

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 run `sudo make` and `sudo make modules_install`.
6. Made some configurations and rebooted the vm.

ADDING A SYSTEM CALL TO KERNEL

1. Create a new directory in /linux-4.19.210 for a system call. In this text, let it be 'assi2'.
2. Create a c file (eg divihello.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 := divihello.o' to it, if the name of the c file is divihello.c.
4. Navigate to /linux-4.19.210/include/linux and open syscalls.h
5. Add the definition of the function created in divihello.c
6. Navigate to /linux-4.19.210/ and open Makefile
7. Add the name of the folder as 'assi2/' 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 grub and Restarted the system, 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)

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

547 x32 pwritev2      _x32_compat_sys_pwritev64v2
548 64 divihello      _x64_sys_divihello
549 64 diviprint      _x64_sys_diviprint
550 64 diviprocess    _x64_sys_diviprocess
551 64 divigetpid     _x64_sys_divigetpid
```

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)

```
Select root@MyVM: ~/linux-4.19.210
GNU nano 2.5.3 File: Makefile

mod_sign_cmd = scripts/sign-file $(CONFIG_MODULE_SIG_HASH) $(MODULE_SIG_KEY_SRCPREFIX)$(CONFIG_MODULE_SIG_KEY) certs/signing_key.x509
else
mod_sign_cmd = true
endif
export mod_sign_cmd

HOST_LIBELF_LIBS = $(shell pkg-config libelf --libs 2>/dev/null || echo -lelf)

ifdef CONFIG_STACK_VALIDATION
has_libelf := $(call try-run, \
echo "int main() {}" | $(HOSTCC) -xc -o /dev/null $(HOST_LIBELF_LIBS) -,1,0)
ifeq ($(has_libelf),1)
objtool_target := tools/objtool FORCE
else
SKIP_STACK_VALIDATION := 1
export SKIP_STACK_VALIDATION
endif
endif

PHONY += prepare0
ifeq ($(KBUILD_EXPORT0))
core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ ass2/

vmlinux-dirs := $(patsubst %/,%, $(filter %/, $(init-y) $(init-m) \
$(core-y) $(core-m) $(drivers-y) $(drivers-m) \
$(net-y) $(net-m) $(libs-y) $(libs-m) $(virt-y)))
vmlinux-alldirs := $(sort $(vmlinux-dirs) $(patsubst %/,%, $(filter %/, \
$(init-y) $(core-y) $(drivers-y) $(net-y) $(libs-y) $(virt-y))))

init-y := $(patsubst %/, %/built-in.a, $(init-y))
core-y := $(patsubst %/, %/built-in.a, $(core-y))
drivers-y := $(patsubst %/, %/built-in.a, $(drivers-y))
net-y := $(patsubst %/, %/built-in.a, $(net-y))
libs-y1 := $(patsubst %/, %/lib.a, $(libs-y))
libs-y2 := $(patsubst %/, %/built-in.a, $(filter-out %.a, $(libs-y)))
virt-y := $(patsubst %/, %/built-in.a, $(virt-y))

# Externally visible symbols (used by link-vmlinux.sh)
export KBUILD_VMLINUX_INIT := $(head-y) $(init-y)
export KBUILD_VMLINUX_MAIN := $(core-y) $(libs-y2) $(drivers-y) $(net-y) $(virt-y)
export KBUILD_VMLINUX_LIBS := $(libs-y2)
export KBUILD_LDS := arch/$(SRCARCH)/kernel/vmlinux.lds
export LDFLAGS_vmlinux

Get Help Write Out Where Is Cut Text Justify Cur Pos Prev Page First Line WhereIs Next Mark Text Indent Text Undo
Exit Exit Read File Replace Uncut Text To Spell To Line Next Page Last Line To Bracket To Bracket Copy Text Unindent Text Redo
```

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

```
azureuser@MyVM: ~/linux-4.19.210/include/linux
GNU nano 2.5.3 File: syscalls.h

        set_personality(personality);

        return old;
}

asmlinkage long sys_divihello(void);
asmlinkage long sys_diviprint(char __user *);
asmlinkage long sys_diviprocess(void);
asmlinkage long sys_divigetpid(void);

#endif
```

WELCOME MESSAGE TO KERNEL LOGS

1. The following code was added to q1.c in /linux-4.19.210/assi2, along with the following Makefile containing 'obj-y := divihello.o'.

```
azureuser@MyVM: ~/linux-4.19.210/assi2
GNU nano 2.5.3

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

SYSCALL_DEFINE0(divihello)
{
    printk("Hiii!! Welcome! I am Divishath \n");
    return 0;
}
```

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

PRINTING SOME STRING FROM USER SPACE TO KERNEL SPACE

3. The following code was added to diviprint.c in /linux-4.19.210/assi2, along with the following Makefile containing 'obj-y := diviprint.o'.

```
azureuser@MyVM: ~/linux-4.19.210/assi2
GNU nano 2.5.3

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

SYSCALL_DEFINE2(diviprint,
                char __user *, input,
                int, input_length)
{
    char buffer[256];
    unsigned long left_length = input_length;
    unsigned long chunklen = sizeof(buffer);
    while( left_length > 0 ){
        if(left_length < chunklen ) chunklen =left_length;
        if( copy_from_user(buffer, input, chunklen) ){
            return -EFAULT;
        }
        left_length -= chunklen;
    }

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

    return 0;
}
```

Here, SYSCALL_DEFINE2 is a macro that is creating a function asmlinkage long diviprint(char __user * str), with one argument str. 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.

1. 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

4. The following code was added to diviprocess.c in/linux-4.19.210/assi2, along with the following Makefile containing 'obj-y := diviprocess.o`.

```
azureuser@MyVM: ~/linux-4.19.210/assi2
GNU nano 2.5.3

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

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

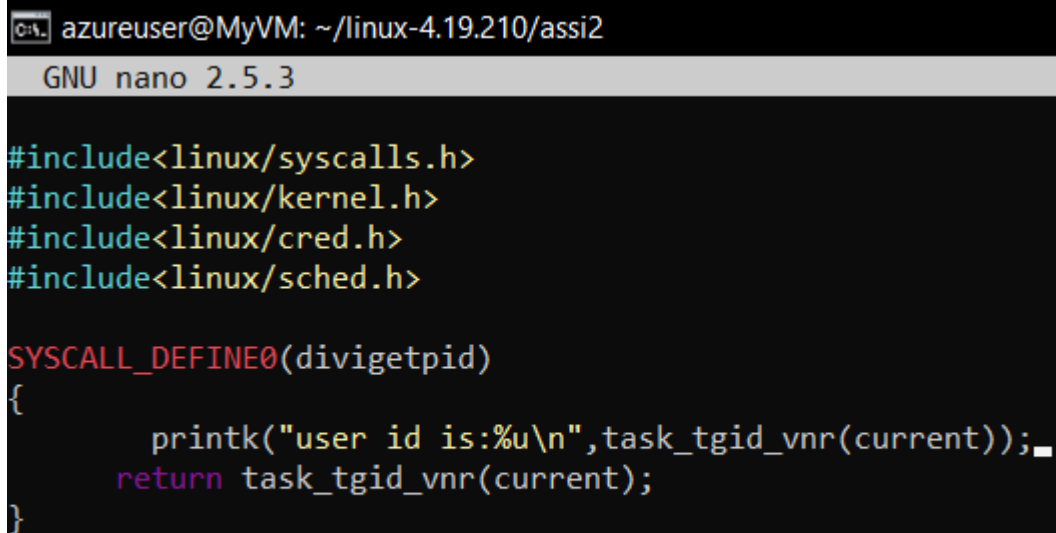
    printk("current_process_pid: %d \n", current->pid);
    return 0;
}
```

'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 pid element will have the current process id. The parent->pid will give parent process id of current process.

1. Appropriate entry in syscall_64.tbl is added

RECREATING A SYSCALL: 'getpid()'

5. The following code was added to divigetpid.c in /linux-4.19.210/assi2, along with the following Makefile containing 'obj-y := divigetpid.o`.



```
azureuser@MyVM: ~/linux-4.19.210/assi2
GNU nano 2.5.3

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

SYSCALL_DEFINE0(divigetpid)
{
    printk("user id is:%u\n",task_tgid_vnr(current));
    return task_tgid_vnr(current);
}
```

Appropriate entry in syscall_64.tbl is added

OUTPUTS

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

The userspace output(printing output of 4th question) and the kernel logs are as follows:

```
root@MyVM:~# ./a.out
1999
root@MyVM:~# dmesg
[ 112.828385] Hiii!! Welcome! I am Divishath
[ 112.828388] testing output of question 2
[ 112.828389] parent_process_pid: 1966
[ 112.828390] current_process_pid: 1999
[ 112.828391] User id is:1999
```

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 diviprocess.c and userspace.c are part of the same process. Calling another function does not necessarily create a new process, hence 1999 in this case refers to our currently running functions.

The parent process id 1966 refers to the parent process of userspace.c, the code within which the system call is being tested.