CS356 Operating System Projects Spring 2017

Project 2: Android Memory Management

yelantingfeng

June 9, 2017

Contents

1 Objectives

I Compile the Android kernel.

II Familiarize Android page replacement algorithm.

III Get the target process's virtual address and physical address.

IV Implement a counting algorithm page replacement.

2 Project Environment

Implementation: AVD(Android Virtual Devices), SDK version r24.4.1
Linux (64-bits): Ubuntu 16.04 LTS

3 Details for Implementation

3.1 Problem I : Compile the kernel

3.1.1 Description

Our task for this problem is to learn how to change the configs of the compilation of the kernel by modifying Makefile and using ncurses-dev. This is the basis of the tasks behind.

3.1.2 Analysis

This problem is the easiest part of the whole project. We just need to follow the following steps and then we can make everything right.

a. Make sure that you have added the following path into your environment variable.

```
ANDROID_NDK_HOME/toolchains/arm-linux-androideabi-4.6/prebuilt/linux-x86_64/bin
```

To achieve this, you can append some codes in ~/.bashrc, a hidden file in you home directory.

b. Open Makefile in KERNEL_SOURCE/goldfish/ and find these:

```
    ARCH        ?= $(SUBARCH)
    CROSS_COMPILE ?=
```

Change it to:

```
    ARCH        ?= arm
    CROSS_COMPILE ?= arm-linux-androideabi-
```

Save it and exit.

c. Execute the following command in terminal to set compiling config:

```
$ make goldfish_armv7_defconfig
```

d. Modify compiling config:

```
$ sudo apt-get install ncurses-dev
$ make menuconfig
```

Then you can see a GUI config. Open the `Compile the kernel with debug info` in `Kernel hacking` and `Enable loadable module support` with `Forced module loading`, `Module unloading` and `Forced module unloading` in it. Save it and exit.

e. Compile.

```
$ make -j4
```

The number of `-j*` depends on the number of cores of your CPU.

3.2 Problem II : Map a target process's Page Table

3.2.1 Description

In the Linux kernel, the page table is broken into multiple levels. For ARM64-based devices, a three-level paging mechanism is used. For this problem, we are going to implement the following two system call interfaces, and then use them to implement an VATranslate program to test our calls, which can get the physical address via command `./VATranslate #pid #VA` . The required system call interfaces are as follows.

```
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);
```

```
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);
```

3.2.2 Analysis

I think it is a better way to add system calls by writing a module. The reason is that once you have already booted your virtual device, if you want to make some change on your system call, you do not need to reboot your device. This is important to shorten our debug process.

The first system call is not too difficult. I firstly searched PGD_SHIFT on linux-Elixir-Free Electrons, and I note that we should select the kernel version v3.4.67. Then I found it defined in some arch/arm/include/asm/pgtable*.h. Since we've already included include/linux/sched.h, which includes pgtable.h, so all that we need to do is just to copy these given macros to user space. You can see some more details in the first code of Solution part.

To finish the second system call, I have done a lot of homework. Before I start to implement this call, I wrote a small application to test the first call and then found that it is using 2-level paging mechanism. To finish this task, I involked three functions in kernel, `find_get_pid`, `walk_page_range` and `remap_pfn_range`, but the second function has not been exported for module programming. To invoke it in our module, we have to modify two files in the kernel. We add `extern` before the function prototype of `walk_page_range` in include/linux/mm.h. And then add two lines below in mm/pagewalk.c.

```
#include <linux/export.h>
EXPORT_SYMBOL(walk_page_range);
```

The function `walk_page_range` can recursively walk the page table for the memory area in VMA. And `remap_pfn_range` can remap the given physical frame to userspace. You can see some more details in the second code of Solution part.

Once we have finished these two system calls, it is easy to implement an VATranslate program. We just need to use the first one to get the layout and the second one to copy necessary page table to our memory space. Then we can use the page table and the fake page directory to translate the virtual address. Note that the lower 12 bits of each entry are used to store some information, so we should mask them when do the translation. For more details, you can read the third code in Solution part.

3.2.3 Solution

I just show the critical part of my solution code.

```c
//get_pagetable_layout implementation.
static int get_pagetable_layout(struct pagetable_layout_info __user* pgtbl_info,
    int size)
{
    unsigned long eleSize=sizeof(uint32_t);
    uint32_t pgd_s=PGDIR_SHIFT;
    uint32_t pmd_s=PMD_SHIFT;
    uint32_t page_s=PAGE_SHIFT;
    if(size<eleSize)
        return size;
    else
    {
        if(copy_to_user(&(pgtbl_info->pgdir_shift),&pgd_s,eleSize))
            return -1;
        if(size<eleSize*2)
            return size;
        else
        {
            if(copy_to_user(&(pgtbl_info->pmd_shift),&pmd_s,eleSize))
                return -1;
            if(size<eleSize*3)
                return size;
            else if(copy_to_user(&(pgtbl_info->page_shift),&page_s,eleSize))
                return -1;
        }
    }
    return 0;
}
```

```c
//expose_page_table implementation.
struct walk_info
{
     unsigned long pgtb_start;
     unsigned long fake_pgd;
     unsigned long * copied_pgd;
};

int cb4pgd(pmd_t *pgd,unsigned long addr,unsigned long end,struct mm_walk *walk)
{
     unsigned long pgdInd=pgd_index(addr);
     unsigned long pgdpg=pmd_page(*pgd);
     unsigned long pfn=page_to_pfn((struct page *)pgdpg);
     if(pgd_none(*pgd)||pgd_bad(*pgd)||!pfn_valid(pfn)) return -EINVAL;
     int err=0;
     struct walk_info *copy_info=walk->private;
     struct vm_area_struct *user_vma=current->mm->mmap;
     if(!user_vma) return -EINVAL;
     struct vm_area_struct *vmp;
     down_write(&current->mm->mmap_sem);
     err=remap_pfn_range(user_vma,copy_info->pgtb_start,pfn,PAGE_SIZE,user_vma->
         vm_page_prot);
     up_write(&current->mm->mmap_sem);
     if(err)
     {
         printk(KERN_INFO"remap_pfn_range failed!\n");
         return err;
     }
```

```
    copy_info->copied_pgd[pgdInd]=copy_info->pgtb_start;
    copy_info->pgtb_start+=PAGE_SIZE;
    return 0;
}

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)
{
    if(begin_vaddr>=end_vaddr)
    {
        printk(KERN_INFO"Range fault!\n");
        return range_fault;
    }
    struct pid* pid_struct=find_get_pid(pid);
    if(!pid_struct)return no_such_pid;
    struct task_struct *tarp=get_pid_task(pid_struct,PIDTYPE_PID);
    struct vm_area_struct *vmp;
    struct mm_walk walk={};
    struct walk_info copy_info={};
    int err=0;
    walk.mm=tarp->mm;
    walk.pgd_entry=&cb4pgd;
    copy_info.pgtb_start=page_table_addr;
    copy_info.fake_pgd=fake_pgd;
    copy_info.copied_pgd=kcalloc(PAGE_SIZE,sizeof(unsigned long),GFP_KERNEL);
    if(!copy_info.copied_pgd)
    {
        printk(KERN_INFO"kcalloc failed!\n");
        return allo_fail;
    }
    walk.private=&copy_info;
    printk(KERN_INFO"The process with pid %d is %s",pid,tarp->comm);
    printk(KERN_INFO"vm area:\n");
    if(!(tarp->mm&&tarp->mm->mmap))
    {
        return no_mem_info;
    }
    for(vmp=tarp->mm->mmap;vmp;vmp=vmp->vm_next)
    {
        printk(KERN_INFO"%08X->%08X\n",vmp->vm_start,vmp->vm_end);
    }
    current->mm->mmap->vm_flags|=VM_SPECIAL;
    down_write(&tarp->mm->mmap_sem);
    err=walk_page_range(begin_vaddr,end_vaddr,&walk);
    up_write(&tarp->mm->mmap_sem);
    if(err)
    {
        printk(KERN_INFO"Walk fail!\n");
        return walk_fail;
    }
    if(copy_to_user(fake_pgd,copy_info.copied_pgd,sizeof(unsigned long)*PAGE_SIZE
       )) return cp_fail;
    kfree(copy_info.copied_pgd);
    return 0;
}
```

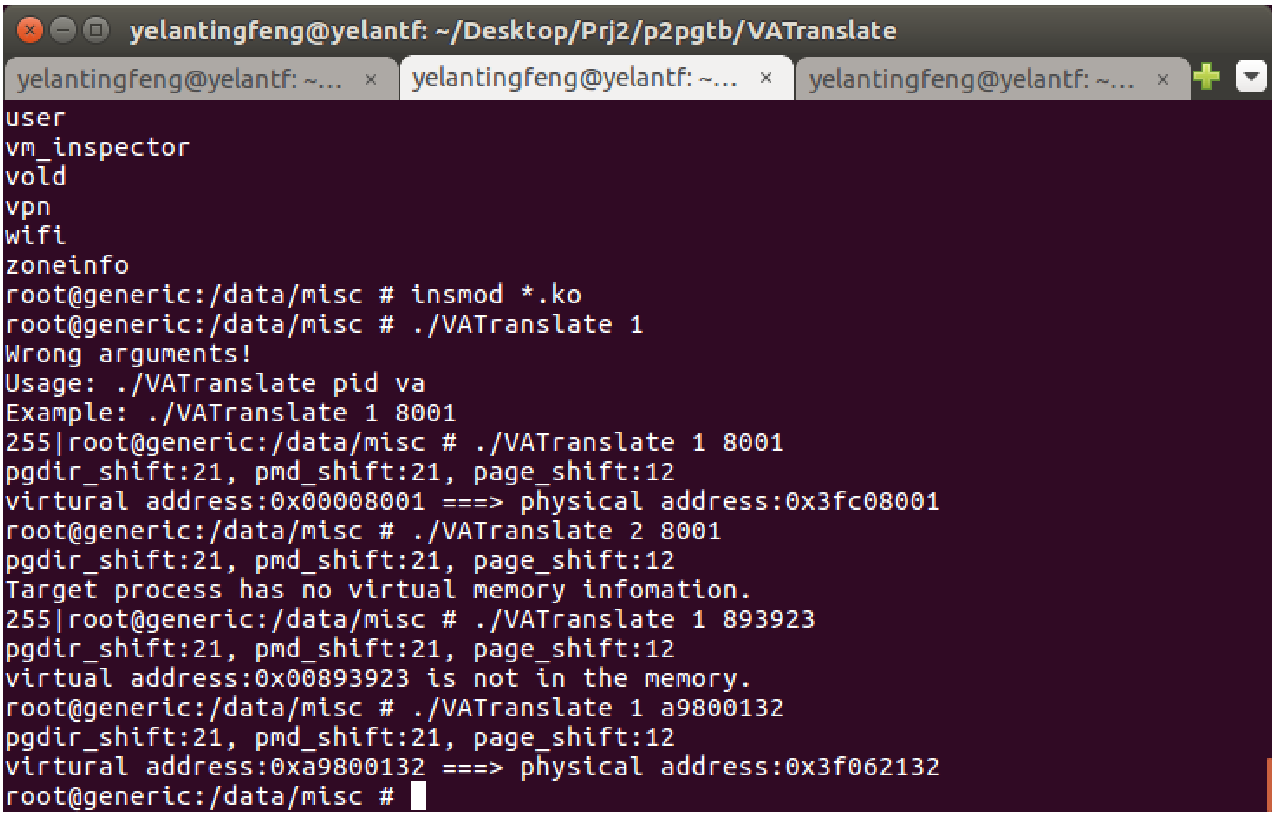
```c
//VATranslate.c
#include<stdio.h>
#include<stdlib.h>
#include<sys/mman.h>
#define pgd_index(va,loinfo) ((va)>>loinfo.pgdir_shift)
#define pte_index(va,loinfo) (((va)>>loinfo.page_shift)&((1<<(loinfo.pmd_shift-
    loinfo.page_shift))-1))
#define no_such_pid 1
#define no_mem_info 2
#define allo_fail   3
#define range_fault 4
#define walk_fail   5
#define cp_fail     -1

struct pagetable_layout_info{
    uint32_t pgdir_shift;
    uint32_t pmd_shift;
    uint32_t page_shift;
};

int main(int argc,char **argv)
{
    unsigned long va;
    unsigned long *table_addr;
    unsigned long *fake_pgd_addr;
    pid_t pid;
    struct pagetable_layout_info loinfo;
    int i;
    unsigned page_size;
    unsigned long pgd_ind,phy_addr;
    unsigned long *phy_base;
    unsigned long mask;
    int err=0;
    if(argc!=3)
    {
        printf("Wrong arguments!\n");
        printf("Usage: ./VATranslate pid va\n");
        printf("Example: ./VATranslate 1 8001\n");
        return -1;
    }
    pid=atoi(argv[1]);
    va=strtoul(argv[2],NULL,16);
    err=syscall(380,&loinfo,4*3);
    if(err<0)
    {
        printf("copy to user failed!\n");
        return -1;
    }
    printf("pgdir_shift:%d, pmd_shift:%d, page_shift:%d\n",loinfo.pgdir_shift,
       loinfo.pmd_shift,loinfo.page_shift);
    err=0;
    page_size=1<<(loinfo.page_shift);
    mask=page_size-1;
    table_addr=mmap(NULL,page_size,PROT_READ | PROT_WRITE, MAP_SHARED |
       MAP_ANONYMOUS, -1, 0);
    fake_pgd_addr=malloc(sizeof(unsigned long)*page_size);
    if(!table_addr||!fake_pgd_addr)
    {
        printf("allocate memory failed!\n");
        return -1;
    }
    err=syscall(381,pid,fake_pgd_addr,0,table_addr,va,va+1);
    switch(err)
    {
        case no_such_pid:
        printf("Process with pid=%d doesn't exist.\n",pid);
        return -1;
        case no_mem_info:
        printf("Target process has no virtual memory infomation.\n");
        return -1;
        case allo_fail:
        printf("allocate memory failed.\n");
        return -1;
        case walk_fail:
        printf("walk_page_range failed.\n");
        return -1;
        case cp_fail:
        printf("copy to user failed!\n");
        return -1;
    }
    pgd_ind=pgd_index(va,loinfo);
    phy_base=fake_pgd_addr[pgd_ind];
    if(phy_base)
    {
        phy_addr=phy_base[pte_index(va,loinfo)];
        phy_addr=phy_addr&~mask;
        if(phy_addr)
        {
            phy_addr=va&mask|phy_addr;
            printf("virtual address:0x%08lx ===> physical address:0x%08lx\n",va,
                phy_addr);
        }
        else printf("virtual address:0x%08lx is not in the memory.\n",va);
    }
    else printf("virtual address:0x%08lx is not in the memory.\n",va);
    free(fake_pgd_addr);
    munmap(table_addr,page_size);
    return 0;
}
```

3.2.4 Output

Firstly, we should install our mod.

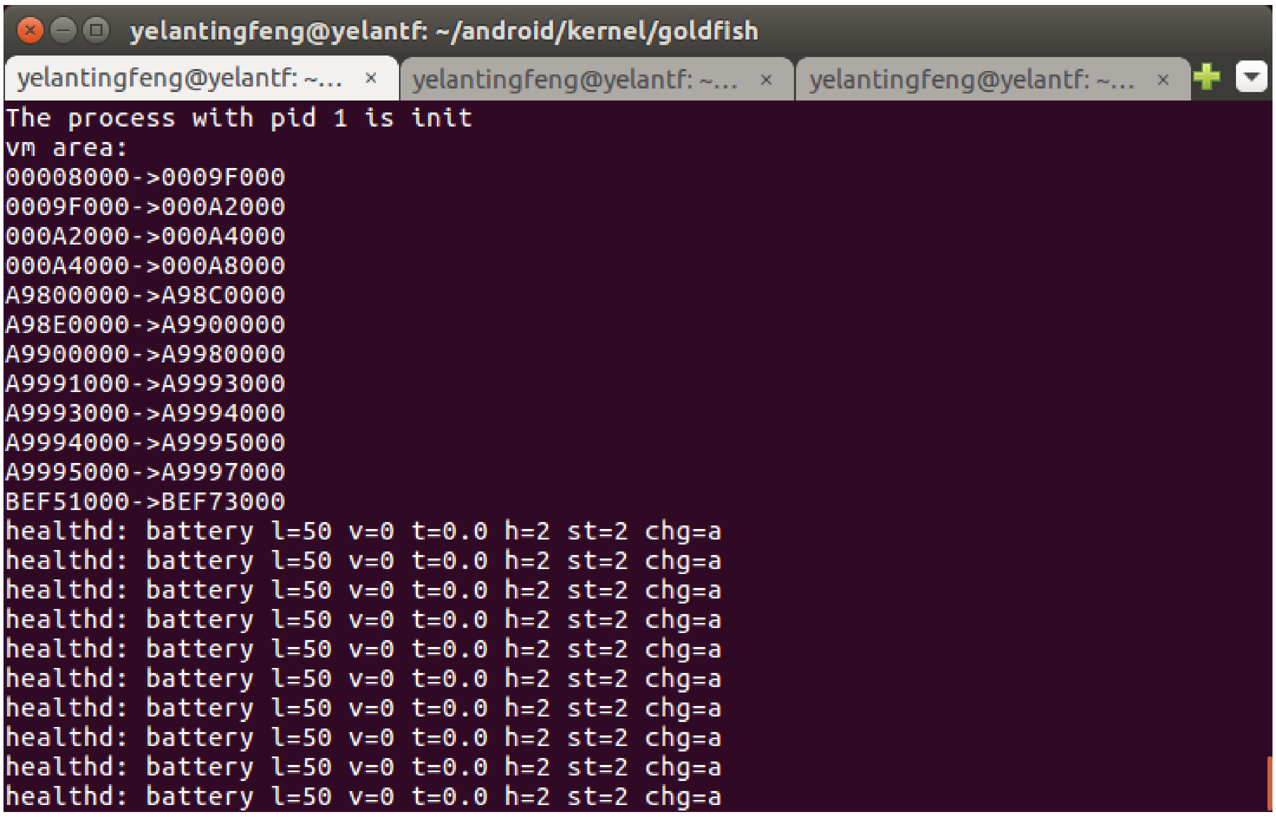
```
$ cd /data/misc
$ insmod *.ko
```

Then we we can use `./VATranslate #pid #VA` to do the translation. Note that not all the process have virtual memory information, to verify this, you can just easily use the command `cat` to check whether the /proc/#pid/maps is empty. Here is a screenshot of my output.


yelantingfeng@yelantf: ~/Desktop/Prj2/p2pgtb/VATranslate
yelantingfeng@yelantf: ~...
yelantingfeng@yelantf: ~...
yelantingfeng@yelantf: ~...

```
user
vm_inspector
vold
vpn
wifi
zoneinfo
root@generic:/data/misc # insmod *.ko
root@generic:/data/misc # ./VATranslate 1
Wrong arguments!
Usage: ./VATranslate pid va
Example: ./VATranslate 1 8001
255|root@generic:/data/misc # ./VATranslate 1 8001
pgdir_shift:21, pmd_shift:21, page_shift:12
virtural address:0x00008001 ===> physical address:0x3fc08001
root@generic:/data/misc # ./VATranslate 2 8001
pgdir_shift:21, pmd_shift:21, page_shift:12
Target process has no virtual memory infomation.
255|root@generic:/data/misc # ./VATranslate 1 893923
pgdir_shift:21, pmd_shift:21, page_shift:12
virtual address:0x00893923 is not in the memory.
root@generic:/data/misc # ./VATranslate 1 a9800132
pgdir_shift:21, pmd_shift:21, page_shift:12
virtural address:0xa9800132 ===> physical address:0x3f062132
root@generic:/data/misc #
```

And some kern info may give me some more help.


yelantingfeng@yelantf: ~/android/kernel/goldfish
yelantingfeng@yelantf: ~...
yelantingfeng@yelantf: ~...
yelantingfeng@yelantf: ~...

```
The process with pid 1 is init
vm area:
00008000->0009F000
0009F000->000A2000
000A2000->000A4000
000A4000->000A8000
A9800000->A98C0000
A98E0000->A9900000
A9900000->A9980000
A9991000->A9993000
A9993000->A9994000
A9994000->A9995000
A9995000->A9997000
BEF51000->BEF73000
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
```

3.3 Problem III : Investigate Android Process Address Space

3.3.1 Description

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`. Use this program to investigate some process' address space, 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.

3.3.2 Analysis

After finishing problem 2, this problem is easy. We just need to round the start virtual address and end virtual address to page start and page end and allocate enough space for our page table and fake pgd. Then we just happily print the entries in the page table from given start virtual address to given end virtual address.

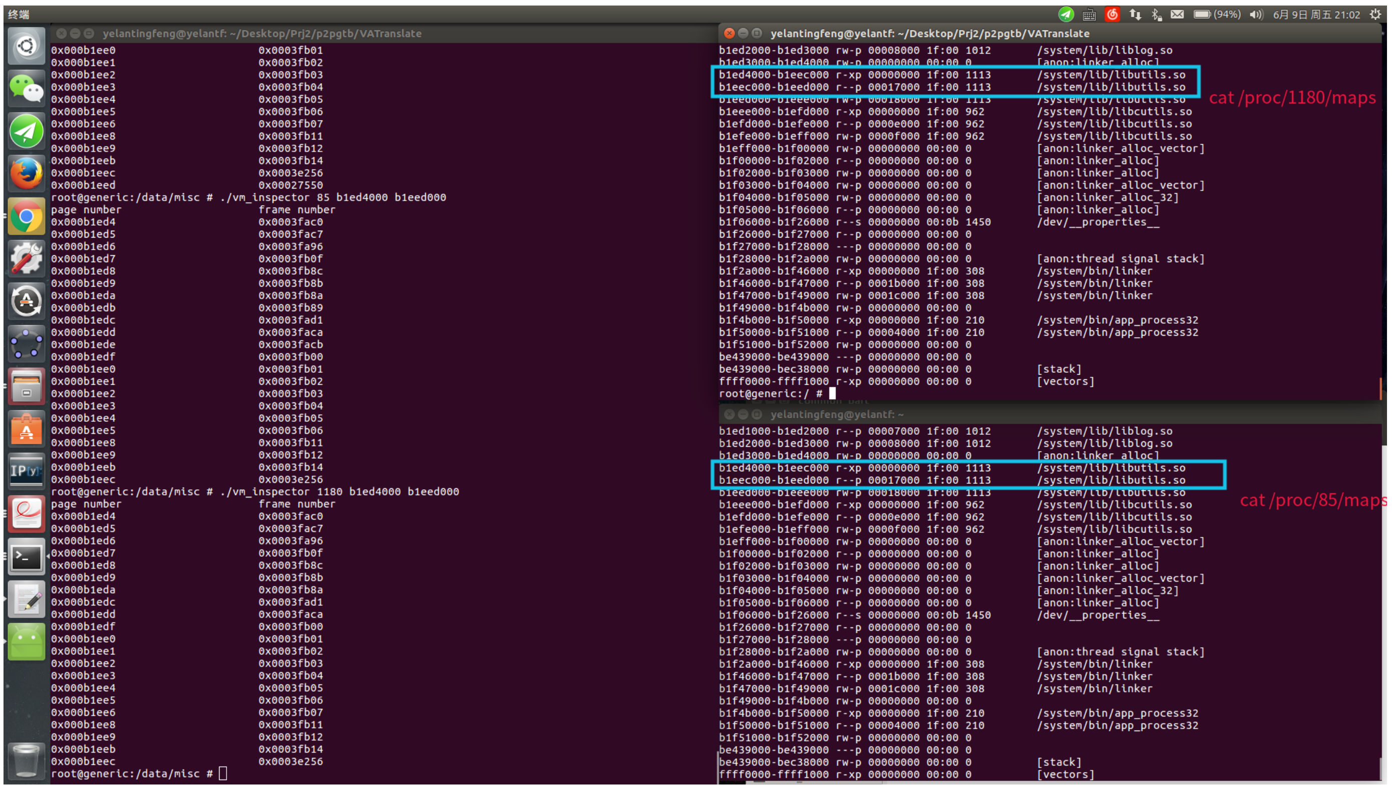
3.3.3 Solution

```c
//vm_inspector.c
#include<stdio.h>
#include<stdlib.h>
#include<sys/mman.h>
#define pgd_index(va,loinfo) ((va)>>loinfo.pgdir_shift)
#define pte_index(va,loinfo) (((va)>>loinfo.page_shift)&((1<<(loinfo.pmd_shift-
    loinfo.page_shift))-1))
#define no_such_pid 1
#define no_mem_info 2
#define allo_fail   3
#define range_fault 4
#define walk_fail   5
#define cp_fail     -1

struct pagetable_layout_info{
    uint32_t pgdir_shift;
    uint32_t pmd_shift;
    uint32_t page_shift;
};

int main(int argc,char **argv)
{
    pid_t pid;
    unsigned long begin_vaddr;
    unsigned long end_vaddr;
    unsigned long *table_addr,*fake_pgd_addr;
    struct pagetable_layout_info loinfo;
    int err=0;
    unsigned long page_size;
    unsigned long rd_begin,rd_end;
    unsigned long mask;
    unsigned long page_nums;
    unsigned long tb_num;
    unsigned long pfn;
    unsigned long pgd_ind,phy_addr;
    unsigned long *phy_base;
    if(argc!=4)
    {
        printf("Wrong arguments!\n");
        printf("Usage: ./vm_inspector pid begin_vaddr end_vaddr\n");
        return -1;
    }
    pid=atoi(argv[1]);
    begin_vaddr=strtoul(argv[2],NULL,16);
    end_vaddr=strtoul(argv[3],NULL,16);
    err=syscall(380,&loinfo,4*3);
    if(err<0)
    {
        printf("copy to user failed!\n");
        return -1;
    }
    err=0;
    err=syscall(380,&loinfo,4*3);
    if(err<0)
    {
        printf("copy to user failed!\n");
        return -1;
    }
    err=0;
    page_size=1<<(loinfo.page_shift);
    mask=page_size-1;
    rd_begin=begin_vaddr&~mask;
    rd_end=(end_vaddr+mask)&~mask;
    page_nums=pgd_index(rd_end-1,loinfo)-pgd_index(rd_begin,loinfo)+1;
    table_addr=mmap(NULL,page_size*page_nums,PROT_READ|PROT_WRITE,MAP_SHARED|
        MAP_ANONYMOUS,-1,0);
    fake_pgd_addr=malloc(sizeof(unsigned long)*page_size);
    if(!table_addr||!fake_pgd_addr)
    {
        printf("allocate memory failed!\n");
        return -1;
    }
    err=syscall(381,pid,fake_pgd_addr,0,table_addr,begin_vaddr,end_vaddr);
    switch(err)
    {
        case no_such_pid:
        printf("Process with pid=%d doesn't exist.\n",pid);
        return -1;
        case no_mem_info:
        printf("Target process has no virtual memory infomation.\n");
        return -1;
        case allo_fail:
        printf("allocate memory failed.\n");
        return -1;
        case range_fault:
        printf("virtual address range fault.\n");
        return -1;
        case walk_fail:
        printf("walk_page_range failed.\n");
        return -1;
        case cp_fail:
        printf("copy to user failed!\n");
        return -1;
    }
    printf("page number\t\t\tframe number\n");
    for(tb_num=rd_begin>>loinfo.page_shift;tb_num<rd_end>>loinfo.page_shift;++
        tb_num)
    {
        pgd_ind=pgd_index(tb_num<<loinfo.page_shift,loinfo);
        phy_base=fake_pgd_addr[pgd_ind];
        if(phy_base)
        {
            phy_addr=phy_base[pte_index(tb_num<<loinfo.page_shift,loinfo)];
            pfn=phy_addr>>loinfo.page_shift;
            if(pfn)
            {
                printf("0x%08lx\t\t\t0x%08lx\n",tb_num,pfn);
            }
        }
    }
    free(fake_pgd_addr);
    munmap(table_addr,page_size*page_nums);
    return 0;
}
```

3.3.4 Output

Still, install the module first and then we can use `vm_inspector #pid #begin_vaddr #end_vaddr` to dump the page tables of target process. Once you start an App, you can use command `ps` to see the pid of the App and you can easily find that page table changes as you play with this App.

Then I dump the page tables of Zygote and Calculator and reference the maps file of them simultaneously. We can easily found that they share so many things in their memory. Then I googled this and found that Zygote is the parent process of most applications, so they share much in their memory. This memory are read-only and this design speed up the launch of applications. Here is a example screenshot of their shared objects.


终端
6月 9日 周五 21:02
yelantingfeng@yelantf: ~/Desktop/Prj2/p2pgtb/VATranslate
yelantingfeng@yelantf: ~/Desktop/Prj2/p2pgtb/VATranslate
yelantingfeng@yelantf: ~
cat /proc/1180/maps
cat /proc/85/maps

```
0x000b1ee0                      0x0003fb01
0x000b1ee1                      0x0003fb02
0x000b1ee2                      0x0003fb03
0x000b1ee3                      0x0003fb04
0x000b1ee4                      0x0003fb05
0x000b1ee5                      0x0003fb06
0x000b1ee6                      0x0003fb07
0x000b1ee8                      0x0003fb11
0x000b1ee9                      0x0003fb12
0x000b1eeb                      0x0003fb14
0x000b1eec                      0x0003e256
0x000b1eed                      0x00027550
root@generic:/data/misc # ./vm_inspector 85 b1ed4000 b1eed000
page number                     frame number
0x000b1ed4                      0x0003fac0
0x000b1ed5                      0x0003fac7
0x000b1ed6                      0x0003fa96
0x000b1ed7                      0x0003fb0f
0x000b1ed8                      0x0003fb8c
0x000b1ed9                      0x0003fb8b
0x000b1eda                      0x0003fb8a
0x000b1edb                      0x0003fb89
0x000b1edc                      0x0003fad1
0x000b1edd                      0x0003faca
0x000b1ede                      0x0003facb
0x000b1edf                      0x0003fb00
0x000b1ee0                      0x0003fb01
0x000b1ee1                      0x0003fb02
0x000b1ee2                      0x0003fb03
0x000b1ee3                      0x0003fb04
0x000b1ee4                      0x0003fb05
0x000b1ee5                      0x0003fb06
0x000b1ee8                      0x0003fb11
0x000b1ee9                      0x0003fb12
0x000b1eeb                      0x0003fb14
0x000b1eec                      0x0003e256
root@generic:/data/misc # ./vm_inspector 1180 b1ed4000 b1eed000
page number                     frame number
0x000b1ed4                      0x0003fac0
0x000b1ed5                      0x0003fac7
0x000b1ed6                      0x0003fa96
0x000b1ed7                      0x0003fb0f
0x000b1ed8                      0x0003fb8c
0x000b1ed9                      0x0003fb8b
0x000b1eda                      0x0003fb8a
0x000b1edc                      0x0003fad1
0x000b1edd                      0x0003faca
0x000b1edf                      0x0003fb00
0x000b1ee0                      0x0003fb01
0x000b1ee1                      0x0003fb02
0x000b1ee2                      0x0003fb03
0x000b1ee3                      0x0003fb04
0x000b1ee4                      0x0003fb05
0x000b1ee5                      0x0003fb06
0x000b1ee6                      0x0003fb07
0x000b1ee8                      0x0003fb11
0x000b1ee9                      0x0003fb12
0x000b1eeb                      0x0003fb14
0x000b1eec                      0x0003e256
root@generic:/data/misc # 
```

```
b1ed2000-b1ed3000 rw-p 00008000 1f:00 1012       /system/lib/liblog.so
b1ed3000-b1ed4000 rw-p 00000000 00:00 0          [anon:linker_alloc]
b1ed4000-b1eec000 r-xp 00000000 1f:00 1113       /system/lib/libutils.so
b1eec000-b1eed000 r--p 00017000 1f:00 1113       /system/lib/libutils.so
b1eed000-b1eee000 rw-p 00018000 1f:00 1113       /system/lib/libutils.so
b1eee000-b1efd000 r-xp 00000000 1f:00 962        /system/lib/libcutils.so
b1efd000-b1efe000 r--p 0000e000 1f:00 962        /system/lib/libcutils.so
b1efe000-b1eff000 rw-p 0000f000 1f:00 962        /system/lib/libcutils.so
b1eff000-b1f00000 rw-p 00000000 00:00 0          [anon:linker_alloc_vector]
b1f00000-b1f02000 r--p 00000000 00:00 0          [anon:linker_alloc]
b1f02000-b1f03000 rw-p 00000000 00:00 0          [anon:linker_alloc]
b1f03000-b1f04000 rw-p 00000000 00:00 0          [anon:linker_alloc_vector]
b1f04000-b1f05000 rw-p 00000000 00:00 0          [anon:linker_alloc_32]
b1f05000-b1f06000 r--p 00000000 00:00 0          [anon:linker_alloc]
b1f06000-b1f26000 r--s 00000000 00:0b 1450       /dev/__properties__
b1f26000-b1f27000 r--p 00000000 00:00 0
b1f27000-b1f28000 ---p 00000000 00:00 0
b1f28000-b1f2a000 rw-p 00000000 00:00 0          [anon:thread signal stack]
b1f2a000-b1f46000 r-xp 00000000 1f:00 308        /system/bin/linker
b1f46000-b1f47000 r--p 0001b000 1f:00 308        /system/bin/linker
b1f47000-b1f49000 rw-p 0001c000 1f:00 308        /system/bin/linker
b1f49000-b1f4b000 rw-p 00000000 00:00 0
b1f4b000-b1f50000 r-xp 00000000 1f:00 210        /system/bin/app_process32
b1f50000-b1f51000 r--p 00004000 1f:00 210        /system/bin/app_process32
b1f51000-b1f52000 rw-p 00000000 00:00 0
be439000-be439000 ---p 00000000 00:00 0
be439000-bec38000 rw-p 00000000 00:00 0          [stack]
ffff0000-ffff1000 r-xp 00000000 00:00 0          [vectors]
root@generic:/ # 
```

```
b1ed1000-b1ed2000 r--p 00007000 1f:00 1012       /system/lib/liblog.so
b1ed2000-b1ed3000 rw-p 00008000 1f:00 1012       /system/lib/liblog.so
b1ed3000-b1ed4000 rw-p 00000000 00:00 0          [anon:linker_alloc]
b1ed4000-b1eec000 r-xp 00000000 1f:00 1113       /system/lib/libutils.so
b1eec000-b1eed000 r--p 00017000 1f:00 1113       /system/lib/libutils.so
b1eed000-b1eee000 rw-p 00018000 1f:00 1113       /system/lib/libutils.so
b1eee000-b1efd000 r-xp 00000000 1f:00 962        /system/lib/libcutils.so
b1efd000-b1efe000 r--p 0000e000 1f:00 962        /system/lib/libcutils.so
b1efe000-b1eff000 rw-p 0000f000 1f:00 962        /system/lib/libcutils.so
b1eff000-b1f00000 rw-p 00000000 00:00 0          [anon:linker_alloc_vector]
b1f00000-b1f02000 r--p 00000000 00:00 0          [anon:linker_alloc]
b1f02000-b1f03000 rw-p 00000000 00:00 0          [anon:linker_alloc]
b1f03000-b1f04000 rw-p 00000000 00:00 0          [anon:linker_alloc_vector]
b1f04000-b1f05000 rw-p 00000000 00:00 0          [anon:linker_alloc_32]
b1f05000-b1f06000 r--p 00000000 00:00 0          [anon:linker_alloc]
b1f06000-b1f26000 r--s 00000000 00:0b 1450       /dev/__properties__
b1f26000-b1f27000 r--p 00000000 00:00 0
b1f27000-b1f28000 ---p 00000000 00:00 0
b1f28000-b1f2a000 rw-p 00000000 00:00 0          [anon:thread signal stack]
b1f2a000-b1f46000 r-xp 00000000 1f:00 308        /system/bin/linker
b1f46000-b1f47000 r--p 0001b000 1f:00 308        /system/bin/linker
b1f47000-b1f49000 rw-p 0001c000 1f:00 308        /system/bin/linker
b1f49000-b1f4b000 rw-p 00000000 00:00 0
b1f4b000-b1f50000 r-xp 00000000 1f:00 210        /system/bin/app_process32
b1f50000-b1f51000 r--p 00004000 1f:00 210        /system/bin/app_process32
b1f51000-b1f52000 rw-p 00000000 00:00 0
be439000-be439000 ---p 00000000 00:00 0
be439000-bec38000 rw-p 00000000 00:00 0          [stack]
ffff0000-ffff1000 r-xp 00000000 00:00 0          [vectors]
```

3.4 Problem IV: Change Linux Page Replacement Algorithm

3.4.1 Description

The task for this problem is to change the page replacement algorithm. We should add a new reference variable to record the last time when this page was referenced. If a page is referenced by process, the referenced value of it should be set to 0. Otherwise, the referenced value increases 1 for every period until it is referenced. We should check these two lists periodically, move the page, whose reference vale is 0, to active list, and move the page, whose referenced value larger than a threshold that defined by yourself, to inactive list. To accomplish this algorithm, `kswapd()` and /proc/meminfo will be helpful.

3.4.2 Analysis

To change the page replacement algorithm, we need to know how the page replacement algorithm works in detail. I think this article helps a lot. The key part of our task is to add a variable in page struct in include/linux/mm_types.h and do some modification on some functions in mm/vmscan.c and mm/swap.c.

3.4.3 Solution

I just show the critical part of my solution code.

```
//include/linux/mm_types.h
struct page{
    //———
    //———
    /* Our added variable.*/
    unsigned long lastRefTime;
};
```

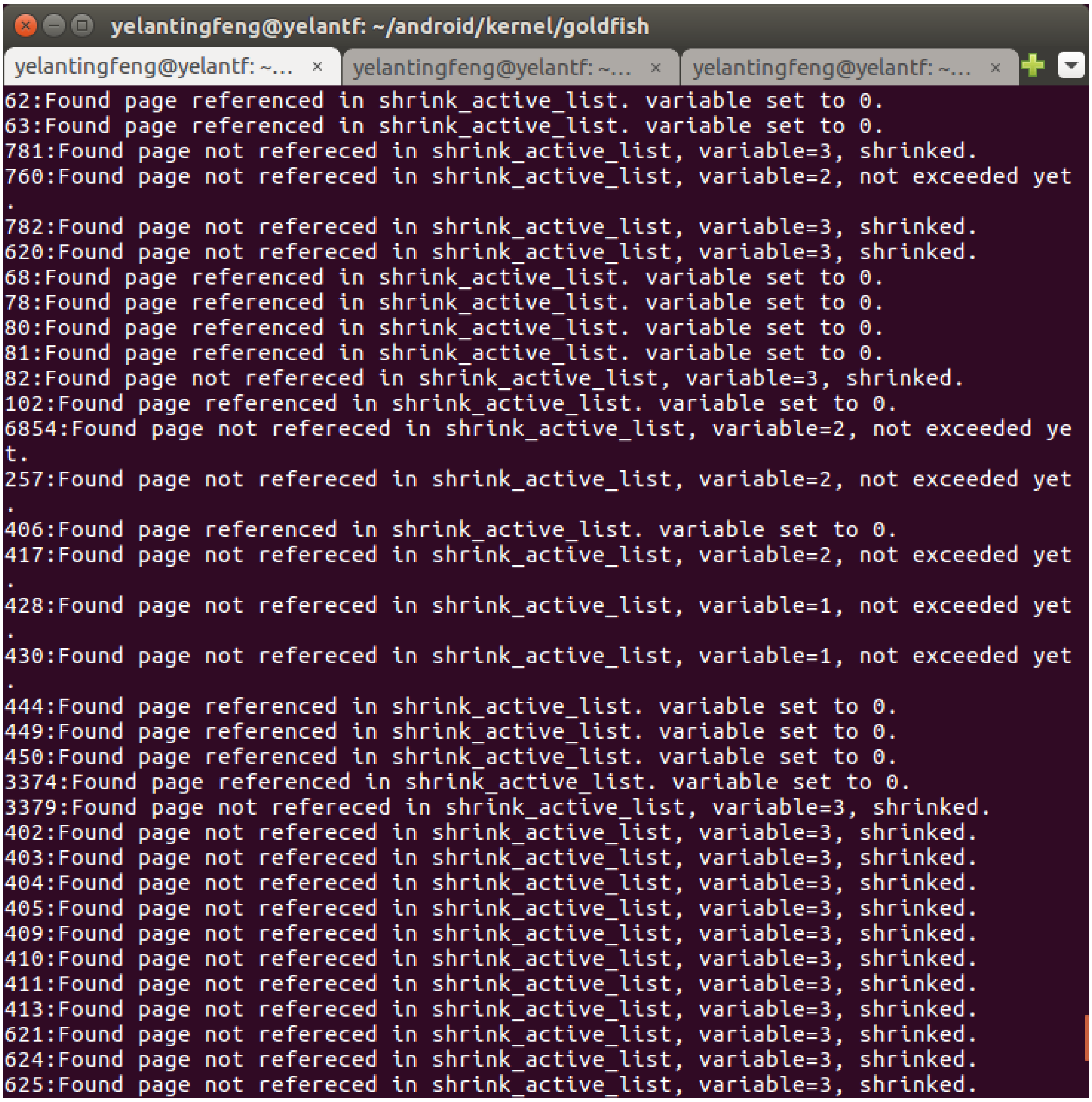
```
//mm/vmscan.c
//———
#define MY_TIME_THRES 2

static void shrink_active_list(unsigned long nr_to_scan,
                                struct mem_cgroup_zone *mz,
                                struct scan_control *sc,
                                int priority, int file)
{
  //———
    if (page_referenced(page, 0, mz->mem_cgroup, &vm_flags)) {
        //let our variable be zero.
        printk(KERN_INFO"%ld:Found page referenced in shrink_active_list.
           variable set to 0.\n",page->index);
        page->lastRefTime=0;
        //—————
    }
    else if\left( ++page->lastRefTime) <= MY_TIME_THRES ){
        printk(KERN_INFO"%ld:Found page not refereced in shrink_active_list,
           variable=%u, not exceeded yet.\n",page->index,page->lastRefTime);
        list_add(&page->lru, &l_active);
        continue;
    }

    printk(KERN_INFO"%ld:Found page not refereced in shrink_active_list, variable
       =%u, shrinked.\n",page->index,page->lastRefTime);\right);
    //————
```

```
}

//—————————

static enum page_references page_check_references(struct page *page,
                          struct mem_cgroup_zone *mz,
                          struct scan_control *sc)
{
  //———————
  if (referenced_ptes) {
            printk(KERN_INFO"%ld:Found referenced in shrink_inactive_list,
               activate page referenced with variable=0.\n",page->index);
            page->lastRefTime=0;
            return PAGEREF_ACTIVATE;
            //———————————
          }
    //———————
}
```

```
//mm/swap.c
//—————————
void mark_page_accessed(struct page *page)
{
    page->lastRefTime=0;
    if (!PageActive(page) && !PageUnevictable(page) &&
                PageLRU(page)) {
        activate_page(page);
    }
}
```

3.4.4 Output

After I write a program to occupy a lot of memory, I get kernel info like this,


yelantingfeng@yelantf: ~/android/kernel/goldfish
yelantingfeng@yelantf: ~...
yelantingfeng@yelantf: ~...
yelantingfeng@yelantf: ~...

```
62:Found page referenced in shrink_active_list. variable set to 0.
63:Found page referenced in shrink_active_list. variable set to 0.
781:Found page not refereced in shrink_active_list, variable=3, shrinked.
760:Found page not refereced in shrink_active_list, variable=2, not exceeded yet
.
782:Found page not refereced in shrink_active_list, variable=3, shrinked.
620:Found page not refereced in shrink_active_list, variable=3, shrinked.
68:Found page referenced in shrink_active_list. variable set to 0.
78:Found page referenced in shrink_active_list. variable set to 0.
80:Found page referenced in shrink_active_list. variable set to 0.
81:Found page referenced in shrink_active_list. variable set to 0.
82:Found page not refereced in shrink_active_list, variable=3, shrinked.
102:Found page referenced in shrink_active_list. variable set to 0.
6854:Found page not refereced in shrink_active_list, variable=2, not exceeded ye
t.
257:Found page not refereced in shrink_active_list, variable=2, not exceeded yet
.
406:Found page referenced in shrink_active_list. variable set to 0.
417:Found page not refereced in shrink_active_list, variable=2, not exceeded yet
.
428:Found page not refereced in shrink_active_list, variable=1, not exceeded yet
.
430:Found page not refereced in shrink_active_list, variable=1, not exceeded yet
.
444:Found page referenced in shrink_active_list. variable set to 0.
449:Found page referenced in shrink_active_list. variable set to 0.
450:Found page referenced in shrink_active_list. variable set to 0.
3374:Found page referenced in shrink_active_list. variable set to 0.
3379:Found page not refereced in shrink_active_list, variable=3, shrinked.
402:Found page not refereced in shrink_active_list, variable=3, shrinked.
403:Found page not refereced in shrink_active_list, variable=3, shrinked.
404:Found page not refereced in shrink_active_list, variable=3, shrinked.
405:Found page not refereced in shrink_active_list, variable=3, shrinked.
409:Found page not refereced in shrink_active_list, variable=3, shrinked.
410:Found page not refereced in shrink_active_list, variable=3, shrinked.
411:Found page not refereced in shrink_active_list, variable=3, shrinked.
413:Found page not refereced in shrink_active_list, variable=3, shrinked.
621:Found page not refereced in shrink_active_list, variable=3, shrinked.
624:Found page not refereced in shrink_active_list, variable=3, shrinked.
625:Found page not refereced in shrink_active_list, variable=3, shrinked.
```

this implies my algorithm works. If we set the threshold to a very high value, like 20, the system will crash since there memory is so tense. And after I change the algorithm and set a proper threshold, everything keeps stable and okay, so it is reasonable to say that I change the algorithm successfully.

4 Discussion

After finish this project, I have a much more deep insight with linux memory management. I know much more about the kernel space and user space, which is exactly what keeps a operating system safe from unsafe operating. The amout of coding of this project is not so much, but we have to read much more than code. By reading the code of our kernel Implementation and the article about kernel memory management, we know more clearly about their actual structure and I think it a precious gift from this project. I am grateful to my classmates who discuss with me and TAs who provide me so much help. Thanks a lot!