**Operating Systems Project Report**

|  |  |
| --- | --- |
| **Project Number (01 / 02 / 03):** | 01 |
| **Name:** | 蕭望緯 |
| **Student ID:** | 0811521 |
| **YouTube link (Format youtube.com/watch?v=[key]):** | <https://youtu.be/Y0dAnP7n9mw> |
| **Date (YYYY-MM-DD):** | 2021-10-20 |
| **Names of the files**  **uploaded to E3:** | OS\_Project01\_0811521.pdf |
| **Physical Machine Total RAM (Example: 8.0 GB):** | 16GB |
| **Physical Machine CPU (Example: Intel i7-2600K):** | 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz |

|  |  |
| --- | --- |
| **Checklist** | |
| **Yes/No** | **Item** |
| **Yes** | **The report name follows the format “OS\_ProjectXX\_StudentID.pdf”.** |
| **Yes** | **The report was uploaded to E3 before the deadline.** |
| **Yes** | **The YouTube video is public, and anyone with the link can watch it.** |
| **Yes** | **The audio of the video has a good volume.** |
| **Yes** | **The pictures in your report and video have a good quality.** |
| **Yes** | **All the questions and exercises were answered inside the report.** |
| **Yes** | **I understand that late submission is late submission, regardless of the time uploaded.** |
| **Yes** | **I understand that any cheating in my report / video / code will not be tolerated.** |

**Individual Questions:**

1. What is a Kernel? What are the differences between *mainline, stable* and *longterm*? What is a Kernel panic?

Ans:

Kernel: lies in the center part of an operating system, responsible for process and memory management, file systems, device control and networking.

*mainline, stable* and *longterm* are three sections that release some kernel versions:

mainline: provide versions that contain new features

stable: versions in the mainline section will be sent to the stable section, and bugfix versions will be released frequently

longterm: bugfix versions are released less frequently

Kernel panic: occurs when any fatal error is detected, and the OS will stop to protect the system

2. What are the differences between *building*, *debugging* and *profiling*?

Ans:

building: install a new kernel

debugging: for the development of kernel

profiling: monitor the performance of processes and resources allocation of the kernel

3. What are GCC, GDB, and KGDB, and what they are used for?

Ans:

GCC: compiler for various programming languages, including C, C++, Fortran, Ada, Go, and etc.

GDB: debugger that aids the development of software programs

KDGB: debugger specific to development of kernel, runs on the Target machine to be debugged

4. What are the /usr/, /boot/, /home/, /boot/grub folders for?

Ans:

/usr: contains important files such as data files, libraries, binaries, etc, which support the system.

/boot: contains files related to the booting of the system

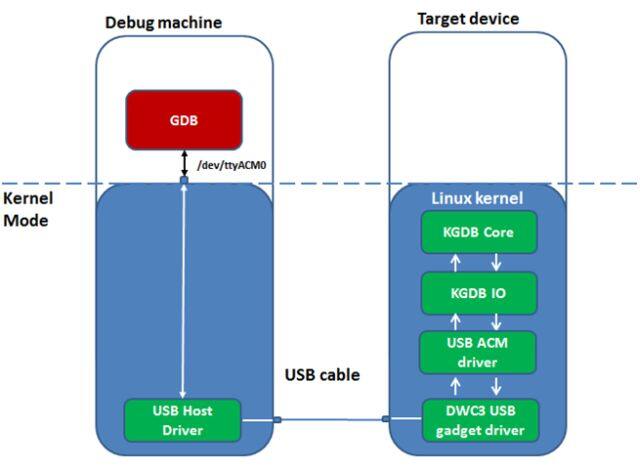
/home: personal workspace for users where the users have full access to writing and creating files

/boot/grub: contains files related to GRUB

5. What are the general steps to debug a Linux Kernel? (Add a figure)

Ans:

* A debugger machine and a target machine
* Connection between them
* gdb on the debugger machine and kgdb on the target machine
* establish debugging session



source: [1]

6. For this project, why do we need two virtual machines?

Ans:

Using kgdb requires two virtual machines. Debugger machine is used as a debugging tool and runs gdb, while Target machine is debugged, runs kgdb, and can be frozen by the debugger. Also, this project simulates remote debugging in real world situation.

7. In Section 3.2, what are the differences between **make**, **make modules\_install** and **make install**?

Ans:

make: compile the kernel

make modules\_install: install compiled kernel modules

make install: install the kernel so that we can execute it

8. In Section 3.3, what are the commands **kgdbwait** and **kgdboc=ttyS1,115200** for?

Ans:

**kgdbwait:** Upon booting the kernel, the system will wait for the connection to the KGDB

**kgdboc:** configure the device through which gdb and kgdb communicate

9. What is **grub**? What is **grub.cfg**?

Ans:

grub: a boot loader that manages the booting process and allows users to launch one of the installed operating system

grub.cfg: GRUB configuration file that contains menu setting and individual setting for each installed kernel. It is generated by several grub.d files.

10. List at least 10 commands you can use with GDB.

Ans:

* connect to remote machine: **target remote /dev/ttyS1**
* unfreeze the target machine: **continue**
* set breakpoint at the beginning of a function: **break** *function*
* list information of all current breakpoints: **info breakpoints**
* delete all breakpoints: **delete**
* delete breakpoints set in the function: **clear** *function*
* print the parameter of the function hitting a breakpoint: **print** *parameter*
* list source code nearby the index: **list *index***
* execute the next line of program (execute the entire function call): **next**
* execute the next line of program (include each line of a function call): **step**
* show the recent locations (several frames, the entire stack) in the program: **backtrace**
* show current stack frame: **frame**
* list information of current frame: **info frame**

11. What is a kernel function? What is a system call?

Ans:

kernel function: jobs supported by the kernel

system call: request services from the kernel

12. What is KASLR? What is it for?

Ans:

“Kernel address space layout randomization”

It enables address space randomization to protect the memory from attack and illegal access.

13. What are GDB’s non-stop and all-stop modes?

Ans:

non-stop: in multi-threaded system, GDB freezes certain threads while other threads keep running

all-stop modes: GDB freezes the entire system (all threads)

14. Explain what the command **echo g > /proc/sysrq-trigger** does.

Ans:

stop the kernel execution and give control to kgdb

15. What are these functions: **clone**, **mmap**, **write** and **open?**

Ans:

**clone:** create a child process

**mmap:** creates a new mapping of files to memory in the virtual address space of the calling process

**write:** writes bytes from the buffer to the file pointed by file descriptor.

**open:** opens the file specified by a pathname.

16. Why is there no **fork** system call? What is the difference between **fork** and **clone?**

Ans:

**fork** will call **clone** system call and do similar things like **clone.**

**fork:** the created child process doesn’t share resources (e.g., memory space, files) with the parent

**clone:** the created child process can share resources with the parent

Screenshot #1

Target(left) & Debugger(right) Machine

A screenshot of a computer

Description automatically generated

Target machine disks configuration

Graphical user interface, text

Description automatically generated

Screenshot #2

Serial port communication testing after some configuration

Message passed from Target to Debugger successfully through serial port communicationA screenshot of a computer

Description automatically generated

Screenshot #3

Use the predefined configuration from one of the other versions of kernel

Copy the .config file to the directory of kernel used for this project

Text

Description automatically generated

Screenshot #4

Open Kernel configuration interface and examine the setup

Graphical user interface, text

Description automatically generated

Screenshot #5

Open the file: /usr/src/linux-4.4.101/.config

Double-check these four lines so that we can use kgdb for this kernel in later sections:

CONFIG\_FRAME\_POINTER=y

CONFIG\_KGDB=y

CONFIG\_KGDB\_SERIAL\_CONSOLE=y

CONFIG\_KGDB\_KDB=y

CONFIG\_KDB\_KEYBOARD=y

Text

Description automatically generated

Screenshot #6

Start building the kernel.

First, clean up the executable or complied object files. Then, build kernel with the maximum available processors.

Text

Description automatically generated

Screenshot #7

After kernel is compiled, install kernel modules

Graphical user interface, text

Description automatically generated

Screenshot #8

Install the complete kernel to the system

Text

Description automatically generated

In the /boot directory, the four files of the kernel 4.4.101 are present.

* 1. a. initrd.img-4.4.101
  2. b. vmlinuz-4.4.101
  3. c. system.map.4.4.101
  4. d. config-4.4.101

A screenshot of a computer

Description automatically generated with medium confidence

Screenshot #9

update initrd.img. Then, update GRUB

Text

Description automatically generated

Screenshot #10

set up parameters for GRUB so that we can see the menu for different kernels and set nokaslr. Then, update GRUB

A screenshot of a computer

Description automatically generated

Screenshot #11

**gdb ./vmlinux:** open gdb

Text

Description automatically generated

**target remote /dev/ttyS1**: connect to remote target machine

Text

Description automatically generated

Screenshot #12

After I enter command **continue** in the debugger machine, the target machine resumes booting its system.

A screenshot of a computer

Description automatically generated

Screenshot #13

[Chromium OS Docs - Linux System Call Table (googlesource.com)](https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md)

A picture containing calendar

Description automatically generated

[Linux System Call Table (nps.edu)](http://faculty.nps.edu/cseagle/assembly/sys_call.html)

Calendar

Description automatically generated

[Linux/i386 system calls (sourceforge.net)](http://asm.sourceforge.net/syscall.html#4)

Graphical user interface, text, application

Description automatically generated

Screenshot #14

set a breakpoint for **rmdir**

trigger the breakpoint and print the parameter *pathname*

Text

Description automatically generated

Screenshot #15

set one breakpoint for **security\_path\_mkdir** that will be triggered when executing **mkdir**



Screenshot #16

parameters of **security\_path\_mkdir:** path, dentry, and mode

print some parameters of the function to gain a closer look into the information of the breakpoint

Text

Description automatically generated

Screenshot #17

after several **continue**, the target machine regains control.

Text

Description automatically generated

Screenshot #18

the perf command

Text

Description automatically generated

Text

Description automatically generated

Screenshot #19

This is the execution of **fileCopyTest.c**.

the execution result of perf, the trace result, and the generated files

Graphical user interface, text

Description automatically generatedA screenshot of a computer

Description automatically generated

Screenshot #20

compare the trace results of **emptyTest.c** and **fileCopyTest.c**

Since **fileCopyTest.c** executes a few more commands such as calling **fork()**, so its trace result shows additional events.

Text

Description automatically generated

**(p.32)**

a. Will the functions’ execution time be longer if the file is bigger?

Ans: According to my experiment, the executions of **clone**, **open**, and **write** take longer time when the text file is larger. The executions of **mmap** for different file size take roughly the same amount of time. But there is some time when the executions of same function take same amount of time for different file sizes; I guess the file sizes are not large enough to generate distinct differences.

b. How is the behavior of each function? Sort them from slowest to fastest.

Ans: from fastest to slowest**: mmap, write, open, clone**

c. Create a graph of file size (in bytes) vs. execution time (ms) of these four functions, using 3 different file sizes.

Screenshot #21

d. Perf also has the report command:

i. What is it for?

Ans: Show details of performance of processes and overhead of each function

ii. For fileCopyTest, show and interpret the results.

Ans: The following functions have the highest overhead:

\_\_lock\_text\_start

generic\_make\_request\_checks

unmap\_page\_range

Screenshot #22 [perf report]

A screenshot of a computer

Description automatically generated

e. Perf has more commands (Section 5.1 step 4). Select another command (besides report, trace and record), explain what it is for and show how to use it. Create your own scenario.

Ans: **perf top**

function: real-time analysis of performance

Screenshot #23 [perf top]

A screenshot of a computer

Description automatically generated

Screenshot #24 [zoom into DSO loop]

I execute a file with infinite while loops that keep calling a function foo().

A picture containing graphical user interface

Description automatically generated

(Below)In the main function, we can inspect the parts that occupy the resources.

A screenshot of a computer

Description automatically generated

Sources:

[1]: <https://www.trendmicro.com/en_us/research/17/a/practical-android-debugging-via-kgdb.html>