OS Project - Reader Writer Problem

Instructor: Safia Baloch

Submitted by:

1. Fatima Liaquat (2023202)
2. Muhammad Sanawar (2023331)
3. Muhammad Abdullah Farrukh (2023345)

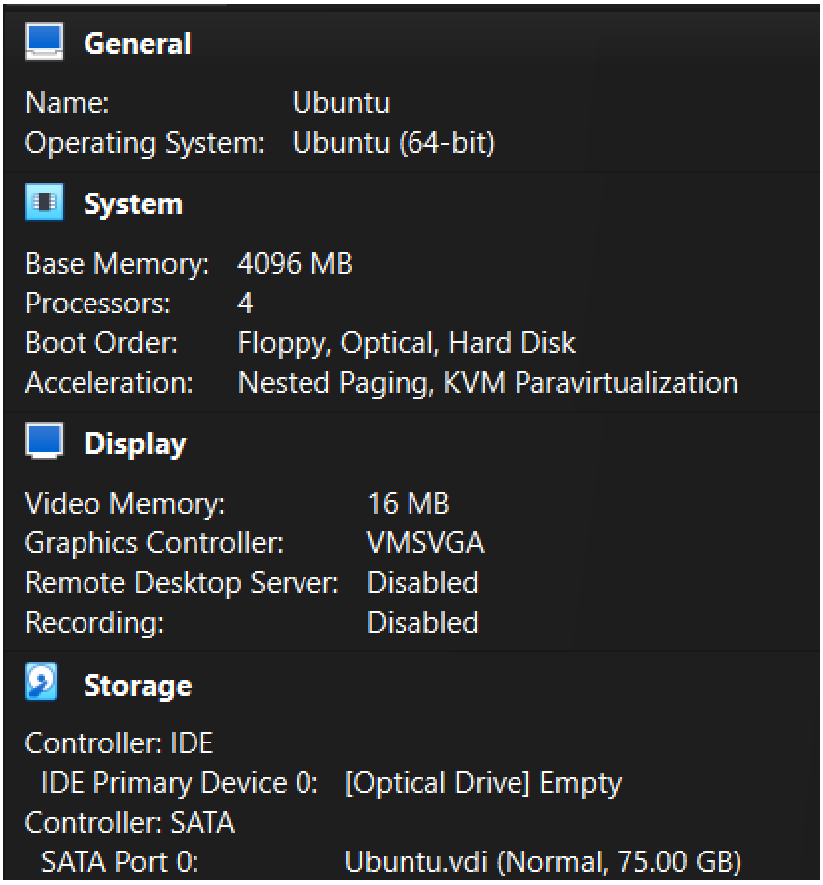
Introduction:

Reader-Writer Problem is a classic problem that occurs in Operating Systems. The problem occurs in a situation when multiple processes, threads or users want to access the same shared resource at the same time. The main idea is to apply a lock whenever a certain process or threads acquires the shared resource and release the lock when the desired operation is formed.

Setup:

Ubuntu is installed on our computers using a virtual machine named [“Virtual Box”]() and a ISO file for [Ubuntu](). We allocated the following resources to the virtual machine:

i. 4 GB RAM
ii. 75 GB Storage Space on SSD
iii. 4 Cores of CPU
iv. 16 MB of Video Memory



General

Name: Ubuntu
Operating System: Ubuntu (64-bit)

System

Base Memory: 4096 MB
Processors: 4
Boot Order: Floppy, Optical, Hard Disk
Acceleration: Nested Paging, KVM Paravirtualization

Display

Video Memory: 16 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage

Controller: IDE
IDE Primary Device 0: [Optical Drive] Empty
Controller: SATA
SATA Port 0: Ubuntu.vdi (Normal, 75.00 GB)

Implementation:

After the installation of Ubuntu, we implmented our project with the following steps:

i. We prepared all the dependencies and installed them using the following commands:

```
sudo apt update

sudo apt install -y \
    build-essential \
    libncurses-dev \
    bison \
    flex \
    libssl-dev \
    libelf-dev \
    bc \
    fakeroot \
    dpkg-dev \
    libncurses5-dev \
    wget \
    git
```

ii. After installing all the dependencies, we cloned the latest version on Linux onto our virtual machine:

```
mkdir ~/kernel-build
cd ~/kernel-build

git                                                          clone
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git  linux
```

iii. After cloning the latest version of Linux, we prepared our kernel for configuration:

```
cd ~/kernel-build/linux
cp /boot/config-$(uname -r) .config
make olddefconfig
mkdir kernel/rw_sync
```

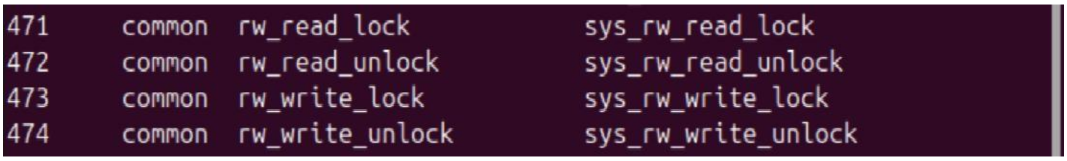
iv. Create the Kernel Source File using the command “nano kernel/rw_sync/rw_syscalls.c”. The code for the file is given below:

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

static DECLARE_RWSEM(rw_lock);

/* Acquire read lock */
SYSCALL_DEFINE0(rw_read_lock)
{
   down_read(&rw_lock);
   printk(KERN_INFO "Reader acquired lock\n");
   return 0;
}

/* Release read lock */
SYSCALL_DEFINE0(rw_read_unlock)
{
   up_read(&rw_lock);
   printk(KERN_INFO "Reader released lock\n");
   return 0;
}

/* Acquire write lock */
SYSCALL_DEFINE0(rw_write_lock)
{
   down_write(&rw_lock);
   printk(KERN_INFO "Writer acquired lock\n");
   return 0;
}

/* Release write lock */
SYSCALL_DEFINE0(rw_write_unlock)
{
   up_write(&rw_lock);
   printk(KERN_INFO "Writer released lock\n");
   return 0;
}
```

v. After this, we created a Makefile using the command “nano kernel/rw_sync/Makefile” with the following code:

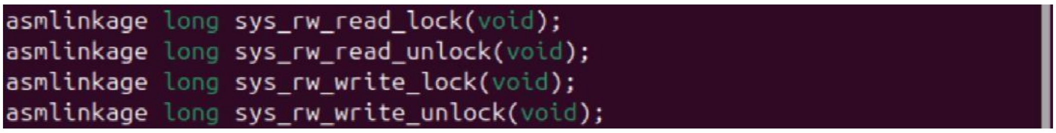
```
obj-y := rw_syscalls.o
obj-y += kernel/rw_sync/
```

vi. We assigned the system calls numbers to the system calls we created in the **master lookup table.** We opened the syscalls_64.tbl using the command “nano arch/x86/entry/syscalls/syscall_64.tbl”.



```
471     common  rw_read_lock            sys_rw_read_lock
472     common  rw_read_unlock          sys_rw_read_unlock
473     common  rw_write_lock           sys_rw_write_lock
474     common  rw_write_unlock         sys_rw_write_unlock
```

vii. We added the declarations in the syscalls.h header file after assigning the numbers to the system calls. First, we opened the syscall.h file using the command “nano include/linux/syscalls.h” and then added the declarations.



```
asmlinkage long sys_rw_read_lock(void);
asmlinkage long sys_rw_read_unlock(void);
asmlinkage long sys_rw_write_lock(void);
asmlinkage long sys_rw_write_unlock(void);
```

viii. After adding the declarations, we recompiled the kernel, updated the boot-loader and rebooted the system using the following commands.

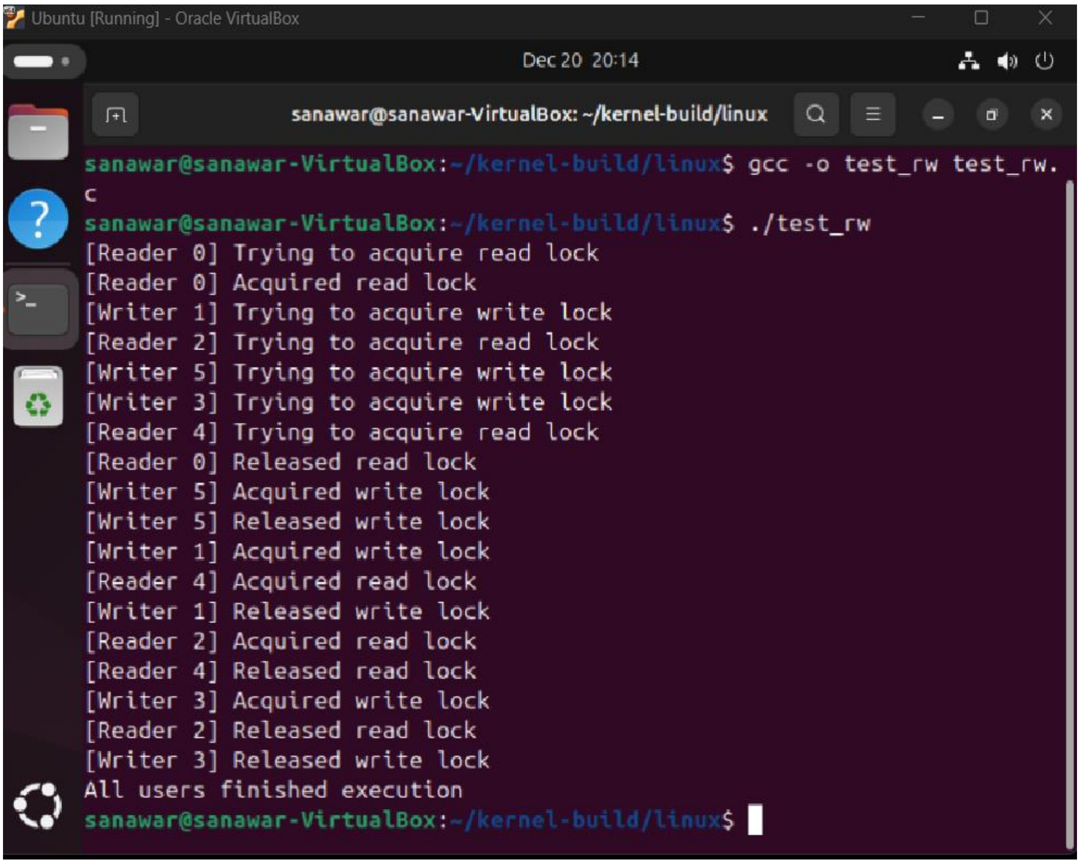
```
make -j4
sudo make modules_install
sudo make install
sudo update-grub
sudo reboot
uname -r
```

ix. In the end, we tested our custom system calls using the following C program:

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

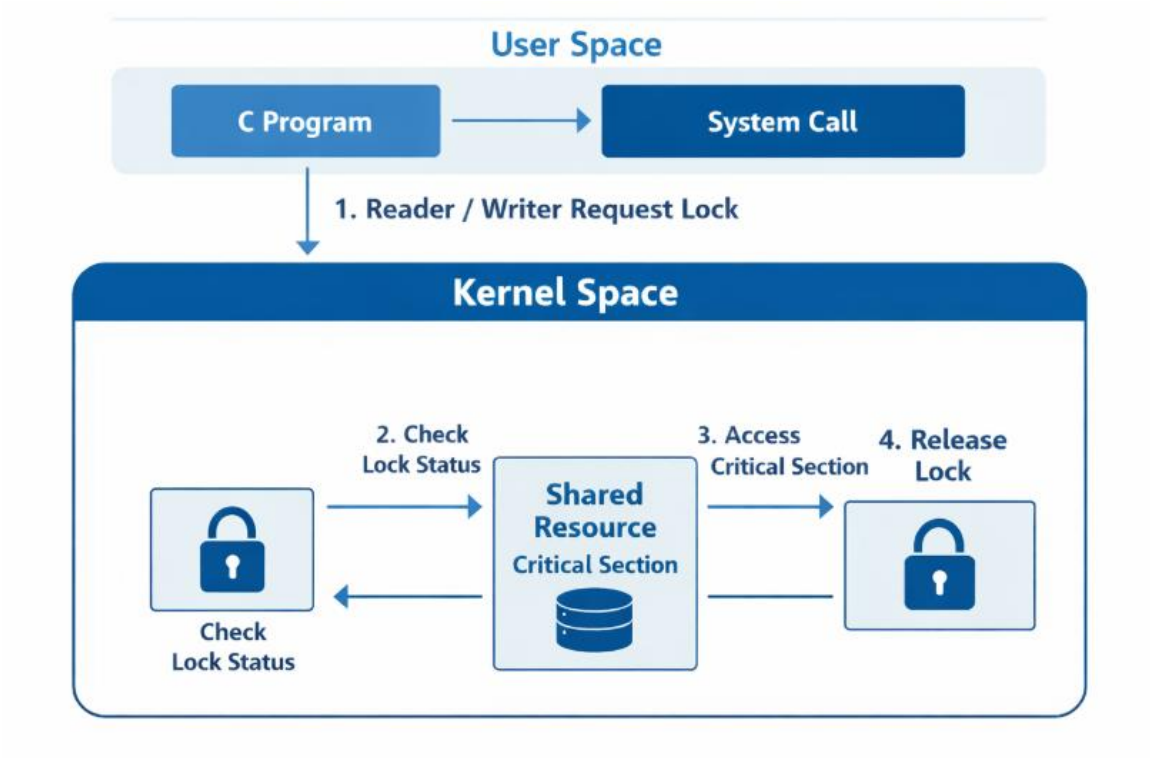
```
#define SYS_RW_READ_LOCK 471
#define SYS_RW_READ_UNLOCK 472
#define SYS_RW_WRITE_LOCK 473
#define SYS_RW_WRITE_UNLOCK 474

#define NUM_USERS 6 // Total processes (readers + writers)

void reader_process(int id)
{
    long ret;

    printf("[Reader %d] Trying to acquire read lock\n", id);
    ret = syscall(SYS_RW_READ_LOCK);
    if (ret == -1)
    {
        printf("[Reader %d] Error: %s\n", id, strerror(errno));
        exit(1);
    }

    printf("[Reader %d] Acquired read lock\n", id);
    sleep(2);

    ret = syscall(SYS_RW_READ_UNLOCK);
    if (ret == -1)
    {
        printf("[Reader %d] Error unlocking: %s\n", id, strerror(errno));
        exit(1);
    }

    printf("[Reader %d] Released read lock\n", id);
    exit(0);
}

void writer_process(int id)
{
    long ret;

    printf("[Writer %d] Trying to acquire write lock\n", id);
    ret = syscall(SYS_RW_WRITE_LOCK);
    if (ret == -1)
    {
        printf("[Writer %d] Error: %s\n", id, strerror(errno));
        exit(1);
    }

    printf("[Writer %d] Acquired write lock\n", id);
    sleep(3);

    ret = syscall(SYS_RW_WRITE_UNLOCK);
    if (ret == -1)
    {
        printf("[Writer %d] Error unlocking: %s\n", id, strerror(errno));
        exit(1);
    }

    printf("[Writer %d] Released write lock\n", id);
    exit(0);
}

int main()
{
    pid_t pid;

    for (int i = 0; i < NUM_USERS; i++)
    {
        pid = fork();

        if (pid < 0)
        {
            perror("fork failed");
            exit(1);
        }

        if (pid == 0)
        {
            // Child process
            if (i % 2 == 0)
                reader_process(i);
            else
                writer_process(i);
        }
    }

    // Parent waits for all children
    for (int i = 0; i < NUM_USERS; i++)
        wait(NULL);
    printf("All users finished execution\n");
    return 0;
}
```

Output:


Ubuntu [Running] - Oracle VirtualBox
Dec 20 20:14
sanawar@sanawar-VirtualBox: ~/kernel-build/linux

```
sanawar@sanawar-VirtualBox:~/kernel-build/linux$ gcc -o test_rw test_rw.
c
sanawar@sanawar-VirtualBox:~/kernel-build/linux$ ./test_rw
[Reader 0] Trying to acquire read lock
[Reader 0] Acquired read lock
[Writer 1] Trying to acquire write lock
[Reader 2] Trying to acquire read lock
[Writer 5] Trying to acquire write lock
[Writer 3] Trying to acquire write lock
[Reader 4] Trying to acquire read lock
[Reader 0] Released read lock
[Writer 5] Acquired write lock
[Writer 5] Released write lock
[Writer 1] Acquired write lock
[Reader 4] Acquired read lock
[Writer 1] Released write lock
[Reader 2] Acquired read lock
[Reader 4] Released read lock
[Writer 3] Acquired write lock
[Reader 2] Released read lock
[Writer 3] Released write lock
All users finished execution
sanawar@sanawar-VirtualBox:~/kernel-build/linux$ 
```

Diagram Flow:


User Space
C Program
System Call
1. Reader / Writer Request Lock
Kernel Space
2. Check Lock Status
Shared Resource
Critical Section
3. Access Critical Section
4. Release Lock
Check Lock Status