**1. Force a Kernel Panic with panic() Function**

* This function immediately crashes the OS by design, bypassing error handling mechanisms.

panic("Forced kernel panic for testing");

* Calling panic() is the most direct way to crash the OS and would immediately halt the system with a kernel panic message.

**2. Divide-by-Zero Operation**

* Performing a divide-by-zero operation in kernel space is another reliable way to cause a kernel panic.

int faulty\_divide = 1 / 0; // Divide by zero

* Placing this inside any function in your driver, such as simple\_open, will cause a CPU exception, leading to a crash.

**3. Page Fault with Invalid Memory Access**

* Accessing memory at an address known to be invalid will also cause a crash, especially in kernel space.

int \*invalid\_ptr = (int \*)0xDEADBEEF;

\*invalid\_ptr = 42; // Accessing invalid memory address

* Using an invalid pointer like this inside any driver function (e.g., simple\_read) will result in a page fault, which typically leads to a crash.

**4. Triggering an OOPS with BUG() or BUG\_ON()**

* BUG() and BUG\_ON(condition) macros are used in kernel code to trigger an OOPS (an error that usually leads to a crash) if a condition is met.

BUG(); // Unconditional crash

BUG\_ON(1); // Conditional crash (always true in this case)

* Placing BUG() or BUG\_ON(1); in any function, such as simple\_open, will lead to an OOPS, halting execution and often crashing the OS.

**5. Trigger a Stack Overflow with Deep Recursion**

* Recursive function calls that don’t terminate will exhaust the kernel stack, leading to a crash.

void recursive\_crash(void) {

recursive\_crash();

}

static int \_\_init crash\_init(void) {

recursive\_crash(); // Calls the function recursively until stack overflow

return 0;

}

* Placing this in the init function of your module will cause a stack overflow immediately on module insertion, usually resulting in a crash.

**6. Blocking Critical Resources with Infinite Loops in Atomic Context**

* Holding resources indefinitely with an infinite loop in atomic (interrupt-disabled) context can cause system hangs.

static int simple\_open(struct inode \*inode, struct file \*file) {

local\_irq\_disable(); // Disable interrupts

while (1); // Infinite loop prevents system progress

return 0;

}

* This can freeze the system since the CPU will not release the resource, causing it to stop responding.