System Programming
Lab Session #3

Kernel Lab

2019/10/01
sysprog@csap.snu.ac.kr

Computer Systems and Platforms Laboratory
School of Computer Science and Engineering
Seoul National University

Goals

- Understand the art of the Linux Kernel
    - Learn how to program a kernel module in Linux
    - Understand the hierarchical structure of page tables and address translation

1. Introduction

What is a Kernel Module

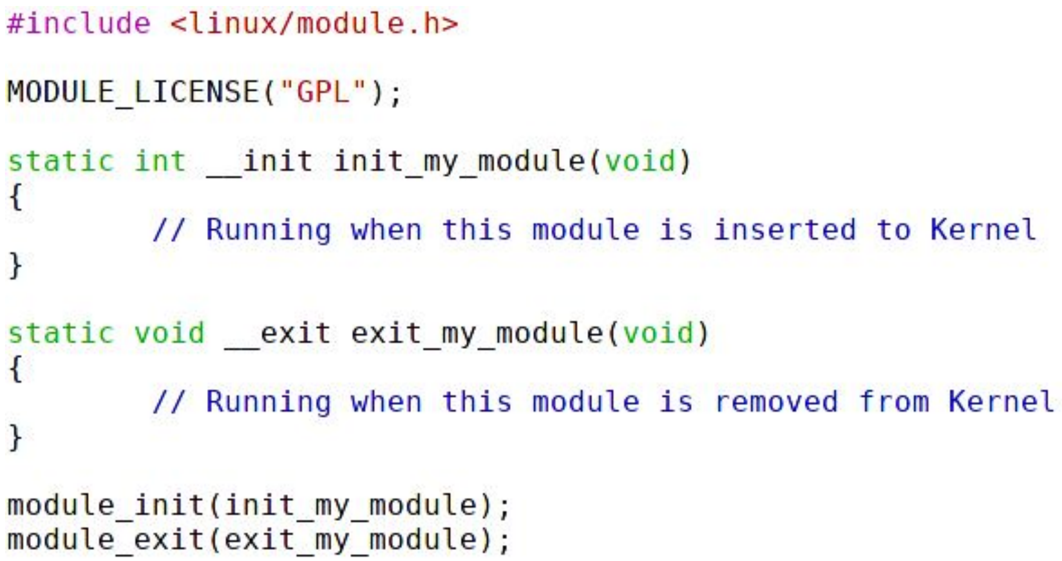
- A module is a piece of code that can be loaded and unloaded into the kernel upon demand.

- Can use privileged instructions without system calls, because a kernel module is loaded and executed within the kernel.

- Module load / unload commands in Linux
    - Load: **root #** insmod < module_name.ko >
    - Unload: **root #** rmmod <module_name>
    - Module list **devel $** lsmod

Linux Kernel Module Programming

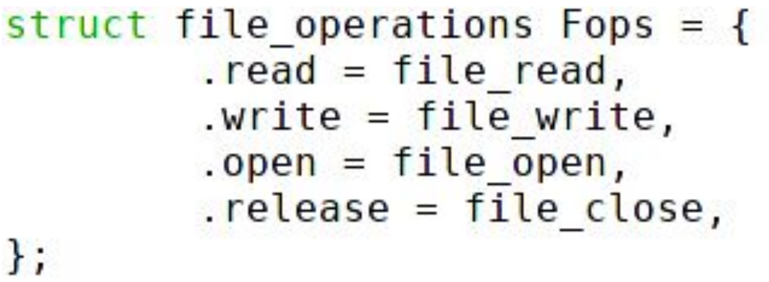
- There are some conventions when programming a kernel module in Linux
- Initialize and Exit module
  - module_init(): Called when the module is inserted
  - module_exit(): Called when the module is removed



```c
#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);
```

Linux Kernel Module Programming

- File operations
  - File operations are used to communicate with files in Device Driver and Debug File System
  - Function pointer structure for the file



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

Linux Kernel Module Programming

- **Debug File System (debugfs)** is a special file system available in the Linux kernel. Debugfs is a simple-to-use RAM-based file system specially designed for debugging purposes. It exists as a simple way for the kernel developers to make information quickly and easily available to user space.

- **Debugfs** has no rules at all. Developers can put any type of information that they want.

- **Debugfs** also supports simple user-to-kernel interfaces in Linux kernel modules. Developers can access Linux kernel information easily using debugfs.

Debugfs APIs

- Code using debugfs must include <linux/debugfs.h>
- Debugfs APIs

  -
    ```
    struct dentry* debugfs_create_dir(const char *name, struct dentry *parent)
    ```

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

  -
    ```
    struct dentry* debugfs_remove_recursive(struct dentry *dentry)
    ```

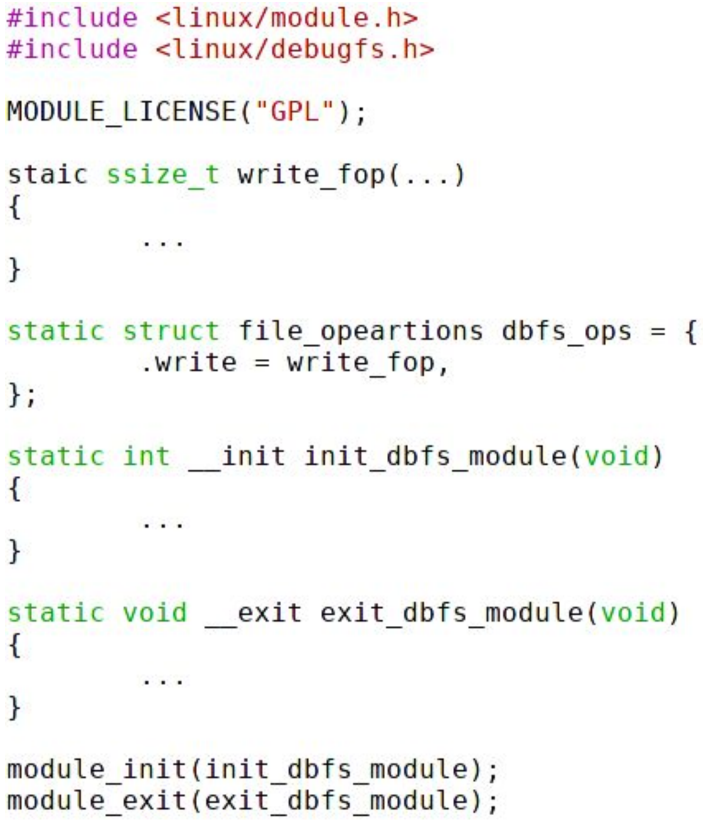
Debugfs APIs

- Make a source file *dbfs.c*
- Make a build script *Makefile*
- Build and Insert your module

```
devel $  make
devel $  sudo insmod dbfs.ko
```

- Check files in the debugfs

```
root #  cd /sys/kernel/debug/dir
root #  ls
```

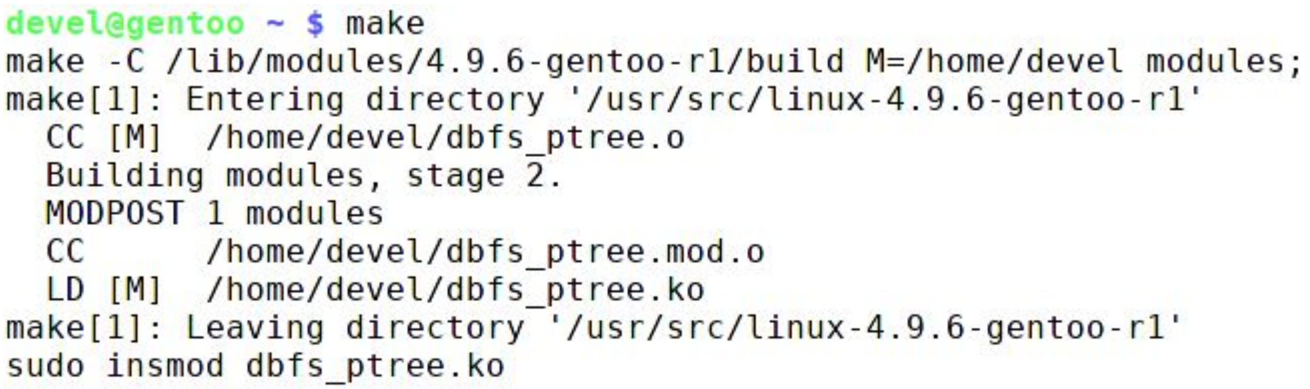


```c
#include <linux/module.h>
#include <linux/debugfs.h>

MODULE_LICENSE("GPL");

staic ssize_t write_fop(...)
{
        ...
}

static struct file_opeartions dbfs_ops = {
        .write = write_fop,
};

static int __init init_dbfs_module(void)
{
        ...
}

static void __exit exit_dbfs_module(void)
{
        ...
}

module_init(init_dbfs_module);
module_exit(exit_dbfs_module);
```

HowTo: Makefile

- GNU *make* is one many build systems that keep track of how to build your program from the sources
- Makefile is the file that contains the instructions for *make*.

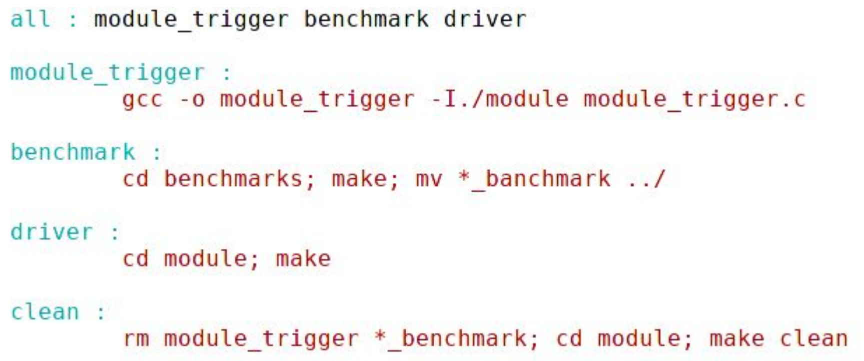
example) **devel $** make



```
devel@gentoo ~ $ make
make -C /lib/modules/4.9.6-gentoo-r1/build M=/home/devel modules;
make[1]: Entering directory '/usr/src/linux-4.9.6-gentoo-r1'
  CC [M]  /home/devel/dbfs_ptree.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/devel/dbfs_ptree.mod.o
  LD [M]  /home/devel/dbfs_ptree.ko
make[1]: Leaving directory '/usr/src/linux-4.9.6-gentoo-r1'
sudo insmod dbfs_ptree.ko
```

HowTo: Makefile

- Simple Example

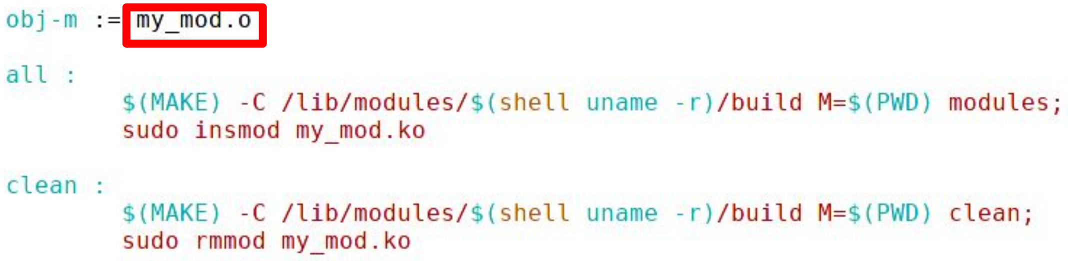


```
all : module_trigger benchmark driver

module_trigger :
        gcc -o module_trigger -I./module module_trigger.c

benchmark :
        cd benchmarks; make; mv *_banchmark ../

driver :
        cd module; make

clean :
        rm module_trigger *_benchmark; cd module; make clean
```

- all : …
    - Defines targets. Each targets must be defined in the followings.
    - When you execute *make* without a target, the first defined target is built.
- clean :
    - Defines removal of all built object files.
    - You can call it as ‘make clean’

HowTo: Makefile for Kernel Modules

- Module Compile Example

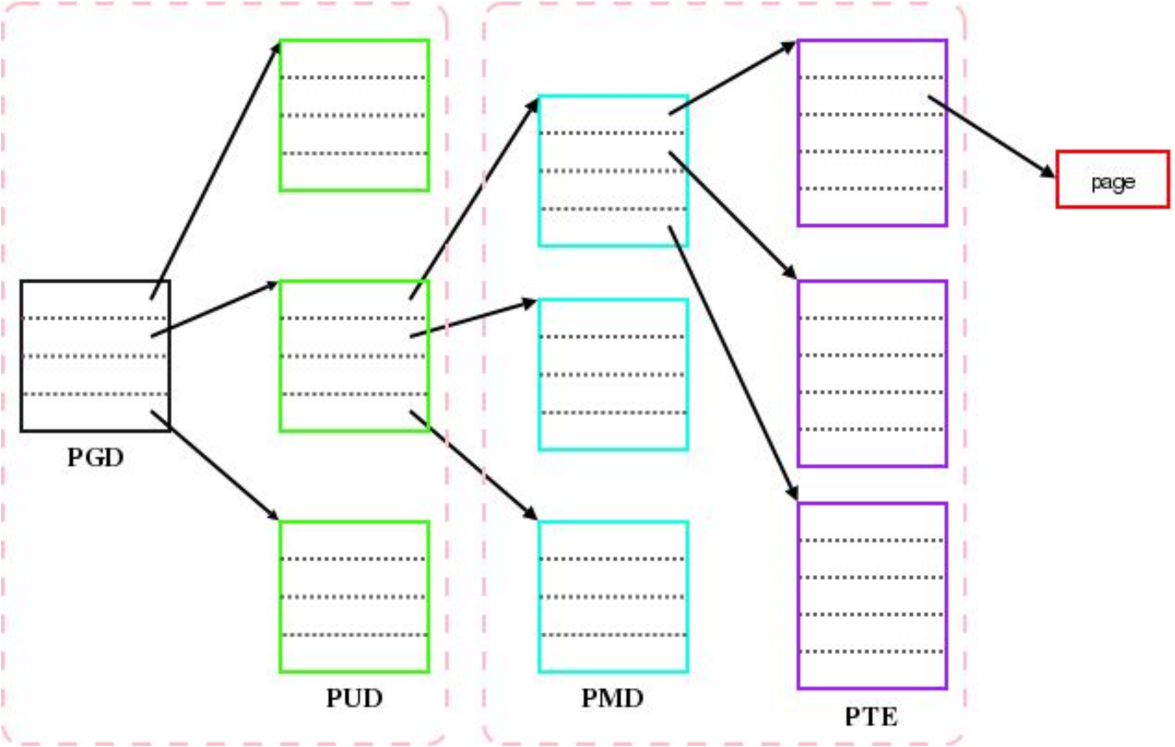


```
obj-m := my_mod.o

all :
	$(MAKE) -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules;
	sudo insmod my_mod.ko

clean :
	$(MAKE) -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean;
	sudo rmmod my_mod.ko
```

- Kernel module is not compiled with general ‘gcc’. It needs to be built using kernel-specific compile tools.
- The example shows a simple makefile for a kernel module build.
- You should change the module name in the red box.

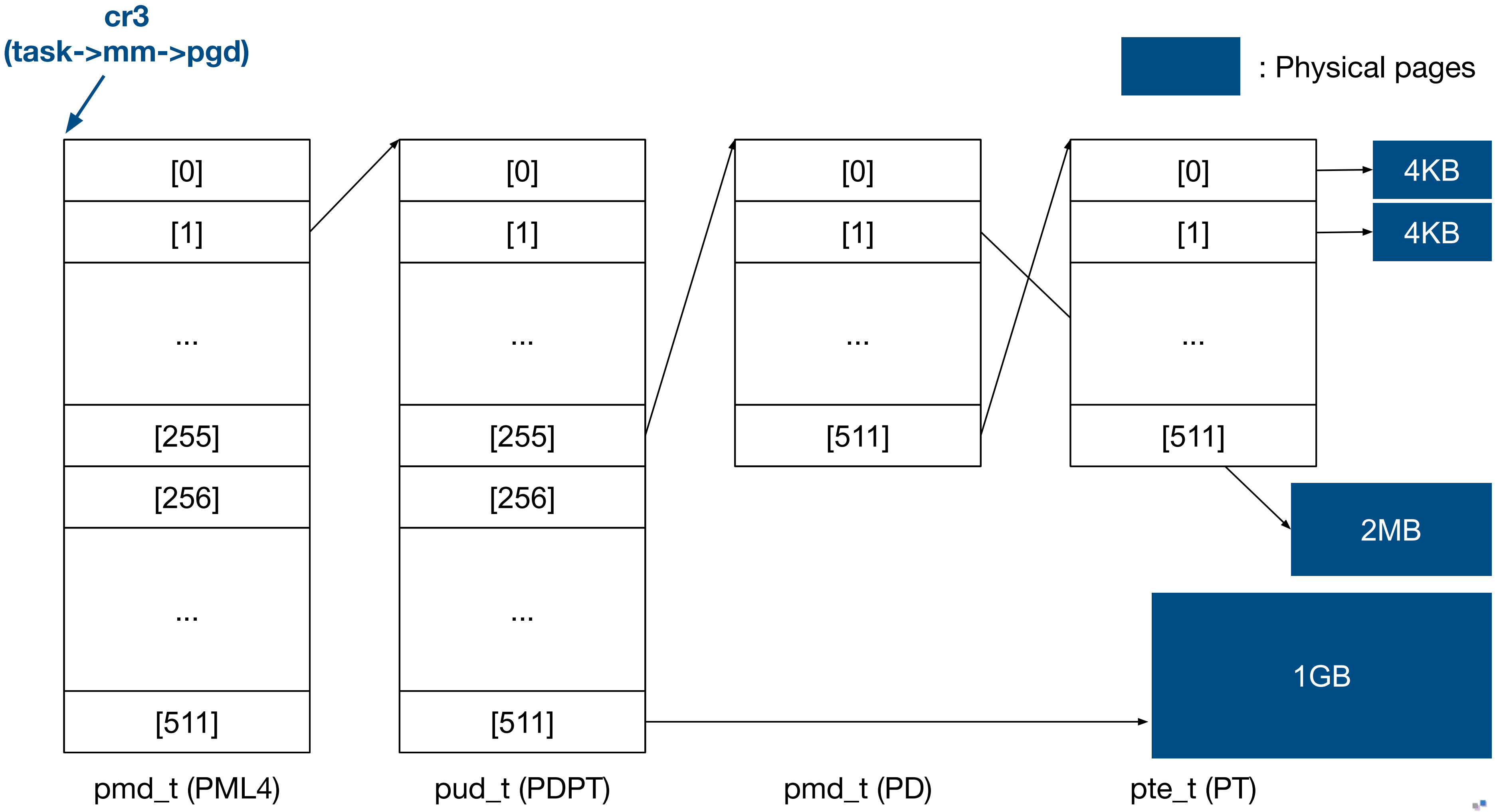
2. Background

Paging on a 64-bit machine



Paging on a 64-bit machine

- Typical page size is 4 KB, however there are some pages with 1 GB/2 MB size
- Page tables resident in 4 KB pages (4 kB / 8 B = 512 entries per table)



Paging on a 64-bit machine

- 512 entries = 2^9 entries
    -

47 ... 39	38 ... 30	29 ... 21	20 ... 12	11 ... 0
pgd index	pud index	pmd index	pte index	page offset

        - Linear address space: 2^{48} Bytes = 256 TB
        - Although virtual addresses are 64 bits wide, current implementations do not allow the entire virtual address space of 2^{64} bytes (16 EB) to be used.

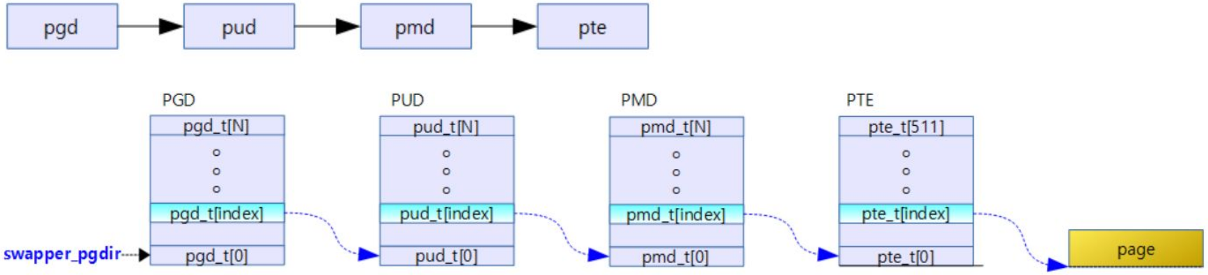
- 0x00000000 00000000 ~ 0x00007FFF FFFFFFFF : Lower half (user space, 128 TB)
    - 0000000000000000 **000000000** 000000000 000000000 000000000 000000000000
    - 0000000000000000 **011111111** 111111111 111111111 111111111 111111111111

- 0xFFFF8000 00000000 ~ 0xFFFFFFFF FFFFFFFF : Higher half (kernel space, 128 TB)
    - 1111111111111111 **100000000** 000000000 000000000 000000000 000000000000
    - 1111111111111111 **111111111** 111111111 111111111 111111111 111111111111

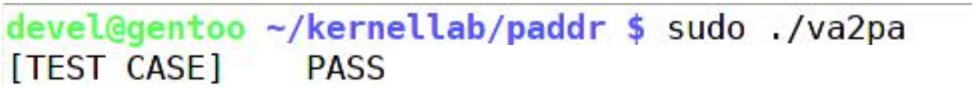
3. Part A

Part A : Find Physical Address

- Your task: print the physical address for a given virtual address in the current process
- Use *getpid( )* to get the PID of the current process
- For verification purposes, we use *mmap()* to fix a pointer to a given virtual address
- Include page walk procedure in your program



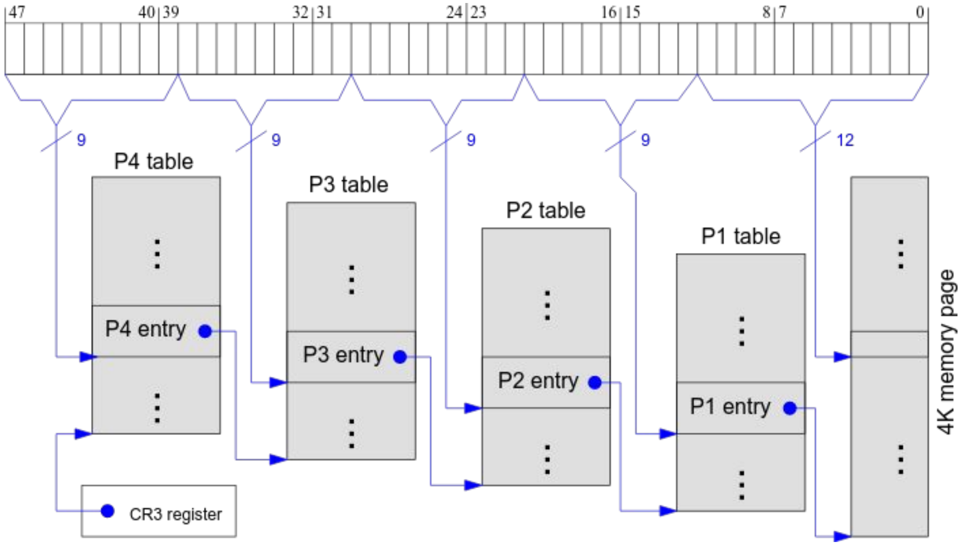
- **Sample Output**



```
devel@gentoo ~/kernellab/paddr $ sudo ./va2pa
[TEST CASE]     PASS
```

Part A : Find Physical Address

- **Hints**
  - You can find the page walk APIs in */usr/src/linux/arch/x86/include/asm/pgtable.h*
  - Basic conversion from virtual address to physical address
  - Access to the next level page table needs to be done in virtual address
    - ✓ Apply this conversion scheme to hierarchical page table.



Part A : Find Physical Address

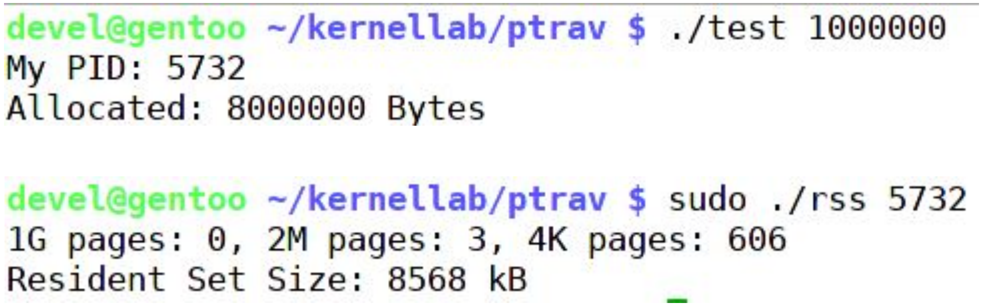
- *make* command include *insmod*.
  - Include *insmod* under the *all* section in your *Makefile*.
- Do not change the debugfs directory and file name.
  - Directory name: *paddr*
  - Physical address output file name: *output*
- Do not modify the file *va2pa.c*
  - We will test your program using uniform *va2pa.c*.
  - Just use *va2pa.c* for testing your implementation.

4. Part B

Part B : Calculate the Resident Set Size of a Process

- **Resident set size (RSS)** is the portion of memory occupied by a process on the memory
- Print the *resident set size* of *test* program
- First, execute the program *test* and get its pid.
- Implement your kernel module to walk through the page tables of *test* program and print result

- **Sample Output**

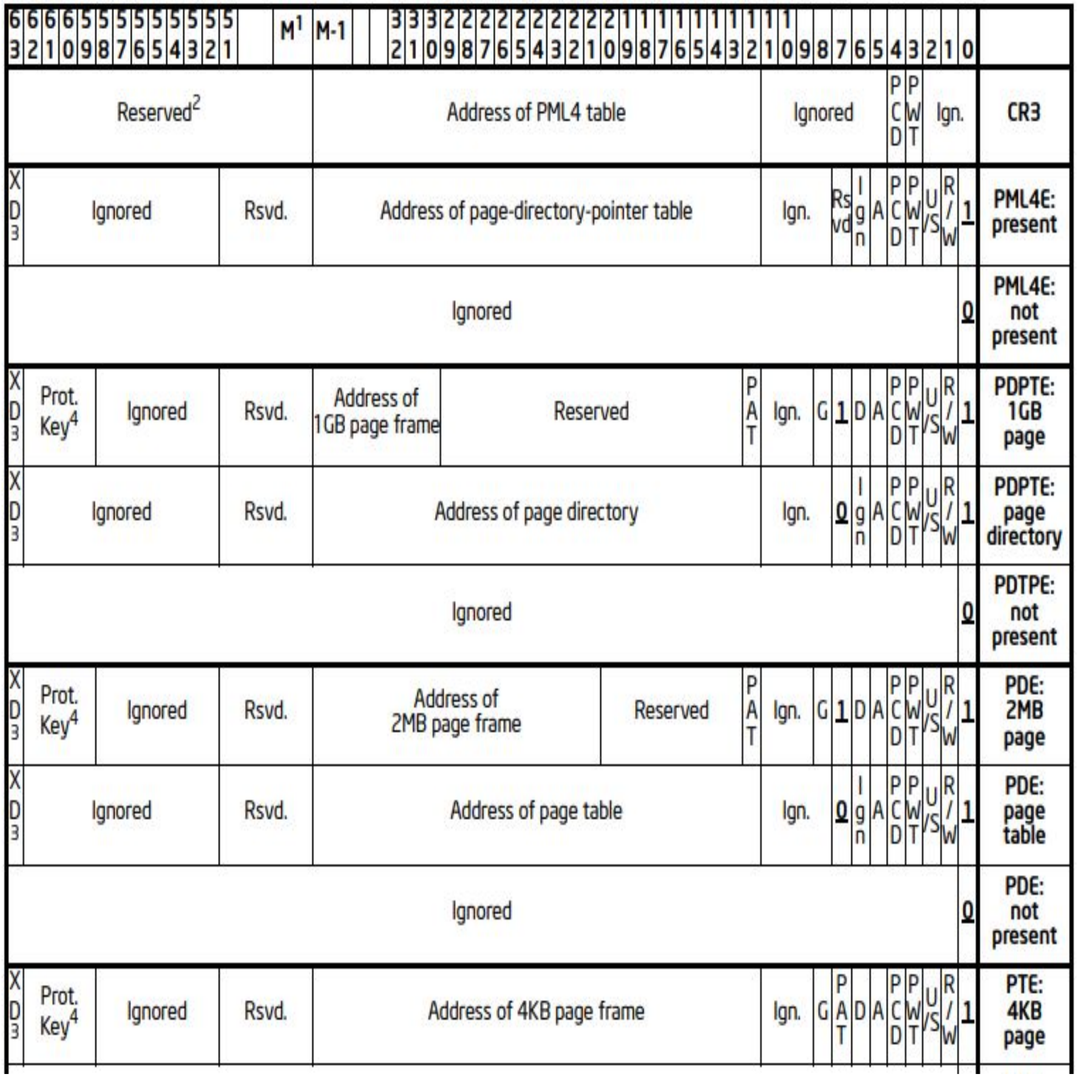


```
devel@gentoo ~/kernellab/ptrav $ ./test 1000000
My PID: 5732
Allocated: 8000000 Bytes

devel@gentoo ~/kernellab/ptrav $ sudo ./rss 5732
1G pages: 0, 2M pages: 3, 4K pages: 606
Resident Set Size: 8568 kB
```

Part B : Calculate the Resident Set Size of a Process

- **Hints**
  - You can find the relevant APIs in */usr/src/linux/arch/x86/include/asm/pgtable.h*
  - task->mm->pgd contains the address of first level page table (pgd or PML4)
  - Page size of a page table is 4KB, how many entries in 64-bit?
  - Virtual address space (user): 0x0000000000000000~0x00007FFFFFFFFFFF (lower 128TB)
  - Refer to the Intel Software Developer’s Manual
    - ✓ **Examine which bits are used for which purposes (present, page size, etc.)**

Part B : Calculate the Resident Set Size of a Process



63	62 61 60 59	58 57 56 55 54 53 52	51 M[1]	M-1 32 31 30	29 28 27 26 25 24 23 22 21	20 19 18 17 16 15 14 13	12	11 10 9	8	7	6	5	4	3	2	1	0	
Reserved[2]				Address of PML4 table				Ignored					PCD	PWT	Ign.			CR3
XD3	Ignored		Rsvd.	Address of page-directory-pointer table				Ign.		Rsvd	Ign	A	PCD	PWT	U/S	R/W	1	PML4E: present
Ignored																	0	PML4E: not present
XD3	Prot. Key[4]	Ignored	Rsvd.	Address of 1GB page frame	Reserved		PAT	Ign.	G	1	D	A	PCD	PWT	U/S	R/W	1	PDPTE: 1GB page
XD3	Ignored		Rsvd.	Address of page directory				Ign.		0	Ign	A	PCD	PWT	U/S	R/W	1	PDPTE: page directory
Ignored																	0	PDTPE: not present
XD3	Prot. Key[4]	Ignored	Rsvd.	Address of 2MB page frame		Reserved	PAT	Ign.	G	1	D	A	PCD	PWT	U/S	R/W	1	PDE: 2MB page
XD3	Ignored		Rsvd.	Address of page table				Ign.		0	Ign	A	PCD	PWT	U/S	R/W	1	PDE: page table
Ignored																	0	PDE: not present
XD3	Prot. Key[4]	Ignored	Rsvd.	Address of 4KB page frame				Ign.		G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page

Part B : Calculate the Resident Set Size of a Process

- *make* command include *insmod*.
    - Include *insmod* under the *all* section in your *Makefile*.
- Do not change the debugfs directory and file name.
    - Directory name: *ptrav*
    - Physical address output file name: *output*
- Never modify the file *rss.c* or *test.c*
    - We will test your program using them.
    - Just use *rss.c* & *test.c* for testing your implementation.

5. Report

Report

- Your report should not be longer than 6 pages (excluding the cover page).
- Avoid copy-pasting screenshots of your code. We have your code. What we ask for here is your thought process applied to solve the lab. (More of a general remark since you are not submitted code but the idea remains)
- You can also attach some diagrams to depict your implementation, if necessary.
- Delete italic text when submitting your report. Those are only guidelines to help you
- The name of the file must match *201X-XXXXX_kernellab_report.pdf* and placed under *report* directory.

6. Submission Guidelines

Submission

- **Due Date**
  - Tue., October 15, 16:59

- **Submission Files**
  - Source code
  - Makefile
  - Report (in PDF format, file name: 201X-XXXXX_kernellab_report.pdf)

- Follow the report template and naming convention
- Do not change the directory structure
- Do not email us your code and report
- You can add more files (.c, .h) for your implementations but the module should be built by using only the *make* command
- Failure to follow any of the submission guidelines can result in a **deduction** of your score

Note

- This lab is quite **tough**
  - You need to look through the kernel source
  - Debugging for the kernel is not an easy task

- **So, start working as soon as possible**

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
- **Address Translation & Page Table Entry**
  - Class Text Book: “*Computer Systems A Programmer’s Perspective, Randal E. Bryant, David R. O’Hallaron, 3rd International Edition, Pearson, 2016*” - page 862
  - Class Text Book: “*Computer Systems A Programmer’s Perspective, Randal E. Bryant, David R. O’Hallaron, 3rd International Edition, Pearson, 2016*” - page 863 ~ 864
  - Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3, Chapter 4
- **Makefile Guide**
  - https://www.cs.duke.edu/~ola/courses/programming/Makefiles/Makefiles.html