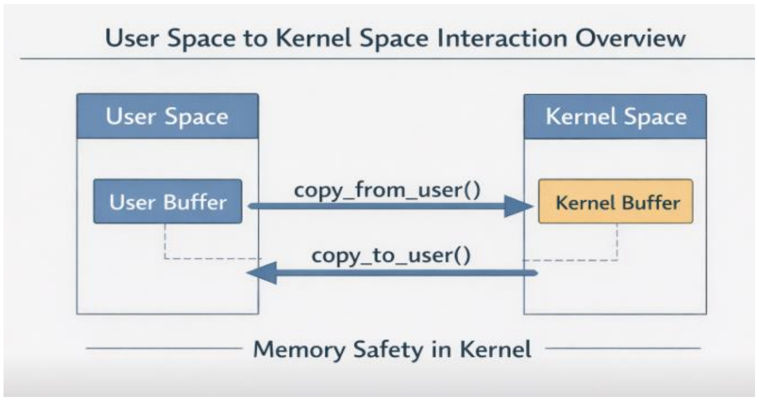
User-Space Program to Test the Custom System Call

1. Introduction

After implementing the custom system call in the Linux kernel, a user-space program is required to test its functionality. This program interacts with the kernel using the system call interface to pass a string and receive the reversed result.

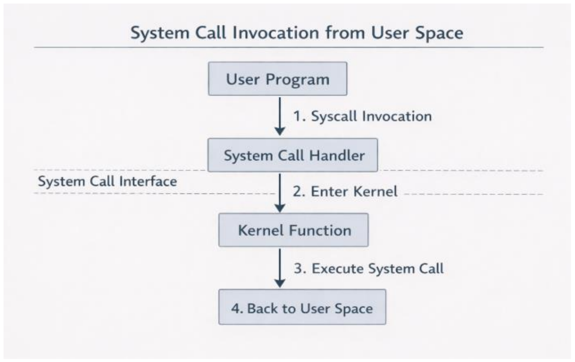


2. Objective

The objectives of this experiment are:
- To write a user-space C program to invoke the custom system call.
- To accept a string input from the user.
- To display the reversed string returned by the kernel.
- To implement proper error handling and memory management.

3. System Call Interface Overview

The user-space program uses the syscall() function to invoke the custom system call by passing the system call number and required parameters. This ensures controlled interaction between user space and kernel space.



4. Program Design

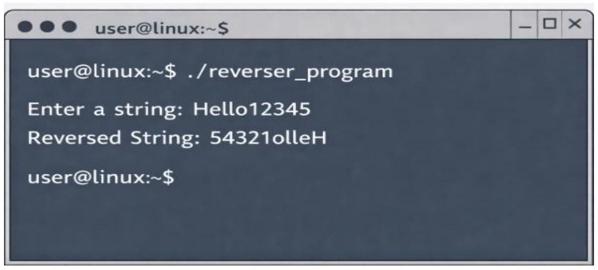
The program follows a structured approach:
1. Prompt the user to enter a string.

2. Allocate memory dynamically for input and output buffers.
3. Invoke the custom system call.
4. Handle errors appropriately.
5. Display the reversed string.
6. Free allocated memory before termination.

5. User-Space C Program

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

#define SYS_reverse_string 451

int main()
{
  char *input;
  char *output;
  int ret;

  input = (char *)malloc(256);
  output = (char *)malloc(256);

  if (!input || !output) {
    perror("Memory allocation failed");
    return 1;
  }

  printf("Enter a string: ");
  fgets(input, 256, stdin);
  input[strcspn(input, "\n")] = '\0';

  ret = syscall(SYS_reverse_string, input, output);

  if (ret < 0) {
    perror("System call failed");
    free(input);
    free(output);
    return 1;
  }
```

```
    printf("Reversed string: %s\n", output);

    free(input);
    free(output);
    return 0;
}
```



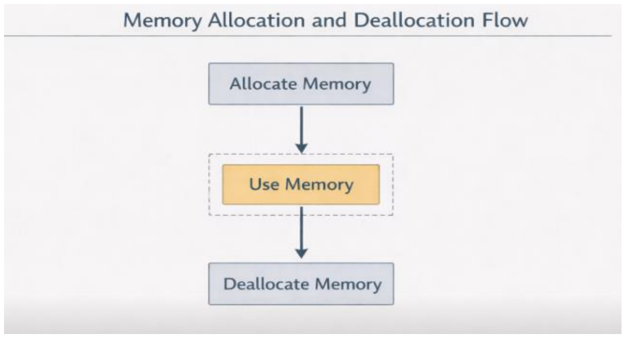
```
user@linux:~$ ./reverser_program
Enter a string: Hello12345
Reversed String: 54321olleH
user@linux:~$
```

6. Error Handling

The program validates memory allocation and checks the return value of the system call. Errors are reported using perror(), ensuring meaningful diagnostics.

7. Memory Management

Dynamic memory allocation is used to store input and output strings. All allocated memory is freed before program termination to avoid memory leaks.

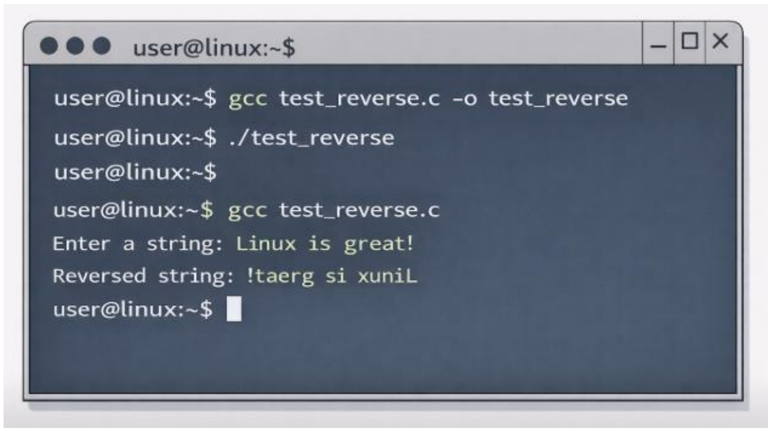


8. Compilation and Execution

The user-space program is compiled using GCC and executed after booting into the modified kernel.

Compilation Command:
```
gcc test_reverse.c -o test_reverse
```

Execution Command:
```
./test_reverse
```



```
user@linux:~$ gcc test_reverse.c -o test_reverse
user@linux:~$ ./test_reverse
user@linux:~$
user@linux:~$ gcc test_reverse.c
Enter a string: Linux is great!
Reversed string: !taerg si xuniL
user@linux:~$ 
```