# System Programming
# Lab #7

2019-04-23

sp-tas

# Lab Assignment #4 : Kernel Lab

- Download skeleton code & pdf from eTL
  - kernellab-handout.tar, kernellab-handout.pdf
- Hand In #1 – Setup
  - capture your development environment
  - Upload your capture image eTL
    - filename for part #1 : [학번]_[이름]_kernellab_ptree.jpg (or .png, etc)
    - filename for part #2: [학번]_[이름]_kernellab_paddr.jpg (or .png, etc)
- Hand In #2 – Your Implementation
  - Upload your files eTL
    - 압축파일 양식 : [학번]_[이름]_kernellab.tar (or .zip, etc)
    - Ex) 2017-12345_홍길동_kernellab.tar
  - A zip file should include
  - (1) a tarball of your implementation directory (2) report
    - tarball 양식 : kernellab-[학번].tar.gz eg) kernellab-2019-12345.tar.gz
    - Report 양식 : [학번]_[이름]_kernellab_report.pdf (or .hwp, .txt etc)
- Please, READ the Hand-out and Lab material thoroughly!

# Lab Assignment #4 : Kernel Lab

- Step 1. Setup
  - (part #0) Load my own kernel module
- Step 2. Implementation
  - (part #1) Tracing process tree from process id
  - (part #2) Finding physical address using virtual address

- Assigned : April 23
- Deadline for Step 1. Setup : April 29, 23:59:59 (Delay NOT allowed)
- Deadline for Step 2. Implementation : May 13, 23:59:59
- Delay policy : Same as before

- Lab sessions will be
  - 4/23: Kernel lab part #0, #1
  - 4/30: Kernel lab part #2
  - 5/7 : Kernel lab Q&A session

# Linux kernel programming

- Two Programmatic ways to access kernel space
  - 1. Adding system call to system(Kernel code).
    - Need to recompile whole kernel.
  - 2. Adding Loadable Kernel Module
    - Load&unload new interface to the system.

# What is a kernel module?

- Module
  - Pieces of code that can be loaded & unloaded to kernel
- How to Compile
  - Kernel module is not compiled with general gcc
  - It needs kernel specific compile tools
- How to load & unload my code
  - Load root # insmod <module_name.ko>
  - Check root # lsmod
  - Unload root # rmmod <module_name>
- # All implemented in Makefile!

# What is debug file system?

- **Debug File System(debugfs)** is Special file system available in the Linux Kernel.

- Provides simple way for kernel developers to make information available to user space.

- User space developers can access Linux Kernel information easily using debugfs.

# Debug file system APIs

- Description of debugfs API
  - https://www.kernel.org/doc/Documentation/filesystems/debugfs.txt

```
struct dentry *debugfs_create_dir(const char *name, struct dentry *parent)
struct dentry *debugfs_create_file(const char *name, umode_t mode,
                                   struct dentry *parent, void *data
                                   const struct file_operations *fops)

struct dentry *debugfs_create_u32(const char *name, umode_t mode,
                                   struct dentry *parent, u32 *value)
struct dentry *debugfs_create_u64(const char *name, umode_t mode,
                                   struct dentry *parent, u64 *value)

struct dentry *debugfs_create_x32(const char *name, umode_t mode,
                                   struct dentry *parent, u32 *value)
struct dentry *debugfs_create_x64(const char *name, umode_t mode,
                                   struct dentry *parent, u64 *value)

struct debugfs_blob_wrapper {
        void *data,
        unsigned long size;
};
struct dentry *debugfs_create_blob(const char *name, umode_t mode,
                                   sturct dentry *parent,
                                   struct debugfs_blob_wrapper *blob)
```

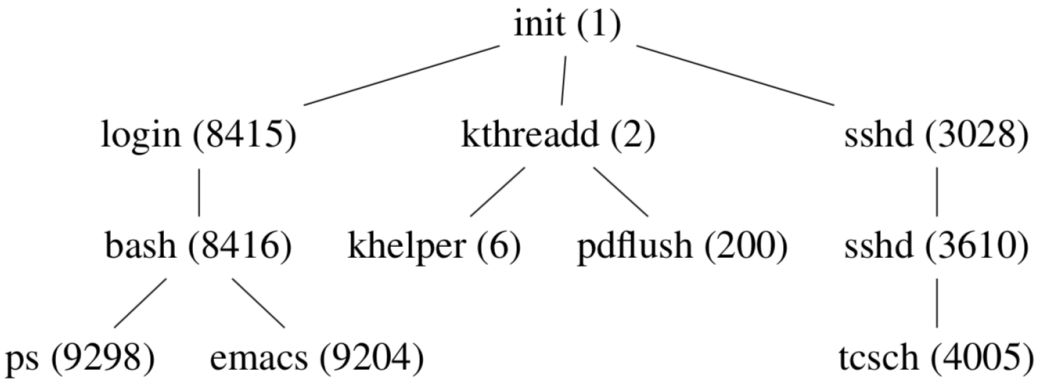
# File Operations

- File Operations
  - The file function pointer structure
  - File Operations are used to communicate with files in Device Driver and Debug File System.

```
struct file_operations Fops = {
        .read = file_read,
        .write = file_write,
        .open = file_open,
        .release = file_close,
};
```

# Kernel lab part #1 – tracing parent process tree

- Trace process from leaf to init process




- Spec
  - Input : [input process id]
  - Output : list of [process name] [process id]

Ex) input : 9204 output : init(1)

login (8415)

bash (8416)

emacs (9204)

# Kernel lab part #1 – tracing parent process tree

- Testing
  - Get root access — user# sudo su
  - Go to ptree dir — root# cd /sys/kernel/debug/ptree
  - Show current process — root# ps
  - Write input PID to file — root# echo [input process id] >> input
  - Read ptree file — root# cat ptree

- Example output

```
unix> cat ptree
init (1)
xfce4-panel (2306)
xfce4-terminal (2408)
bash (2413)
sudo (2881)
```

# With Skeleton Code

```
static int __init dbfs_module_init(void)
{
        // Implement init module code


        dir = debugfs_create_dir("ptree", NULL);

        if (!dir) {
                printk("Cannot create ptree dir\n");
                return -1;
        }

        inputdir = debugfs_create_file("input", , , , );
        ptreedir = debugfs_create_("ptree", , , ); // Find suitable debugfs API


  printk("dbfs_ptree module initialize done\n");

        return 0;
}

static void __exit dbfs_module_exit(void)
{
        // Implement exit module code

  printk("dbfs_ptree module exit\n");
}

module_init(dbfs_module_init);
module_exit(dbfs_module_exit);
```

<- executed when module is inserted

<- file to read input

<- file to write output

<- executed when module is deleted

# With Skeleton Code

```
static ssize_t write_pid_to_input(struct file *fp,
                                  const char __user *user_buffer,
                                  size_t length,
                                  loff_t *position)
{
        pid_t input_pid;

        sscanf(user_buffer, "%u", &input_pid);
        //curr = // Find task_struct using input_pid. Hint: pid_task

        // Tracing process tree from input_pid to init(1) process

        // Make Output Format string: process_command (process_id)

        return length;
}

static const struct file_operations dbfs_fops = {
        .write = write_pid_to_input,
};
```

<- read input_pid

Begin of code <- file write operation

# Hints. Helpful kernel functions & data structures

- struct dentry
- struct task_struct

- struct list_head
- INIT_LIST_HEAD()
  - list_add()
  - list_for_each_entry()

# Kernel programming 101 (utilities)

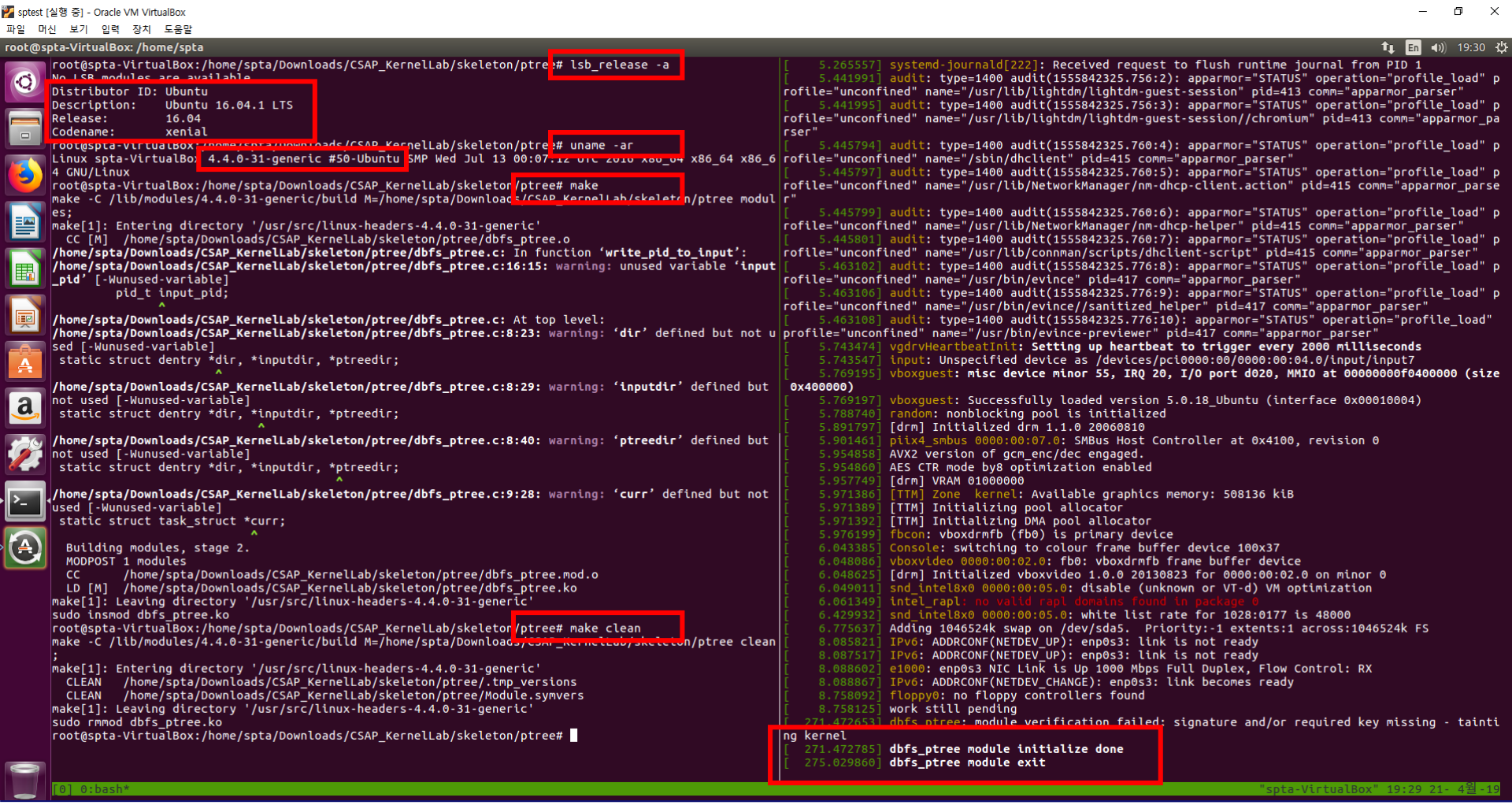
- dmesg
  - dmesg –w
- printk
- insmod / rmmod / lsmod

# Prepare your own development environment

- Oracle VirtualBox + Ubuntu 16.04
- we will check your preparation status
  - capture kernel module load/exit image as follow
  - deadline: 4/29 (Mon) 23:59:59
  - No delay allowed
  - upload your capture image file to eTL
    - filename for part#1 : [학번]_[이름]_kernellab_ptree.jpg (or .png, etc)
    - filename for part #2: [학번]_[이름]_kernellab_paddr.jpg (or .png, etc)
    - you will **lose 5 points** if you missing upload your capture image until 4/29

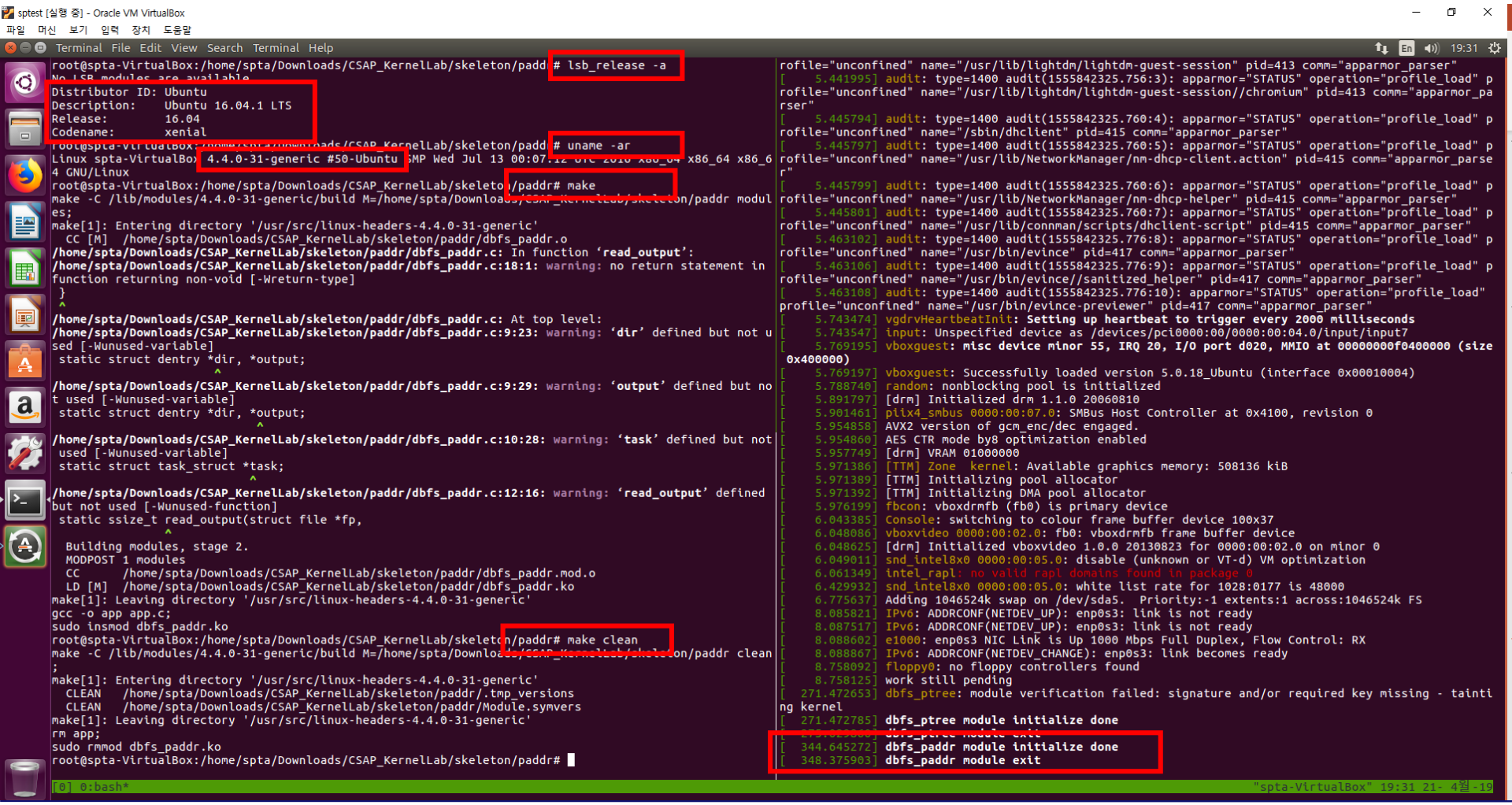
# Prepare your own development environment

- Part #1: ptree module compile preparation



# Prepare your own development environment

- Part #2: paddr module compile preparation



# Demo

# etc

- sudo apt-get install error

```
yschoi@yschoi-VirtualBox:~$ sudo apt-get install tmux
E: Could not get lock /var/lib/dpkg/lock-frontend - open (11: Resource temporarily unavailable)
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), is another process using it?
```

- Fix

```
root@yschoi-VirtualBox:/home/yschoi# sudo rm /var/lib/apt/lists/lock
root@yschoi-VirtualBox:/home/yschoi# sudo rm /var/cache/apt/archives/lock
root@yschoi-VirtualBox:/home/yschoi# sudo rm /var/lib/dpkg/lock
root@yschoi-VirtualBox:/home/yschoi# dpkg --configure -a
```

# References

- Linux Kernel Module Programming Guide
  - http://www.tldp.org/LDP/lkmpg/2.6/html/
- Debugfs APIs
  - https://www.kernel.org/doc/Documentation/filesystems/debugfs.txt
- Makefile Guide
  - https://www.cs.duke.edu/~ola/courses/programming/Makefiles/Makefiles.html