Custom System Call

Below is a step-by-step guide on how you can add and test a new system call in Linux. Note that this process involves modifying the kernel source, recompiling the kernel, and writing a user-space test program. This is an advanced task, so make sure you have a safe testing environment (for example, a virtual machine) before proceeding.

---

1. Set Up Your Environment

Install Required Tools:
Ensure you have development tools (like gcc, make, and the required libraries) installed. On many distributions you can install them with:

sudo apt-get install build-essential libncurses-dev bison flex libssl-dev libelf-dev

-
- **Obtain the Kernel Source:**
  Download the kernel source for the version you want to modify. For example, you can download it from kernel.org or use your distribution's source package.

---

2. Create Your New System Call

a. Write the System Call Function

Inside the kernel source tree, choose a source file (or create a new one) where you will implement your syscall. For example, you might add a new file under `kernel/` or use an existing one. A simple example of a syscall that logs a message could look like this:

```
// File: kernel/helloworld.c
#include <linux/kernel.h>
#include <linux/syscalls.h>

SYSCALL_DEFINE0(helloworld)
{
   printk(KERN_INFO "Hello from the new syscall!\n");
   return 0;
}
```

- **Explanation:**
  - `SYSCALL_DEFINE0(helloworld)` declares a syscall with no arguments.

- Inside the function, `printk` is used to print a kernel log message.

b. Declare the Prototype

Add a prototype for your syscall in an appropriate header file (often in `include/linux/syscalls.h`):

asmlinkage long sys_helloworld(void);

c. Add Your System Call to the Syscall Table

For the target architecture (e.g., x86_64), locate the syscall table. For example, on x86_64 this file is usually found at:
`arch/x86/entry/syscalls/syscall_64.tbl`

Add a new entry for your syscall, choosing an unused syscall number. For example:

333 common helloworld __x64_sys_helloworld

- **Note:**
  The syscall number (here 333) must be unique and not conflict with other syscalls. The naming convention (`__x64_sys_helloworld`) must match your function's exported name (typically generated from `SYSCALL_DEFINE0`).

---

3. Rebuild and Install Your Kernel

a. Configure the Kernel

- Run `make menuconfig` (or another configuration tool) if you need to adjust settings.

b. Compile the Kernel

Compile your kernel and modules:
make -j$(nproc)
make modules_install
make install

-

c. Reboot

- Update your bootloader if necessary, then reboot into the new kernel.

---

4. Write a User-Space Test Program

Now that the new syscall is part of the kernel, write a simple user-space program that calls it.

```
// File: user_test_syscall.c
#include <stdio.h>
#include <sys/syscall.h>
#include <unistd.h>

// Define the syscall number you added (must match what you added in the syscall table)
#define __NR_helloworld 333

int main() {
    long result = syscall(__NR_helloworld);
    printf("System call returned: %ld\n", result);
    return 0;
}
```

- **Explanation:**
  - The program uses the `syscall()` function (from `<sys/syscall.h>`) to invoke the new system call.
  - It prints the return value (in this case, expected to be 0).

Compile the Program:

```
gcc -o test_syscall user_test_syscall.c
```

---

5. Run and Verify

Run Your Test Program:

```
./test_syscall
```

-

Check the Kernel Log:

Use `dmesg` to view the kernel log output:

```
dmesg | tail
```

- You should see your "Hello from the new syscall!" message if everything worked correctly.

---

6. Troubleshooting and Tips

- **Safety First:**
  Always test on a non-production system (a VM or test machine) since kernel modifications can lead to system instability.

- **Debugging:**
  - Use `printk` generously to log messages inside your syscall implementation.
  - Check the output of `dmesg` for any error messages.
- **Documentation:**
  Consult the Linux kernel documentation (found in the `Documentation/` directory of the kernel source) for additional details on syscalls and kernel development.

- **Permissions:**
  Running the user-space program generally does not require root privileges if the syscall does not perform privileged operations. However, installing and booting a new kernel typically requires administrative rights.

Summary

1. **Set up** your development environment and download the kernel source.
2. **Implement** your new syscall in the kernel source (write the function, declare its prototype, and add it to the syscall table).
3. **Rebuild and install** your kernel.
4. **Write, compile, and run** a user-space test program that calls your new syscall.
5. **Verify** the syscall's behavior using `dmesg` and the program output.