ASSIGNMENT 2: ADDING SYSTEM CALLS

Aim: To add new system calls to a given kernel version and observe the results

Requirement: All work should be done on linux kernel version 4.19.210.

ADDING KERNEL 4.19.210 TO SYSTEM

1. Created an azure student account using iiit mail id.
2. Created vm with **UBUNTU 16**, 4 core (linux-4.19.210).
3. Deployed vm on azure and used the public ip address to run the vm on the local machine.(using SSH client)
4. Created new folder in sudo mode and installed the pre-requisites required to run the kernel.
5. Go to above directory, created a new folder which contains all the required .c files, and then built and ran `sudo make` and `sudo make modules_install install`.
6. Made some configurations and rebooted the vm.

ADDING A SYSTEM CALL TO KERNEL

1. Created a new directory in /linux-4.19.210 for a system call. In this text, let it be ‘ass2’.
2. Created a .c files (eg q1.c) and add the necessary kernel space c code to it. We can use either asmlinkage long functionname(void) or the macro SYSCALL_DEFINE#(syscall_name) where # is the number of arguments expected and syscall_name is the name given to it.
3. Create a file named ‘Makefile’ and add the line ‘obj-y := q1.o’ to it, if the name of the c file is q1.c.
4. Navigate to / linux-4.19.210/include/linux and open syscalls.h
5. Add the definition of the function created in q1.c
6. Navigate to / linux-4.19.210/ and open Makefile
7. Add the name of the folder as ‘ass2/’ to core-y assignments as shown in subsequent headers.
8. Navigate to /linux-4.19.210/arch/x86/entry/syscalls and open syscall_64.tbl (or syscall_32.tbl if 32 bit system) and add appropriate entries to it. If function was made directly, we can make entry as shown for number 548, but if SYSCALL_DEFINE# macro was used, the following 3 lines should be used as references (x64(or32)__sys_ prefix should be added to the last column).
9. Finally, navigate to /linux-4.19.210 and run the following commands:

   `sudo make modules_install `
   `sudo make install `

10. Updated the GRUB menu and restarted the vm and, then enter the same kernel again
11. Test the system call via a c code.

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
3. Printing current and parent process id to kernel logs
4. Recreating an existing syscall and implementing that (in this case, getpid() has been recreated)

SYSCALL_64.TBL, MAKEFILE AND SYSCALLS.H

This list contains syscall entries and reference numbers for them. For each new syscall that we create, we need to add the corresponding entry. For the above 4 tasks, the last 4 entries in the given image were added (548, 549, 550, 551)

```
root@myVM: ~/linux-4.19.210/arch/x86/entry/syscalls
  GNU nano 2.5.3                                              File: syscall_64.tbl

534     x32     preadv                  __x32_compat_sys_preadv64
535     x32     pwritev                 __x32_compat_sys_pwritev64
536     x32     rt_tgsigqueueinfo       __x32_compat_sys_rt_tgsigqueueinfo
537     x32     recvmmsg                __x32_compat_sys_recvmmsg
538     x32     sendmmsg                __x32_compat_sys_sendmmsg
539     x32     process_vm_readv        __x32_compat_sys_process_vm_readv
540     x32     process_vm_writev       __x32_compat_sys_process_vm_writev
541     x32     setsockopt              __x32_compat_sys_setsockopt
542     x32     getsockopt              __x32_compat_sys_getsockopt
543     x32     io_setup                __x32_compat_sys_io_setup
544     x32     io_submit               __x32_compat_sys_io_submit
545     x32     execveat                __x32_compat_sys_execveat/ptregs
546     x32     preadv2                 __x32_compat_sys_preadv64v2
547     x32     pwritev2                __x32_compat_sys_pwritev64v2
548     64      saihello                __x64_sys_saihello
549     64      saiprint                __x64_sys_saiprint
550     64      saiprocess              __x64_sys_saiprocess
551     64      saigetpid               __x64_sys_saigetpid
```

Each system call was added in a different folder under the linux-4.19.210 directory, so each of the folders have to be added in the Makefile for the kernel itself (cust1 through 4 for each question 1 to 4)

Finally, the required definitions are also added into syscalls.h

```
Select root@myVM: ~/linux-4.19.210/include/linux
  GNU nano 2.5.3                                                    Fi

        return old;
}

asmlinkage long sys_saihello(void);
asmlinkage long sys_saiprint(char __user *);
asmlinkage long sys_saiprocess(void);
asmlinkage long sys_saigetpid(void);

#endif
```

WELCOME MESSAGE TO KERNEL LOGS

1. The following code was added to saihello.c in /linux-4.19.210/ass2, along with the following Makefile containing ‘obj-y := saihello.o`.

```
root@myVM: ~/linux-4.19.210/ass2
  GNU nano 2.5.3

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

SYSCALL_DEFINE0(saihello)
{
    printk("Hello, I am Sai Teja\n");
    return 0;
}
```

2. A new syscall entry was added in syscall_64.tbl

PRINTING SOME STRING FROM USER SPACE TO KERNEL SPACE

1. The following code was added to saiprint.c in /linux-4.19.210/ass2, along with the following Makefile containing ‘obj-y := saiprint.o`.

```
root@myVM: ~/linux-4.19.210/ass2
  GNU nano 2.5.3                                                  File

#include <linux/kernel.h>
#include <linux/linkage.h>
#include <linux/syscalls.h>
#include <linux/uaccess.h>

SYSCALL_DEFINE2(saiprint,
                char __user *, src,
                int, len)
{
        char buf[256];
        unsigned long lenleft = len;
        unsigned long chunklen = sizeof(buf);
        while( lenleft > 0 ){
                if( lenleft < chunklen ) chunklen = lenleft;
                if( copy_from_user(buf, src, chunklen) ){
                return -EFAULT;
        }
                lenleft -= chunklen;
        }

        printk("%s\n", buf);

        return 0;
}
```

Here, SYSCALL_DEFINE2 is a macro that is creating a function asmlinkage long saiprint(char __user * str,int len), with one argument str and one int. The '__user' marks the pointer as one to the user space. 'Copy_from_user' is an API call that allows copying data from a pointer in userspace to kernelspace buffer. If there is some sort of error while fetching information, EFAULT error flag will be returned.

2. A new syscall entry is added in syscall_64.tbl. It is different from the first entry in that it has a new sys_ prefix and __x64_sys_ prefix for alias and syscall name respectively. This is a result of using the SYSCALL_DEFINE# macro.

PRINTING CURRENT AND PARENT PROCESS ID

1. The following code was added to saiprocess.c in /linux-4.19.210/ass2, along with the following Makefile containing 'obj-y := saiprocess.o`.

```
Select root@myVM: ~/linux-4.19.210/ass2
  GNU nano 2.5.3                                                    File

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

SYSCALL_DEFINE0(saiprocess)
{
    struct task_struct *parent=current->parent;
    printk("parent_process_pid: %d \n", parent->pid);

    printk("current_process_pid: %d \n", current->pid);
    return 0;

}
```

‘Current’ refers to a structure of type ‘task’ that stores information about the current process (that includes the entirety of the Process Control Block). The current->pid element will have the current process id. The parent->pid call will return the pid of the parent process given a task structure for current process.

2. Appropriate entry in syscall_64.tbl is added

RECREATING A SYSCALL: ‘getpid()’

1. The following code was added to saigetpid.c in /linux-4.19.210/ass2, along with the following Makefile containing ‘obj-y := saigetpid.o`.

```
root@myVM: ~/linux-4.19.210/ass2
  GNU nano 2.5.3                                                    File

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

SYSCALL_DEFINE0(saigetpid)
{
    printk("%u\n",task_tgid_vnr(current));
    return task_tgid_vnr(current);
}
```

The uidgid.h header contains the k_uid structure that will have the user id stored in it. Syscall getuid() itself fetches this information from this place. Current_uid() will return the k_uid structure and val will return the uid unsigned long from it.

2. Appropriate entry in syscall_64.tbl is added

<u>OUTPUTS</u>

```
root@myVM: ~
  GNU nano 2.5.3                                                        File: t

#include<stdio.h>
#include<unistd.h>
#include<sys/syscall.h>

int main(int argc,char** argv){
        syscall(548);
        syscall(549, "Hello World ", 100);
        syscall(550);
        printf("%ld",syscall(551));
        return 0;
}
```

The above code has been used to test the newly created syscalls 548 to 551.

The kernel logs are as follows

```
root@myVM: ~
root@myVM:~# ./a.out
2062root@myVM:~# dmesg
[  422.979801] Hello, I am Sai Teja
[  422.979806] Hello World
[  422.979807] parent_process_pid: 1998
[  422.979808] current_process_pid: 2062
[  422.979809] 2062
root@myVM:~#
```

Parent and current process ids are different. Why?

The process id refers to the process that is handling both the syscall and the function that called the syscall. That is, in this case saiprocess.c and test.c are part of the same process. Calling another function does not necessarily create a new process, hence 2062 in this case refers to our currently running functions.

The parent process id 1998 refers to the parent process of test.c, the code within which the system call is being tested.