Report 1-B

0710012 電資 11 何權祐

Video link: https://youtu.be/HIbxGNPOhXs

Questions:

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

   Kernel function: function that is called in the kernel operation, including access computer resource, memory management, and so on.

   System call: syscall, the interface between process and OS, when the user processes need the service from OS, it will use syscall to send request to the OS kernel, call kernel function, each kernel function has its syscall number

2. What is KASLR? What is it for?

   kernel address space layout randomization, to randomly place the address of the kernel, having a little offset from the link address, to make sure the outside won't have direct access to the kernel

3. What are GDB's non-stop and all-stop modes?

   Non-stop mode: when the thread stops to report a debugging event, only the thread is stopped

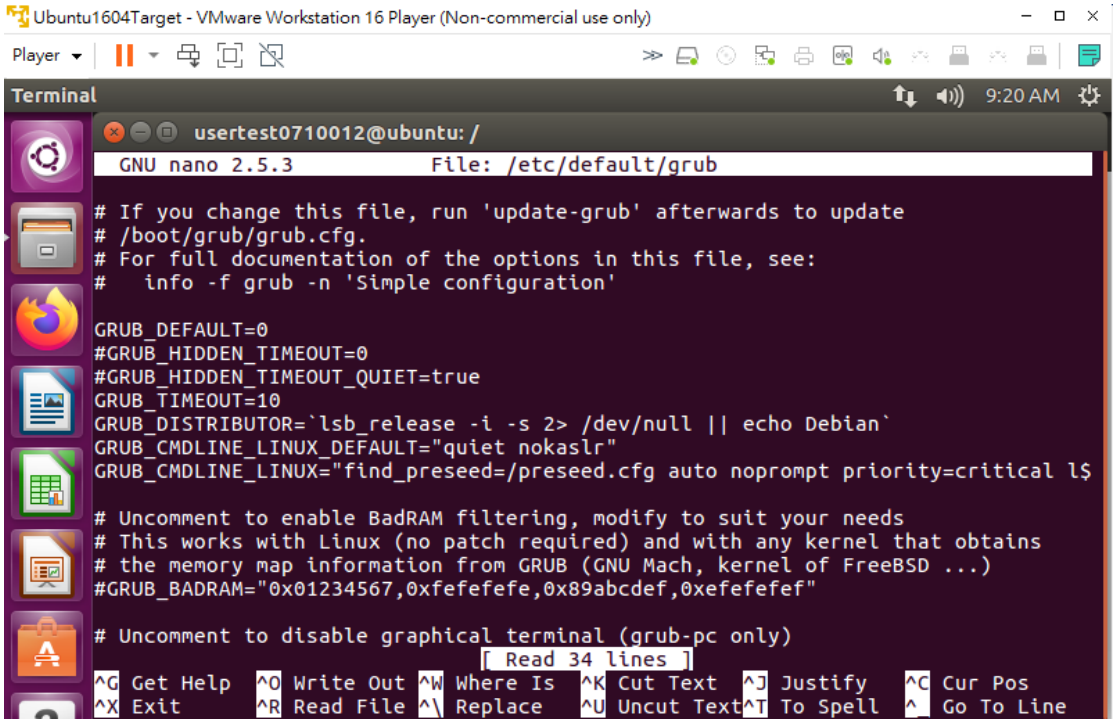
   All-stop mode: when the program stops under GDB for any reason, all threads of execution stop

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

   The command gives back the control of the vm back to the machine which is debugging the current machine with gdb

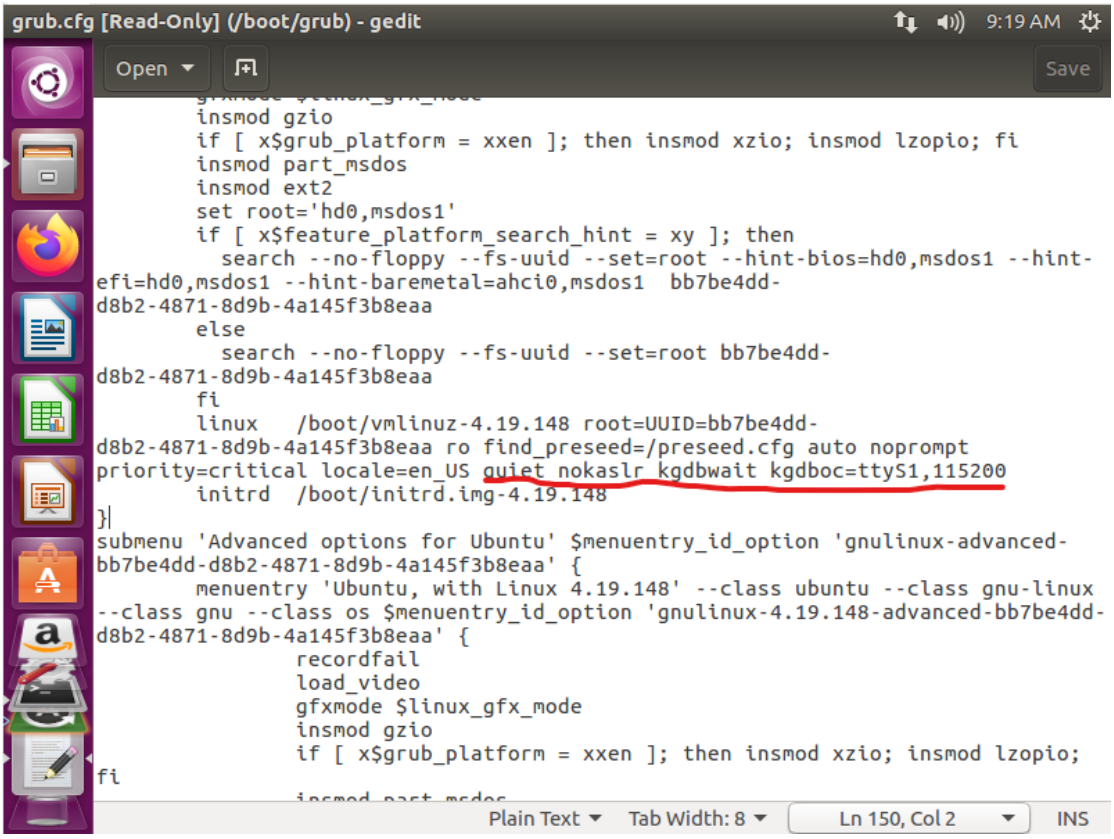
Screenshot 1:

Add “nokaslr” in the comment to make sure that the host machine can directly debug the kernel in target machine



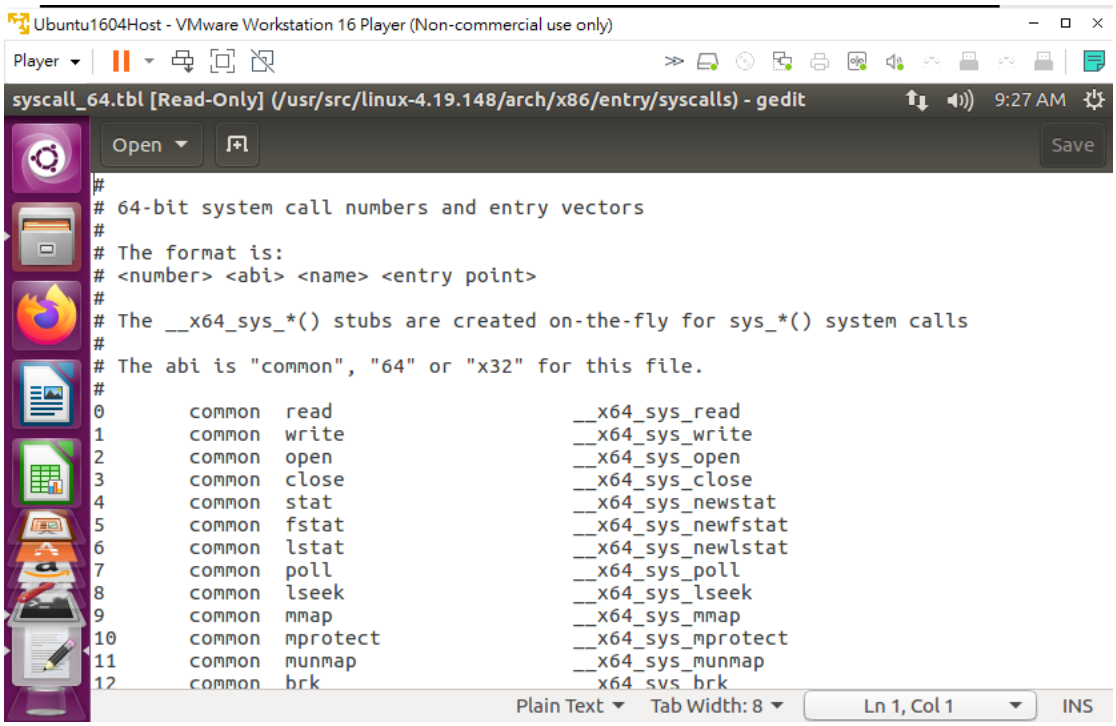
Screenshot 2:

Same as screenshot 1, to make sure that the host can get direct access to the kernel in target



Screenshot 3:

Directly check the kernel functions in the kernel package



Screenshot 4:

Make sure the syscall tables find online are coherent to one another

79	getcwd	man/ cs/	0x4f	char *buf	unsigned long size
80	chdir	man/ cs/	0x50	const char *filename	-
81	fchdir	man/ cs/	0x51	unsigned int fd	-
82	rename	man/ cs/	0x52	const char *oldname	const char *newname
83	mkdir	man/ cs/	0x53	const char *pathname	umode_t mode
84	rmdir	man/ cs/	0x54	const char *pathname	-
85	creat	man/ cs/	0x55	const char *pathname	umode_t mode

Screenshot 5:

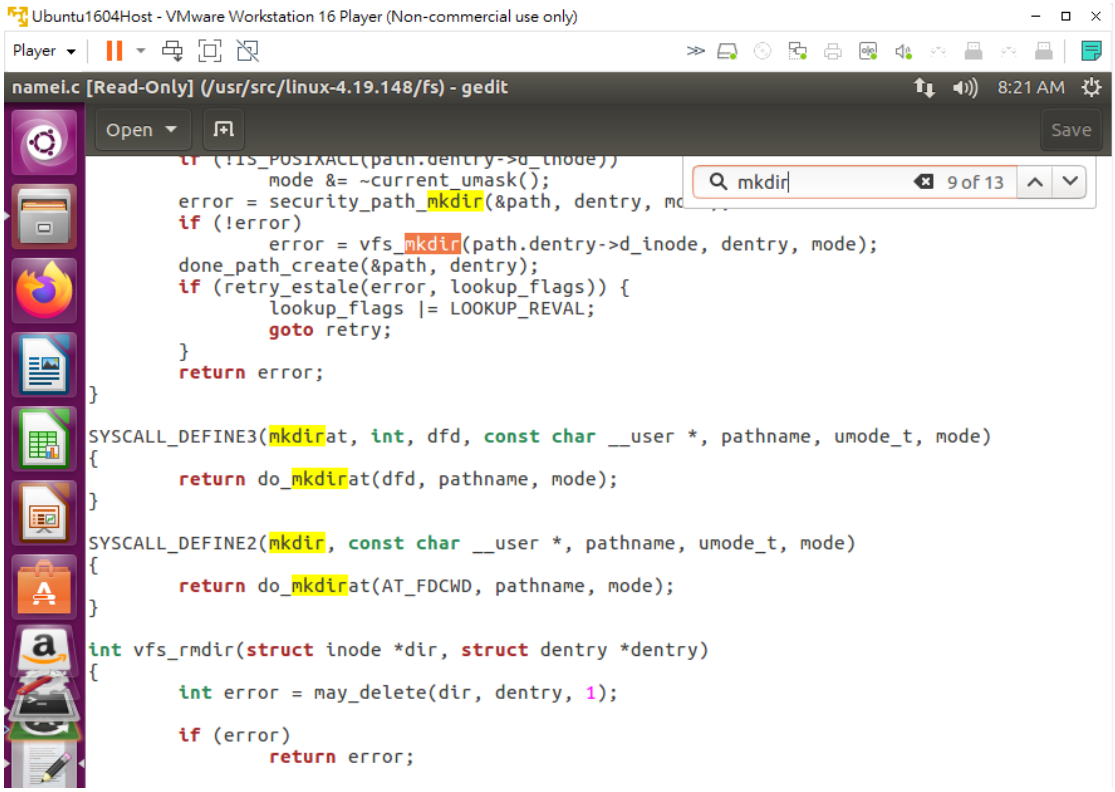
76	sys_truncate	const char *path	long length	
77	sys_ftruncate	unsigned int fd	unsigned long length	
78	sys_getdents	unsigned int fd	struct linux_dirent *dirent	unsigned int count
79	sys_getcwd	char *buf	unsigned long size	
80	sys_chdir	const char *filename		
81	sys_fchdir	unsigned int fd		
82	sys_rename	const char *oldname	const char *newname	
83	sys_mkdir	const char *pathname	int mode	
84	sys_rmdir	const char *pathname		
85	sys_creat	const char *pathname	int mode	
86	sys_link	const char *oldname	const char *newname	
87	sys_unlink	const char *pathname		

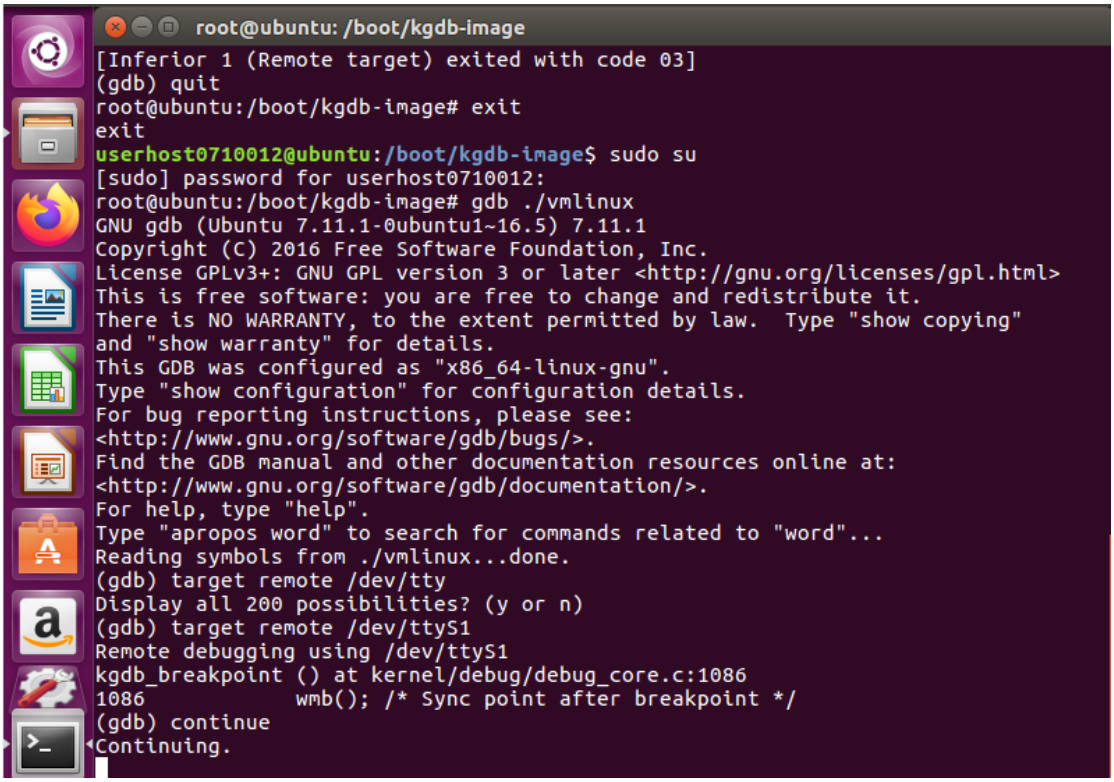
Screenshot 6:

76	truncate	sys_truncate	fs/open.c
77	ftruncate	sys_ftruncate	fs/open.c
78	getdents	sys_getdents	fs/readdir.c
79	getcwd	sys_getcwd	fs/dcache.c
80	chdir	sys_chdir	fs/open.c
81	fchdir	sys_fchdir	fs/open.c
82	rename	sys_rename	fs/namei.c
83	mkdir	sys_mkdir	fs/namei.c
84	rmdir	sys_rmdir	fs/namei.c
85	creat	sys_creat	fs/open.c
86	link	sys_link	fs/namei.c
87	unlink	sys_unlink	fs/namei.c

Screenshot 7:

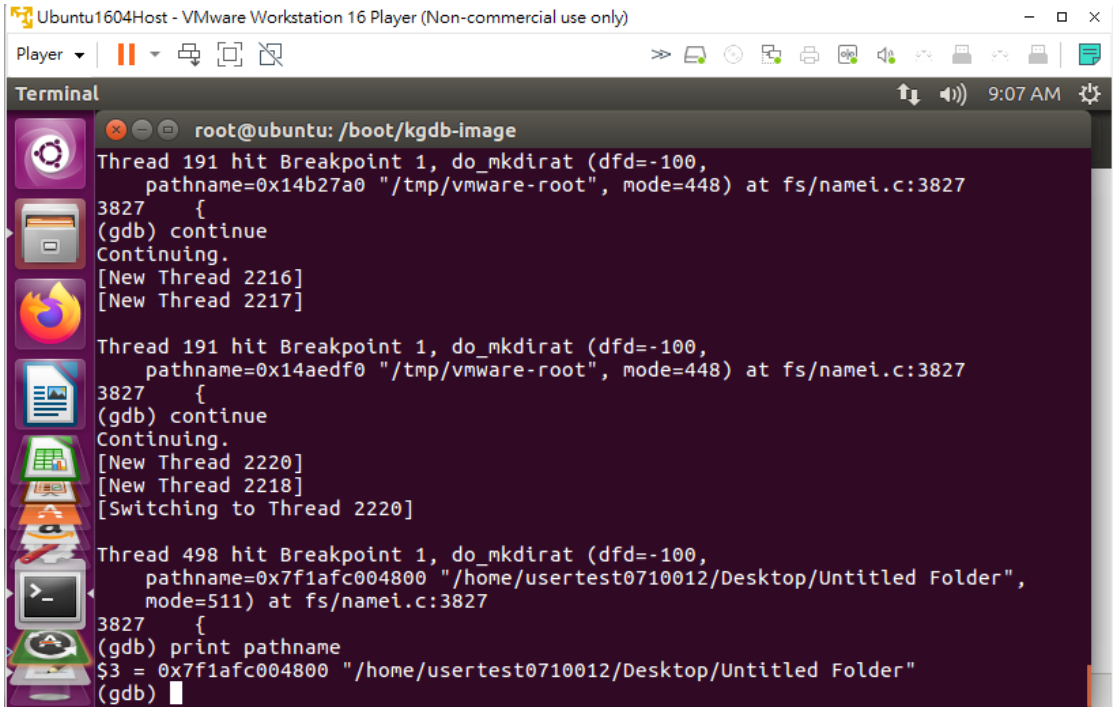
As we are making a break point in mkdir, we have to find the actual function that the syscall call, which is do_mkdirat, so we use command “break do_mkdirat” to set up the break point





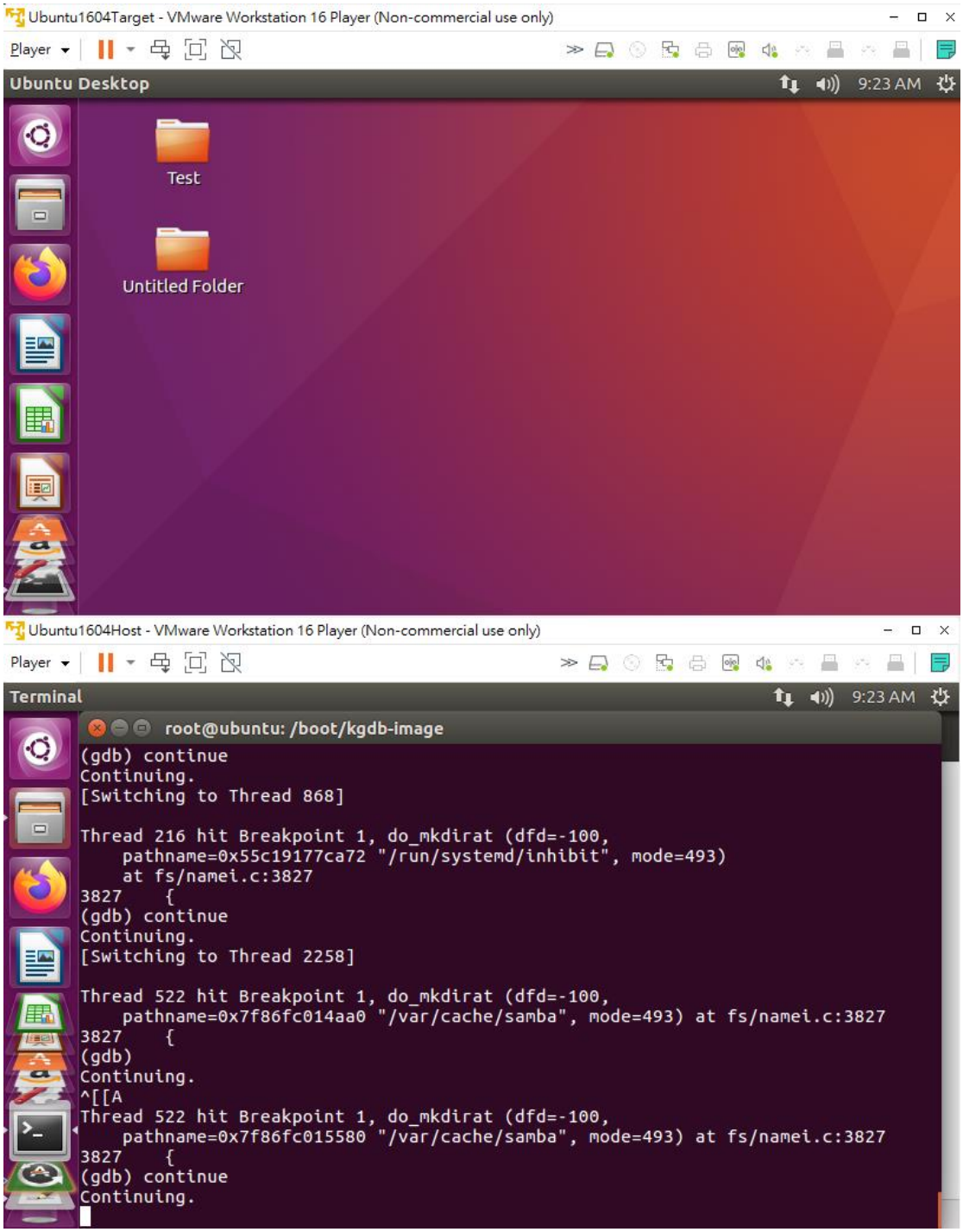
Screenshot 8:

When the break point is hit, the path will show where it is hit, the example is that we are trying to call mkdir in desktop by creating a new folder



Screenshot 9:

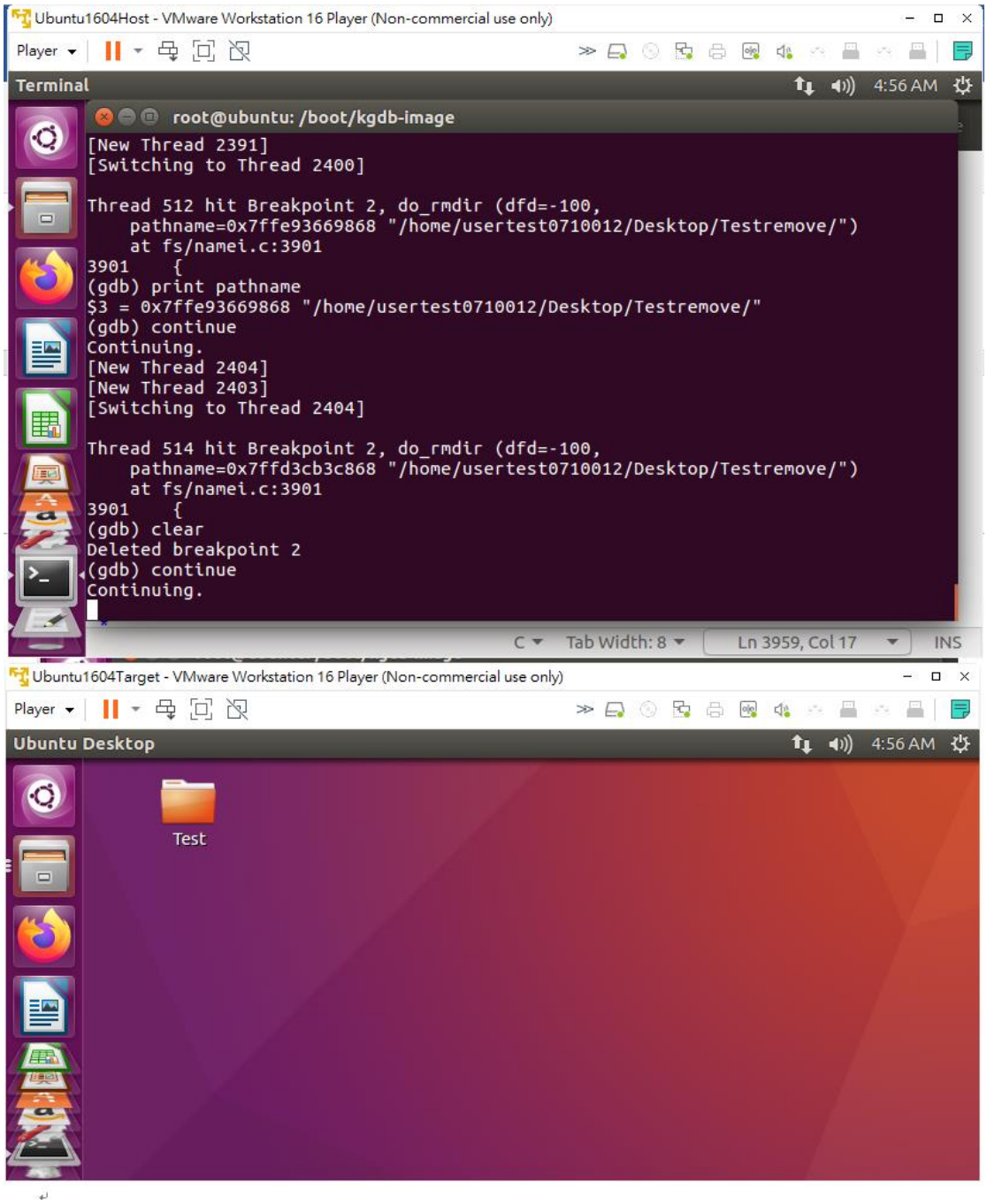
Actually we get through many break point before we reach the status that the target can back to normal mode, as the mkdir is called in many process that we don't know, however, at last, the folder is create successfully and the target resume to work mode



Do it yourself:

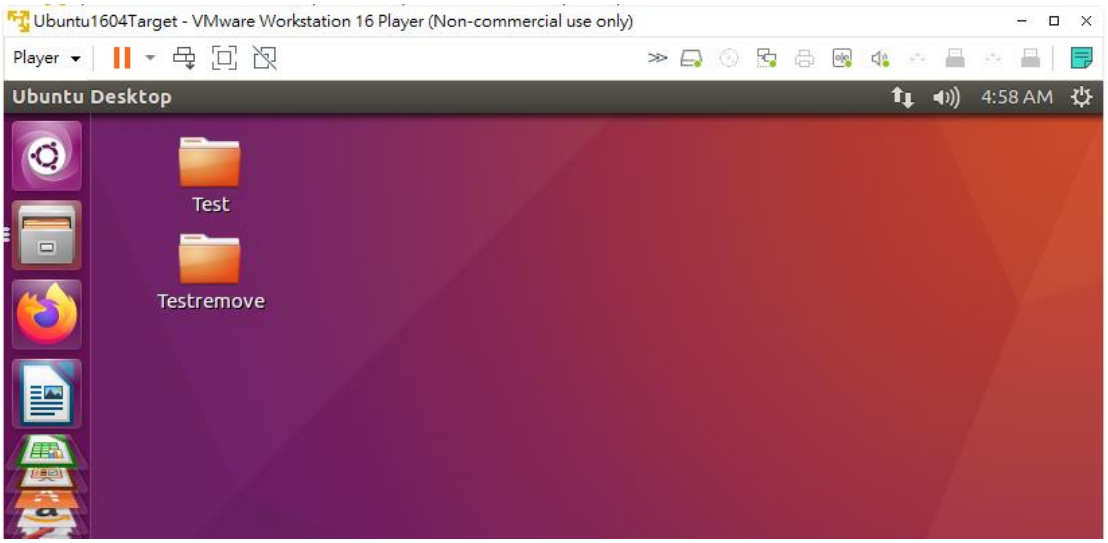
Screenshot 10:

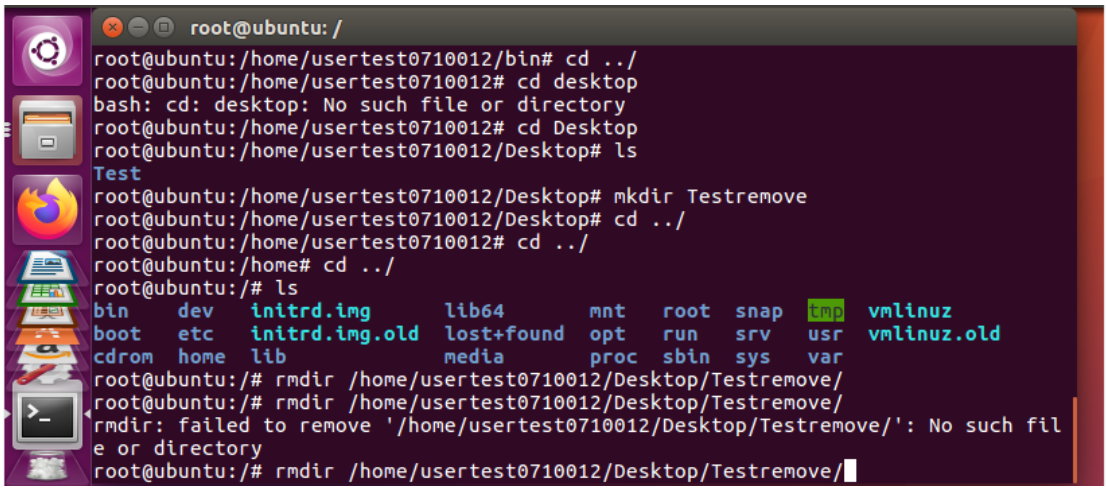
The machine is responsive



Screenshot 11:

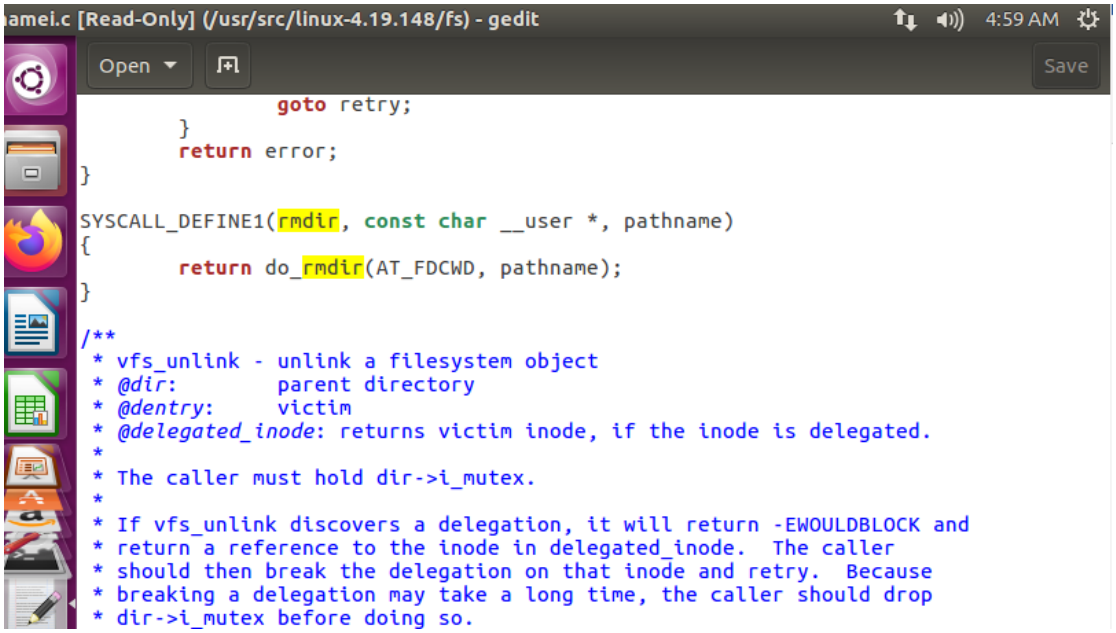
I choose the “rmdir” syscall function as the “do it yourself” example, and I plan to display the example by deleting a folder from the desktop, therefore, there is a folder called “Testremove” to help me explain the idea, and enter the rmdir syscall to hit the breakpoint





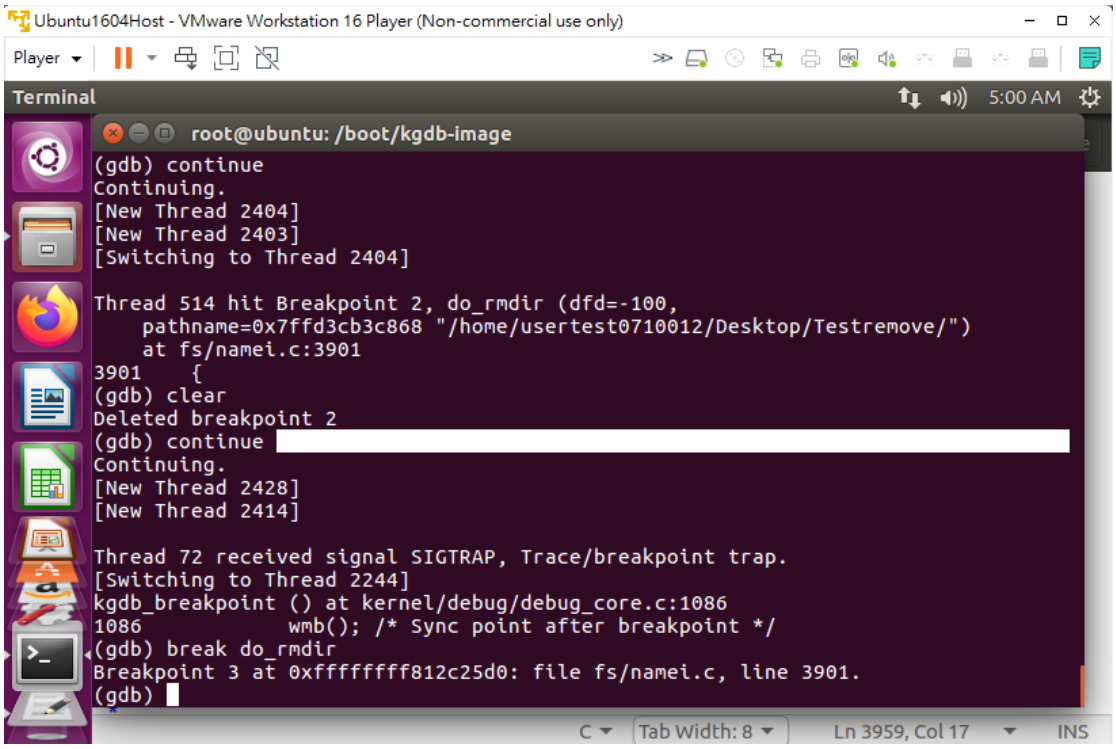
Screenshot 12:

Find the rmdir function in the syscall file and find out that the actual function it calls is do_rmdir



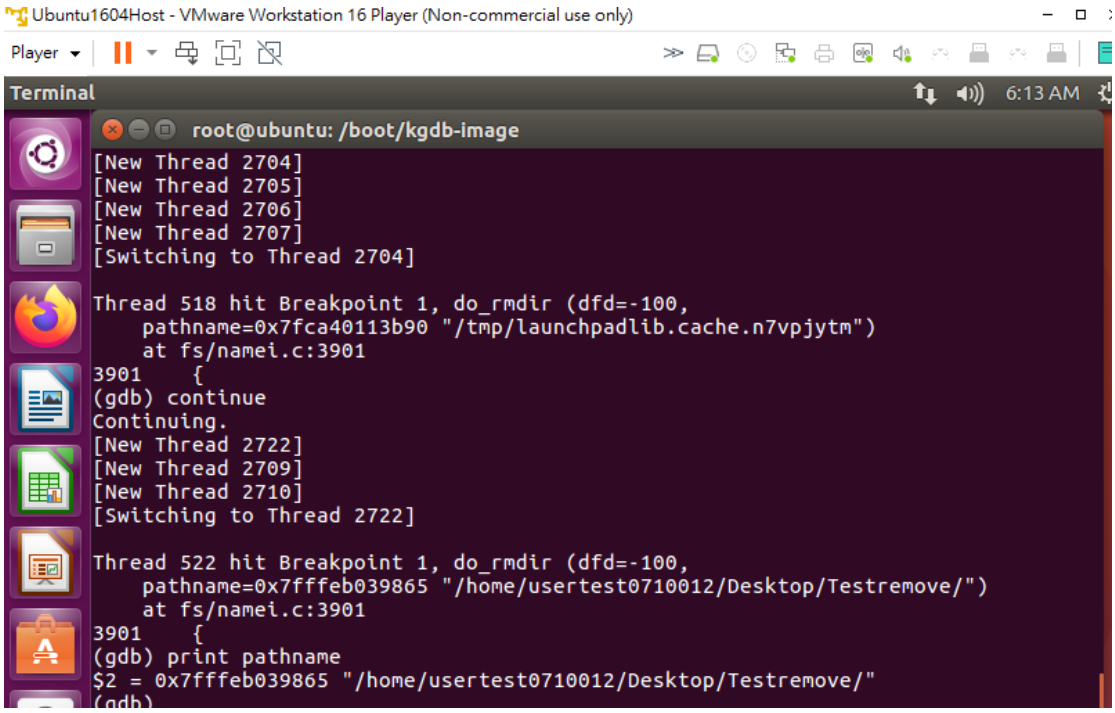
Screenshot 13:

Create a break point at do_rmdir



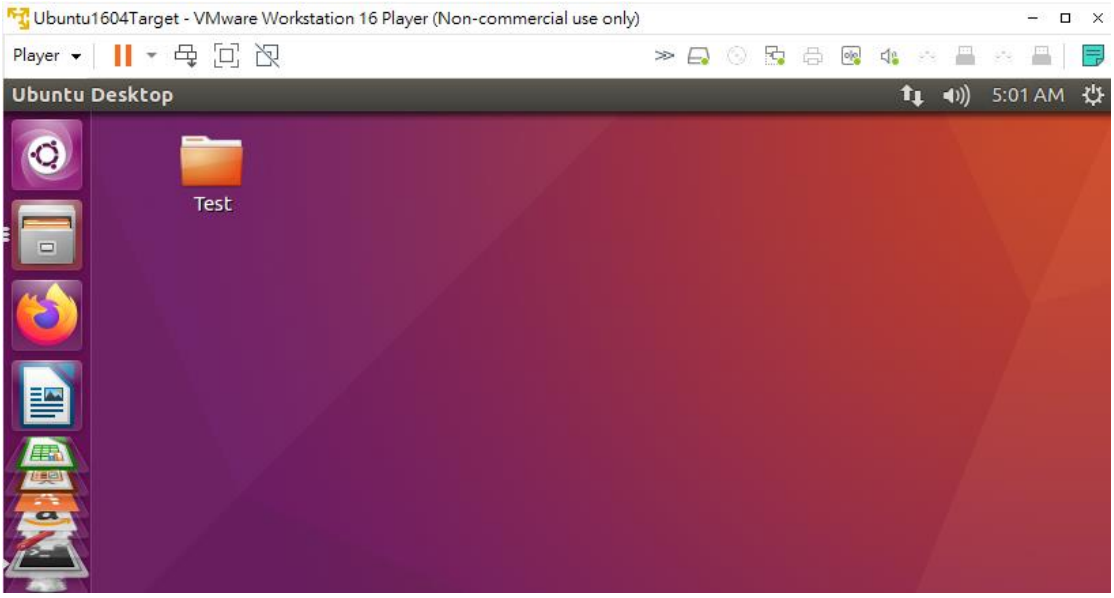
Screenshot 14:

The path name showed that the breakpoint in rmdir function is hit in /home/usertest0710012/Desktop/Testremove



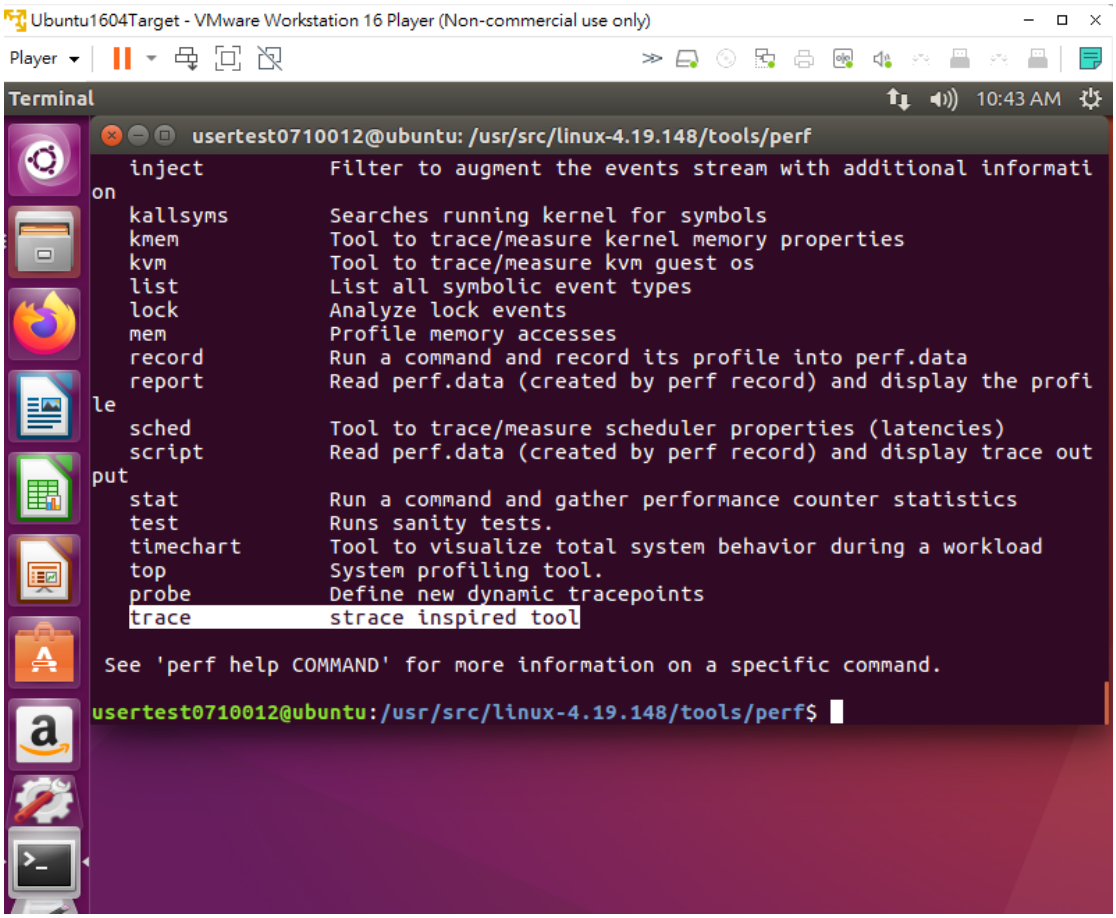
Screenshot 15:

The “Testremove” folder is successfully removed



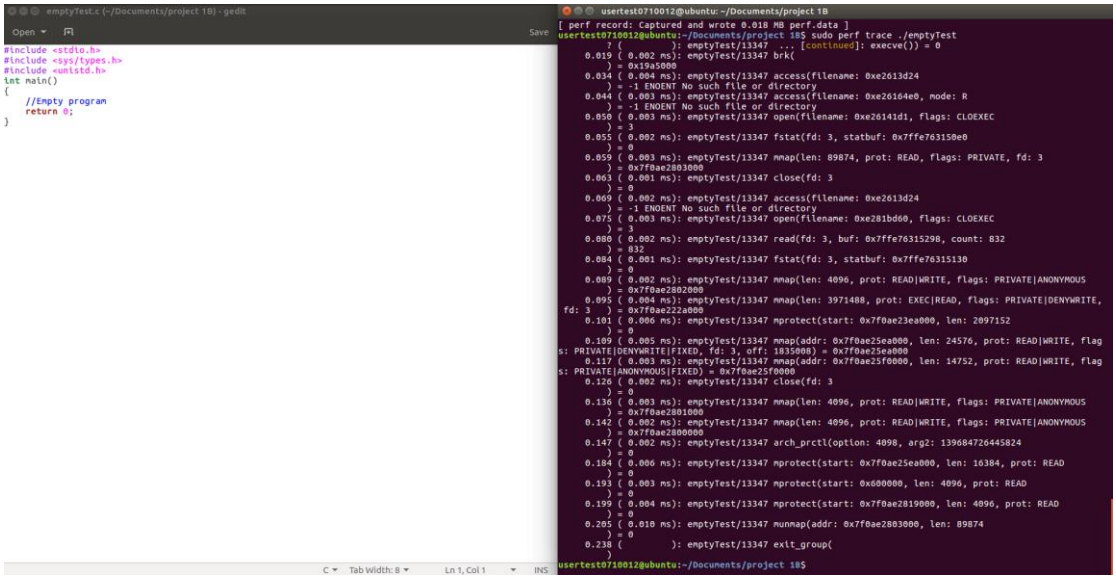
Screenshot 16:

The perf trace function is successfully installed



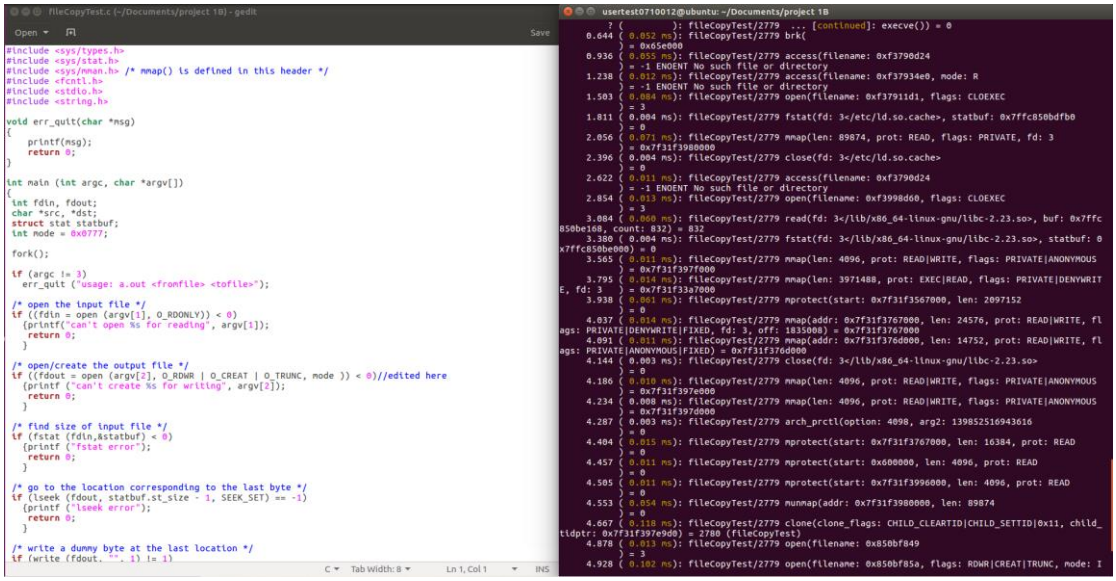
Screenshot 17:

The program code and the trace result of emptyTest.c



Screenshot 18

The program code and the trace result of fileCopyTest.c



Screenshot 19:

We can see that the trace result of fileCopyTest.c has a bulk of text more than emptyTest.c, which should be the actual code that is executed in fileCopyTest.c

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

   Clone: create new(child) process

   Mmap: create a new mapping in the virtual address space of the calling process

   Write: write the context in the buffer into the file that is referred to

   Open: open the file that is referred to

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

As it is unnecessary to create a new process that is totally independent to the origin process, clone can save more resources

The child process created by clone allows them to share parts of its execution context with the calling process

3. Will the functions' execution time be longer if the file is bigger?

Yes, as the data the program take in and print out is different. More data should take more time.

emptyTest.txt

```
? (          ): emptyTest/2604  ... [continued]: execve()) = 0
    0.693 ( 0.098 ms): emptyTest/2604 brk(                                                                  )
    1.176 ( 0.141 ms): emptyTest/2604 access(filename: 0x7aa94d24                                           )
    1.564 ( 0.013 ms): emptyTest/2604 access(filename: 0x7aa974e0, mode: R                                  )
    1.799 ( 0.058 ms): emptyTest/2604 open(filename: 0x7aa951d1, flags: CLOEXEC                             )
    2.105 ( 0.004 ms): emptyTest/2604 fstat(fd: 3</etc/ld.so.cache>, statbuf: 0x7ffd0cbbfee0                )
    2.292 ( 0.070 ms): emptyTest/2604 mmap(len: 89874, prot: READ, flags: PRIVATE, fd: 3                    )
    2.400 ( 0.003 ms): emptyTest/2604 close(fd: 3</etc/ld.so.cache>                                         )
    2.505 ( 0.012 ms): emptyTest/2604 access(filename: 0x7aa94d24                                           )
    2.558 ( 0.011 ms): emptyTest/2604 open(filename: 0x7ac9cd60, flags: CLOEXEC                             )
    2.605 ( 0.057 ms): emptyTest/2604 read(fd: 3</lib/x86_64-linux-gnu/libc-2.23.so>, buf: 0x7ffd0cbc0098, co
    2.700 ( 0.003 ms): emptyTest/2604 fstat(fd: 3</lib/x86_64-linux-gnu/libc-2.23.so>, statbuf: 0x7ffd0cbbff3
    2.735 ( 0.010 ms): emptyTest/2604 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS            )
    2.778 ( 0.006 ms): emptyTest/2604 mmap(len: 3971488, prot: EXEC|READ, flags: PRIVATE|DENYWRITE, fd: 3   )
    2.787 ( 0.007 ms): emptyTest/2604 mprotect(start: 0x7fb07a86b000, len: 2097152                          )
    2.795 ( 0.005 ms): emptyTest/2604 mmap(addr: 0x7fb07aa6b000, len: 24576, prot: READ|WRITE, flags: PRIVATE
    2.805 ( 0.003 ms): emptyTest/2604 mmap(addr: 0x7fb07aa71000, len: 14752, prot: READ|WRITE, flags: PRIVATE
    2.815 ( 0.002 ms): emptyTest/2604 close(fd: 3</lib/x86_64-linux-gnu/libc-2.23.so>                       )
    2.825 ( 0.003 ms): emptyTest/2604 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS            )
    2.832 ( 0.002 ms): emptyTest/2604 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS            )
    2.837 ( 0.002 ms): emptyTest/2604 arch_prctl(option: 4098, arg2: 140395950909184                        )
    2.875 ( 0.006 ms): emptyTest/2604 mprotect(start: 0x7fb07aa6b000, len: 16384, prot: READ                )
    2.884 ( 0.004 ms): emptyTest/2604 mprotect(start: 0x600000, len: 4096, prot: READ                       )
    2.891 ( 0.005 ms): emptyTest/2604 mprotect(start: 0x7fb07ac9a000, len: 4096, prot: READ                 )
    2.898 ( 0.010 ms): emptyTest/2604 munmap(addr: 0x7fb07ac84000, len: 89874                               )
    2.932 (          ): emptyTest/2604 exit_group(                                                          )
```

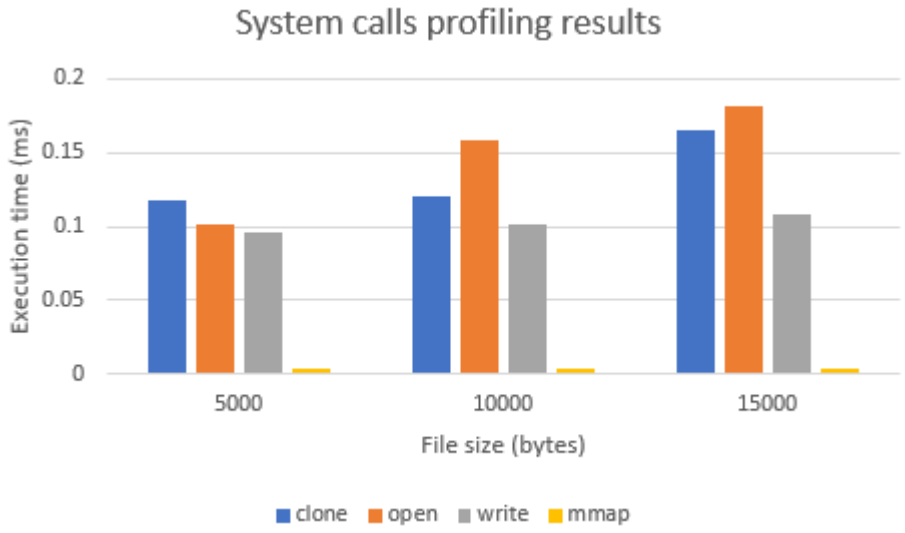
fileCopyTest.txt

```
? (          ): fileCopyTest/2779  ... [continued]: execve()) = 0
    0.644 ( 0.052 ms): fileCopyTest/2779 brk(
    0.936 ( 0.055 ms): fileCopyTest/2779 access(filename: 0xf3790d24
    1.238 ( 0.012 ms): fileCopyTest/2779 access(filename: 0xf37934e0, mode: R
    1.503 ( 0.084 ms): fileCopyTest/2779 open(filename: 0xf37911d1, flags: CLOEXEC
    1.811 ( 0.004 ms): fileCopyTest/2779 fstat(fd: 3</etc/ld.so.cache>, statbuf: 0x7ffc850bdfb0
    2.056 ( 0.071 ms): fileCopyTest/2779 mmap(len: 89874, prot: READ, flags: PRIVATE, fd: 3
    2.396 ( 0.004 ms): fileCopyTest/2779 close(fd: 3</etc/ld.so.cache>
    2.622 ( 0.011 ms): fileCopyTest/2779 access(filename: 0xf3790d24
    2.854 ( 0.013 ms): fileCopyTest/2779 open(filename: 0xf3998d60, flags: CLOEXEC
    3.084 ( 0.060 ms): fileCopyTest/2779 read(fd: 3</lib/x86_64-linux-gnu/libc-2.23.so>, buf: 0x7ffc850be168,
    3.380 ( 0.004 ms): fileCopyTest/2779 fstat(fd: 3</lib/x86_64-linux-gnu/libc-2.23.so>, statbuf: 0x7ffc850b
    3.565 ( 0.011 ms): fileCopyTest/2779 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS
    3.795 ( 0.014 ms): fileCopyTest/2779 mmap(len: 3971488, prot: EXEC|READ, flags: PRIVATE|DENYWRITE, fd: 3
    3.938 ( 0.061 ms): fileCopyTest/2779 mprotect(start: 0x7f31f3567000, len: 2097152
    4.037 ( 0.014 ms): fileCopyTest/2779 mmap(addr: 0x7f31f3767000, len: 24576, prot: READ|WRITE, flags: PRIV
    4.091 ( 0.011 ms): fileCopyTest/2779 mmap(addr: 0x7f31f376d000, len: 14752, prot: READ|WRITE, flags: PRIV
    4.144 ( 0.003 ms): fileCopyTest/2779 close(fd: 3</lib/x86_64-linux-gnu/libc-2.23.so>
    4.186 ( 0.010 ms): fileCopyTest/2779 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS
    4.234 ( 0.008 ms): fileCopyTest/2779 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS
    4.287 ( 0.003 ms): fileCopyTest/2779 arch_prctl(option: 4098, arg2: 139852516943616
    4.404 ( 0.015 ms): fileCopyTest/2779 mprotect(start: 0x7f31f3767000, len: 16384, prot: READ
    4.457 ( 0.011 ms): fileCopyTest/2779 mprotect(start: 0x600000, len: 4096, prot: READ
    4.505 ( 0.011 ms): fileCopyTest/2779 mprotect(start: 0x7f31f3996000, len: 4096, prot: READ
    4.553 ( 0.054 ms): fileCopyTest/2779 munmap(addr: 0x7f31f3980000, len: 89874
    4.667 ( 0.118 ms): fileCopyTest/2779 clone(clone_flags: CHILD_CLEARTID|CHILD_SETTID|0x11, child_tidptr: 0:
    4.878 ( 0.013 ms): fileCopyTest/2779 open(filename: 0x850bf849
    4.928 ( 0.102 ms): fileCopyTest/2779 open(filename: 0x850bf85a, flags: RDWR|CREAT|TRUNC, mode: IRUGO|ISGI
    5.073 ( 0.003 ms): fileCopyTest/2779 fstat(fd: 3</home/usertest0710012/Documents/project 1B/originalFile.
    5.118 ( 0.003 ms): fileCopyTest/2779 lseek(fd: 4</home/usertest0710012/Documents/project 1B/copiedFile.tx
    5.212 ( 0.141 ms): fileCopyTest/2779 write(fd: 4</home/usertest0710012/Documents/project 1B/copiedFile.tx
    5.397 ( 0.011 ms): fileCopyTest/2779 mmap(len: 9947, prot: READ, flags: SHARED, fd: 3
    5.444 ( 0.005 ms): fileCopyTest/2779 mmap(len: 9947, prot: READ|WRITE, flags: SHARED, fd: 4
    5.483 (          ): fileCopyTest/2779 exit_group(
```

Screenshot 20:

-from fastest to slowest- :mmap, write, clone, open

Though open takes less time when executing the 5000 bytes file, the average time is longer than clone



Screenshot 21:

a.read the perf.data file and display the context

b.The percentage under the overhead column shows the overall samples collected in the corresponding function and its children, therefore, the fileCppyTest function and its chilren, which is dynamically linked and have the shared library named “mptbase” contain 100% of the overall samples, while the fileCopyTest function(the one that come from the kernel) and its children contains 0% of the overall samples.

Actually can’y really understand the chart, as it looks totally different with the one I find online, no matter it is the parents function display or the percentage. Moreover, I think my perf.data may have some problem as the percentage is shown in red font, which is not normal, however, no matter how many time I tried to rerun the program or “record” a new perf.data, the result is always the same.

usertest0710012@ubuntu: ~/Documents/project 1B

```
# To display the perf.data header info, please use --header/--header-only opti
#
#
# Total Lost Samples: 0
#
# Samples: 1  of event 'cpu-clock'
# Event count (approx.): 250000
#
# Children      Self  Command       Shared Object      Symbol
# ........  ........  ............  .................  .......................
#
   100.00%   100.00%  fileCopyTest  [mptbase]          [k] mpt_put_msg_frame
            |
            ---entry_SYSCALL_64_after_hwframe
               do_syscall_64
               0xffffffff8109a6d8
               do_group_exit
               do_exit
               task_work_run
               ____fput
               __fput
               ext4_release_file
               ext4_alloc_da_blocks
               filemap_flush
               __filemap_fdatawrite_range
               do_writepages
               ext4_writepages
               blk_finish_plug
               blk_flush_plug_list
               queue_unplugged
               __blk_run_queue
               scsi_request_fn
               scsi_dispatch_cmd
               mptspi_qcmd
               mptscsih_qcmd
               mpt_put_msg_frame

   100.00%     0.00%  fileCopyTest  [kernel.kallsyms]  [k] entry_SYSCALL_64_af
            |
            ---entry_SYSCALL_64_after_hwframe
               do_syscall_64
               0xffffffff8109a6d8
               do_group_exit
               do_exit
               task_work_run
               ____fput
               __fput
               ext4_release_file
               ext4_alloc_da_blocks
               filemap_flush
               __filemap_fdatawrite_range
               do_writepages
               ext4_writepages
usertest0710012@ubuntu:~/Documents/project 1B$
```

The example I found online:

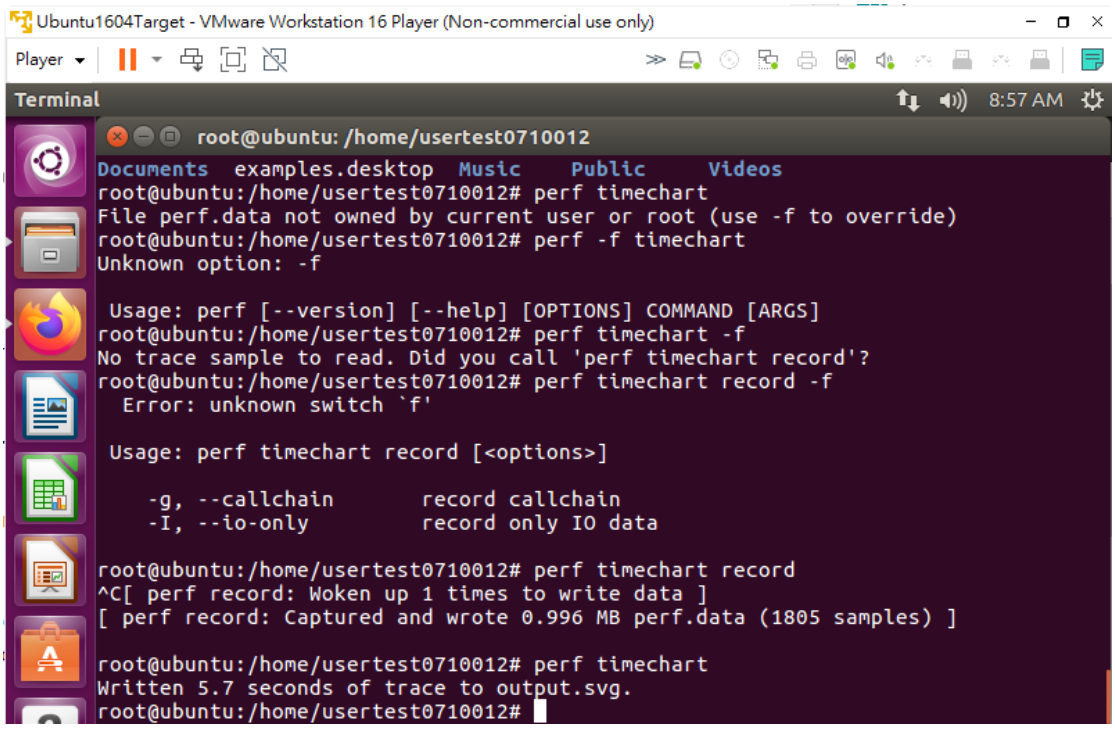
```
Children      Self  Symbol
........  ........  ....................
 100.00%     0.00%  __libc_start_main
          |
          --- __libc_start_main

 100.00%     0.00%  main
          |
          --- main
              __libc_start_main

 100.00%    40.00%  bar
          |
          --- bar
              main
              __libc_start_main

  60.00%    60.00%  foo
          |
          --- foo
              bar
              main
              __libc_start_main
```

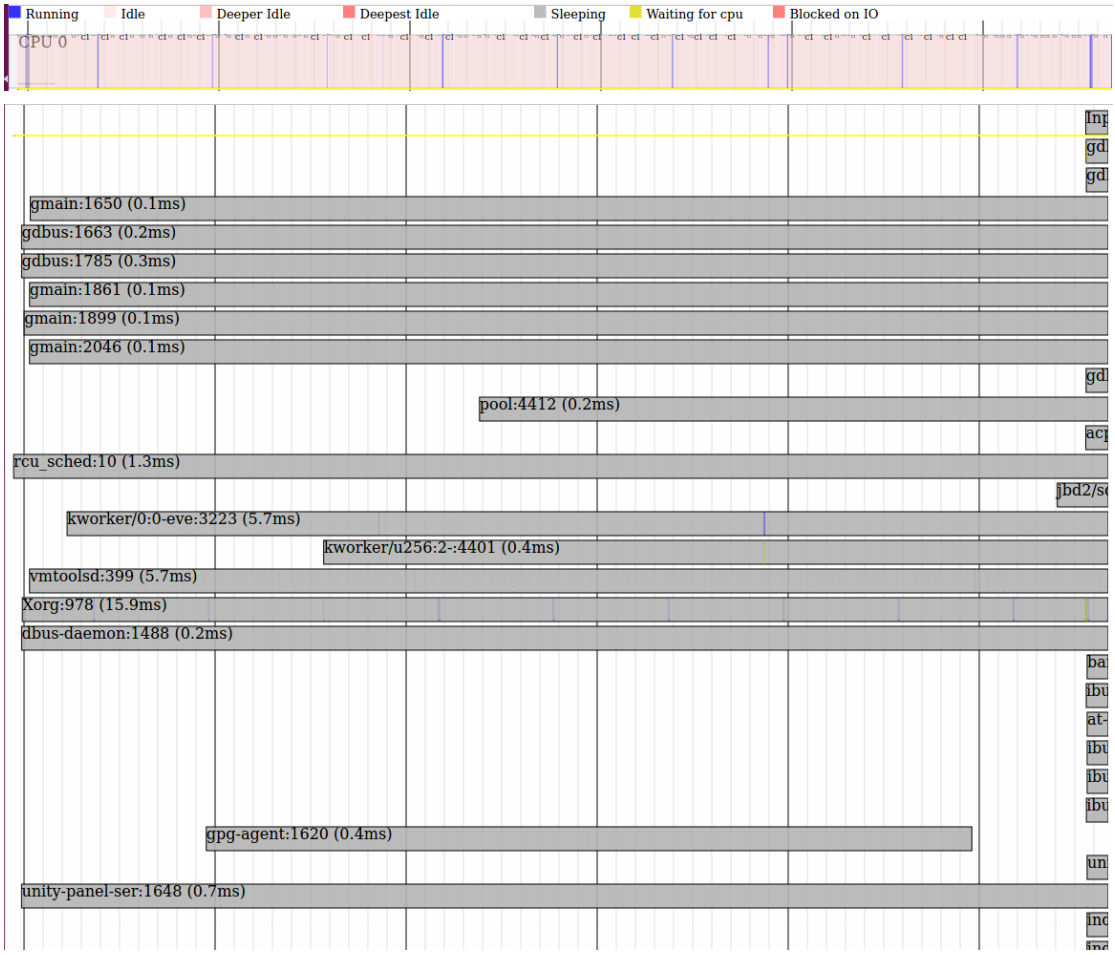
Screenshot 22:

I choose “perf timechart” as my example, the function is simply record(perf.data) the time table (system behavior) of the program and print

(Tool to visualize total system behavior during a workload)

The function will output a file called output.svg(graphic file), which is shown in below, we can see the execution of the function in the graph, showing the relative time and feature of parallel among the functions.




Running
Idle
Deeper Idle
Deepest Idle
Sleeping
Waiting for cpu
Blocked on IO
CPU 0
gmain:1650 (0.1ms)
gdbus:1663 (0.2ms)
gdbus:1785 (0.3ms)
gmain:1861 (0.1ms)
gmain:1899 (0.1ms)
gmain:2046 (0.1ms)
pool:4412 (0.2ms)
rcu_sched:10 (1.3ms)
kworker/0:0-eve:3223 (5.7ms)
kworker/u256:2-:4401 (0.4ms)
vmtoolsd:399 (5.7ms)
Xorg:978 (15.9ms)
dbus-daemon:1488 (0.2ms)
gpg-agent:1620 (0.4ms)
unity-panel-ser:1648 (0.7ms)