

HW3 M11315051 林子新

Hardware :

1. CPU : i5-12400
2. GPU : RTX4070
3. RAM : 32GB (partition 512 MB for Virtual Machine)
4. SSDd : 1TB (partition 60GB for Virtual Machine)

Software :

1. VMware(虛擬環境)
2. ubuntu-24.04.1-desktop-amd64(虛擬作業系統)

此次作業沒有不能使用 compiled image of any packaged distribution 的限制，因此我們選定 ubuntu 做為我們的作業系統

實作步驟：

1. 使用 HW2 的虛擬環境，因此不重新介紹。
2. 在 VMware 上面進行 Linux kernel 的製作 – 修改 kernel

進行 kernel 的修改我們需要修改四個部分

1. 修改 syscall_64.tbl -> 做為 系統呼叫的索引號、名稱及對應的內核函數 (給新增函數 id)
2. 修改 sys.c -> 實現系統呼叫功能 (實現函數)
3. 修改 syscalls.h -> 聲明系統呼叫的原型。(告訴 kernel 新增函數)
4. 新增 prinfo.h -> 將 prinfo.h 的結構宣告好，讓大家知道這個 struct 包含甚麼東西。

實做步驟 修改 syscall_64.tbl

(1) cd arch/x86/entry/syscalls/

(2) nano syscall_64.tbl

(3) 增加 549 64 prinfo sys_prinfo

548 是系統呼叫的號碼

64 是系統的位元數，表示 64 位

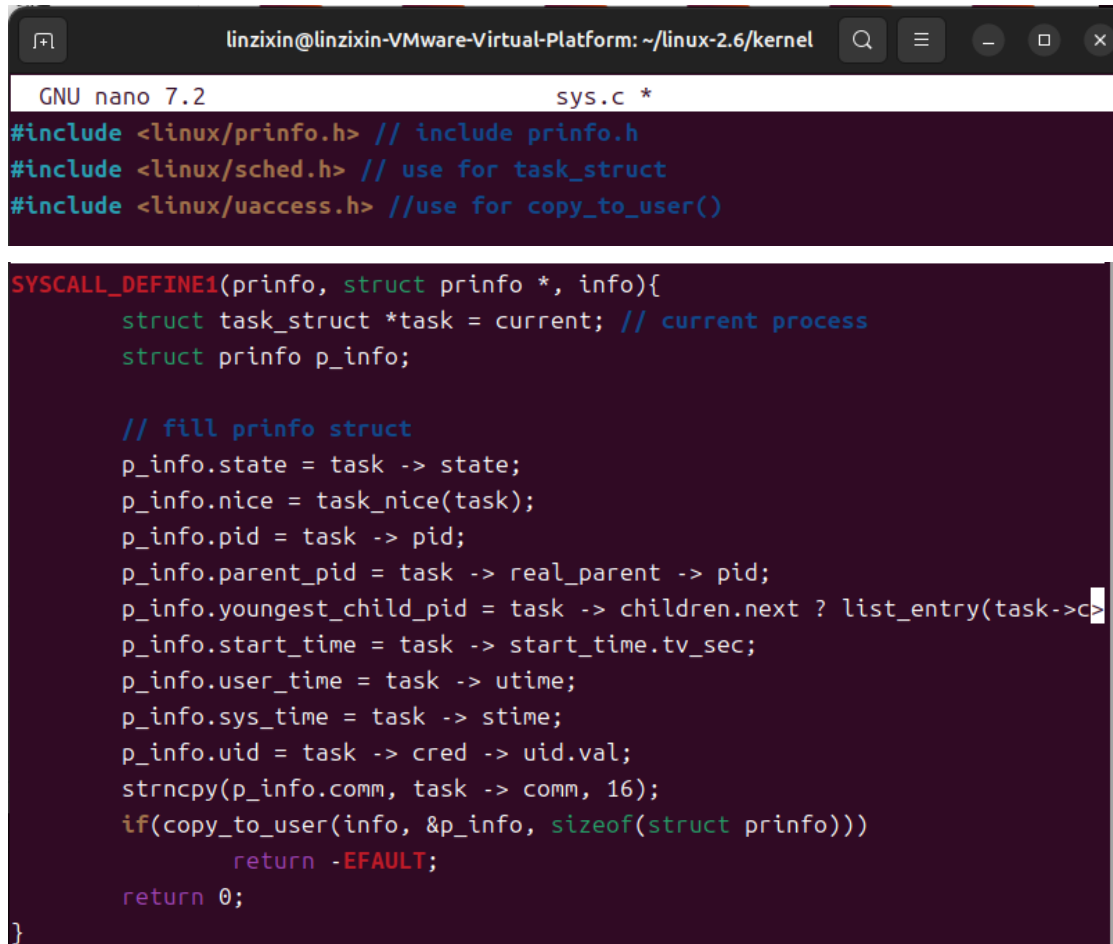
prinfo 給系統呼叫分配的名稱

sys_prinfo 是內核中實際實現系統呼叫的函數名

```
543    x32    io_setup          compat_sys_io_setup
544    x32    io_submit         compat_sys_io_submit
545    x32    execveat          compat_sys_execveat
546    x32    preadv2           compat_sys_preadv64v2
547    x32    pwritev2          compat_sys_pwritev64v2
548    64     print_school_id    sys_print_school_id
549    64     prinfo            sys_prinfo
```

實做步驟 修改 sys.c

- (1) cd /kernel
- (2) nano sys.c
- (3) 找個位置寫入函數



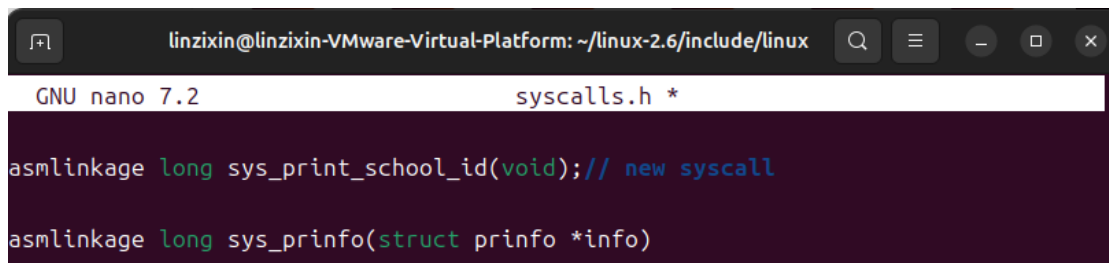
```
linzixin@linzixin-VMware-Virtual-Platform: ~/linux-2.6/kernel
GNU nano 7.2 sys.c *
#include <linux/prinfo.h> // include prinfo.h
#include <linux/sched.h> // use for task_struct
#include <linux/uaccess.h> //use for copy_to_user()

SYSCALL_DEFINE1(prinfo, struct prinfo *, info){
    struct task_struct *task = current; // current process
    struct prinfo p_info;

    // fill prinfo struct
    p_info.state = task -> state;
    p_info.nice = task_nice(task);
    p_info.pid = task -> pid;
    p_info.parent_pid = task -> real_parent -> pid;
    p_info.youngest_child_pid = task -> children.next ? list_entry(task->c>
    p_info.start_time = task -> start_time.tv_sec;
    p_info.user_time = task -> utime;
    p_info.sys_time = task -> stime;
    p_info.uid = task -> cred -> uid.val;
    strncpy(p_info.comm, task -> comm, 16);
    if(copy_to_user(info, &p_info, sizeof(struct prinfo)))
        return -EFAULT;
    return 0;
}
```

實做步驟 修改 syscalls.h

- (1) cd include/linux/
- (2) nano syscalls.h
- (3) 寫入 asmlinkage long sys_prinfo(struct prinfo *info);



```
linzixin@linzixin-VMware-Virtual-Platform: ~/linux-2.6/include/linux
GNU nano 7.2 syscalls.h *

asmlinkage long sys_print_school_id(void); // new syscall

asmlinkage long sys_prinfo(struct prinfo *info)
```

實做步驟 新增 prinfo.h

- (1) cd include/linux/
- (2) nano prinfo.h

```
linzixin@linzixin-VMware-Virtual-Platform: ~/linux-2.6/include/linux
GNU nano 7.2 prinfo.h *
#ifndef _LINUX_PRINFO_H
#define _LINUX_PRINFO_H

struct prinfo{
    long state;
    long nice;
    pid_t pid;
    pid_t parent_pid;
    pid_t youngest_child_pid;
    unsigned long start_time;
    long user_time;
    long sys_time;
    long uid;
    char comm[16];
};

#endif /* _LINUX_PRINFO_H */
```

3. 在 VMware 上面進行 Linux kernel 的製作 – Part3 make time

(1) make menuconfig

```
General setup --->
[*] 64-bit kernel (NEW)
Processor type and features --->
[*] Mitigations for CPU vulnerabilities (NEW) --->
Power management and ACPI options --->
Bus options (PCI etc.) --->
Binary Emulations --->
[*] Virtualization (NEW) --->
```

會看到像是這樣的畫面

(2) Save & exit

(3) sudo make j\$(nproc)

(4) sudo make modules_install

(5) sudo make install

(6) 完成後重啟系統

4. 在 VMware 上面檢查是否已經新增 system call

(1) 撰寫 C 程式來調用新增的系統呼叫

(2) nano test_prinfo.c

```
linzixin@linzixin-VMware-Virtual-Platform: ~/linux-2.6
linzixin@linzixin-VMware-Virtual-Platform:~/linux-2.6$ nano test_prinfo.c
```

(3) 輸入程式



```
linzixin@linzixin-VMware-Virtual-Platform: ~/linux-2.6
GNU nano 7.2 test_prinfo.c
#include <stdio.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <string.h>

struct prinfo{
    long state;
    long nice;
    pid_t pid;
    pid_t parent_pid;
    pid_t youngest_child_pid;
    unsigned long start_time;
    long user_time;
    long sys_time;
    long uid;
    char comm[16];
};

int main(){
    struct prinfo info;

    if(syscall(549, &info.pid) == 0){
        printf("Process ID:%d\n", info.pid);
        printf("Parent PID:%d\n", info.parent_pid);
        printf("Youngest Child PID:%d\n", info.youngest_child_pid);
        printf("State: %ld\n", info.state);
        printf("Nice: %ld\n", info.nice);
        printf("Start time: %lu\n", info.start_time);
        printf("User time: %ld\n", info.user_time);
        printf("System time: %ld\n", info.sys_time);
        printf("UID: %ld\n", info.uid);
        printf("Command length: %lu\n", strlen(info.comm));
        printf("Command: %s\n", info.comm);
    }else{
        perror("sys_prinfo failed");
    }
    return 0;
}
```

(4) gcc test_prinfo.c -o test_prinfo (編譯這個程式)

(5) ./test_prinfo (運行測試程式)

```
linzixin@linzixin-VMware-Virtual-Platform: ~/linux-2.6
linzixin@linzixin-VMware-Virtual-Platform:~/linux-2.6$ gcc test_prinfo.c -o test_prinfo
linzixin@linzixin-VMware-Virtual-Platform:~/linux-2.6$ ./test_prinfo
```

(6) 就可以看到結果了 (這次 shell 是使用 bash)

```
linzixin@linzixin-VMware-Virtual-Platform:~/linux-2.6$ ./test_prinfo
Process ID:0
Parent PID:0
Youngest Child PID:0
State: 648540061702
Nice: 0
Start time: 13490492289232
User time: 0
System time: 225740315609186
UID: 1000000
Command length: 3
Command: @B
*** stack smashing detected ***: terminated
中止 (核心已傾印)
```

3. 使用 zsh shell call system call

實做步驟

(1) 安裝 zsh shell , 輸入 sudo apt-get install zsh

```
linzixin@linzixin-VMware-Virtual-Platform: ~/linux-2.6
linzixin@linzixin-VMware-Virtual-Platform:~/linux-2.6$ sudo apt-get install zsh
```

(2) 切換新終端，輸入 zsh 就會進入新的 shell

```
linzixin@linzixin-VMware-Virtual-Platform: ~/linux-2.6
linzixin@linzixin-VMware-Virtual-Platform:~/linux-2.6$ zsh
```

```
--- Type one of the keys in parentheses ---
Aborting.
The function will be run again next time. To prevent this, execute:
touch ~/.zshrc
linzixin-VMware-Virtual-Platform%
```

(3) 輸入 ./test_prinfo

```
linixin-VMware-Virtual-Platform% ./test_prinfo
Process ID:0
Parent PID:0
Youngest Child PID:0
State: 648540061702
Nice: 0
Start time: 57763015177359
User time: 0
System time: 226268169195783
UID: 2000000
Command length: 3
Command: ♦♦
*** stack smashing detected ***: terminated
zsh: IOT instruction (core dumped) ./test_prinfo
```

實作過程問題 (由處理花費的時間排序)：

1. Command 顯示不出來 stack smashing detected : (未能解決)

這次 HW，Command，一直顯示不出來，並且都是顯示 stack smashing detected，這代表說我的 stack overflow，所以**第一步**我先檢查了 command 的長度

```
Command length: 3
Command: @B
*** stack smashing detected ***: terminated
中止 (核心已傾印)
```

發現長度是 3，而我們 command 的設定是 char comm[16]，所以看起來不是因為 task command 長度太長的關係，那有可能是我的 mem 沒有被 initial，所以裡面可能有資料造成 overflow，所以**第二步**我就去初始化我的 struct prinfo，我在程式中加入 memset(p_info.comm, 0, sizeof(p_info.comm))，以讓我的 p_info.comm 的 mem 都是 0

```
GNU nano 7.2 sys.c
SYSCALL_DEFINE1(prinfo, struct prinfo *, info){
    struct task_struct *task = current; // current process
    struct prinfo p_info;

    memset(p_info.comm, 0, sizeof(p_info.comm));
```

並且為了確實的防止 overflow，所以我手動設定終止符號

```
strncpy(p_info.comm, task->comm, 15);
p_info.comm[15] = '\0';
```

我將 task->comm 的前 15 個 char 貼到 p_info.comm，並手動設定 p_info.comm[15] = '\0'，這樣我可以確定最後一格 char 是設定好的。

但還是遇到一樣的問題 `stack smashing detected`，那我又想到，我直接用 `printk` 來 `task->comm` 到底是什麼不就可以確認到底是什麼問題了嗎？

```
linzixin@linzixin-VMware-Virtual-Platform: ~/linux-2.6/kernel
GNU nano 7.2 sys.c *
memset(p_info.comm, 0, sizeof(p_info.comm));

printk(KERN_INFO "Command: %s\n", task->comm);
```

所以我就加入了 `printk(KERN_INFO "Command: %s\n", task->comm);` 想確定到底發生了甚麼事，於是從 `dmeg` 可以發現是 `process memory` 的問題，並且推薦我去 `coredump` 裡面去看紀錄

```
[ 134.945622] Command: test_prinfo
[ 136.058918] coredump: 3120(test_prinfo): Error writing out the process memory
[ 136.058926] coredump: 3120(test_prinfo): written to |/usr/share/apport/apport
: VMAs: 23, size 438272; core: 61440 bytes, pos 61440
```

所以我就按照上述提示的路徑想去找 `coredump` 的紀錄，但就是很明顯沒有這個檔案，所以我就進入了死胡同，我重試很多次都是一樣的結果。

```
linzixin@linzixin-VMware-Virtual-Platform: ~/linux-2.6$ cd usr/share/apport/apport
bash: cd: usr/share/apport/apport: 沒有此一檔案或目錄
```

2. Struct `task_struct` 的理解

這部分其實沒有花很多時間，就是進到 `sched.h` 看 `task_struct` 的設定是甚麼，我們才能使用這個結構下面是我有找到然後有截圖的部分

```
GNU nano 7.2 sched.h

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
    unsigned int              __state;

#ifdef CONFIG_VIRT_CPU_ACCOUNTING_GEN
    struct vtime              vtime;

    u64                       utime;
    u64                       stime;
```

```
/* Monotonic time in nsecs: */
u64 start_time;
```

```
struct task_struct __rcu *real_parent;

/* Recipient of SIGCHLD, wait4() reports: */
struct task_struct __rcu *parent;

/*
 * Children/sibling form the list of natural children:
 */
struct list_head children;
struct list_head sibling;
```

這次 HW 看完以為應該不難也不會花很多時間，但是意外地遇到了 `stack smashing detected`，我其他的資訊都有正常的顯示出來，唯獨 `command` 沒有顯示出來，我用了很多方法想解決這個問題，但還是沒有成功，之後如果有機會可能會想再研究看看到底是要怎麼解決。