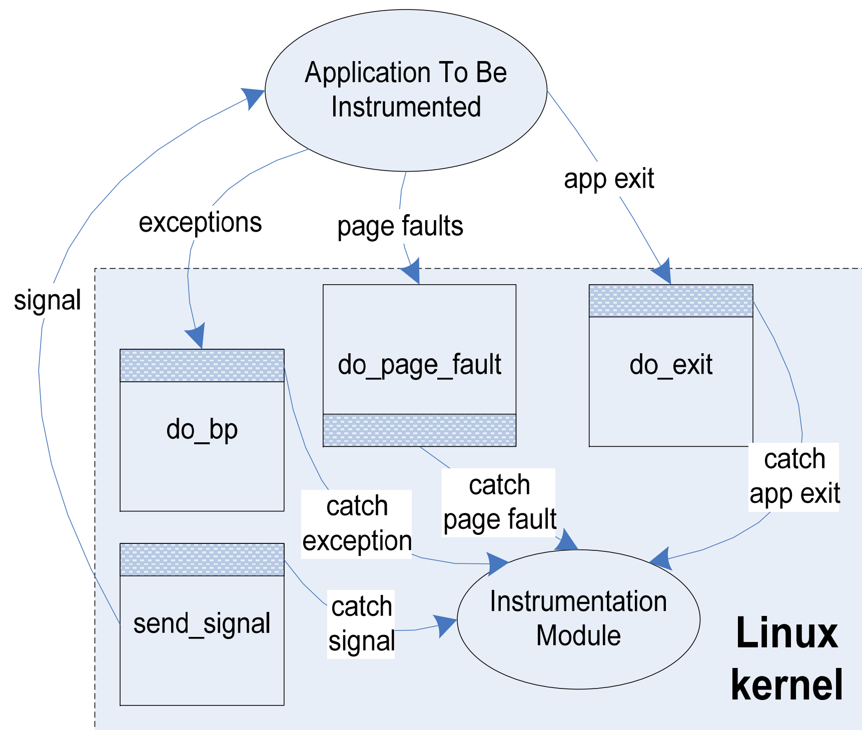
- Our approach :
  - Starting from the function entry point, when the 1st inst is invoked, return address is changed with the 2nd instruction address.
  - Replace the 2nd inst. w/ the break inst.
  - During execution, the 2nd inst. causes an exception to occur and retprobe_trampoline in kernel space is executed.
  - In the exception handler, go to the universal trampoline to instrument a function return-value .
  - This trampoline is common for both kernel and user function.



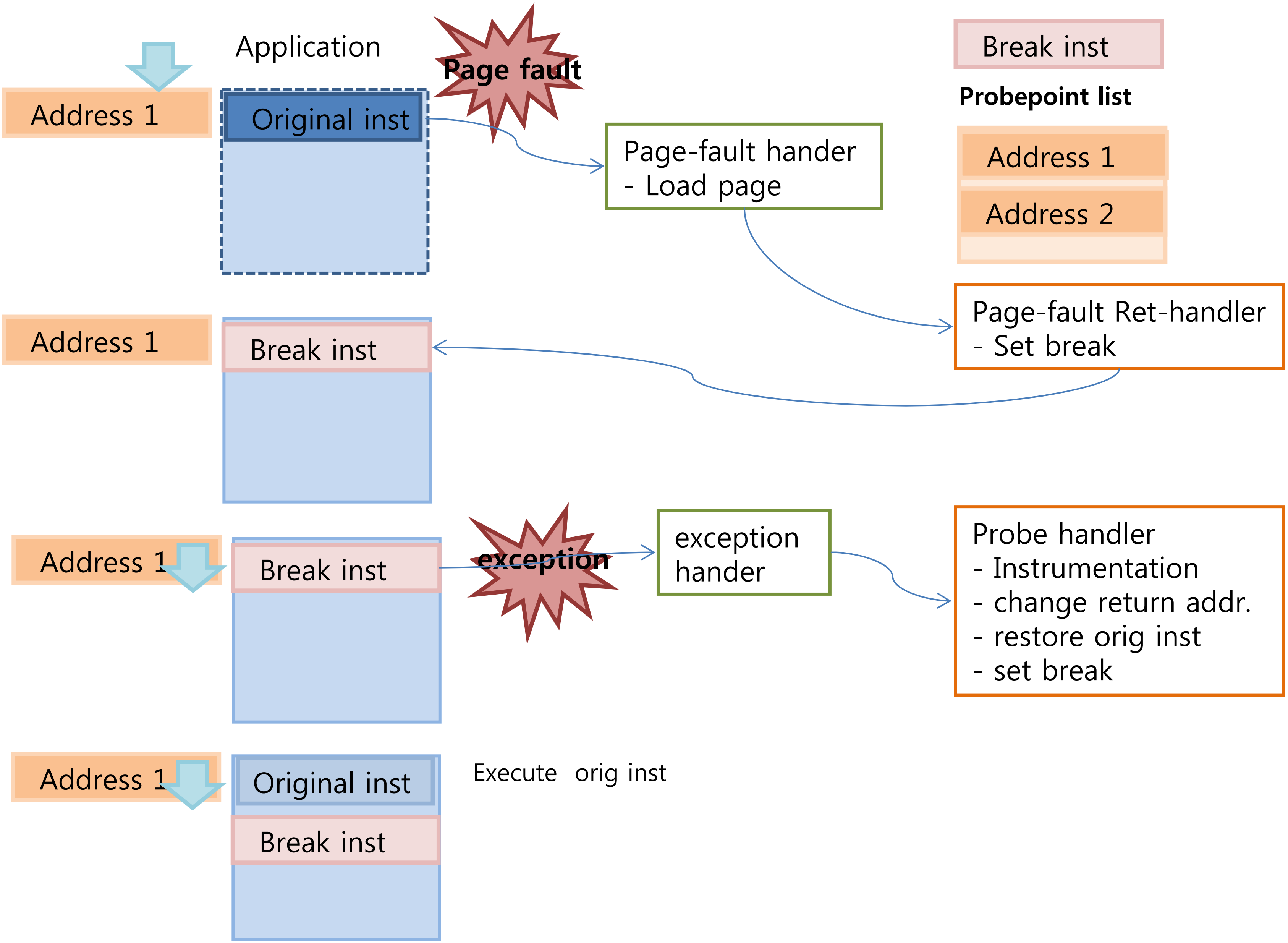
SWAP - Architecture Summary

- SWAP Tracing tool
  - Control instrumentation modules to insert/remove probepoints.
- SWAP Trace Buffer
  - Use a common buffer to save both kernel and user space traced events.
- SWAP Implementation Module
  - Manage probepoints and their handlers based on kprobe infra-structures.

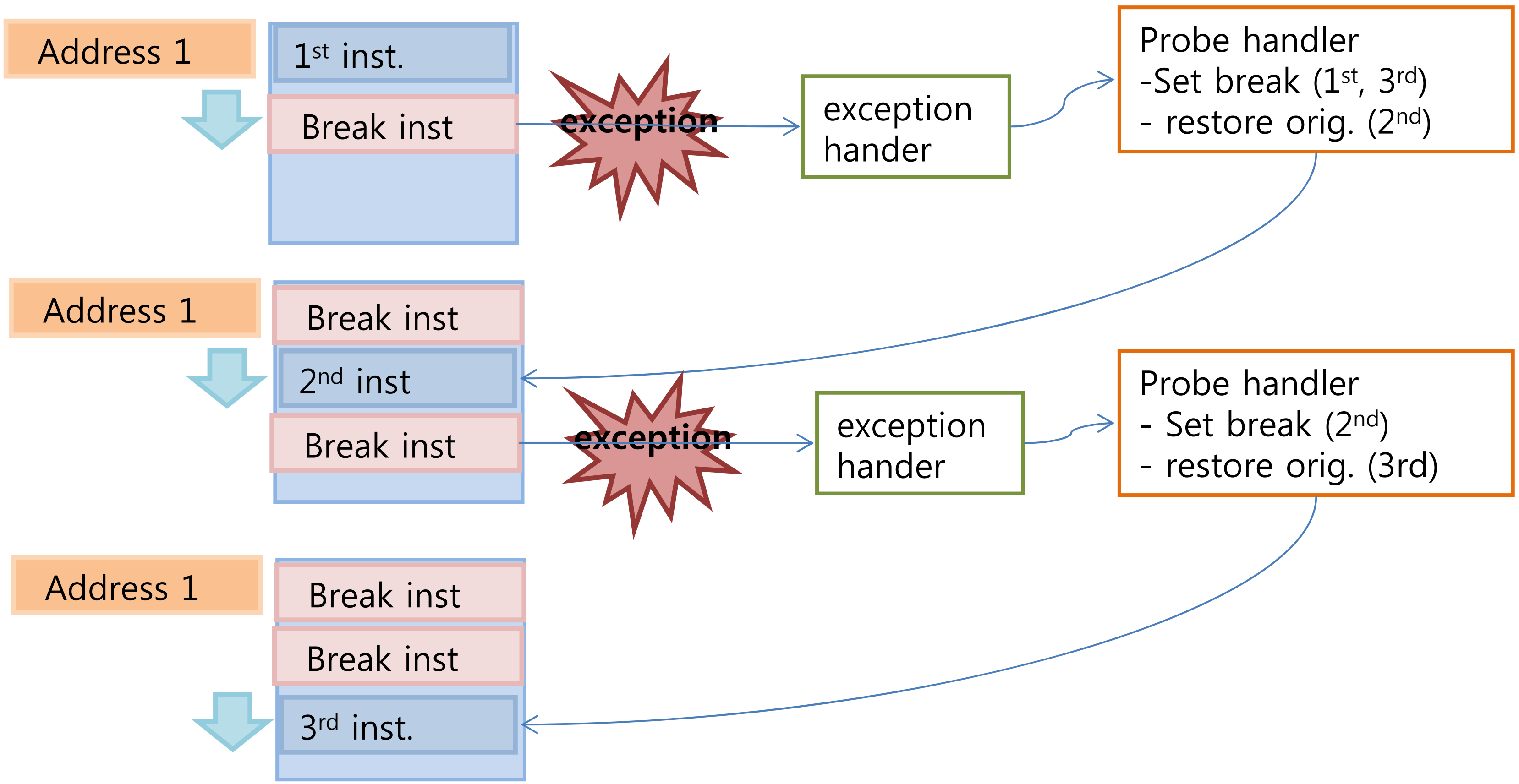




SWAP- Procedure summary(1/3)

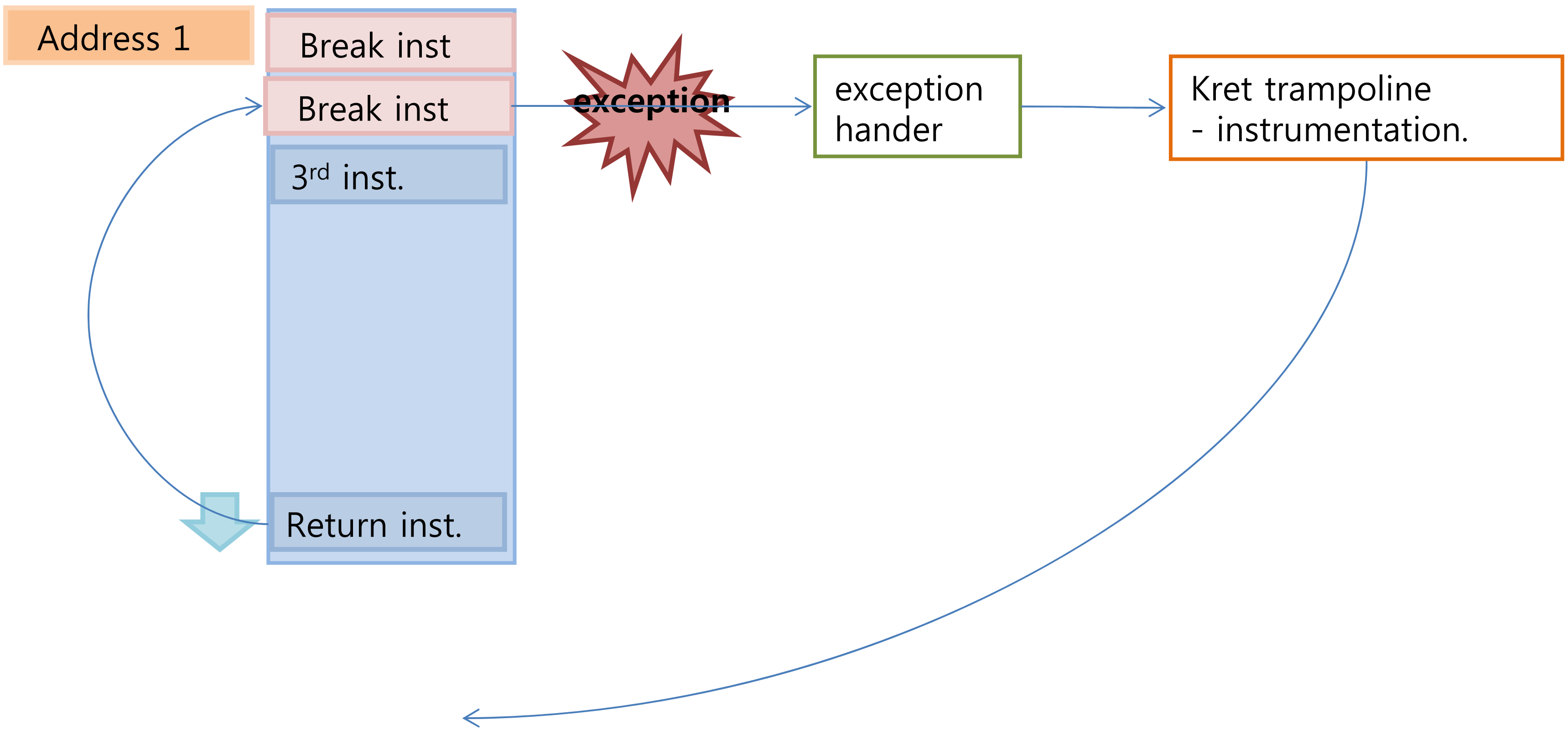


Procedure summary(2/3)



Function body is executed!!

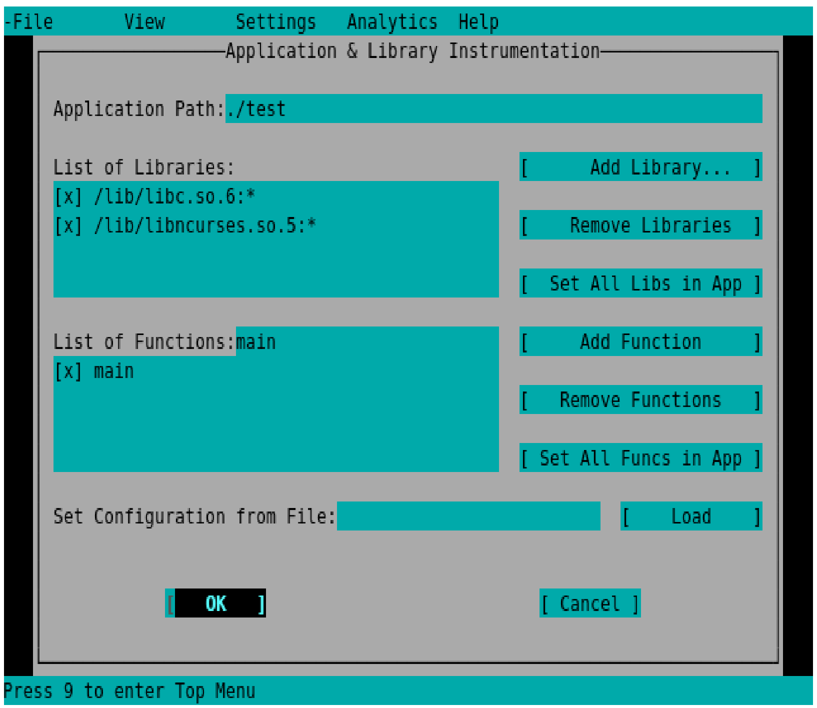
Procedure summary(3/3)

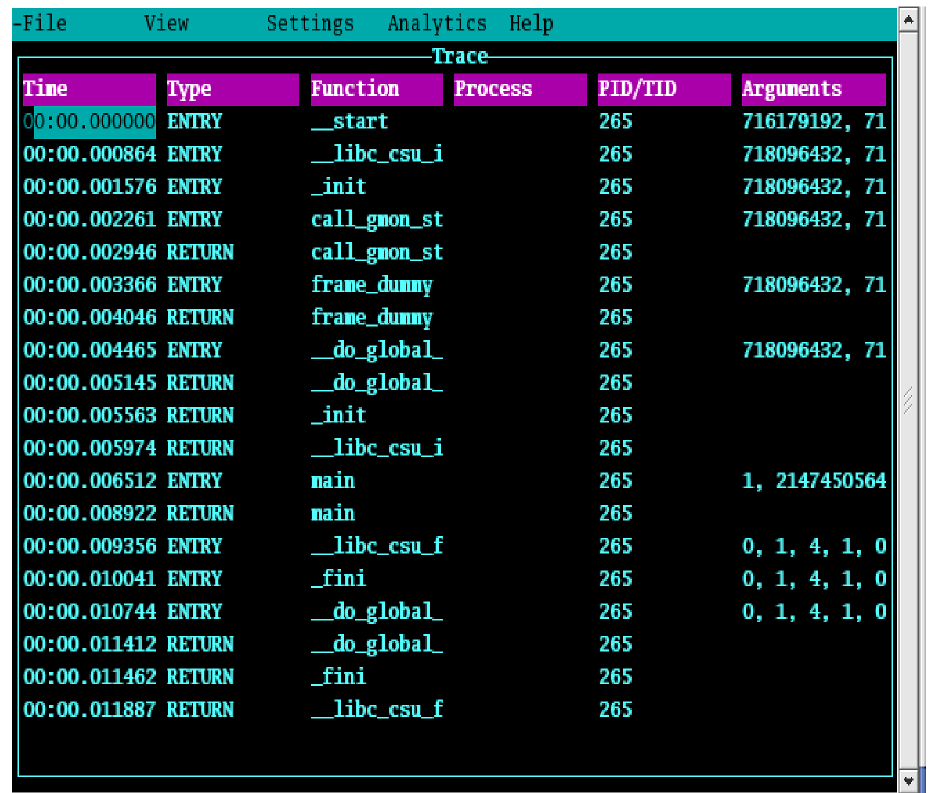


Return to the original return address

SWAP - Implementation Result

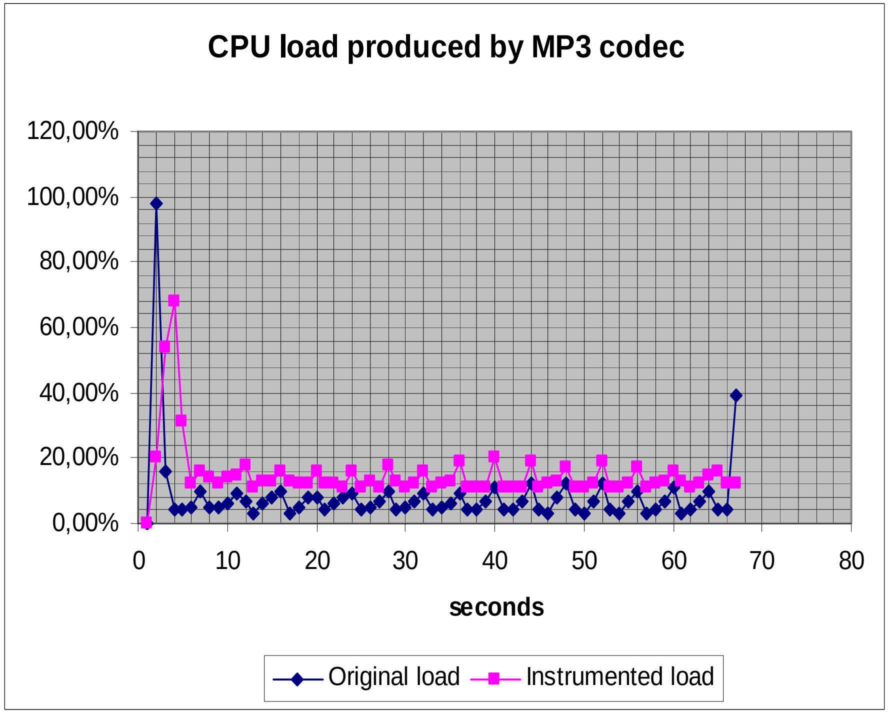
- In the SWAP tracing tool, check an application and its lib to be probed.
- Start tracing, start an application, and stop tracing.
- Show traced events.





SWAP - Instrumentation Overhead

- CPU load overhead
  - **original = 6,87%**
  - **libc instrumentation = 13,27%**
  - **application instrumentation = 22,23%**



- Page-fault overhead

Without instrumentation:
Maximum Page_Fault_Duration = 222 usec
Average Page_Fault_Duration = 64 usec

Libc instrumentation:
Maximum Page_Fault_Duration = 334280 usec
Average Page_Fault_Duration = 7740 usec

Application instrumentation:
Maximum Page_Fault_Duration = 8049 usec
Average Page_Fault_Duration = 199 usec

It depends on
the number of probepoints.

SWAP - Future work

- Reduce instrumentation overhead
  - Reduce the number of break instructions.
  - Optimize lookup overhead of probepoints

- Stripped binaries….
  - Problem: Most CE products use stripped binaries to minimize resources.
    - **Can not extract symbol information from a stripped binary.**
  - Solution: a cross instrumentation environment between target and host
    - **Target : instrumentation only**
    - **Host : select a probepoint , make its handler, and analyze traced events.**
    - similar to a cross-compile environment

Open Source Plan

- Now, cleaning source code and fixing bugs.
  - Plan to open SWAP by 3Q 2009.
  - Plan to post on the CELF website or Sourceforge.net.

- Your ideas/inputs/comments are welcomed!!!