**Adding a Custom System Call in the Linux Kernel (Version 6.5.7)**

TEAM MEMBERS:

**ANUKEERTHANA - 22i204**

**ASHWIN - 22i207**

**SHYAM SARAN -22i257**

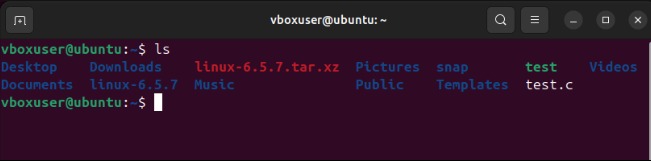
**SOWBARNIKAA – 22i262**

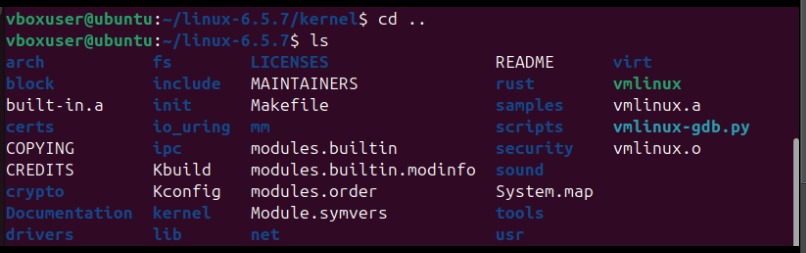
**VINU CHANDRAN – 22i273**

# Introduction

The Linux kernel is a powerful and flexible operating system core that interacts directly with the hardware and manages system resources. One of the key interfaces between user applications and the kernel is the system call. System calls allow user-space programs to request services from the kernel. This report will explain, in detail, the step-by-step process of adding a new system call to the Linux kernel. In this case, Linux kernel version 6.5.7 is used for the demonstration. The steps include downloading the kernel, making source code changes, compiling, and testing the custom system call.

# Step 1: Downloading and Setting Up Linux in a Virtual Machine

To begin, the current version of Linux is installed on a virtual machine using VirtualBox. This virtual machine provides a controlled environment where kernel modifications can be made without affecting the primary system. The first task is to download and set up Linux on this virtual machine.  
  
 Once the virtual machine is running with Linux installed, the next step is to download the specific kernel version, 6.5.7. This version is chosen for its recent updates and compatibility with the virtual machine setup. The kernel source code for this version is downloaded as a compressed tar file from the official Linux repository. After the download, the tar file is extracted, creating a directory containing all the necessary files for kernel development. 



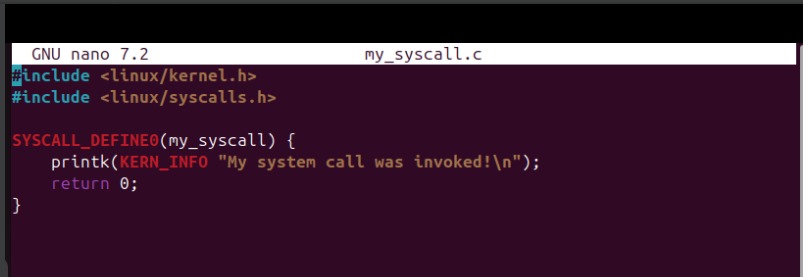
# Step 2: Navigating to the Kernel Directory

After extracting the kernel files, it is important to navigate through the directory structure to locate where system calls are defined. In Linux, system calls for different architectures are stored in specific subdirectories. For x86 architectures (commonly used on modern computers), the system call implementation files are located in the `arch/x86/entry/syscalls/` directory.  
  
 In this guide, system calls are added to the x86\_64 architecture (64-bit systems). Therefore, we will be working inside this directory to define and register a new system call.



# Step 3: Creating a New System Call (my\_syscall.c)

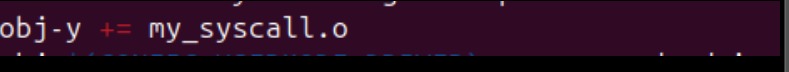
The core of this project is to create a new system call. This involves writing a C file that defines the behavior of the system call. Inside the Linux kernel source code, navigate to the `kernel/` folder, which is where custom kernel code can be placed.  
  
 To implement the system call, create a new C file called `my\_syscall.c`. This file will contain a function that prints a simple message to the kernel logs, confirming that the system call has been successfully invoked. The function then returns 0, indicating success.

  
Once this file is saved, the kernel now has the logic for a custom system call, but it still needs to be linked into the rest of the system.

# Step 4: Editing the Makefile

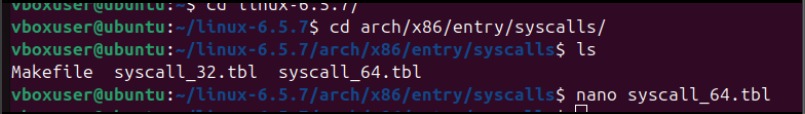
The next step is to ensure that the newly created system call is compiled into the kernel. This is done by adding the new C file (`my\_syscall.c`) to the kernel’s Makefile, which controls the build process.  
  
 To do this, open the main Makefile of the Linux kernel located in the `kernel/` directory. Search for the section that lists object files and add the corresponding object file for `my\_syscall.c` so that the kernel builds it.

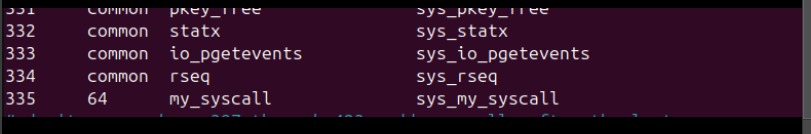




# Step 5: Modifying the System Call Table (syscall\_64.tbl)

After defining the system call logic and ensuring that it will be compiled, the next step is to register it by assigning a unique system call number. System call numbers map specific function calls in user space to corresponding kernel functions.  
  
Navigate to the `arch/x86/entry/syscalls/` directory, which contains the file `syscall\_64.tbl`. This file lists all system calls available for 64-bit systems, along with their corresponding numbers.





# Step 6: Building the Kernel

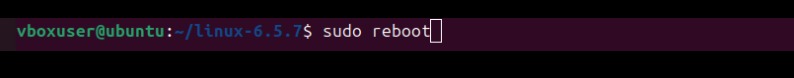
Now that the system call is defined, registered, and included in the kernel build process, it is time to compile the kernel. Building the Linux kernel can be a time-consuming process depending on system resources, so patience is required.  
  
To compile the kernel, execute the following commands:







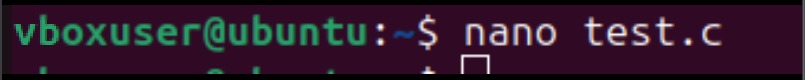
  
  
make defconfig  
make modules\_install  
sudo make install  
  
After the kernel is installed, reboot the system to load the new kernel.

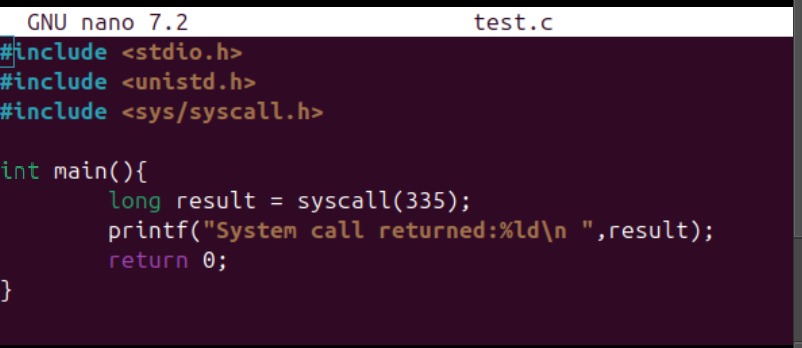


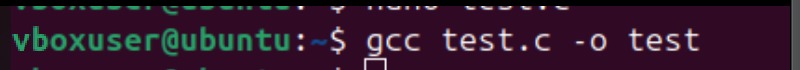
# Step 7: Testing the System Call

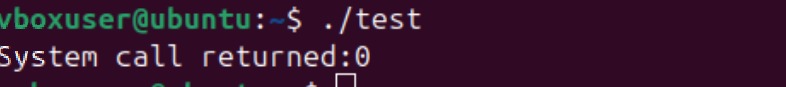
After rebooting into the new kernel.

Create a simple C program in user space to test the system call. This program will invoke the custom system call and check the return value.  
  
 This program uses the syscall function to invoke the system call with number 335 (as assigned earlier). It prints the result returned by the system call.

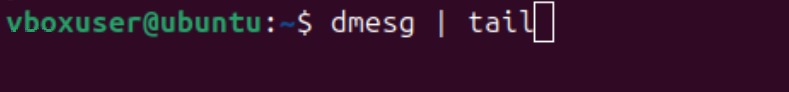


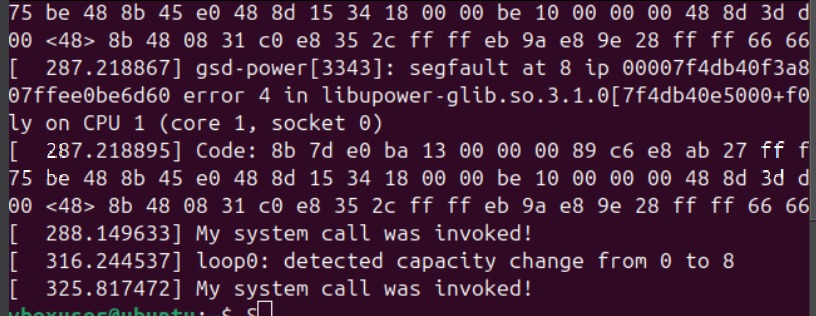






**OUTPUT :**





# Steps to Contribute Changes to the Official Linux Kernel

Contributing changes to the official Linux kernel is a structured process that involves collaboration and adherence to strict coding standards. This section outlines the detailed steps required to successfully submit modifications or enhancements to the mainline Linux kernel.

## Step 1: Understand the Contribution Guidelines

Before proceeding with any changes, it is essential to thoroughly understand the [Linux Kernel Contributor Guide](https://www.kernel.org/doc/html/latest/process/submit-checklist.html). This guide explains the submission process, coding standards, and how to interact with the Linux community. Adhering to the code of conduct is crucial in maintaining a positive and collaborative environment.

## Step 2: Set Up the Development Environment

Ensure the development environment is properly configured with necessary tools such as `gcc`, `make`, `git`, and other kernel development dependencies. Once the environment is ready, the first step is to download the latest Linux kernel source code using the following command:  
  
```  
git clone https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git  
cd linux  
```  
  
This will clone the latest version of the Linux kernel from the official repository.

## Step 3: Make the Required Changes

When making any changes, it is important to follow the Linux kernel's strict [coding style guidelines](https://www.kernel.org/doc/html/latest/process/coding-style.html). The changes should be tested locally to ensure they are necessary and functional without breaking existing kernel features.   
  
Once the required modifications are complete, they must be tested by building and running the updated kernel.

## Step 4: Prepare the Patch

After making changes, they should be committed locally following the kernel's commit message guidelines. Each commit must include a clear, concise description of the changes made. Use the `-s` flag to sign off the commit, indicating agreement to the Developer Certificate of Origin (DCO):  
  
```  
git add <modified\_files>  
git commit -s  
```  
  
Next, format the patch using `git format-patch` to prepare it for submission:  
  
```  
git format-patch origin  
```

## Step 5: Identify and Contact the Right Maintainers

It is important to send the patch to the correct maintainers responsible for the subsystem that was modified. This can be done using the `get\_maintainer.pl` script to identify the appropriate maintainers:  
  
```  
./scripts/get\_maintainer.pl <modified\_files>  
```  
  
Once the maintainers are identified, the patch can be sent via email using `git send-email`, ensuring it is formatted as plain text. The patch should include a detailed explanation of the changes:  
  
```  
git send-email --to=<maintainer-email> \*.patch  
```

## Step 6: Respond to Feedback

After submission, maintainers and other developers will review the patch and provide feedback. Responding to the feedback in a timely manner is important, making the necessary revisions and resubmitting the patch if needed. Revisions should be indicated as new versions (e.g., `[PATCH v2]`).

## Step 7: Wait for Merge

Once the patch is accepted, it will be merged into the maintainers’ subsystem tree and eventually into the mainline Linux kernel. The progress can be tracked through the Linux mailing lists and repositories.

# Conclusion

This report outlines the detailed process of adding a custom system call to the Linux kernel. By carefully following these steps, we demonstrated how to download, modify, and compile the kernel with new functionality. This method serves as a foundational example, showing how custom features can be added to the Linux kernel and how the kernel’s extensibility allows for the integration of user-defined logic.