IMPLEMENTATION OF A NEW SYSTEM CALL

Submitted By:

K.A.SIVA VARDHAN-B160333CS

B.MAHESWARARAO-B160349CS

R.VAMSI KRISHNA-B160109CS

P.SATYANARAYANA-B160340CS

INTRODUCTION

- Implementing a new system call called *sys_hello .*
- This system call helps us to know the FILE NAME and PID of the runnable, terminated, sleeping and blocked processes along with their total count . Along with this the user also get to know the information of the high priority , static priority ,
  and normal priority of the processes along with their count.

STEPS INVOLVED

1. Creating a directory which contains source code.
2. Linking the new system call with kernel .
3. Modifying the file *syscall_64.tbl* .
4. Modifying the file *syscalls.h* .
5. Compiling and booting.

CREATING DIRECTORY

- Create a new directory “*hello*” and open that directory.
- Create a new file “*hello.c*” and write the source code for the system call in that.
- Create a Makefile which contains:

```
obj -y := hello.o
```

- It implies that our code is compiled and included in the source code.

LINKING THE SYSTEM CALL TO KERNEL

- Add *hello* directory to the Makefile .
- Through this we are specifying to the compiler that the source files of system call are present in *hello* directory.

```
    export SKIP_STACK_VALIDATION
  endif
endif


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 %/, \
```

MODIFYING *SYSCALL_64.TBL*

- It is located in /arch/x86/entry/syscalls .
- You can also know it using the command:

  "find –name syscall_64.tbl"
- Include the new system call number and it's entry point.

```
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
323 common  userfaultfd         __x64_sys_userfaultfd
324 common  membarrier          __x64_sys_membarrier
325 common  mlock2              __x64_sys_mlock2
326 common  copy_file_range         __x64_sys_copy_file_range
327 64  preadv2             __x64_sys_preadv2
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      hello               sys_hello

#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation. The __x32_compat_sys stubs are created
# on-the-fly for compat_sys_*() compatibility system calls if X86_X32
# is defined.
#
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
```

MODIFYING *SYSCALLS.H*

- It is located in /include/linux .
- Add the following line at the end of the file.

    asmlinkage long sys_hello(void) ;

```
asmlinkage long sys_old_mmap(struct mmap_arg_struct __user *arg);


/*
 * Not a real system call, but a placeholder for syscalls which are
 * not implemented -- see kernel/sys_ni.c
 */
asmlinkage long sys_ni_syscall(void);
asmlinkage long sys_hello(void);
#endif /* CONFIG_ARCH_HAS_SYSCALL_WRAPPER */


/*
 * Kernel code should not call syscalls (i.e., sys_xyzyyz()) directly.
 * Instead, use one of the functions which work equivalently, such as
 * the ksys_xyzyyz() functions prototyped below.
 */

int ksys_mount(char __user *dev_name, char __user *dir_name, char __user *type,
           unsigned long flags, void __user *data);
int ksys_umount(char __user *name, int flags);
```

COMPILING AND BOOTING

- To use the new system call we should recompile the kernel first.
- Use the following commands for that:

i. ***sudo make –j 4***
ii. ***sudo make modules_install –j 4***
iii. ***sudo make install –j 4***
iv. ***sudo update-grub***

- Restart the system.

TESTING OUR NEW SYSTEM CALL

- To test the new system call write a simple program "*test.c*" .
- Compile and execute this program.
- If it runs successfully, it'll give the corresponding cause and we can use the '*dmesg*' command to check the kernel log.

```c
#include<stdio.h>
#include<linux/kernel.h>
#include<sys/syscall.h>
#include<unistd.h>
int main()
{
printf("Invoking 'listProcessInfo' system call\n");
long int ret_status=syscall(335);
if(ret_status==0)
    printf("System call 'listProcessInfo' executed correctly. Use dmesg to check processInfo\n");
else
    printf("System call 'listProcessInfo' did not execute as expected\n");
    return 0;
}
```

THANK YOU