

# Operating System

## project 1A

Department: 土木5B

ID: 0511330

Name: 劉紘華

Vedio Link: <https://youtu.be/YTXkAP0tN2c>

## Goal

Get familiar with tools which is used to setup an environment for Linux kernel including kernel debugging, compilation and profiling.

## Basic concepts and Tools Introduction

### 1. GDB:

**GNU Debugger**, 支援C, C++, Fortran等高階語言的除錯, 主要用在user program (application)的開發, 特色是支援Remote debugging模式, 可用在embedded system開發, 能設定breakpoint方便特定的程式碼區段除錯。

### 2. KGDB:

**Kernel GNU Debugger**, 支援Linux, NetBSD和FreeBSD的kernel除錯, Serial Connection在透過UDP/IP protocol連接**Target Machine** & **Host Machine**, 對Target Machine的kernel進行Remote Debug。

**註:** **Target Machine** is the one **being debugged** and running the patched kernel.  
**Host Machine** is the one which runs gdb to debug the target.

### 3. Grub:

**GNU Grub**是一個Opensouce的專案, 是一種bootloader, 讓電腦可以安裝多種作業系統, 幫助引導到使用者到欲使用的作業系統上。

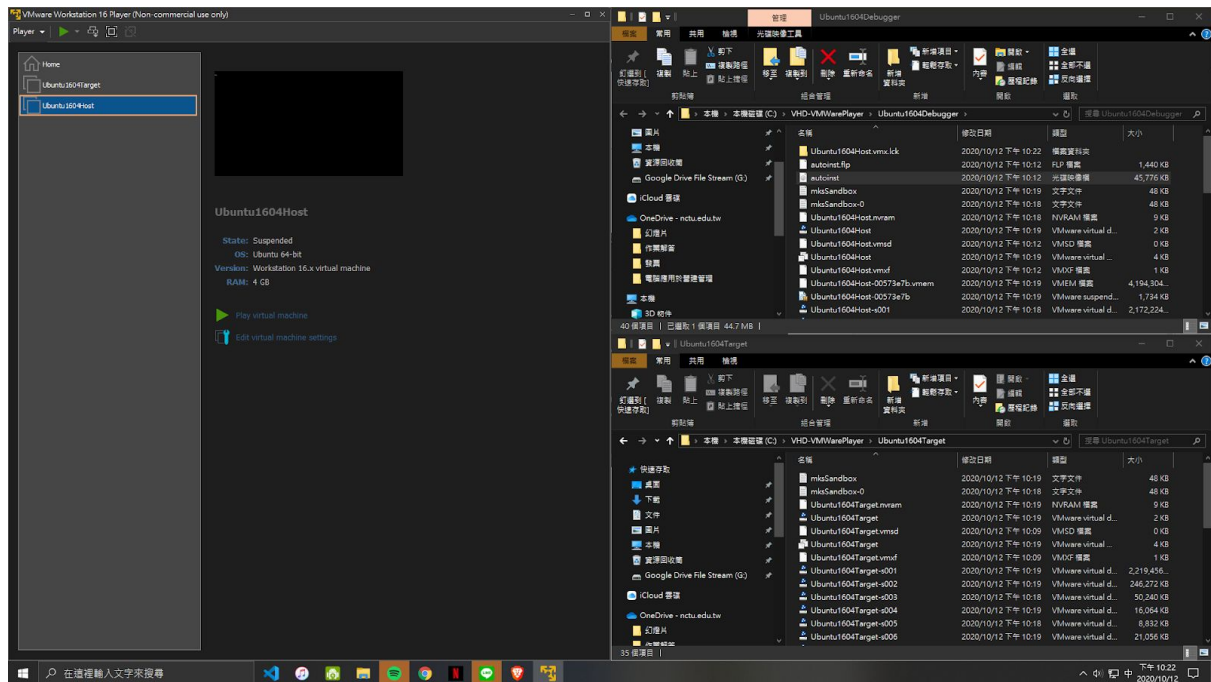
### 4. SSH and OpenSSH:

SSH是一種通訊協定(protocol), 在Client-Server model間傳遞的訊息, 都被會經過加密, 只有在認證的機器上才能被安全解密; OpenSSH是一個OpenSource的套件, 能透過Command Line Interface在Linux上安裝並執行。

## [Section 1-1]

### Screenshot 1 - Virtual Machine setup

When doing kernel debugging, we need two machines - **Target** and **Host** mentioned above. Consequently, we use VMware to simulate the Target and Host, which provides us to run two machines independently just on a single PC.



## [Section 1-2]

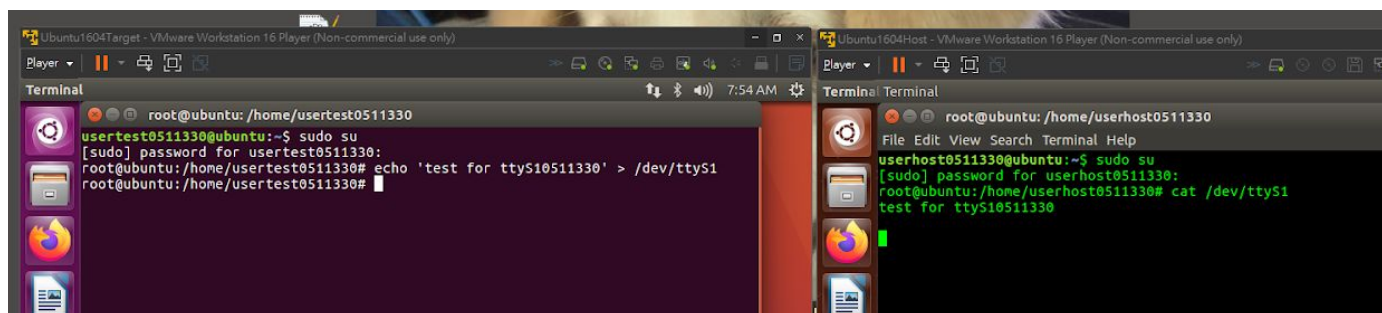
### Screenshot 2 - Verifying the connection is built correctly.

In the real world, we need to connect two computers through wired (e.g. ethernet) or wireless (e.g. WiFi); But for our two virtual machines, we can simply build the physical connection through "Serial port". To check the port was built correctly, using commands as follows:

Host: **# cat /dev/ttyS1**

Target: **#echo 'test for ttyS1 0511330' > /dev/ttyS1**

The host will capture the message from /dev/ttyS1 port, and the target is making an echo message in the port ttyS1 (like broadcasting).

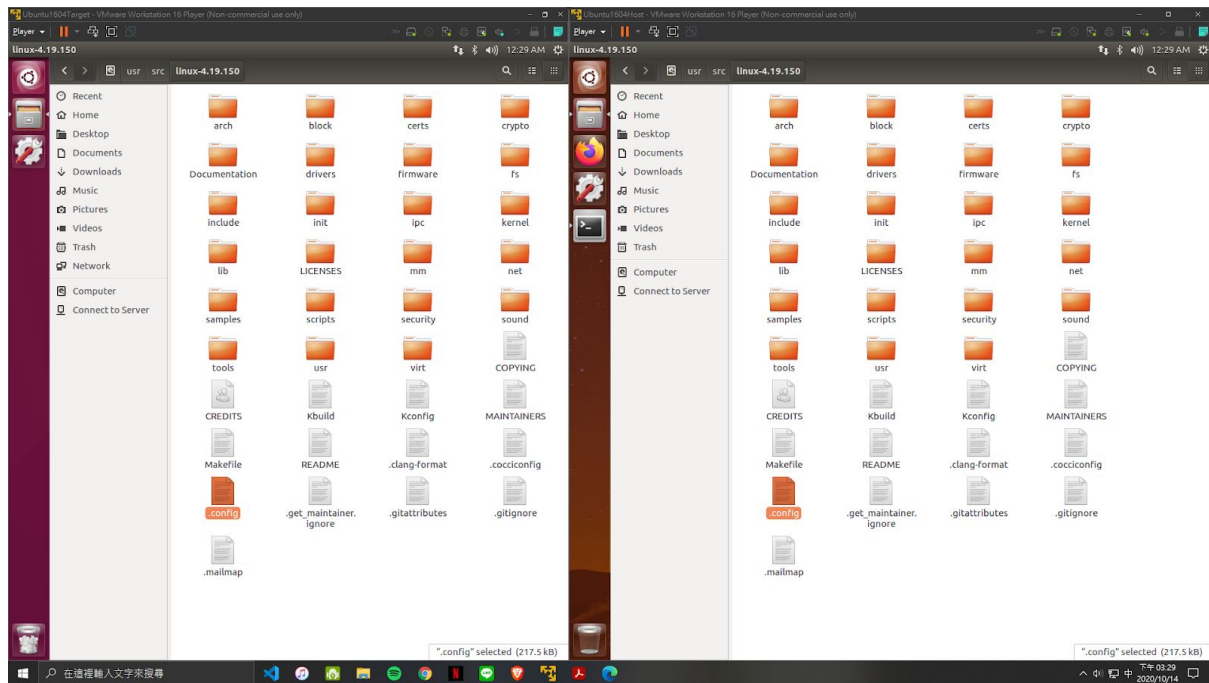


## [Section 2-1~2-2]

### Screenshot 3 - Download the source code for Linux kernel

We download the Linux source code, and use the following command to prebuild the .config file: **\$sudo cp -v /boot/config-\$(uname -r) .config**

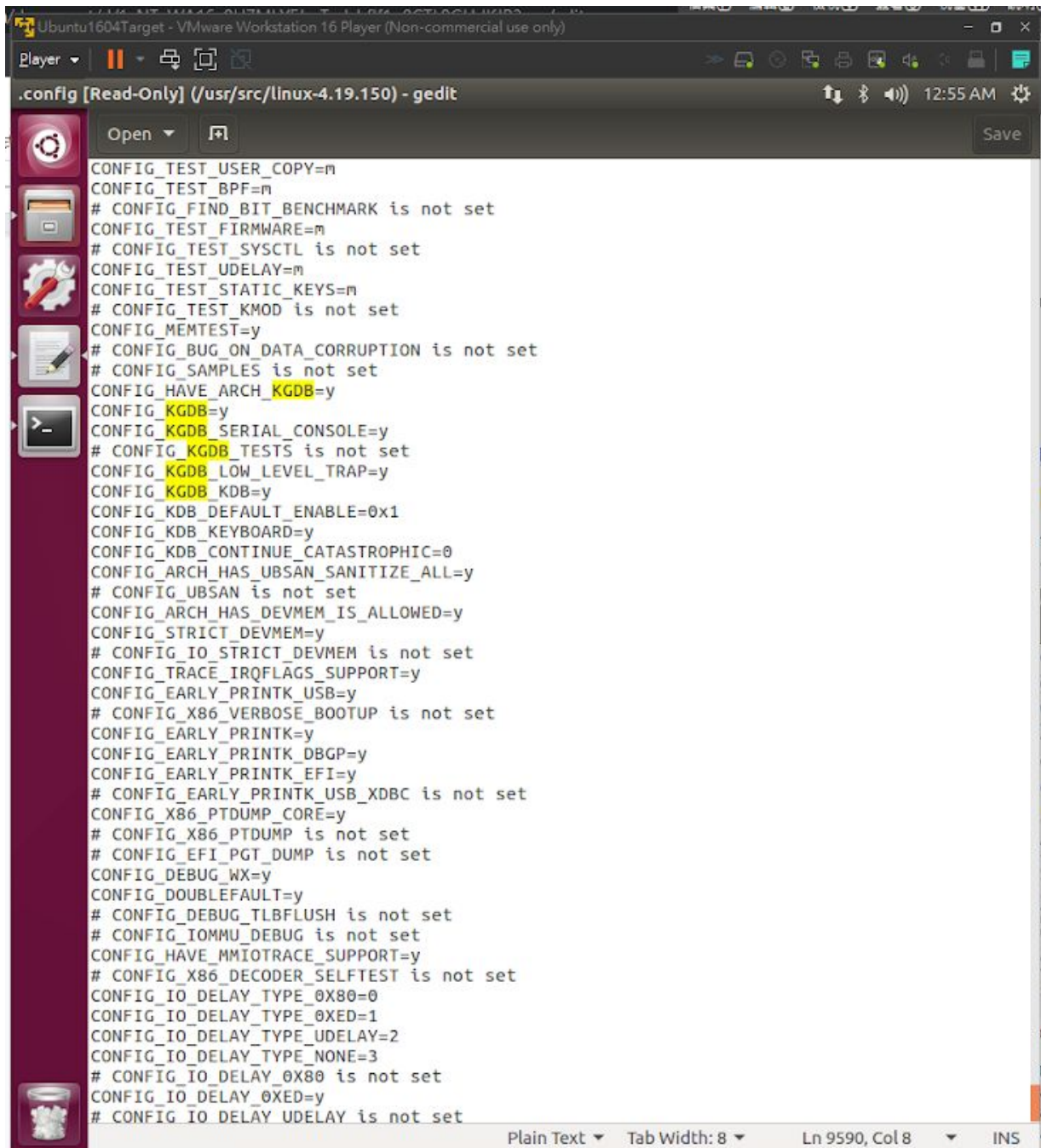
Config file like a function table of the kernel, allowing us to modify the functions which told the underlying hardware how to work. For example, we can set the functions like power management options, virtualization, I/O bus, etc.



### [Section 3-1~3-2] Linux kernel patching

#### Screenshot 4 - check the KGDB setup

checking the KGDB settings in .config file is set to true. The thing we modify the settings in the.config file calls kernel patching. Use “Ctrl” + “F” to search the keyword KGDB, and remember the line shown on the right corner. (9590).



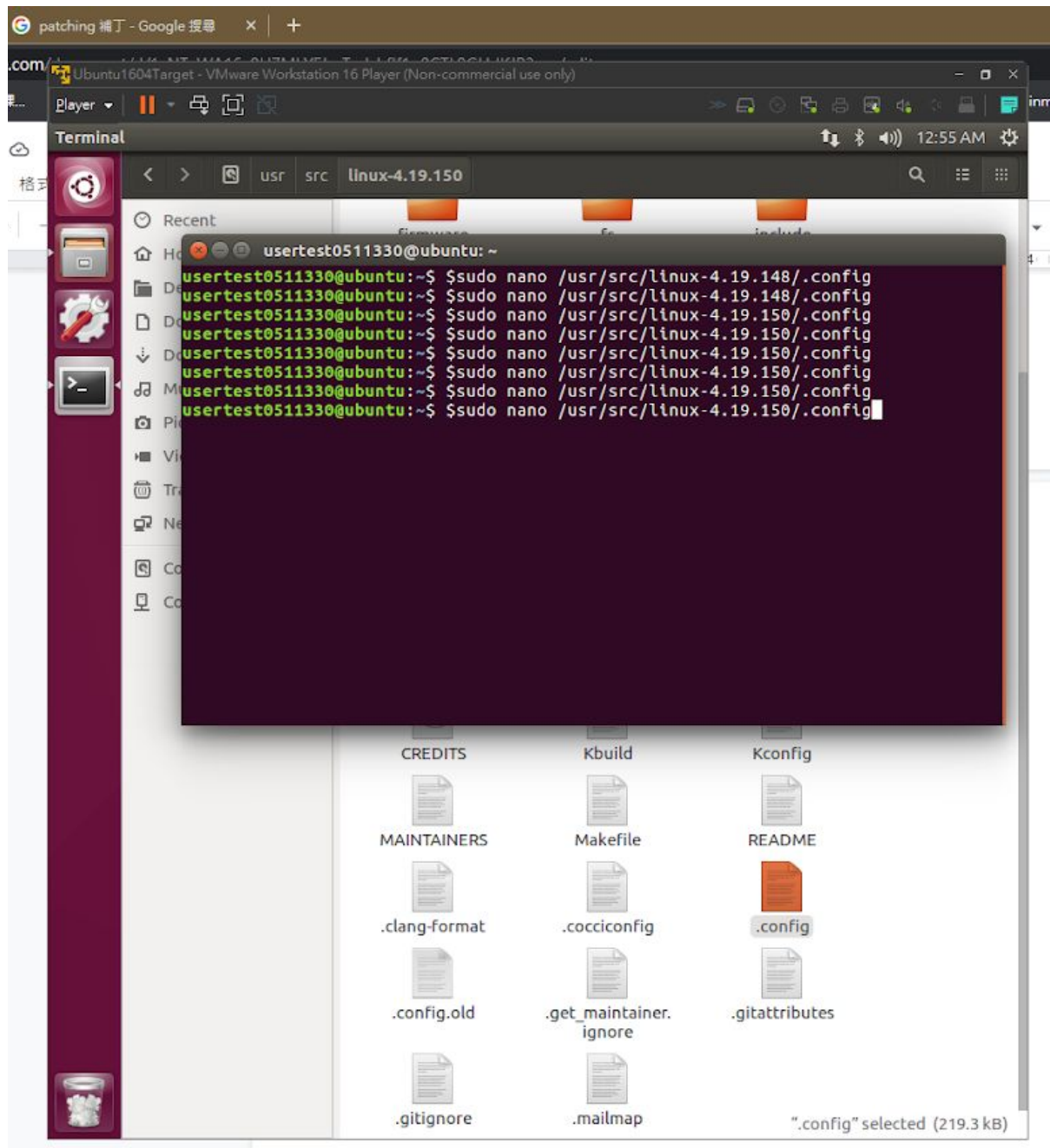
```
CONFIG_TEST_USER_COPY=m
CONFIG_TEST_BPF=m
# CONFIG_FIND_BIT_BENCHMARK is not set
CONFIG_TEST_FIRMWARE=m
# CONFIG_TEST_SYSCALL is not set
CONFIG_TEST_UDELAY=m
CONFIG_TEST_STATIC_KEYS=m
# CONFIG_TEST_KMOD is not set
CONFIG_MEMTEST=y
# CONFIG_BUG_ON_DATA_CORRUPTION is not set
# CONFIG_SAMPLES is not set
CONFIG_HAVE_ARCH_KGDB=y
CONFIG_KGDB=y
CONFIG_KGDB_SERIAL_CONSOLE=y
# CONFIG_KGDB_TESTS is not set
CONFIG_KGDB_LOW_LEVEL_TRAP=y
CONFIG_KGDB_KDB=y
CONFIG_KDB_DEFAULT_ENABLE=0x1
CONFIG_KDB_KEYBOARD=y
CONFIG_KDB_CONTINUE_CATASTROPHIC=0
CONFIG_ARCH_HAS_UBSAN_SANITIZE_ALL=y
# CONFIG_UBSAN is not set
CONFIG_ARCH_HAS_DEVMEM_IS_ALLOWED=y
CONFIG_STRICT_DEVMEM=y
# CONFIG_IO_STRICT_DEVMEM is not set
CONFIG_TRACE_IRQFLAGS_SUPPORT=y
CONFIG_EARLY_PRINTK_USB=y
# CONFIG_X86_VERBOSE_BOOTUP is not set
CONFIG_EARLY_PRINTK=y
CONFIG_EARLY_PRINTKDBG=y
CONFIG_EARLY_PRINTK_EFI=y
# CONFIG_EARLY_PRINTK_USB_XDBC is not set
CONFIG_X86_PTDUMP_CORE=y
# CONFIG_X86_PTDUMP is not set
# CONFIG_EFI_PGT_DUMP is not set
CONFIG_DEBUG_WX=y
CONFIG_DOUBLEFAULT=y
# CONFIG_DEBUG_TLBFLUSH is not set
# CONFIG_IOMMU_DEBUG is not set
CONFIG_HAVE_MMIOTRACE_SUPPORT=y
# CONFIG_X86_DECODER_SELFTEST is not set
CONFIG_IO_DELAY_TYPE_0X80=0
CONFIG_IO_DELAY_TYPE_0XED=1
CONFIG_IO_DELAY_TYPE_UDELAY=2
CONFIG_IO_DELAY_TYPE_NONE=3
# CONFIG_IO_DELAY_0X80 is not set
CONFIG_IO_DELAY_0XED=y
# CONFIG_IO_DELAY_UDELAY is not set
```

Plain Text ▾ Tab Width: 8 ▾ Ln 9590, Col 8 ▾ INS

### Screenshot 5 - modify the KGDB settings through CLI

if you need to change some settings, use the command:

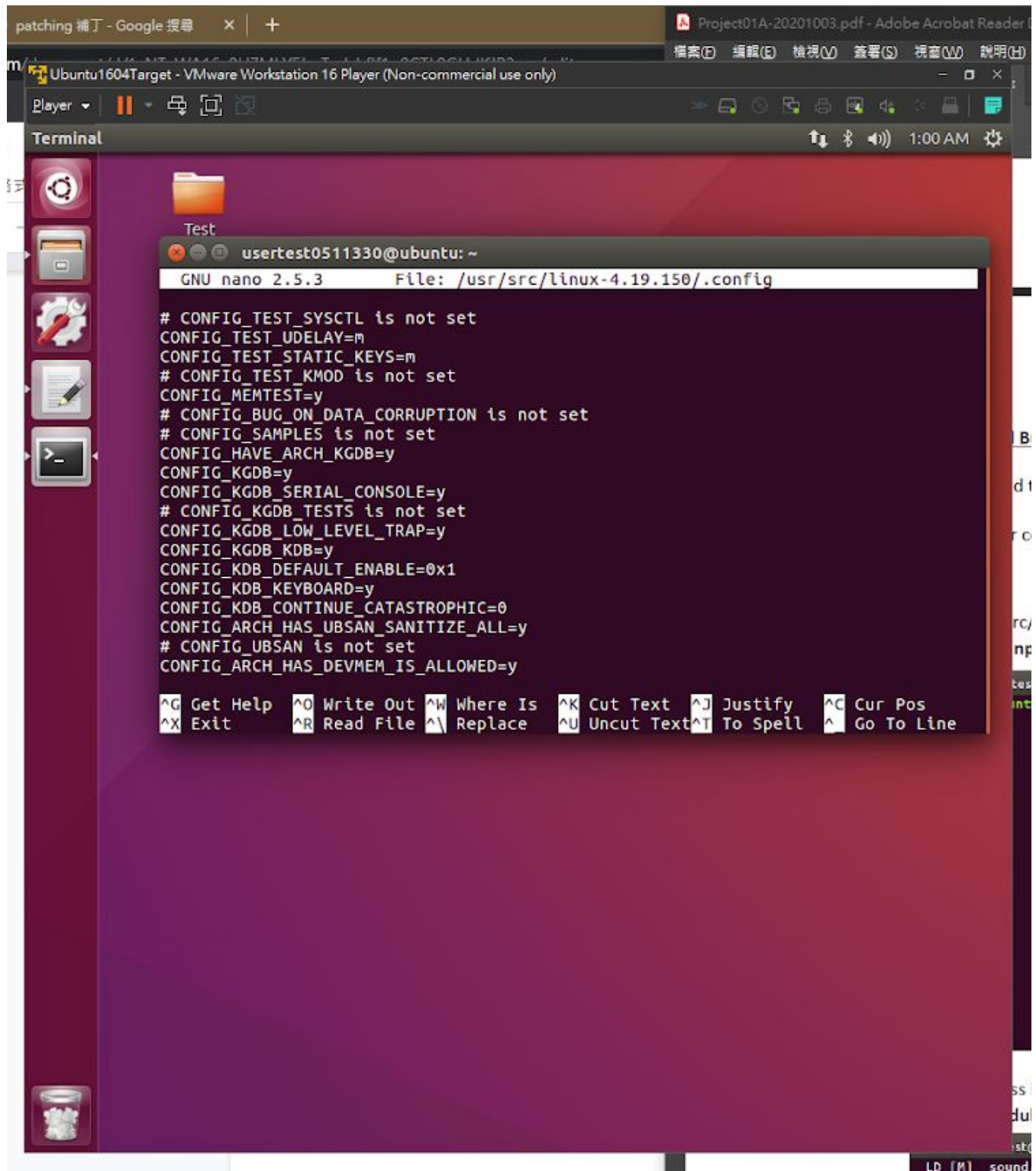
```
$sudo nano /usr/src/linux-4.19.150/.config
```





## Screenshot 6 - check the KGDB setup

press the button of combination "Ctrl" + "Shift" + "-" to go to line:9590 ("- is the button next to "=" on keyboard) and you can modify it.



[Section 3-1~3-2] proceed to build the Linux kernel

### Screenshot 6 - Compilation to .ko

We compile it to .ko files (kernel object code).

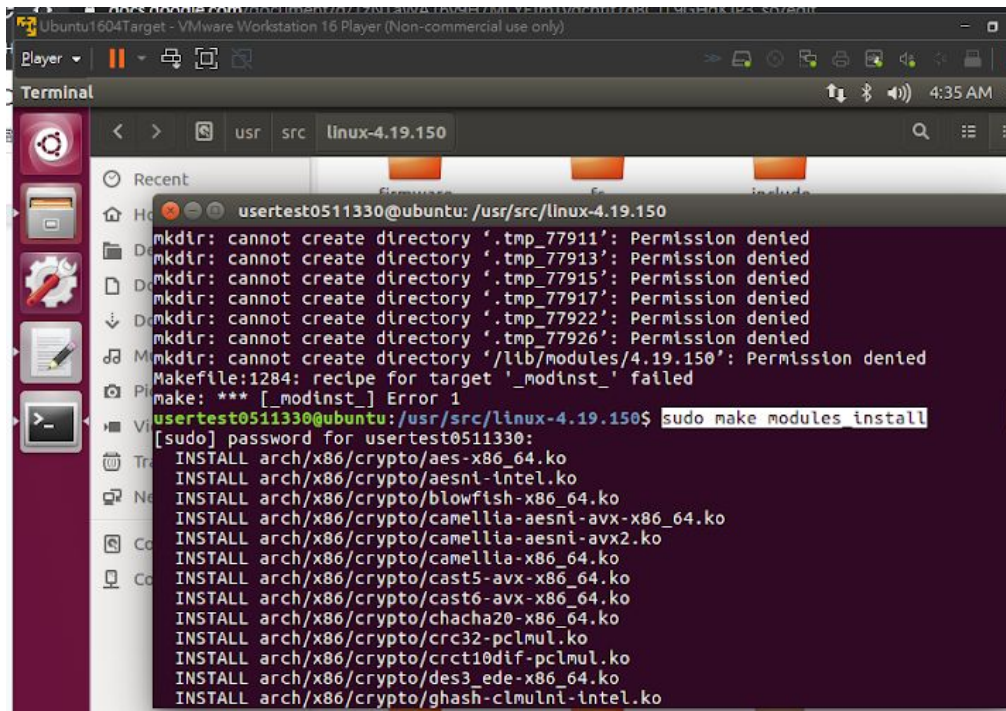
Use the command: **\$sudo make -j \$(nproc)**

x86!!

```
# CONFIG_FIND_BIT_BENCHMARK is not set
CONFIG_TEST_FIRMWARE=m
# CONFIG_...
LD [M] drivers/platform/chrome/chromeos_laptop.ko
LD [M] drivers/platform/chrome/chromeos_pstore.ko
LD [M] drivers/platform/chrome/cros_ec_lpcs.ko
LD [M] drivers/platform/chrome/cros_kbd_led_backlight.ko
LD [M] drivers/platform/x86/acer-wmi.ko
LD [M] drivers/platform/x86/acerhdf.ko
LD [M] drivers/platform/x86/amllo-rfkill.ko
LD [M] drivers/platform/x86/alienware-wmi.ko
LD [M] drivers/platform/x86/apple-gmux.ko
LD [M] drivers/platform/x86/asus-laptop.ko
LD [M] drivers/platform/x86/asus-nb-wmi.ko
LD [M] drivers/platform/x86/asus-wireless.ko
LD [M] drivers/platform/x86/asus-wmi.ko
LD [M] drivers/platform/x86/classmate-laptop.ko
LD [M] drivers/platform/x86/compal-laptop.ko
LD [M] drivers/platform/x86/dell-laptop.ko
LD [M] drivers/platform/x86/dell-rbtn.ko
LD [M] drivers/platform/x86/dell-smbios.ko
LD [M] drivers/platform/x86/dell-smo8800.ko
LD [M] drivers/platform/x86/dell-wmi-aio.ko
LD [M] drivers/platform/x86/dell-wmi-descriptor.ko
LD [M] drivers/platform/x86/dell-wmi-led.ko
LD [M] drivers/platform/x86/dell-wmi.ko
LD [M] drivers/platform/x86/eeepc-wmi.ko
CONFIG_EARLY_PRINTK_EFI=y
# CONFIG_EARLY_PRINTK_USB_XDBC is not set
CONFIG_X86_PTDUMP_CORE=v
```

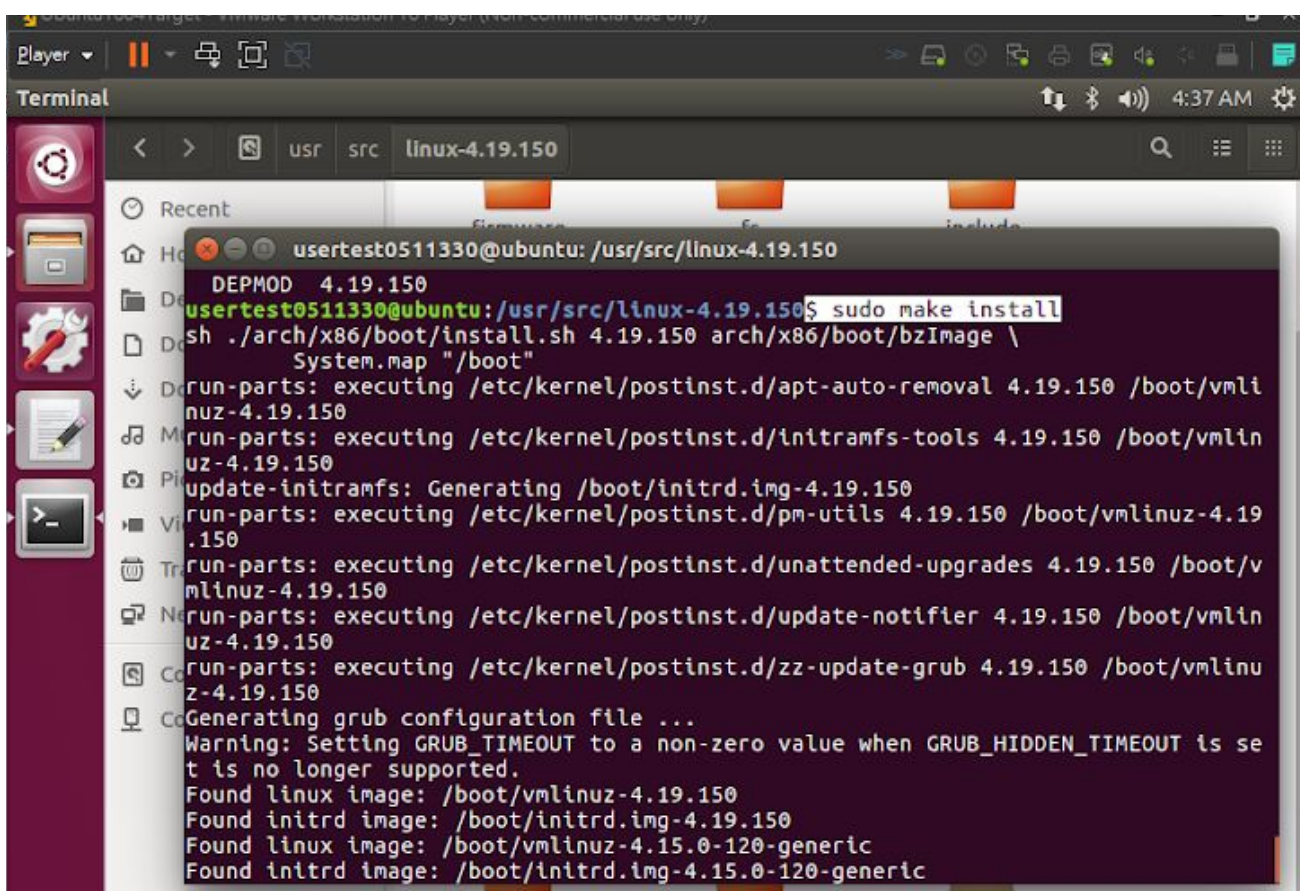
### Screenshot 7, 8 -

use the command: **\$sudo make modules\_install**, **\$sudo make install**  
Install the modules which we've just compiled to the default path.



The screenshot shows a terminal window with the command `sudo make modules_install` being executed. The output lists several kernel modules being installed to `/lib/modules/4.19.150`. The modules include `arch/x86/crypto/aes-x86_64.ko`, `arch/x86/crypto/aesni-intel.ko`, `arch/x86/crypto/blowfish-x86_64.ko`, `arch/x86/crypto/camellia-aesni-avx-x86_64.ko`, `arch/x86/crypto/camellia-aesni-avx2.ko`, `arch/x86/crypto/camellia-x86_64.ko`, `arch/x86/crypto/cast5-avx-x86_64.ko`, `arch/x86/crypto/cast6-avx-x86_64.ko`, `arch/x86/crypto/chacha20-x86_64.ko`, `arch/x86/crypto/crc32-pclmul.ko`, `arch/x86/crypto/crc10dif-pclmul.ko`, `arch/x86/crypto/des3_ede-x86_64.ko`, and `arch/x86/crypto/ghash-clmulni-intel.ko`. The terminal window also shows a file manager in the background with the path `usr/src/linux-4.19.150`.

```
userstest0511330@ubuntu: /usr/src/linux-4.19.150
mkdir: cannot create directory '.tmp_77911': Permission denied
mkdir: cannot create directory '.tmp_77913': Permission denied
mkdir: cannot create directory '.tmp_77915': Permission denied
mkdir: cannot create directory '.tmp_77917': Permission denied
mkdir: cannot create directory '.tmp_77922': Permission denied
mkdir: cannot create directory '.tmp_77926': Permission denied
mkdir: cannot create directory '/lib/modules/4.19.150': Permission denied
Makefile:1284: recipe for target '_modinst_' failed
make: *** [_modinst_] Error 1
userstest0511330@ubuntu: /usr/src/linux-4.19.150$ sudo make modules_install
[sudo] password for userstest0511330:
INSTALL arch/x86/crypto/aes-x86_64.ko
INSTALL arch/x86/crypto/aesni-intel.ko
INSTALL arch/x86/crypto/blowfish-x86_64.ko
INSTALL arch/x86/crypto/camellia-aesni-avx-x86_64.ko
INSTALL arch/x86/crypto/camellia-aesni-avx2.ko
INSTALL arch/x86/crypto/camellia-x86_64.ko
INSTALL arch/x86/crypto/cast5-avx-x86_64.ko
INSTALL arch/x86/crypto/cast6-avx-x86_64.ko
INSTALL arch/x86/crypto/chacha20-x86_64.ko
INSTALL arch/x86/crypto/crc32-pclmul.ko
INSTALL arch/x86/crypto/crc10dif-pclmul.ko
INSTALL arch/x86/crypto/des3_ede-x86_64.ko
INSTALL arch/x86/crypto/ghash-clmulni-intel.ko
```



The screenshot shows a terminal window with the command `sudo make install` being executed. The output shows the installation of the kernel image and the generation of the grub configuration file. The terminal window also shows a file manager in the background with the path `usr/src/linux-4.19.150`.

```
DEPMOD 4.19.150
userstest0511330@ubuntu: /usr/src/linux-4.19.150$ sudo make install
sh ./arch/x86/boot/install.sh 4.19.150 arch/x86/boot/bzImage \
    System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 4.19.150 /boot/vmlinuz-4.19.150
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 4.19.150 /boot/vmlinuz-4.19.150
update-initramfs: Generating /boot/initrd.img-4.19.150
run-parts: executing /etc/kernel/postinst.d/pm-utils 4.19.150 /boot/vmlinuz-4.19.150
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 4.19.150 /boot/vmlinuz-4.19.150
run-parts: executing /etc/kernel/postinst.d/update-notifier 4.19.150 /boot/vmlinuz-4.19.150
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 4.19.150 /boot/vmlinuz-4.19.150
Generating grub configuration file ...
Warning: Setting GRUB_TIMEOUT to a non-zero value when GRUB_HIDDEN_TIMEOUT is set is no longer supported.
Found linux image: /boot/vmlinuz-4.19.150
Found initrd image: /boot/initrd.img-4.19.150
Found linux image: /boot/vmlinuz-4.15.0-120-generic
Found initrd image: /boot/initrd.img-4.15.0-120-generic
```



## [Section 3-4(2)] Grub update(2)

### Screenshot 9

We update the grub through the commands:

```
$sudo update-initramfs -c -k 4.19.150
```

```
$sudo update grub
```

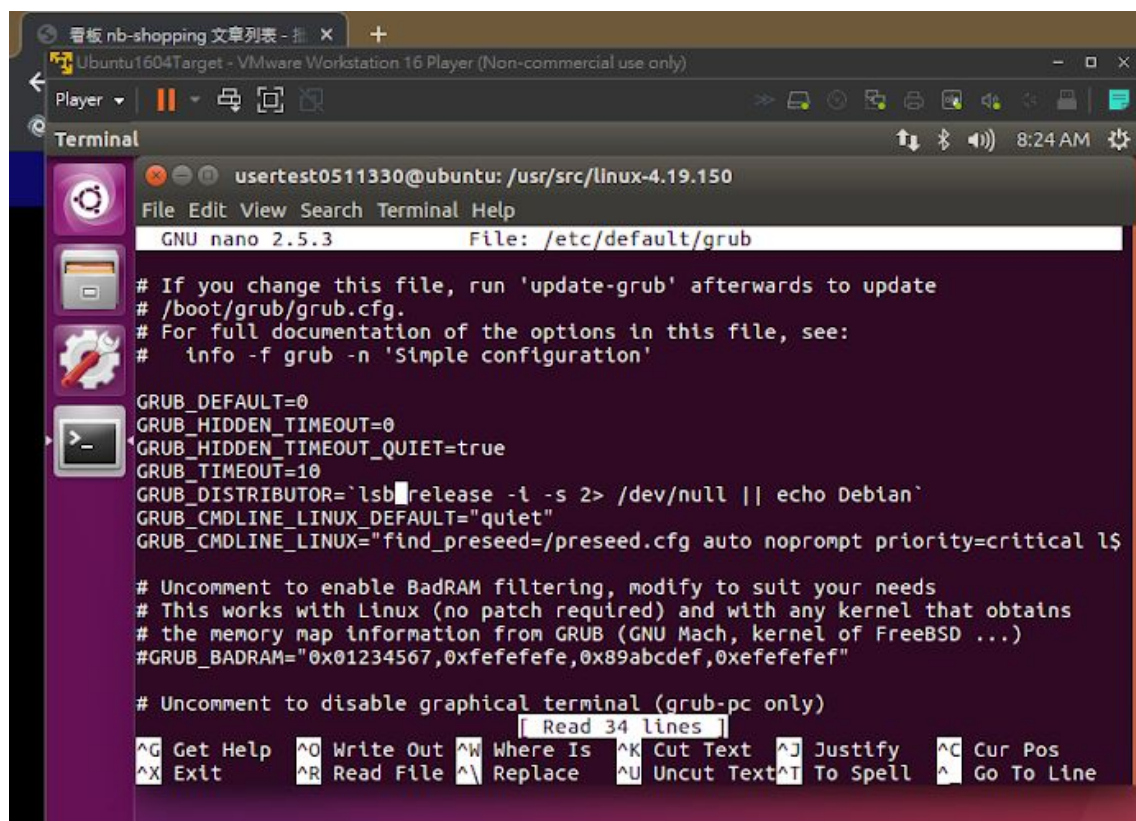
then we modify the settings in grub files through the command:

```
$sudo nano /etc/default/grub
```

disable the following 2 settings.

```
#GRUB_HIDDEN_TIMEOUT=0
```

```
#GRUB_HIDDEN_TIMEOUT_QUIET=true
```



```
看板 nb-shopping 文章列表 - 推 X +
Ubuntu1604Target - VMware Workstation 16 Player (Non-commercial use only)
Player [Icons] [Settings] [Full Screen] [Help]
Terminal 8:24 AM
user@ubuntu: /usr/src/linux-4.19.150
File Edit View Search Terminal Help
GNU nano 2.5.3 File: /etc/default/grub

# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
# info -f grub -n 'Simple configuration'

GRUB_DEFAULT=0
GRUB_HIDDEN_TIMEOUT=0
GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT=10
GRUB_DISTRIBUTOR=`lsb_release -t -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet"
GRUB_CMDLINE_LINUX="find_preseed=/preseed.cfg auto noprompt priority=critical ls

# Uncomment to enable BadRAM filtering, modify to suit your needs
# This works with Linux (no patch required) and with any kernel that obtains
# the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)
#GRUB_BADRAM="0x01234567,0xfefefefe,0x89abcdef,0xefefefef"

# Uncomment to disable graphical terminal (grub-pc only)
Read 34 lines
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line
```

### Screenshot 10, 11 - Assign the serial port for gdb connection.

Open the grub.cfg under the path `/boot/grub/grub.cfg`, and search for the keyword "Ubuntu". In addition, Remember the line number.

Then we use nano to add two commands at the end of the line :

**kgdbwait**

**kgdboc=ttyS1,115200**

kgdbwait: when we boot the machine, the kernel will wait for the gdb connection.

kgdboc: kgdb over console, we choose the second serial port (**ttyS1**) as console, and the baud rate is set to **115200** (bits per second).

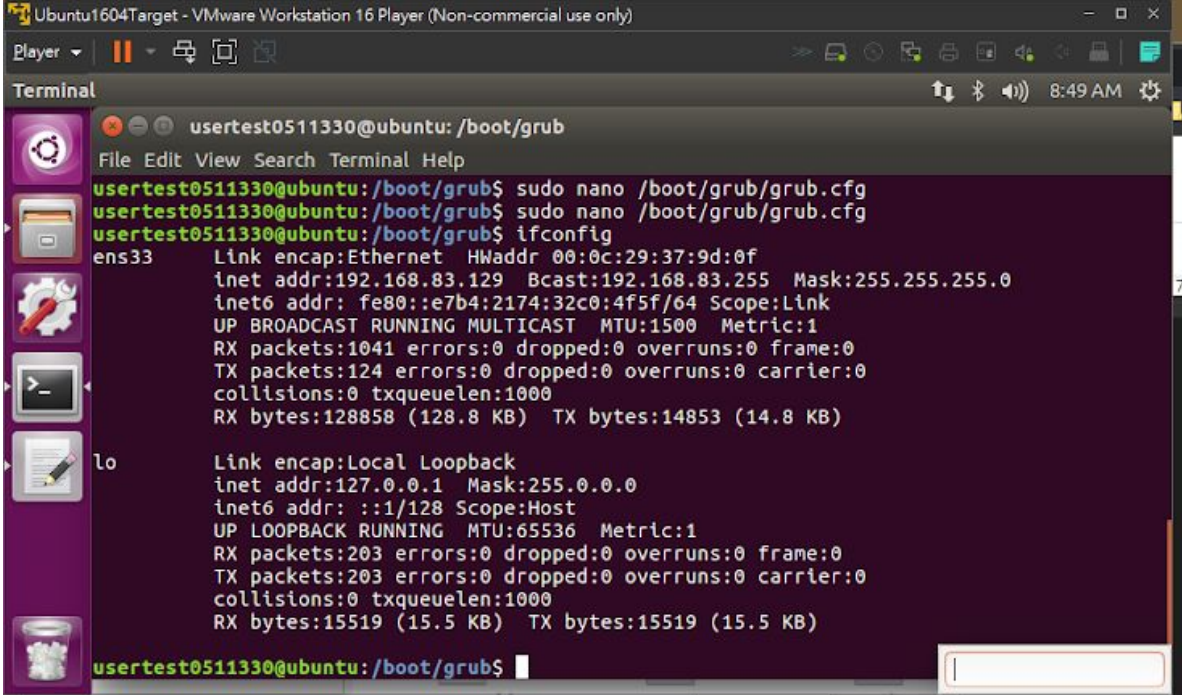
After things being settled down, when we boot the machine, the kernel will wait for the gdb connection and we can debug remotely.

### Screenshot 12 - building ssh protocol

The scp command means we can remotely command the target through the ssh protocol.

**\$sudo scp usertest0511330@192.168. 83.129:/usr/src/linux-4.19.150/vmlinux .**

Note that we can use the **\$ifconfig** to find the Target IP : 192.168.83.129



```
Ubuntu1604Target - VMware Workstation 16 Player (Non-commercial use only)
Player
Terminal
usertest0511330@ubuntu: /boot/grub
File Edit View Search Terminal Help
usertest0511330@ubuntu:/boot/grub$ sudo nano /boot/grub/grub.cfg
usertest0511330@ubuntu:/boot/grub$ sudo nano /boot/grub/grub.cfg
usertest0511330@ubuntu:/boot/grub$ ifconfig
ens33      Link encap:Ethernet  HWaddr 00:0c:29:37:9d:0f
           inet addr:192.168.83.129  Bcast:192.168.83.255  Mask:255.255.255.0
           inet6 addr: fe80::e7b4:2174:32c0:4f5f/64 Scope:Link
           UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
           RX packets:1041 errors:0 dropped:0 overruns:0 frame:0
           TX packets:124 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:1000
           RX bytes:128858 (128.8 KB)  TX bytes:14853 (14.8 KB)

lo         Link encap:Local Loopback
           inet addr:127.0.0.1  Mask:255.0.0.0
           inet6 addr: ::1/128 Scope:Host
           UP LOOPBACK RUNNING  MTU:65536  Metric:1
           RX packets:203 errors:0 dropped:0 overruns:0 frame:0
           TX packets:203 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:1000
           RX bytes:15519 (15.5 KB)  TX bytes:15519 (15.5 KB)

usertest0511330@ubuntu:/boot/grub$
```

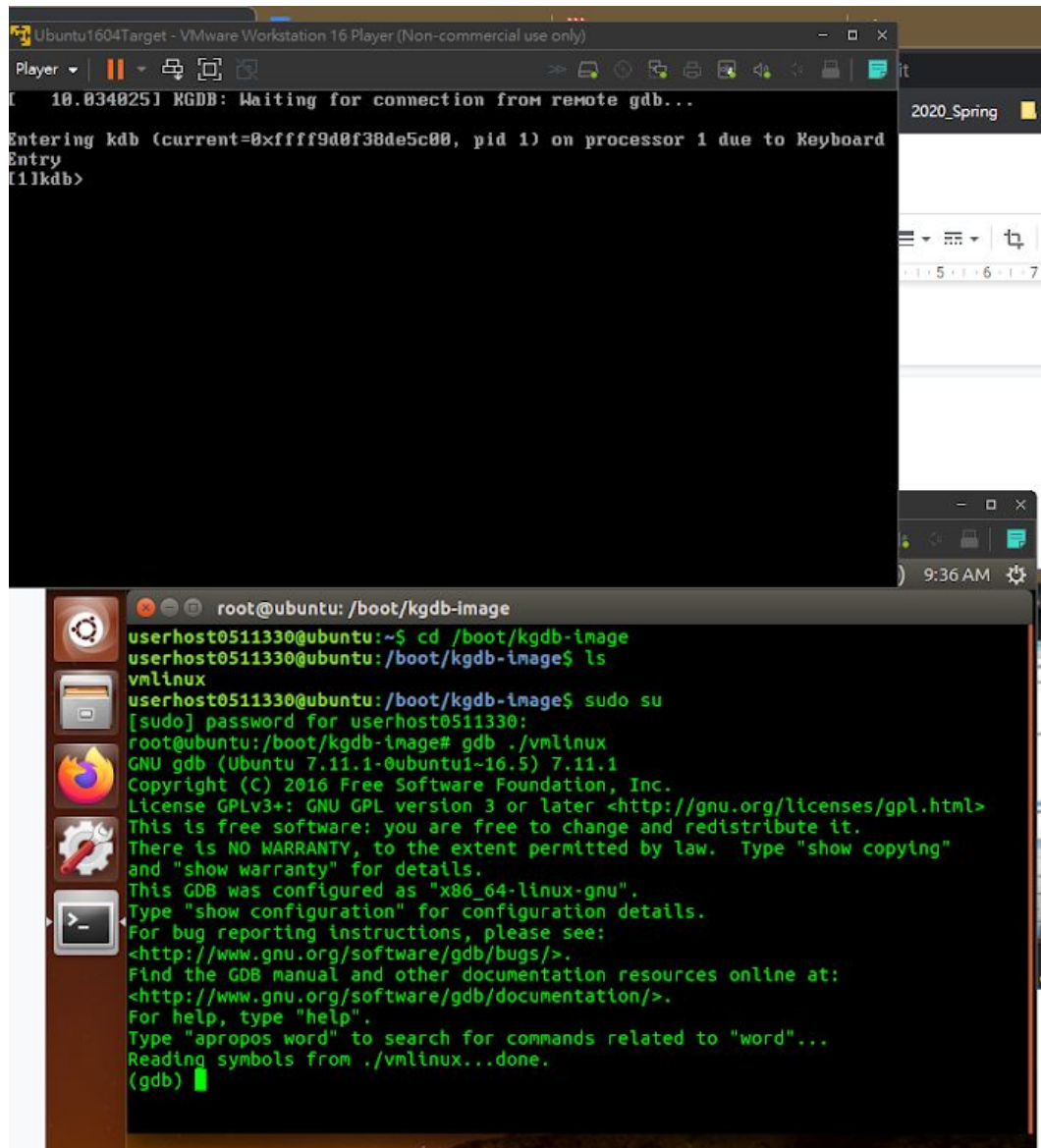
## [Section 4] Kernel Debugging

### Screenshot 13

As shown in the figure, the target is waiting for connection from remote gdb.

It means that the commands **kgdbwait & kgdboc=ttyS1,115200** works.

Now we need the highest authority of the Host machine, use the command **\$sudo su** to access root authority.



The screenshot shows two windows. The top window is a VMware Workstation 16 Player titled 'Ubuntu1604Target - VMware Workstation 16 Player (Non-commercial use only)'. It displays a terminal window with the following text:

```
10.034025] KGDB: Waiting for connection from remote gdb...  
Entering kdb (current=0xffff9d0f38de5c00, pid 1) on processor 1 due to Keyboard  
Entry  
[1]kdb>
```

The bottom window is a terminal on the host machine, titled 'root@ubuntu: /boot/kgdb-image'. It shows the following commands and output:

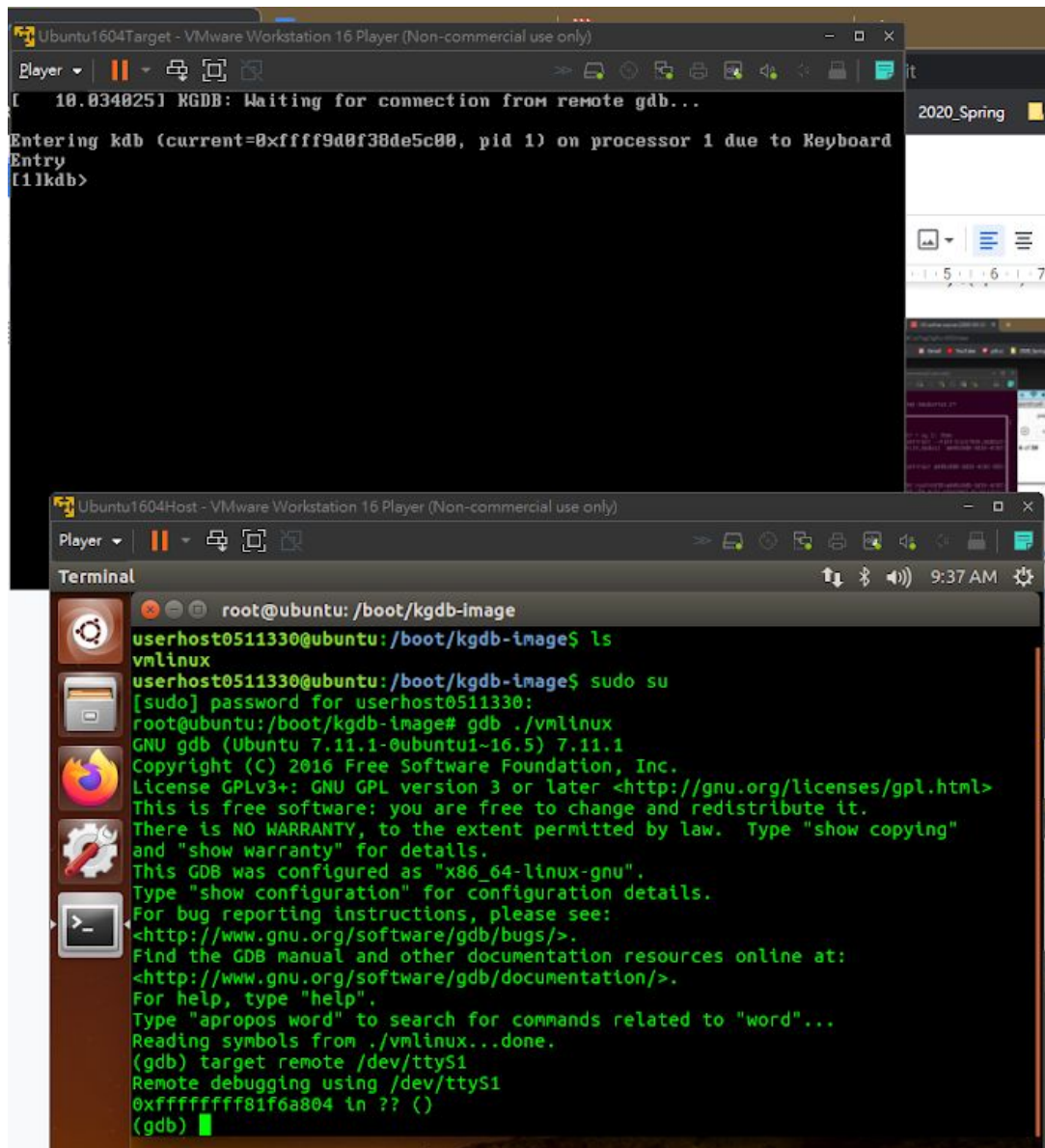
```
userhost0511330@ubuntu:~$ cd /boot/kgdb-image  
userhost0511330@ubuntu:/boot/kgdb-image$ ls  
vmlinux  
userhost0511330@ubuntu:/boot/kgdb-image$ sudo su  
[sudo] password for userhost0511330:  
root@ubuntu:/boot/kgdb-image# gdb ./vmlinux  
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1  
Copyright (C) 2016 Free Software Foundation, Inc.  
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>  
This is free software: you are free to change and redistribute it.  
There is NO WARRANTY, to the extent permitted by law. Type "show copying"  
and "show warranty" for details.  
This GDB was configured as "x86_64-linux-gnu".  
Type "show configuration" for configuration details.  
For bug reporting instructions, please see:  
<http://www.gnu.org/software/gdb/bugs/>.  
Find the GDB manual and other documentation resources online at:  
<http://www.gnu.org/software/gdb/documentation/>.  
For help, type "help".  
Type "apropos word" to search for commands related to "word"...  
Reading symbols from ./vmlinux...done.  
(gdb)
```



#### Screenshot 14

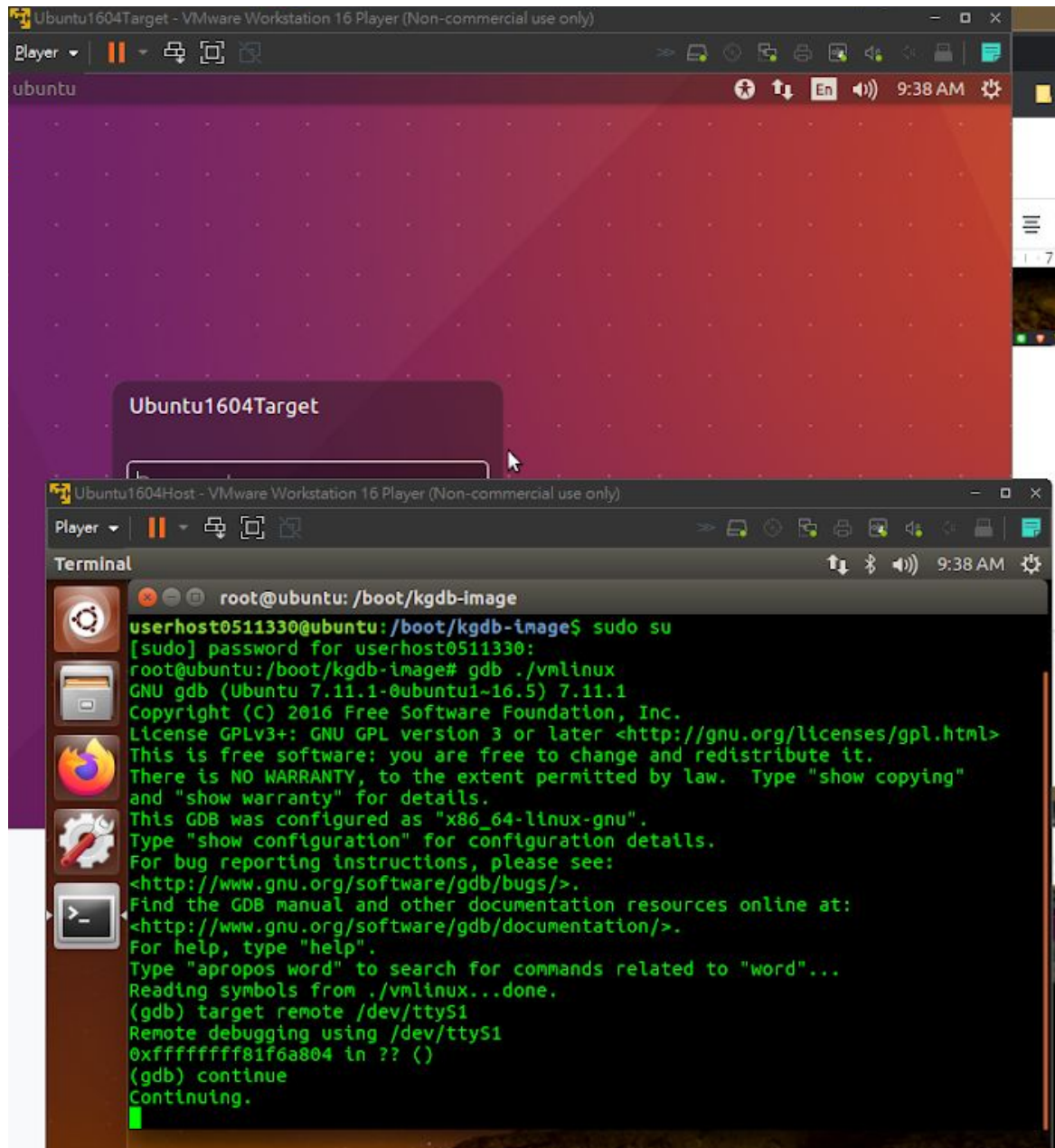
Then we run the command `# gdb ./vmlinux` and `(gdb) target remote /dev/ttyS1`.

Finally, when `(gdb)` shows on the terminal, it means that we will be able to remotely debug the patched kernel on the target machine.



### Screenshot 15 -

When kernel debugging is done, runs the command **(gdb) continue**. The target will be able to boot into the OS.



## **Q&A**

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

mainline: 由Linus Torvalds親自制作的kernel版本 (Mainline tree is maintained by Linus Torvalds.)

stable: Stable version. After each mainline kernel is released, it is considered "stable."

longterm: Long term support version. There are usually several "longterm maintenance" kernel releases provided for the purposes of backporting bugfixes for older kernel trees. Only important bugfixes are applied to such kernels and they don't usually see very frequent releases, especially for older trees.

kernel panic: Kernel panic occurs when there's some fatal error founded by Operating System. The machine will freeze and we need to reboot it.

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

debugging: debugging is the process of running a debug session in a local development environment attached to a remotely deployed application.

profiling: profiling is the process of measuring an application or system by running an analysis tool called a profiler.

building: compile the source code.

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

(As mentioned in the previous page.)

GCC: **G**NU **C**ompiler **C**ollection, a compiler used for compiling C Language.

GDB: **G**NU **D**ebugger, a debugger used for software development by high-level Language.

KGDB: **K**ernel **G**NU **D**ebugger, a debugger used when developing the kernel.

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

/usr : 存放系統資訊和目錄，目錄用來存放程式與指令。

/boot/ : 你的開機檔所在位置，這裡就是放置你 Linux 核心與開機相關檔案的地方。

/boot/grub : grub bootloader 的程式碼也放在開機檔所在位置，當你開機時grub會引導你做開機後選擇。

/home : 系統預設的home directory，新版本的Linux中，http和ftp等程式的home directory也會存放在此。

#### **5. What are the general steps to debug a Linux Kernel?**

1. download the source code
2. prebuilt the config file
3. patch the kernel and build it.
4. Setup grub and kgdb settings. (vmlinux files)
5. used kgdb to remotely debug the kernel.

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

Since each time we recompile the kernel take us a long time. Besides, after the compilation, the system may freeze due to others bugs.

So we use the kgdb to remotely debug the specific partition of the kernel code, then it will be more efficient when developing the kernel.

#### **7. In Section 3.3, what are the differences between make, make modules\_install and make install?**

make: the command is used to compile the kernel.

make\_modules\_install: the command is used to download the kernel's modules.

make install: install the kernel which has been built int /.vmlinux file.



**8. In Section 3.4, what are the commands kgdbwait and kgdboc=ttyS1,115200 for?**

kgdbwait: grub will told the target machine wait for gdb connection instead of booting into the system

kgdboc=ttyS1,115200: kgdb over console, we choose the second serial port (**ttyS1**) as console, and the baud rate is set to **115200** (bits per second).

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

(As mentioned in the first page)

**GNU Grub**是一個Opensouce的專案，是一種bootloader，讓電腦可以安裝多種作業系統，幫助引導到使用者到欲使用的作業系統上。

**Grub.cfg**是Grub config files，能設定開機後Grub需要的參數，例如kgdbwait就是一種參數，能讓電腦等待kgdb連線。

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

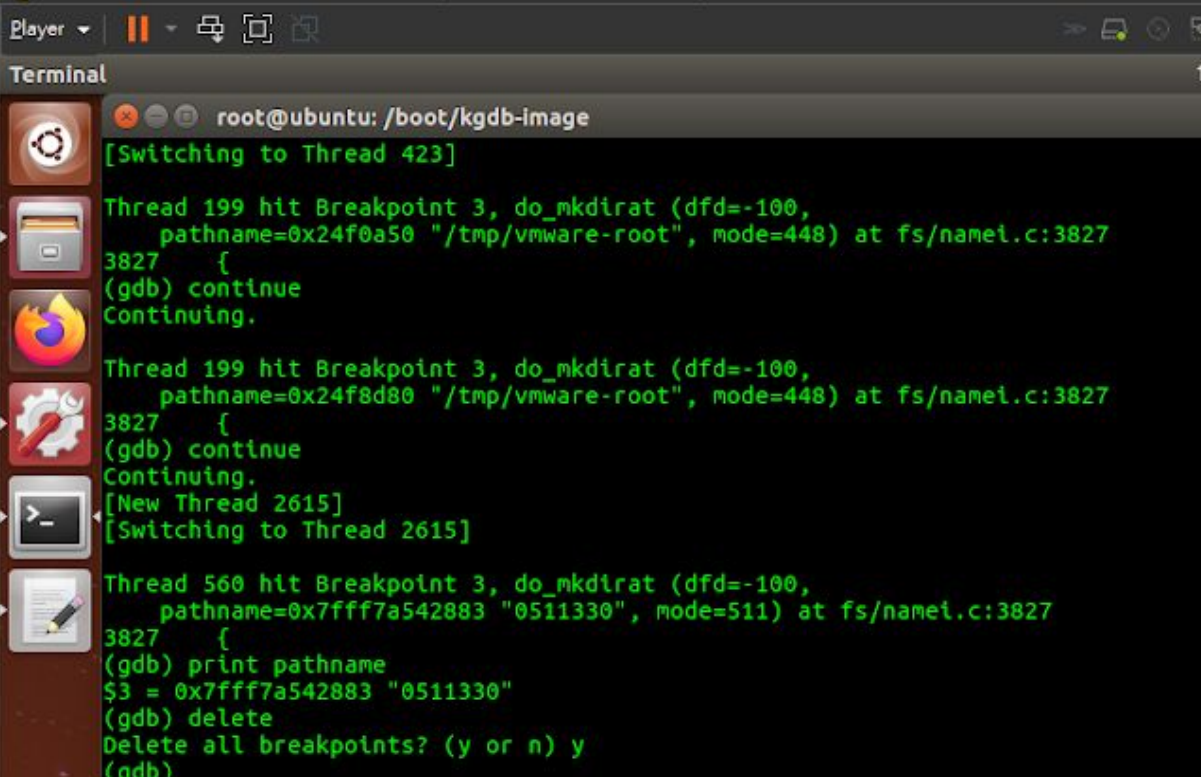
print: 印出變數內容。  
whatis: 印出變數的型態。  
continue: 繼續執行。和 breakpoint 搭配使用。  
breakpoint: 設定中斷點。  
clear/delete: 刪除某個/全部的 breakpoint。  
next: 單步執行。  
kill: 中止程式的執行。  
list: 印出程式碼。  
quit: 離開 gdb。  
backtrace: 堆疊追蹤。  
run: 執行程式，或是從頭再執行程式。

## gdb commands screenshot

### 1.Delete breakpoint

(gdb) delete

delete all breakpoints



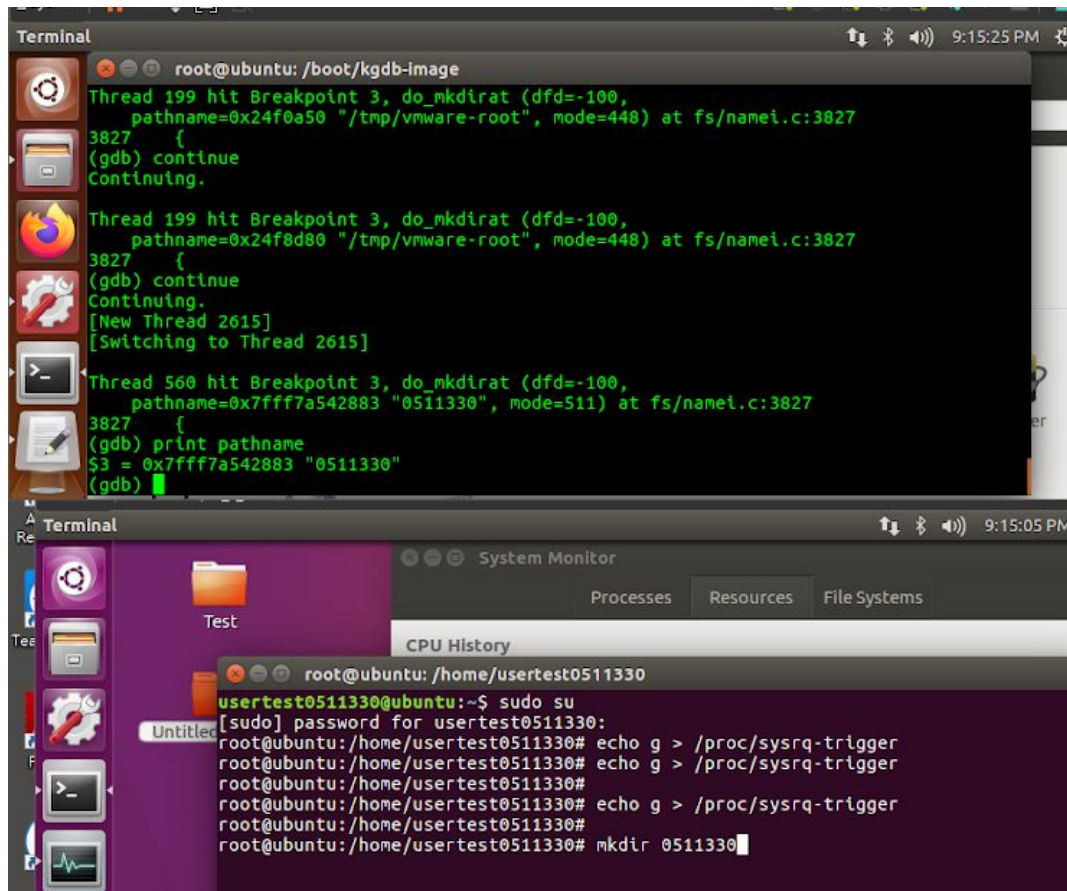
The screenshot shows a terminal window titled "Terminal" with a dark background. The prompt is "root@ubuntu: /boot/kgdb-image". The output shows several threads hitting breakpoint 3 in the function do\_mkdirat. The user enters "(gdb) continue" and "(gdb) delete". The prompt then asks "Delete all breakpoints? (y or n) y".

```
root@ubuntu: /boot/kgdb-image
[Switching to Thread 423]
Thread 199 hit Breakpoint 3, do_mkdirat (dfd=-100,
    pathname=0x24f0a50 "/tmp/vmware-root", mode=448) at fs/namei.c:3827
3827  {
(gdb) continue
Continuing.
Thread 199 hit Breakpoint 3, do_mkdirat (dfd=-100,
    pathname=0x24f8d80 "/tmp/vmware-root", mode=448) at fs/namei.c:3827
3827  {
(gdb) continue
Continuing.
[New Thread 2615]
[Switching to Thread 2615]
Thread 560 hit Breakpoint 3, do_mkdirat (dfd=-100,
    pathname=0x7fff7a542883 "0511330", mode=511) at fs/namei.c:3827
3827  {
(gdb) print pathname
$3 = 0x7fff7a542883 "0511330"
(gdb) delete
Delete all breakpoints? (y or n) y
(gdb)
```

## 2. Set breakpoint

**(gdb) print [variable name]**

e.g. Print out the function's (syscall: mkdir) parameter "pathname", we got the name of directory "0511330"



The image consists of two screenshots of a Linux desktop environment. The top screenshot shows a terminal window with GDB debugging a process. The bottom screenshot shows a terminal window with a user test script and a System Monitor window in the background.

```
Terminal
root@ubuntu: /boot/kgdb-image
Thread 199 hit Breakpoint 3, do_mkdirat (dfd=-100,
  pathname=0x24f0a50 "/tmp/vmware-root", mode=448) at fs/namei.c:3827
3827 {
(gdb) continue
Continuing.
Thread 199 hit Breakpoint 3, do_mkdirat (dfd=-100,
  pathname=0x24f8d80 "/tmp/vmware-root", mode=448) at fs/namei.c:3827
3827 {
(gdb) continue
Continuing.
[New Thread 2615]
[Switching to Thread 2615]
Thread 560 hit Breakpoint 3, do_mkdirat (dfd=-100,
  pathname=0x7fff7a542883 "0511330", mode=511) at fs/namei.c:3827
3827 {
(gdb) print pathname
$3 = 0x7fff7a542883 "0511330"
(gdb)
```

```
Terminal
root@ubuntu: /home/user0511330
user0511330@ubuntu:~$ sudo su
[sudo] password for user0511330:
root@ubuntu: /home/user0511330# echo g > /proc/sysrq-trigger
root@ubuntu: /home/user0511330# echo g > /proc/sysrq-trigger
root@ubuntu: /home/user0511330# echo g > /proc/sysrq-trigger
root@ubuntu: /home/user0511330# echo g > /proc/sysrq-trigger
root@ubuntu: /home/user0511330# mkdir 0511330
```

The bottom screenshot also shows a System Monitor window with tabs for Processes, Resources, and File Systems. The CPU History tab is active, showing a graph of CPU usage over time.