CS5250 – Advanced Operating Systems
Assignment 2

AY2018/2019 Semester 2
Name: SHI Jingli
Student ID: A0163341N

[Assignment Answer]

Task 1:

a. All trace data are recorded in ring buffer, which can be used in either an overwrite mode or in producer/consumer mode, what will cause the lose of most recent events.
Under producer / consumer mode, producer will stop writing to the buffer if producer were to fill up the buffer before the consumer could free up anything.
Under overwrite mode, producer will overwrite the older data if producer were to fill up the buffer before the consumer could free up anything, what will lead to lose the oldest events.

b. echo xxx > buffer_size_kb (xxx: size value)

c.
In kernel source code, ftrace.h is included inside futex.c file to use trace_printk function to output Hello World in ftrace.

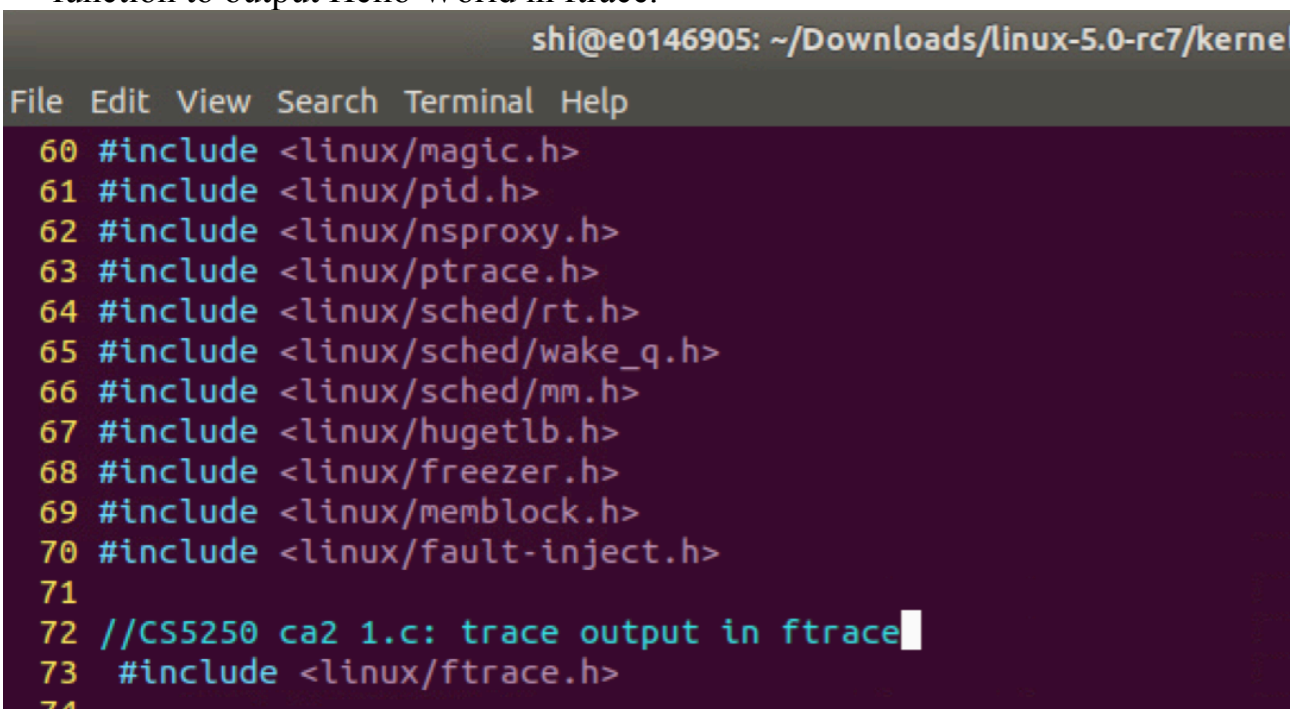

Figure 1. file futex.c

```
/*
 * Wake up waiters matching bitset queued on this futex (uaddr).
 */
static int
futex_wake(u32 __user *uaddr, unsigned int flags, int nr_wake, u32 bitset)
{
        trace_printk("Hello World1[A0163341N]!");
        struct futex_hash_bucket *hb;
        struct futex_q *this, *next;
        union futex_key key = FUTEX_KEY_INIT;
        int ret;
        DEFINE_WAKE_Q(wake_q);
```

Figure 2. function futex_wake

After the changes are saved in kernel source code, rebuild the kernel and install new kernel.

Reboot to new kernel, create directory “Debug” under “/home/shi/” then run “mount -t bugfs nodev debug/”.
Finally the following script is utilized to trace modified futex_wake function.

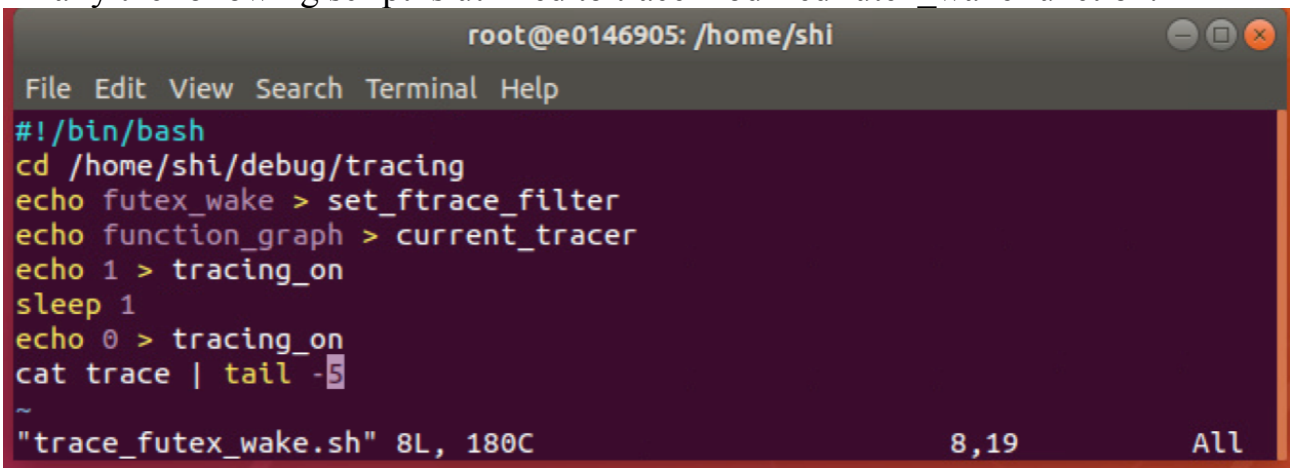

Figure 3. bash script trace_futex_wake.sh

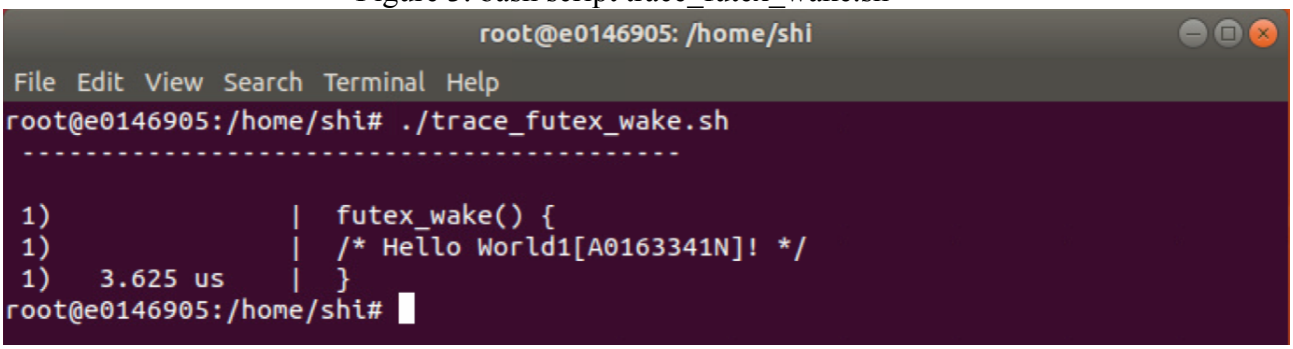

Figure 4. running script trace_futex_wake.sh

d. Function tracer

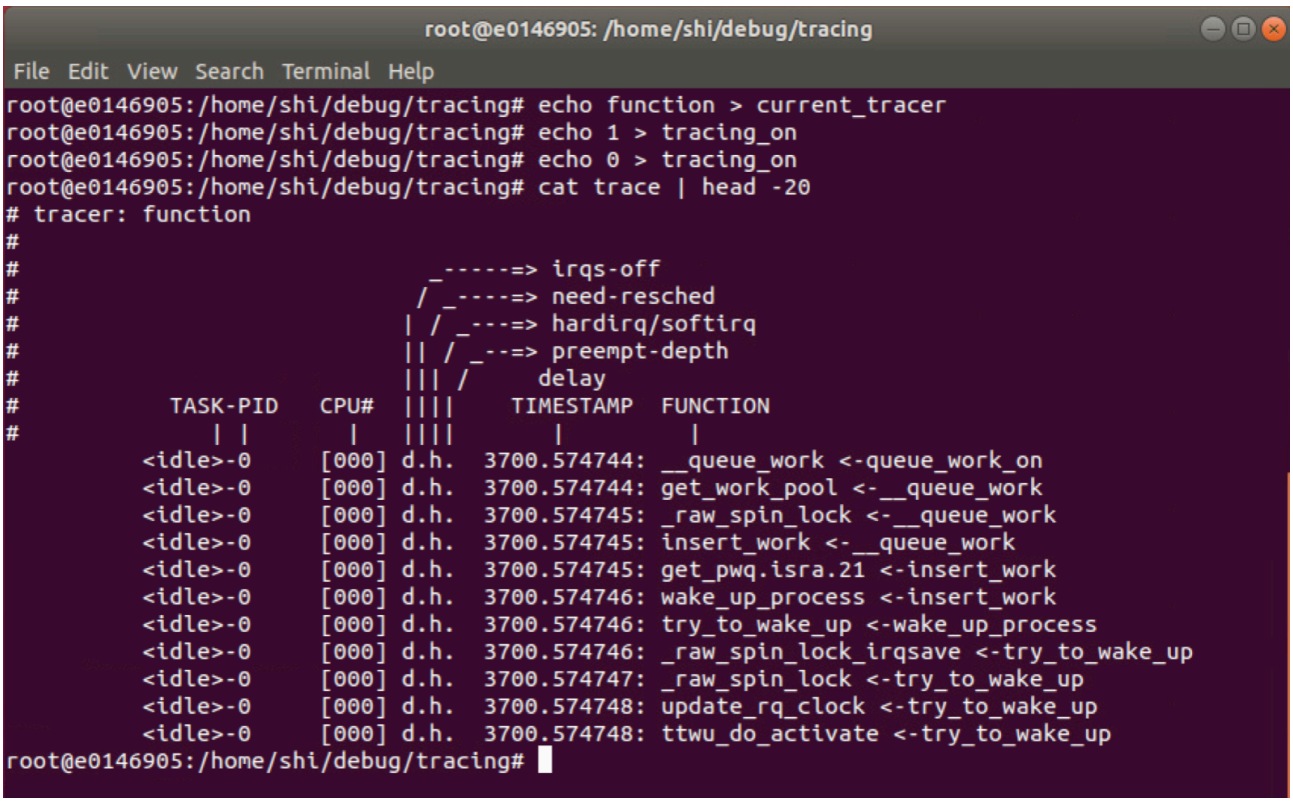

Figure 5. function tracer

the above output shows that the process name(<idle>), PID(0), the CPU ID([000]) on which trace is executed, the time-stamp <secs>.<usecs> format (3700.574744), the function name that was traced (__queue_work) and the parent function that called this function(queue_work_on).

Task 2. Use function_graph tracer

Run the following command to trace vfs_open, vfs_read and vfs_write.

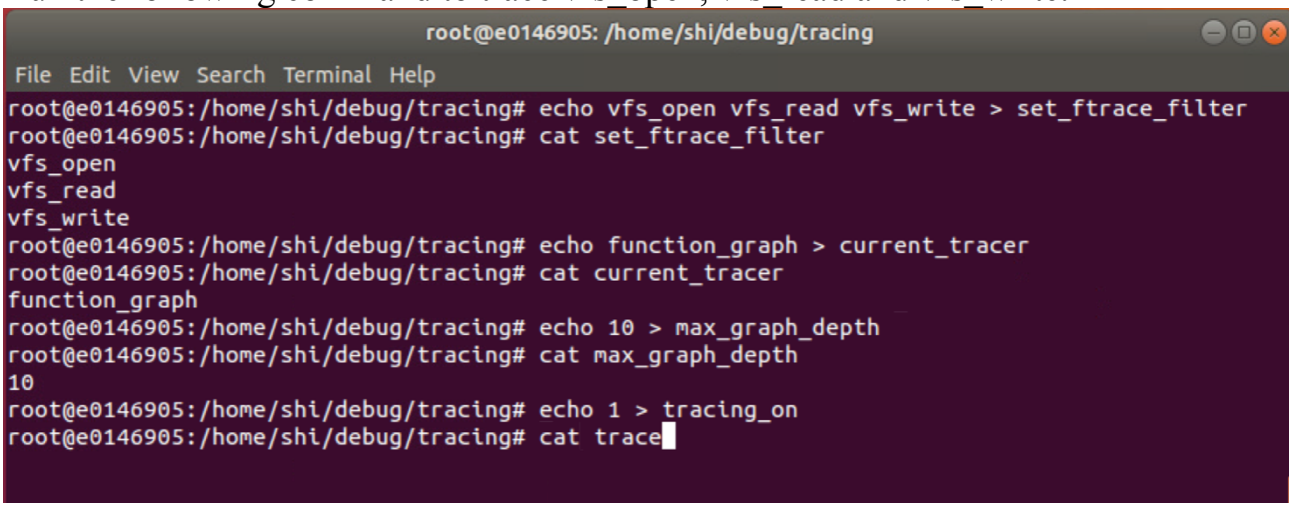

Figure 6 function_graph tracer

After configure ftrace, trace output is captured as following.

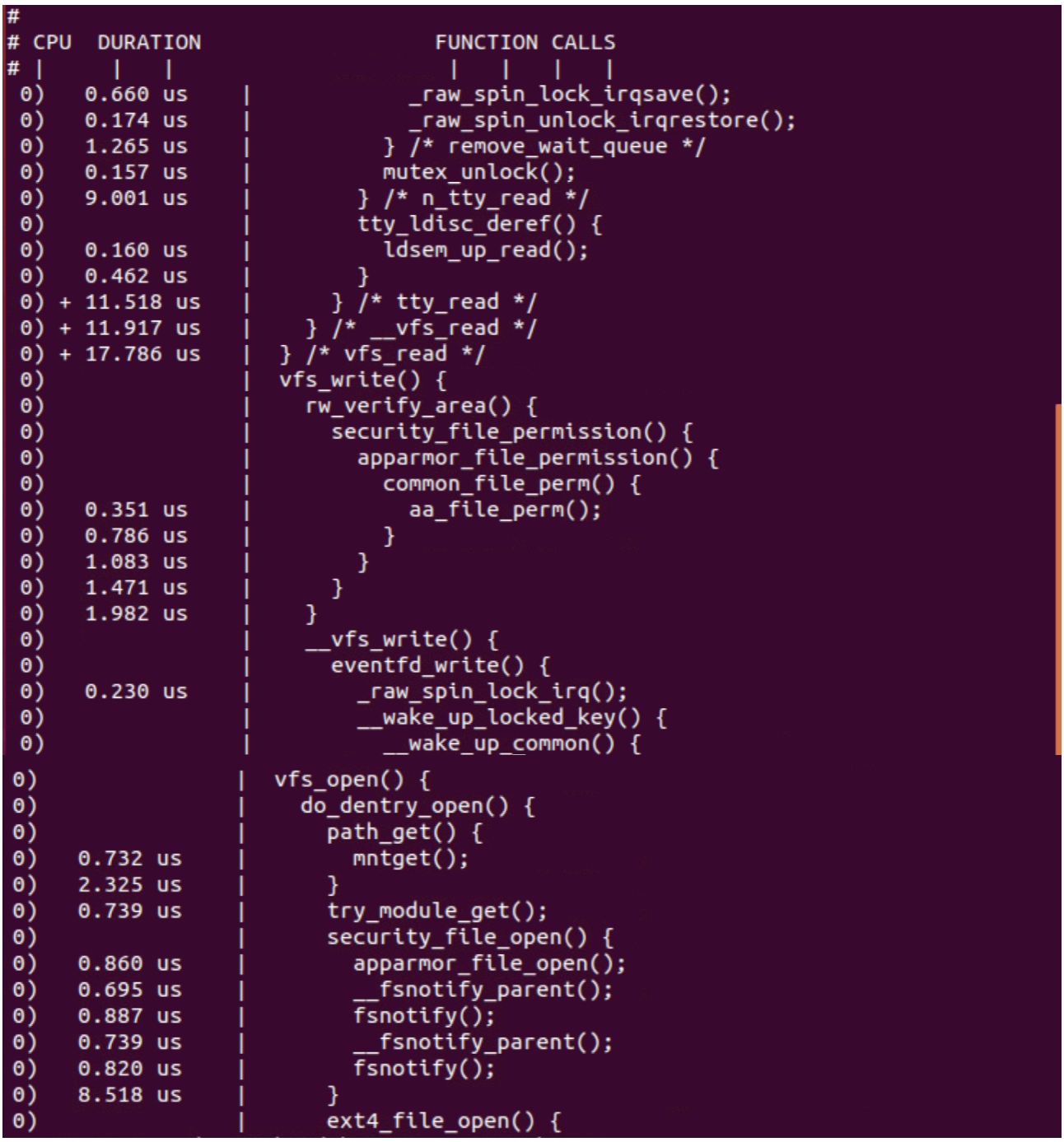

Figure 7 output of function_graph for vfs_open, vfs_read and vfs_write

Function graph tracer can trace both entry and exit of a function. “{” means start a function and “}” is at the end of a function. The function end with “;” is leaf function, which do not call other functions. Duration column shows the time spent in the corresponding function. The function graph tracer records the time the function was entered and exited and reports the difference as the duration. When the duration is greater than 10 ms, annotation “+” will be shown (e.g. the duration for last vfs_read is 17.786). If the duration is greater than 100 ms, annotation “!” will be shown.

Task 3:

a. Adding function into kernel

switch into linux source code directory and create a new directory, ***mkdir add2num***. switch into folder **add2num** and a **add2num.c** file.

shi@e0146905: ~/advOS/ca2

File Edit View Search Terminal Help

```
 1 #include <linux/kernel.h>       /* for printk */
 2 #include <linux/syscalls.h>     /* for SYSCALL_DEFINE3 macro */
 3
 4 /* add2num:     calculate a + b
 5  * a:           first number
 6  * b:           second number
 7  * from:        syscall number
 8  * */
 9 SYSCALL_DEFINE3(add2num, int, a, int, b, int, from)
10 {
11         printk(KERN_DEBUG "Hello! This is A0163341N from %d.\n",from);
12         printk(KERN_DEBUG "Calculating a(%d) + b(%d).\n",a,b);
13         return (a + b);
14 }
~
~
~
"add2num.c" 14L, 397C                                 1,1           All
```

Figure 8. file add2num.c

note: **printk** is a kernel level function, whereas **printf** is C Standard Library.

Switch to home directory, cd ***/home/shi/***, create c file **call_add2num.c** as below.

shi@e0146905: ~

File Edit View Search Terminal Help

```
 1 #include <stdio.h>
 2 #include <linux/unistd.h>
 3
 4 #define __NR_add2num    335
 5
 6 int add2num(int a, int b, int from)
 7 {
 8         return syscall(__NR_add2num,a,b,from);
 9 }
10
11 int main(int argc, char** argv)
12 {
13         int a = 1;
14         int b = 2;
15         int res = add2num(1,2,335);
16         printf("The result[%d+%d] from add2num is %d\n",a,b,res);
17         return 0;
"call_add2num.c" 18L, 312C                            1,1           Top
```

Figure 9. file call_add2num.c

b. system call table path:

<linux sourcecode>/arch/x86/entry/syscalls/syscall_64.tbl

the function add2num is added into system call table as below.

```
340 329     common  pkey_mprotect           __x64_sys_pkey_mprotect
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  add2num                 __x64_sys_add2num
347 #
```

Figure 10. system call table

under ***<linux sourcecode>/include/linux/syscalls.h***, add the following code.

```
1130 asmlinkage long sys_mmap_pgoff(unsigned long addr, unsigned long len,
1131                         unsigned long prot, unsigned long flags,
1132                         unsigned long fd, unsigned long pgoff);
1133 asmlinkage long sys_old_mmap(struct mmap_arg_struct __user *arg);
1134
1135
1136 /*
1137  * Not a real system call, but a placeholder for syscalls which are
1138  * not implemented -- see kernel/sys_ni.c
1139  */
1140 asmlinkage long sys_ni_syscall(void);
1141
1142
1143 /*
1144  * CS5250 CA2 [A013341N] - ADD new function to sys call
1145  * */
1146 asmlinkage int sys_add2num(int a, int b, int from);
1147
```

Figure 11. file syscalls.h

c.

switch to home directory, ***cd /home/shi***/, compile the c file using gcc, ***gcc call_add2num.c –o call_add2num*** , and given that no errors during compilation.

```
shi@e0146905: ~
File  Edit  View  Search  Terminal  Help
shi@e0146905:~$ gcc call_add2num.c  -o call_add2num
call_add2num.c: In function ‘add2num’:
call_add2num.c:8:9: warning: implicit declaration of function ‘syscall’; did yo
u mean ‘sscanf’? [-Wimplicit-function-declaration]
  return syscall(__NR_add2num,a,b,from);
         ^~~~~~~
         sscanf
shi@e0146905:~$
```

Figure 12. compile call_add2num.c using gcc

Run ***./call_add2num*** to check the program. if all steps are correct, the output will be the below printed onto the console.

```
shi@e0146905: ~
File  Edit  View  Search  Terminal  Help
shi@e0146905:~$ ./call_add2num
The result[1+2] from add2num is 3
shi@e0146905:~$
```

Figure 13. run call_add2num

all **printk** information will print to the kernel’s log, so we can use **dmesg** to check

if all messages have been successfully printed.

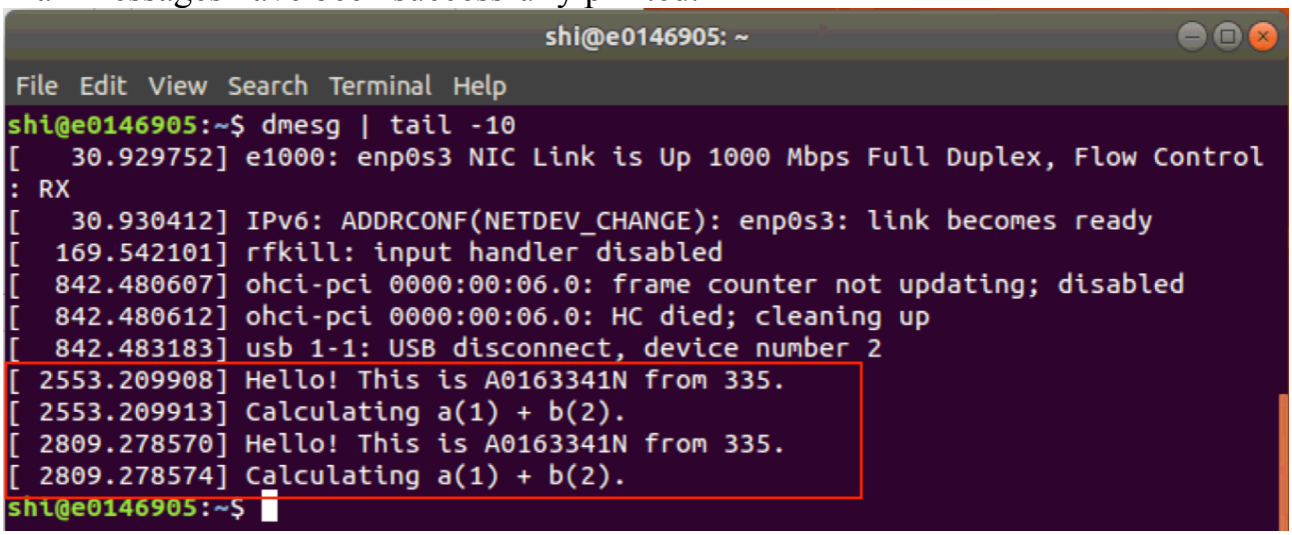

Figure 14. kernel log message output

call_add2num is executed twice, so we can find the message from **add2num** function twice.

d. use ftrace to trace add2num.

switch to home director, cd ***/home/shi/,*** create the following bash script **trace_add2num.sh** to trace system call **add2num**. **call_add2num** is the executable file built from question C.

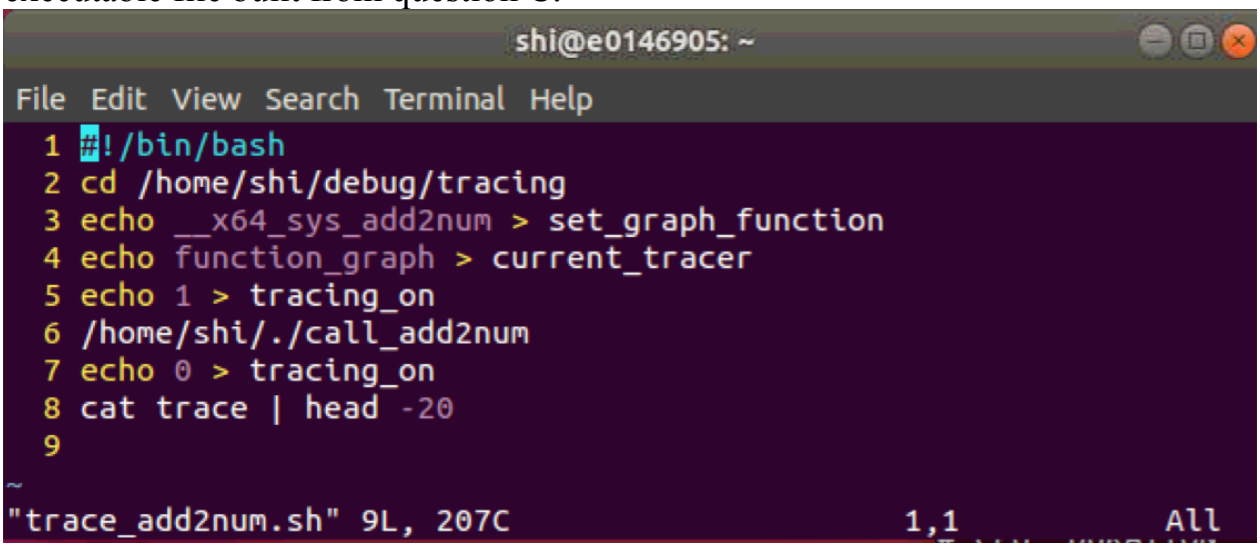

Figure 15. file trace_add2num.sh

Then run the script and the below message is printed on console.

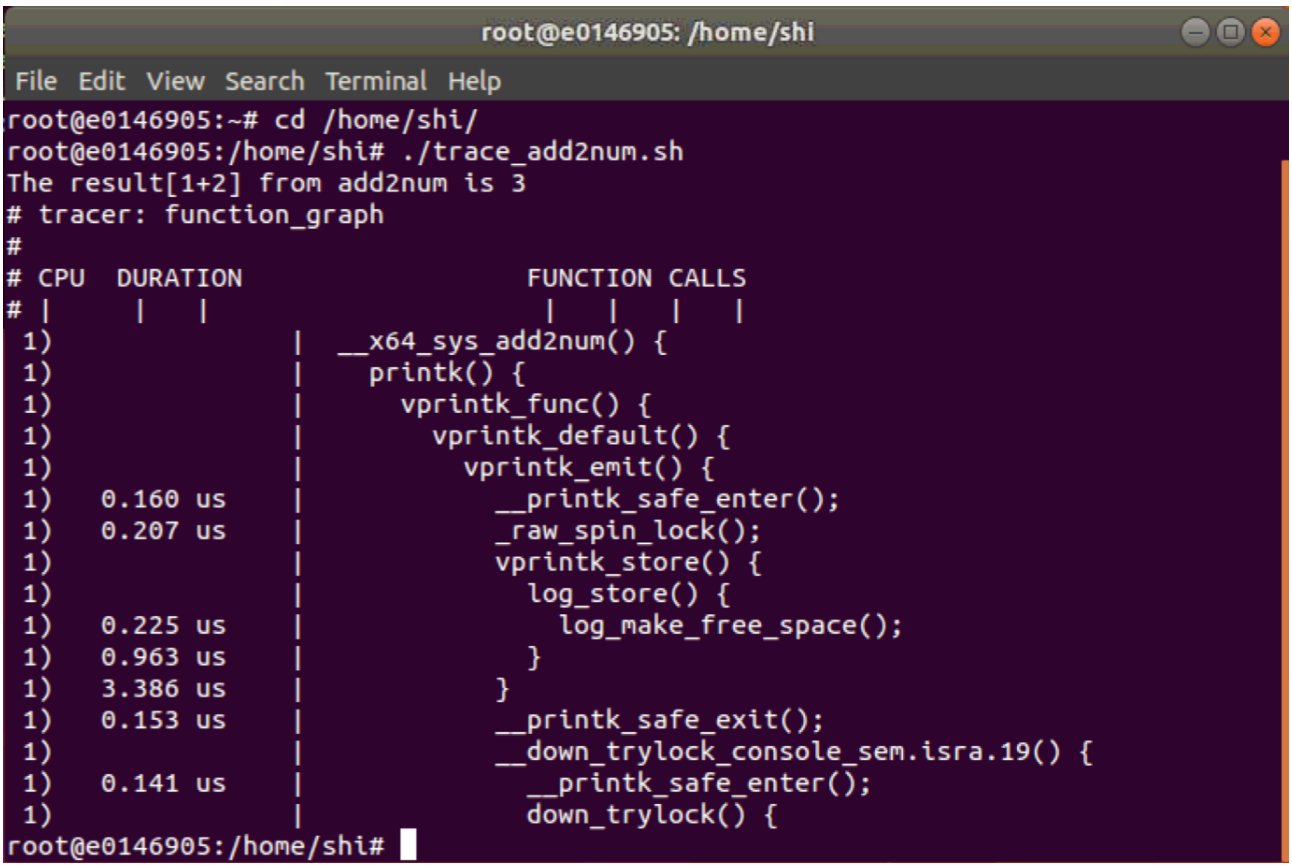

Figure 16. trace log of system call add2num

From above output, we know the first function called by add2num is printk. Other system functions are called and the executed time is shown as duration.

Part C: Additional exercise on Bloom filters.

1. **Why deletion is not supported in the bit-vector form of the Bloom filter?**
   When delete one element from Bloom filter, the corresponding bit 1s need to be reset to 0s directly. Maybe the bit is shared by other elements, so deletion may lead to false negative results for other elements.
   For example, three elements, (x,y,x) inside the Bloom filter set shown in figure 17.
   - Before delete element z from set, w maybe in the set(x,y,z).
   - After delete element z from set, some bit shared with element x and y are also reset to 0. When we check if w is in set(x,y) again, we find w is not in set(x,y). actually, w is in the set. The deletion of element z leads to the false negative.

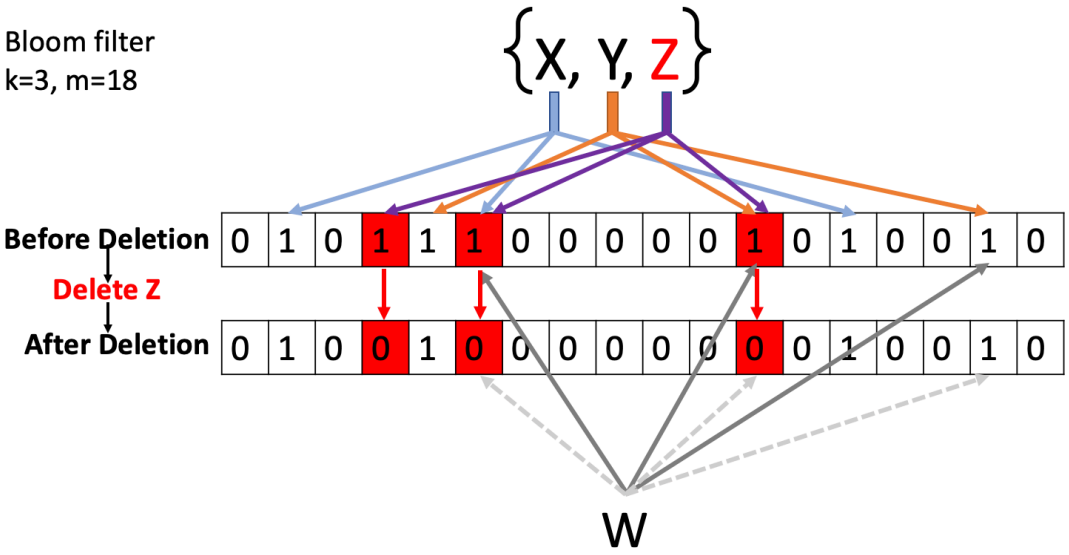

Figure 17. Bloom filter

2. Bloom filter supporting deletion

As shown in figure 18, we use byte to replace bit in one cell and the number of cell is counter. When one element is added into the set, the corresponding counter will be added by 1, and when one element is deleted from the set, the corresponding counter will be subtracted by 1. One limitation of such Bloom filter is that the maximum of one cell value is only 2^8-1=255.

Figure 18 and figure 19 will demonstrate that the deletion of element X4 will not lead to false negative.

In figure 18, w1 maybe in set(X1,X2,X3,X4) and w2 is not in the set. After delete element X4 shown in figure 19, the corresponding counter change to [2,3,1] from [3,4,2], whereas we still find W1 maybe in set(X1,X2,X3) and W2 is not in the set based on the new Bloom filter.

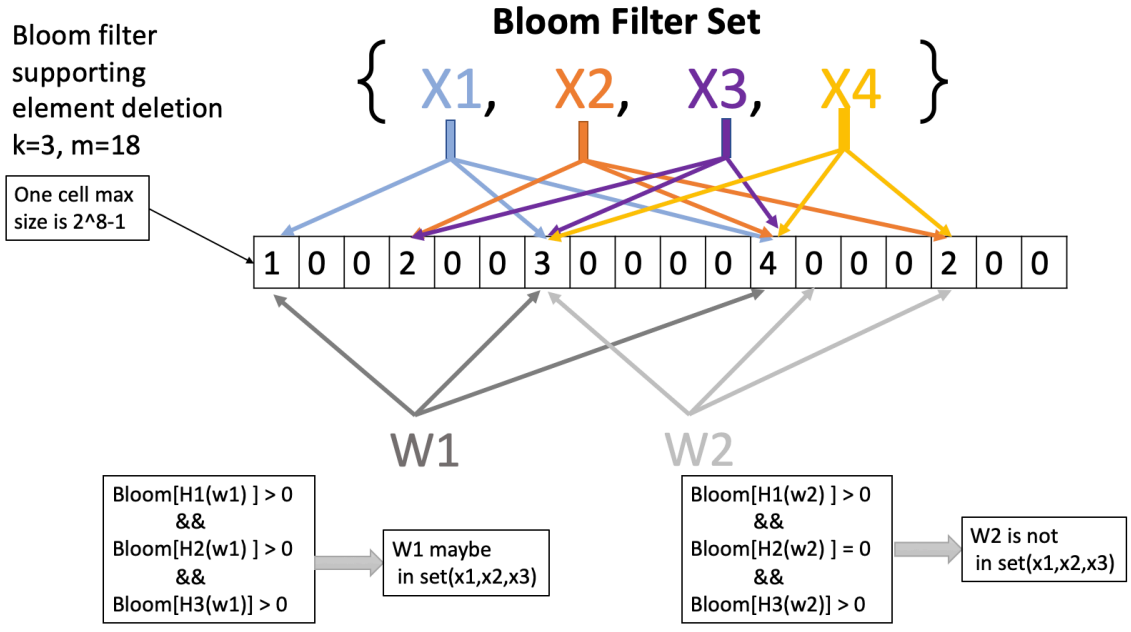

Figure 18. Bloom filter before delete X4

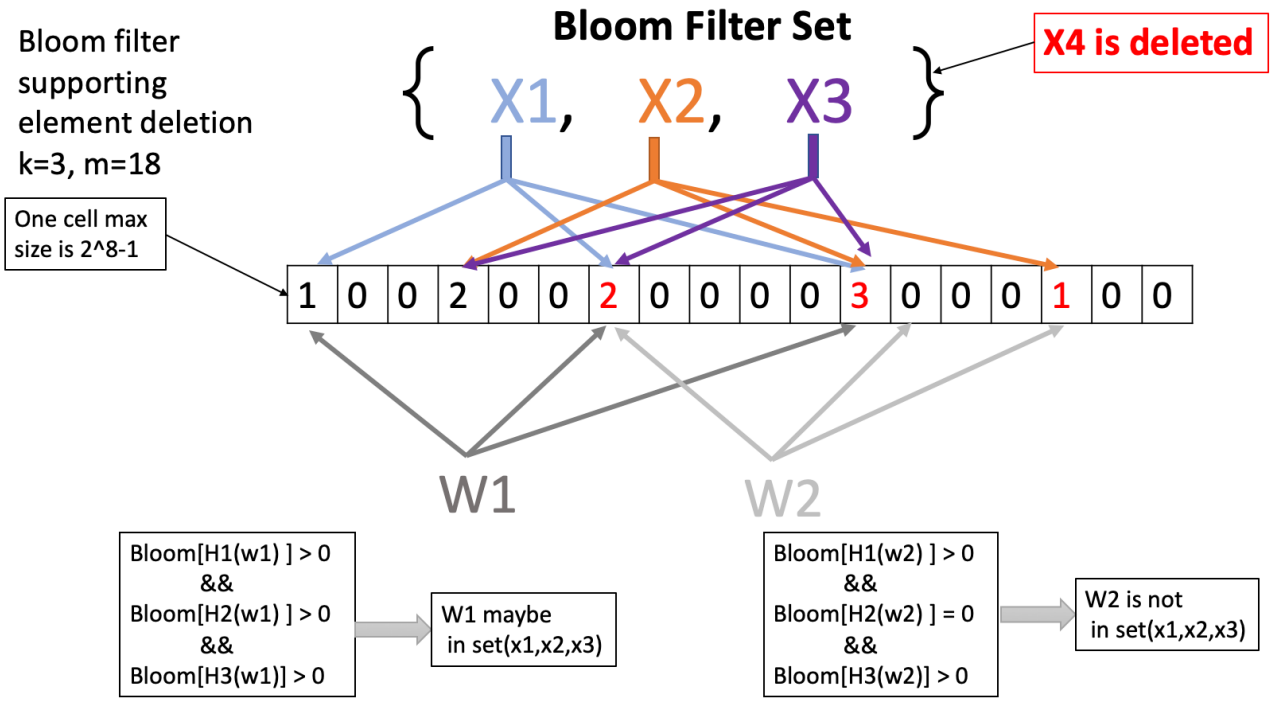

Figure 19. Bloom filter after delete X4.