Advanced Operating Systems
Assignment 2
Adding System Calls
RollNo: 2021201077
Name: Husen kagdi

Aim: To add new custom-built system calls to a kernel version. The Kernel version should be Linux-4.19.200.

Adding Kernel Linux-4.19.200 to the system:

1) Download the required kernel version from this Link using ***wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.19.212.tar.xz.***
2) Extract it to the directory **/C/usr/src** using ***sudo tar -xvf linux-4.19.200.tar.xz -C/usr/src/.***
3) Navigate to the directory Linux-4.19.200.
4) Compile the kernel using the command the ***sudo make -jn*** command, where n is the number of processor cores.
5) Install the kernel using the ***sudo make modules_install install*** command.
6) Restart the system and boot into kernel Linux-4.19.200 using Advanced Ubuntu Reboot Option.

Adding a new system call to the system:

1) Navigate to the directory Linux-4.19.200.
2) Create a new folder with the name you like. Use command ***sudo mkdir folder_name.***
3) Use ***cd folder_name*** command to enter into the folder.
4) Create a new file using the command ***sudo gedit file_name.c.*** Write the code pertaining to that system call in the file.
5) Create a Makefile using the command ***sudo gedit Makefile.*** Add this line to the file. **Obj-y := file_name.o,** where file_name is the name of c file created in the above step.
6) Then navigate to the parent directory using the ***cd ..*** command.
7) Open the Makefile using the ***sudo gedit Makefile*** command.
8) Navigate using the command ***cd arch/x86/entry/syscalls.***
9) Execute the command ***sudo gedit syscall_64.tbl.***
10) Enter a record to the end of the file. The record should be

548	64	folder_name	name_of _system_call_function

11) Navigate to the Linux-4.19.200 directory.
12) Open the Makefile using the command ***sudo gedit Makefile.***
13) Search core-y+= in the file.
14) At the end of the line add / folder_name. Ensure space between / and folder_name. Save and Exit.
15) Navigate using the command ***cd /include/linux.***
16) Open the file syscalls.h using ***sudo gedit syscalls.h*** command. At the end of the file before endif# add the line ***asmlinkage long system_call_name(Parameters);***
17) Now navigate to the Linux-4.19.200 directory. To the config file do this. Make **CONFIG_SYSTEM_TRUSTED_KEYS="".** Earlier it was **CONFIG_SYSTEM_TRUSTEM_KEYS="debian/canonical-certs.pem".**
18) Compile the kernel using ***sudo make -jn,*** where n is the number of processor cores.
19) Install the kernel using the command ***sudo make modules_install install.***
20) Restart the system and boot into Linux-4.19.200.
21) Test your system call by creating a test.c file.

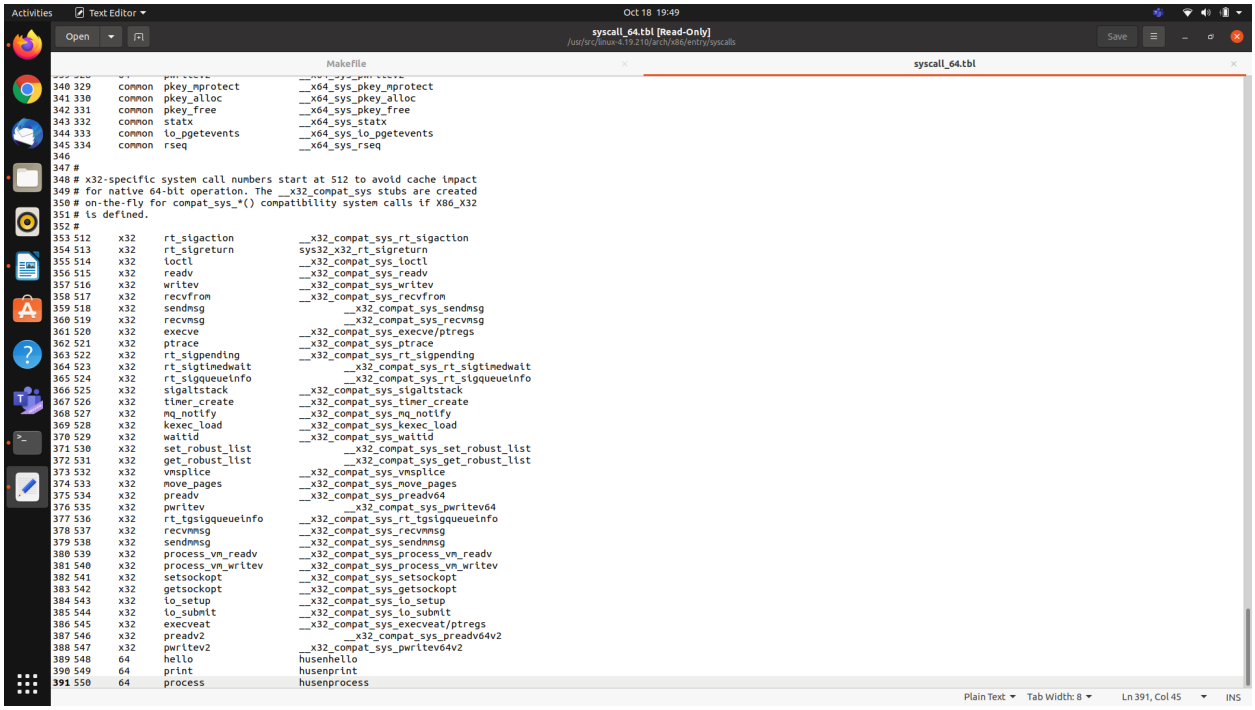
SYSCALLS TO CREATE:

Four system calls are to be created, given below:

1. Printing a welcome message to kernel logs.
2. Printing a given string to kernel logs by parameter passing.
3. Printing current process id and parent process id to the kernel log.
4. Recreating an existing syscall and implementing that.

SYSCALL_64.tbl file

The syscall_64.tbl file contains the record pertaining to the system calls.



MAKEFILE:

```
986 ifeq ($(KBUILD_EXTMOD),)
987 core-y         += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ hello/ print/ process
988
```

SYSCALL.H File

```
1294 }
1295 asmlinkage long husenhello(void);
1296 asmlinkage long husenprint(char *);
1297 asmlinkage long husenprocess(void);
1298 #endif
```

SYSTEM CALLS:

1) Printing Welcome message to the kernel

```c
#include <linux/kernel.h>

asmlinkage long husenhello(void)
{
      printk("Welcome Back Hussain\n");
      return 0;
}
```

It prints “Welcome Back Hussain” to the kernel.

2) Printing some string from userspace to kernel space by parameter passing.

```c
#include<linux/syscalls.h>
#include<linux/kernel.h>

SYSCALL_DEFINE0(husenprint, char * ,msg){
    char buffer[512];
    long copy=strncpy_from_user(buffer,msg,sizeof(buffer));
    if(copy<0 || copy==sizeof(buffer)){
       return -EFAULT;
    }
    printk(KERN_INFO "Husen passed the parameter %s in the systen
call \n",buffer);
    return 0;
}
```

SYSCALL DEFINE0 is a macro that creates the asmlinkage long husenprint(char __user * str) function, which has only one argument str. The '__user' specifies that the pointer is in the userspace. The API call 'Copy from user' allows you to copy data from a userspace reference to a kernel space buffer. If an error occurs while retrieving data, the EFAULT error flag will be returned. In syscall 64.tbl, a new syscall entry is introduced. It differs from the first entry in that it has a new sys_ prefix for alias and a new __x64 sys_ prefix for syscall name. This is due to the SYSCALL DEFINE# macro being used.

Printing current process Id and parent process id:

1. The following code was added to husenprocess.c in usr/src/linux-4.19.210/print , along with the following Makefile containing ‘obj-y := husenprocess.o`.

```c
#include<linux/syscalls.h>
#include<linux/kernel.h>
#include<linux/cred.h>
#include<linux/sched.h>

SYSCALL_DEFINE0(husenprocess,void){
    struct task_struct *parent_proc=current->parent;
    printk("(Parent->pid): %d\n",parent_proc->pid);
    printk("(Parent->tgetid): %d\n",parent_proc->tgid);
    printk("(Current->pid): %d\n", current->pid);
```

```
    printk("(Current->tgetid): %d\n", current->tgid);
    return 0;
}
```

The term 'current' refers to a task-type structure that maintains data about the current process (that includes the entirety of the Process Control Block). The current process id will be stored in the pid element. Given a task structure for the current process, the task ppid nr API method returns the pid of the parent process. In syscall 64.tbl, the appropriate entry is added.

The process ids for the parent and current processes are different.
Why?

The process id identifies the process that is responsible for both the syscall and the function that invoked it. That is, husenprocess.c and test.c are both part of the same process in this example. Because calling another function does not always result in the creation of a new process. The parent process id relates to the parent process of test.c, the code that is used to test the system call.

Recreating a system call
The following code was added to husengetuid.c in usr/src/linux-4.19.210/recreate, as well as the Makefile 'obj-y:= recreate.o'.

```
#include<linux/kernel.h>
#include<linux/syscalls.h>
#include<linux/cred.h>
#include<linux/sched.h>
#include<linux/uidgid.h>

SYSCALL_DEFINEO(husengetuid) {
    printk("User ID = %u\n", current_uid().val);
    return current_uid().val;
}
```

The k uid structure, which will hold the user id, is found in the uidgid.h header. This information is retrieved from this location via the syscall getuid(). The k uid structure is returned by current uid(), and val returns the uid unsigned long from it. In syscall 64.tbl, the appropriate entry is added.

Output of husenhello system call: