### **User Defined System Call: Graceful Shutdown**

### **Team:**

Arjun – 22i305

Dhanush – 22i308

Gokul – 22i313

Malli Santosh – 22i330

Nitish - 22i337

### **What is a User-Defined System Call?**

**Definition**:

* System calls allow programs to interact with the operating system.
* User-defined system calls involve creating or modifying system behaviors with custom code.

**Example Uses**:

* File I/O, process management, hardware access.

### **Concept: Graceful Shutdown Using a Signal Handler**

**Definition**:

* Handling SIGINT to perform cleanup before shutdown.

**Goal**:

* Save data and gracefully shut down when Ctrl+C is pressed.

### **Overview of the Custom System Call**

**Objective**:

* Create a custom system call in Linux that saves data and initiates system shutdown.

**Key Steps**:

1. Define the system call in the kernel.
2. Modify the syscall table.
3. Recompile the kernel.
4. Call the system call from user space.

### **Define the System Call**

**Creating the System Call in the Kernel (mycall.c)**:

#include <linux/kernel.h>  
#include <linux/syscalls.h>  
#include <linux/fs.h>  
#include <linux/uaccess.h>  
#include <linux/fcntl.h>  
  
SYSCALL\_DEFINE0(graceful\_shutdown) {  
 struct file \*file;  
 mm\_segment\_t old\_fs;  
 char buf[256];  
 char \*filepath = "/home/user/progress.txt";  
 ssize\_t ret;  
 unsigned long current\_time = (unsigned long)ktime\_get\_real\_seconds();  
  
 snprintf(buf, sizeof(buf), "Progress saved at: %lu seconds\n", current\_time);  
  
 old\_fs = get\_fs();  
 set\_fs(KERNEL\_DS);  
  
 file = filp\_open(filepath, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);  
 if (IS\_ERR(file)) {  
 printk(KERN\_ERR "Error opening file for saving progress.\n");  
 set\_fs(old\_fs);  
 return -1;  
 }  
  
 ret = kernel\_write(file, buf, strlen(buf), &file->f\_pos);  
 if (ret < 0) {  
 printk(KERN\_ERR "Error writing progress to file.\n");  
 filp\_close(file, NULL);  
 set\_fs(old\_fs);  
 return -1;  
 }  
  
 filp\_close(file, NULL);  
 set\_fs(old\_fs);  
 printk(KERN\_INFO "Progress successfully saved to progress.txt.\n");  
  
 kernel\_power\_off();  
 return 0;  
}

**Explanation**:

* Saves progress to a file (progress.txt).
* Calls kernel\_power\_off() to shut down the system.

### **Step 2 - Modify the Syscall Table**

**Adding the System Call to the Syscall Table**:

1. Open the syscall table:
   1. File: arch/x86/entry/syscalls/syscall\_64.tbl
2. Add an entry for your system call:

bash

549 64 graceful\_shutdown sys\_graceful\_shutdown

**Note**: Replace 549 with the next available syscall number.

### **Step 3 - Configure the Kernel**

**Steps to Configure the Kernel**:

1. Navigate to the kernel source directory:

cd /usr/src/linux

1. Clean the source tree (optional but recommended for a fresh start):

make clean

1. Update the kernel configuration to include your custom changes:

make menuconfig

1. Save and exit the configuration tool.

### **Step 4 - Compile the Kernel**

**Commands to Compile the Kernel**:

1. Compile the kernel:

make -j$(nproc)

**Explanation**:

$(nproc) automatically sets the number of jobs to the number of CPU cores, speeding up the process.

1. Compile and install the modules:

sudo make modules\_install

### **Step 5 - Install and Reboot the Kernel**

**Commands to Install and Boot the New Kernel**:

1. Install the new kernel:

sudo make install

1. Update GRUB to recognize the new kernel:

sudo update-grub

1. Reboot the system to use the new kernel:

sudo reboot

### **Step 6 - Call the System Call from User Space**

**User-Space Program to Invoke System Call**:

#include <stdio.h>  
#include <unistd.h>  
#include <sys/syscall.h>  
#include <errno.h>  
  
int main() {  
 long res = syscall(549); // Replace 549 with your system call number  
  
 if (res == -1) {  
 perror("syscall");  
 } else {  
 printf("System call executed successfully.\n");  
 }  
  
 return 0;  
}

A computer screen shot of code

Description automatically generated

A computer screen shot of code

Description automatically generated

### **Step 7 - Compile and Run the User-Space Program**

**Compile the User-Space Program**:

gcc test\_graceful\_shutdown.c -o test\_graceful\_shutdown

**Run the Program**:

./test\_graceful\_shutdown

* The program calls the custom system call to save progress and initiate shutdown.

### **Modifying sudoers for Password-Free Shutdown**

**Allow Shutdown Without Password**:

* Open the sudoers file:

sudo visudo

* Add this line (replace your\_username with your actual username):

your\_username ALL=(ALL) NOPASSWD: /sbin/shutdown

A computer screen shot of a program code

Description automatically generated

### **Use Case and Benefits of Graceful Shutdown**

**Use Case**:

* Server applications needing a controlled shutdown process.

**Benefits**:

* **Data Integrity**: Saves progress before shutting down, preventing data loss.
* **Resource Management**: Releases resources (files, connections) during shutdown.
* **Automation**: Allows the system to automatically manage shutdown gracefully.

### **Output During Execution**

**Example Output**:

* On SIGINT (Ctrl+C):

kotlin

Copy code

Caught SIGINT, saving data and shutting down...

* Progress saved to progress.txt, then system shutdown is initiated.

### **Conclusion**

**Summary**:

* Implemented a custom system call in Linux.
* Demonstrated how to handle signals and initiate a graceful shutdown.
* Enhanced control over system behavior, ensuring data integrity and proper resource management.

**Final Note**:

* Custom system calls and signal handling are powerful Linux features for creating system-level control.

**GitHub Link:**

<https://github.com/Malli-Santhosh/Linux-Custom-Signals>