Kprobe

 A **kprobe** is a dynamic debugging mechanism in Linux that lets you “hook” into a kernel function at runtime.

 In this lab, you’re attaching a probe to the function omap_gpio_get() (the kernel’s internal function for reading a GPIO value on OMAP/AM335x/BeagleBone processors).

 Once attached, whenever the kernel calls omap_gpio_get(), your kprobe’s **pre_handler** and **post_handler** functions will run.

 You’ll then trigger this function by reading GPIO from **sysfs** (e.g., cat /sys/class/gpio/gpioXX/value).

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/kprobes.h>
static struct kprobe kp;
/* Pre-handler: runs before omap_gpio_get executes */
static int handler_pre(struct kprobe *p, struct pt_regs *regs)
{
   pr_info("kprobe pre_handler: function=%s\n", p->symbol_name);
   return 0;
}

/* Post-handler: runs after omap_gpio_get executes */
static void handler_post(struct kprobe *p, struct pt_regs *regs,
                         unsigned long flags)
{
   pr_info("kprobe post_handler: function=%s\n", p->symbol_name);
}
```

```
/* Fault-handler (optional, if instruction causes fault) */
static int handler_fault(struct kprobe *p, struct pt_regs *regs, int trapnr)
{
    pr_info("kprobe fault_handler\n");
    return 0;
}

static int __init kprobe_init(void)
{
    kp.symbol_name = "omap_gpio_get";  // function to probe
    kp.pre_handler = handler_pre;
    kp.post_handler = handler_post;
    kp.fault_handler = handler_fault;

    if (register_kprobe(&kp) < 0) {
        pr_err("register_kprobe failed\n");
        return -1;
    }
    pr_info("Kprobe registered for %s\n", kp.symbol_name);
    return 0;
}
static void __exit kprobe_exit(void)
{
    unregister_kprobe(&kp);
    pr_info("Kprobe unregistered\n");
}
module_init(kprobe_init)
module_exit(kprobe_exit)
MODULE_LICENSE("GPL");
```

Nice — you've reached **Experiment 23: Kprobe** 🙂
Let me break this lab down step by step, so you know exactly what's happening and why.

🔎 What is happening here?

- A **kprobe** is a dynamic debugging mechanism in Linux that lets you "hook" into a kernel function at runtime.
- In this lab, you're attaching a probe to the function omap_gpio_get() (the kernel's internal function for reading a GPIO value on OMAP/AM335x/BeagleBone processors).
- Once attached, whenever the kernel calls omap_gpio_get(), your kprobe's **pre_handler** and **post_handler** functions will run.
- You'll then trigger this function by reading GPIO from **sysfs** (e.g., cat /sys/class/gpio/gpioXX/value).

🛠 Steps for the Experiment

1. Write the Kprobe Module

Here's a minimal template for probing omap_gpio_get:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/kprobes.h>

static struct kprobe kp;

/* Pre-handler: runs before omap_gpio_get executes */
static int handler_pre(struct kprobe *p, struct pt_regs *regs)
{
    pr_info("kprobe pre_handler: function=%s\n", p->symbol_name);
    return 0;
}
```

```
/* Post-handler: runs after omap_gpio_get executes */
static void handler_post(struct kprobe *p, struct pt_regs *regs,
                         unsigned long flags)
{
    pr_info("kprobe post_handler: function=%s\n", p->symbol_name);
}

/* Fault-handler (optional, if instruction causes fault) */
static int handler_fault(struct kprobe *p, struct pt_regs *regs, int trapnr)
{
    pr_info("kprobe fault_handler\n");
    return 0;
}

static int __init kprobe_init(void)
{
    kp.symbol_name = "omap_gpio_get";  // function to probe
    kp.pre_handler = handler_pre;
    kp.post_handler = handler_post;
    kp.fault_handler = handler_fault;

    if (register_kprobe(&kp) < 0) {
        pr_err("register_kprobe failed\n");
        return -1;
    }

    pr_info("Kprobe registered for %s\n", kp.symbol_name);
    return 0;
}
```

```
static void __exit kprobe_exit(void)
{
    unregister_kprobe(&kp);
    pr_info("Kprobe unregistered\n");
}

module_init(kprobe_init)
module_exit(kprobe_exit)
MODULE_LICENSE("GPL");
```

---

2. Cross-Compile the Module

On your host (with cross-toolchain set up for BeagleBone):

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -C ~/bbb-kernel M=$PWD modules
```

---

3. Insert Module on BBB

```
sudo insmod kprobe_gpio.ko
dmesg | tail
```

You should see:

```
Kprobe registered for omap_gpio_get
```

---

4. Trigger the Function

Use sysfs GPIO to trigger omap_gpio_get:

```
cd /sys/class/gpio
echo 49 > export        # Example: GPIO1_17
echo in > gpio49/direction
cat gpio49/value        # <-- This invokes omap_gpio_get()
```

---

5. Check Logs

Now check dmesg:

kprobe pre_handler: function=omap_gpio_get

kprobe post_handler: function=omap_gpio_get

You'll also see stack traces if you enable them inside the handler.

---

6. Cleanup

sudo rmmod kprobe_gpio

dmesg | tail
