OPERATING SYSTEM AND SYSTEM PROGRAMMING

SYSTEM CALL

SUBMITTED TO
Mr. Wendimu Baye

Abiy Shiferaw
1306078

TABLE OF CONTENTS

INTRODUCTION	2
WHAT/WHY/HOW DOES THIS SYSTEM CALL?	2
BRIEFLY DESCRIBE ABOUT THE LIST OF PARAMETERS AND FLAGS	2
LIST THE FLAGS, THEIR PURPOSE AND THE CODE OF IMPLEMENTATION	3
CODE IMPLEMENTATION	4
REFERENCE	10

INTRODUCTION

A system call (syscall) is the programmatic method by which a computer program requests a service from the kernel of the operating system on which it is running. This may comprise hardware-related services (such as accessing a hard disk drive or the device's camera), process creation and execution, and communication with integral kernel functions such as process scheduling. System calls serve as a vital link between a process and the operating system.

So, for this assignment I will try to demonstrate the kcmp system call.

1. WHAT / WHY / HOW, THIS SYSTEM CALL?

kcmp is a system call that is used to compare two processes to determine if they share a kernel resource . The kcmp () system call can be used to check whether the two processes identified by pid1 and pid2 share a kernel resource such as virtual memory, file descriptors, and so on. The syntax looks like the following.

```
int kcmp(pid_t pid1, pid_t pid2, int type, unsigned long idx1,
unsigned long idx2)
```

2. BRIEFLY DESCRIBE ABOUT THE LIST OF PARAMETERS AND FLAGS

```
int kcmp(pid_t pid1, pid_t pid2, int type, unsigned long idx1,
unsigned long idx2)
```

- ✓ `pid1` refers to the first process ID
- ✓ `pid2` refers to the second process ID
- ✓ `type` refers to type of resource to compare
- ✓ `idx1` refers to flag-specific resource index
- ✓ `idx2` refers to flag-specific resource index

3. LIST THE FLAGS, THEIR PURPOSE WITH CODE IMPLEMENTATION (GIVE EXAMPLE SOURCE CODE WITH OUTPUT)

The type flags include

KCMP_FILE

check if file descriptors specified in idx1 and idx2 are shared by both processes

KCMP_FILES

check if the two processes share the same set of open file descriptors (idx1 and idx2 are not used)

KCMP_FS

check if the two processes share the same filesystem information (for example, the filesystem root, mode creation mask, working directory, etc.)

KCMP_IO

check if processes share the same I/O context

KCMP_SIGHAND

check if processes share same table of signal dispositions

KCMP_SYSVSEM

check if processes share same semaphore undo operations

KCMP_VM

check if processes share same address space

KCMP_EPOLL_TFD

check if file descriptor referenced in idx1 of process pid1 is present in epoll referenced by idx2 of process pid2, where idx2 is a structure kcmp_epoll_slot describing target file

```
struct kcmp_epoll_slot {
    __u32 efd;
    __u32 tfd;
    __u64 toff;
};
```

CODE IMPLEMENTATION

The program below uses **kcmp**() to test whether pairs of file descriptors refer to the same open file description. The program tests different cases for the file descriptor pairs, as described in the program output.

Program source

```
#define _GNU_SOURCE
#include <sys/syscall.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <linux/kcmp.h>

#define errExit(msg)    do { perror(msg); exit(EXIT_FAILURE); \
                    } while (0)

static int
kcmp(pid_t pid1, pid_t pid2, int type,
     unsigned long idx1, unsigned long idx2)
{
   return syscall(SYS_kcmp, pid1, pid2, type, idx1, idx2);
}

static void
test_kcmp(char *msg, id_t pid1, pid_t pid2, int fd_a, int fd_b)
{
   printf("\t%s\n", msg);
   printf("\t\tkcmp(%ld, %ld, KCMP_FILE, %d, %d) ==> %s\n",
         (long) pid1, (long) pid2, fd_a, fd_b,
         (kcmp(pid1, pid2, KCMP_FILE, fd_a, fd_b) == 0) ?
                  "same" : "different");
}

int
main (int argc, char *argv[])
{
   int fd1, fd2, fd3;
   char pathname[] = "/tmp/kcmp.test";
```

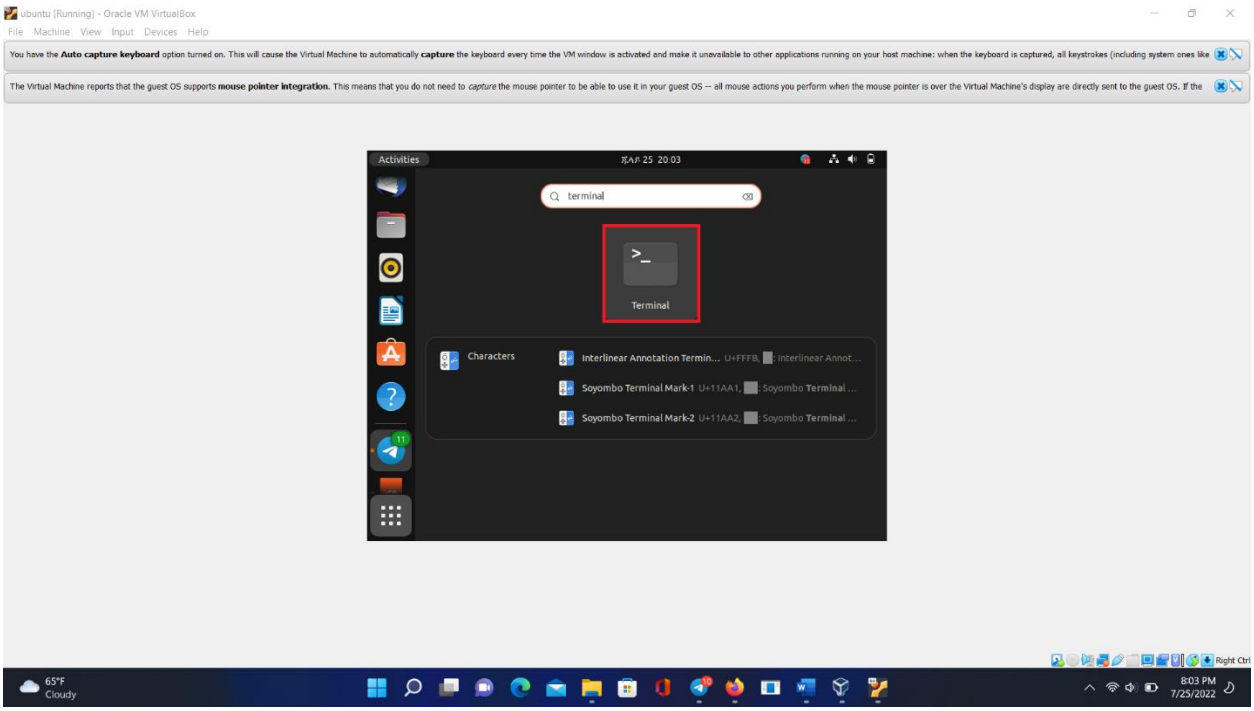
```
    fd1 = open(pathname, O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
    if (fd1 == -1)
        errExit("open");

    printf("Parent PID is %ld\n", (long) getpid());
    printf("Parent opened file on FD %d\n\n", fd1);

    switch (fork()) {
    case -1:
        errExit("fork");

    case 0:
        printf("PID of child of fork() is %ld\n", (long) getpid());

        test_kcmp("Compare duplicate FDs from different processes:",
                getpid(), getppid(), fd1, fd1);

        fd2 = open(pathname, O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
        if (fd2 == -1)
            errExit("open");
        printf("Child opened file on FD %d\n", fd2);

        test_kcmp("Compare FDs from distinct open()s in same process:",
                getpid(), getpid(), fd1, fd2);

        fd3 = dup(fd1);
        if (fd3 == -1)
            errExit("dup");
        printf("Child duplicated FD %d to create FD %d\n", fd1, fd3);

        test_kcmp("Compare duplicated FDs in same process:",
                getpid(), getpid(), fd1, fd3);
        break;

    default:
        wait(NULL);
    }

    exit(EXIT_SUCCESS);
}
```

An example run of the program is as follows:

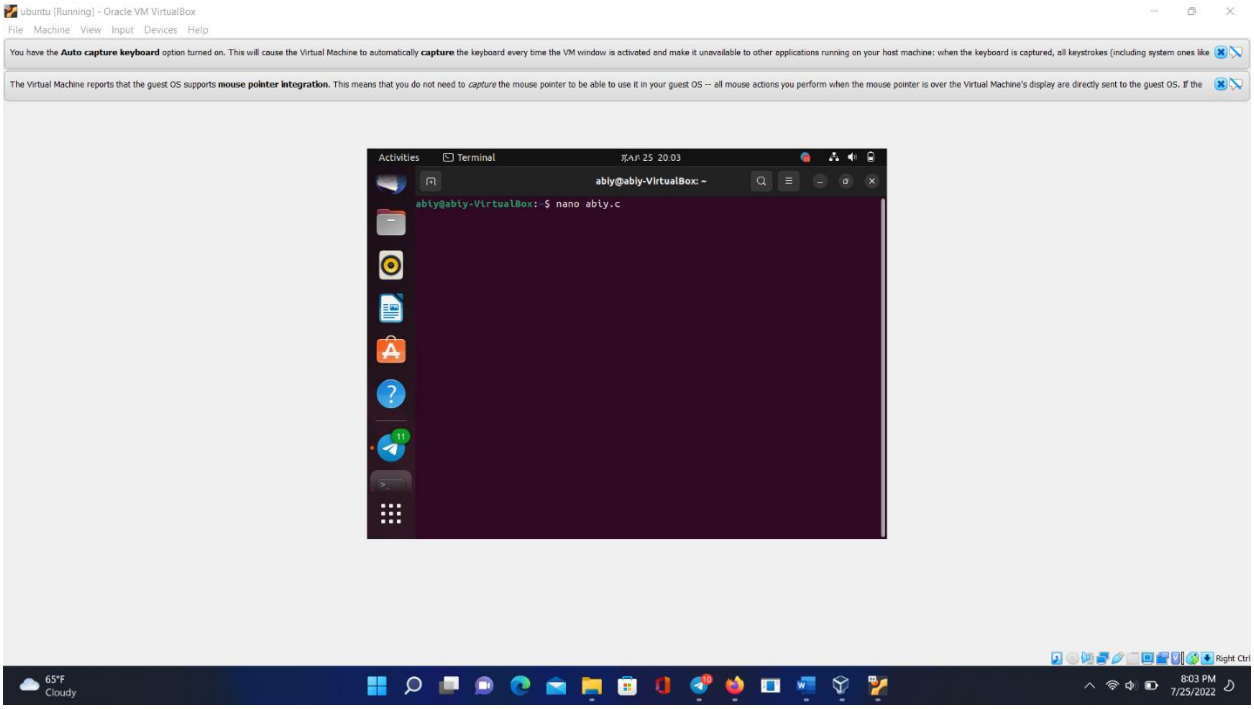
```
$ ./a.out
Parent PID is 1144
Parent opened file on FD 3

PID of child of fork() is 1145
    Compare duplicate FDs from different processes:
        kcmp(1145, 1144, KCMP_FILE, 3, 3) ==> same
Child opened file on FD 4
    Compare FDs from distinct open()s in same process:
        kcmp(1145, 1145, KCMP_FILE, 3, 4) ==> different
Child duplicated FD 3 to create FD 5
    Compare duplicated FDs in same process:
        kcmp(1145, 1145, KCMP_FILE, 3, 5) ==> same
```

We can show this code using virtual box.

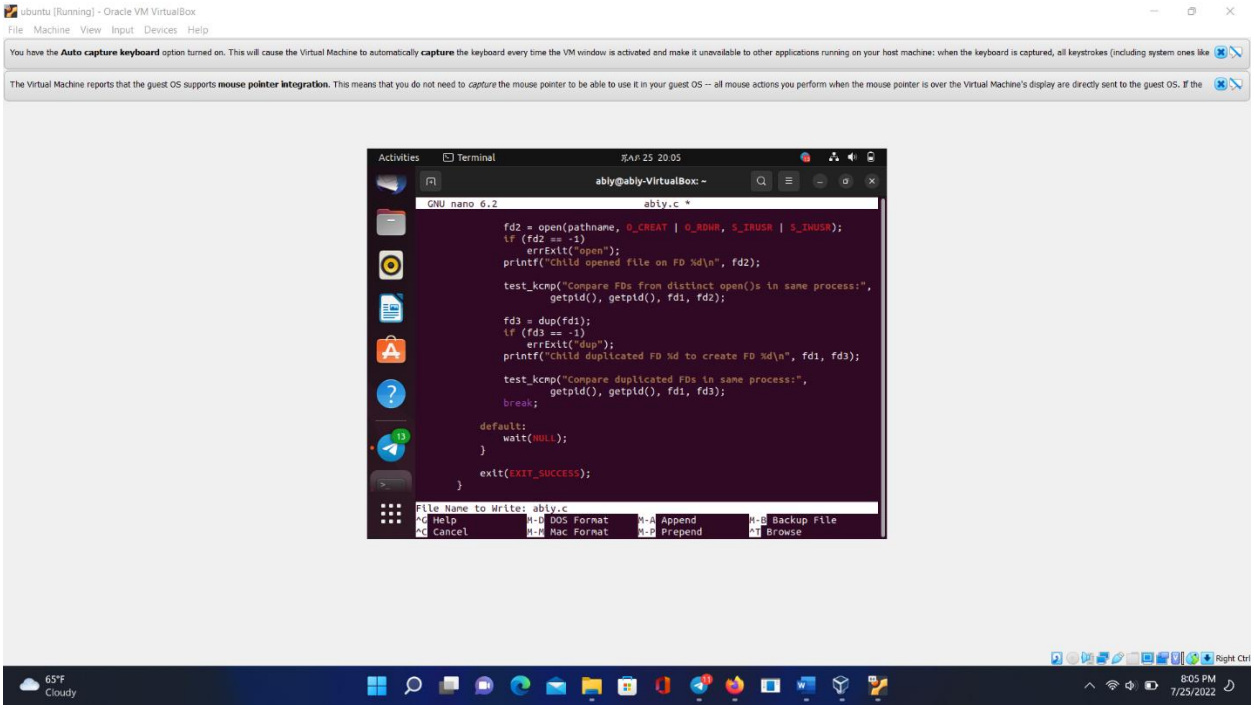
1. Open terminal



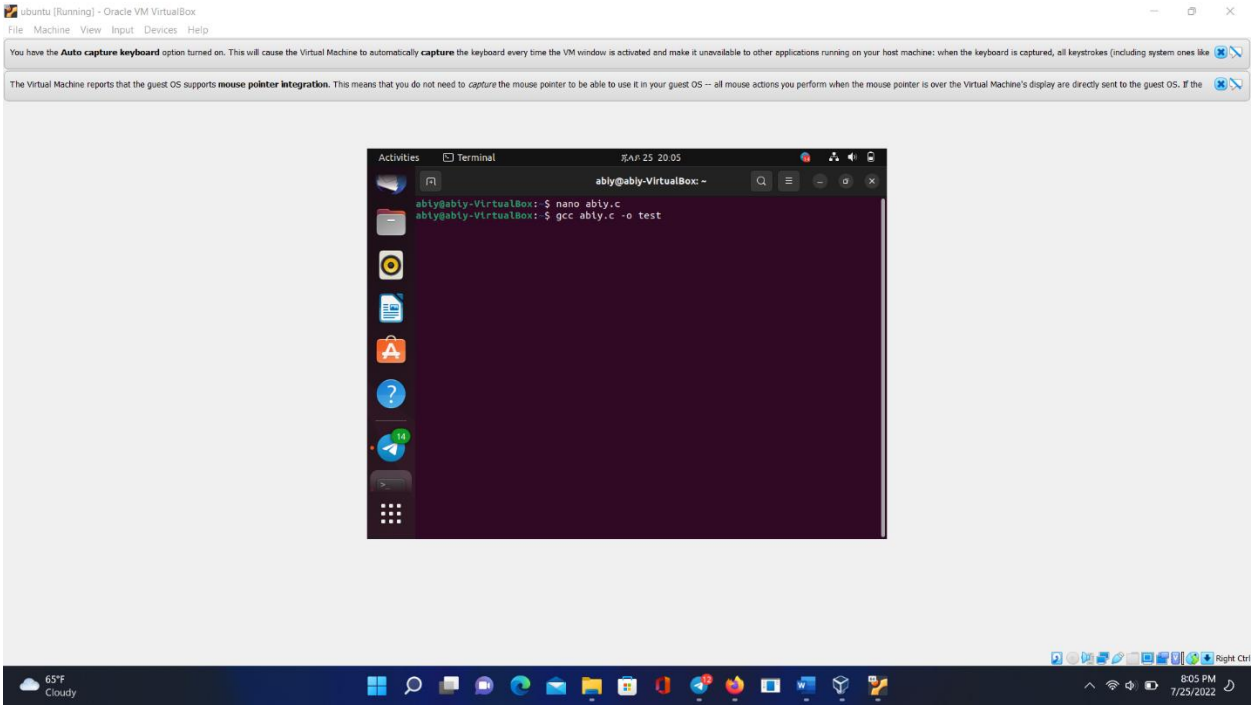
2. Open a C file



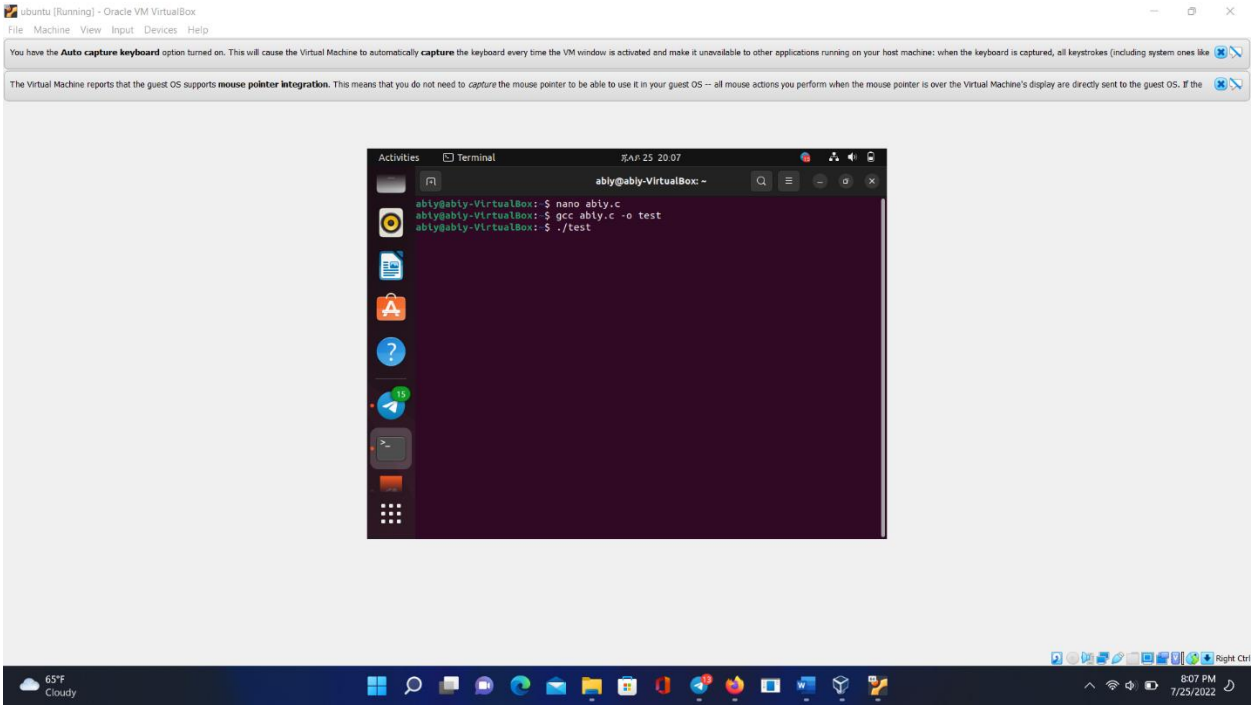
3. Write the code and press Ctrl + O to write out and then press Enter.



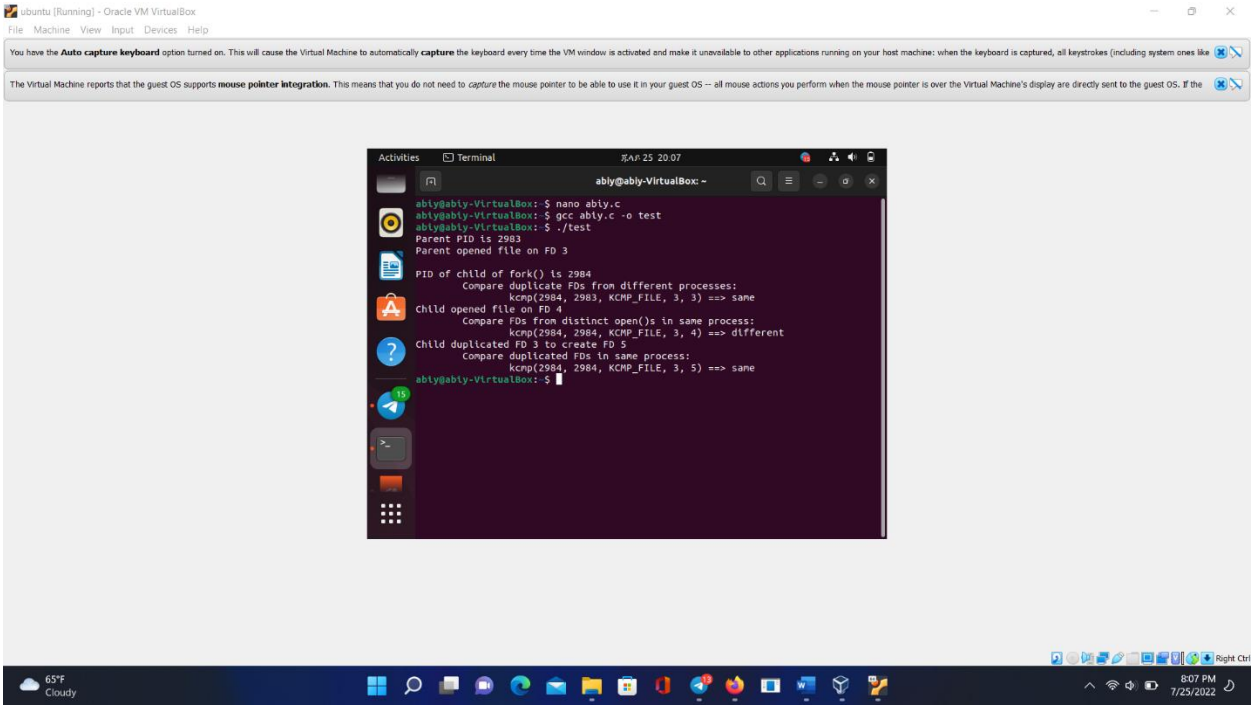
4. Then press Ctrl + X and compile the file.



5. Since this code has no errors, it will be compiled correctly. Therefore, execute the file to display the output of the code.



6. The output should look like this.



REFERENCES

https://manpages.ubuntu.com/manpages/bionic/man2/kcmp.2.html

https://linuxhint.com/list_of_linux_syscalls/#kcmp

https://en.wikipedia.org/wiki/System_call