Linux System Calls

Class Notes

System Calls

- System calls are kernel functions which serve as an interface for kernel services.
- Applications in the user mode use system calls to transmit into kernel mode and execute kernel service operations.

Adding System Call to Linux Source Code

- Download stable kernel source code (www.kernel.org)
- Unzip the downloaded source code using **tar** command and enter into kernel source directory.
- $tar xvf <linux-source>
- I used the kernel linux-3.0. After entering into kernel source, we can see kernel tree as below

```
arch    CREDITS      drivers  include  Kbuild  lib        mm     REPORTING-BUGS  security  usr
block   crypto       firmware init     Kconfig MAINTAINERS net    samples         sound     virt
COPYING Documentation fs      ipc      kernel  Makefile   README scripts         tools
```

- Reserve unique ID for the system call
  - $**vim arch/x86/include/asm/unistd_32.h**
  - **lsproc** is the function name and manually assign the number to our syscall.
  - Add the system call as shown below

```
#define __NR_rt_tgsigqueueinfo  335
#define __NR_perf_event_open    336
#define __NR_recvmmsg           337
#define __NR_fanotify_init      338
#define __NR_fanotify_mark      339
#define __NR_prlimit64          340
#define __NR_name_to_handle_at  341
#define __NR_open_by_handle_at  342
#define __NR_clock_adjtime      343
#define __NR_syncfs             344
#define __NR_sendmmsg           345
#define __NR_setns              346
#define __NR_lsproc             347
#ifdef __KERNEL__

#define NR_syscalls 348

#define __ARCH_WANT_IPC_PARSE_VERSION
#define __ARCH_WANT_OLD_READDIR
#define __ARCH_WANT_OLD_STAT
#define __ARCH_WANT_STAT64
#define __ARCH_WANT_SYS_ALARM
```

- Declare system call prototype

  $**vim arch/x86/include/asm/syscalls.h**

```
asmlinkage int sys_set_thread_area(struct user_desc __user *);
asmlinkage int sys_get_thread_area(struct user_desc __user *);

/* X86_32 only */
#ifdef CONFIG_X86_32

/* kernel/signal.c */
asmlinkage int sys_sigsuspend(int, int, old_sigset_t);
asmlinkage int sys_sigaction(int, const struct old_sigaction __user *,
                             struct old_sigaction __user *);
unsigned long sys_sigreturn(struct pt_regs *);

/* kernel/vm86_32.c */
int sys_vm86old(struct vm86_struct __user *, struct pt_regs *);
int sys_vm86(unsigned long, unsigned long, struct pt_regs *);

asmlinkage int sys_lsproc(void);

#else /* CONFIG_X86_32 */

/* X86_64 only */
```

- Assign system call address to appropriate offset of the system call switch table

  $**vim arch/x86/kernel/syscall_table_32.S**

```
        .long sys_preadv
        .long sys_pwritev
        .long sys_rt_tgsigqueueinfo     /* 335 */
        .long sys_perf_event_open
        .long sys_recvmmsg
        .long sys_fanotify_init
        .long sys_fanotify_mark
        .long sys_prlimit64             /* 340 */
        .long sys_name_to_handle_at
        .long sys_open_by_handle_at
        .long sys_clock_adjtime
        .long sys_syncfs
        .long sys_sendmmsg              /* 345 */
        .long sys_setns
        .long sys_lsproc
```

- Implement syscall routine, syscall routine can be added to any source branch either by modifying existing source file or appending a new source. Here we have added to an existing source file **sys.c**

```
                kernel_power_off();
        }

        return ret;
}
EXPORT_SYMBOL_GPL(orderly_poweroff);

asmlinkage int sys_lsproc (void)
{
        struct task_struct *p;
        printk("\n \t process \t pid \t state \n");
        for_each_process(p){
                        printk("\n %15s \t %u \t %ld",p->comm,task_pid_nr(p),p->state);
        }
        return 0;
}
```

Compiling Linux kernel and building kernel image (x86-32-bit arch)

All of the following commands must be executed in the kernel source root folder

- Assign unique build name: Open root make file and assign build name to EXTRAVERSION macro.
  - $**vim Makefile**

```
VERSION = 3
PATCHLEVEL = 0
SUBLEVEL = 0
EXTRAVERSION =.syscall
NAME = Sneaky Weasel
```

- Choose kernel configuration

- $**make menuconfig**

  Select the required configuration, then save and exit. The configurations are saved on **.config** file.

- Initiate compile and build process
  - $**make**

  If the build got finished successfully, we could find a file **vmlinux** in the kernel source root folder.

- Install kernel headers and module objects.
  - $**make modules_install**

  Above command creates a folder in **cd /lib/modules** with the name of our syscall. Only root user can perform the above operation.

- Modify boot loader configuration with updated kernel image name and path.
  - $**make install**

- After completion, **reboot** the system. After rebooting verify the running kernel using
  - $**uname –r**

Invoking System Call

- Store syscall ID in the processor's accumulator (%eax).
- Starting with right most argument move each argument on to other processor accumulators (**ebx –eix**).
- Generate trap exception using processor specific instruction.
- Read return value of the system call from **eax** accumulator.

```c
#include<stdio.h>
#include<stdlib.h>

int lsproc()
{
        int ret;
        __asm__("movl $347,%eax");
        __asm__("int $0x80");
        __asm__("movl %eax,-4 (%ebp)");
        return ret;
}

int main()
{
        int ret;
        printf("Invoking System Call \n");
        ret=lsproc();
        if (ret < 0)
                exit (1);
        return 0;
}
```

- Using above C program we can invoke our system call. Execute it and check the output. Use **dmesg** to see the **printk** messages.

END