ADVANCED OPERATING SYSTEMS

ASSIGNMENT - 2

Report By:
Ziyad Naseem (2021201021)

STEP BY STEP PROCEDURE

1- Download the Linux kernel version 4.19.210 from www.kernel.org.

2- Extract the kernel source code by typing the following command in the terminal: sudo tar -xvf linux-4.19.210.tar.xz -C/usr/src/

3- Go to the directory where the files are extracted, i.e., C/usr/src/linux.4.19.20/

4- Create a new directory here, say ‘hello/’. Create .c files for all the questions separately and save them.

For example, here's a snapshot of ziyadhello.c file I created.

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

SYSCALL_DEFINE0(ziyadhello)
{
        printk("Hello New Linux Kernel\n");
        return 0;
}
```

The printk function prints to the kernel's log file.

SYSCALL_DEFINE0 is a macro that defines a system call with 0 parameters.

'ziyadhello' is the name of the system call.

5- Create a ‘Makefile’ in the ‘hello’ directory and write the following command in it.

Open

```
obj-y := ziyadhello.o ziyadprint.o ziyadprocess.o ziyadgetpid.o
```

This ensures that all our .c files are compiled and included in the kernel source code.

6- Go to the parent directory. Open ‘Makefile’. Search for core-y in the file and add ‘hello/’ at the end as shown below.

```
Open ▾                                                    M
                                                    /usr/src
  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/ hello/

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-))))
```

This is to tell the compiler that the source files of our new system calls are present in the ‘hello’ directory.

7- From the linux-4.19.210 directory, go to the directory arch/x86/entry/syscalls/ and open the file syscall_64.tbl (if your system runs a 64-bit version).

Add the new system calls to the system call table as shown below:

```
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      ziyadhello              __x64_sys_ziyadhello
549     64      ziyadprint              __x64_sys_ziyadprint
550     64      ziyadprocess            __x64_sys_ziyadprocess
551     64      ziyadgetpid             __x64_sys_ziyadgetpid
```

8- From the linux-4.19.210 directory, go to include/linux/ directory and open the file ‘syscalls.h’.

Add new system calls to the system call header file as shown below:

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

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

        return old;
}
asmlinkage long sys_ziyadhello(void);
asmlinkage long sys_ziyadprint(char __user *);
asmlinkage long sys_ziyadprocess(void);
asmlinkage long sys_ziyadgetpid(void);

#endif
```

This defines the prototypes of our system call functions.

9- After making sure the necessary packages are installed, compile the kernel using the following command:

sudo make

10- Install the kernel using the following command:

sudo make modules_install install

Reboot the system to update the kernel.

11- To test our new system calls, create a file ‘test.c’ as shown in the next slide:

```c
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
int main()
{
        char* a="My string";
        long int i = syscall(548);
        printf("System call ziyadhello returned %ld\n", i);
       long int x = syscall(549,a);
        printf("System call ziyadprint returned %ld\n", x);
       long int y = syscall(550);
        printf("System call ziyadprocess returned %ld\n", y);
       long int z = syscall(551);
        printf("System call ziyadgetpid returned %ld\n", z);
        return 0;
}
```

Note that the numbers 548,549,550, and 551 are the numbers that we provided to our system calls respectively in the system call table.

12- The output is as shown below:

```
ziyad@ziyad-VirtualBox: ~
File  Edit  View  Search  Terminal  Help
ziyad@ziyad-VirtualBox:~$ uname -r
4.19.210
ziyad@ziyad-VirtualBox:~$ gedit test.c

** (gedit:1741): WARNING **: 13:17:01.037: Set document metadata failed: Unable
to set metadata key
ziyad@ziyad-VirtualBox:~$ gcc test.c
ziyad@ziyad-VirtualBox:~$ ./a.out
System call ziyadhello returned 0
System call ziyadprint returned 0
System call ziyadprocess returned 0
System call ziyadgetpid returned 1769
ziyad@ziyad-VirtualBox:~$
```

Our first 3 systems calls returned 0, signifying successful completion.

The 4th system call returns the process ID, i.e., 1769.

13- To display the messages from the Linux logs, type ‘dmesg’. The output is as shown:

```
ziyad@ziyad-VirtualBox: ~
File  Edit  View  Search  Terminal  Help
ation="profile_load" profile="unconfined" name="snap.gnome-logs.gnome-logs" pid=
608 comm="apparmor_parser"
[   35.801037] audit: type=1400 audit(1634456442.032:34): apparmor="STATUS" oper
ation="profile_load" profile="unconfined" name="snap-update-ns.gnome-calculator"
 pid=622 comm="apparmor_parser"
[   35.824341] audit: type=1400 audit(1634456442.056:35): apparmor="STATUS" oper
ation="profile_load" profile="unconfined" name="snap-update-ns.gnome-system-moni
tor" pid=625 comm="apparmor_parser"
[   35.825964] audit: type=1400 audit(1634456442.056:36): apparmor="STATUS" oper
ation="profile_load" profile="unconfined" name="snap.gnome-system-monitor.gnome-
system-monitor" pid=610 comm="apparmor_parser"
[   49.831741] IPv6: ADDRCONF(NETDEV_UP): enp0s3: link is not ready
[   49.833654] IPv6: ADDRCONF(NETDEV_UP): enp0s3: link is not ready
[   49.839684] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control:
 RX
[   49.840037] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s3: link becomes ready
[   82.144023] rfkill: input handler disabled
[   83.931637] ISO 9660 Extensions: Microsoft Joliet Level 3
[   83.935357] ISO 9660 Extensions: RRIP_1991A
[  436.401304] Hello New Linux Kernel
[  436.401352] The received string is : My string.
[  436.401355] Parent process ID : 1700
[  436.401355] Current process ID : 1769
ziyad@ziyad-VirtualBox:~$
```

Ques- Are both process IDs same or different? Why?

Ans- No. They have different IDs simply because they are two different processes. When you call a function/run a code, a new process is created. As a result, we get two different processes.

----------THANK YOU----------