Operating Systems Lab

Assignment 9

Name: Viraj Sonawane
Roll number: 33382
Batch: N11

Aim:

Implement a new system call, add this new system call in the Linux kernel (any kernel source, any architecture and any Linux kernel distribution) and demonstrate the use of same.

Objective:

Add a new system call, swipe(), to the Linux kernel that transfers the remaining time slice of each process in a specified set to a target process. You will also demonstrate various uses of the system call (both advantageous and detrimental) **Theory:**

Adding a simple system call:

1. Download the kernel source:

In your terminal type the following command: wget

https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.17.4.tar.xz Else go to

kernel.org and download the latest version.

2. Extract the kernel source code:

sudo tar -xvf linux-4.17.4.tar.xz -C/usr/src/

tar — Tar stores and extracts files from a tape or disk archive.

-x — extract files from an archive

-v — requested using the –verbose option, when extracting archives

-f — file archive; use archive file or device archive

-C — extract to the directory specified after it.(in this case /usr/src/) Now, we'll change the directory to where the files are extracted:

3. Define a new system call sys_hello( ):

Create a directory named hello/ and change the directory to hello/:

mkdir

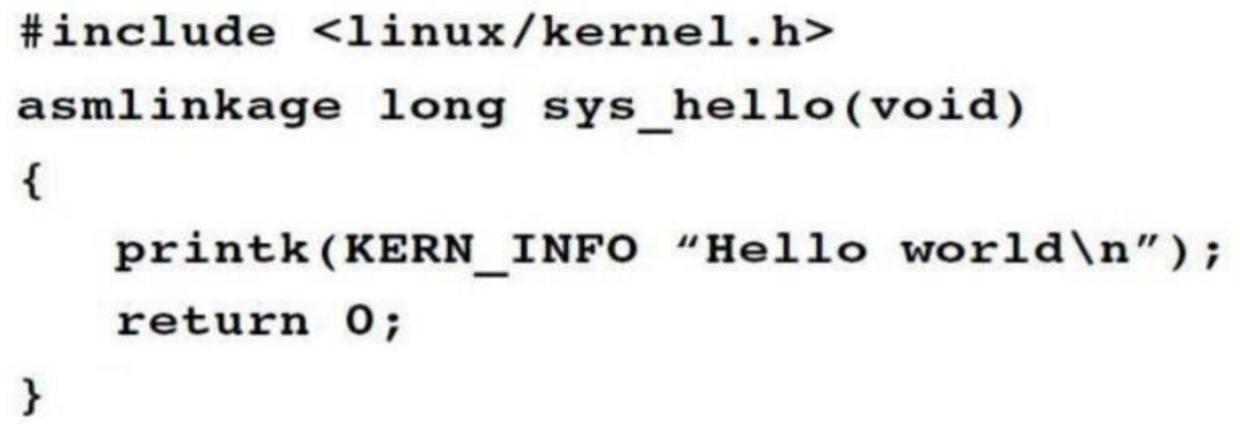
hello cd

Hello

Create a file **hello.c** using text editor:

gedit hello.c

Write the following code in the editor:



```
#include <linux/kernel.h>
asmlinkage long sys_hello(void)
{
   printk(KERN_INFO "Hello world\n");
   return 0;
}
```

printk prints to the kernel's log file.

Create a "Makefile" in the hello directory:

gedit Makefile and add the following

line to it:

obj-y := hello.o

This is to ensure that the hello.c file is compiled and included in the kernel source code.

Note: There is no space in between"obj-y".

4. Adding hello/ to the kernel's Makefile:

Go back to the parent dir i.e. cd ../ and open "Makefile" **gedit Makefile**

search for core-y in the document, you'll find this line as the second instance of your search:

core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ Add 'hello/' to

the end of this line: core-y += kernel/ mm/ fs/ ipc/ security/ crypto/

block/ hello/

Note: There is a space between "block/" and "hello/". (Doing such a mistake may cause errors in further steps)

This is to tell the compiler that the source files of our new system call (sys_hello()) are in present in the hello directory.

5. Add the new system call to the system call table:

If you are on a 32-bit system you'll need to change 'syscall_32.tbl'.

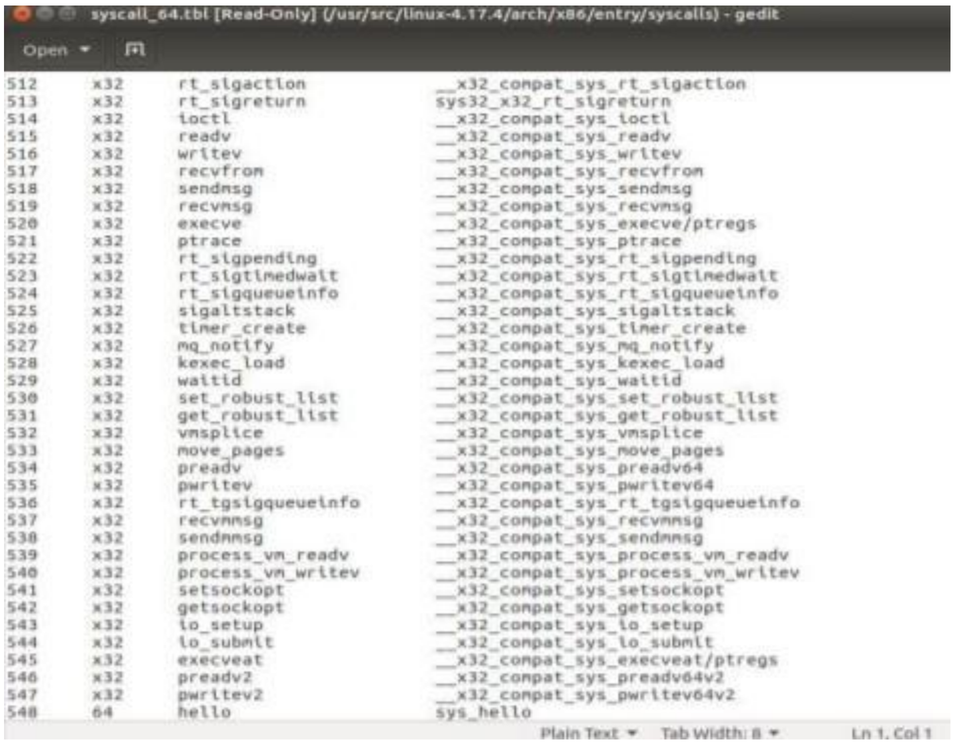
For 64-bit, change 'syscall_64.tbl'.

Run the following commands in your terminal from linux-4.17.4/ directory:

cd arch/x86/entry/syscalls / gedit

syscall_64.tbl

You'll get a file like the following in your editor:


syscall_64.tbl [Read-Only] (/usr/src/linux-4.17.4/arch/x86/entry/syscalls) - gedit
Open
Plain Text
Tab Width: 8
Ln 1, Col 1

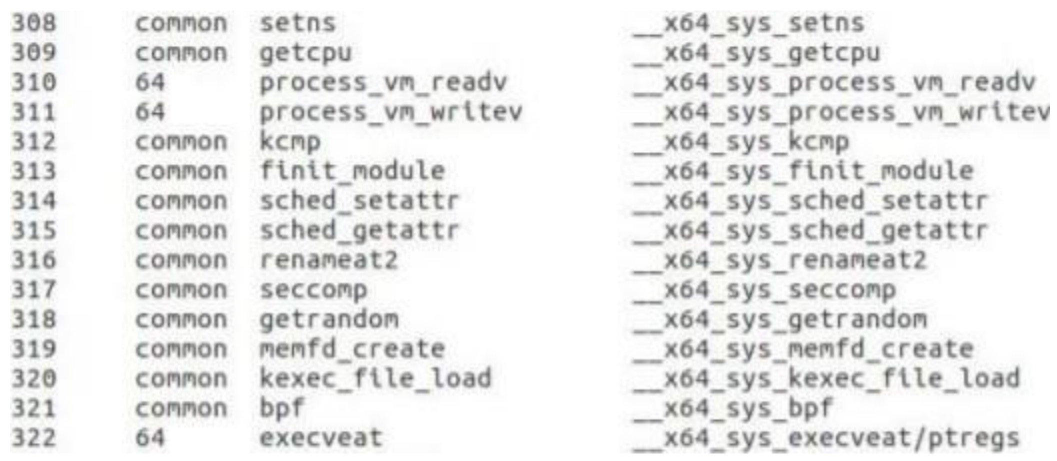
```
512	x32	rt_sigaction		__x32_compat_sys_rt_sigaction
513	x32	rt_sigreturn		sys32_x32_rt_sigreturn
514	x32	ioctl			__x32_compat_sys_ioctl
515	x32	readv			__x32_compat_sys_readv
516	x32	writev			__x32_compat_sys_writev
517	x32	recvfrom		__x32_compat_sys_recvfrom
518	x32	sendmsg			__x32_compat_sys_sendmsg
519	x32	recvmsg			__x32_compat_sys_recvmsg
520	x32	execve			__x32_compat_sys_execve/ptregs
521	x32	ptrace			__x32_compat_sys_ptrace
522	x32	rt_sigpending		__x32_compat_sys_rt_sigpending
523	x32	rt_sigtimedwait		__x32_compat_sys_rt_sigtimedwait
524	x32	rt_sigqueueinfo		__x32_compat_sys_rt_sigqueueinfo
525	x32	sigaltstack		__x32_compat_sys_sigaltstack
526	x32	timer_create		__x32_compat_sys_timer_create
527	x32	mq_notify		__x32_compat_sys_mq_notify
528	x32	kexec_load		__x32_compat_sys_kexec_load
529	x32	waitid			__x32_compat_sys_waitid
530	x32	set_robust_list		__x32_compat_sys_set_robust_list
531	x32	get_robust_list		__x32_compat_sys_get_robust_list
532	x32	vmsplice		__x32_compat_sys_vmsplice
533	x32	move_pages		__x32_compat_sys_move_pages
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	hello			sys_hello
```

Go to the last of the document and add a new line like so:

548 64 hello sys_hello

Note: Here 548 is written because in the previous line the number entry was 547. Remember this number it will be used in the later steps.

Also, note that I've written 64 in my system because it is 64 bit. You may have to write i586 or x32. For knowing what is to be written check in this file itself in many of the lines you may find entries like so:



```
308     common  setns                   __x64_sys_setns
309     common  getcpu                  __x64_sys_getcpu
310     64      process_vm_readv        __x64_sys_process_vm_readv
311     64      process_vm_writev       __x64_sys_process_vm_writev
312     common  kcmp                    __x64_sys_kcmp
313     common  finit_module            __x64_sys_finit_module
314     common  sched_setattr           __x64_sys_sched_setattr
315     common  sched_getattr           __x64_sys_sched_getattr
316     common  renameat2               __x64_sys_renameat2
317     common  seccomp                 __x64_sys_seccomp
318     common  getrandom               __x64_sys_getrandom
319     common  memfd_create            __x64_sys_memfd_create
320     common  kexec_file_load         __x64_sys_kexec_file_load
321     common  bpf                     __x64_sys_bpf
322     64      execveat                __x64_sys_execveat/ptregs
```

64 written at 310, 311 and 322 line numbers.

This will tell you whether to write i586 or something else.

Save and exit.

6. Add new system call to the system call header file:

Go to the linux-4.17.4/ directory and type the following commands:

cd include/linux/

gedit syscalls.h

Add the following line to the end of the document before the #endif statement:

asmlinkage long sys_hello(void);

Save and exit. This defines the prototype of the function of our system call. “asmlinkage” is a key word used to indicate that all parameters of the function would be available on the stack.

7. Compile the kernel:

Before starting to compile you need to install a few packages. Type the following commands in your terminal:

sudo apt-get install gcc

sudo apt-get install libncurses5-dev

sudo apt-get ev sudo apt-get install

bison sudo apt-get install flex sudo

apt-get install libssl-dev sudo apt-

get install libelf-dev sudo apt-get

update sudo apt-get upgrade

to configure your kernel use the following command in your **linux-4.17.4/** directory:

sudo make menuconfig

Once the above command is used to configure the Linux kernel, you will get a pop up window with the list of menus and you can select the items for the new configuration. If your unfamiliar with the configuration just check for the file systems menu and check whether “ext4” is chosen or not, if not select it and save the configuration.

Now to compile the kernel you can use the make command:

sudo make

Pro Tip:

The make command can take a lot of time in compiling, to speed up the process you can take advantage of the multiple cores that our systems have these days. Simply type, **sudo make -jn** where n is the number of cores that you have in your linux system.
For example if you have a Quad core(4) processor, you can write:

sudo make -j4 this will speed up my make process 4x

times.

This might take an hours or more depending on your system.

8. Install / update Kernel:

Run the following command in your terminal:

sudo make modules_install install

It will create some files under /boot/ directory and it will automatically make a entry in your grub.cfg.

To check whether it made correct entry, check the files under /boot/ directory .

If you have followed the steps without any error you will find the following files in it in addition to others.

1. System.map-4.17.4
2. vmlinuz-4.17.4
3. initrd.img-4.17.4
4. config-4.17.4

Now to update the kernel in your system reboot the system. You can use the following command:

shutdown -r now

After rebooting you can verify the kernel version using the following command:

uname -r

It will display the kernel version like so:

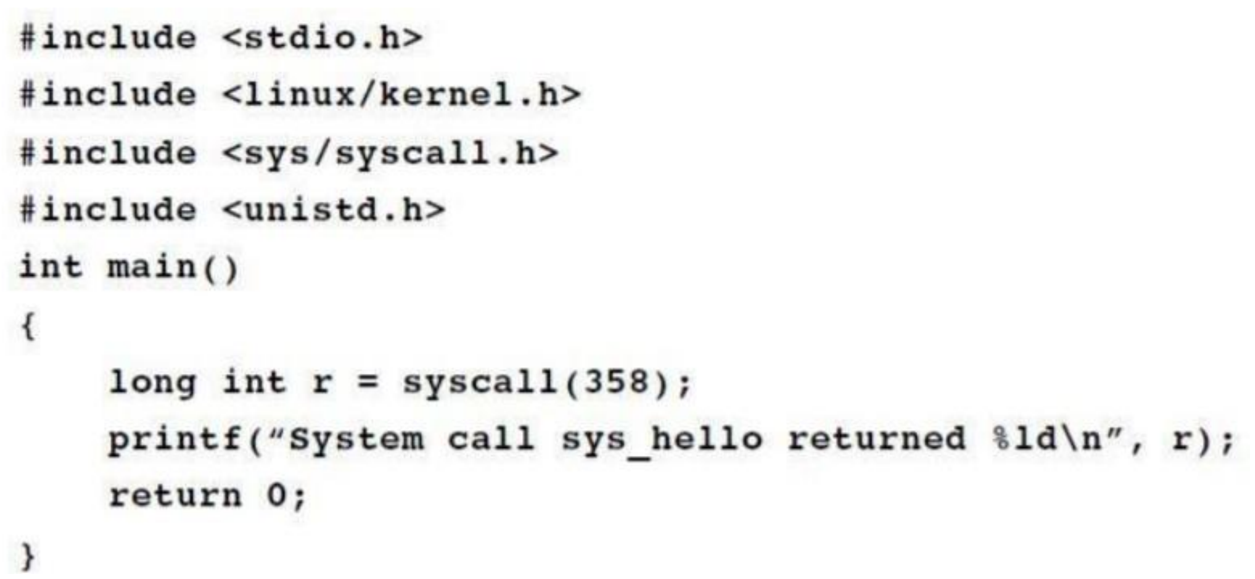
4.17.4

9. Test system call:

Go to your home(~) directory using the following commands and create a userspace.c file.

cd ~ gedit

userspace.c



```
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
int main()
{
    long int r = syscall(358);
    printf("System call sys_hello returned %ld\n", r);
    return 0;
}
```

Note: Remember to keep in mind the number of system call that is added in syscalls_64.tbl? In my case the number was 548. Write that same number in your userspace.c file as an argument in syscall() function.

Now, compile and run the program: gcc

userspace.c

./a.out

If all the steps are done correctly you'll get an output like below:

System call sys_hello returned 0

Now, to check the message of your kernel run the following command:

dmesg

This will display Hello world at the end of the kernel's message.

Write the following code in this file:

Conclusion:

A new system call, add this new system call in the Linux kernel (any kernel source, any architecture, and any Linux kernel distribution) was studied.