Project 2 Report

Alyssa Sallean, Dillon Huff, Nicholas Pape

The Code:

The purpose of this project was to learn how to implement system calls in the Linux kernel. We stepped through a tutorial of how to implement the system call hello_utcs and then were left to our own devices to implement get_system_time.

The first step in implementing the system calls hello_utcs and get_system_time is to add it to the syscall table. This helps interface the user space with kernel space function calls. We added the following lines to the file syscall_32.tbl:

```
351     i386    hello_utcs              sys_hello_utcs
352     i386    get_system_time         sys_get_system_time
```

The next step was to add the function declaration headers in the syscall.h file. This declares the system calls defined in the syscall table. We added the following lines of code to syscall.h:

```
asmlinkage long sys_hello_utcs(const char __user *msg, int len);
asmlinkage long sys_get_system_time(struct timeval __user *tv);
```

Next, we had to actually define the functions. We will not bother repeating the code that was already provided for the implementation for hello_utcs, but the following code is out implementation of get_system_time in my_syscall.c:

```
asmlinkage long sys_get_system_time(struct timeval __user *tv){
        struct timeval dummy;
        do_gettimeofday(&dummy);

        if(copy_to_user(tv, &dummy, sizeof(struct timeval))){
                return -EFAULT;
        }

        return 0;
}
```

We create a temporary struct timeval object to call on do_gettimeofday. Broken down, do_gettimeofday basically just reads a sequence of bytes from a location in memory that keeps track of the system time. We have to use a temporary struct timeval because the variable passed in as a parameter is in user space and could cause virtual memory errors. We then copy the dummy variable into the tv pointer using copy_to_user. If there is an error, copy_to_user will be greater than zero. We did not do any error checking on do_gettimeofday because it is a void function that is assumed to succeed.

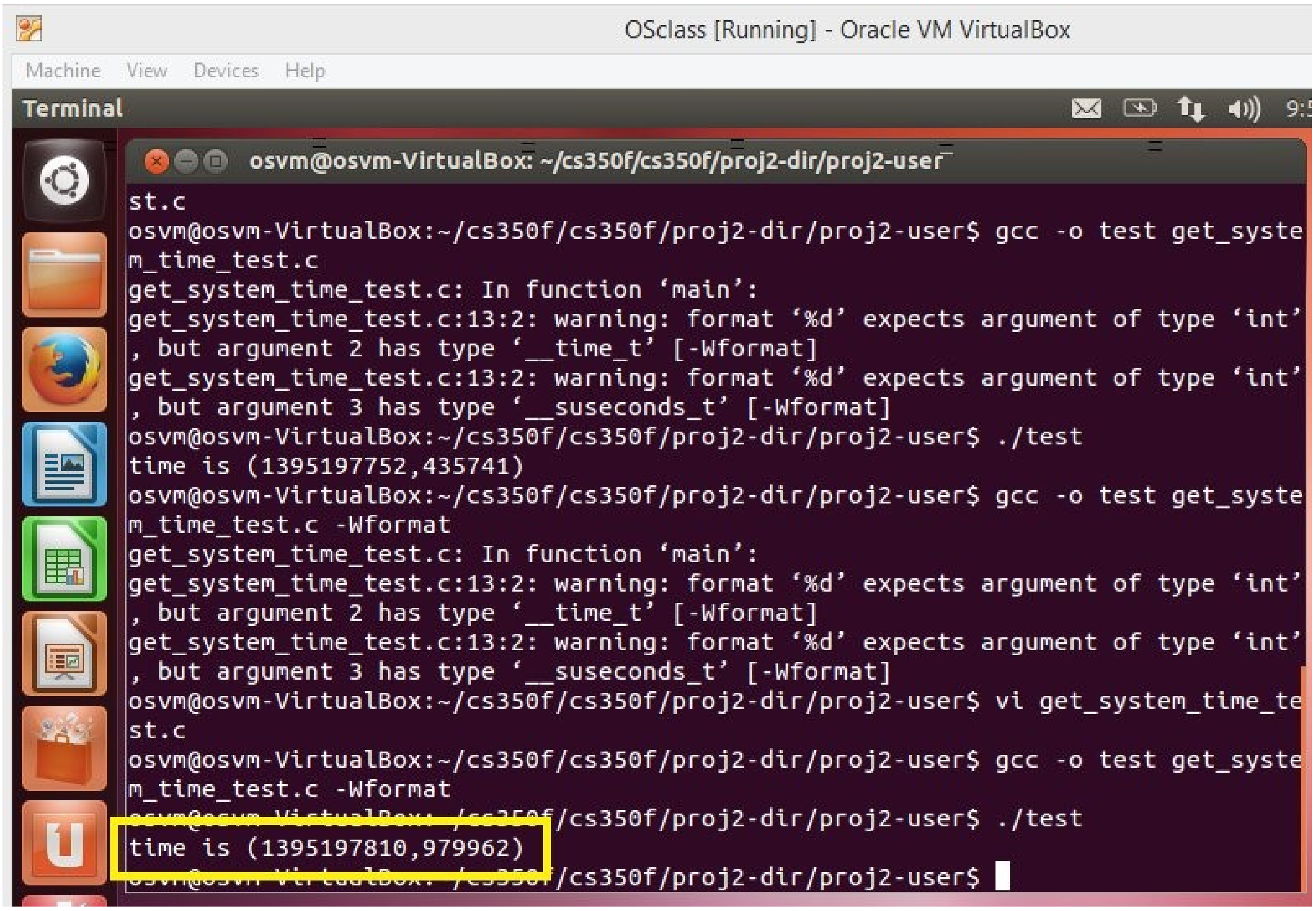
Testing:

In order to test our system call, we wrote a simple c function that calls the system call, and then prints out the values in the timeval struct upon success. We wrote the following code in get_system_time_test.c:

```
int main(){
        struct timeval t;
        int n;
        if((n = syscall(352,&t)) < 0)
                printf("Calling get_system_time failed with error 
code %d\n", n);

        printf("time is (%d,%d)\n", (int)t.tv_sec, (int)t.tv_usec);
        return 0;
}
```

Below is a screenshot of the output when the sample program is run.


OSclass [Running] - Oracle VM VirtualBox
Machine
View
Devices
Help
Terminal
osvm@osvm-VirtualBox: ~/cs350f/cs350f/proj2-dir/proj2-user

```
st.c
osvm@osvm-VirtualBox:~/cs350f/cs350f/proj2-dir/proj2-user$ gcc -o test get_syste
m_time_test.c
get_system_time_test.c: In function ‘main’:
get_system_time_test.c:13:2: warning: format ‘%d’ expects argument of type ‘int’
, but argument 2 has type ‘__time_t’ [-Wformat]
get_system_time_test.c:13:2: warning: format ‘%d’ expects argument of type ‘int’
, but argument 3 has type ‘__suseconds_t’ [-Wformat]
osvm@osvm-VirtualBox:~/cs350f/cs350f/proj2-dir/proj2-user$ ./test
time is (1395197752,435741)
osvm@osvm-VirtualBox:~/cs350f/cs350f/proj2-dir/proj2-user$ gcc -o test get_syste
m_time_test.c -Wformat
get_system_time_test.c: In function ‘main’:
get_system_time_test.c:13:2: warning: format ‘%d’ expects argument of type ‘int’
, but argument 2 has type ‘__time_t’ [-Wformat]
get_system_time_test.c:13:2: warning: format ‘%d’ expects argument of type ‘int’
, but argument 3 has type ‘__suseconds_t’ [-Wformat]
osvm@osvm-VirtualBox:~/cs350f/cs350f/proj2-dir/proj2-user$ vi get_system_time_te
st.c
osvm@osvm-VirtualBox:~/cs350f/cs350f/proj2-dir/proj2-user$ gcc -o test get_syste
m_time_test.c -Wformat
osvm@osvm-VirtualBox:~/cs350f/cs350f/proj2-dir/proj2-user$ ./test
time is (1395197810,979962)
osvm@osvm-VirtualBox:~/cs350f/cs350f/proj2-dir/proj2-user$ 
```