Project Overview

Jaebaek Seo
jaebaek@cps.kaist.ac.kr

Term Projects

1. You will measure the elapsed time to provide system call in user layer
2. You will break down the time into a few parts in kernel layer (Due of 1 & 2: 4th of Oct)
3. You will analyze some phenomenon causing fluctuation of the time (Due of 3: 25th of Oct)
4. Suggest some ideas for improvement and show evidence (Due of 4: 22nd of Nov)

Organizing a Team

- A team consists of 3 students at maximum
- A member of each team should give the team information as a comment on a notice about organizing a team, which will be posted in KLMS by TA right after this TA session
  - E.g., 20001234 / 20001235 / 20001236

Kernel Programming

(0. Set up a working environment)

1. Check out the version of your kernel

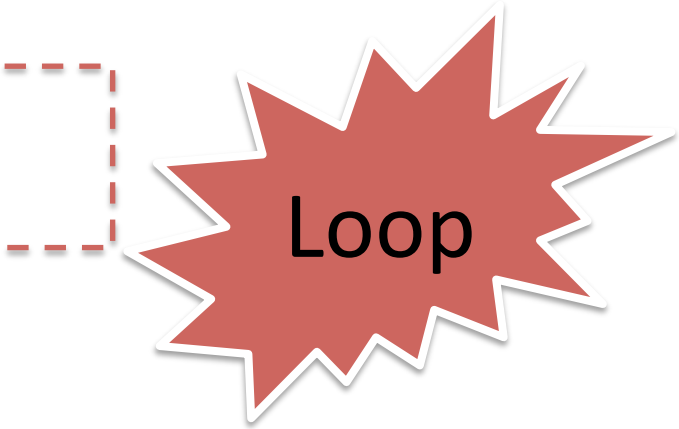
   $ uname -r # in Ubuntu

   3.19.0-25-generic

2. Download the kernel source from kernel.org

   or you can use another repository like
   https://github.com/torvalds/linux.git

3. Modify source code
4. Compile the kernel
5. Run a test application on it



Kernel Compile

```
$ sudo apt-get install build-essential     # install build tools in Ubuntu
$ make mrproper        # make the state of source code as an initial state
$ make menuconfig      # set up the compile options and generate .config
                       # "cp /boot/config-`uname -r`* .config" is an alternative
$ make dep             # check out dependencies and generate .depend
$ make modules         # compile modules and generate *.o files
                       # add -j<# of threads> options to make it faster
                       # you can use this option when you have multi cores
                       # EX) $ make -j4 modules
$ make bzImage         # generate the kernel image using the objective files
```

You can easily find out how to compile Linux kernel in online

Running Compiled Kernel on QEMU

```
$ cp path/to/linux-source/arch/x86_64/boot/bzImage .
$ gcc -static init.c -o init
$ echo init | cpio -o --format=newc > initramfs
$ qemu-system-x86_64 -m 4096 -kernel bzImage \
-initrd initramfs -serial stdio -append “console=ttyS0”
```

You can run a simple C program as the `init` process

ref:

https://www.kernel.org/doc/Documentation/filesystems/ramfs-rootfs-initramfs.txt
https://nostillsearching.wordpress.com/2012/09/22/compiling-linux-kernel-and-running-it-using-qemu/
https://balau82.wordpress.com/2012/03/31/compile-linux-kernel-3-2-for-arm-and-emulate-with-qemu/

Running Compiled Kernel on Your Machine

$ sudo make modules_install

$ sudo update-grub2

We recommend running the kernel on your machine rather than Virtual Machine (i.e., VM) to measure performance without overheads from virtualization

If you run the kernel on VM, you must have your own strategy to avoid the overheads from virtualization

System Call

In x86

```
movl $1, %eax
movl $13, %ebx
int $0x80
```

In ARM

```
mov r7, #1
mov r0, #13
svc #0
```

In x86_64

```
movq $60, %rax
movq $13, %rdi
syscall
```

Ref: http://docs.cs.up.ac.za/programming/asm/derick_tut/syscalls.html

Measuring Delay of a System Call

You do not need to use assembly because C libraries (e.g., libc) provides wrappers for system calls

```
int fd;
struct timeval start, now;
gettimeofday( &start, NULL );

fd = open( "/home/cs530/tmp\0", O_RDWR );

gettimeofday( &now, NULL );
print_time_gap( &start, &now );
```

open System Call Defined in Kernel

SYSCALL_DEFINE3(open, const char __user *, filename, int, flags, umode_t, mode)

in path/to/kernel/fs/open.c

You can use source code exploration tools such as **grep, ctags** and so on

```
long do_sys_open(int dfd, const char __user *filename, int flags, umode_t mode)
{
    struct open_flags op;
    int fd = build_open_flags(flags, mode, &op);
    struct filename *tmp;

    if (fd)
        return fd;

    tmp = getname(filename);
    if (IS_ERR(tmp))
        return PTR_ERR(tmp);

    fd = get_unused_fd_flags(flags);
    if (fd >= 0) {
        struct file *f = do_filp_open(dfd, tmp, &op);
        if (IS_ERR(f)) {
            put_unused_fd(fd);
            fd = PTR_ERR(f);
        } else {
            fsnotify_open(f);
            fd_install(fd, f);
        }
    }
    putname(tmp);
    return fd;
}
```

You can use **printk()** in kernel rather than printf()

Caveats

- Modifying system calls has effects on every user processes
  - Running many user processes on the compiled kernel can cause faults
  - In order to control this, you must provide an interface to your target user process to turn on/off your kernel logging
    EX) a system call or a device driver
- Dynamic CPU frequency and the priority of your target user process can yield unexpected delay
  - Read files in path/to/linux-source/Documentation/cpu-freq/

Any Question?