System Programming
Lab #4
Kernel Lab

2023-05-02

sp-tas

Lab Assignment #4 : Kernel Lab

- Download skeleton code & pdf from eTL
  - kernellab-handout.tar, kernellab-handout.pdf
- Hand In – Your Implementation
  - Upload your files eTL
    - 압축파일 양식 : [학번]_kernellab.zip
    - Ex) 2023-12345_kernellab.zip
  - A zip file should include
  - (1) a zip file of your implementation directory (2) report
    - 압축파일 양식 : [학번]_kernellab.zip eg) 2023-12345_kenrnellab.zip
    - Report 양식 : [학번]_kernellab_report.pdf
- Please, READ the Hand-out and Lab material thoroughly!
- Assigned : 5.2(tue)
- Deadline : 5.22(mon), 23:59
- Delay : same as before
- Only use your own VM(**Do not use server machine**)

Lab Assignment #4 : Kernel Lab

- Implementation
  - (part #1) Tracing process tree from process id
  - (part #2) Finding physical address using virtual address

Linux Kernel Programming

- Kernel : “Core of OS”

- Why?
  - Scheduling
  - Cache replacement
  - File system
  - Get kernel information
  - Or other reasons

Linux Kernel Programming

- How to modify kernel?

>> modify source and compile your own kernel

- Good idea, but it is difficult and takes too much time
  - It may take 1-2 hours to compile kernel in your VM.

- Any simpler way?

>> Make kernel modules(Kernel Lab)

Linux Kernel Module

- What is kernel module?
  - A module is pieces of code that can be **loaded and unloaded into the kernel** upon demand.
  - Can **use privileged instructions without system calls**, because a kernel module is loaded and executed within a kernel.

- Need special compile
  - Kernel module is not compiled with general gcc
  - It needs kernel specific compile tools

- Module load / unload command
  - **Load** $ sudo insmod < module_name.ko >
  - **Unload** $ sudo rmmod < module_name >
  - **Module list** $ sudo lsmod

Kernel Module Programming

- Kernel module convention

```
#include <linux/module.h>

MODULE_LICENSE("GPL");

static int __init init_my_module(void)
{
        // Running when this module is inserted to Kernel
}

static void __exit exit_my_module(void)
{
        // Running when this module is removed from Kernel
}

module_init(init_my_module);
module_exit(exit_my_module);
```

Kernel Module Programming

- module_init(func)
  - 드라이버의 초기화 진입점
  - func : 커널 부트나 모듈 삽입 시 실행할 함수
  - __init : 모듈 삽입 시 한 번만 실행하게 하는 매크로

- module_exit(func)
  - 드라이버의 종료 진입점
  - func : 드라이버가 제거될 때 실행할 함수
  - __exit : 종료 시에 실행하게 하는 매크로

* __init, __exit은 특정 섹션에 넣어 놓고 필요 없어지면 메모리에서 해제하는 식으로 동작

* 설명 : <kernel source>/include/linux/init.h

Kernel Module Compile Example

- Makefile for kernel module

```
KDIR = /lib/modules/$(shell uname -r)/build

obj-m := hello.o

all :
	$(MAKE) -C $(KDIR) M=$(PWD) modules;

clean :
	$(MAKE) -C $(KDIR) M=$(PWD) clean;
```

- What are those options?

Kernel Module Compile Options

- $(MAKE) : ‘make’ command
- -C $(KDIR) : move to KDIR directory before running make
- M : ‘M’ is variable. It means where you build module or where your project is
- $(PWD) : current directory
- ojb-m: build the object file as module

```
KDIR = /lib/modules/$(shell uname -r)/build

obj-m := hello.o

all :
	$(MAKE) -C $(KDIR) M=$(PWD) modules;

clean :
	$(MAKE) -C $(KDIR) M=$(PWD) clean;
```

- == KDIR로 가서, ‘make modules’ 명령을 실행하여 현재 디렉토리에서 hello.o를 빌드하여 모듈로 만들어라.

Make interface!

- App needs interface to kernel. Why?
  - To get kernel info
  - To send app info
  - To do HW something(read/write..)
  - Or any other things

- Then how?
  - System call(need to modify kernel itself)
  - **Debug Files System**

Debugfs

- **Debug File System (debugfs)**
  - Special file system(RAM –based)
  - Designed for debugging purposes
  - Simple way for kernel developers
    to make information available to user space
  - Has no rules at all.
    The developers can put any information they want.
  - Supports simple user-to-kernel interfaces in
    Linux Kernel Modules.

Debugfs API

- **APIs in *<linux/debugfs.h>***
- **struct dentry*** debugfs_create_dir(**const char** *name, **struct dentry** *parent)
  - make <name> directory under <parent> directory
  - If NULL in parent, make directory under root directory
- **struct dentry*** debugfs_create_file(**const char** *name, **umode_t** mode,
  struct dentry *parent, **void** *data, **const struct file_operations** *fops)
  - make <name> file under <parent> directory with <mode> permission(e.g. 0644).
  - when you do file operation(open/r/w/close …) use <fops>
  - caller gets <data>. inode.i_privte point this on open() call
- **struct dentry*** debugfs_remove_recursive(**struct dentry** *dentry)
  - recursively remove <dentry> file/dir.

- **dentry? file_operations?**

- *Ref: DebugFS — The Linux Kernel documentation

dentry

- “Directory Entry”
- Connect file name with inode
- Simply, a file or a directory in this lab
- Defined in *<kernel source>/include/linux/dcache.h*

file_operations

- *struct file*
  - **Abstraction** on all types of files (i.e. regular file, dev file)
  - file inode, name, type, size etc.
  - **file_operations**

- mapping file operation(.read) to your function(file_read)

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

- Defined in *<kernel source>/include/linux/fs.h*

file_operations

- C language does not support struct inheritance

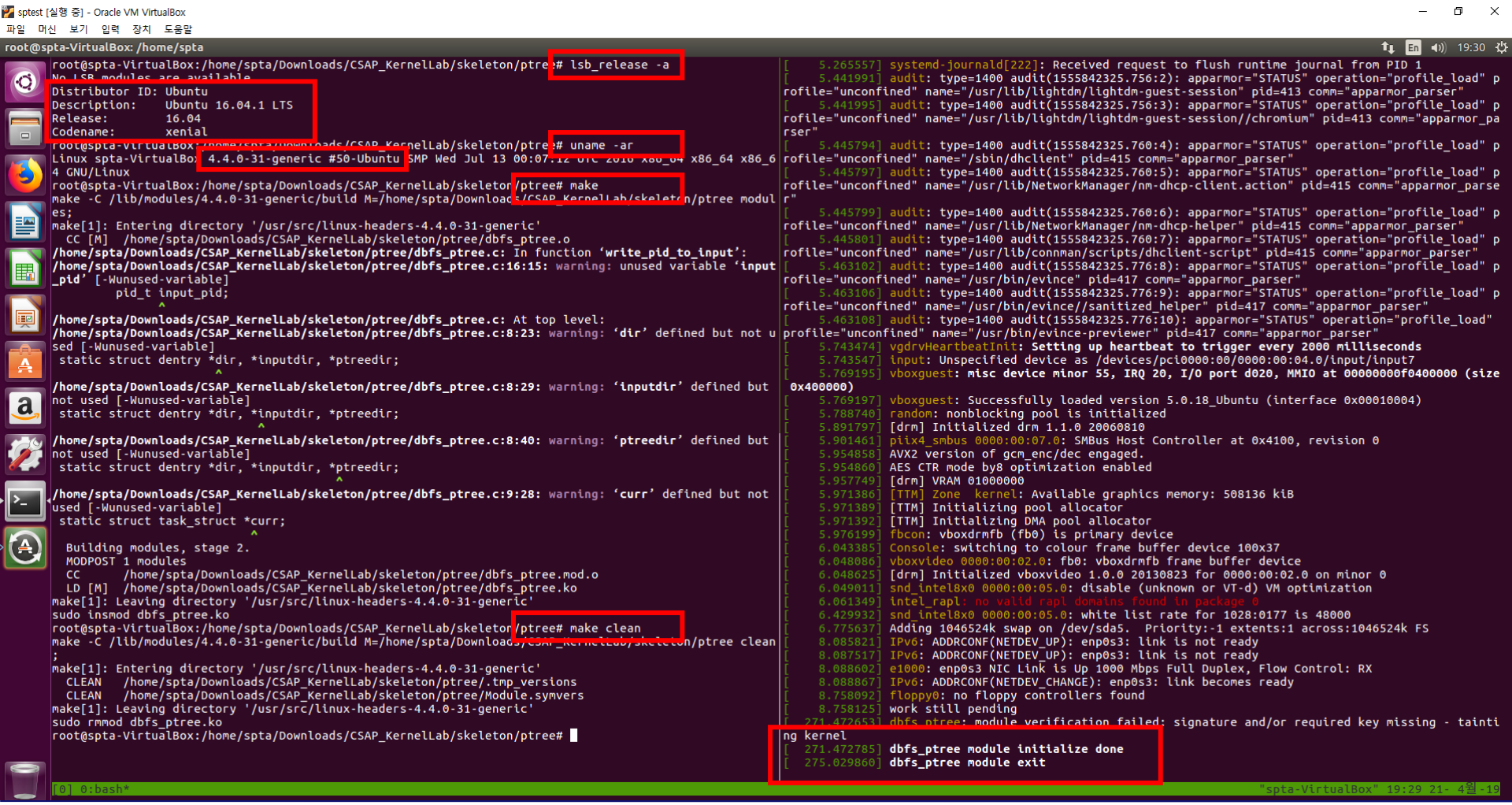
- For using **same interface** (syscall) for different files

- ex) syscall *write()* (*file->fops->write() is called*)
  - ext4_write() for file in ext4
  - f2fs_write() for file in f2fs
  - blkdev_write() for file block device

  *(this is just an example for simple explanation)

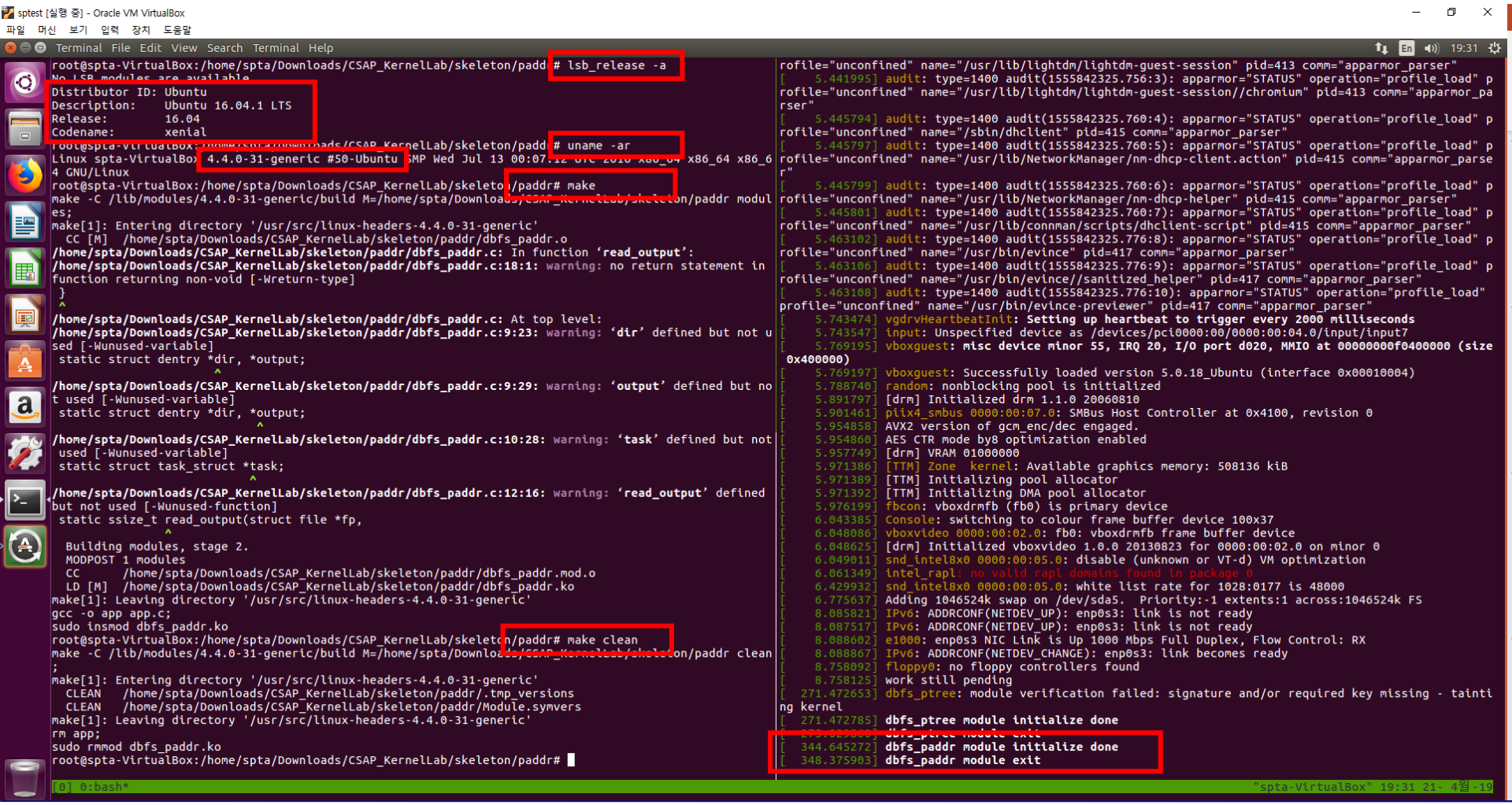
Prepare your own development environment

- Part #1: ptree module compile preparation
- Use ‘dmesg’ command



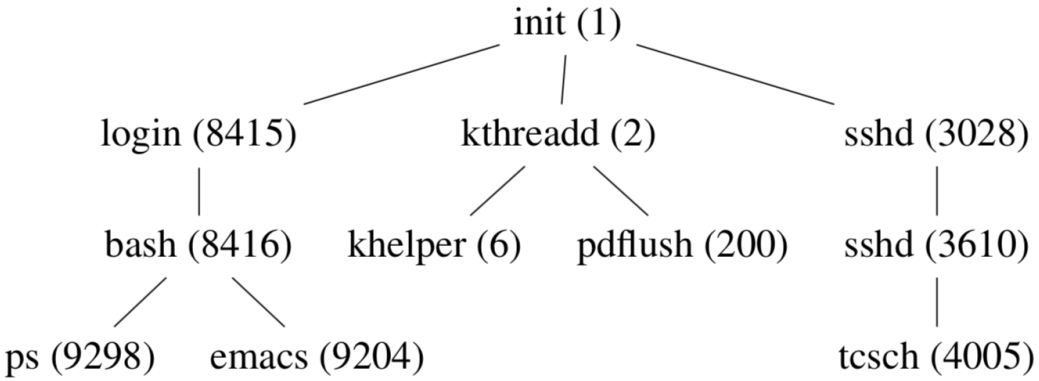
Prepare your own development environment

- Part #2: paddr module compile preparation



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

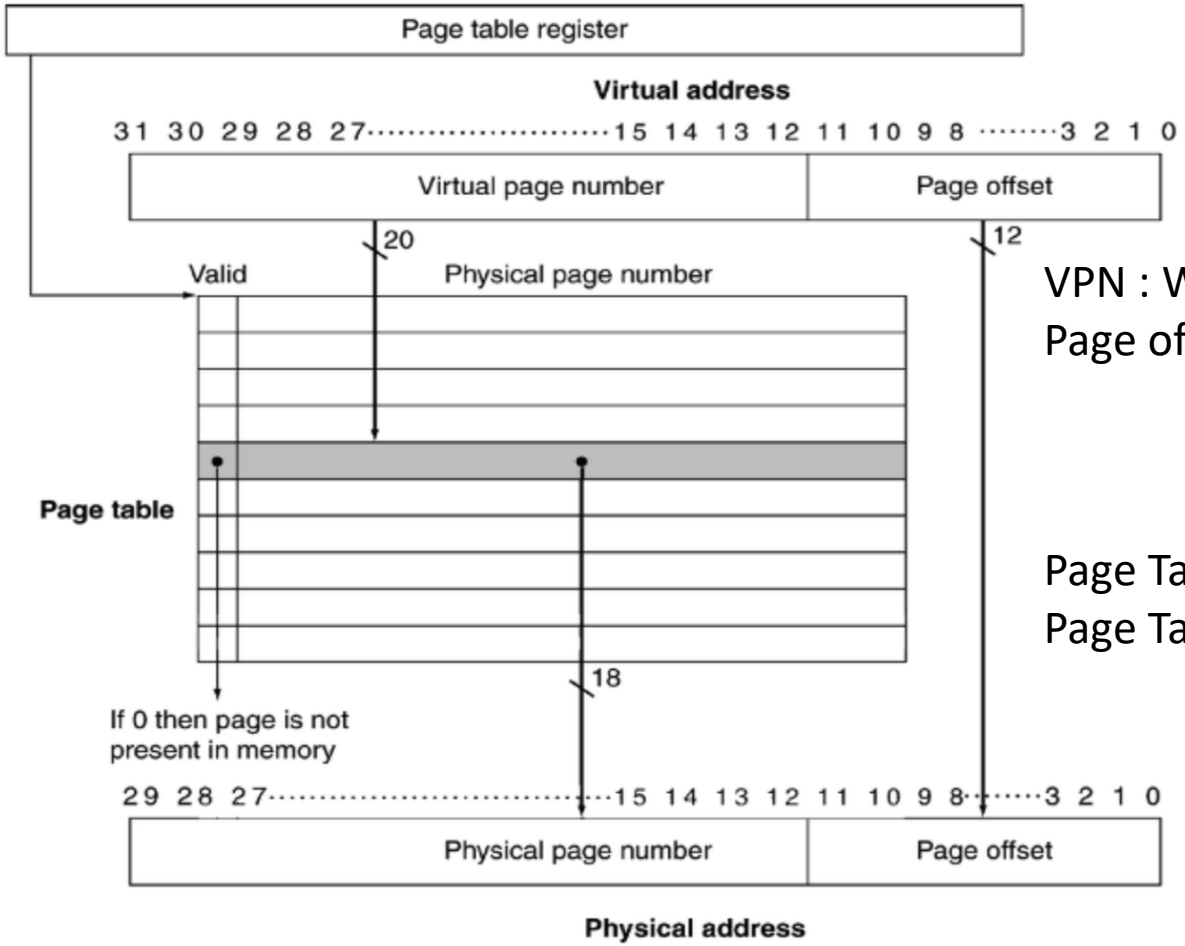
With Skeleton Code

```
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

Part2. Virtual to Physical Address Translation

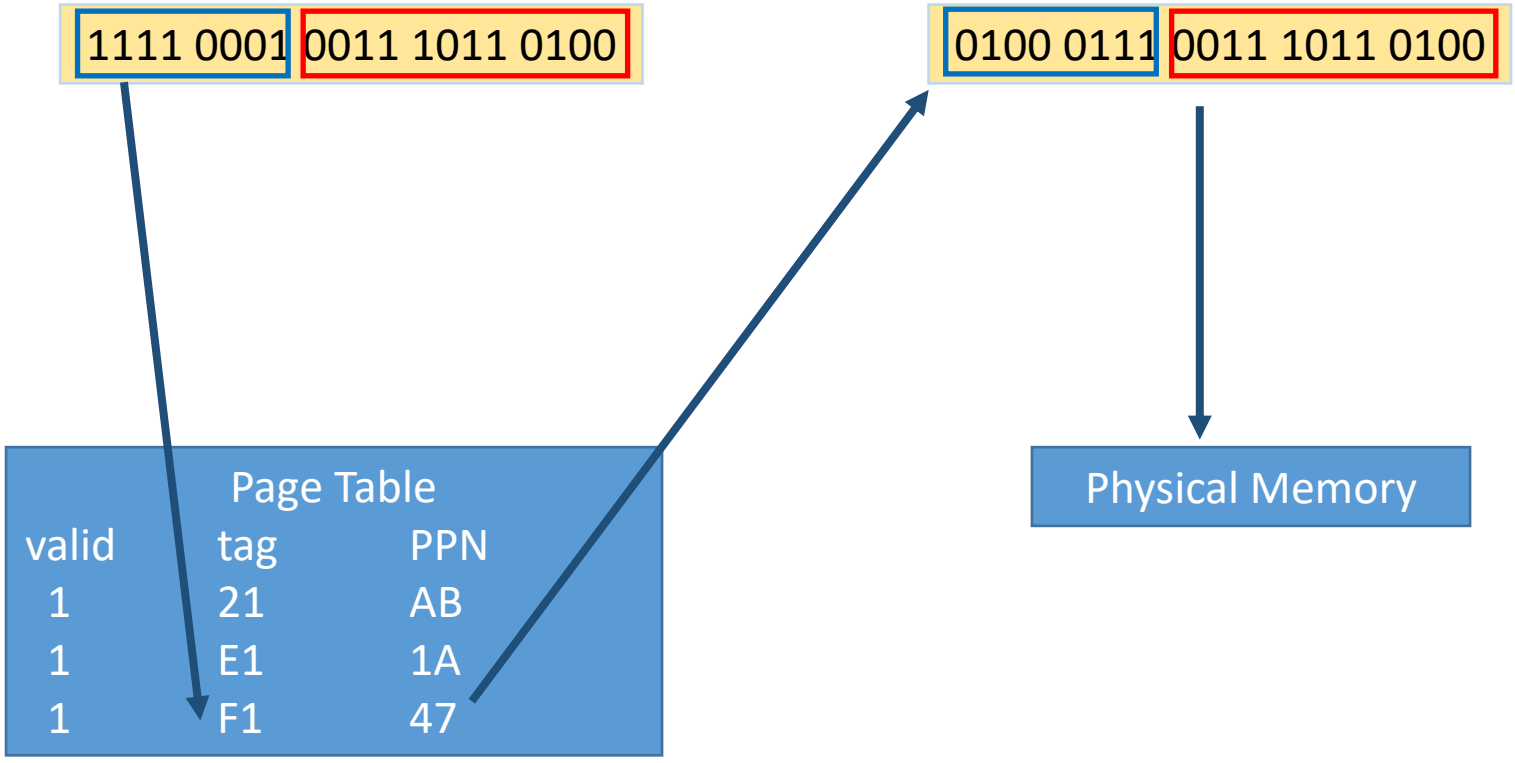


VPN : Which page to look for
Page offset : Where in the page to look for

Page Table : Contains process page info
Page Table Entry : Record of Page Table

Example

- Virtual to Physical Translation
  - 4KB page size, 20bit virtual address
  - VA : 0xF13B4



Multilevel Page Tables

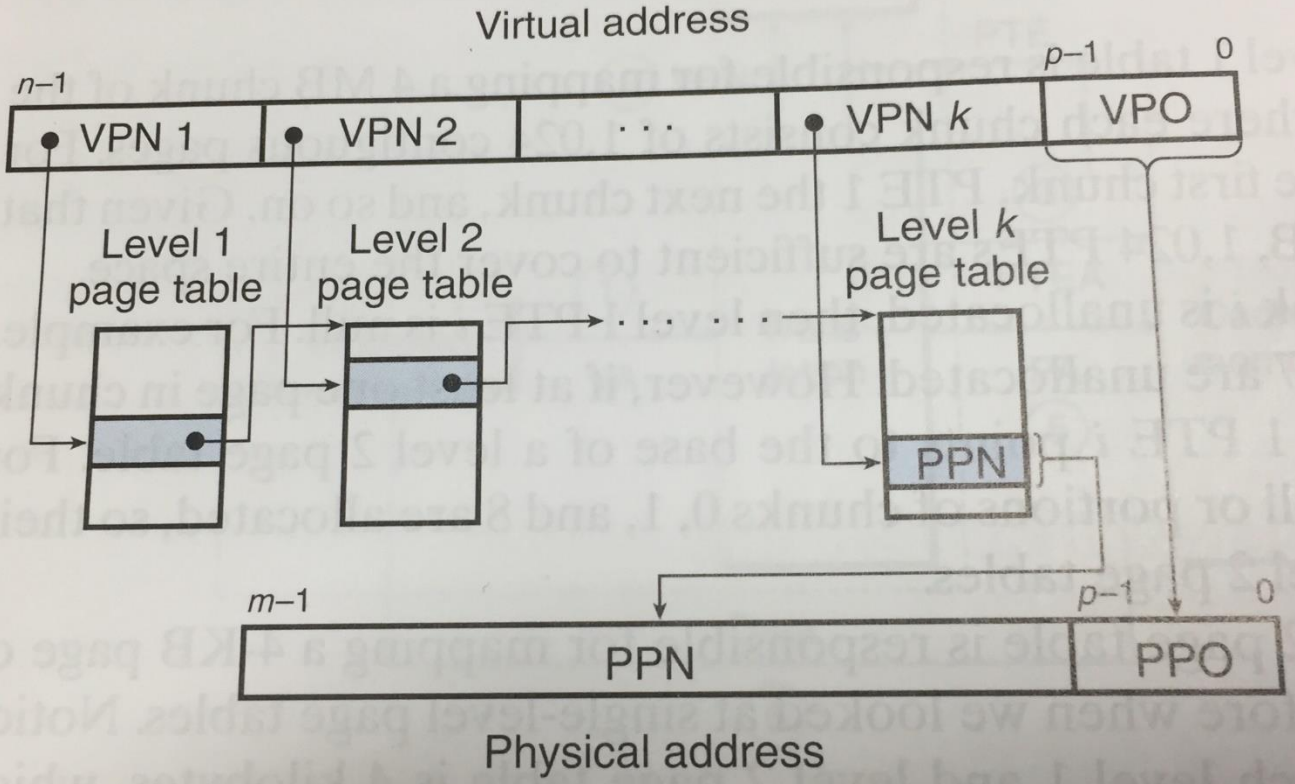


Figure 9.18 Address translation with a k-level page table.

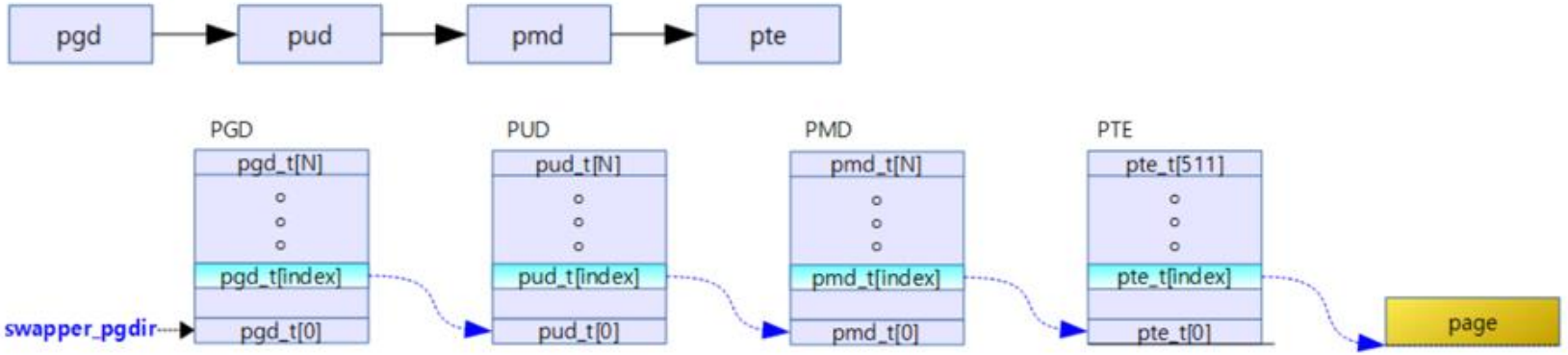
Specification

- In app.c (do not change)
  - makes a virtual address mapped to predefined physical address
  - Compares return value from kernel module with predefined value.

- In your kernel module
  - 1. get pid of app and virtual address
  - 2. find physical address from page table walk
  - 3. returns physical address

Hints

- Page walk API
  - headers in */usr/src/linux/*
  - *<kernel source>/arch/**x86**/include/asm/pgtable.h*
  - *<kernel source>/include/linux/pgtable.h*
- Look for the schemes how virtual address is translated to physical address
- Page walk procedure in linux 4.4.0 (5 level in higher ver.)



Testing your program

- Step
  - 0. sudo su
  - 1. make
  - 2. ./app

```
root@yschoi-VirtualBox:/home/yschoi/kernellab_full/solution/paddr# sudo su
root@yschoi-VirtualBox:/home/yschoi/kernellab_full/solution/paddr# make
make -C /lib/modules/4.15.0-47-generic/build M=/home/yschoi/kernellab_full/solut
ion/paddr modules;
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-47-generic'
  CC [M]  /home/yschoi/kernellab_full/solution/paddr/dbfs_paddr.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/yschoi/kernellab_full/solution/paddr/dbfs_paddr.mod.o
  LD [M]  /home/yschoi/kernellab_full/solution/paddr/dbfs_paddr.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-47-generic'
gcc -o app app.c;
sudo insmod dbfs_paddr.ko
root@yschoi-VirtualBox:/home/yschoi/kernellab_full/solution/paddr# ./app
vaddr { 7ffbfe301000 } paddr { 0 }
vaddr { 7ffbfe301000 } paddr { 0 }
vaddr { 7ffbfe301000 } paddr { 234512000 }
[TEST CASE]    PASS
root@yschoi-VirtualBox:/home/yschoi/kernellab_full/solution/paddr#
```

Hints. Helpful kernel functions & data structures

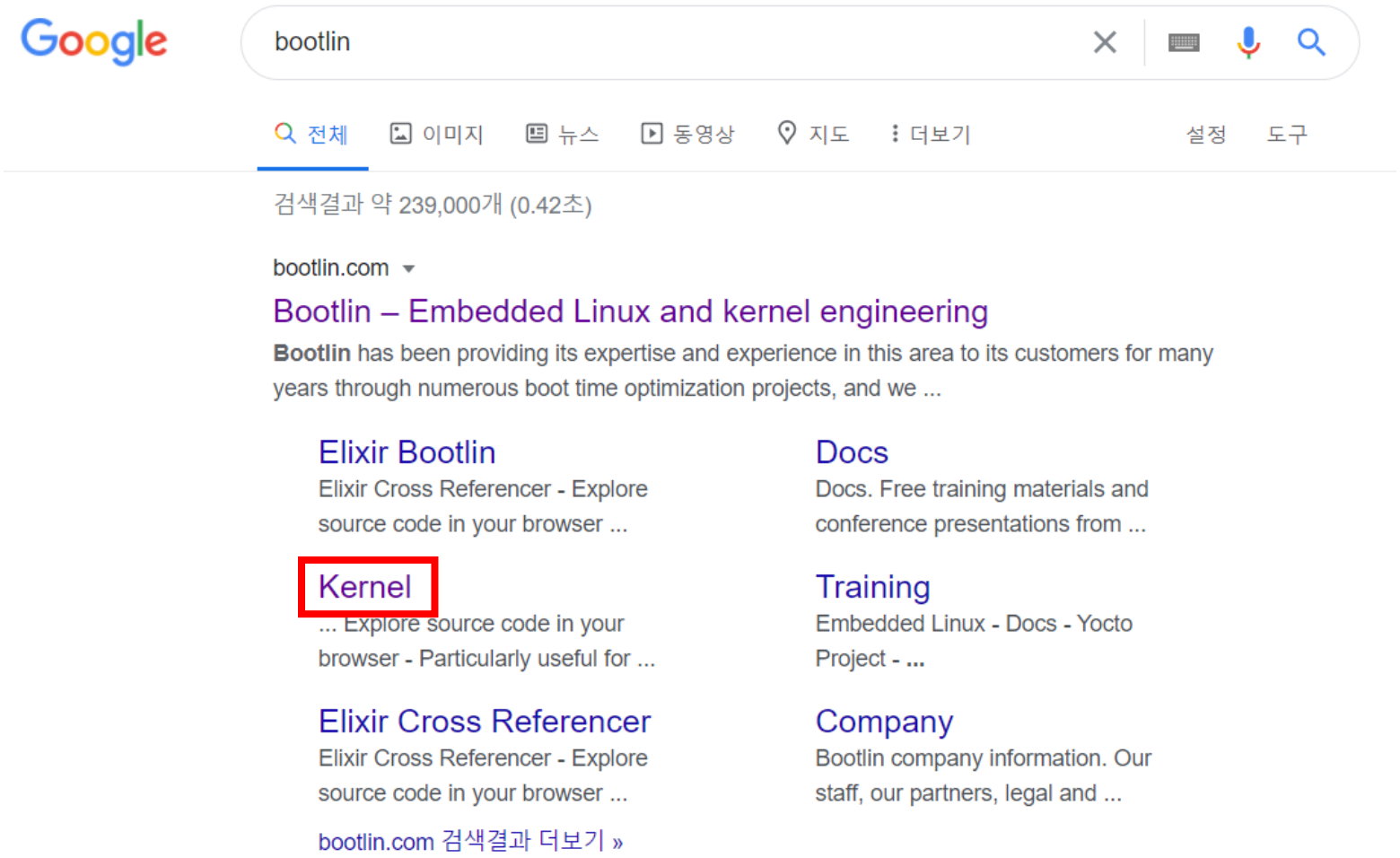
- struct dentry
- struct task_struct

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

- copy_from_user()
- copy_to_user()
- debugfs_remove_recursive()

Bootlin

- Bootlin(kernel - Linux source code (v5.10.11) – Bootlin)



Kernel Debugging

- dmesg
  - dmesg –w
- printk

(extra) Tools for kernel programming

- Some useful tools in kernel programming
  - tmux
  - ctags
  - cscope

tmux (terminal multiplexer)

- Installation
  - sudo apt install tmux

- Basic Commands

Command	Description
tmux	start tmux
tmux new -s <name>	start tmux with <name>
tmux ls	shows the list of sessions
tmux a #	attach the detached-session
tmux a -t <name>	attach the detached-session to <name>
tmux kill-session –t <name>	kill the session <name>
tmux kill-server	kill the tmux server

Split windows with tmux

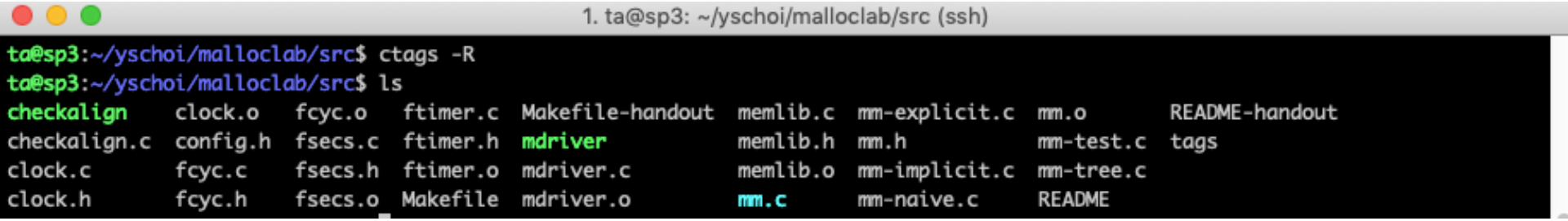
- **1. ctrl + b (or a)** to type tmux command
- 2. Split vertically : %(shift 5)
- 3.Split horizontally : ″(shift ‘)
- https://tmuxguide.readthedocs.io/en/latest/tmux/tmux.html

Ctags

- What is Ctags?
  - A Tool that makes it easy to navigate big source code projects.
- Ctags generates **database** of tag file
  - for global variables, functions, macros, etc
  - to point where they are declared & defined
- Installation
  - sudo apt-get install ctags (or exuberant-ctags)
- Check
  - ctags --version
- Help
  - Ctags --help

Ctags – how to make tags file

- Steps
  - 1. go to root directory of codes you want to navigate.
    - cd /(where your root directory of code is)
  - 2. generate tags file
    - type ctags -R (recursive)
    - or ctags file1, file2, …
  - 3. Check tags file
    - ls



  - 4. Remove tags file
    - rm tags

Ctags – how to use

- Case 1. In code file
  - 1. place cursor on the keyword you want to locate where it is defined
  - 2. type ctrl +]

```
227     if ((bp = mem_sbrk(size)) == (void *)-1)
228         return NULL;
229
230     /* Initialize free block header/footer and the epilogue header */
231     PUT(HDRP(bp), PACK(size, 0));         /* free block header */
232     PUT(FTRP(bp), PACK(size, 0));         /* free block footer */
233     PUT(HDRP(NEXT_BLKP(bp)), PACK(0, 1)); /* new epilogue header */
234
235     /* Coalesce if the previous block was free */
236     return coalesce(bp);
237 }
238 /* $end mmextendheap */
239
240 /*
241  * place - Place block of asize bytes at start of free block bp
242  *         and split if remainder would be at least minimum block size
243  */
244 /* $begin mmplace */
245 /* $begin mmplace-proto */
246 static void place(void *bp, size_t asize)
 227:17 [62%]
"mm.c" 369L, 9640C
```

```
58 void *mem_sbrk(int incr)
59 {
60     char *old_brk = mem_brk;
61
62     if ( (incr < 0) || ((mem_brk + incr) > mem_max_addr)) {
63     errno = ENOMEM;
64     fprintf(stderr, "ERROR: mem_sbrk failed. Ran out of memory...\n");
65     return (void *)-1;
66     }
67     mem_brk += incr;
68     return (void *)old_brk;
69 }
70
71 /*
72  * mem_heap_lo - return address of the first heap byte
73  */
74 void *mem_heap_lo()
75 {
76     return (void *)mem_start_brk;
77 }
 58:1 [61%]
'memlib.c" 101L, 2270C
```

  - 3. type ctrl + t to go back

Ctags – how to use

- Case 2. In tags file
  - 1. vi tags
  - 2. type :tj [tag name] to find

```
 35 FTRP    mm-implicit.c   71;"    d   file:
 36 FTRP    mm.c    71;"    d   file:
 37 GET mm-implicit.c   62;"    d   file:
 38 GET mm.c    62;"    d   file:
 1:1 [Top]
:tj mem_sbrk
```

  - 3. type :po to comeback

```
58 void *mem_sbrk(int incr)
59 {
60     char *old_brk = mem_brk;
61
62     if ( (incr < 0) || ((mem_brk + incr) > mem_max_addr)) {
63     errno = ENOMEM;
64     fprintf(stderr, "ERROR: mem_sbrk failed. Ran out of memory...\n");
65     return (void *)-1;
66     }
67     mem_brk += incr;
68     return (void *)old_brk;
69 }
70
71 /*
72  * mem_heap_lo - return address of the first heap byte
73  */
74 void *mem_heap_lo()
75 {
76     return (void *)mem_start_brk;
77 }
58:1 [61%]
:po
```

Ctags – how to use

- Case 3. Vim?
  - 구글 선생님께 여쭤볼 것.

Using Ctags in Linux kernel code

```
!_TAG_FILE_FORMAT       2       /extended format; --format=1 will
not append ;" to lines/
!_TAG_FILE_SORTED       1       /0=unsorted, 1=sorted, 2=foldcase/
!_TAG_PROGRAM_AUTHOR    Darren Hiebert  /dhiebert@users.sourceforg
e.net/
!_TAG_PROGRAM_NAME      Exuberant Ctags //
!_TAG_PROGRAM_URL       http://ctags.sourceforge.net    /official
site/
!_TAG_PROGRAM_VERSION   5.9~svn20110310 //
$0      arch/mips/include/asm/mach-cavium-octeon/kernel-entry-init
.h    /^      dins    v0, $0, 0, 6$/;"        v
$0      arch/mips/include/asm/mach-cavium-octeon/kernel-entry-init
.h    /^      sd      $0, -32768(v0)$/;"      v
$0      arch/mips/include/asm/stackframe.h      /^              LO
NG_S  $0, PT_R0(sp)$/;"       v
$0      arch/mips/include/asm/stackframe.h      /^              no
r     v1, $0, v1$/;"  v
$1      arch/mips/include/asm/asmmacro.h        /^      _cfcmsa $1
, MSA_CSR$/;" v
$1      arch/mips/include/asm/asmmacro.h        /^      sw      $1
, THREAD_MSA_CSR(\\thread)$/;"        v
$10     arch/mips/include/asm/stackframe.h      /^              cf
i_ld  $10, PT_R10, \\docfi$/;"        v
$10     arch/mips/include/asm/stackframe.h      /^              cf
i_st  $10, PT_R10, \\docfi$/;"        v
$11     arch/mips/include/asm/stackframe.h      /^              cf
i_ld  $11, PT_R11, \\docfi$/;"        v
$11     arch/mips/include/asm/stackframe.h      /^              cf
i_st  $11, PT_R11, \\docfi$/;"        v
$12     arch/mips/include/asm/stackframe.h      /^              cf
i_ld  $12, PT_R12, \\docfi$/;"        v
$12     arch/mips/include/asm/stackframe.h      /^              cf
i_st  $12, PT_R12, \\docfi$/;"        v
$13     arch/mips/include/asm/stackframe.h      /^              cf
i_ld  $13, PT_R13, \\docfi$/;"        v
$13     arch/mips/include/asm/stackframe.h      /^              cf
i_st  $13, PT_R13, \\docfi$/;"        v
$14     arch/mips/include/asm/stackframe.h      /^              cf
i_ld  $14, PT_R14, \\docfi$/;"        v
$14     arch/mips/include/asm/stackframe.h      /^              cf
i_st  $14, PT_R14, \\docfi$/;"        v
:tj task_struct
```

```
#include <linux/debugfs.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/uaccess.h>
#include <linux/list.h>
#include <linux/slab.h>

#define COMM_STR_SIZE   32
#define BLOB_SIZE       128

MODULE_LICENSE("GPL");

static struct dentry *dir, *inputdir, *ptreedir;
static struct task_struct *curr;

struct task_list {
        struct list_head list;
        pid_t pid;
        char comm[COMM_STR_SIZE];
};

struct task_list t_list;

struct debugfs_blob_wrapper p_tree;
char blob[BLOB_SIZE];

static void add_task(struct task_struct *task)
{
        struct task_list *node;

        node = (struct task_list*)kmalloc(sizeof(struct task_lis
t), GFP_KERNEL);
        node->pid = task->pid;
        strncpy(node->comm, task->comm, COMM_STR_SIZE);

        list_add((struct list_head*)node, &(t_list.list));
}

static void build_blob(void)
{
        char buffer[BLOB_SIZE] = "";
"dbfs_ptree.c" 104 lines, 2481 characters
```

Using Ctags in Linux kernel code

```
                u8                      exp_need_qs;

                /* Otherwise the compiler can store garbage here:
*/
                u8                      pad;
        } b; /* Bits. */
        u32 s; /* Set of bits. */
};

enum perf_event_task_context {
        perf_invalid_context = -1,
        perf_hw_context = 0,
        perf_sw_context,
        perf_nr_task_contexts,
};

struct wake_q_node {
        struct wake_q_node *next;
};

struct task_struct {
#ifdef CONFIG_THREAD_INFO_IN_TASK
        /*
         * For reasons of header soup (see current_thread_info()),
 this
         * must be the first element of task_struct.
         */
        struct thread_info              thread_info;
#endif
        /* -1 unrunnable, 0 runnable, >0 stopped: */
        volatile long                   state;

        /*
         * This begins the randomizable portion of task_struct. On
ly
         * scheduling-critical items should be added above here.
         */
        randomized_struct_fields_start

        void                            *stack;
        atomic_t                        usage;
:po
```

```
#include <linux/debugfs.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/uaccess.h>
#include <linux/list.h>
#include <linux/slab.h>

#define COMM_STR_SIZE   32
#define BLOB_SIZE       128

MODULE_LICENSE("GPL");

static struct dentry *dir, *inputdir, *ptreedir;
static struct task_struct *curr;

struct task_list {
        struct list_head list;
        pid_t pid;
        char comm[COMM_STR_SIZE];
};

struct task_list t_list;

struct debugfs_blob_wrapper p_tree;
char blob[BLOB_SIZE];

static void add_task(struct task_struct *task)
{
        struct task_list *node;

        node = (struct task_list*)kmalloc(sizeof(struct task_lis
t), GFP_KERNEL);
        node->pid = task->pid;
        strncpy(node->comm, task->comm, COMM_STR_SIZE);

        list_add((struct list_head*)node, &(t_list.list));
}

static void build_blob(void)
{
        char buffer[BLOB_SIZE] = "";
"dbfs_ptree.c" 104 lines, 2481 characters
```

Cscope

- A tool to navigate in big source code.

- Diff with Ctags?
  - Able to locate functions where they are called too.
- Installation
  - sudo apt-get install cscope
- Check
  - cscope --version

```
ta@sp3:~/yschoi/malloclab/src$ cscope --version
cscope: version 15.8b
ta@sp3:~/yschoi/malloclab/src$
```

- Help
  - cscope --help

Cscope – how to make cscope database file

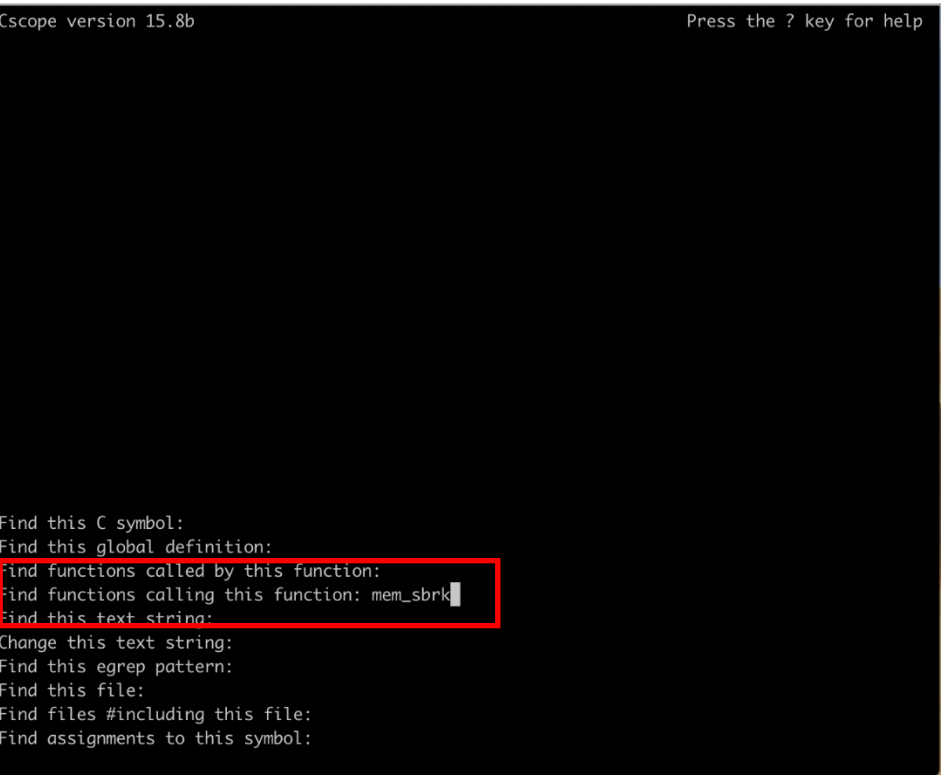
- Steps
  - 1. go to root directory of codes you want to navigate.
    - cd /(where your root directory of code is)
  - 2. generate cscope database file
    - find ./ -name '*[cCsShH]]' > file_list
    - cscope -i file_list
  - 3. Check cscope.out file
    - ls

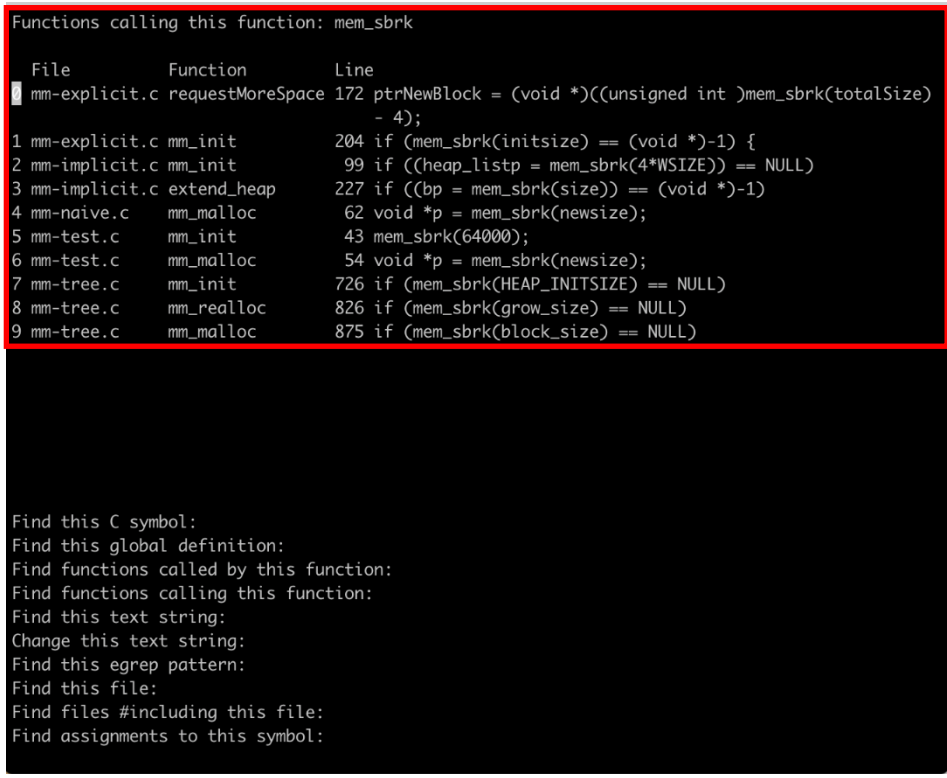
```
ta@sp3:~/yschoi/malloclab/src$ find ./ -name '*[cCsShH]' > file_list
ta@sp3:~/yschoi/malloclab/src$ cscope -i file_list
ta@sp3:~/yschoi/malloclab/src$ ls
checkalign    cscope.out  fsecs.h   Makefile-handout  memlib.o       mm.o
checkalign.c  fcyc.c      fsecs.o   mdriver           mm.c           mm-test.c
clock.c       fcyc.h      ftimer.c  mdriver.c         mm-explicit.c  mm-tree.c
clock.h       fcyc.o      ftimer.h  mdriver.o         mm.h           README
clock.o       file_list   ftimer.o  memlib.c          mm-implicit.c  README-handout
config.h      fsecs.c     Makefile  memlib.h          mm-naive.c
```

  - 4. Remove tags file
    - rm cscope.out file_list

Cscope – how to use

- 1. type cscope –d to execute
- 2. type ctrl+d to break out





FAQ part1

- (1) init process의 이름이 systemd
  - 상관 없습니다.
- (2) Corner cases?
  - 채점 기준: 다음 명령이 올바르게 작동 하는지 확인, 모든 pid는 valid
    - # make
    - # cd /sys/kernel/debug/ptree
    - # echo pid1 >> input
    - # cat ptree
    - # echo pid2 >> input
    - # cat ptree
    - ...
    - # echo pidn >> input
    - # cat ptree
    - # cd -
    - # make clean

FAQ part2

- 1. mmap fails in app
  - Try using different PADDRs
  - PADDR must be 4KB-aligned (ex. 0x1234**000**)
  - Please report the value used to TA
- 2. Accessing user buffer data in kernel
  - Recommended: use copy_from_user()
- 3. Flushing kernel data to user
  - Recommended: use copy_to_user()

Next Time

- Questions
  - eTL Q&A Board

- Please, read the handout & start early!

- Next time (May. 16th)
  - **Kernel LAB Q&A session in this class room**

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

References

- Tmux guide
  - https://tmuxguide.readthedocs.io/en/latest/tmux/tmux.html
- Ctags
  - https://bowbowbow.tistory.com/15
- Cscope
  - https://harryp.tistory.com/131
- Address Translation, Multilevel Page table
  - P.849~855, R. E. Briant, D. R. O'Hallaron, Computer Systems, A programmer's perspective 3rd edition