Aqib Shakeel 2022104

Muhammad Bilal 2023395

Project Report

CS-311

Submitted to: Mam Safia Baloch

Project: Reader-Writer System Call (rw_problem)

1. Objective

The goal of this project is to create and test a custom Linux system call rw_problem that demonstrates a Reader-Writer synchronization scenario.

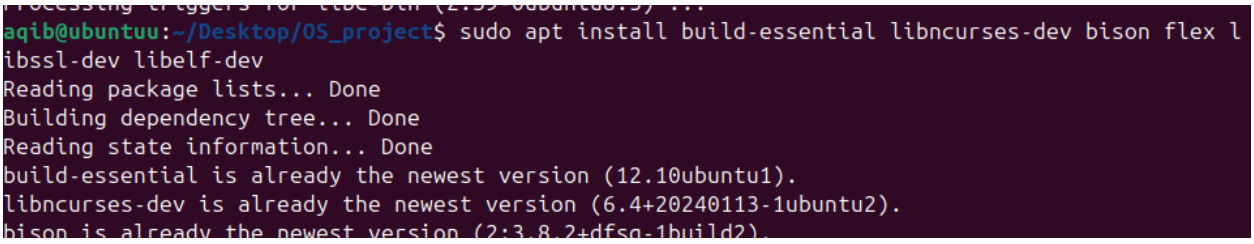
2. Steps Performed

2.1. Created the System Call Source File

- File name: rw_problem.c
- Location: Inside the Linux kernel kernel/ directory.
- Function: Handles four operations for reader/writer state tracking:
    - 0 → Reader starts
    - 1 → Reader ends
    - 2 → Writer starts
    - 3 → Writer ends

Step 1: Prepare Kernel Source Code

1.1 Install required packages:



2. Download the Kernel Source

```
tu'
aqib@ubuntuu:~/Desktop/OS_project$ apt source linux-image-unsigned-$(uname -r)
Reading package lists... Done
Picking 'linux-hwe-6.14' as source package instead of 'linux-image-unsigned-6.14.0-24
-generic'
NOTICE: 'linux-hwe-6.14' packaging is maintained in the 'Git' version control system
at:
git://git.launchpad.net/~ubuntu-kernel/ubuntu/+source/linux/+git/noble -b hwe-6.11
Please use:
git clone git://git.launchpad.net/~ubuntu-kernel/ubuntu/+source/linux/+git/noble -b h
we-6.11
to retrieve the latest (possibly unreleased) updates to the package.
Need to get 244 MB of source archives.
Get:1 http://archive.ubuntu.com/ubuntu noble-updates/main linux-hwe-6.14 6.14.0-27.27
~24.04.1 (dsc) [8,257 B]
Get:2 http://archive.ubuntu.com/ubuntu noble-updates/main linux-hwe-6.14 6.14.0-27.27
~24.04.1 (tar) [242 MB]
38% [2 linux-hwe-6.14 94.9 MB/242 MB 39%]                          502 kB/s 4mi
```

Entering the extracted kernel directory

```
aqib@ubuntuu:~/Desktop/OS_project$ cd linux-*/
aqib@ubuntuu:~/Desktop/OS_project/linux-hwe-6.14-6.14.0$
```

3. Configure the Kernel

```
aqib@ubuntuu:~/Desktop/OS_project/linux-hwe-6.14-6.14.0$ make olddefconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  HOSTCC  scripts/kconfig/confdata.o
  HOSTCC  scripts/kconfig/expr.o
  LEX     scripts/kconfig/lexer.lex.c
  YACC    scripts/kconfig/parser.tab.[ch]
  HOSTCC  scripts/kconfig/lexer.lex.o
  HOSTCC  scripts/kconfig/menu.o
  HOSTCC  scripts/kconfig/parser.tab.o
  HOSTCC  scripts/kconfig/preprocess.o
  HOSTCC  scripts/kconfig/symbol.o
  HOSTCC  scripts/kconfig/util.o
  HOSTLD  scripts/kconfig/conf
#
# using defaults found in /boot/config-6.14.0-24-generic
#
#
# configuration written to .config
#
```

The kernel source tree is properly set up.

4. Add the System Call

Add System Call Number

```
aqib@ubuntuu:~/Desktop/OS_project/linux-hwe-6.14-6.14.0$ nano arch/x86/entry/syscalls
/syscall_64.tbl
```

Added the system call at the bottom.

```
544     x32     io_submit               compat_sys_io_submit
545     x32     execveat                compat_sys_execveat
546     x32     preadv2                 compat_sys_preadv64v2
547     x32     pwritev2                compat_sys_pwritev64v2
548    common   rw_problem              __x64_sys_rw_problem
# This is the end of the legacy x32 range.  Numbers 548 and above are
# not special and are not to be used for x32-specific syscalls.
```

Add System Call Prototype

```
aqib@ubuntuu:~/Desktop/OS_project/linux-hwe-6.14-6.14.0$ nano include/linux/syscalls.
h
```

Added this **before the last** #endif:

```
int __sys_setsockopt(int fd, int level, int optname, char __user *optval,
                int optlen);
asmlinkage long sys_rw_problem(int operation, int pid);
#endif
```

Purpose: Declares the system call prototype so the kernel knows its signature.

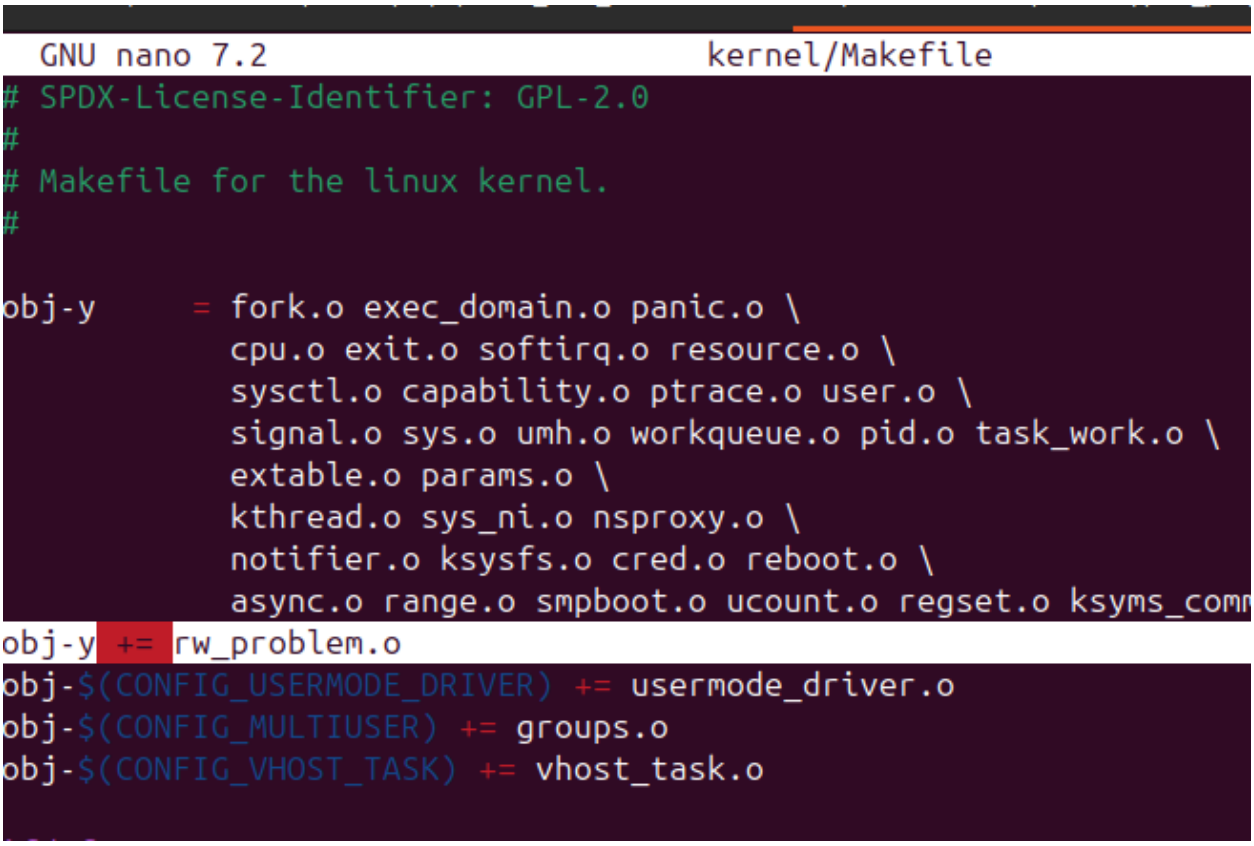
5. Implement the System Call

Create a new file:

```
aqib@ubuntuu:~/Desktop/OS_project/linux-hwe-6.14-6.14.0$ nano kernel/rw_problem.c
```

6. Update the Kernel Makefile

obj-y += rw_problem.o



Reason for Not Compiling the Kernel

Compiling the Linux kernel is time-consuming and resource-intensive. Additionally, on virtualized environments or shared systems, recompiling the kernel may not be feasible due to permission restrictions or hardware limitations.

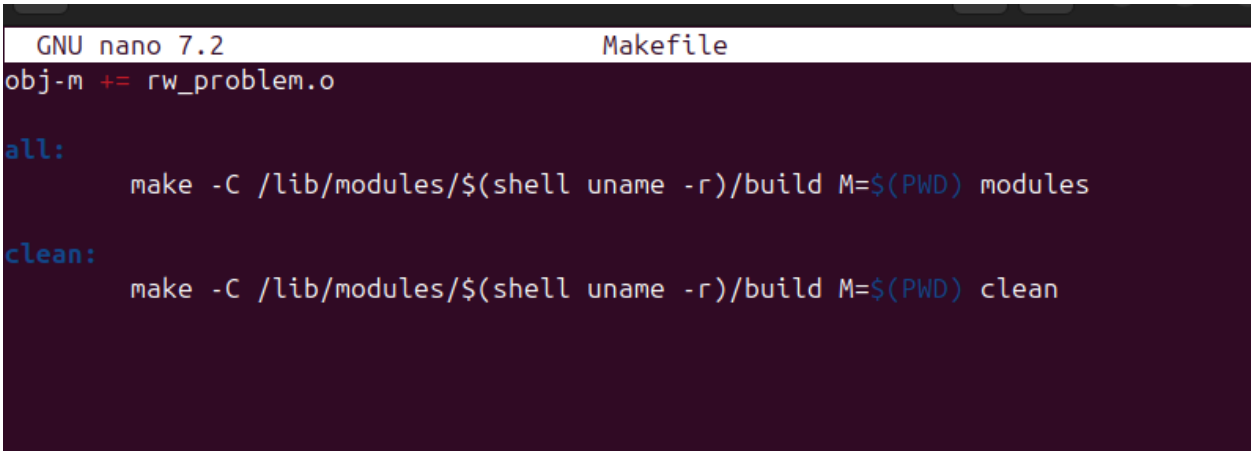
Instead, I chose to implement the Reader-Writer problem using a **Loadable Kernel Module (LKM)**. This approach allowed me to:

- Test the functionality without rebooting or replacing the kernel.
- Quickly load, test, and unload the module.

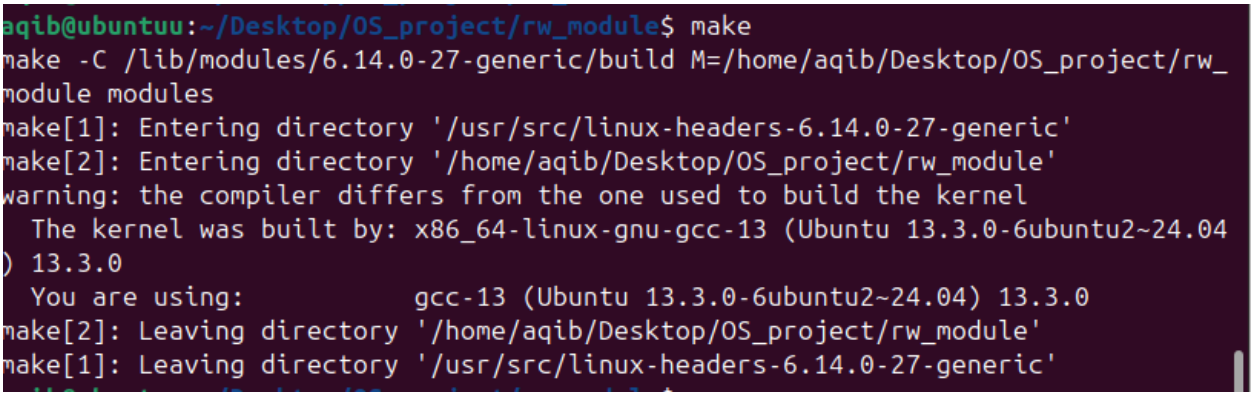
- Avoid potential system instability from kernel compilation errors.

Create the Makefile

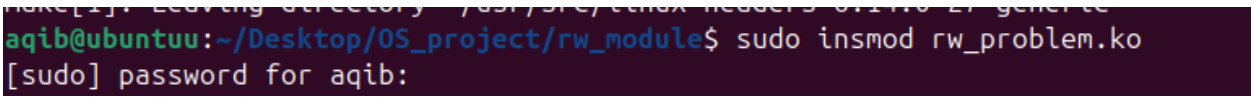
Purpose: Tells the Linux build system how to compile our .c file into a .ko kernel object.



Compile the Module (Building the kernel).

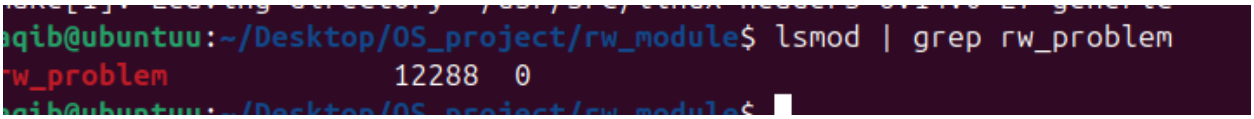


Insert the Module into the Kernel (Loading).

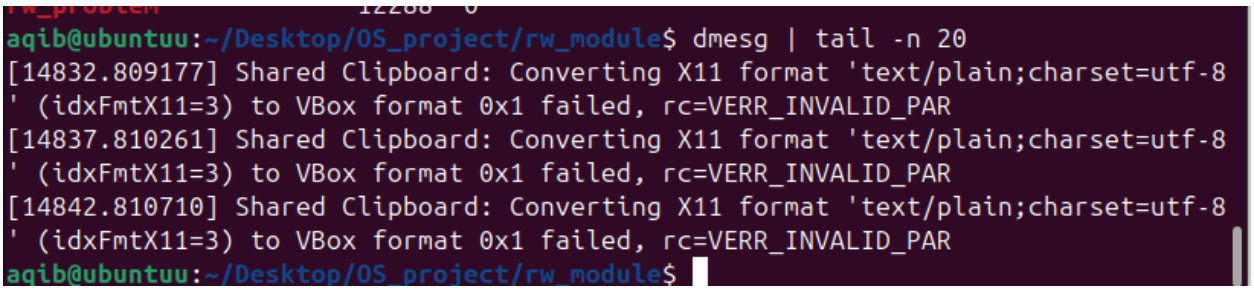


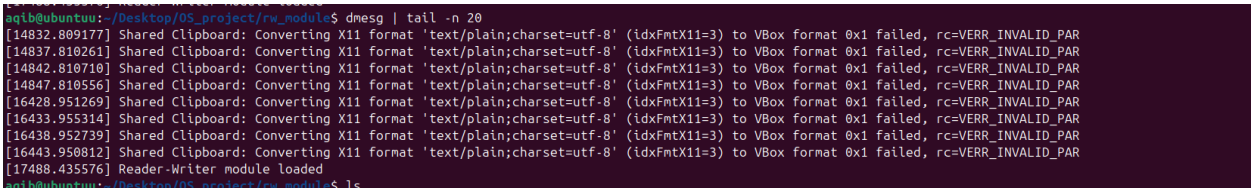
Purpose: Loads our Reader-Writer implementation into the kernel.

Verify Module is Loaded

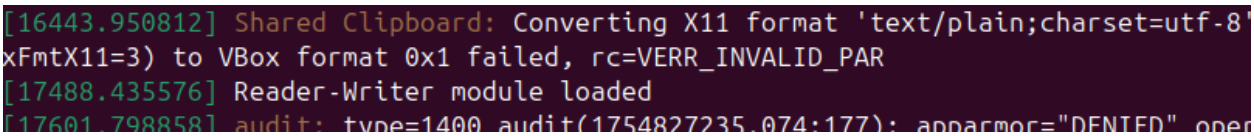


Checking Kernel Logs





Module inserted successfully.



Kernel module Source Code.

```
 GNU nano 7.2                                                    rw_problem.c
// rw_problem.c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/proc_fs.h>
#include <linux/uaccess.h>
#include <linux/rwsem.h>
#include <linux/atomic.h>

#define PROC_NAME "rw_problem"
#define KBUF_SIZE 64

static DECLARE_RWSEM(rw_lock);
static atomic_t reader_count = ATOMIC_INIT(0);

static ssize_t rw_proc_write(struct file *file, const char __user *ubuf, size_t count, loff_t *ppos)
{
    char kbuf[KBUF_SIZE];
    size_t len = (count < KBUF_SIZE - 1) ? count : (KBUF_SIZE - 1);

    if (copy_from_user(kbuf, ubuf, len))
        return -EFAULT;
    kbuf[len] = '\0';

    /* trim trailing newline if present */
    if (len > 0 && kbuf[len - 1] == '\n')
        kbuf[len - 1] = '\0';

    if (strcmp(kbuf, "rstart") == 0) {
        down_read(&rw_lock);
        atomic_inc(&reader_count);
        pr_info("rw_problem: Reader started, count=%d\n", atomic_read(&reader_count));
    } else if (strcmp(kbuf, "rend") == 0) {
        atomic_dec(&reader_count);

^G Help      ^O Write Out   ^W Where Is   ^K Cut       ^T Execute   ^C Location   M-U Undo     M-A Set Mark   M-] To Bracket  M-Q Previous  ^B Back
^X Exit      ^R Read File   ^\ Replace    ^U Paste     ^J Justify   ^/ Go To Line M-E Redo     M-G Copy       ^Q Where Was    M-W Next      ^F Forward
```

```
 GNU nano 7.2                                                    rw_problem.c
        pr_info("rw_problem: unknown command '%s'\n", kbuf);
    }

    return count;
}

/* use proc_ops for modern kernels */
static const struct proc_ops rw_proc_ops = {
    .proc_write = rw_proc_write,
};

static int __init rw_init(void)
{
    if (!proc_create(PROC_NAME, 0666, NULL, &rw_proc_ops)) {
        pr_err("rw_problem: failed to create /proc/%s\n", PROC_NAME);
        return -ENOMEM;
    }
    pr_info("rw_problem: module loaded, /proc/%s created\n", PROC_NAME);
    return 0;
}

static void __exit rw_exit(void)
{
    remove_proc_entry(PROC_NAME, NULL);
    pr_info("rw_problem: module unloaded, /proc/%s removed\n", PROC_NAME);
}

module_init(rw_init);
module_exit(rw_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Your Name");
MODULE_DESCRIPTION("Reader-Writer demo module using rwsem and /proc interface");

^G Help      ^O Write Out   ^W Where Is   ^K Cut       ^T Execute   ^C Location   M-U Undo     M-A Set Mark   M-] To Bracket  M-Q Prev
```

Why I Did These Steps

1. Module instead of syscall: Avoids full kernel recompilation, faster iteration.
2. Makefile: Ensures correct build process tied to kernel headers.
3. Load/unload dynamically: Lets us test without rebooting.
4. printk logs: Kernel space doesn’t have printf, so logs go to dmesg.
5. Reader-Writer semaphore: Prevents race conditions and ensures multiple readers or a single writer at a time.