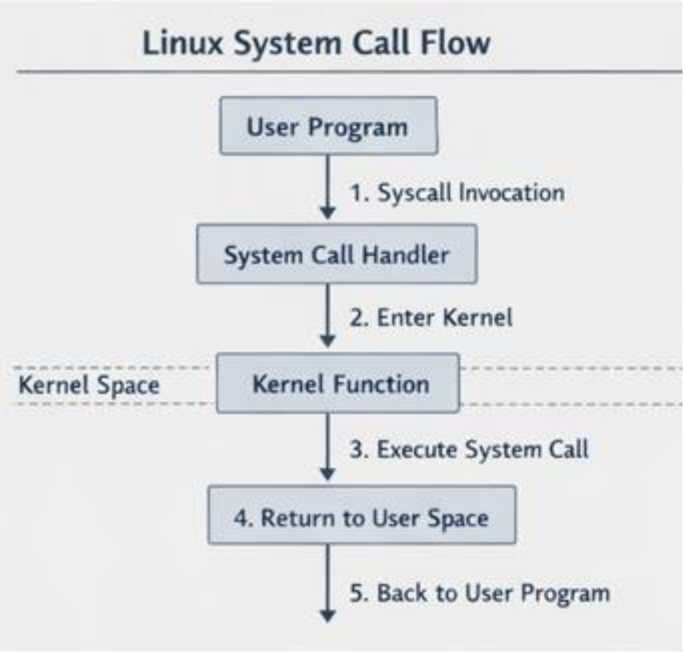
Developing a More Complex Custom System Call in Linux

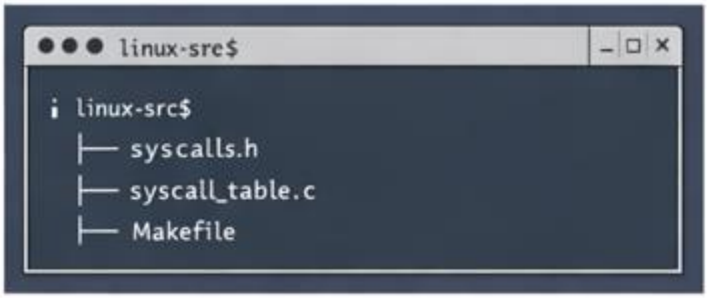
1. Introduction

System calls provide the interface between user-space applications and the Linux kernel. In this experiment, a custom Linux system call is implemented to reverse a string passed from user space and return the reversed string back safely. This task demonstrates kernel-level string manipulation, memory safety, and data transfer between user and kernel spaces.



2. Understanding System Calls

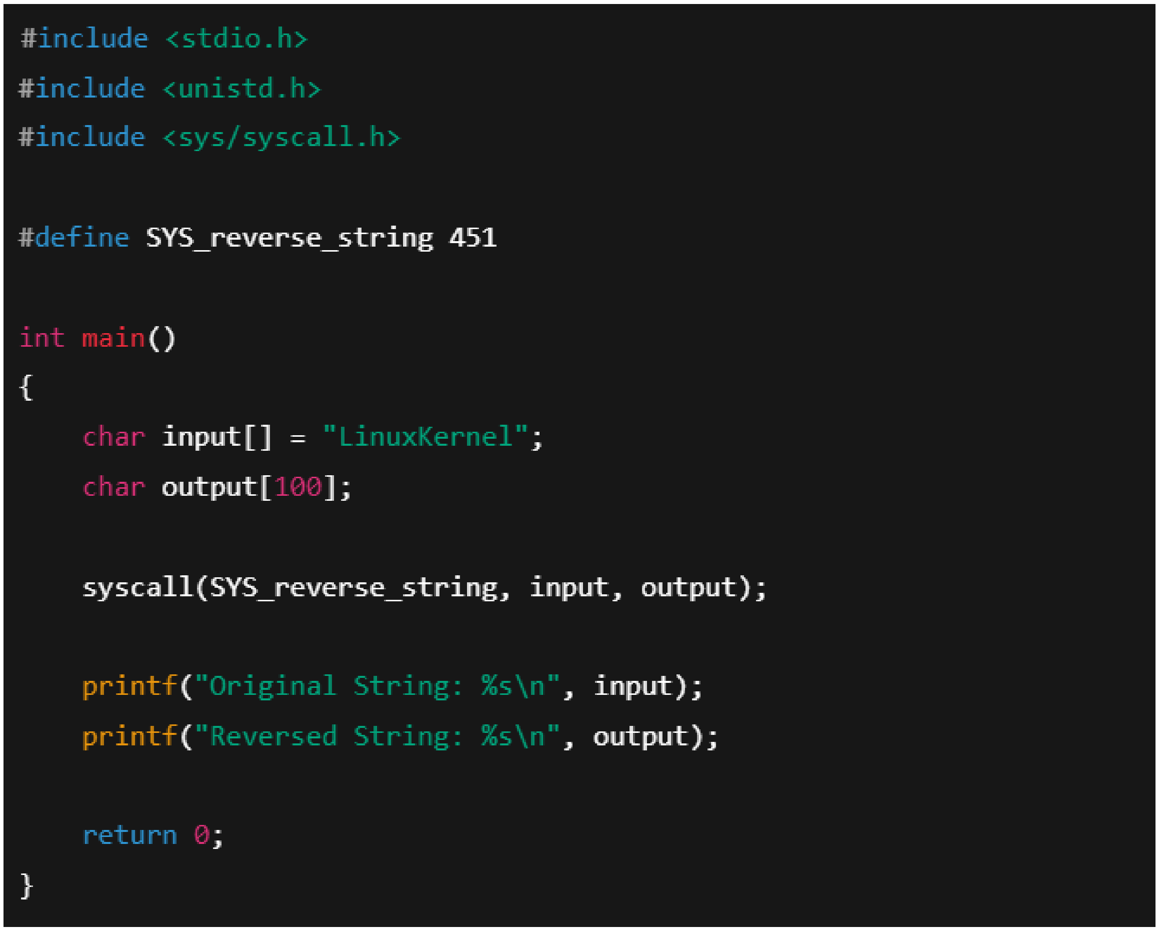
System calls allow user programs to request services from the kernel. They operate through a controlled interface ensuring security and stability. Custom system calls require kernel source modification and recompilation.



```
i linux-src$
  ├── syscalls.h
  ├── syscall_table.c
  ├── Makefile
```

3. Modifying the Linux Kernel Source Code

To add a custom system call, the Linux kernel source must be modified. This includes defining the system call, adding it to the syscall table, and exposing it via appropriate headers.



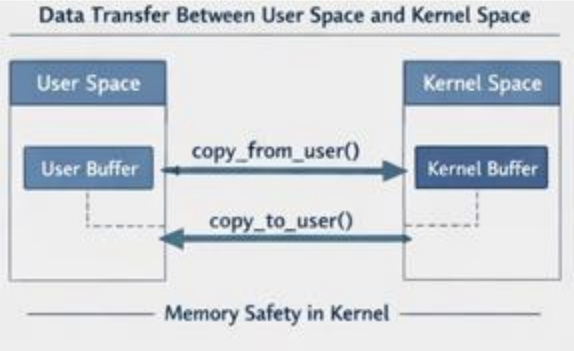
```
#include <stdio.h>
#include <unistd.h>
#include <sys/syscall.h>

#define SYS_reverse_string 451

int main()
{
    char input[] = "LinuxKernel";
    char output[100];

    syscall(SYS_reverse_string, input, output);

    printf("Original String: %s\n", input);
    printf("Reversed String: %s\n", output);

    return 0;
}
```

4. System Call Design

The custom system call accepts two user-space pointers: one for the input string and one for storing the reversed output string. Proper validation and memory handling are essential to prevent kernel crashes.

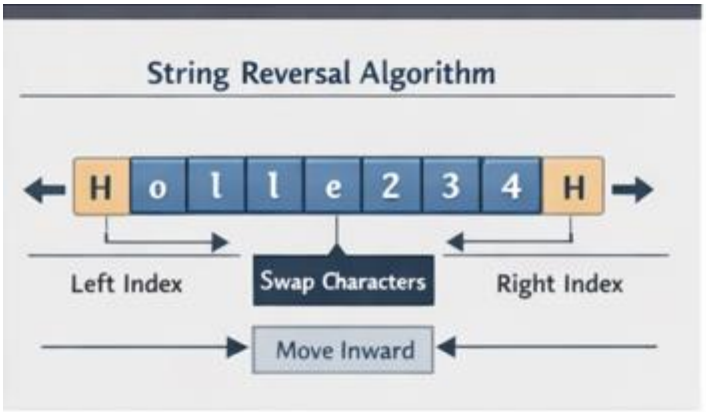
5. Handling User and Kernel Space Data

Direct access to user-space memory is unsafe in kernel space. Functions such as copy_from_user() and copy_to_user() are used to safely transfer data between user space and kernel space.



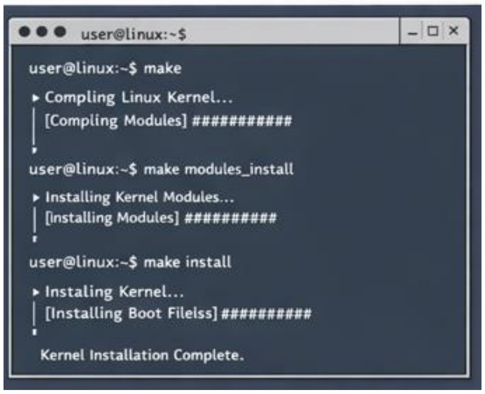
6. String Reversal Logic in Kernel Space

After copying the string into kernel space, the string is reversed using a simple two-pointer approach. Care is taken to ensure proper memory allocation and null termination.



7. Rebuilding and Installing the Kernel

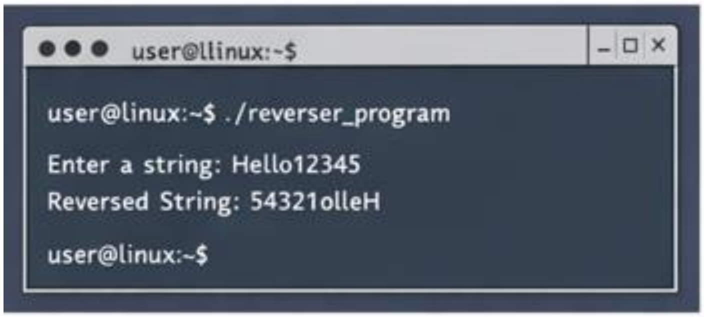
Once the system call is implemented, the kernel must be rebuilt and installed. This process involves compiling the kernel, installing modules, and rebooting into the modified kernel.



```
user@linux:~$ make
 ▸ Compling Linux Kernel...
 │ [Compling Modules] ###########
 ╵
user@linux:~$ make modules_install
 ▸ Installing Kernel Modules...
 │ [Installing Modules] ##########
 ╵
user@linux:~$ make install
 ▸ Instaling Kernel...
 │ [Installing Boot Fileiss] ##########
 ╵
  Kernel Installation Complete.
```

8. Testing the Custom System Call

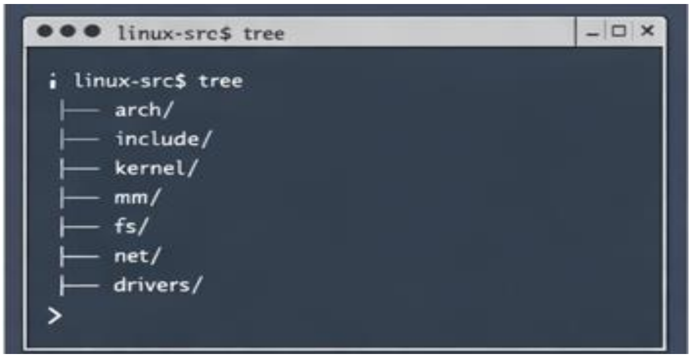
A user-space C program is written to invoke the custom system call. The program passes a string and receives the reversed string as output, verifying the correctness of the implementation.



```
user@linux:~$ ./reverser_program

Enter a string: Hello12345
Reversed String: 54321olleH

user@linux:~$
```

9. Workflow Summary

The complete workflow includes kernel modification, system call implementation, kernel compilation, and user-space testing. Each step ensures correct interaction between user space and kernel space.



```
¡ linux-src$ tree
 ├── arch/
 ├── include/
 ├── kernel/
 ├── mm/
 ├── fs/
 ├── net/
 ├── drivers/
>
```