Lab 4

Members: Fangwen Liao (3439869) | Yijin Wang (3476217)

Exercise 1

(a) According to man page of strace, it is used to intercept and record system calls made by a process. Run the following command in terminal and record the result in fork.log:

```
fangwenliao@debian:~/Downloads/OS_Lab2/fork$ strace -o fork.log ./fork
Main process PID: 1047
Child PID: 1048
Child PID: 1049
Child PID: 1050
Child PID: 1051
Child PID: 1053
Child PID: 1054
Child PID: 1052
Press ENTER key to Continue

Process 1051 ended

Process 1048 ended

Process 1049 ended

Process 1050 ended

Process 1053 ended

Process 1054 ended

Process 1052 ended

Process 1047 ended
```

The content in the log file is below:

```
  1 execve("./fork", ["./fork"], [/* 20 vars */]) = 0
  2 brk(NULL)                               = 0x62c000
  3 access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or
directory)
  4 mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb77a2000
  5 access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or
directory)
  6 open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
  7 fstat64(3, {st_mode=S_IFREG|0644, st_size=99587, ...}) = 0
```

```
  8 mmap2(NULL, 99587, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7789000
  9 close(3)                                = 0
 10 access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or
directory)
 11 open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
 12 read(3,
"\177ELF\1\1\1\3\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\0\204\1\0004\0\0\0"..., 512) =
512
 13 fstat64(3, {st_mode=S_IFREG|0755, st_size=1791908, ...}) = 0
 14 mmap2(NULL, 1800700, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0xb75d1000
 15 mprotect(0xb7782000, 4096, PROT_NONE)   = 0
 16 mmap2(0xb7783000, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b1000) = 0xb7783000
 17 mmap2(0xb7786000, 10748, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb7786000
 18 close(3)                                = 0
 19 set_thread_area({entry_number:-1, base_addr:0xb77a3100, limit:1048575,
seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages    :1,
seg_not_present:0, useable:1}) = 0 (entry_number:6)
 20 mprotect(0xb7783000, 8192, PROT_READ)   = 0
 21 mprotect(0x4aa000, 4096, PROT_READ)     = 0
 22 mprotect(0xb77cb000, 4096, PROT_READ)   = 0
 23 munmap(0xb7789000, 99587)               = 0
 24 getpid()                                = 1047
 25 fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
 26 brk(NULL)                               = 0x62c000
 27 brk(0x64d000)                           = 0x64d000
 28 write(1, "Main process PID: 1047\n", 23) = 23
 29 clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0xb77a3168) = 1048
 30 clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0xb77a3168) = 1051
 31 clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0xb77a3168) = 1054
 32 getpid()                                = 1047
 33 nanosleep({tv_sec=1, tv_nsec=0}, 0xbfca0ae8) = 0
 34 write(1, "Press ENTER key to Continue\n", 28) = 28
 35 nanosleep({tv_sec=1, tv_nsec=0}, 0xbfca0ae8) = 0
 36 fstat64(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
 37 read(0, 0x62c410, 1024)                 = ? ERESTARTSYS (To be restarted if
SA_RESTART is set)
 38 --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1051, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
 39 read(0, 0x62c410, 1024)                 = ? ERESTARTSYS (To be restarted if
SA_RESTART is set)
 40 --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1048, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
 41 read(0, 0x62c410, 1024)                 = ? ERESTARTSYS (To be restarted if
SA_RESTART is set)
 42 --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1054, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
 43 read(0, "\n", 1024)                      = 1
 44 write(1, "Process 1047 ended\n", 19)    = 19
```

```
45 exit_group(0)                          = ?
46 +++ exited with 0 +++
```

In fork.c line 9 calls clone(), the correspoding line in log is line 29: clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0xb77a3 168) = 1048

In fork.c line 16 calls getpid() and write(), this is called by child processes, thus the trace are not found in this log file (to increase readability of log file, the -f option is not used, however when switched on those calls will be recorded.

In fork.c line 18 calls nanosleep(), the correspoding line in log is line 33: nanosleep({tv_sec=1, tv_nsec=0}, 0xbfca0ae8) = 0

In fork.c line 22 also calls nanosleep(), the correspoding line in log is line 35: nanosleep({tv_sec=1, tv_nsec=0}, 0xbfca0ae8) = 0

In fork.c line 24 calls read(), and since it is getchar() so the correspoding line in log is line 43, not the ones before: read(0, "\n", 1024)

(b) The clone() in line 9: According to man page of fork(2), at the time of fork the child has the same memory content as parent, so they excute the same stack, so the child_stack parameter should be NULL, according to man page of clone(2), CLONE_CHILD_CLEARCTID clear the child thread ID when child exits, CLONE_CHILD_SETTID store the child tid, SIGCHLD sent this signal to parent when child dies, child_tidptr stores the child tid in parents memory. Return value is the childs pid if success.

The getpid() in line 16: According to man page of getpid(2), it returns the pid of the calling process.

The write() in line 16: to get the log, another strace with -f is runned and one write() call of a child is picked out as below, 1277 write(1, "Child PID: 1277\n", 16 <unfinished ...> 1277 <... write resumed>) = 16 According to man page of write(2), the first argument is file discriptor which is a reference number of the operating file, in this case the fd is 1, which is the one for standard output, the second argument is the start pointer of the content to write, and the third argument is the count of how many bytes to write from the start pointer. The return value is the number of bytes written, which is the third argument, when success.

The nanosleep() in line 18 and 22: According to man page in nanosleep(2), this call is used to suspend the process for a time specified in the argument, which is a strucure consists of second and nanosecond, which provides high precision. Return value is 0 when sucessful.

The read in line 24: Arrording to man page of read(2), the first argument is also a file descriptor, in this case is 0, which is the one for standard input, the second argument is the content to read and the third one is the count number of bytes to read. Return value is the number of byte read, \n count as one byte.

Exercise 2

(a)

```
#include <fcntl.h>
#include <unistd.h>
```

```c
int main() {
    int fd = open("./output.txt", O_RDWR|O_CREAT|O_APPEND, S_IRWXU);
    write(fd, "Write something!\n", 17);
    close(fd);
}
```

First included header is fcntl.h, which define the O_APPEND O_RDWR and O_CREAT needed in open(), the second included header is unistd.h, which defines close() and write().

The first argument of open() is the path of the file to open, the second are flags: one of the access mode flag must included, in our case is read/write. O_RDWR, O_APPEND is used to write content at the end of the file, O_CREAT is used to create the file when the file doesn't exist, when using this flag a mode must be specified, in our case the S_IRWXU is used, which grants user read, write and execution access. A int which indicates the file descriptor is returned.

write() is discussed before.

close() according to man page of close(2) takes one argument which is the file descriptor, return 0 when success. The same strace command is used as before and the log file is below:

```
  1 execve("./a.out", ["./a.out"], [/* 20 vars */]) = 0
  2 brk(NULL)                               = 0x18c7000
  3 access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or
directory)
  4 mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb7771000
  5 access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or
directory)
  6 open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
  7 fstat64(3, {st_mode=S_IFREG|0644, st_size=99587, ...}) = 0
  8 mmap2(NULL, 99587, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7758000
  9 close(3)                                = 0
 10 access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or
directory)
 11 open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
 12 read(3,
"\177ELF\1\1\1\3\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\0\204\1\0004\0\0\0"..., 512) =
512
 13 fstat64(3, {st_mode=S_IFREG|0755, st_size=1791908, ...}) = 0
 14 mmap2(NULL, 1800700, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0xb75a0000
 15 mprotect(0xb7751000, 4096, PROT_NONE)   = 0
 16 mmap2(0xb7752000, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b1000) = 0xb7752000
 17 mmap2(0xb7755000, 10748, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb7755000
 18 close(3)                                = 0
 19 set_thread_area({entry_number:-1, base_addr:0xb7772100, limit:1048575,
seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages    :1,
seg_not_present:0, useable:1}) = 0 (entry_number:6)
 20 mprotect(0xb7752000, 8192, PROT_READ)   = 0
```

```
 21 mprotect(0x46b000, 4096, PROT_READ)     = 0
 22 mprotect(0xb779a000, 4096, PROT_READ)   = 0
 23 munmap(0xb7758000, 99587)               = 0
 24 open("./output.txt", O_RDWR|O_CREAT|O_APPEND, 0700) = 3
 25 write(3, "Write something!\n", 17)      = 17
 26 close(3)                                = 0
 27 exit_group(0)                           = ?
 28 +++ exited with 0 +++
```

As we can see above, from line 24 to line 26 the system calls are sucessfull.

(b) The argument in open() are explained before, it returns fd = 3, which is after the fd of standard error = 2. The argument and return value of write() is explained before. The argument and return value of close() is explained before.

(c) User mode can't access dirctly to hardware devices, only kernel mode can, thus system call is needed. System calls add a layer of protection to the system, so the user can't easily crash the system.

Exercise 3

(a) Run the follwing command in termimal:

```
fangwenliao@debian:~$ watch -n.1 "cat /proc/interrupts"
```

```
Every 0.1s: cat /proc/interrupts
debian: Tue Dec  7 23:03:48 2021

           CPU0       CPU1       CPU2       CPU3
  0:         35          0          0          1   IO-APIC   2-edge      timer
  1:          1          0          0          9   IO-APIC   1-edge      i8042
  8:          0          0          0          0   IO-APIC   8-edge      rtc0
  9:          0          0          0          2   IO-APIC   9-fasteoi   acpi
 12:          0          0          0        162   IO-APIC  12-edge      i8042
 14:          0          0          0          0   IO-APIC  14-edge      ata_piix
 15:          0          0          0        469   IO-APIC  15-edge      ata_piix
 16:          0       3993          0         62   IO-APIC  16-fasteoi   enp0s8
 18:          0          0          0          0   IO-APIC  18-fasteoi   vmwgfx
 19:          0          4        173          7   IO-APIC  19-fasteoi   enp0s3
 20:          0          0          0        193   IO-APIC  20-fasteoi   vboxguest
 21:          0          0          0       8722   IO-APIC  21-fasteoi
ahci[0000:00:0d.0], snd_intel8x0
 22:          0          0          0         27   IO-APIC  22-fasteoi
ohci_hcd:usb1
NMI:          0          0          0          0   Non-maskable interrupts
LOC:      14967      10315       6945       8643   Local timer interrupts
SPU:          0          0          0          0   Spurious interrupts
PMI:          0          0          0          0   Performance monitoring
interrupts
IWI:          0          0          0          0   IRQ work interrupts
```

```
RTR:          0          0          0          0   APIC ICR read retries
RES:       5885       3888       3884       3227   Rescheduling interrupts
CAL:       2486       2380       3062       1133   Function call interrupts
TLB:        456        464        343        426   TLB shootdowns
TRM:          0          0          0          0   Thermal event interrupts
THR:          0          0          0          0   Threshold APIC interrupts
DFR:          0          0          0          0   Deferred Error APIC interrupts
MCE:          0          0          0          0   Machine check exceptions
MCP:          2          2          2          2   Machine check polls
ERR:          0
MIS:          0
PIN:          0          0          0          0   Posted-interrupt notification
event
PIW:          0          0          0          0   Posted-interrupt wakeup event
```

Each column has its meanings: The first one is IRQ number, the next four are interrupt numbers of each CPU, the next is the type of interrupts and the last is the device that caused the interrupt.

IRQ_0 timer

(a)A software clock, update time and date, update time passed after system start. When a process is running, the system timer record its running time, and when it exceed its allocated time, the timer will send a interrupt to preempt it.

IRQ_1 i8042

(a) Keyboard, Handels keyboard input. (b) When press and release a key, the system will call two interruput to handel the input content.

IRQ_8 rtc0

(a) Real-time clock, Another hardware clock, (b) according to man page of rtc(4), it record the wall clock time, its has own backup power when the machine is turned off, it is not system lock. It will generate interrupts when a previously set alarm time is reached.

IRQ 9 acpi

(a)ACPI stands for advanced Configuration and Power Interface. (b)It provide a open standard for OS to perform power management. It defines a hardware abstraction interface between device firmware, computer hardware and OS. It raise interrupts when a General purpose event happens such as plug the AC adapter or close the lid of laptop.

IRQ 12 i8042

(a) The mouse controller, like in IRQ 1. (b) When the mouse moves, it will send interrupts.

IRQ 14 ata_piix

（a）Hard drive controller, Intel PATA/SATA controllers. (b) When user read or write files in hardisk, it will send interrupts.

IRQ 15 ata_piix

(a）Second hard drive controller. (b) Just like IRQ_14.

IRQ 16 enp0s8

(a) Ehternet network perepherial, (b) en means ethernet, p0 is bus number 0 and s3 is slot number 8, it is a example of name scheme for a network device. It will generate interrupts when the system need to send package or to deal with received package.

IRQ 18 vmwgfx

(a) A graphic driver for linux from VMware, because we use the VMSVGA in our virtual box setting. If we turn to use VBoxSVGA, this controller will become vboxvideo. (b) It provides an acceleration architecure for 2d and 3d When something happens in the GPU it will raise a interrupt.

IRQ 19 enp0s3

Another ethernet interface like the one in IRQ 16, this one is in slot 3.

IRQ 20 vboxguest

(1) This comes from VBox Guest Addtion installed in VM, (b) It improves the usability and performance of guest operating system runs in virtualbox. When the guest need to synchronize time with the host, it will call a interuppt.

IRQ 21 ahci[0000:00:0d.0], snd_intel8x0

(1) The sound controller. (2) It is used to manage the sound in linux system, includng generating sound from a device like microphone to application and delivering sound from an application to an output device like earphone.

IRQ 22 ohci_hcd:usb1

(1) USB interface. (2) Enables a USB hardware to communicate with a host controller driver in software. When a USB device is plugged in or out it will raise an interrupt.