Advanced OS Assignment 2

Objective: **Add new system calls to the Linux kernel version 4.19.210 and test them.**

Steps to add system call in linux kernel

- After downloading the desired kernel version (4.19.210) extract the kernel source code in /src/usr/

  ```
  sudo tar -xvf linux-4.19.210.tar.xz -C/usr/src/
  ```

- Add a directory to the extracted folder with the desired system call name. This directory will contain the required source code and Makefile of the system call that we are going to implement.

- Once the system call source code is added, modify the Makefile of the kernel to include the newly added syscall

- Now that the kernel is configured, we can start compiling our system call. However, creating a system call requires editing a table that is included by a truly huge amount of code. The file containing the system call table for x86_64 is located in *arch/x86/entry/syscalls/syscall_64.tbl.*

- This table is read by scripts and used to generate some of the boilerplate code, go to the bottom of the first group (it ends at syscall 334 in version 4.19.210), and add the new system call that we have made

  ```
  335      64     vyshakprint     _x64_sys_vyshakprint
  ```

  The first column is the system call number. Chose the next available number in the table, which in this case was 335. You should also choose the next available number, which may not be 335. The second column says that this system call is for 64-bit CPUs. The third column is the name of the system call, and the fourth is the name of the function implementing it.

- Finally before compiling add the systemcall to systemcall header file located at *include/linux/syscalls.h*

- Now that we have added the systemcall after compiling kernel and a reboot of the system changes will be reflected.

Question 1: **Syscall to print a welcome message to Linux logs**

```
root@iresh: /usr/src/linux-4.19.210/vyshakhello
root@iresh:/usr/src/linux-4.19.210# mkdir vyshakhello
root@iresh:/usr/src/linux-4.19.210# cd vyshakhello/
root@iresh:/usr/src/linux-4.19.210/vyshakhello#
```

Creating directory for systemcall

After extracting the new kernel image we create a new directory under the name `vyshakhello`(*same as syscall name*)

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

asmlinkage long vyshak_hello(void)
{
        printk("Hello .......!\n");
        return 0;
}
```

In the newly created the directory add the above source code which will do the required functionality of our systemcall. In this case we are just printing the string Hello……! and returning the value of 0.

```
root@iresh:/usr/src/linux-4.19.210/vyshakhello# ls
Makefile  vyshakhello.c
root@iresh:/usr/src/linux-4.19.210/vyshakhello# cat Makefile
obj-y := vyshakhello.o
root@iresh:/usr/src/linux-4.19.210/vyshakhello# cd ..
root@iresh:/usr/src/linux-4.19.210#
```

Create systemcall Makefile

We also add a Makefile along with source code in the same directory.

Now that we have added the source code we need to mention the source code location in the Makefile of kernel and we need to mention the same in the systemcall table of the kernel.

```
ifeq ($(KBUILD_EXTMOD),)
core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ vyshakhello/

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)))
```

Modify Kernel Makefile

```
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      vyshakhello             vyshak_hello
```

Adding new sytemcall in systemcall table.

Finally we need add the systemcall to the systmcall header file *(include/linux/syscalls.h)*

```
        return do_sys_truncate(pathname, length);
}

static inline unsigned int ksys_personality(unsigned int personality)
{
        unsigned int old = current->personality;

        if (personality != 0xffffffff)
                set_personality(personality);

        return old;
}

asmlinkage long vyshak_hello(void);

#endif
```

Adding systemcall in systemcall header file

Now that we have added the system call and the made the changes in the required files we can compile(*sudo make -j4*) the kernel and reboot the system to verify it.

For verifying we are using the following code and and calling the syscall with the its corresponding number

```
#include <stdio.h>
#include<linux/kernel.h>
#include<sys/syscall.h>
#include<unistd.h>

int main()
{
        long int data= syscall(548);
        printf("System call vyshak_hello returned %ld/n", data);
        return 0;
}
```

Verification code for systemcall

For this code we are getting the output 0, which implies successful run of systemcall.

```
root@iresh: /home/iresh
user->./a.out
System call vyshak_hello returned 0
user->
```

Now to verify the objective of our system call we check the `dmesg` logs

```
[  514.303083] Hello......!
[  554.935477] Hello......!
[  949.467466] Hello......!
[  965.833856] Hello......!
[ 1027.662002] Hello......!
[ 1076.663473] Hello......!
```

Output of the system call in dmesg

Question 2: **Syscall which will receive string parameter and print it along with some message to kernel logs**

```
user->sudo mkdir vyshakprint
user->cd vyshakprint/
user->gedit vyshakprint.c
^C
user->sudo !!
sudo gedit vyshakprint.c

(gedit:4279): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was n
ot provided by any .service files

** (gedit:4279): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported

** (gedit:4279): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported

** (gedit:4279): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
user->gedit Makefile
user->sudo !!
sudo gedit Makefile

(gedit:4333): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was n
ot provided by any .service files

** (gedit:4333): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported

** (gedit:4333): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported

** (gedit:4333): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
user->ls
Makefile  vyshakprint.c
user->cat Makefile
obj-y := vyshakprint.o
user->
```

Creating systemcall dir and adding the source code

Initially similar to previous question we add a new directory under the systemcall name and add the source code and Makefile required for the systemcall fucntionality.

```c
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/sched.h>
#include <linux/cred.h>
SYSCALL_DEFINE1(vyshakprint, char __user *, data){
	char buffer[1500];
	if(copy_from_user(buffer,data, sizeof(buffer))){
		return -EFAULT;
	}
	printk ("Recieved input = %s. \n",buffer);
	return 0;
}
```

For this systemcall we are receiving string input from the user and printing it in the kernel logs. The above syscall will take input into data (*max 1500 char*) and print it along with the `Recieved input =` message.

```makefile
PHONY += prepare0

ifeq ($(KBUILD_EXTMOD),)
core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ vyshakhello/ vyshakprint/ vyshakprocess/ vyshakgetpid/

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)))
```

Kernel Makefile modification

```
328     64      pwritev2                __x64_sys_pwritev2
329     common  pkey_mprotect           __x64_sys_pkey_mprotect
330     common  pkey_alloc              __x64_sys_pkey_alloc
331     common  pkey_free               __x64_sys_pkey_free
332     common  statx                   __x64_sys_statx
333     common  io_pgetevents           __x64_sys_io_pgetevents
334     common  rseq                    __x64_sys_rseq
335     64      vyshakprint             __x64_sys_vyshakprint
```

Systemcall table

Now we add the systemcall to systemcall table appending to the existing ones. And add the new system call in the systemcall header file (*`include/linux/syscalls.h`*)

```c
static inline unsigned int ksys_personality(unsigned int personality)
{
	unsigned int old = current->personality;

	if (personality != 0xffffffff)
		set_personality(personality);

	return old;
}

asmlinkage long vyshakhello(void);
asmlinkage long vyshakprint(void);
asmlinkage long vyshakprocess(void);
asmlinkage long vyshakgetpid(void);

#endif
```

Adding the systemcall in systemcall header file.

Now that we have added the source code and made the required changes in the systemcall files we are verifying our systemcall with the below C code.

```c
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
int main()
{

        char st[256];

        sprintf(st, "It's vyshak here!!");
        long int stat = syscall(336, st);
        printf("System call vyshakprint returned: %ld\n", stat);

return 0;
}
```

Verification code

Here we are inputting the string(`It's vyshak here!!`) to systemcall with the help of sprintf function.

```
user->sudo gcc userspace.c
user->./a.out
System call vyshakprint returned: 0
user->
```

Output of verification code

For this code we are getting the output 0, which implies successful run of systemcall.

Now to verify the objective of our system call we check the `dmesg` logs

```
[ 4387.196865] Recieved input = It's vyshak here!!.
[ 4408.951173] Recieved input = It's vyshak here!!.
```

Output of the system call in dmesg

Question 3: **Syscall to print the parent process id and current process id upon calling it**

```
user->sudo mkdir vyshakprocess
user->cd vyshakprocess/
user->sudo gedit vyshakprocess.c

(gedit:4697): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was n
ot provided by any .service files

** (gedit:4697): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported

** (gedit:4697): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported

** (gedit:4697): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
user->gedit Makefile
user->sudo !!
sudo gedit Makefile

(gedit:4757): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was n
ot provided by any .service files

** (gedit:4757): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported

** (gedit:4757): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported

** (gedit:4757): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
user->ls
Makefile  vyshakprocess.c
user->cat Makefile
obj-y := vyshakprocess.o
user->
```

Creating systemcall dir and adding the source code

For implementing new systemcall we add a directory for system call in the kernel folder. In this folder add source code and Makefile required for the systemcall fucntionality.

```c
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/sched.h>
#include <linux/cred.h>
asmlinkage long vyshakprocess(void){

        struct task_struct *parents=current->parent;
        printk ("Current id= %d. \n", current->pid);
        printk ("Parent id= %d. \n", parents->pid);

        return 0;
}
```

In the above implemented systemcall we are getting the current PID and parent PID of the process and printing them in the logs.

```makefile
PHONY += prepare0

ifeq ($(KBUILD_EXTMOD),)
core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ vyshakhello/ vyshakprint/ vyshakprocess/ vyshakgetpid/

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-) $(core-) $(drivers-) $(net-) $(libs-) $(virt-))))
```

Kernel Makefile

After adding the required systemcall we mention the systemcall directory in the Makefile of the kernel.

Now we make changes in the systemcall table and the system call header file to add our sytemcall (*include/linux/syscalls.h*)

```
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      vyshakhello             vyshak_hello
549     64      vyshakprocess           vyshakprocess
550     64      vyshakgetpid            vyshakgetpid
```

Systemcall table

```
        if (personality != 0xffffffff)
                set_personality(personality);

        return old;
}

asmlinkage long vyshakhello(void);
asmlinkage long vyshakprint(void);
asmlinkage long vyshakprocess(void);
asmlinkage long vyshakgetpid(void);

#endif
```

Modification in Systemcall header file

Now that we have added the systemcall we can compile the kernel with the systemcall changes and reboot the system for the systemcall changes to take over.

We verify our systemcall with the following code and check the status oby checking the returned value.

```
#include <stdio.h>
#include<linux/kernel.h>
#include<sys/syscall.h>
#include<unistd.h>

int main()
{
        long int data= syscall(549);
        printf("System call vyshakprocess returned %ld/n", data);
        return 0;
}
```

Verification code

```
user->sudo gcc userspace.c
user->./a.out
System call vyshakprocess returned 0
user->
```

Output of verification code

Now we check the output of our syscall in the dmesg logs.

```
[ 4990.273157] Current id= 5096.
[ 4990.273159] Parent id= 3237.
[ 5014.102197] Current id= 5117.
[ 5014.102198] Parent id= 3237.
[ 5015.195299] Current id= 5118.
[ 5015.195300] Parent id= 3237.
[ 5015.682489] Current id= 5119.
[ 5015.682490] Parent id= 3237.
[ 5016.095773] Current id= 5120.
[ 5016.095774] Parent id= 3237.
[ 5016.448020] Current id= 5121.
[ 5016.448021] Parent id= 3237.
```

Output in dmesg

Here we can see that current PID and parent PID is different.

The current process is the child process of its parent (*obviously:)*), this child process is a process created by a parent process in an operating system using a fork() system call. After forking the parent process continues and creates a child process that will do the required subtask, both processes will exist in parallel.
Once the child process task is complete it will exit and the parent process will continue until its completion.

Since parent and child processes are different they will have different PIDs. In the above image, we can see that parent PID remains the same (3237) on multiple executions while the child process PID keeps on changing. This is because once the child process reaches its completion the process is terminated, now when we execute the same the same parent is getting forked and creating a new child process and this will have a new PID.

Question 4: **Syscall to execute some predefined system call from your written system call**

```
user->sudo mkdir vyshakgetpid
user->cd vyshakgetpid/
user->sudo gedit vyshakgetpid.c

(gedit:5098): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was n
ot provided by any .service files

** (gedit:5098): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported

** (gedit:5098): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported

** (gedit:5098): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported

** (gedit:5098): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported

** (gedit:5098): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
user->sudo gedit Makefile

(gedit:5141): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was n
ot provided by any .service files

** (gedit:5141): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported

** (gedit:5141): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported

** (gedit:5141): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
user->ls
Makefile  vyshakgetpid.c
user->cat Makefile
obj-y := vyshakgetpid.o
user->
```

Creating systemcall dir and adding the source code

Similar to first question we add a new directory under the desired systemcall name and add the source code and Makefile required for the systemcall fucntionality.

Here we are returning the PID of the running function

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

asmlinkage long vyshakgetpid(void){
        return task_tgid_vnr(current);
}
```

```
  else
    SKIP_STACK_VALIDATION := 1
    export SKIP_STACK_VALIDATION
  endif
endif

PHONY += prepare0

ifeq ($(KBUILD_EXTMOD),)
core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ vyshakhello/ vyshakprint/ vyshakprocess/ vyshakgetpid/

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)))
```

Modification in Kernel Makefile

After adding the required systemcall we mention the systemcall in the Makefile required for compilation of the kernel.

```
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      vyshakhello             vyshak_hello
549     64      vyshakprocess           vyshakprocess
550     64      vyshakgetpid            vyshakgetpid
```

Systemcall table

Now we make changes in the systemcall table and the system call header file to add our sytemcall (*include/linux/syscalls.h*)

```
static inline unsigned int ksys_personality(unsigned int personality)
{
        unsigned int old = current->personality;

        if (personality != 0xffffffff)
                set_personality(personality);

        return old;
}

asmlinkage long vyshakhello(void);
asmlinkage long vyshakprint(void);
asmlinkage long vyshakprocess(void);
asmlinkage long vyshakgetpid(void);

#endif
```

Systemcall headerfile

Now that we have added the source code and made the required changes in the systemcall files we can compile the kernel with the changes and verify our systemcall with the below C code.

```
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
int main()
{


        long int cur_pid = syscall(554);
        printf("System call vyshakgetpid returned %ld\n", cur_pid);

        return 0;
}
```

Verification code for vyshakgetpid systemcall

```
user->sudo gcc userspace.c
user->./a.out
System call vyshakgetpid returned 5588
user->./a.out
System call vyshakgetpid returned 5589
user->./a.out
System call vyshakgetpid returned 5590
user->./a.out
System call vyshakgetpid returned 5591
user->
```

Output returning current PID

Here we can see our verification code is returing the PID of the current process as expected.

References

- https://brennan.io/2016/11/14/kernel-dev-ep3/
- https://medium.com/anubhav-shrimal/adding-a-hello-world-system-call-to-linux-kernel-dad32875872
- https://www.stolaf.edu/people/rab/os/lab/newsyscall.html