**Implementation of rename() System Call in Lubuntu**

Lubuntu is a lightweight, fast, and energy-efficient Linux-based operating system built on the Ubuntu core but designed to run smoothly on older or low-spec hardware. It uses the LXQt desktop environment, which is clean, simple, and requires fewer system resources. Lubuntu is ideal for students, developers, or anyone looking for a free and reliable system that boots quickly and handles daily tasks like browsing, writing, coding, and more. It comes with essential applications pre-installed and supports thousands more through the Ubuntu Software Center.

Installing Lubuntu is a straightforward process once you have the ISO file and a bootable USB ready. The system uses the ext4 file system by default, ensuring stability, speed, and long-term support. You can choose to erase your disk or install it alongside another operating system, like Windows. The installer walks you through each step—from selecting your language to creating a user account. Once installed, Lubuntu offers a clean interface and smooth experience, even on computers with limited RAM or old processors.

In this project, I implemented the rename() system call on a custom Linux kernel used by Lubuntu. The goal was to understand how system calls work by creating my own version of rename() and integrating it into the Linux kernel source. This document explains the process step by step, including setup, code implementation, compilation, and testing

Well-Organized Implementation of my\_rename() System Call in Lubuntu

**1. Create the Source File**

Navigate to the fs/ directory inside the kernel source and create a new file:

cd ~/kernel/linux-5.10/fs touch my\_rename.c

2. **Code: my\_rename.c**

#include <linux/kernel.h>

#include <linux/syscalls.h>

#include <linux/fs.h>

#include <linux/namei.h>

#include <linux/uaccess.h>

SYSCALL\_DEFINE2(my\_rename, const char \_\_user

\*oldname, const char \_\_user \*newname) {

char old[PATH\_MAX], new[PATH\_MAX];

struct path old\_path, new\_path;

int err;

// Copy file paths from user space

if (copy\_from\_user(old, oldname, PATH\_MAX)

|| copy\_from\_user(new, newname, PATH\_MAX))

return -EFAULT;

// Get the parent directory of the old path

err = kern\_path(old, LOOKUP\_PARENT, &old\_path);

if (err)

return err;

// Get the parent directory of the new path

err = kern\_path(new, LOOKUP\_PARENT, &new\_path);

if (err) {

path\_put(&old\_path);

return err;

}

// Perform rename operation

err = vfs\_rename(d\_inode(old\_path.dentry),

old\_path.dentry,

d\_inode(new\_path.dentry), new\_path.dentry,

NULL, 0);

// Release resources

path\_put(&old\_path);

path\_put(&new\_path);

return err;

}

**3. Register the System Call**

a. Add to Makefile

Edit fs/Makefile and add:

obj-y += my\_rename.o

b. Add Prototype

In include/linux/syscalls.h, add:

asmlinkage long sys\_my\_rename(const char \_\_user

\*oldname, const char \_\_user \*newname);

**c. Add to Syscall Table**

Edit arch/x86/entry/syscalls/syscall\_64.tbl:

440 common my\_rename \_\_x64\_sys\_my\_rename

4. **Recompile and Reboot**

make -j$(nproc)

sudo make modules\_install

sudo make install

sudo update-grub

sudo reboot

**5. Test the System Call**

Create test program test.c:

#include <stdio.h>

#include <unistd.h>

#include <sys/syscall.h>

#define \_\_NR\_my\_rename 440

int main() {

long result = syscall(\_\_NR\_my\_rename, "old.txt", "new.txt");

if (result == 0)

printf("Rename successful\n");

else

perror("Rename failed");

return 0;

}

**Compile and Run:**

gcc test.c -o test ./test

Conclusion

Through this project, I learned how system calls are handled by the Linux kernel. By implementing a simplified version of rename(), I understood file system handling, kernel space vs user space communication, and the process of customizing the kernel. This project helped me gain practical experience in kernel development Virtualization also plays a key role in cloud computing, which relies on the dynamic provisioning of computing resources. Cloud service providers use virtualization to offer customers scalable and flexible virtual environments. Customers can deploy virtual machines in the cloud for web hosting, data storage, software development, and more. Without virtualization, cloud computing as we know it would not be possible, as the ability to quickly create, manage, and move virtual environments is essential for cloud infrastructure.