



OS Project - Reader Writer Problem

Instructor: Safia Baloch

Submitted by:

1. Fatima Liaquat (2023202)
2. Muhammad Sanawar (2023331)

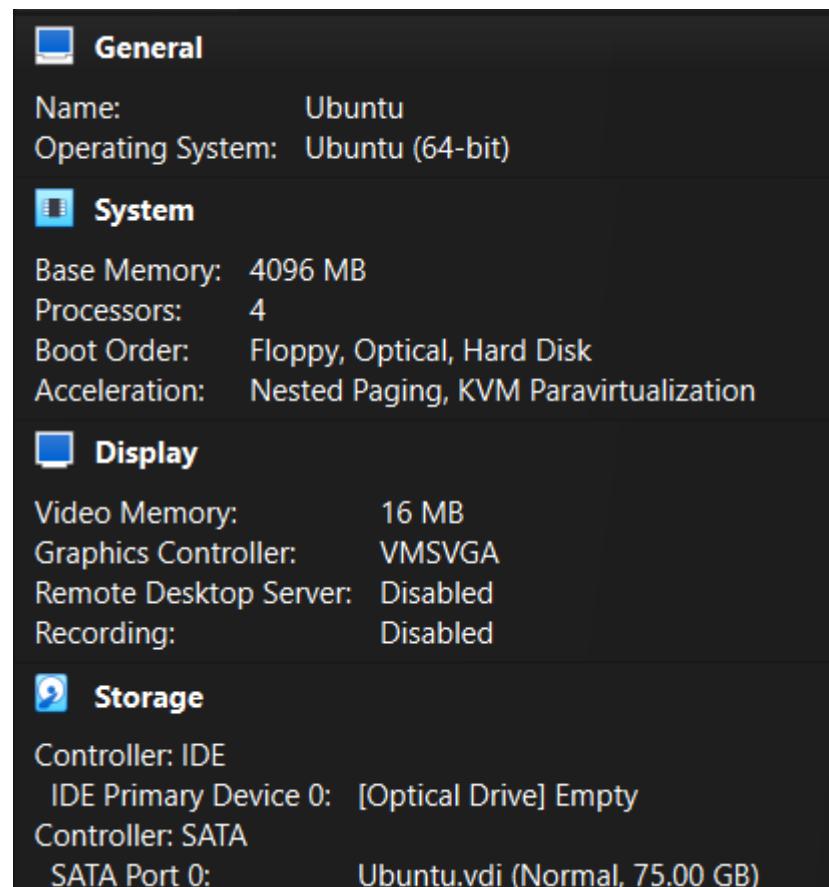
Introduction:

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



Implementation:

After the installation of Ubuntu, we implemented 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.

```
asm linkage long sys_rw_read_lock(void);  
asm linkage long sys_rw_read_unlock(void);  
asm linkage long sys_rw_write_lock(void);  
asm linkage 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 <errno.h>  
#include <string.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

int main() {
    long ret;

    // Reader acquires lock
    printf("Reader trying to acquire lock\n");
    ret = syscall(SYS_RW_READ_LOCK);
    if (ret == -1) {
        printf("Error acquiring read lock: %s\n", strerror(errno));
        return 1;
    }
    printf("Reader acquired lock\n");

    sleep(2); // Simulate reading

    // Reader releases lock
    ret = syscall(SYS_RW_READ_UNLOCK);
    if (ret == -1) {
        printf("Error releasing read lock: %s\n", strerror(errno));
        return 1;
    }
    printf("Reader released lock\n");

    // Writer acquires lock
    printf("Writer trying to acquire lock\n");
    ret = syscall(SYS_RW_WRITE_LOCK);
    if (ret == -1) {
        printf("Error acquiring write lock: %s\n", strerror(errno));
        return 1;
    }
    printf("Writer acquired lock\n");

    sleep(2); // Simulate writing

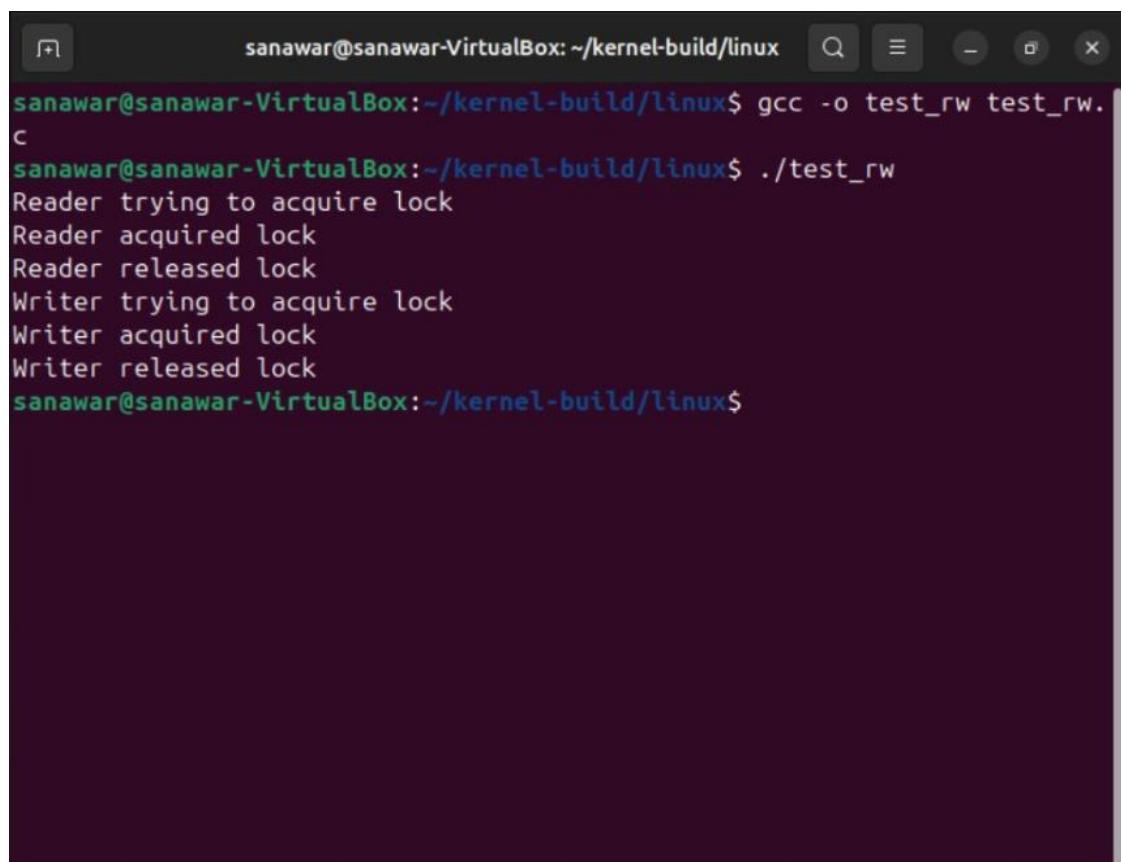
    // Writer releases lock
    ret = syscall(SYS_RW_WRITE_UNLOCK);
    if (ret == -1) {
        printf("Error releasing write lock: %s\n", strerror(errno));
    }
}

```

```
    return 1;
}
printf("Writer released lock\n");

    return 0;
}
```

Output:



The screenshot shows a terminal window titled "sanawar@sanawar-VirtualBox: ~/kernel-build/linux". The user runs the command "gcc -o test_rw test_rw.c" to compile the C program. After compilation, they run "./test_rw". The output shows the sequence of lock acquisitions and releases by two threads: "Reader" and "Writer".

```
sanawar@sanawar-VirtualBox:~/kernel-build/linux$ gcc -o test_rw test_rw.c
sanawar@sanawar-VirtualBox:~/kernel-build/linux$ ./test_rw
Reader trying to acquire lock
Reader acquired lock
Reader released lock
Writer trying to acquire lock
Writer acquired lock
Writer released lock
sanawar@sanawar-VirtualBox:~/kernel-build/linux$
```