

Department of Computer Science and Information Engineering CSC006, Assignment 2 System calls

60947038S

Hw2-1 Implementation Steps:

Step1: 建立並確保 hi_60947038S.c 能和 kernel source code 一起被 compiler 編譯

至 linux-5.6.9/kernel 建立 hi_60947038S.c 並打上程式碼。

在 Makefile 中找到一行 obj-y 在裡面加入：hi_60947038S.o

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows files: groups.c, groups.o, hi_60947038S.c (selected), hi_60947038S.o, and hung_task.c.
- Editor Bar:** Buttons for Open, Save, etc., and the file name hi_60947038S.c.
- Code Content:**

```
1 #include <linux/kernel.h>
2 #include <linux/linkage.h>
3 #include <linux/init.h>
4 #include <linux/syscalls.h>
5
6 asmlinkage long sys_60947038S(void)
7 {
8     printk(KERN_EMERG "Hi_Sam 60947038S HW2-1 finished");
9     return 0;
10 }
```
- Makefile Snippet:**

```
obj-y = fork.o exec_domain.o panic.o \
cpu.o exit.o softirq.o resource.o \
sysctl.o sysctl_binary.o capability.o ptrace.o user.o \
signal.o sys.o hi_60947038S.o umh.o workqueue.o pid.o task_work.o \
extable.o params.o \
kthread.o sys_ni.o nsproxy.o \
notifier.o ksysfs.o cred.o reboot.o \
async.o range.o smpboot.o ucount.o
```

Step2: 定義我們 new system call function 的 prototype

至 linux-5.6.9/include/linux/syscalls.h,

加入程式碼: asmlinkage long sys_60947038S(void);

```
1000 asmlinkage long sys_fsmount(int fs_fd, unsigned int flags, unsigned int
1001 ms_flags);
1001 asmlinkage long sys_fspick(int dfd, const char __user *path, unsigned int
1002 flags);
1002 asmlinkage long sys_pidfd_send_signal(int pidfd, int sig,
1003                                         struct siginfo __user *info,
1004                                         unsigned int flags);
1005 asmlinkage long sys_pidfd_getfd(int pidfd, int fd, unsigned int flags);
1006
1007 asmlinkage long sys_60947038S(void);
1008 /*
1009  * Architecture-specific system calls
1010 */
1011
```

Step3: 設定 system call 編號為 335
至 `linux/arch/x86/syscalls/syscall_64.tbl`,
加入一行:335 64 hi_60947038S sys_60947038S

341 330	common	pkey_alloc	__x64_sys_pkey_alloc
342 331	common	pkey_free	__x64_sys_pkey_free
343 332	common	statx	__x64_sys_statx
344 333	common	io_pgetevents	__x64_sys_io_pgetevents
345 334	common	rseq	__x64_sys_rseq
346 335	common	hi_60947038S	sys_60947038S

Step4: 安裝需要使用的套件

輸入指令 :

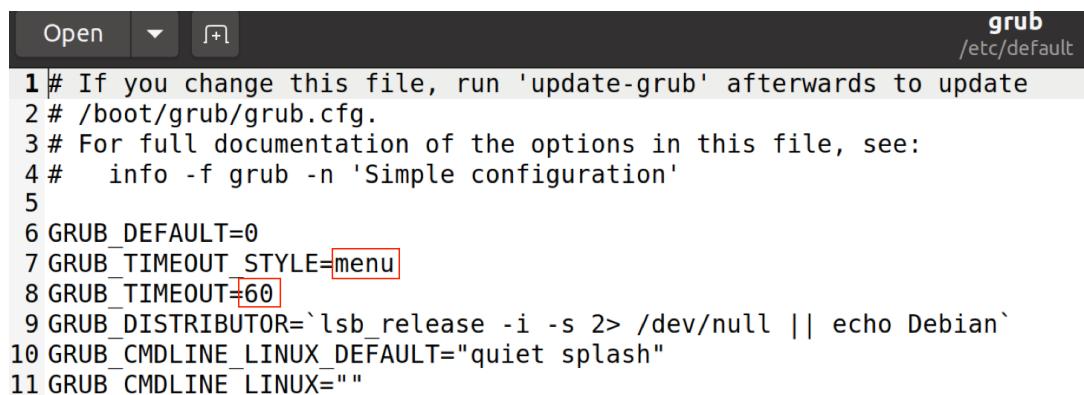
```
sudo apt-get install build-essential manpages-dev libncurses5-dev flex bison openssl libssl-dev libelf-dev libudev-dev libpci-dev libiberty-dev autoconf ncurses-dev
```

Step5: Compile Kernel/Install Kernel

```
sudo make mrproper  
cp -v /boot/config-$(uname -r) .config  
sudo make menuconfig  
sudo make clean  
sudo make -j 8  
sudo make modules  
sudo make modules_install  
sudo make install
```

Step6: Demonstration 測試 system call

```
cd /etc/default  
sudo gedit grub  
sudo update-grub
```



```
1# If you change this file, run 'update-grub' afterwards to update  
2# /boot/grub/grub.cfg.  
3# For full documentation of the options in this file, see:  
4#   info -f grub -n 'Simple configuration'  
5  
6GRUB_DEFAULT=0  
7GRUB_TIMEOUT_STYLE=menu  
8GRUB_TIMEOUT=60  
9GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`  
10GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"  
11GRUB_CMDLINE_LINUX=""
```

reboot 開啟編譯好的 kernel:

```
Terminal └── sam@sam-VirtualBox: ~/Desktop
sam@sam-VirtualBox:~/Desktop$ uname -r
5.6.9
sam@sam-VirtualBox:~/Desktop$ hostnamectl
  Static hostname: sam-VirtualBox
    Icon name: computer-vm
    Chassis: vm
      Machine ID: 89f3efe440094ec08d20858e90136040
        Boot ID: 7369b2dcc60d46709b4921c0973eb4e1
  Virtualization: oracle
Operating System: Ubuntu 20.04.1 LTS
      Kernel: Linux 5.6.9
Architecture: x86-64
sam@sam-VirtualBox:~/Desktop$
```

寫一個 C program :test_syscall.c 來測試 syscall

完成 coding 並編譯:gcc test_syscall.c

執行 test_syscall 將 kernel log 印在 terminal 上: dmesg -f kern -l emerg

結果 system call 成功回傳 0 並且印出 kernel log 在 terminal 上！

測試成功！！

```
Open + test_syscall.c ~/Desktop Save
1 #include <sys/syscall.h>
2 #include <sys/types.h>
3 #include <linux/kernel.h>
4 #include <unistd.h>
5 #include <stdio.h>
6 int main()
7 {
8     long int i = syscall(335);
9     printf("System call sys_60947038S return %ld\n",i );
10    return 0;
11}
```

```
sam@sam-VirtualBox:~/Desktop$ vim test_syscall.c
sam@sam-VirtualBox:~/Desktop$ gcc test_syscall.c
sam@sam-VirtualBox:~/Desktop$ ./test_syscall
System call sys_60947038S return 0
sam@sam-VirtualBox:~/Desktop$ dmesg -f kern -l emerg
[ 1365.548324] Hi,Sam 60947038S HW2-1 finished
[ 1535.820660] Hi,Sam 60947038S HW2-1 finished
[ 1750.692317] Hi,Sam 60947038S HW2-1 finished
[ 1953.435967] Hi,Sam 60947038S HW2-1 finished
sam@sam-VirtualBox:~/Desktop$
```

Hw2-2 Implementation Steps:

2-2 的部分因為環境問題所以改用 linux-5.0.0，且 2-1 的部份必須也在 linux-5.0.0 上重做一次。

Step1:新增一個標頭檔

至 linux-5.0.0/include/linux 建立 prinfo.h 並打上程式碼

```
1 ifndef _LINUX_PRINFO_H
2 define _LINUX_PRINFO_H
3
4 ifndef __KERNEL__
5 //if the header files are from loadable kernel modules.
6 #include <linux/types.h>
7 #include <linux/pid.h>
8
9 else
10 //else the header files are from the user-space programs.
11
12 #include <asm/unistd.h>
13 #include <sys/types.h>
14
15 endif
16 struct prinfo{
17     long state; /*current process state*/
18     long nice; /*process nice value*/
19     pid_t pid; /*process id*/
20     pid_t parent_pid; /*process id of parent*/
21     pid_t youngest_child_pid; /*process of youngest child*/
22     unsigned long start_time; /*process start time*/
23     long user_time; /*CPU time for user mode*/
24     long sys_time; /*CPU time for kernel mode*/
25     long uid; /*user id for process owner*/
26     char comm[16]; /*name of the program executed */
27 };
28
29 endif
```

Step2:建立 syscall 程式碼、以及 Makefile 到 info 目錄下:

至 linux-5.0.0/info/info.c，輸入程式碼，程式碼請見 info.c

至 linux-5.0.0/info/Makefile，輸入一行: obj-y := info.o

回到 linux-5.0.0/Makefile 中，找到 1017 行的位置加入...../info

(hint:注意空白)，目的是告知系統 syscall 在此目錄下

```
ifeq ($(_KBUILD_EXTMOD),)
core-y      += kernel/certs/mm/fs/ipc/security/crypto/block/info/
vmlinux-dirs := $(patsubst %,%,$(filter %, $(init-y) $(init-m) \
$(core-y) $(core-m) $(drivers-y) $(drivers-m) \
$(net-y) $(net-m) $(libs-y) $(libs-m) $(virt-y)))
vmlinux-alldirs := $(sort $(vmlinux-dirs) Documentation \
$(patsubst %,%,$(filter %, $(init-) $(core-) \
$(drivers-) $(net-) $(libs-) $(virt-))))
```

至 linux-5.0.0/include/linux/syscalls.h，定義系統呼叫標頭檔

```
#include <linux/personality.h>
#include <linux/prinfo.h>
#include <trace/syscall.h>

asmlinkage long sys_pidfd_getfd(int pidfd, int fd, unsigned int flags);

asmlinkage long sys_60947038S(void);
asmlinkage long sys_getPrinfo(struct prinfo *info);/*
*/
/* Architecture-specific system calls
*/
```

至 linux-5.0.0/arch/x86/entry/syscalls/syscall_64.tbl，設定 syscall table

333	common	io_pgetevents	__x64_sys_io_pgetevents
334	common	rseq	__x64_sys_rseq
335	common	hi_609470385	sys_609470385
336	common	info	sys_getPrinfo

Step3: 編譯 kernel 並 reboot

Step4: Demonstration 測試 system call

寫一個 C program: test_syscall2.c 測試 sys_getPrinfo

```
1 #include "prinfo.h"
2 #include <linux/unistd.h>
3 #include <stdio.h>
4
5 long getProcessInfo(struct prinfo *info){
6     return syscall(336,info);
7 }
8
9 int main(void){
10     struct prinfo info;
11     long num;
12     int f_name;
13     for(int i=0;i<=30000;i++)
14     {
15         for(int j=0;j<=1000;j++)
16             f_name=f_name+1;
17     }
18     num = getProcessInfo(&info);
19     printf("state:%ld\n",info.state);
20     printf("nice:%d\n",info.nice);
21     printf("pid:%d\n",info.pid);
22     printf("parent pid:%d\n",info.parent_pid);
23     printf("pid youngest child:%d\n",info.youngest_child_pid);
24     printf("start time:%ld\n",info.start_time);
25     printf("user time:%ld\n",info.user_time);
26     printf("system time:%ld\n",info.sys_time);
27     printf("user id:%ld\n",info.uid);
28     printf("name of program:%s\n",info.comm);
29 }
30 }
```

測試成功！

gcc test_syscall2.c

./a.out

dmesg

UserSpace result: (./a.out)

Kernel log result: (dmesg)

```
[ 314.953277] Current state of process: 0
[ 314.953279] Process nice value: -817180573
[ 314.953280] Process id: 2591
[ 314.953281] Process id of parent: 2452
[ 314.953281] Pid of youngest child: 2591
[ 314.953282] Start time value: 315005906486
[ 314.953282] CPU time spent in user mode: 72000000
[ 314.953282] CPU time spend in system mode: 4000000
[ 314.953283] User id of process owner: 1000
[ 314.953284] Name of program executed: a.out
```

Comment:

Hw2-1:剛開始做新增一個簡單的 system call 的題目時，對於很多步驟都不太了解其內容、目的及原因，所以導致做得非常混亂且編譯一直沒有辦法過關，甚至也不知道編譯失敗要執行 make clean/make mrproper，清除那些編譯過程會出現的多餘的檔案以及一些.o 檔，才不會重複出錯。後來在網路上研讀了”鳥哥 Linux”學到了很多，才慢慢的能一步一步完成功課。Hw2-2:對我來說真的比較困難一點，因為我對 C program 比較不熟悉，所以上面的程式碼我做了好長一段時間的查詢、debugging 才完成。成功完成後真的覺得學到了很多，對於客製 system call 的做法有很深刻的了解，也更清楚了解 kernel 底層對於 system call 是如何運作的。

Reference:

<https://it.livekn.com/2013/01/kernel-system-call.html>

<https://medium.com/anubhav-shrimai/adding-a-hello-world-system-call-to-linux-kernel-dad32875872>

<https://linuxconfig.org/introduction-to-the-linux-kernel-log-levels>

<https://stackoverflow.com/questions/59994792/error-conflicting-types-when-trying-to-make-a-simple-syscall>

<https://dev.to/stalim/comment/lp3f>

<https://stackoverflow.com/questions/33398815/ubuntu-15-10-kernel-4-2-hello-world-kernel-module-makefile-error>

<https://huenlil.pixnet.net/blog/post/23507650>