Project 2: Android Memory Management

Department of Computer Science and Engineering
Shanghai Jiao Tong University
Spring 2019

Objectives

- Compile the Android kernel.
- Familiarize Android page replacement algorithm.
- Get the target process's virtual address and physical address.
- Implement a page replacement aging algorithm.

Enviroment

- Implementation
  - AVD(Android Virtual Devices)
    - SDK version r24.4.1
- Development
  - Linux（64-bits）
    - Ubuntu (recommended)
    - Debian
    - Fedora

What to Submit

- A“tar” file of your DIRECTORY, containing:
  - All *.c, *.h files you have changed in Linux kernel.
  - Any “readme” or “.pdf” files asked for in the project
  - Screen captures of the scheduler test
    - If you cannot get your program to work, submit a run of whatever you can get to work as you can get partial credit

- DO NOT SUBMIT your object or executable files, REMOVE them before you pack your directory.

How to Submit

- Pack your code in a project directory

  tar –cvf Prj2+StudentID.tar project2

- Send your Prj2+StudentID.tar file to sjtu_os_2019@163.com
  - Ex. Prj2+115XXXXXXXXX

For Help?

- Teaching Assistant
  - Dongyu Lu
    - Email: sjtuldy@sjtu.edu.cn
  - Liyi Guo
    - Email: liyig1114@qq.com

Statement

- Word in blue means it is variable or function.
- Word in red means it is an absolute path in system.
- Word in green means it is the path in kernel files.
- Word in blue means it is an application.

Outline

- Problem 1:
  - Compile the Kernel
- Problem 2:
  - Map a Target Process's Page Table
- Problem 3:
  - Investigate Android Process Address Space
- Problem 4:
  - Change Linux Page Replacement Algorithm

Compile the Kernel

Compile the Linux Kernel

- Make sure that your environment variable is correct.

```
export JAVA_HOME=/usr/lib/jdk1.8.0_73
export JRE_HOME=/usr/lib/jdk1.8.0_73/jre
export CLASSPATH=.:$CLASSPATH:$JAVA_HOME/lib:$JRE_HOME/lib
export PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin
export PATH=~/Kit/android-sdk-linux/platform-tools:$PATH
export PATH=~/Kit/android-sdk-linux/tools:$PATH
export PATH=~/Kit/android-ndk-linux:$PATH
export PATH=~/Kit/android-ndk-linux/toolchains/arm-linux-androideabi-4.9/prebuilt/linux-x86_64/bin:$PATH
```

Compile the Linux Kernel (cont.)

- Modify Makefile in the kernel
  - Change
    - ARCH ?= $(SUBARCH)
    - CROSS_COMPILE ?=
  - To

```
export KBUILD_BUILDHOST := $(SUBARCH)
ARCH            ?= arm
CROSS_COMPILE   ?= arm-linux-androideabi-

# Architecture as present in compile.h
```

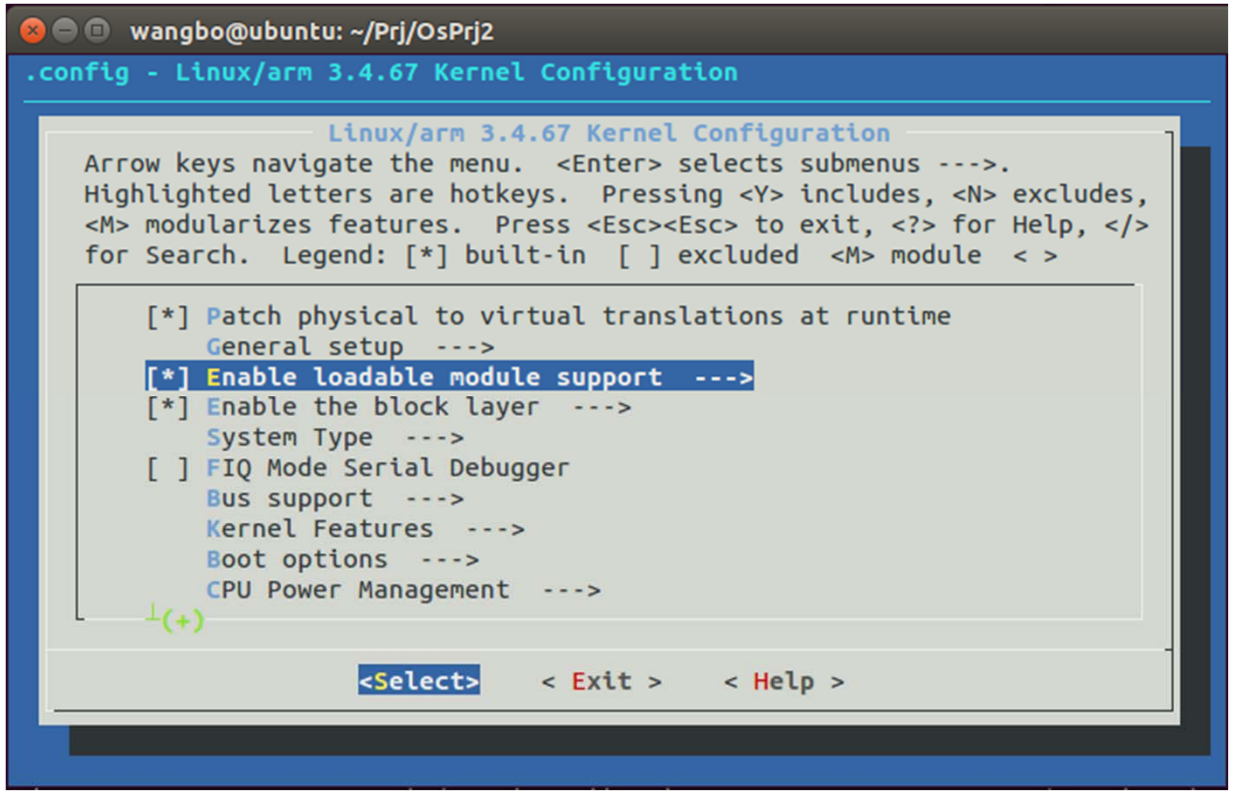
Compile the Linux Kernel (cont.)

- Execute the following command:

```
wangbo@ubuntu:~/Prj/OsPrj2$ make goldfish_armv7_defconfig
#
# configuration written to .config
#
wangbo@ubuntu:~/Prj/OsPrj2$ sudo apt-get install ncurses-dev
wangbo@ubuntu:~/Prj/OsPrj2$ make menuconfig
```

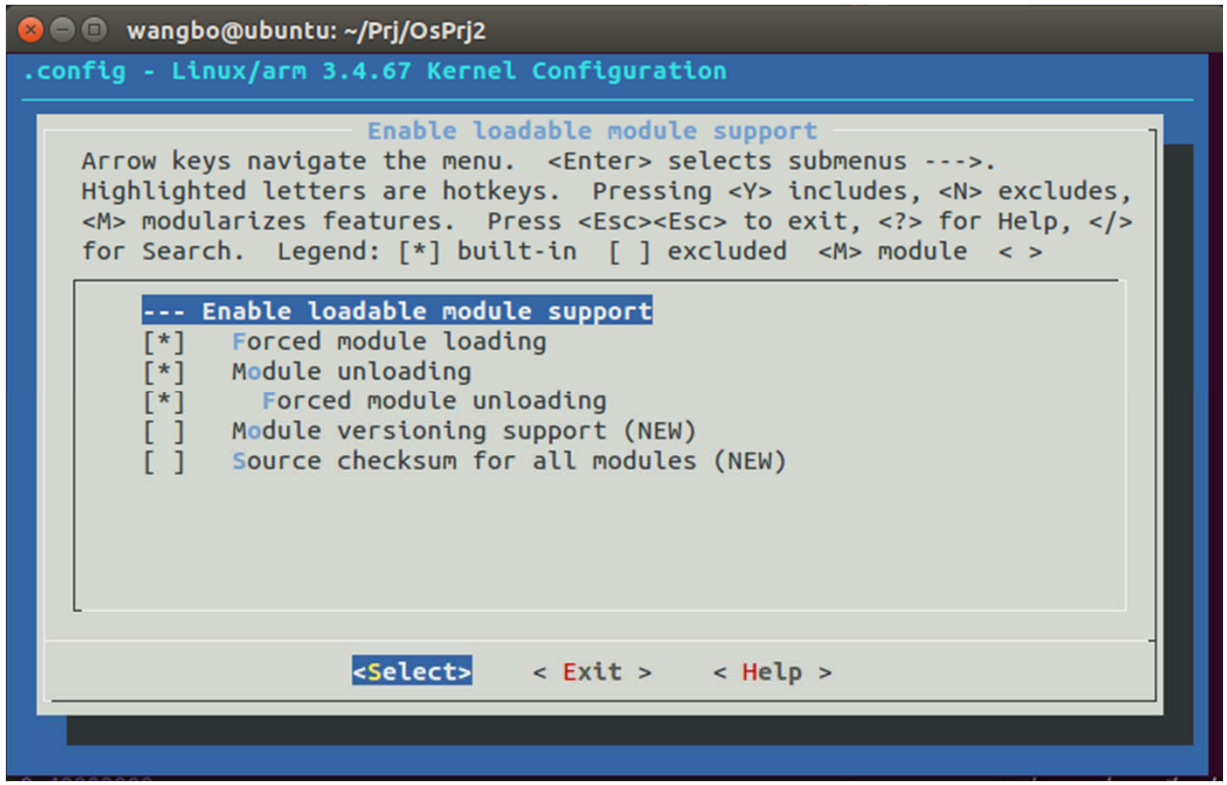
Compile the Linux Kernel (cont.)

- Then you can see:



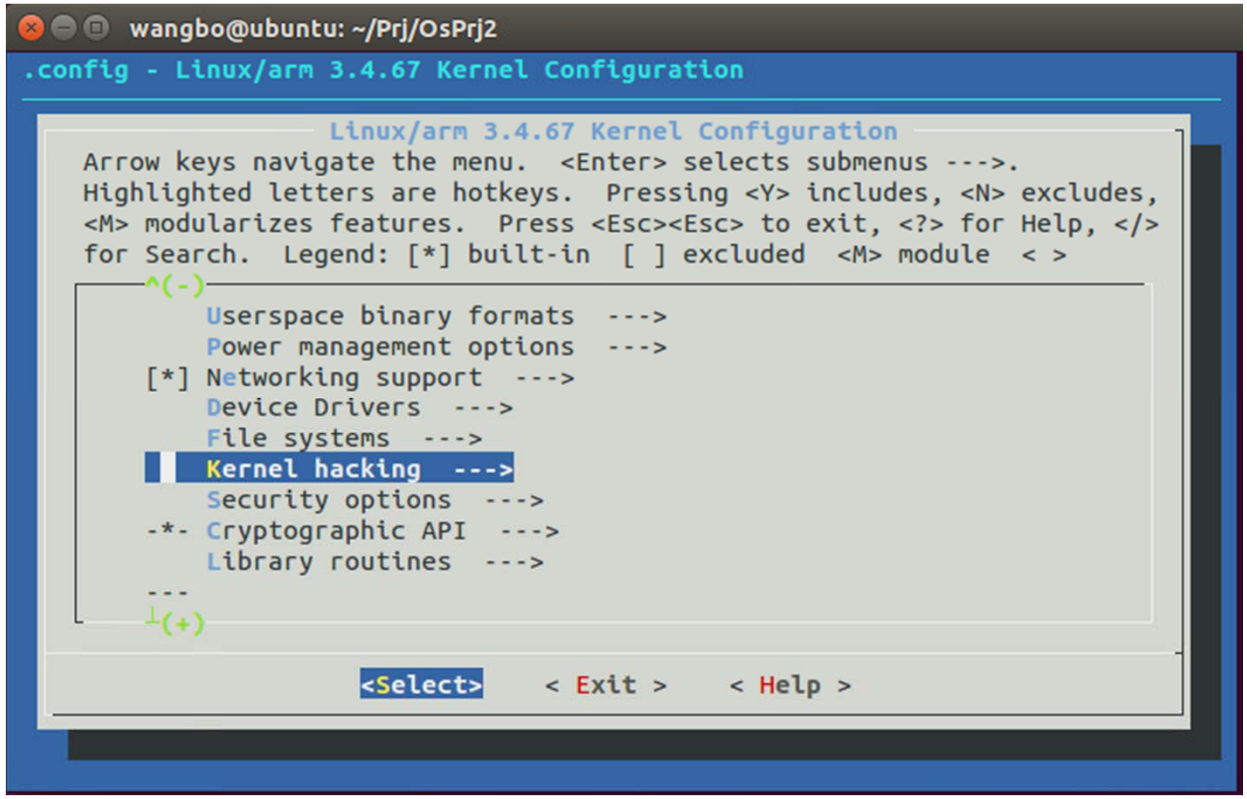
Compile the Linux Kernel (cont.)

- Then you can see:



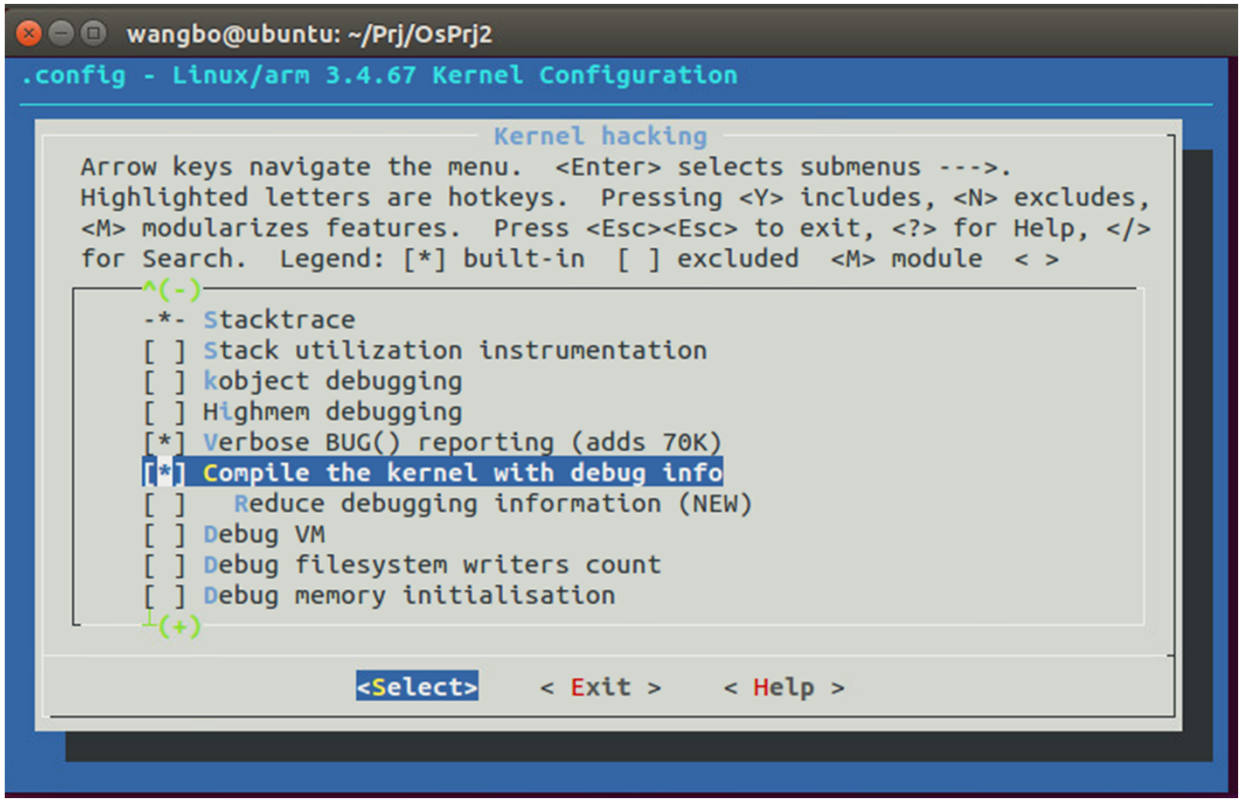
Compile the Linux Kernel (cont.)

- Then you can see:



Compile the Linux Kernel (cont.)

- Then you can see:



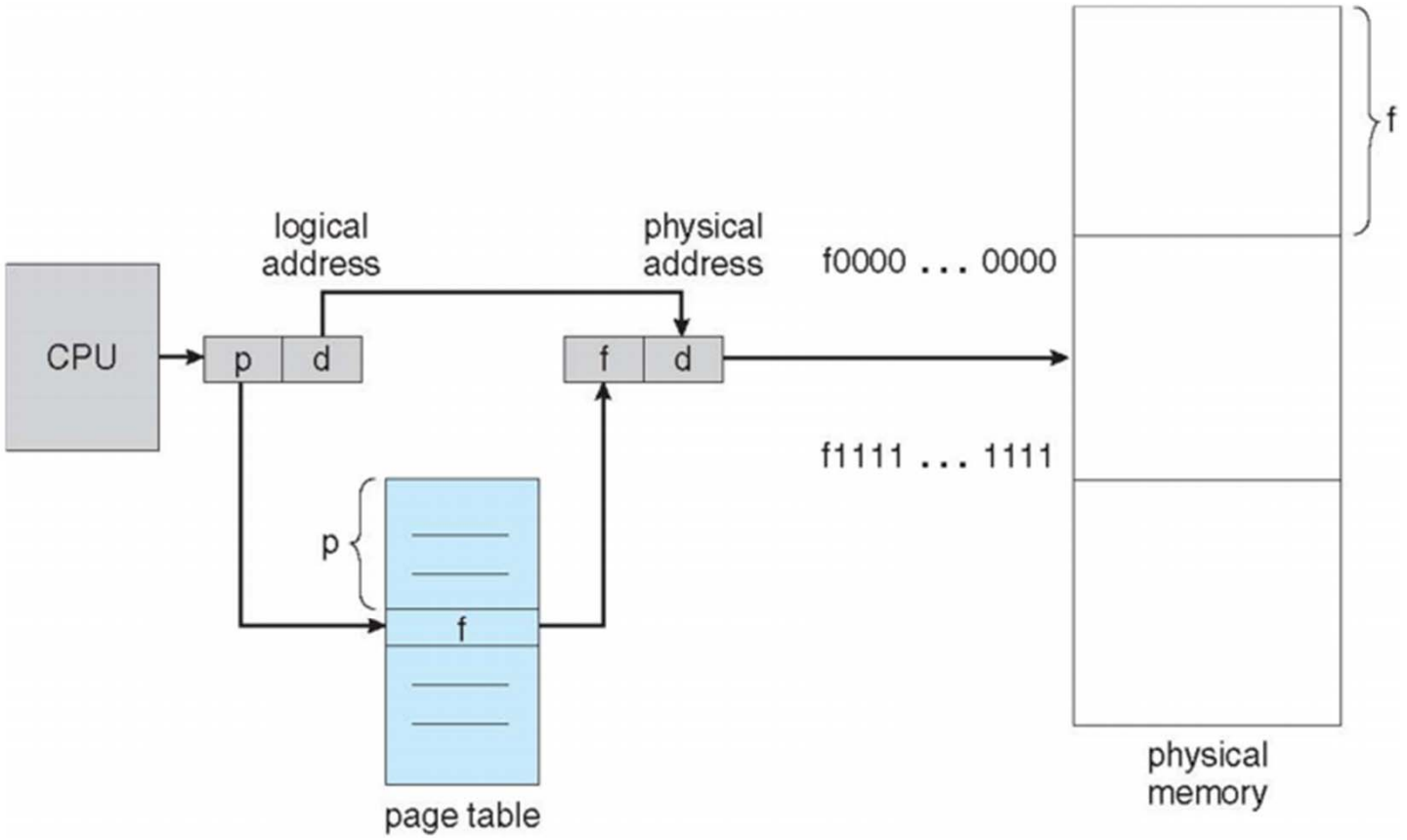
Compile the Linux Kernel (cont.)

- Compile it

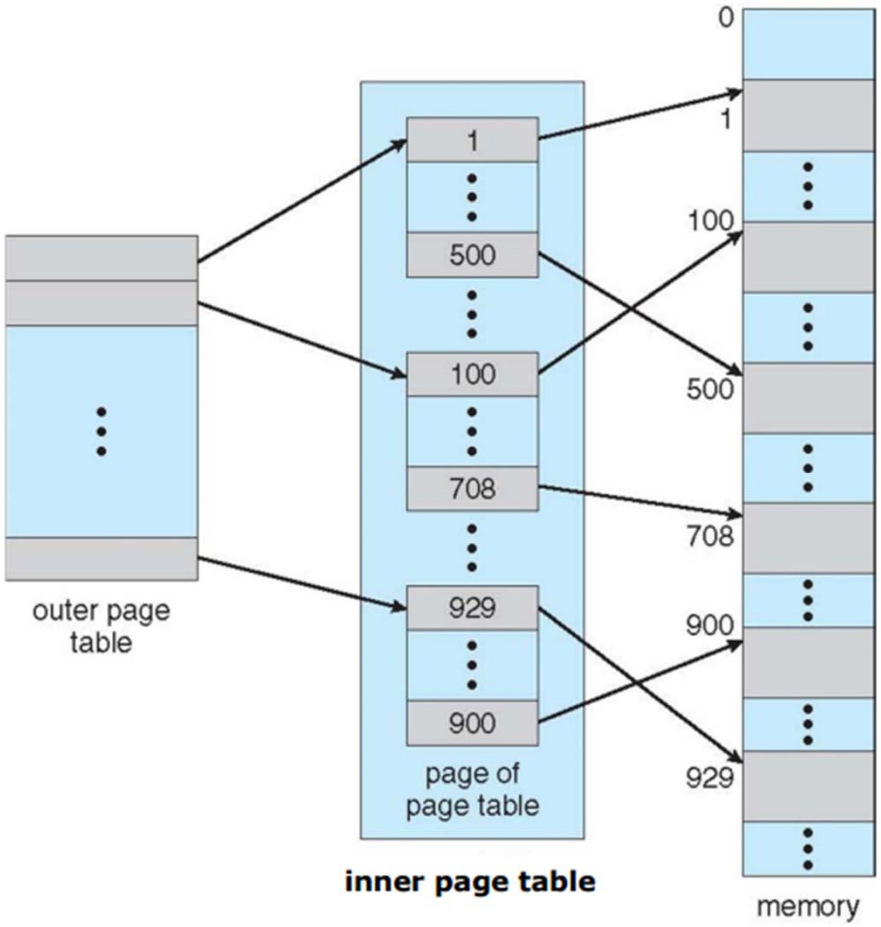
```
wangbo@ubuntu:~/Prj/OsPrj2$ make -j4
  SYSMAP  System.map
  SYSMAP  .tmp_System.map
  OBJCOPY arch/arm/boot/Image
  Kernel: arch/arm/boot/Image is ready
  GZIP    arch/arm/boot/compressed/piggy.gzip
  AS      arch/arm/boot/compressed/piggy.gzip.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
wangbo@ubuntu:~/Prj/OsPrj2$
```

Map a Target Process's Page Table

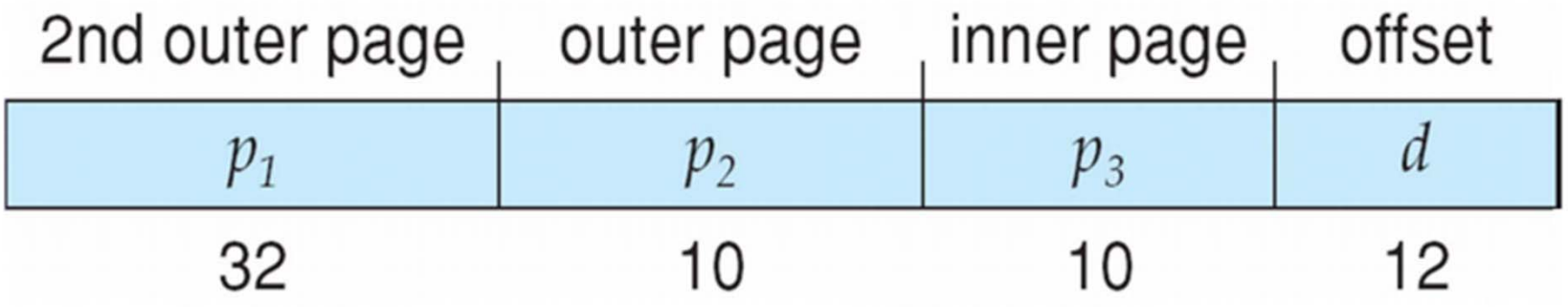
Paging


CPU
logical address
p
d
physical address
f
d
f0000 . . . 0000
f1111 . . . 1111
f
p
f
page table
physical memory

Two-Level Page-Table Scheme

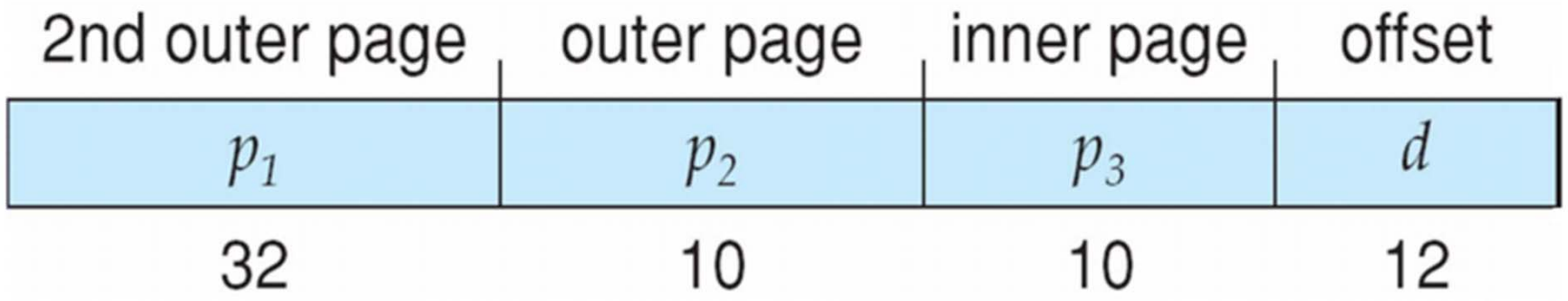


Linux Paging



Investigate the page table layout

- struct pagetable_layout_info {
  uint32_t pgdir_shift;
  uint32_t pmd_shift;
  uint32_t page_shift;
  };
- int get_pagetable_layout(struct pagetable_layout_info __user * pgtbl_info, int size);
- # 3561



Map a target process's Page Table

- int expose_page_table(pid_t pid,
  unsigned long fake_pgd,
  unsigned long fake_pmds,
  unsigned long page_table_addr,
  unsigned long begin_vaddr,
  unsigned long end_vaddr);

@pid: pid of the target process you want to investigate,

@fake_pgd: base address of the fake pgd

@fake_pmds: base address of the fake pmds

@page_table_addr: base address in user space the ptes mapped to

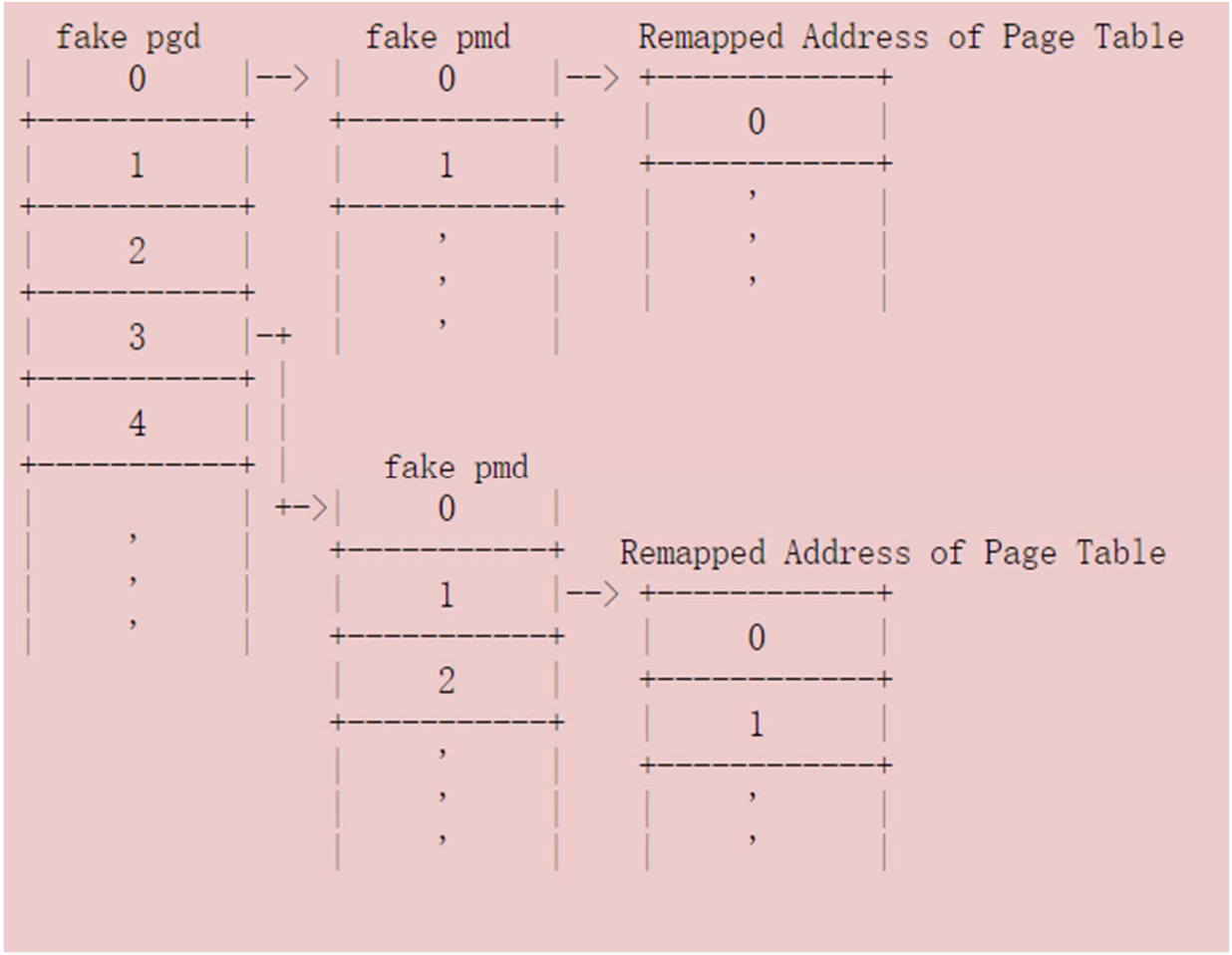
[@begin_vaddr, @end_vaddr]: remapped memory range of the target process

3562

Map a target process's Page Table

- When you try to find the address of the remapped page table that translates a given address ADDR:
  - You will have to first get the pgd_index => (pgd_index = pgd_index(ADDR)).
  - From the entry fake_pgd_base + (pgd_index * sizeof(each_entry)), you will either get a null for non-exist pmd or the address of a fake pmd.
  - Repeat the above process with the pmd_index => (pmd_index = pmd_index(ADDR)), you can get the remapped address of a Page Table by reading the content at the address fake_pmd_base + (pmd_index * sizeof(each_entry)).
  - Finally, you have the read access of the remapped Page Table.

Map a target process's Page Table



Map a target process's Page Table

- Get the index for fake pgd by calling pgd_index(VA).
- Use the index to find the corresponding entry in fake pgd, and fetch the base address of the fake pmd.
- Get the index for fake pmd by calling pmd_index(VA).
- Use the index to find the corresponding entry in fake pmd, and fetch the base address of the remapped page table.
- Get the index for the 3rd level page table, and find the corresponding PTE (page table entry) of the VA.
- Interpret the PTE and get the PA for the VA.

Hints

- You can find the definition of some page table information in:
  - include/asm-generic/pgtable*.h
- remap_pfn_range() will be of use to you.
- It's a bad idea to use malloc to prepare a memory section fo mapping page tables, because malloc cannot allocate memory more than MMAP_THRESHOLD (128kb in default). Instead you should consider to use the mmap system call, take a look at do_mmap_pgoff in mm/mmap.c to set up the proper flags to pass to mmap.
- Once you've mmap'ed a memory area for remapping page tables, the kernel will create a Virtual Memory Area (e.g. vma, struct vm_area_struct in the kernel) for it. You should set up the proper vm_flags for this mmaped region to make sure that it will not be merged with some other vmas.

Test the system calls

- Wirte an application named VATranslate
  - Run on AVD
  - ./VATranslate VA
  - Output PA

Investigate Android Process Address Space

Investigate Android Process Address Space

- Implement a program called vm_inspector to dump the page table entries of a process in given range.
- To dump the PTEs of a target process, you will have to specify a process identifier "pid" to vm_inspector.
- ./vm_inspector pid va_begin va_end

Investigate Android Process Address Space

- You should only print pages that are present. If a page is not present, it should be omitted from your output.
- Try open an Android App in your AVD and play with it. Then use vm_inspector to dump its page table entries for multiple times and see if you can find some changes in your PTE dump.

Investigate Android Process Address Space

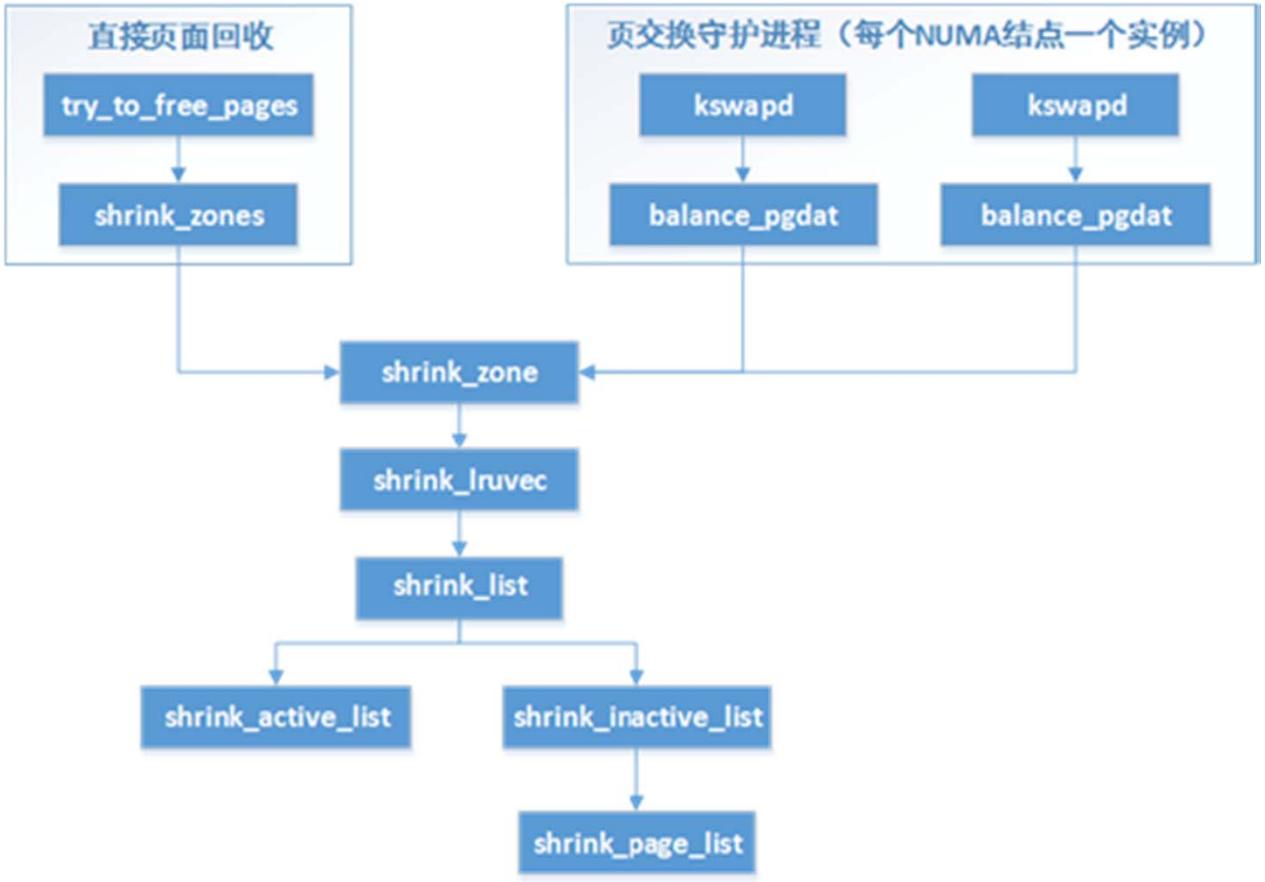
- Use vm_inspector to dump the page tables of multiple processes, including Zygote. Refer to /proc/pid/maps in your AVD to get the memory maps of a target process and use it as a reference to find the different and the common parts of page table dumps between Zygote and an Android app.

Change Linux Page Replacement Algorithm

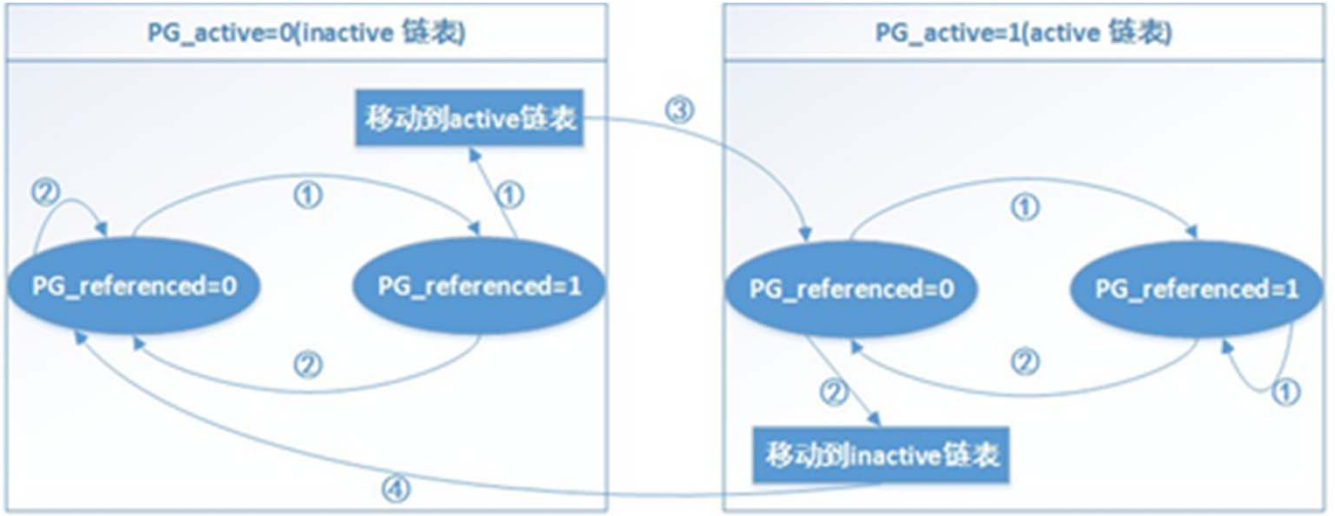
Linux Page Replacement Algorithm

- Two lists in struct zone
  - active_list, inactive_list
- Two bits in struct page
  - PG_active: is page on active list?
  - PG_referenced: has page been referenced recently?
- Approximate LRU algorithm
  - Replace a page in inactive list
  - Move from active to inactive under memory pressure
  - Need two accesses to go from inactive to active

Linux Page Replacement Algorithm



Linux Page Replacement Algorithm



Linux Page Replacement Algorithm

- mark_page_accessed():
  - called twice to move a page from inactive to active
- page_referenced():
  - test if a page is referenced
- refill_inactive_zone():
  - move pages from active to inactive
- lru_cache_add*():
  - add to inactive or active list

Change Linux Page Replacement Algorithm

- PG_referenced reflects the importance of a page.
- Shift PG_referenced one bit to the right for every period.
- If a page is referenced by process, add PG_referenced after shifting by 2^{K} which is defined by yourself.
- For inactive list, move the page whose PG_referenced is the largest one to active list.
- For active list, move the page whose PG_referenced is smaller than a threshold that defined by yourself.

Change Linux Page Replacement Algorithm

- kswapd() will check the pages periodically.
- /proc/meminfo shows the information of active and inactive memory.
- page is defined in

  include/linux/mm_types.h
- kswapd() is defined in

  mm/vmscan.c

Report

- Explain how you system-call work in detail.
- Explain what you found through vm_inspector.
- Explain how you achieve new algorithm.

Deadline

Mid-night, June. 14, 2019

Demo & Presentation

- Demo:
  - **June 15-16, 2019**. Demo slots will be posted in the Wechat group. Please sign your name in one of the available slots.
- Presentation:
  - You are encouraged to present your design of the project optionally. The presentation date is Jun. 16, 2019.

For Help?

- Teaching Assistant
  - Dongyu Lu
    - Email: sjtuldy@sjtu.edu.cn
  - Liyi Guo
    - Email: liyig1114@qq.com

For Help?

- Teaching Assistant
  - Dongyu Lu
    - Email: sjtuldy@sjtu.edu.cn
  - Liyi Guo
    - Email: liyig1114@qq.com
- Some useful website
  - http://www.csdn.net/
  - http://stackoverflow.com/
  - http://developer.android.com/