Task 15: User-Space Program to Test a Custom Linux System Call

Objective:
The objective of this task is to write a user-space C program that tests a custom Linux system call developed earlier. The program prompts the user to enter a string, passes it to the custom system call for reversal, and prints the reversed string returned from kernel space. This task demonstrates correct system call invocation, memory management, and error handling in user space.

1. Background

User-space programs interact with the Linux kernel through system calls. A custom system call added to the kernel can be invoked from user space using the syscall() function along with the assigned system call number. Since kernel and user spaces are isolated, careful memory handling is required.

2. Requirements

The user-space test program must:
- Prompt the user to enter a string
- Allocate memory dynamically for input and output buffers
- Invoke the custom system call correctly
- Handle errors returned by the system call
- Display the reversed string returned from kernel space

3. System Call Number

The custom system call is identified by a unique system call number. This number must match the number assigned in the kernel syscall table.

Example:
```
#define SYS_reverse_string 548
```

4. User-Space Program Code

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

 #define SYS_reverse_string 548

 int main() {
          char *input, *output;
          size_t size = 256;

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

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

          printf("Enter a string: ");
          fgets(input, size, stdin);

          if (syscall(SYS_reverse_string, input, output) < 0) {
        perror("System call failed");
          free(input);
          free(output);
          return 1;
          }

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

          free(input);
          free(output);

          return 0;
}
```

5. Explanation of the Program

• Dynamic memory is allocated using malloc() for both input and output buffers.
• fgets() is used to safely read the input string from the user.
• syscall() is used to invoke the custom system call.
• The return value of syscall() is checked to detect errors.
• perror() is used to display descriptive error messages.
• Allocated memory is freed before program termination.

6. Error Handling

Proper error handling is essential when invoking system calls. The program checks for memory allocation failures and system call errors. If an error occurs, appropriate messages are displayed and resources are released.

7. Memory Management

Memory is dynamically allocated to ensure flexibility and safety. The allocated buffers are freed using free() to avoid memory leaks. Dynamic memory allocation is preferred when handling user input of variable length.

8. Compilation and Execution

Compile the program using a C compiler:

```
gcc test_reverse_syscall.c -o test_reverse_syscall
```

Run the program:

```
./test_reverse_syscall
```

9. Expected Output

Example:
Enter a string: LinuxSystem
Reversed string: metsySmuxiL

10. Conclusion

This task completes the user-space validation of the custom Linux system call. The program demonstrates safe memory management, proper error handling, and correct interaction between user space and kernel space. Such testing is essential to ensure kernel modifications work as intended.