Assignment: Tracking Page Faults for a Process

Objective

You will learn how to use kernel-level probing mechanisms to intercept and track kernel events, here specifically page-faults for a specific process.

Description

Kprobes enables you to dynamically set a breakpoint in any kernel routine and collect debugging and performance information non-disruptively. You can trap at almost any kernel code address, specifying a handler routine to be invoked when the breakpoint is hit.

1. Learn how to use kprobes in the Linux kernel. https://www.kernel.org/doc/Documentation/kprobes.txt
2. Try these example kernel modules for practice: https://elixir.bootlin.com/linux/latest/source/samples/kprobes
3. Study the handle_mm_fault() function in the Linux kernel to understand when it is invoked and what it does.
4. Part A: Now create a kernel module that takes the process-ID of an active process as an argument (either module parameter or via an ioctl interface), and tracks all the virtual addresses on which the target process faults. Print the virtual addresses to the system log using printk(). Show that your code works for any arbitrary target process.
5. Part B: Plot the virtual addresses you tracked as a scatter-plot graph with X-axis representing the time and Y-axis representing the virtual address. Try at least three different types of target applications, such as kernel compilation (compute and I/O intensive), sysbench (compute intensive), iperf (network I/O intensive), or some other applications. See if you can observe any interesting trends in memory access patterns of a process. Describe your results in a concise report.
6. Bonus Section: (optional) Store the tracked virtual addresses and the time at which each address was trapped in a kernel buffer, possibly a circular array. Retrieve the buffer from user space using an ioctl() interface.

Grading guidelines

50 - Part A: Implementation that works for any arbitrary process
20 - Part B: Report explaining scatterplot results from three different applications.
50 - Demo and explanation of code and report.
10 - Handle all major error conditions. Clean, modular, and commented code. Clean readable output. No "giant" functions.
Total = 130

10 - Bonus section: (optional) Demonstrate that your code can retrieve tracked virtual addresses using the ioctl() interface from a user space program for any arbitrary process (and explain the code during demo).