CS5250

Assignment 3

Zhang Shuhao

A0120258N

## Question 1:

1. *Ftrace uses memory to keep the trace record. Before starting a new trace, the old traces are still kept in the memory by ftrace has to be emptied. How can this be done?*

**Answer:** just use the following commands: `` echo > trace”, this will erase the entire contents of the trace buffer.

1. *What is the command to change the maximum* *size of the trace file of ftrace?*

**Answer**: ``echo $size > buffer\_size\_kb”. This will reset the buffer size to be $size kb.

1. *Assuming you are changing the kernel code to insert a printk-like code to output hello world in ftrace and the message given by the print code is the only thing you want to trace, show the code and also the tracer that should use in ftrace.*

**Answer:**

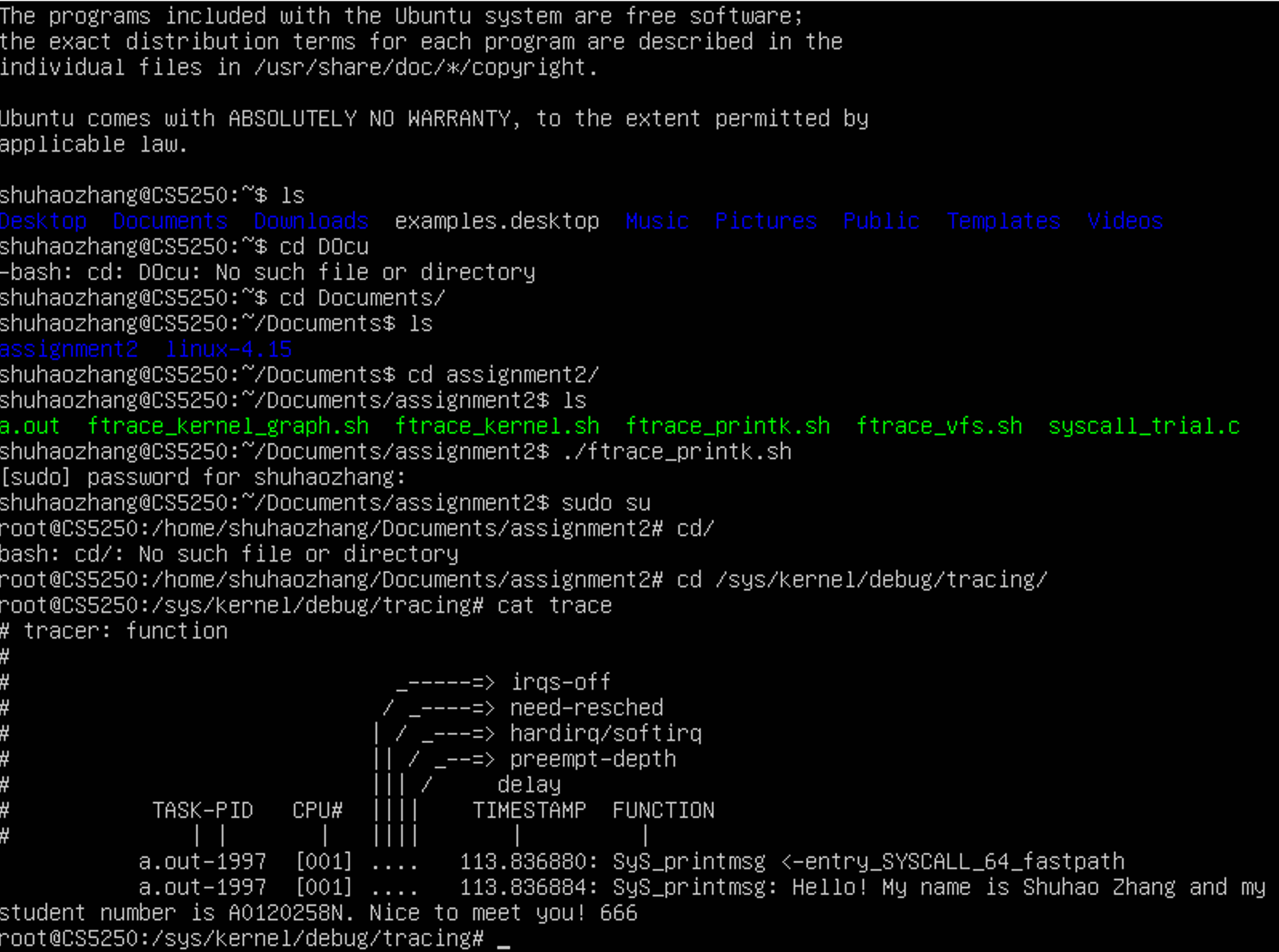
In the syscall program:

trace\_printk("Hello! My name is Shuhao Zhang and my student number is A0120258N. Nice to meet you! %d\n", i);

call the following commands:

echo function > current\_tracer;//this sets the function tracer.

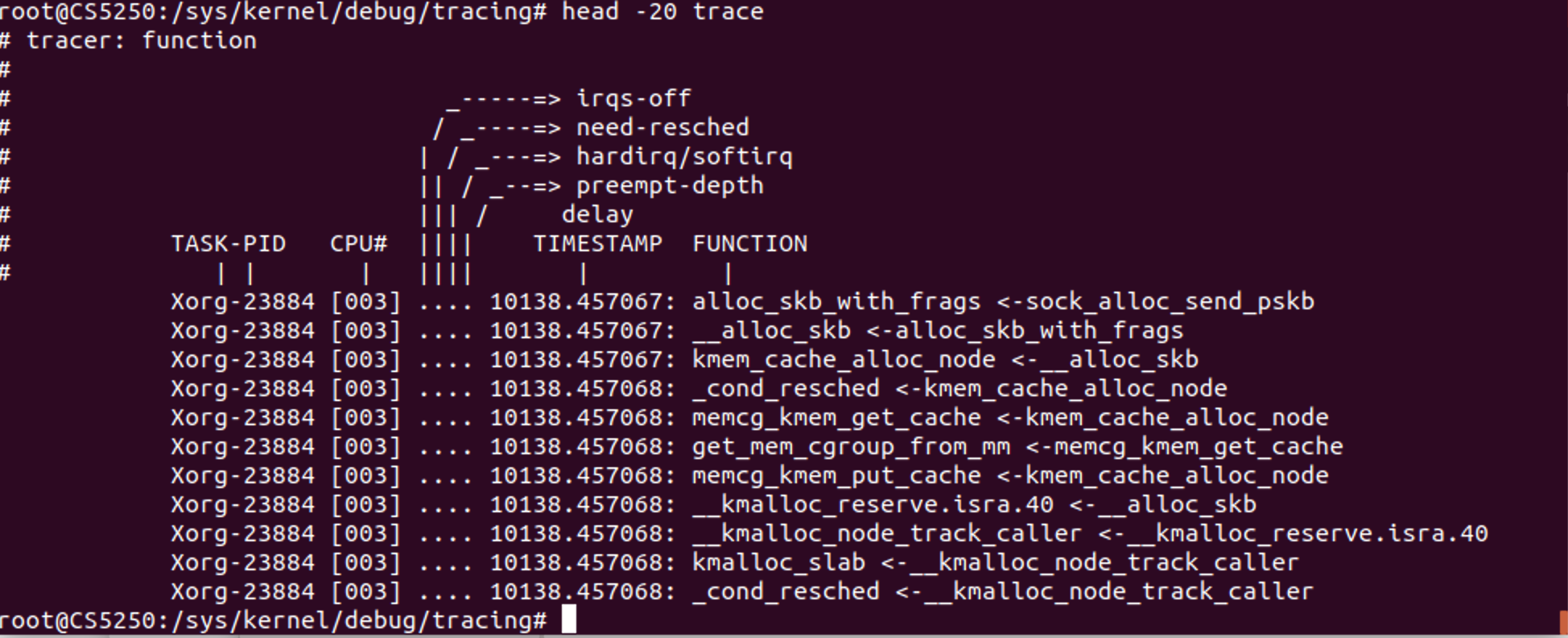
Here’s the results showing in trace.



1. *Use the* ***function*** *tracer to trace the kernel for about a few seconds. Give the screenshots of the first 20 lines of the trace file and analyse the basic structure of it.*

**Answer:**

I use the script named “ftrace\_kernel.sh” to perform this task.



The header explains the format of the output pretty well. The first two items are the traced task name and PID. The CPU that the trace was executed on is within the brackets. The timestamp is the time since boot, followed by the function name. The function in this case is the function being traced with its parent following the "<-" symbol.

## Question 2:

*Use the* ***function\_graph*** *tracer in ftrace to trace the function call stack of three kernel functions,* ***vfs\_open, vfs\_read and vfs\_write*** *and set the max function call depth to record as 10. Give the complete commands you use and analyse the trace result.*

**Answer:**

Here’s the complete commands that I used for this task. Alternatively, they are also presented in “ftrace\_vfs.sh”.

*DEBUGFS=`grep debugfs /proc/mounts | awk '{ print $2; }'` // this is to obtain the tracing folder path.*

*sudo su -c " \ // it has to be accessed with sudo right.*

*echo > $DEBUGFS/tracing/trace; \ // clear trace buffer.*

*echo 0 > $DEBUGFS/tracing/tracing\_on; \ // disable trace.*

*echo vfs\_write vfs\_open vfs\_read > $DEBUGFS/tracing/set\_graph\_function; \ // register the three functions to trace.*

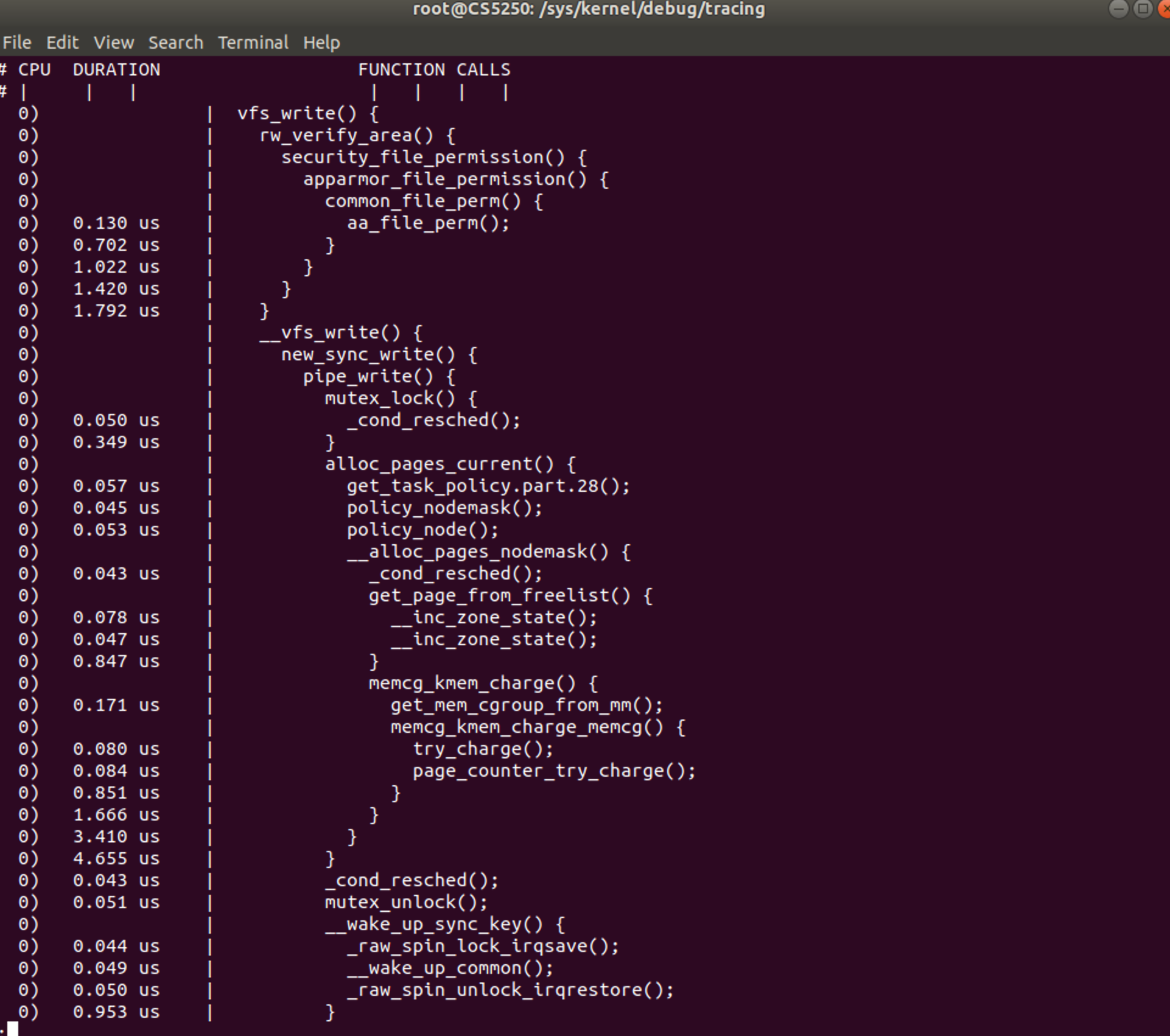
*echo 10 > $DEBUGFS/tracing/max\_graph\_depth; \ // set max function call depth to 10.*

*echo function\_graph > $DEBUGFS/tracing/current\_tracer; \ // use function graph tracer.*

*echo 1 > $DEBUGFS/tracing/tracing\_on;" // enable trace*

*sleep 0.01 // tracing for 0.01 second*

*echo 0 > $DEBUGFS/tracing/tracing\_on // disable trace*



Here’s part of the results.

A VFS specifies an interface (or a "contract") between the kernel and a concrete file system.   
From the trace results, we made two observations. First, we can see that all three function needs to first call rw\_verify\_area(), this is for security purpose. Second, the actual open, read, and write operation is first synchronized, and it requires to use mutex\_lock to ensure mutual exclusive.

## Question 3:

*This task is finished with the following steps.*

1. *Finish the steps for adding a kernel function into kernel in the tutorial. Supply the two C files.*

**Answer:**

Add a ``printmsg.c” under kernel/ folder.

Add a ``myprint.c” to call the new kernel function.

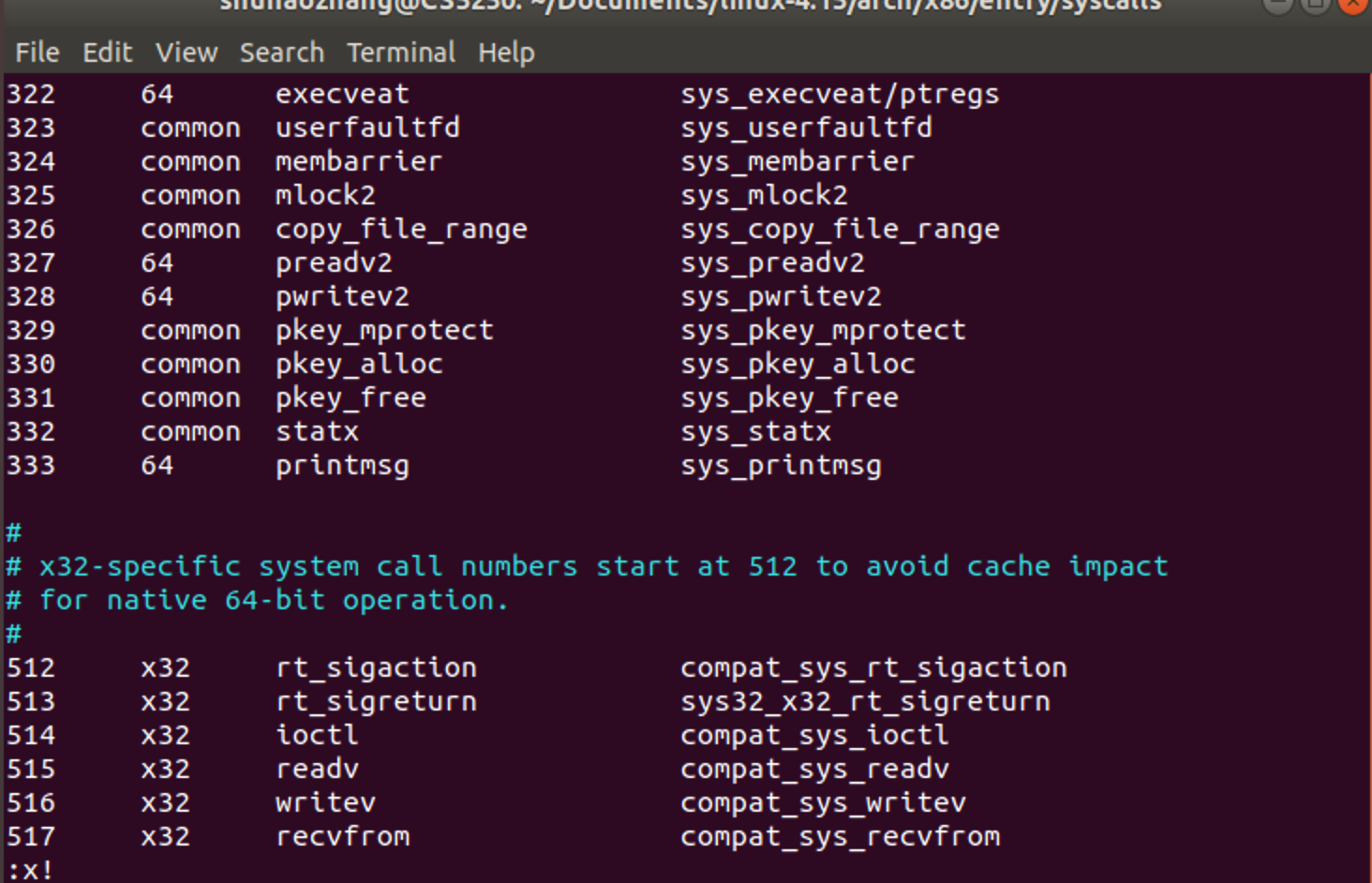
The relevant file will be attached as appendix at the end of this report.

1. *What is the full path of the system call table? Show the line you added in the system call table.*

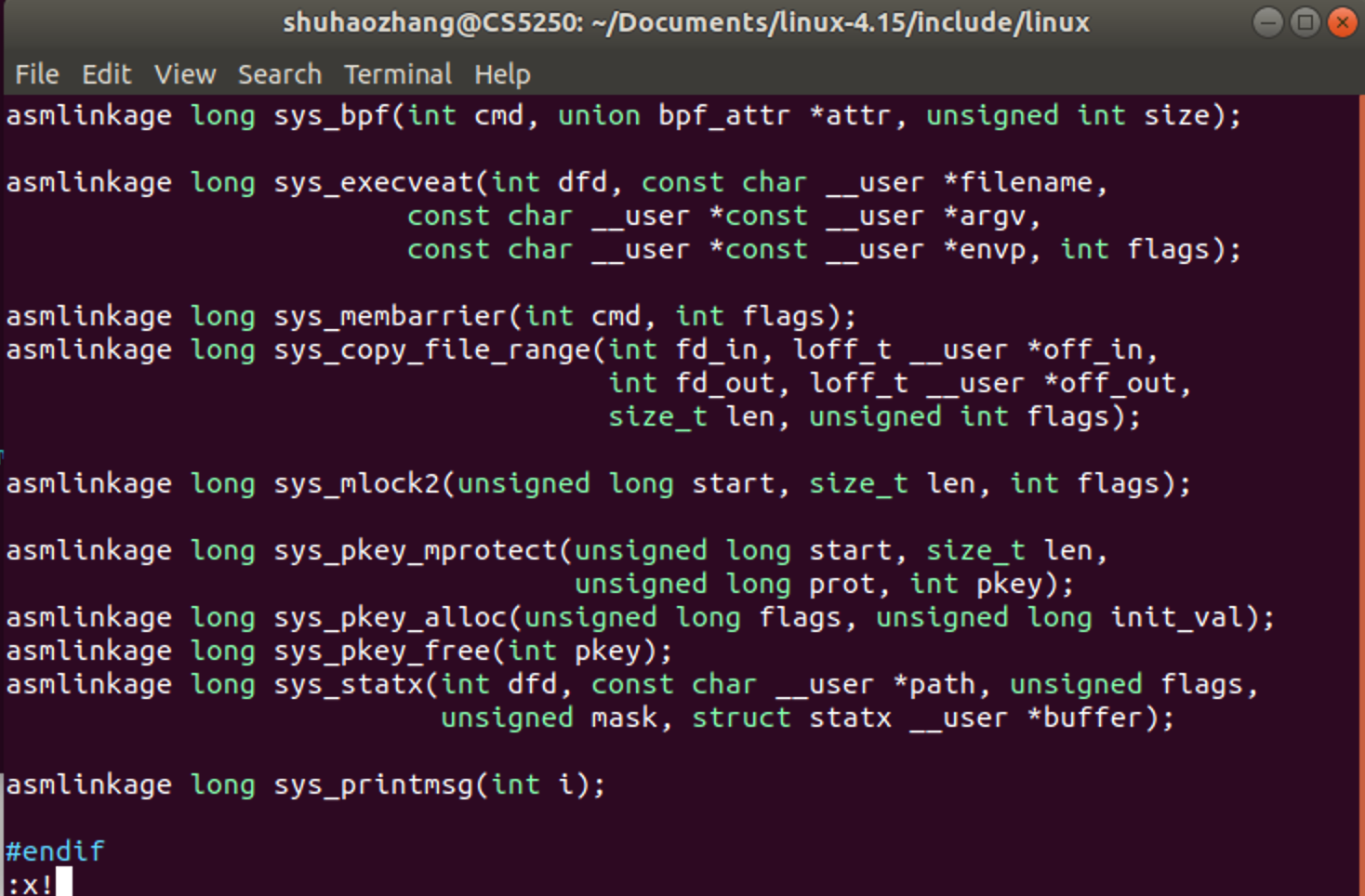
**Answer:**

As my system is a 64-bit system, I need to alter the syscall\_64.tbl file, which is located at `` $Kernel\_SRC/arch/x86/entry/syscalls”.

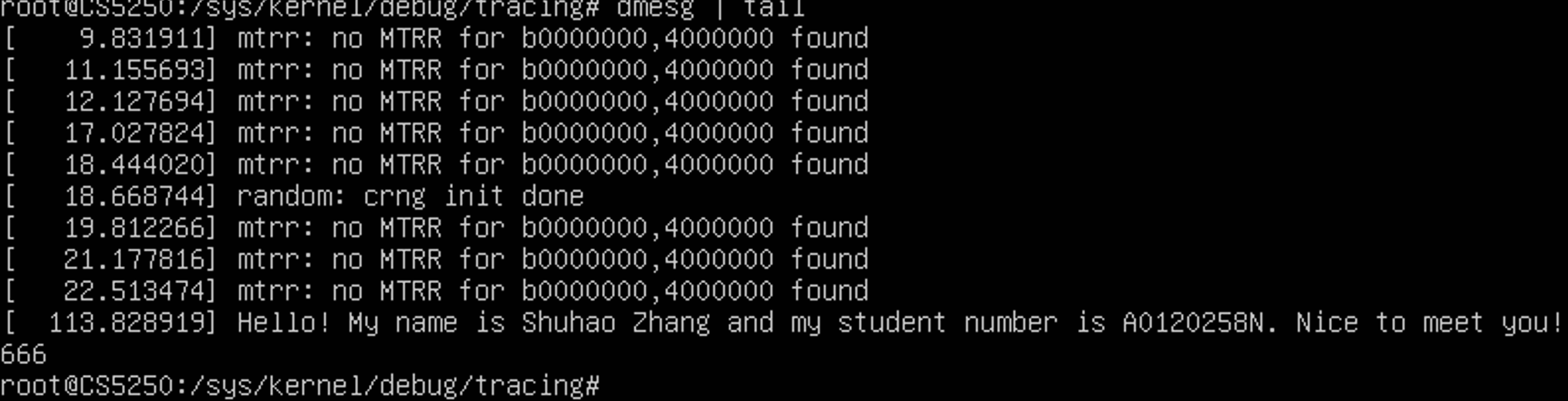
Add the following line: `` 333 64 printmsg sys\_printmsg”.



Also, we need to add declare of the function as follows.

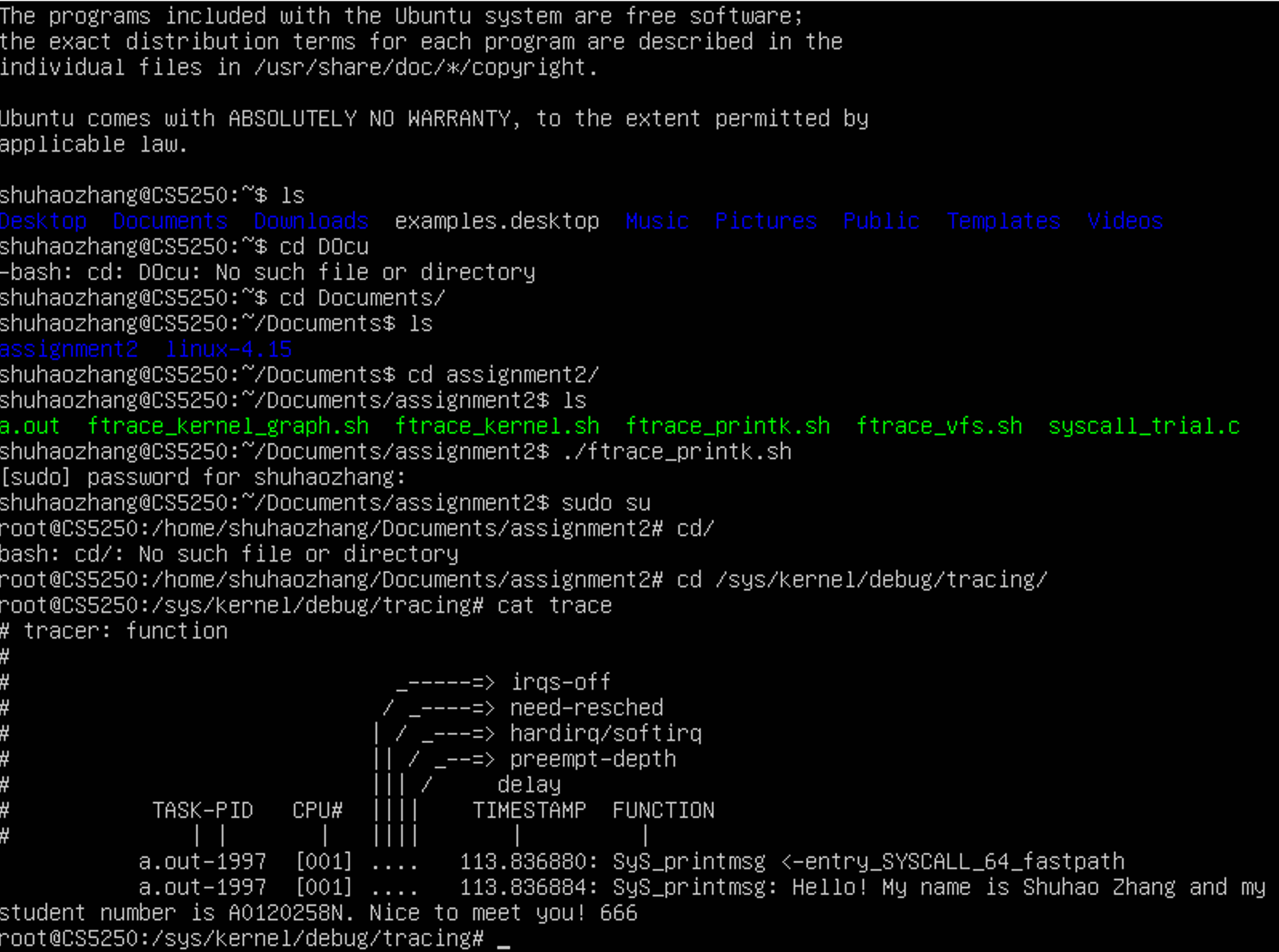


1. *Show the screenshot of the “dmesg | tail” containing the print message of your new kernel function.*

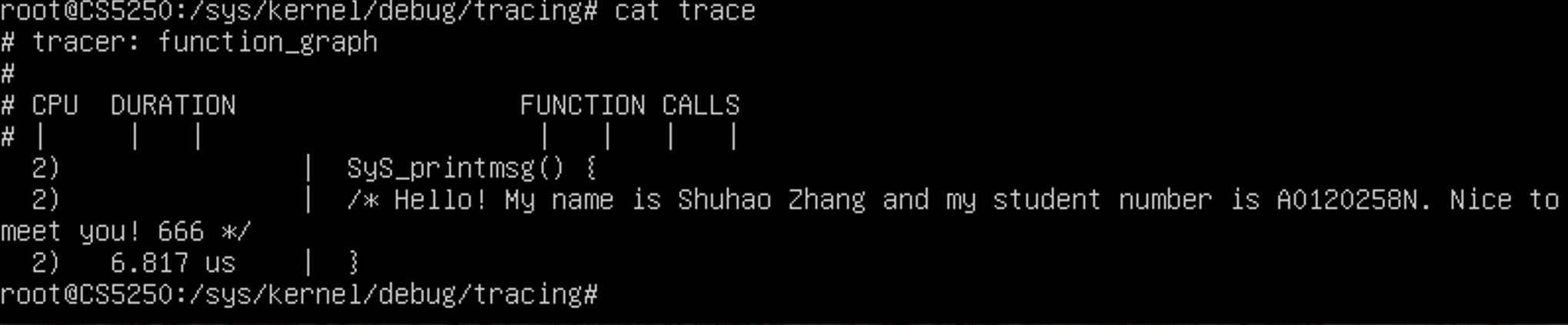


1. *Give the first 20 lines of the trace file of ftrace and analyse it.*

I use commands listed in my “ftrace\_printk.sh” script to do this task. The results are attached in the following screens.



By using function\_graph tracer, I obtain the following results.



I find that printk calls to no other functions, and it has a parent of entry\_SYSCALL\_64\_fastpath, which should related to sysenter function that switches user mode to kernel mode. The trace\_printk outputs its contents like comments in function\_graph tracer.

# **APPENDIX**

## Printmsg.c // the custom system call.

#include <linux/kernel.h> /\*for printk \*/

#include <linux/syscalls.h> /\* for SYSCALL\_DEFINE1 macro \*/

SYSCALL\_DEFINE1(printmsg,

int, i)

{

printk(KERN\_DEBUG

"Hello! My name is Shuhao Zhang and my student number is A0120258N. Nice to meet you! %d\n", i);

return 0;

}

## syscall\_trial.c // the user program to call system call.

#include <linux/unistd.h>

#define \_\_NR\_printmsg 333

int printmsg(int i);

int main() {

int rc = printmsg(666);

return rc;

}

int printmsg(int i)

{

return syscall(\_\_NR\_printmsg,

i);

}

## ftrace\_vfs.sh // the script to test vfs functions.

#!/bin/sh

DEBUGFS=`grep debugfs /proc/mounts | awk '{ print $2; }'`

sudo su -c " \

echo > $DEBUGFS/tracing/trace; \

echo 0 > $DEBUGFS/tracing/tracing\_on; \

echo vfs\_write vfs\_open vfs\_read > $DEBUGFS/tracing/set\_graph\_function; \

echo 10 > $DEBUGFS/tracing/max\_graph\_depth; \

echo function\_graph > $DEBUGFS/tracing/current\_tracer; \

echo 1 > $DEBUGFS/tracing/tracing\_on; \

sleep 2; \

echo 0 > $DEBUGFS/tracing/tracing\_on; "

## ftrace\_prink.sh // the scripts to test my system call.

#!/bin/sh

DEBUGFS=`grep debugfs /proc/mounts | awk '{ print $2; }'`

sudo su -c " \

echo > $DEBUGFS/tracing/trace; \

echo 0 > $DEBUGFS/tracing/tracing\_on; \

echo SyS\_printmsg > $DEBUGFS/tracing/set\_ftrace\_filter; \

echo printk > $DEBUGFS/tracing/set\_event; \

echo function > $DEBUGFS/tracing/current\_tracer; \

echo 1 > $DEBUGFS/tracing/tracing\_on; \

./a.out; \

echo 0 > $DEBUGFS/tracing/tracing\_on"

## ftrace\_kernel.sh // the scripts to test kernel execution.

#!/bin/sh

DEBUGFS=`grep debugfs /proc/mounts | awk '{ print $2; }'`

sudo su -c " \

echo > $DEBUGFS/tracing/trace; \

echo 0 > $DEBUGFS/tracing/tracing\_on; \

echo function > $DEBUGFS/tracing/current\_tracer; \

echo 1 > $DEBUGFS/tracing/tracing\_on; \

sleep 5; \

echo 0 > $DEBUGFS/tracing/tracing\_on"