**Experiment No. 10**

**Title: Kernel space programming.**

**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.

**THEORY:**

Followings are the steps to add a new system call in linux.

**Step 1:**

change to the kernel source directory using, **cd /usr/src/linux-****3.18.5/**

**Step 2:**

* Define a new system call sys\_hello()
* Create a directory hello in the kernel source directory **mkdir hello**
* Change into this directory **cd hello**

**Step 3:**

Create a “hello.c” file in this folder and add the definition of the system call to it as given below.

**gedit hello.c**

Add the following code

#include <linux/kernel.h>

asmlinkage long sys\_hello(void)

{

printk(“Hello world\n”);

return 0;

}

Note that **printk** prints to the kernel’s log file.

**Step 4:**

Create a “Makefile” in the hello folder and add the given line to it.

* **gedit Makefile**

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.

**Step 5:**

* Add the hello directory to the kernel’s Makefile
* change back into the linux-3.18.5 folder and open Makefile
* **gedit Makefile**

goto line number 842 which says :-

“core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ ”

change this to

“core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ hello/”

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

**Step 6:**

* Add the new system call (sys\_hello() ) into the system call table (syscall\_32.tbl file)
* If your system is a 64 bit system you will need to alter the syscall\_64.tbl file.
* **cd arch/x86/syscalls**
* **gedit syscall\_32.tbl**
* **add the following line in the end of the file :-**

**358 i386 hello sys\_hello**

358 – It is the number of the system call . It should be one plus the number of the last system call.

**Step 7:**

* Add the new system call(sys\_hello() ) in the system call header file.
* **cd include/linux/**
* **gedit syscalls.h**
* add the following line to the end of the file just before the #endif statement at the very bottom.

**asmlinkage long sys\_hello(void);**

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

**Step 8:**

Compile this kernel on your system and reboot the system

**Step 9:** To test the system call.

Create a “userspace.c” program in your home folder and type in the following code :-

#include <stdio.h>

#include <linux/kernel.h>

#include <sys/syscall.h>

#include <unistd.h>

int main()

{

long int mycall = syscall(354);

printf(“System call sys\_hello returned %ld\n”, mycall);

return 0;

}

Now compile this program using the following command.

**gcc userspace.c**

Now run the program using the following command.

**./a.out**

You will see the following line getting printed in the terminal if all the steps were followed

correctly.

**“System call sys\_hello returned 0“.**

Now to check the message of the kernel run the following command.

**dmesg**

This will display “Hello world” at the end of the kernel’s message

**FAQs**

1. What is system call mechanism?
2. Explain X86 protection ring.
3. Explain library functions vs. system call. with strace and ltrace.
4. Explain steps to insert a new system call in kernel
5. Explain the use of Makefile.
6. Explain make command in detail
7. Explain make - j2/- j4 in detail.
8. Explain difference in between monolythic and micro kernel.