Task 14: Developing a Custom Linux System Call for String Reversal

Objective:
The objective of this task is to develop a custom system call in the Linux kernel that performs a non-trivial operation. The system call accepts a string from user space, reverses the string safely inside kernel space, and returns the reversed string back to user space. This task demonstrates kernel modification, safe memory handling, and controlled data transfer between user space and kernel space.

1. Background: User Space vs Kernel Space

Linux strictly separates user space and kernel space to ensure system stability and security. User applications cannot directly access kernel memory or hardware. System calls provide a controlled interface that allows user programs to request services from the kernel. Data transfer between these spaces must be done using special kernel functions.

2. Overview of the Implementation

This task involves the following steps:
- Adding a new system call to the Linux kernel source
- Accepting user-space pointers as arguments
- Copying data safely from user space to kernel space
- Performing string reversal in kernel space
- Copying the result back to user space
- Rebuilding and booting the modified kernel

3. Prerequisites

Before starting the implementation, the following prerequisites are required:
- Linux system (preferably Ubuntu) running on a virtual machine
- Kernel source code and headers installed
- build-essential package installed
- Root or sudo privileges

4. Creating the System Call Source File

A new source file is created inside the kernel source tree. The system call is defined using the SYSCALL_DEFINE macro, which automatically handles name mangling and calling conventions.

```
#include <linux/kernel.h>
 #include <linux/syscalls.h>
 #include <linux/uaccess.h>

 SYSCALL_DEFINE2(reverse_string, char __user *, input, char __user *, output)
 {
        char kbuf[256];
        int len, i;

        if (copy_from_user(kbuf, input, sizeof(kbuf)))
        return -EFAULT;

        kbuf[255] = '\0';
        len = strlen(kbuf);

        for (i = 0; i < len / 2; i++) {
        char tmp = kbuf[i];
        kbuf[i] = kbuf[len - i - 1];
        kbuf[len - i - 1] = tmp;
        }

        if (copy_to_user(output, kbuf, len + 1))
          return -EFAULT;

        return 0;
 }
```

5. Registering the System Call

The new system call must be registered in the architecture-specific system call table so the kernel can map a system call number to its implementation.

Example entry in arch/x86/entry/syscalls/syscall_64.tbl:

```
 548    common       reverse_string __x64_sys_reverse_string
```

A free and unused system call number must be selected.

6. Declaring the System Call Prototype

The system call prototype is declared in include/linux/syscalls.h. This ensures the kernel build system recognizes the new system call.

```
asmlinkage long sys_reverse_string(char __user *input, char __user *output);
```

7. Updating the Kernel Makefile

The kernel Makefile must be updated so the new system call source file is compiled and linked into the kernel.

```
obj-y += reverse_string_syscall.o
```

8. Memory Handling and Data Transfer

Kernel code cannot directly access user-space memory. The copy_from_user() function safely copies data from user space into kernel buffers, while copy_to_user() copies results back to user space. These functions also perform access checks to prevent invalid memory access.

9. Rebuilding and Loading the Kernel

After adding the system call, the kernel must be rebuilt and installed. The system must then be rebooted to load the modified kernel so the new system call becomes available.

10. User-Space Test Program

A user-space C program is written to invoke the custom system call using the syscall() function. The program passes a string to the kernel and receives the reversed string.

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

 #define SYS_reverse_string 548

 int main() {
        char input[256] = "LinuxKernel";
        char output[256];

        syscall(SYS_reverse_string, input, output);
        printf("Reversed string: %s\n", output);
        return 0;
 }
```

11. Expected Output

When the user program is executed, the output will display the reversed string returned from kernel space, confirming correct operation.

12. Conclusion

This task demonstrates the complete process of implementing a custom Linux system call. It covers kernel modification, safe string manipulation, memory protection, and controlled communication between user space and kernel space, which are fundamental concepts in Linux system programming.