CS5250 - Assignment 2

Daniel Alfred - A0184588J

Part A

We could mount it to /sys/kernel/tracing, by using `mount -t tracefs nodev /sys/kernel/tracing`

There are a couple of parameters in the form of files that we could change in the folder.

```
daniel@tpad:~$ sudo !!
sudo ls /sys/kernel/tracing
available_events            enabled_functions         max_graph_depth      set_event_pid       stack_trace_filter  trace_stat
available_filter_functions  error_log                 options              set_ftrace_filter   synthetic_events    tracing_cpumask
available_tracers           events                    per_cpu              set_ftrace_notrace  timestamp_mode      tracing_max_latency
buffer_percent              free_buffer               printk_formats       set_ftrace_pid      trace               tracing_on
buffer_size_kb              function_profile_enabled  README               set_graph_function  trace_clock         tracing_thresh
buffer_total_size_kb        hwlat_detector            saved_cmdlines       set_graph_notrace   trace_marker        uprobe_events
current_tracer              instances                 saved_cmdlines_size  snapshot            trace_marker_raw    uprobe_profile
dynamic_events              kprobe_events             saved_tgids          stack_max_size      trace_options
dyn_ftrace_total_info       kprobe_profile            set_event            stack_trace         trace_pipe
```

We could set the *current_tracer* to check what we are currently tracing. By default, it should have “nop” inside where it means there is nothing to trace. To check what we can trace, we could see the contents of *available_tracers*. Then we can echo what we want to trace to the *current_tracer* file.

There is a *tracing_on* file. This basically the file that controls whether we start or stop the tracing. We set 1 on this file if we want to start tracing and set it to 0 if we want to stop. There are also commands *traceon* and *traceoff* which do literally what it says, turning the trace on or off. The command could be set in the *set_ftrace_filter* file.

The trace is stored in the *trace* file. As we can see to the screenshot below, this is what we get if we open the file.

echo function_graph > current_tracer
echo 1 > tracing_on
head trace

```
root@tpad:/sys/kernel/tracing# head trace
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 0)   0.201 us    |                    rcu_all_qs();
 0)   0.573 us    |                  } /* _cond_resched */
 0)   0.985 us    |                } /* mutex_lock */
 0)               |                tty_write_room() {
 0)               |                  pty_write_room() {
 0)   0.205 us    |                    tty_buffer_space_avail();
```

You can set *set_graph_function* to filter which function to trace. For instance, if you only wants to filter *__do_fault*, we can

echo __do_fault > set_graph_function

We can limit the depth of the trace, by using

echo 2 > max_graph_depth

Part B

I realize that printk is printing to the kernel while printf is printing to file descriptor. So we will use printk because it's available. printf can specify a loglevel, however the kernel uses loglevel to decide whether to print the message to the console.

I add printmsg.c file which contains the code in the assignment pdf. Then I add to the table in arch/x86/entry/syscalls/syscall_64.tbl

```
600     common  printmsg                 sys_printmsg
```

I add this line which allows us to call printmsg in both x86 and x32 with the number code of 600. Number 548+ are free to use.

Then after reinstalling the kernel, I reboot.

Task 4

1. No. 1
    a. clear the 'trace' file
        i. echo > trace
    b. *buffer_size_kb* will modify the number of entries that can be recorded.
        i. echo 10 > *buffer_size_kb*
        ii. This command will limit 10kilobytes for each CPU
    c.
```c
#include <linux/ftrace.h>

int main() {
    // trace printk will create a comment of
    // Hello World in the main() function.
    trace_printk("Hello World!");
}

```
        i. We can use dynamic trace to trace this specific function.
        ii. To run the shell, we need to do
            1. sudo su
            2. cd /sys/kernel/tracing
            3.
```
echo mmiotrace_printk > set_ftrace_filter
echo function > current_tracer
echo 1 > tracing_on
usleep 1
echo 0 > tracing_on
```
            4. run the shell script.

d. 

```
root@tpad:/sys/kernel/tracing# head -n 20 trace
# tracer: function
#
# entries-in-buffer/entries-written: 204854/35068654   #P:4
#
#                                _-----=> irqs-off
#                               / _----=> need-resched
#                              | / _---=> hardirq/softirq
#                              || / _--=> preempt-depth
#                              ||| /     delay
#           TASK-PID     CPU#  ||||   TIMESTAMP  FUNCTION
#              | |         |   ||||      |         |
          <...>-38132   [003] .... 37967.617275: __fdget <-ksys_ioctl
          <...>-38132   [003] .... 37967.617275: __fget_light <-__fdget
          <...>-38132   [003] .... 37967.617275: __fget <-__fget_light
          <...>-38132   [003] .... 37967.617275: security_file_ioctl <-ksys_ioctl
          <...>-38132   [003] .... 37967.617275: do_vfs_ioctl <-ksys_ioctl
          <...>-38132   [003] .... 37967.617275: VBoxDrvLinuxIOCtl_6_1_16 <-do_vfs_ioctl
          <...>-38132   [003] .... 37967.617275: supdrvIOCtlFast <-VBoxDrvLinuxIOCtl_6_1_16
          <...>-38132   [003] .... 37967.617275: VBoxHost_RTThreadNativeSelf <-0xffffa8c40567dc0b
          <...>-38132   [003] .... 37967.617275: VBoxHost_RTThreadPreemptDisable <-0xffffa8c40567dcc2
```

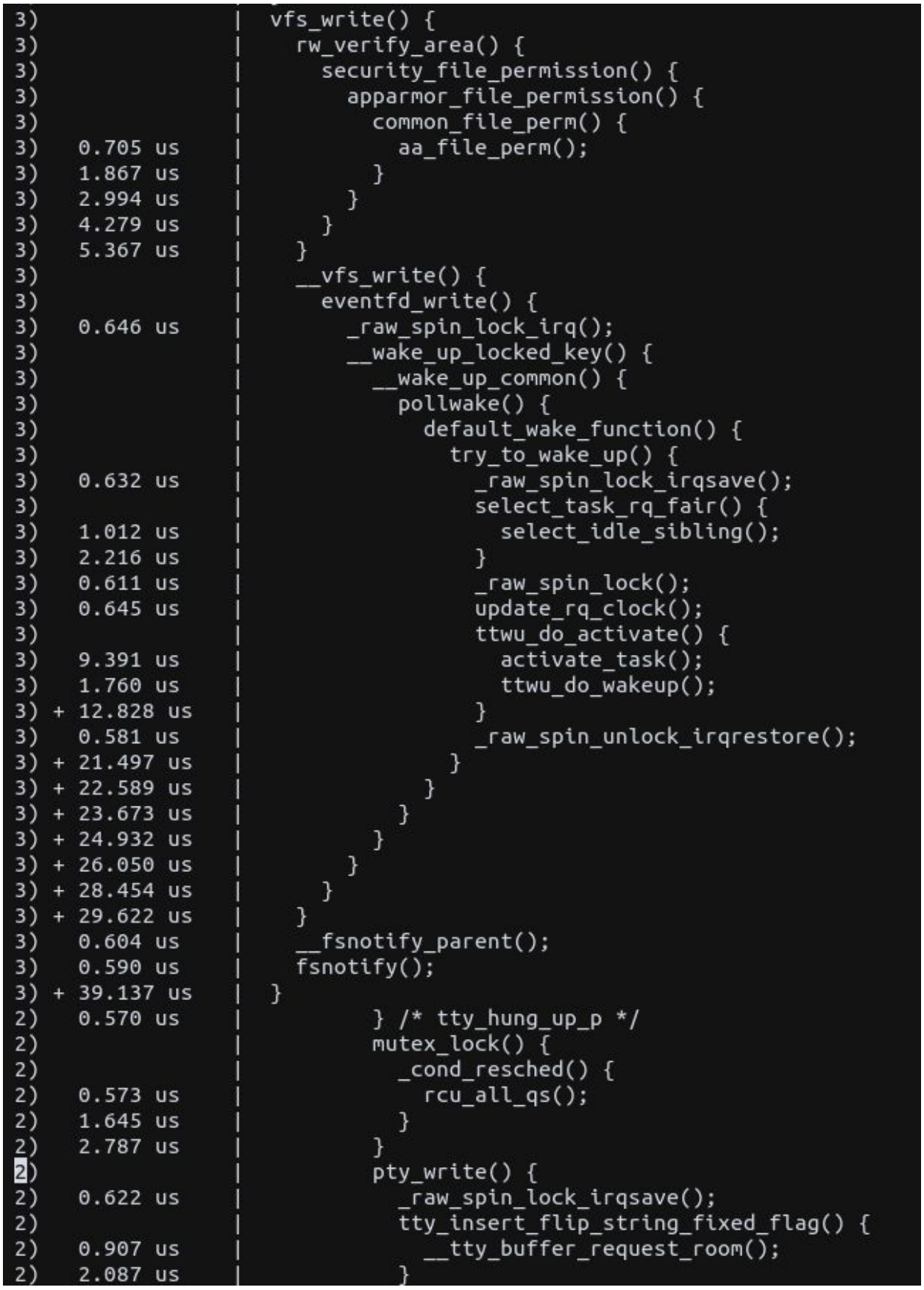
There are couple of things that we can see

i. There are 204854 entries in buffer and 35068654 events in total. So some of them is not in the trace file because it's gonna be too large.
ii. Task PID - The process ID that called the function
iii. The CPU ID that call the function
iv. Timestamp in seconds when the function was entered.
v. function name. There are 2 names, the first name is the name of the function and the second one is the parent function who called it. In the first example
    1. The function is __fdget,
    2. It's called from ksys_ioctl

```
root@tpad:/sys/kernel/tracing# echo vfs_open > set_graph_function
root@tpad:/sys/kernel/tracing# echo vfs_read >> set_graph_function
root@tpad:/sys/kernel/tracing# echo vfs_write >> set_graph_function
root@tpad:/sys/kernel/tracing# cat set_graph_function
vfs_read
vfs_open
vfs_write
root@tpad:/sys/kernel/tracing# echo > trace
root@tpad:/sys/kernel/tracing# cat max_graph_depth
2
root@tpad:/sys/kernel/tracing# echo 10 > max_graph_depth
root@tpad:/sys/kernel/tracing# echo 1 > tracing_on
root@tpad:/sys/kernel/tracing# wc -n trace
wc: invalid option -- 'n'
Try 'wc --help' for more information.
root@tpad:/sys/kernel/tracing# wc -l trace
780 trace
root@tpad:/sys/kernel/tracing# wc -l trace
828 trace
root@tpad:/sys/kernel/tracing# wc -l trace
903 trace
root@tpad:/sys/kernel/tracing# wc -l trace
847 trace
root@tpad:/sys/kernel/tracing# echo 0 > tracing_on
root@tpad:/sys/kernel/tracing# head -20 trace
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 1)   2.265 us    |          fsnotify();
 1)   9.876 us    |        } /* security_file_permission */
 1) + 11.248 us   |      } /* rw_verify_area */
 1)               |      __vfs_read() {
 1)               |        eventfd_read() {
 1)   0.612 us    |          _raw_spin_lock_irq();
 1)   1.986 us    |        }
 1)   3.330 us    |      }
 1)   0.602 us    |      __fsnotify_parent();
 1)   0.600 us    |      fsnotify();
 1) + 20.343 us   |    } /* vfs_read */
 ------------------------------------------
 1)   gdbus-2050   =>   gmain-1015
 ------------------------------------------

 1)               |    vfs_write() {
root@tpad:/sys/kernel/tracing#
```

2. We can echo those 3 functions to track the tracing. Then set the max_graph_depth to 10 so that it can trace deeper.



As we can see here, cpu 3 is doing vfs_write, then cpu 2 continue doing it's tracing as well. Here they're doing the tracing at the same time

3. No 3
    a. Code:

i.
```
daniel@monmouth:~/work/linux-5.10.6/kernel$ cat printmsg.c
#include <linux/kernel.h>
#include <linux/syscalls.h>

SYSCALL_DEFINE1(printmsg, int, i)
{
        printk(KERN_DEBUG "Hello! This is A0184588J from %d", i);
        return 1;
}
```

ii.
```
daniel@monmouth:~$ cat test.c
#include <linux/unistd.h>
#include <stdlib.h>
#define __NR_printmsg 600

int printmsg(int i) {
        return syscall(__NR_printmsg, i);
}

int main(int argc, char** argv)
{
        printmsg(atoi(argv[1]));
        return 0;
}
```

b.
```
daniel@monmouth:~/work/linux-5.10.6/arch/x86/entry/syscalls$ pwd
/home/daniel/work/linux-5.10.6/arch/x86/entry/syscalls
```

c.
```
daniel@monmouth:~$ ./a.out 9090
daniel@monmouth:~$ dmesg | tail
[    1.243045] systemd[1]: Mounted Kernel Trace File System.
[    1.252046] systemd[1]: modprobe@drm.service: Succeeded.
[    1.252440] systemd[1]: Finished Load Kernel Module drm.
[    1.258216] systemd[1]: Finished Uncomplicated firewall.
[    1.265964] systemd[1]: Started Journal Service.
[    1.269758] EXT4-fs (sda3): re-mounted. Opts: errors=remount-ro
[    1.576219] random: crng init done
[    2.856792] e2scrub_all (290) used greatest stack depth: 13432 bytes left
[    9.532707] hrtimer: interrupt took 4731021 ns
[   96.831746] Hello! This is A0184588J from 9090
```

d. I installed the tiny kernel which makes the installation faster, but I realized that the available_tracer are missing. The only available one are only blk and nop, where we need to set the current_tracer to function. I need more time to install the whole kernel.

Part C

1. Let there are 2 hash functions h1 and h2 for the bloom filter function. And there is 4 data a, b, c, d.

Variable	h1	h2
a	x	y
b	x	z

- If we insert a to bloom filter, the set (bit-vector) would be {x, y}.
- Then we insert b to bloom filter, the set would be {x, y, z}.
- If I remove b from the bloom filter, the set will be {y}.
- However, when I check a, x is no longer there, so it results a wrong value.

2. no 2
    a.
        i. 3: 8b 15 00 04 fa aa
    b. rodata stores constant data. One should expect string literals, and other constant values to reside there. It is marked as read-only (although usually resides in a read and executable segment). I think to move the pointer in rodata file.
    c. So instead of having relocation, we can have a GOT which stores all the offset and when queried, it will point to the correct location.
    While PLT will handle function calls. I read from this website and it's quite clear how they explain the usage of GOT and PLT.
    https://www.technovelty.org/linux/plt-and-got-the-key-to-code-sharing-and-dynamic-libraries.html