Introduction to Operating Systems

Project 1B: Linux kernel debugging (KGDB) and profiling (PERF)

Deadline:

2020-10-30 (Fri) 23:59:59

Q&A:

If you have any questions, please post it on the E3 discussion board, and it will be answered in two days.

Deliverables:

1. **Demo video** (5-7 minutes – upload to YouTube – add link in the first page of the report).
2. **Report** (pdf file with file name of the form **OS_Project1B_StudentID.pdf**)
   - Screenshots + one explanation paragraph per screenshot.
   - Answers to questions below.
3. In the demo video and report, for each screenshot, explain:
   a. What has been done, and
   b. The reasoning behind the steps.

Objective: The objective of this project is to help the student to get familiar kernel debugging and profiling. The student will learn the definitions of debugging and profiling, how to prepare its working environment, how to trigger break points and how to profile and interpret the profiling results of kernel functions.

Scope: Understand the concept of kernel debug, kernel profile, and to learn how to prepare the working environment and how to interpret the profiling results.

Table of contents:

Questions to be answered in the report:

1. What is a kernel function? What is a system call?
2. What is KASLR? What is it for?
3. What are GDB's non-stop and all-stop modes?
4. Explain what the command **echo g > /proc/sysrq-trigger** does.
5. Questions of section 2.2 (page 23).

   Also, in your report, remember to include:
6. **Do it yourself** exercise of Section 1 (page 12).
7. **Do it yourself** exercise of Section 2 (page 24).

Useful links:

- Project 1A tutorial:
  https://www.youtube.com/playlist?list=PL4l5tG9L6WbQrIvxdfg9KXt0OLxfrwdlu
- How to disable KASLR:
  https://stackoverflow.com/questions/49360506/in-kgdb-i-cannot-set-the-breakpoint
  https://askubuntu.com/questions/964540/gdb-qemu-cant-put-break-point-on-kernel-function-kernel-4-10-0-35
- Linux system calls table:
  https://filippo.io/linux-syscall-table/
- Linux kernel 4.19.148 source code:
  https://elixir.bootlin.com/linux/v4.19.148/source
- Examples on KGDB:
  https://developer.ridgerun.com/wiki/index.php/How_to_use_kgdb
- How to build the perf tool:
  https://blog.csdn.net/tang05505622334/article/details/103057179
- Notepad ++ compare plugin:
  http://www.technicaloverload.com/compare-two-files-using-notepad/

If you have any questions, please remember to search online before posting them in the forum.

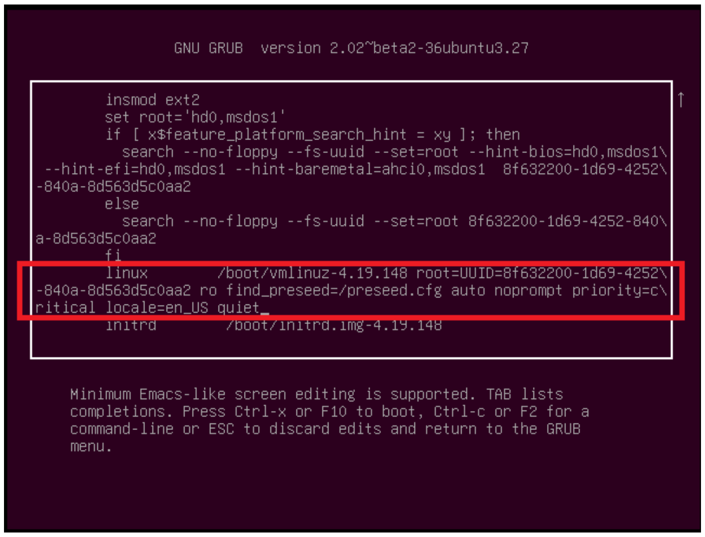
SECTION 1:
DEBUGGING KERNEL FUNCTIONS

Section 1: Debugging kernel functions

In this section, we will debug some basic kernel functions.

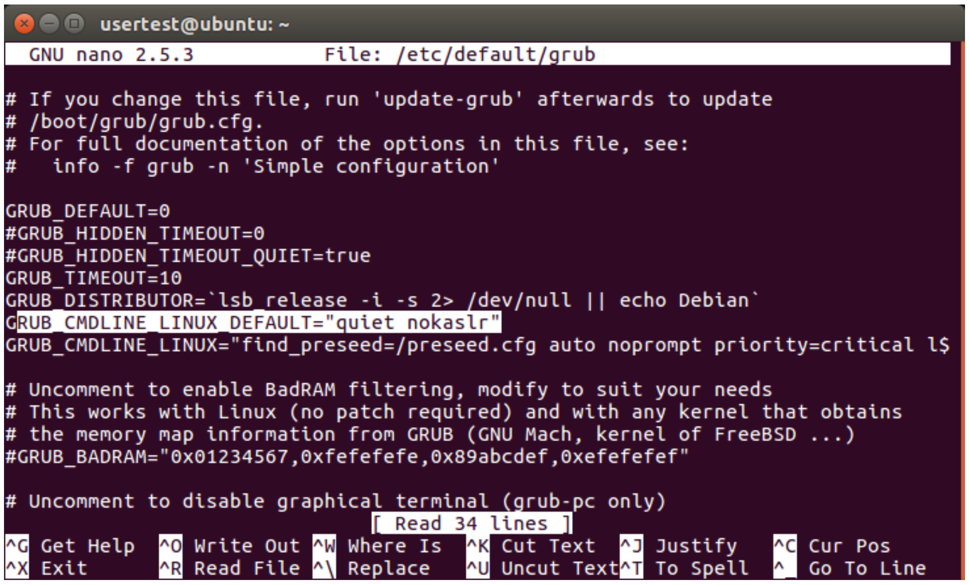
NOTE: You need to finish project 1A in order to do this one. If you haven't finished it, please refer to these tutorial videos: https://www.youtube.com/playlist?list=PL4l5tG9L6WbQrIvxdfg9KXt0OLxfrwdlu

1. Turn on the Target virtual machine and bypass the kgdb commands in the grub entry. (Refer to Section 4.0.2 in project 1A)



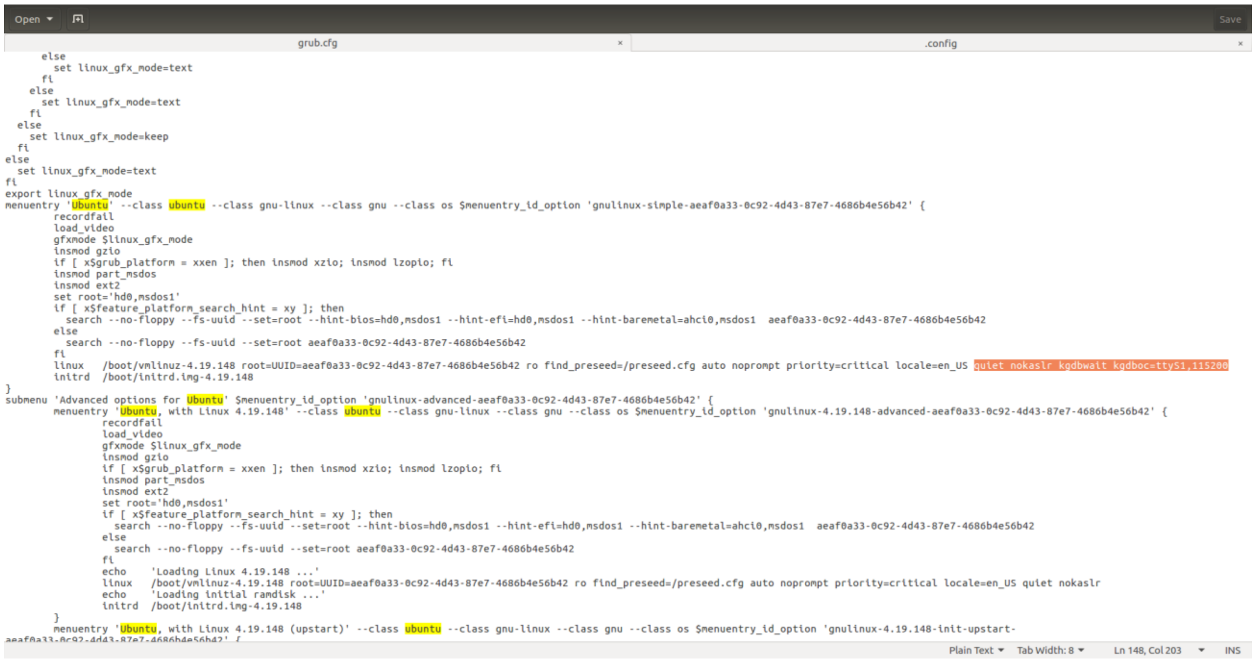
```
                 GNU GRUB  version 2.02~beta2-36ubuntu3.27

         insmod ext2
         set root='hd0,msdos1'
         if [ x$feature_platform_search_hint = xy ]; then
           search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1\
 --hint-efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1  8f632200-1d69-4252\
-840a-8d563d5c0aa2
         else
           search --no-floppy --fs-uuid --set=root 8f632200-1d69-4252-840\
a-8d563d5c0aa2
         fi
         linux        /boot/vmlinuz-4.19.148 root=UUID=8f632200-1d69-4252\
-840a-8d563d5c0aa2 ro find_preseed=/preseed.cfg auto noprompt priority=c\
ritical locale=en_US quiet_
         initrd        /boot/initrd.img-4.19.148

     Minimum Emacs-like screen editing is supported. TAB lists
     completions. Press Ctrl-x or F10 to boot, Ctrl-c or F2 for a
     command-line or ESC to discard edits and return to the GRUB
     menu.
```

2. We need to disable KASLR from the grub.
   https://stackoverflow.com/questions/49360506/in-kgdb-i-cannot-set-the-breakpoint
   Run the command
   $sudo nano /etc/default/grub
   And in the GRUB_CMDLINE_LINUX_DEFAULT line, add "nokaslr".



```
usertest@ubuntu: ~
  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 -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet nokaslr"
GRUB_CMDLINE_LINUX="find_preseed=/preseed.cfg auto noprompt priority=critical l$

# 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
```

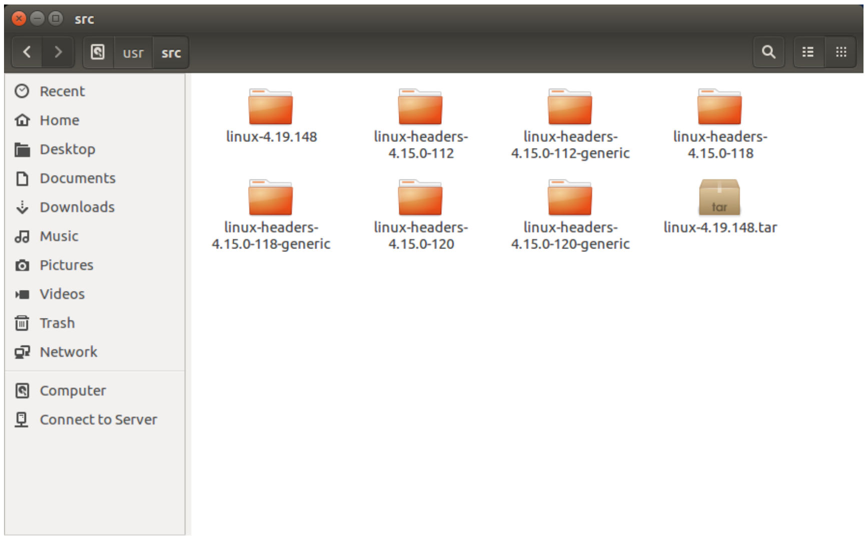
3. Update the grub with the command
   $sudo update-grub
4. In the /boot/grub/grub.cfg file, re-add the commands **kgdbwait kgdboc=ttyS1,115200** (You need to do this step after every time you run the update-grub command).



```
      else
        set linux_gfx_mode=text
      fi
    else
      set linux_gfx_mode=text
    fi
  else
    set linux_gfx_mode=keep
  fi
else
  set linux_gfx_mode=text
fi
export linux_gfx_mode
menuentry 'Ubuntu' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id_option 'gnulinux-simple-aeaf0a33-0c92-4d43-87e7-4686b4e56b42' {
        recordfail
        load_video
        gfxmode $linux_gfx_mode
        insmod gzio
        if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
        insmod part_msdos
        insmod ext2
        set root='hd0,msdos1'
        if [ x$feature_platform_search_hint = xy ]; then
          search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1 --hint-efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1  aeaf0a33-0c92-4d43-87e7-4686b4e56b42
        else
          search --no-floppy --fs-uuid --set=root aeaf0a33-0c92-4d43-87e7-4686b4e56b42
        fi
        linux   /boot/vmlinuz-4.19.148 root=UUID=aeaf0a33-0c92-4d43-87e7-4686b4e56b42 ro find_preseed=/preseed.cfg auto noprompt priority=critical locale=en_US quiet nokaslr kgdbwait kgdboc=ttyS1,115200
        initrd  /boot/initrd.img-4.19.148
}
submenu 'Advanced options for Ubuntu' $menuentry_id_option 'gnulinux-advanced-aeaf0a33-0c92-4d43-87e7-4686b4e56b42' {
        menuentry 'Ubuntu, with Linux 4.19.148' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id_option 'gnulinux-4.19.148-advanced-aeaf0a33-0c92-4d43-87e7-4686b4e56b42' {
                recordfail
                load_video
                gfxmode $linux_gfx_mode
                insmod gzio
                if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
                insmod part_msdos
                insmod ext2
                set root='hd0,msdos1'
                if [ x$feature_platform_search_hint = xy ]; then
                  search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1 --hint-efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1  aeaf0a33-0c92-4d43-87e7-4686b4e56b42
                else
                  search --no-floppy --fs-uuid --set=root aeaf0a33-0c92-4d43-87e7-4686b4e56b42
                fi
                echo    'Loading Linux 4.19.148 ...'
                linux   /boot/vmlinuz-4.19.148 root=UUID=aeaf0a33-0c92-4d43-87e7-4686b4e56b42 ro find_preseed=/preseed.cfg auto noprompt priority=critical locale=en_US quiet nokaslr
                echo    'Loading initial ramdisk ...'
                initrd  /boot/initrd.img-4.19.148
        }
        menuentry 'Ubuntu, with Linux 4.19.148 (upstart)' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id_option 'gnulinux-4.19.148-init-upstart-
aeaf0a33-0c92-4d43-87e7-4686b4e56b42' {
```

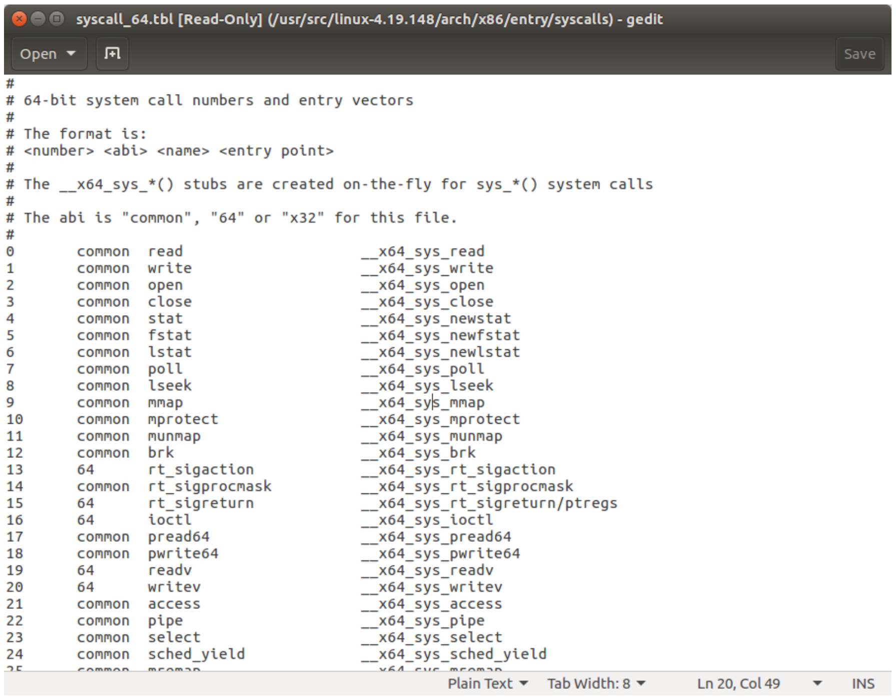
[Screenshot # 1 and #2: Create screenshots showing how you update the grub and the grub.cfg file.]

5. (This step is optional, but it could be useful for later steps).
   In the Host machine, download again **the same kernel source code that you have in the target machine**. Download the tar.xz, decompress it and untar it and save it in /usr/src/.



   (An online version is here https://elixir.bootlin.com/linux/v4.19.148/source, but beware that it may not be accurate. It is advised to re-download the kernel source code).

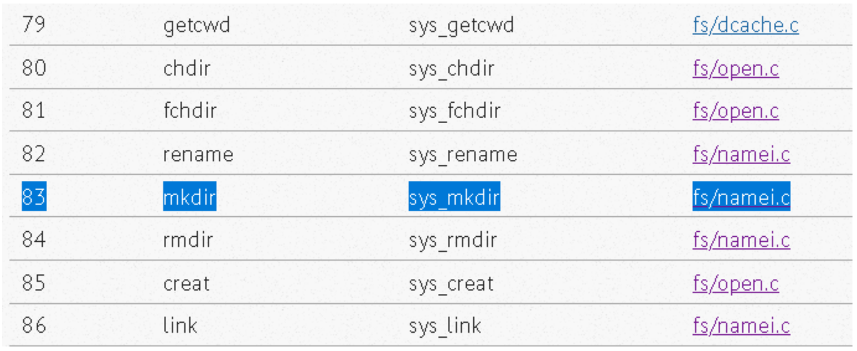
6. Restart the target machine. It should reach the "*Waiting for connection from remote gdb*" message.
7. KGDB will debug **KERNEL FUNCTIONS**. In the kernel source code folder, check for the **/arch/x86/entry/syscalls/syscall_64.tbl** file. This has the list of the kernel functions with its system call number.



syscall_64.tbl [Read-Only] (/usr/src/linux-4.19.148/arch/x86/entry/syscalls) - gedit
Open
Save
Plain Text
Tab Width: 8
Ln 20, Col 49
INS

```
#
# 64-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The __x64_sys_*() stubs are created on-the-fly for sys_*() system calls
#
# The abi is "common", "64" or "x32" for this file.
#
0       common  read                    __x64_sys_read
1       common  write                   __x64_sys_write
2       common  open                    __x64_sys_open
3       common  close                   __x64_sys_close
4       common  stat                    __x64_sys_newstat
5       common  fstat                   __x64_sys_newfstat
6       common  lstat                   __x64_sys_newlstat
7       common  poll                    __x64_sys_poll
8       common  lseek                   __x64_sys_lseek
9       common  mmap                    __x64_sys_mmap
10      common  mprotect                __x64_sys_mprotect
11      common  munmap                  __x64_sys_munmap
12      common  brk                     __x64_sys_brk
13      64      rt_sigaction            __x64_sys_rt_sigaction
14      common  rt_sigprocmask          __x64_sys_rt_sigprocmask
15      64      rt_sigreturn            __x64_sys_rt_sigreturn/ptregs
16      64      ioctl                   __x64_sys_ioctl
17      common  pread64                 __x64_sys_pread64
18      common  pwrite64                __x64_sys_pwrite64
19      64      readv                   __x64_sys_readv
20      64      writev                  __x64_sys_writev
21      common  access                  __x64_sys_access
22      common  pipe                    __x64_sys_pipe
23      common  select                  __x64_sys_select
24      common  sched_yield             __x64_sys_sched_yield
```

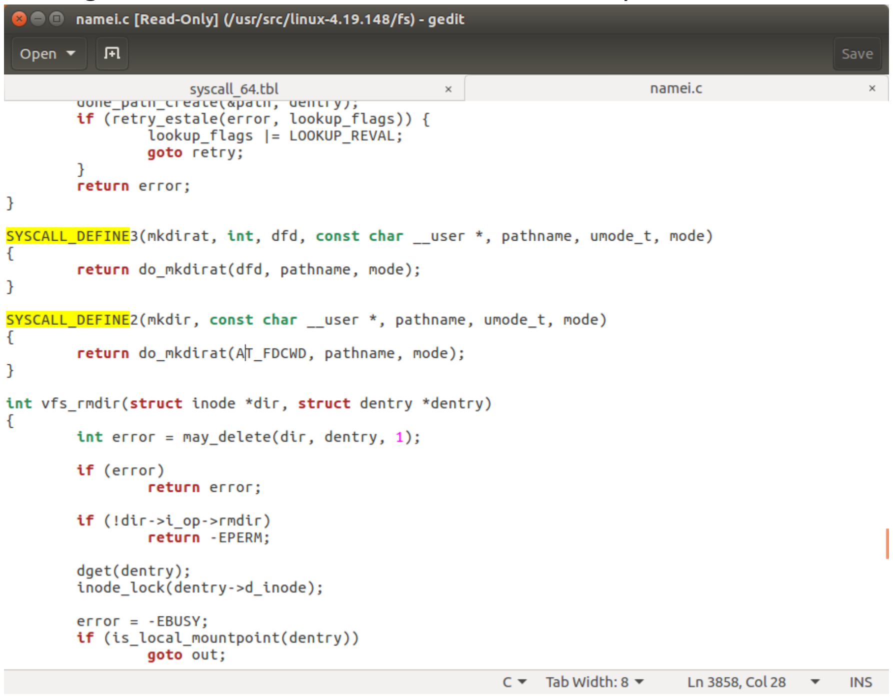
8. We will create a break point for the **mkdir** function, which will be triggered when we create a new folder.
   Look online for at least 3 tables showing where each kernel function is implemented. An example is https://filippo.io/linux-syscall-table/, but keep in mind that some functions inside the syscall_64.tbl file might be missing in the online tables. (Check for Linux system calls only, not other OSs, and the tables should be consistent between them).
   In the table, we find that the mkdir function is implemented in the **fs/namei.c** file. (These paths are inside the kernel folder).



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

[Screenshot # 3, #4, #5 and #6: Create a screenshot showing your syscall_64.tbl file (displaying your student ID) and a screenshot showing each syscall table you found online.]

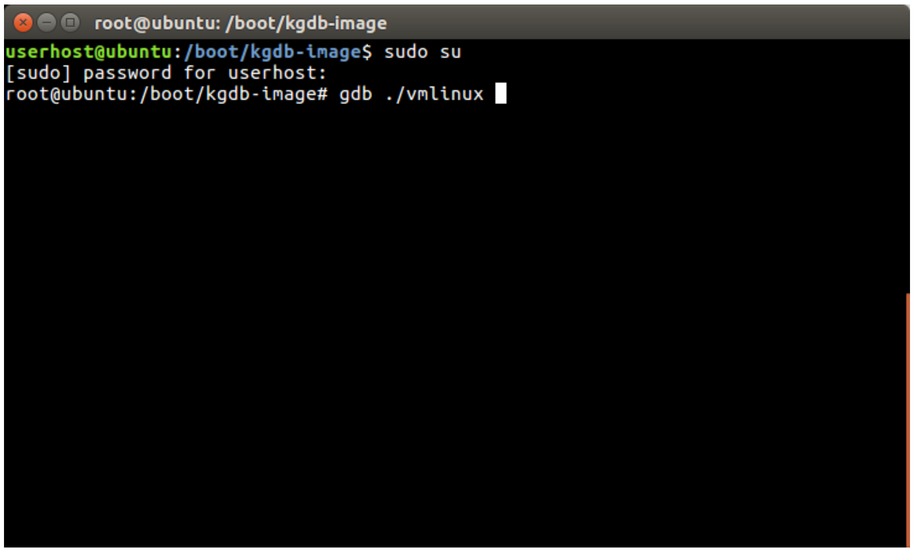
In the namei.c file, check for an entry of the form **SYSCALL_DEFINE[N](mkdir,…)**, where [N] is an integer number. In this case, we find the entry as below



```
        done_path_create(&path, dentry);
        if (retry_estale(error, lookup_flags)) {
                lookup_flags |= LOOKUP_REVAL;
                goto retry;
        }
        return error;
}

SYSCALL_DEFINE3(mkdirat, int, dfd, const char __user *, pathname, umode_t, mode)
{
        return do_mkdirat(dfd, pathname, mode);
}

SYSCALL_DEFINE2(mkdir, const char __user *, pathname, umode_t, mode)
{
        return do_mkdirat(AT_FDCWD, pathname, mode);
}

int vfs_rmdir(struct inode *dir, struct dentry *dentry)
{
        int error = may_delete(dir, dentry, 1);

        if (error)
                return error;

        if (!dir->i_op->rmdir)
                return -EPERM;

        dget(dentry);
        inode_lock(dentry->d_inode);

        error = -EBUSY;
        if (is_local_mountpoint(dentry))
                goto out;
```

From the last picture we see that the mkdir function calls the **do_mkdirat** kernel function, and has the pathname as one of the parameters. We will create a breakpoint in this function.

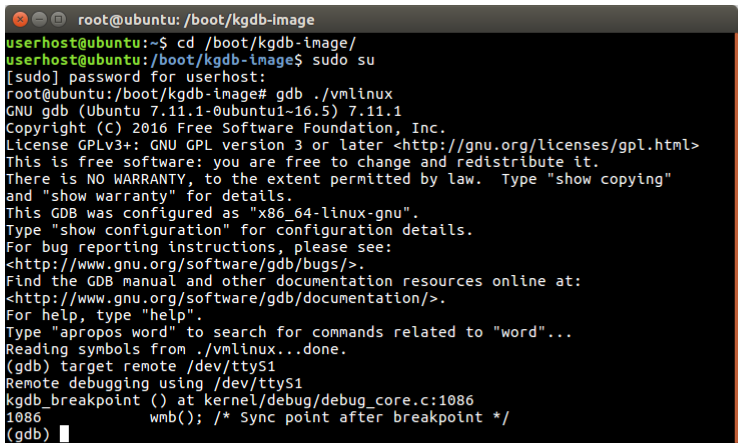
9. In the host machine, connect KGDB to the target machine (project 1A – Section 4.1) with the commands
   cd /boot/kgdb-image
   sudo su
   # gdb ./vmlinux



```
userhost@ubuntu:/boot/kgdb-image$ sudo su
[sudo] password for userhost:
root@ubuntu:/boot/kgdb-image# gdb ./vmlinux
```

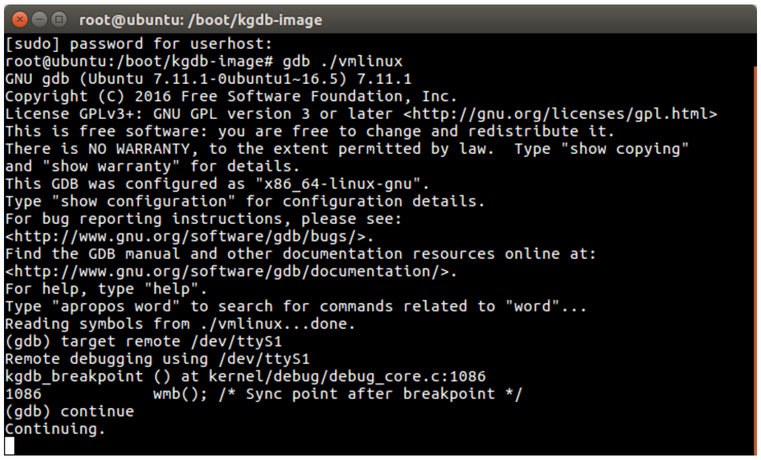
And connect to the target by using

(gdb) target remote /dev/ttyS1



```
root@ubuntu: /boot/kgdb-image
userhost@ubuntu:~$ cd /boot/kgdb-image/
userhost@ubuntu:/boot/kgdb-image$ sudo su
[sudo] password for userhost:
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) target remote /dev/ttyS1
Remote debugging using /dev/ttyS1
kgdb_breakpoint () at kernel/debug/debug_core.c:1086
1086            wmb(); /* Sync point after breakpoint */
(gdb) 
```

Type **continue**, so the target machine boots normally.

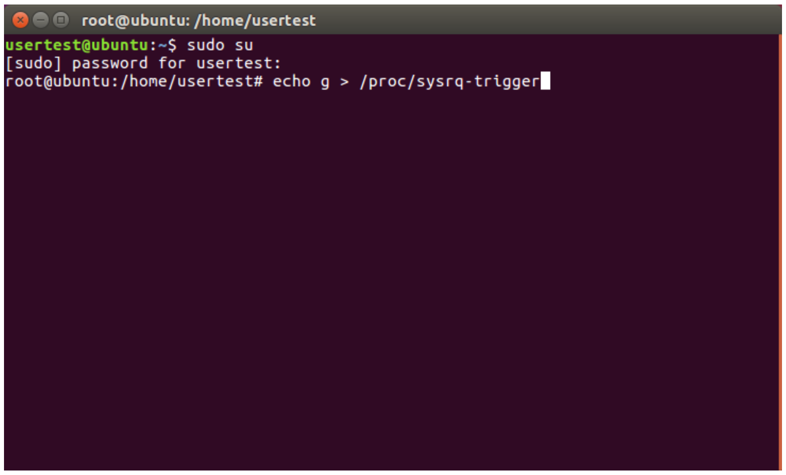


```
root@ubuntu: /boot/kgdb-image
[sudo] password for userhost:
root@ubuntu:/boot/kgdb-image# gdb ./vmlinux
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: 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) target remote /dev/ttyS1
Remote debugging using /dev/ttyS1
kgdb_breakpoint () at kernel/debug/debug_core.c:1086
1086            wmb(); /* Sync point after breakpoint */
(gdb) continue
Continuing.
```

[Screenshot # 7: Create a screenshot (showing your student ID) of this step.]

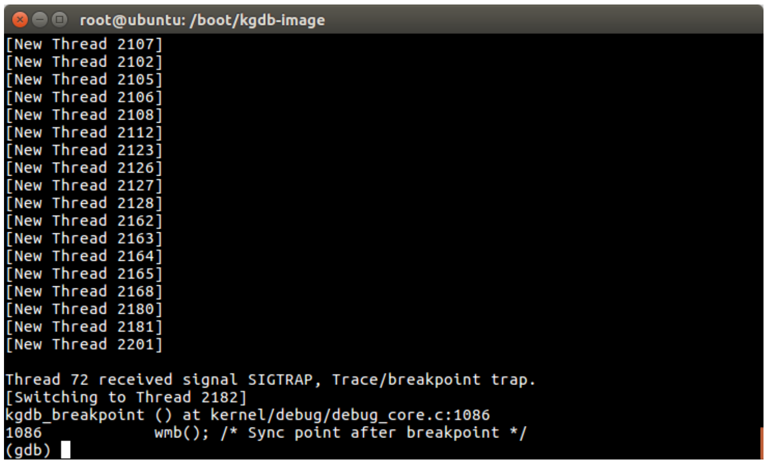
10. In the target machine, we need to send a signal to the host to re-take control in GDB. In a terminal (as root), run the command

    # echo g > /proc/sysrq-trigger



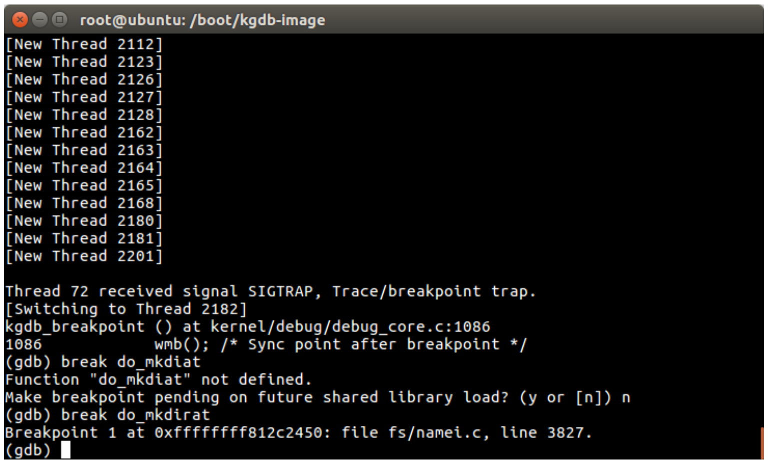
```
root@ubuntu: /home/usertest
usertest@ubuntu:~$ sudo su
[sudo] password for usertest:
root@ubuntu:/home/usertest# echo g > /proc/sysrq-trigger
```

The Target machine will freeze and in the host machine, you will get access again to GDB.



```
root@ubuntu: /boot/kgdb-image
[New Thread 2107]
[New Thread 2102]
[New Thread 2105]
[New Thread 2106]
[New Thread 2108]
[New Thread 2112]
[New Thread 2123]
[New Thread 2126]
[New Thread 2127]
[New Thread 2128]
[New Thread 2162]
[New Thread 2163]
[New Thread 2164]
[New Thread 2165]
[New Thread 2168]
[New Thread 2180]
[New Thread 2181]
[New Thread 2201]

Thread 72 received signal SIGTRAP, Trace/breakpoint trap.
[Switching to Thread 2182]
kgdb_breakpoint () at kernel/debug/debug_core.c:1086
1086            wmb(); /* Sync point after breakpoint */
(gdb) 
```

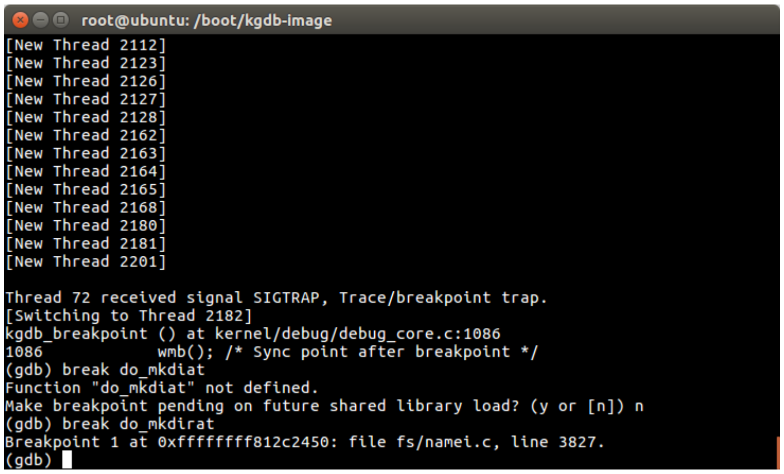
11. We create a breakpoint with the command
    (gdb) break do_mkdirat



```
root@ubuntu: /boot/kgdb-image
[New Thread 2112]
[New Thread 2123]
[New Thread 2126]
[New Thread 2127]
[New Thread 2128]
[New Thread 2162]
[New Thread 2163]
[New Thread 2164]
[New Thread 2165]
[New Thread 2168]
[New Thread 2180]
[New Thread 2181]
[New Thread 2201]

Thread 72 received signal SIGTRAP, Trace/breakpoint trap.
[Switching to Thread 2182]
kgdb_breakpoint () at kernel/debug/debug_core.c:1086
1086            wmb(); /* Sync point after breakpoint */
(gdb) break do_mkdiat
Function "do_mkdiat" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) break do_mkdirat
Breakpoint 1 at 0xffffffff812c2450: file fs/namei.c, line 3827.
(gdb) 
```

It will create a breakpoint and give to you its ID (in this case is 1).

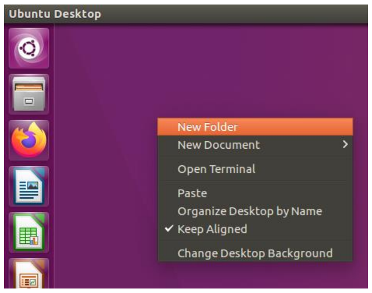
12. Type **(gdb) continue**, and the target machine will unfreeze.



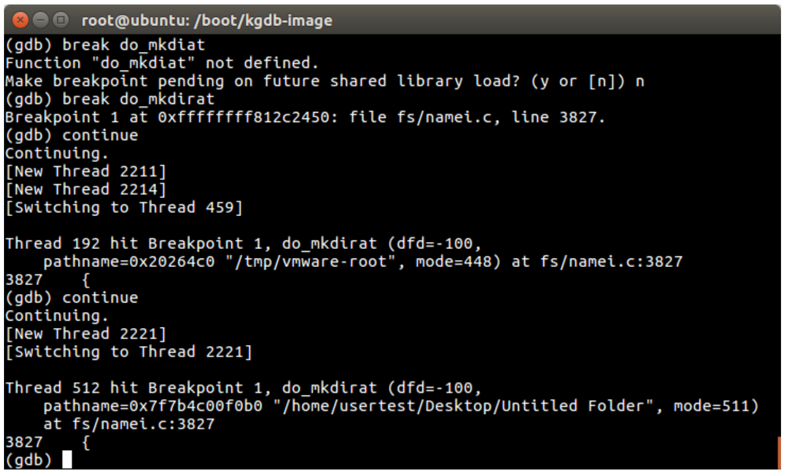
```
root@ubuntu: /boot/kgdb-image
[New Thread 2112]
[New Thread 2123]
[New Thread 2126]
[New Thread 2127]
[New Thread 2128]
[New Thread 2162]
[New Thread 2163]
[New Thread 2164]
[New Thread 2165]
[New Thread 2168]
[New Thread 2180]
[New Thread 2181]
[New Thread 2201]

Thread 72 received signal SIGTRAP, Trace/breakpoint trap.
[Switching to Thread 2182]
kgdb_breakpoint () at kernel/debug/debug_core.c:1086
1086            wmb(); /* Sync point after breakpoint */
(gdb) break do_mkdiat
Function "do_mkdiat" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) break do_mkdirat
Breakpoint 1 at 0xffffffff812c2450: file fs/namei.c, line 3827.
(gdb) 
```

(When you go back to the target, if it is still frozen, come back to GDB and check if a breakpoint was hit. Type continue until the target machine is responsive again).

13. Now in the Target's desktop, create a directory. This will trigger the break point and the Target machine will freeze again.

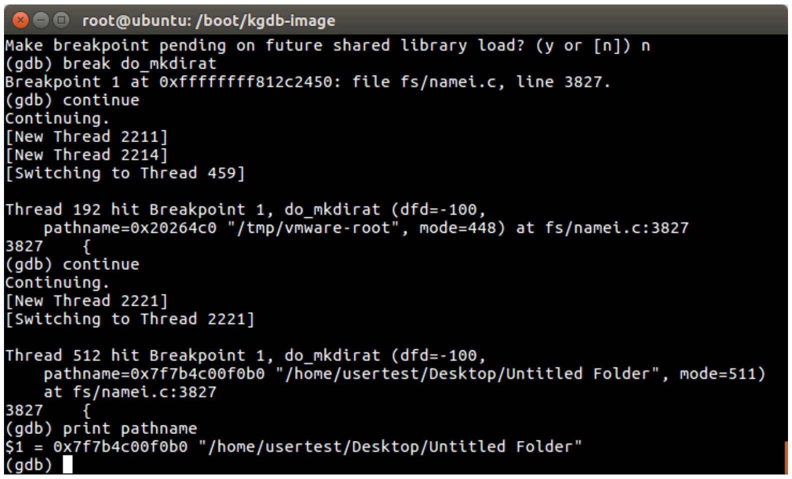


14. In the host machine, we see that a breakpoint was hit.



```
root@ubuntu: /boot/kgdb-image
(gdb) break do_mkdiat
Function "do_mkdiat" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) break do_mkdirat
Breakpoint 1 at 0xffffffff812c2450: file fs/namei.c, line 3827.
(gdb) continue
Continuing.
[New Thread 2211]
[New Thread 2214]
[Switching to Thread 459]

Thread 192 hit Breakpoint 1, do_mkdirat (dfd=-100,
    pathname=0x20264c0 "/tmp/vmware-root", mode=448) at fs/namei.c:3827
3827    {
(gdb) continue
Continuing.
[New Thread 2221]
[Switching to Thread 2221]

Thread 512 hit Breakpoint 1, do_mkdirat (dfd=-100,
    pathname=0x7f7b4c00f0b0 "/home/usertest/Desktop/Untitled Folder", mode=511)
    at fs/namei.c:3827
3827    {
(gdb) 
```

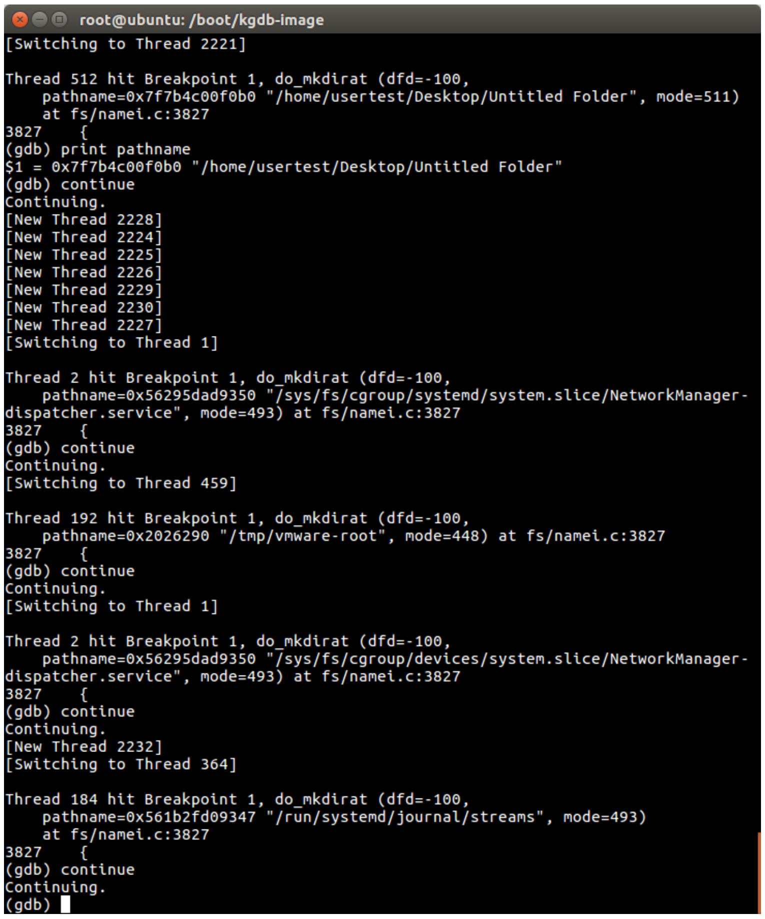
We can check the parameters of this function. By using the command **(gdb) print pathname**, we should get the path where the new folder was created in the target machine.



```
root@ubuntu: /boot/kgdb-image
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) break do_mkdirat
Breakpoint 1 at 0xffffffff812c2450: file fs/namei.c, line 3827.
(gdb) continue
Continuing.
[New Thread 2211]
[New Thread 2214]
[Switching to Thread 459]

Thread 192 hit Breakpoint 1, do_mkdirat (dfd=-100,
    pathname=0x20264c0 "/tmp/vmware-root", mode=448) at fs/namei.c:3827
3827    {
(gdb) continue
Continuing.
[New Thread 2221]
[Switching to Thread 2221]

Thread 512 hit Breakpoint 1, do_mkdirat (dfd=-100,
    pathname=0x7f7b4c00f0b0 "/home/usertest/Desktop/Untitled Folder", mode=511)
    at fs/namei.c:3827
3827    {
(gdb) print pathname
$1 = 0x7f7b4c00f0b0 "/home/usertest/Desktop/Untitled Folder"
(gdb) 
```

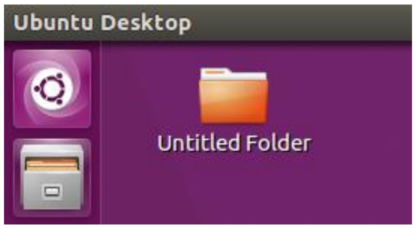
[Screenshot # 8: Create a screenshot showing the pathname parameter value. (It should include your student ID)]

15. Type **(gdb) continue** until the folder is created successfully.



root@ubuntu: /boot/kgdb-image

```
[Switching to Thread 2221]

Thread 512 hit Breakpoint 1, do_mkdirat (dfd=-100,
    pathname=0x7f7b4c00f0b0 "/home/usertest/Desktop/Untitled Folder", mode=511)
    at fs/namei.c:3827
3827    {
(gdb) print pathname
$1 = 0x7f7b4c00f0b0 "/home/usertest/Desktop/Untitled Folder"
(gdb) continue
Continuing.
[New Thread 2228]
[New Thread 2224]
[New Thread 2225]
[New Thread 2226]
[New Thread 2229]
[New Thread 2230]
[New Thread 2227]
[Switching to Thread 1]

Thread 2 hit Breakpoint 1, do_mkdirat (dfd=-100,
    pathname=0x56295dad9350 "/sys/fs/cgroup/systemd/system.slice/NetworkManager-
dispatcher.service", mode=493) at fs/namei.c:3827
3827    {
(gdb) continue
Continuing.
[Switching to Thread 459]

Thread 192 hit Breakpoint 1, do_mkdirat (dfd=-100,
    pathname=0x2026290 "/tmp/vmware-root", mode=448) at fs/namei.c:3827
3827    {
(gdb) continue
Continuing.
[Switching to Thread 1]

Thread 2 hit Breakpoint 1, do_mkdirat (dfd=-100,
    pathname=0x56295dad9350 "/sys/fs/cgroup/devices/system.slice/NetworkManager-
dispatcher.service", mode=493) at fs/namei.c:3827
3827    {
(gdb) continue
Continuing.
[New Thread 2232]
[Switching to Thread 364]

Thread 184 hit Breakpoint 1, do_mkdirat (dfd=-100,
    pathname=0x561b2fd09347 "/run/systemd/journal/streams", mode=493)
    at fs/namei.c:3827
3827    {
(gdb) continue
Continuing.
(gdb) 
```



[Screenshot # 9: Create a screenshot showing both virtual machines, the Host already passed all the continues, and the Target with the new folder created].

A video showing all the steps in this section can be found here:

https://www.youtube.com/watch?v=e-RgDwHOlPk

16. **[Do it yourself]** Select another function from the **syscall_64.tbl** table, look for where it is implemented and create your own scenario to trigger it.

    Required to explain in both the report and in the video:

    a. Which function you selected.
    b. The file that contains it (with its path).
    c. The kernel function that is called.
    d. Which parameters the kernel function has.
    e. Be creative on how to trigger it. If you need to create a c program to trigger it, feel free to do so.

    Requirements:

    a. The Target machine **must be totally on** and responsive before hitting the breakpoint.
    b. You must display the value of at least one of the parameters that the function receives.
    c. You must include at least 6 screenshots showing:
        i. That the machine was on and responsive **[Screenshot # 10]**,
        ii. What is your scenario and how to trigger it **[Screenshot # 11 - #12]**,
        iii. How do you create the break point **[Screenshot # 13]**,
        iv. The host machine hitting the break point, and the value of at least one parameter **[Screenshot # 14]**, and
        v. The Target machine working again, with the action totally finished **[Screenshot # 15]**.

SECTION 2:

PROFILING KERNEL FUNCTIONS

<u>Section 2: Profiling Kernel functions</u>

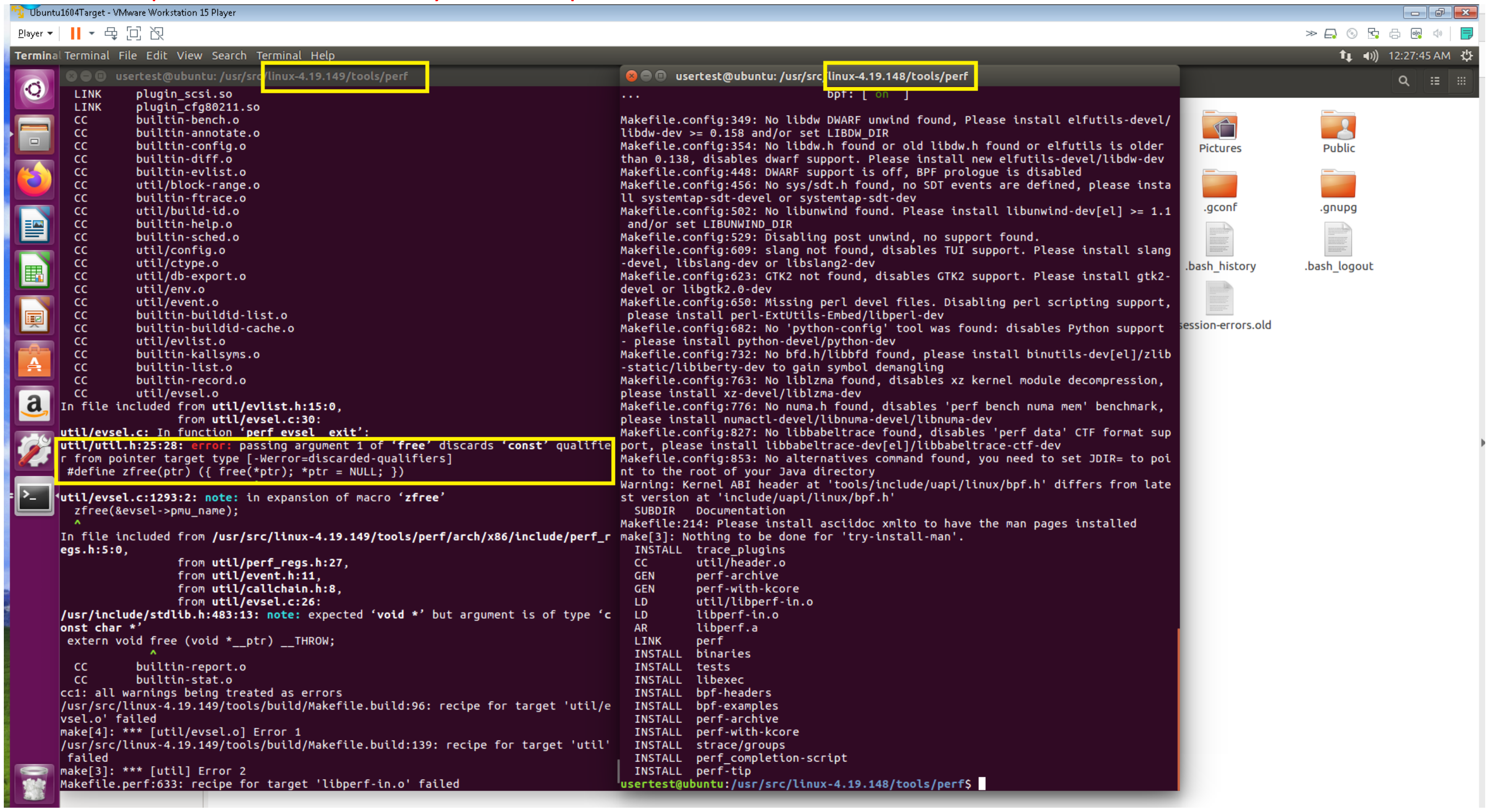
In this section, we will install the required tool to profile the kernel, and proceed to show how to profile and interpret some example functions.

AT THIS POINT, YOU CAN TURN OFF THE HOST MACHINE.

<u>Section 2.0: (IMPORTANT) Perf source code patching</u>

<u>**NOTE:**</u>

A problem with kernels 4.19.149 – 150 – 151 – 152 source code was recently found that prevents to successfully build the perf tool.


Ubuntu1604Target - VMware Workstation 15 Player
Player
Terminal Terminal File Edit View Search Terminal Help
12:27:45 AM
usertest@ubuntu: /usr/src/linux-4.19.149/tools/perf
usertest@ubuntu: /usr/src/linux-4.19.148/tools/perf
Pictures
Public
.gconf
.gnupg
.bash_history
.bash_logout
session-errors.old

```
  LINK     plugin_scsi.so
  LINK     plugin_cfg80211.so
  CC       builtin-bench.o
  CC       builtin-annotate.o
  CC       builtin-config.o
  CC       builtin-diff.o
  CC       builtin-evlist.o
  CC       util/block-range.o
  CC       builtin-ftrace.o
  CC       util/build-id.o
  CC       builtin-help.o
  CC       builtin-sched.o
  CC       util/config.o
  CC       util/ctype.o
  CC       util/db-export.o
  CC       util/env.o
  CC       util/event.o
  CC       builtin-buildid-list.o
  CC       builtin-buildid-cache.o
  CC       util/evlist.o
  CC       builtin-kallsyms.o
  CC       builtin-list.o
  CC       builtin-record.o
  CC       util/evsel.o
In file included from util/evlist.h:15:0,
                 from util/evsel.c:30:
util/evsel.c: In function ‘perf_evsel__exit’:
util/util.h:25:28: error: passing argument 1 of ‘free’ discards ‘const’ qualifie
r from pointer target type [-Werror=discarded-qualifiers]
 #define zfree(ptr) ({ free(*ptr); *ptr = NULL; })

util/evsel.c:1293:2: note: in expansion of macro ‘zfree’
  zfree(&evsel->pmu_name);
  ^
In file included from /usr/src/linux-4.19.149/tools/perf/arch/x86/include/perf_r
egs.h:5:0,
                 from util/perf_regs.h:27,
                 from util/event.h:11,
                 from util/callchain.h:8,
                 from util/evsel.c:26:
/usr/include/stdlib.h:483:13: note: expected ‘void *’ but argument is of type ‘c
onst char *’
 extern void free (void *__ptr) __THROW;
             ^
  CC       builtin-report.o
  CC       builtin-stat.o
cc1: all warnings being treated as errors
/usr/src/linux-4.19.149/tools/build/Makefile.build:96: recipe for target 'util/e
vsel.o' failed
make[4]: *** [util/evsel.o] Error 1
/usr/src/linux-4.19.149/tools/build/Makefile.build:139: recipe for target 'util'
 failed
make[3]: *** [util] Error 2
Makefile.perf:633: recipe for target 'libperf-in.o' failed
```

```
...                          bpf: [ on  ]

Makefile.config:349: No libdw DWARF unwind found, Please install elfutils-devel/
libdw-dev >= 0.158 and/or set LIBDW_DIR
Makefile.config:354: No libdw.h found or old libdw.h found or elfutils is older
than 0.138, disables dwarf support. Please install new elfutils-devel/libdw-dev
Makefile.config:448: DWARF support is off, BPF prologue is disabled
Makefile.config:456: No sys/sdt.h found, no SDT events are defined, please insta
ll systemtap-sdt-devel or systemtap-sdt-dev
Makefile.config:502: No libunwind found. Please install libunwind-dev[el] >= 1.1
 and/or set LIBUNWIND_DIR
Makefile.config:529: Disabling post unwind, no support found.
Makefile.config:609: slang not found, disables TUI support. Please install slang
-devel, libslang-dev or libslang2-dev
Makefile.config:623: GTK2 not found, disables GTK2 support. Please install gtk2-
devel or libgtk2.0-dev
Makefile.config:650: Missing perl devel files. Disabling perl scripting support,
 please install perl-ExtUtils-Embed/libperl-dev
Makefile.config:682: No 'python-config' tool was found: disables Python support
- please install python-devel/python-dev
Makefile.config:732: No bfd.h/libbfd found, please install binutils-dev[el]/zlib
-static/libiberty-dev to gain symbol demangling
Makefile.config:763: No liblzma found, disables xz kernel module decompression,
please install xz-devel/liblzma-dev
Makefile.config:776: No numa.h found, disables 'perf bench numa mem' benchmark,
please install numactl-devel/libnuma-devel/libnuma-dev
Makefile.config:827: No libbabeltrace found, disables 'perf data' CTF format sup
port, please install libbabeltrace-dev[el]/libbabeltrace-ctf-dev
Makefile.config:853: No alternatives command found, you need to set JDIR= to poi
nt to the root of your Java directory
Warning: Kernel ABI header at 'tools/include/uapi/linux/bpf.h' differs from late
st version at 'include/uapi/linux/bpf.h'
  SUBDIR   Documentation
Makefile:214: Please install asciidoc xmlto to have the man pages installed
make[3]: Nothing to be done for 'try-install-man'.
  INSTALL  trace_plugins
  CC       util/header.o
  GEN      perf-archive
  GEN      perf-with-kcore
  LD       util/libperf-in.o
  LD       libperf-in.o
  AR       libperf.a
  LINK     perf
  INSTALL  binaries
  INSTALL  tests
  INSTALL  libexec
  INSTALL  bpf-headers
  INSTALL  bpf-examples
  INSTALL  perf-archive
  INSTALL  perf-with-kcore
  INSTALL  strace/groups
  INSTALL  perf_completion-script
  INSTALL  perf-tip
usertest@ubuntu:/usr/src/linux-4.19.148/tools/perf$
```

<u>**Git comments:**</u>

This problem was discovered by the official developers on Sept. 29-30/2020:

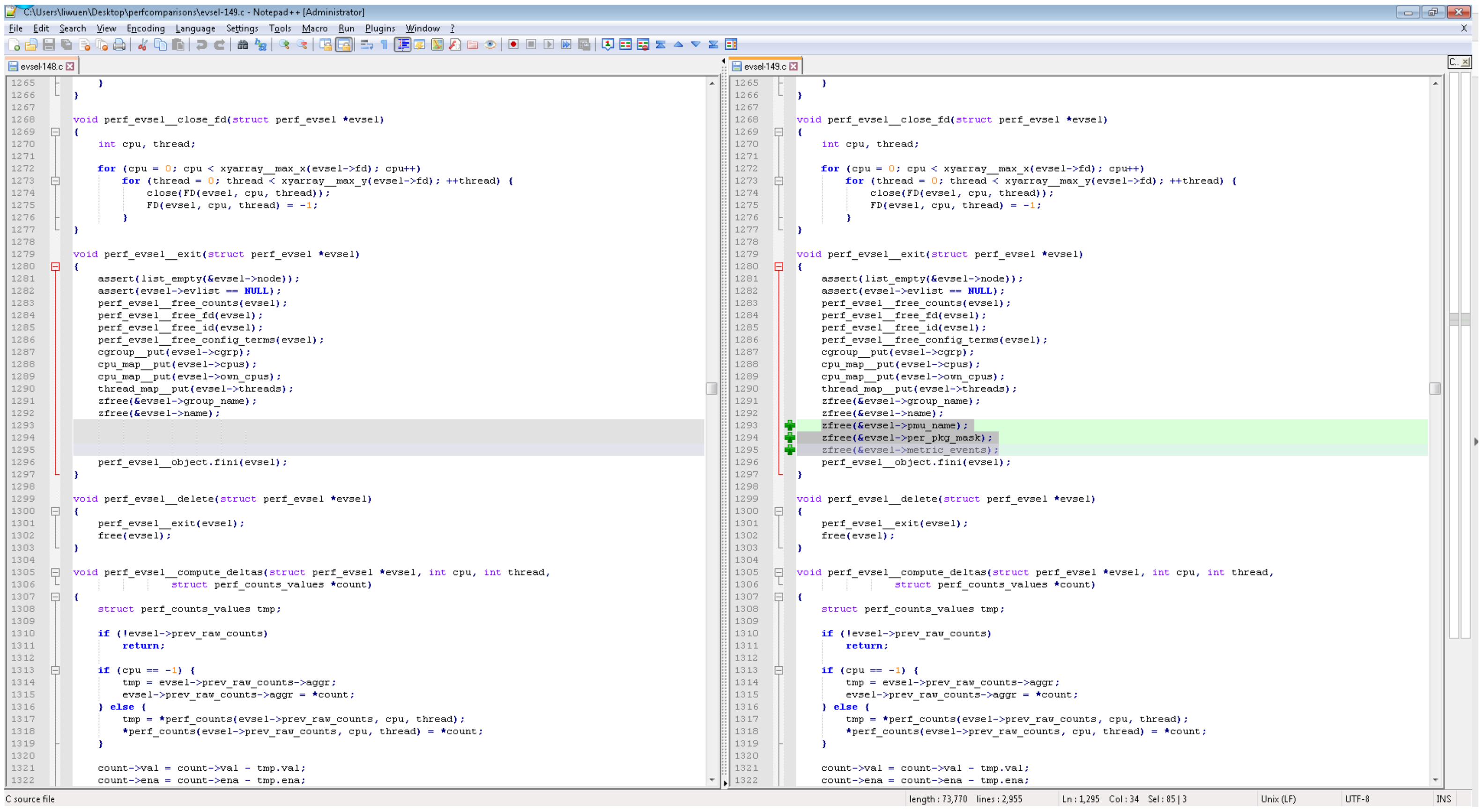
- https://lkml.org/lkml/2020/9/29/2330
- https://lkml.org/lkml/2020/9/30/1024

Solutions:

There are two official solutions for this problem:

1. Use kernel 4.19.148:
   https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.19.148.tar.xz
   This solution requires Section 2 and Section 3 of project 1A.

2. In the affected kernels, go to **linux-4.19.149/tools/perf/util/evsel.c**.
   Locate the following function:
   void perf_evsel__exit(struct perf_evsel *evsel)
   and comment the lines:
   zfree(&evsel->pmu_name);
   zfree(&evsel->per_pkg_mask);
   zfree(&evsel->metric_events);
   (Around lines 1293, 1294 and 1295).


C:\Users\liwuen\Desktop\perfcomparisons\evsel-149.c - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
evsel-148.c
evsel-149.c

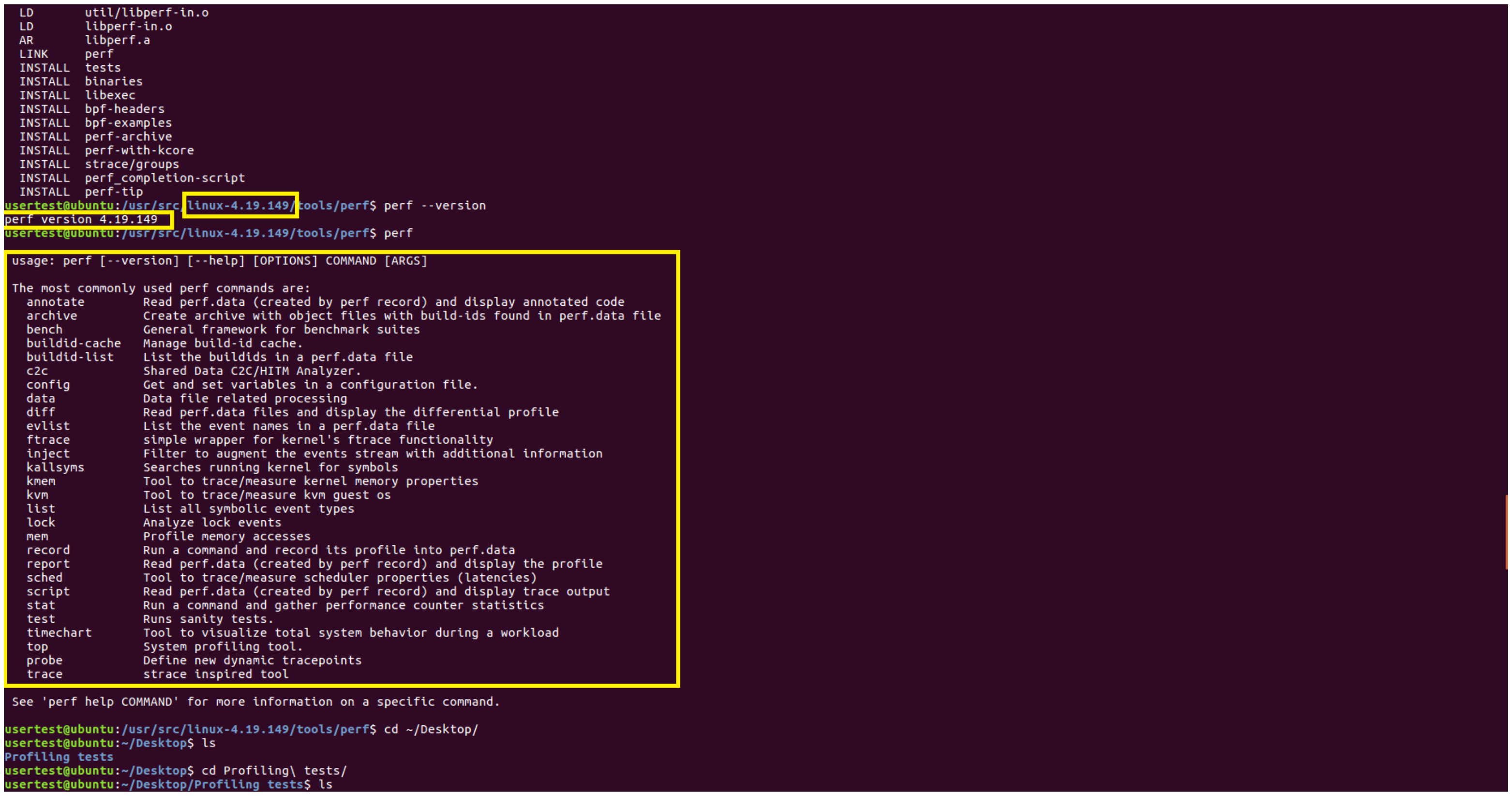
```
1265        }
1266    }
1267
1268    void perf_evsel__close_fd(struct perf_evsel *evsel)
1269    {
1270        int cpu, thread;
1271
1272        for (cpu = 0; cpu < xyarray__max_x(evsel->fd); cpu++)
1273            for (thread = 0; thread < xyarray__max_y(evsel->fd); ++thread) {
1274                close(FD(evsel, cpu, thread));
1275                FD(evsel, cpu, thread) = -1;
1276            }
1277    }
1278
1279    void perf_evsel__exit(struct perf_evsel *evsel)
1280    {
1281        assert(list_empty(&evsel->node));
1282        assert(evsel->evlist == NULL);
1283        perf_evsel__free_counts(evsel);
1284        perf_evsel__free_fd(evsel);
1285        perf_evsel__free_id(evsel);
1286        perf_evsel__free_config_terms(evsel);
1287        cgroup__put(evsel->cgrp);
1288        cpu_map__put(evsel->cpus);
1289        cpu_map__put(evsel->own_cpus);
1290        thread_map__put(evsel->threads);
1291        zfree(&evsel->group_name);
1292        zfree(&evsel->name);
1293
1294
1295
1296        perf_evsel__object.fini(evsel);
1297    }
1298
1299    void perf_evsel__delete(struct perf_evsel *evsel)
1300    {
1301        perf_evsel__exit(evsel);
1302        free(evsel);
1303    }
1304
1305    void perf_evsel__compute_deltas(struct perf_evsel *evsel, int cpu, int thread,
1306                                    struct perf_counts_values *count)
1307    {
1308        struct perf_counts_values tmp;
1309
1310        if (!evsel->prev_raw_counts)
1311            return;
1312
1313        if (cpu == -1) {
1314            tmp = evsel->prev_raw_counts->aggr;
1315            evsel->prev_raw_counts->aggr = *count;
1316        } else {
1317            tmp = *perf_counts(evsel->prev_raw_counts, cpu, thread);
1318            *perf_counts(evsel->prev_raw_counts, cpu, thread) = *count;
1319        }
1320
1321        count->val = count->val - tmp.val;
1322        count->ena = count->ena - tmp.ena;
```

```
1265        }
1266    }
1267
1268    void perf_evsel__close_fd(struct perf_evsel *evsel)
1269    {
1270        int cpu, thread;
1271
1272        for (cpu = 0; cpu < xyarray__max_x(evsel->fd); cpu++)
1273            for (thread = 0; thread < xyarray__max_y(evsel->fd); ++thread) {
1274                close(FD(evsel, cpu, thread));
1275                FD(evsel, cpu, thread) = -1;
1276            }
1277    }
1278
1279    void perf_evsel__exit(struct perf_evsel *evsel)
1280    {
1281        assert(list_empty(&evsel->node));
1282        assert(evsel->evlist == NULL);
1283        perf_evsel__free_counts(evsel);
1284        perf_evsel__free_fd(evsel);
1285        perf_evsel__free_id(evsel);
1286        perf_evsel__free_config_terms(evsel);
1287        cgroup__put(evsel->cgrp);
1288        cpu_map__put(evsel->cpus);
1289        cpu_map__put(evsel->own_cpus);
1290        thread_map__put(evsel->threads);
1291        zfree(&evsel->group_name);
1292        zfree(&evsel->name);
1293        zfree(&evsel->pmu_name);
1294        zfree(&evsel->per_pkg_mask);
1295        zfree(&evsel->metric_events);
1296        perf_evsel__object.fini(evsel);
1297    }
1298
1299    void perf_evsel__delete(struct perf_evsel *evsel)
1300    {
1301        perf_evsel__exit(evsel);
1302        free(evsel);
1303    }
1304
1305    void perf_evsel__compute_deltas(struct perf_evsel *evsel, int cpu, int thread,
1306                                    struct perf_counts_values *count)
1307    {
1308        struct perf_counts_values tmp;
1309
1310        if (!evsel->prev_raw_counts)
1311            return;
1312
1313        if (cpu == -1) {
1314            tmp = evsel->prev_raw_counts->aggr;
1315            evsel->prev_raw_counts->aggr = *count;
1316        } else {
1317            tmp = *perf_counts(evsel->prev_raw_counts, cpu, thread);
1318            *perf_counts(evsel->prev_raw_counts, cpu, thread) = *count;
1319        }
1320
1321        count->val = count->val - tmp.val;
1322        count->ena = count->ena - tmp.ena;
```

```
C source file        length : 73,770    lines : 2,955        Ln : 1,295    Col : 34    Sel : 85 | 3        Unix (LF)        UTF-8        INS
```

After commenting these lines, save the evsel.c file, and run the commands
$ sudo make
$ sudo make install
(Please refer to Section 2.1 Project 1B)

After patching this file, the build process should be successful.



```
  LD       util/libperf-in.o
  LD       libperf-in.o
  AR       libperf.a
  LINK     perf
  INSTALL  tests
  INSTALL  binaries
  INSTALL  libexec
  INSTALL  bpf-headers
  INSTALL  bpf-examples
  INSTALL  perf-archive
  INSTALL  perf-with-kcore
  INSTALL  strace/groups
  INSTALL  perf_completion-script
  INSTALL  perf-tip
usertest@ubuntu:/usr/src/linux-4.19.149/tools/perf$ perf --version
perf version 4.19.149
usertest@ubuntu:/usr/src/linux-4.19.149/tools/perf$ perf

 usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]

 The most commonly used perf commands are:
   annotate        Read perf.data (created by perf record) and display annotated code
   archive         Create archive with object files with build-ids found in perf.data file
   bench           General framework for benchmark suites
   buildid-cache   Manage build-id cache.
   buildid-list    List the buildids in a perf.data file
   c2c             Shared Data C2C/HITM Analyzer.
   config          Get and set variables in a configuration file.
   data            Data file related processing
   diff            Read perf.data files and display the differential profile
   evlist          List the event names in a perf.data file
   ftrace          simple wrapper for kernel's ftrace functionality
   inject          Filter to augment the events stream with additional information
   kallsyms        Searches running kernel for symbols
   kmem            Tool to trace/measure kernel memory properties
   kvm             Tool to trace/measure kvm guest os
   list            List all symbolic event types
   lock            Analyze lock events
   mem             Profile memory accesses
   record          Run a command and record its profile into perf.data
   report          Read perf.data (created by perf record) and display the profile
   sched           Tool to trace/measure scheduler properties (latencies)
   script          Read perf.data (created by perf record) and display trace output
   stat            Run a command and gather performance counter statistics
   test            Runs sanity tests.
   timechart       Tool to visualize total system behavior during a workload
   top             System profiling tool.
   probe           Define new dynamic tracepoints
   trace           strace inspired tool

 See 'perf help COMMAND' for more information on a specific command.

usertest@ubuntu:/usr/src/linux-4.19.149/tools/perf$ cd ~/Desktop/
usertest@ubuntu:~/Desktop$ ls
Profiling tests
usertest@ubuntu:~/Desktop$ cd Profiling\ tests/
usertest@ubuntu:~/Desktop/Profiling tests$ ls
```

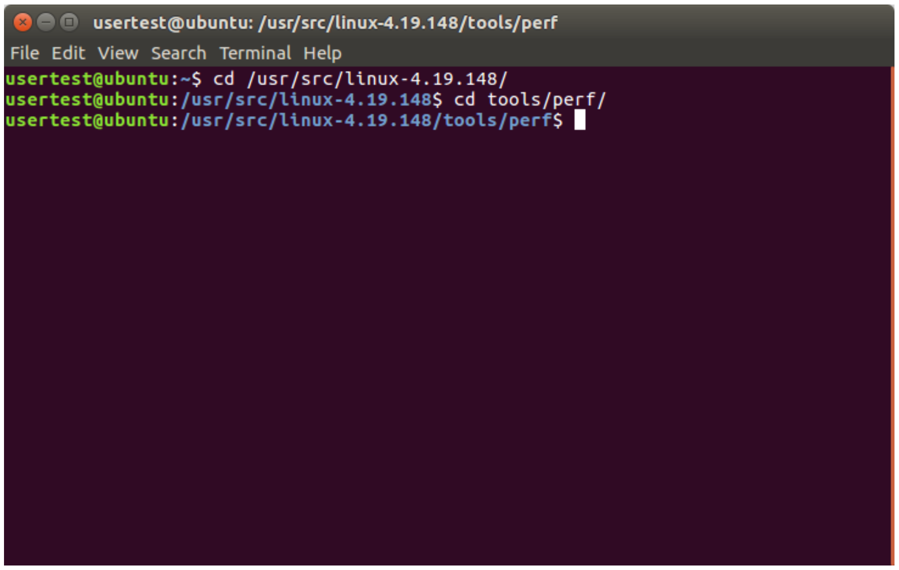
IMPORTANT NOTES:

1. This patch is a **best-effort** solution (so no guarantees that this will solve all the possible problems).
   If more problems are found later in the affected kernels, please use kernel 4.19.148.
2. In order to avoid this situation in the future, **Project 2 and Project 3 will require to use kernel 4.19.148** (so you will have to download and compile kernel 4.19.148)**.**
3. For more information, please check the related post on E3 forum:
   https://e3new.nctu.edu.tw/mod/forum/discuss.php?d=72257

Section 2.1: Profiling tool installation

In this section, we will build the profiling function **perf**.

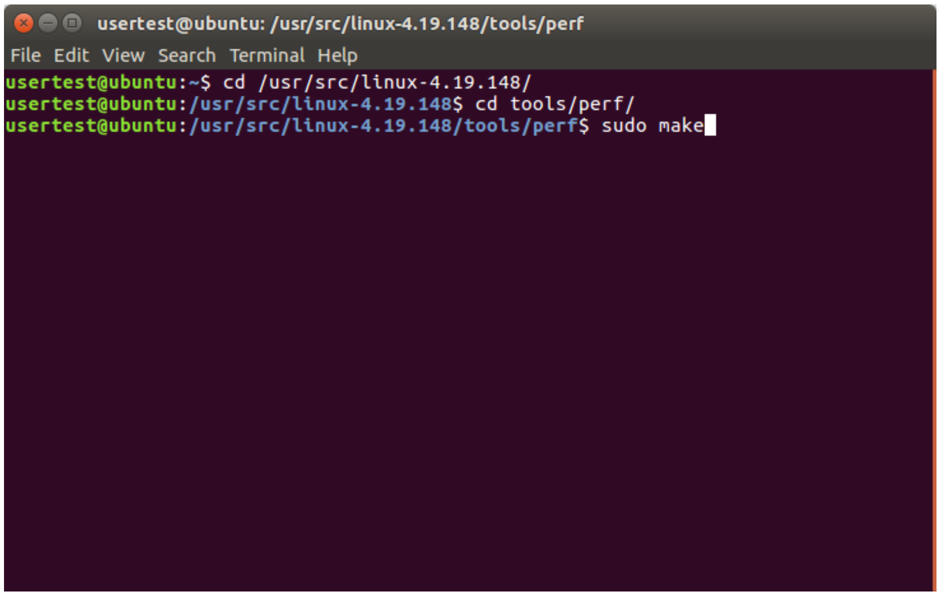
1. Go to the Linux source code folder, and go to **/tools/perf.**



usertest@ubuntu: /usr/src/linux-4.19.148/tools/perf
File Edit View Search Terminal Help

```
usertest@ubuntu:~$ cd /usr/src/linux-4.19.148/
usertest@ubuntu:/usr/src/linux-4.19.148$ cd tools/perf/
usertest@ubuntu:/usr/src/linux-4.19.148/tools/perf$ 
```

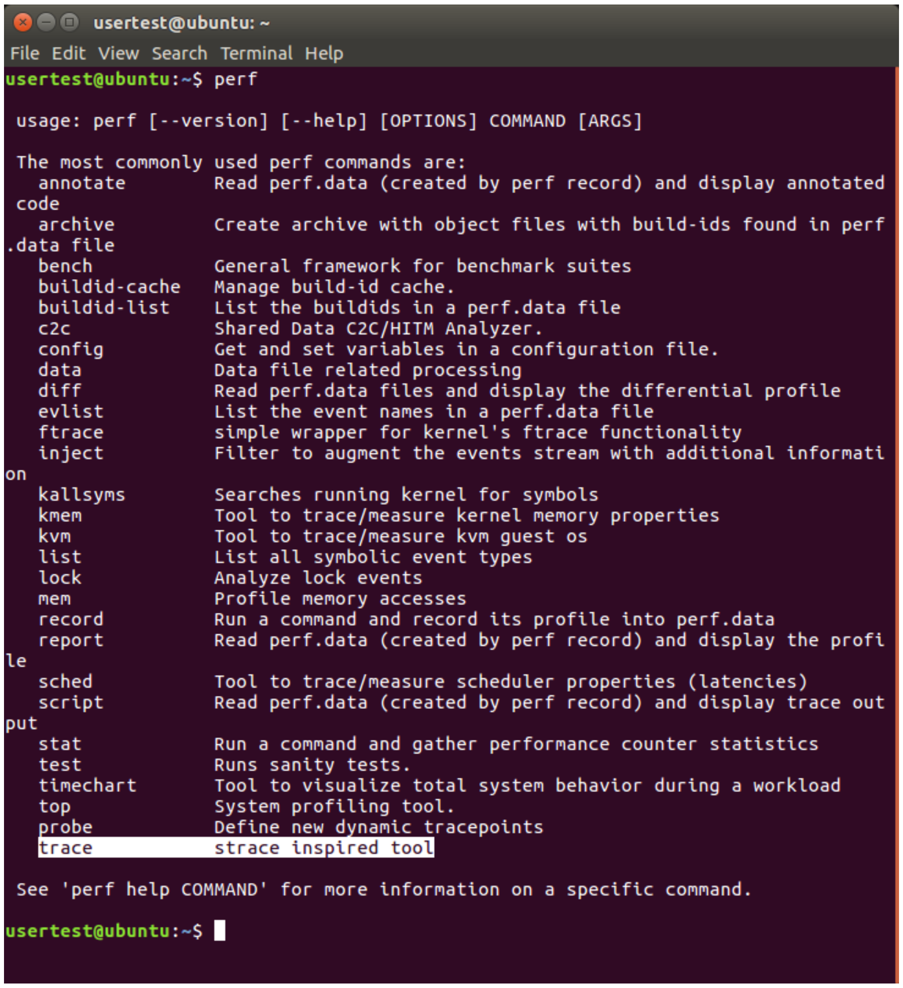
2. Run the commands
   $sudo make
   $sudo make install



usertest@ubuntu: /usr/src/linux-4.19.148/tools/perf
File Edit View Search Terminal Help

```
usertest@ubuntu:~$ cd /usr/src/linux-4.19.148/
usertest@ubuntu:/usr/src/linux-4.19.148$ cd tools/perf/
usertest@ubuntu:/usr/src/linux-4.19.148/tools/perf$ sudo make
```

3. At this point, the perf tool should be created. Type **$perf** to verify that the tool was built successfully.

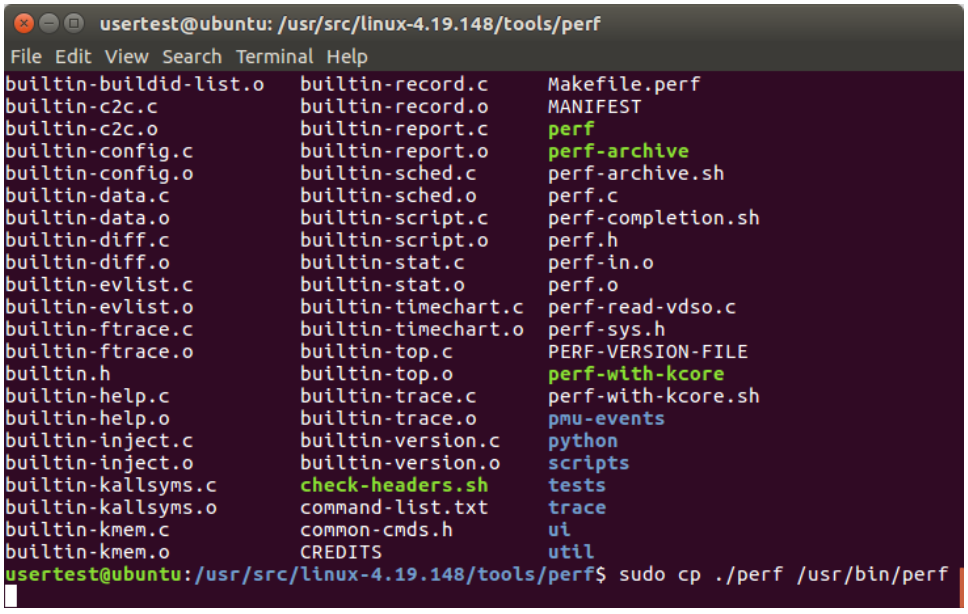


```
usertest@ubuntu:~$ perf

 usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]

 The most commonly used perf commands are:
   annotate        Read perf.data (created by perf record) and display annotated
 code
   archive         Create archive with object files with build-ids found in perf
.data file
   bench           General framework for benchmark suites
   buildid-cache   Manage build-id cache.
   buildid-list    List the buildids in a perf.data file
   c2c             Shared Data C2C/HITM Analyzer.
   config          Get and set variables in a configuration file.
   data            Data file related processing
   diff            Read perf.data files and display the differential profile
   evlist          List the event names in a perf.data file
   ftrace          simple wrapper for kernel's ftrace functionality
   inject          Filter to augment the events stream with additional informati
on
   kallsyms        Searches running kernel for symbols
   kmem            Tool to trace/measure kernel memory properties
   kvm             Tool to trace/measure kvm guest os
   list            List all symbolic event types
   lock            Analyze lock events
   mem             Profile memory accesses
   record          Run a command and record its profile into perf.data
   report          Read perf.data (created by perf record) and display the profi
le
   sched           Tool to trace/measure scheduler properties (latencies)
   script          Read perf.data (created by perf record) and display trace out
put
   stat            Run a command and gather performance counter statistics
   test            Runs sanity tests.
   timechart       Tool to visualize total system behavior during a workload
   top             System profiling tool.
   probe           Define new dynamic tracepoints
   trace           strace inspired tool

 See 'perf help COMMAND' for more information on a specific command.

usertest@ubuntu:~$
```

IMPORTANT NOTE: There is a chance that the command **trace** was not generated. If that happens, re-do step 2 and now it should appear in the list of commands.

[Screenshot # 16: Create a screenshot showing the perf tool commands like in the previous screenshot. Your student ID must be visible.]

4. We still need to move the compiled perf tool to a location where all the system can access it. Run the command
   $ sudo cp ./perf /usr/bin/perf


usertest@ubuntu: /usr/src/linux-4.19.148/tools/perf
File Edit View Search Terminal Help

```
builtin-buildid-list.o    builtin-record.c     Makefile.perf
builtin-c2c.c             builtin-record.o     MANIFEST
builtin-c2c.o             builtin-report.c     perf
builtin-config.c          builtin-report.o     perf-archive
builtin-config.o          builtin-sched.c      perf-archive.sh
builtin-data.c            builtin-sched.o      perf.c
builtin-data.o            builtin-script.c     perf-completion.sh
builtin-diff.c            builtin-script.o     perf.h
builtin-diff.o            builtin-stat.c       perf-in.o
builtin-evlist.c          builtin-stat.o       perf.o
builtin-evlist.o          builtin-timechart.c  perf-read-vdso.c
builtin-ftrace.c          builtin-timechart.o  perf-sys.h
builtin-ftrace.o          builtin-top.c        PERF-VERSION-FILE
builtin.h                 builtin-top.o        perf-with-kcore
builtin-help.c            builtin-trace.c      perf-with-kcore.sh
builtin-help.o            builtin-trace.o      pmu-events
builtin-inject.c          builtin-version.c    python
builtin-inject.o          builtin-version.o    scripts
builtin-kallsyms.c        check-headers.sh     tests
builtin-kallsyms.o        command-list.txt     trace
builtin-kmem.c            common-cmds.h        ui
builtin-kmem.o            CREDITS              util
usertest@ubuntu:/usr/src/linux-4.19.148/tools/perf$ sudo cp ./perf /usr/bin/perf
```

At this point, we are ready to profile kernel functions.

Useful links:

- https://blog.csdn.net/tang05505622334/article/details/103057179

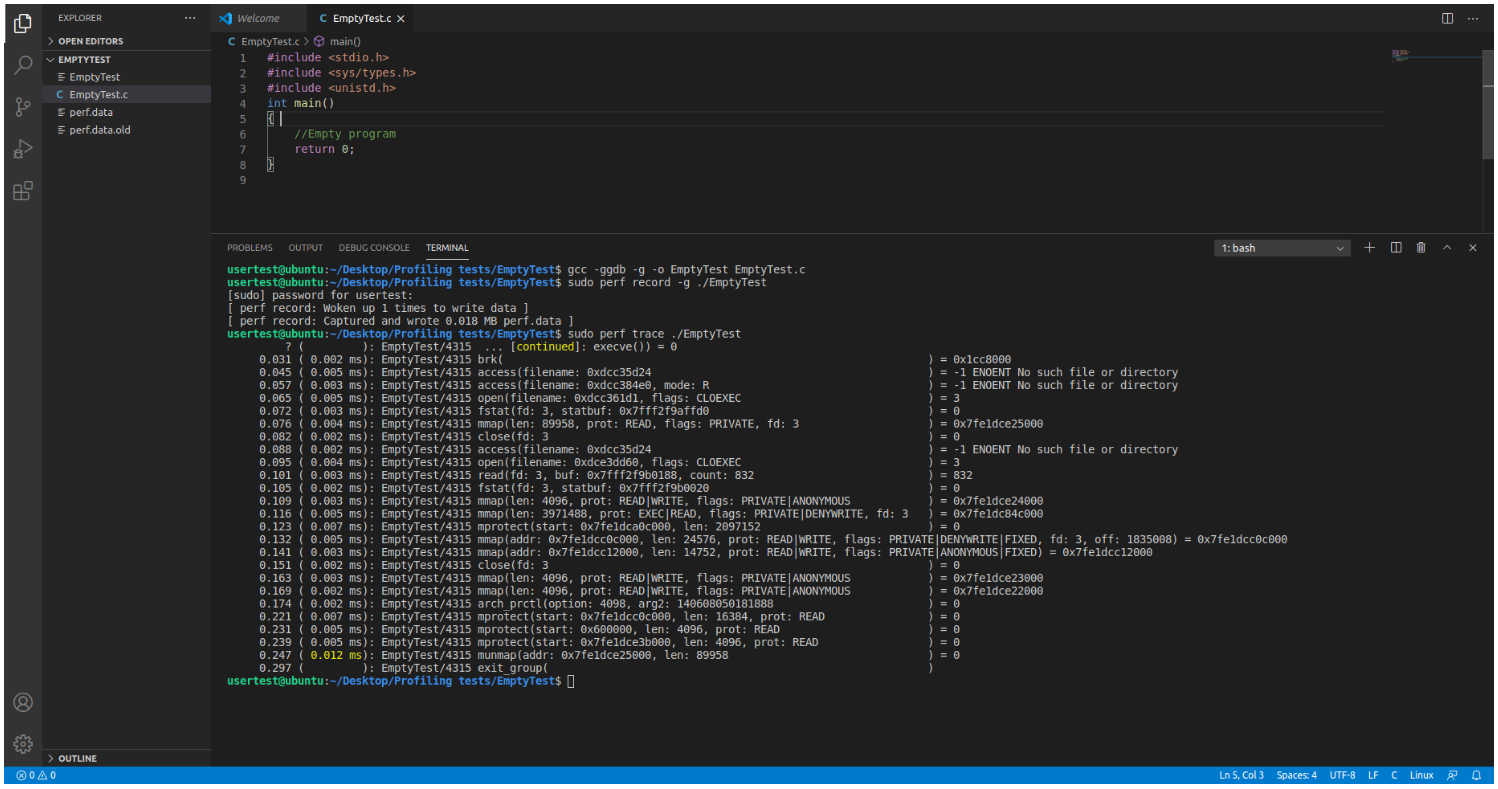
Section 2.2: Profiling functions

In this section, we will show how to use **perf**.

Notes:

- Perf will be used to profile c programs that use system calls.
- We need to use special headers in our programs (they depend on each system call).
- Perf must be executed as root (so sudo is necessary).
- The programs used in this section are available in E3.

1. Create a folder called **Profiling tests** in your Desktop.
2. Create two folders: **emptyTest** and **fileCopyTest**.
3. For **emptyTest**, create the following program and call it **emptyTest.c**



```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    //Empty program
    return 0;
}
```

```
usertest@ubuntu:~/Desktop/Profiling tests/EmptyTest$ gcc -ggdb -g -o EmptyTest EmptyTest.c
usertest@ubuntu:~/Desktop/Profiling tests/EmptyTest$ sudo perf record -g ./EmptyTest
[sudo] password for usertest:
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.018 MB perf.data ]
usertest@ubuntu:~/Desktop/Profiling tests/EmptyTest$ sudo perf trace ./EmptyTest
         ? (         ): EmptyTest/4315  ... [continued]: execve()) = 0
     0.031 ( 0.002 ms): EmptyTest/4315 brk(                                                                  ) = 0x1cc8000
     0.045 ( 0.005 ms): EmptyTest/4315 access(filename: 0xdcc35d24                                           ) = -1 ENOENT No such file or directory
     0.057 ( 0.003 ms): EmptyTest/4315 access(filename: 0xdcc384e0, mode: R                                  ) = -1 ENOENT No such file or directory
     0.065 ( 0.005 ms): EmptyTest/4315 open(filename: 0xdcc361d1, flags: CLOEXEC                             ) = 3
     0.072 ( 0.003 ms): EmptyTest/4315 fstat(fd: 3, statbuf: 0x7fff2f9affd0                                  ) = 0
     0.076 ( 0.004 ms): EmptyTest/4315 mmap(len: 89958, prot: READ, flags: PRIVATE, fd: 3                    ) = 0x7fe1dce25000
     0.082 ( 0.002 ms): EmptyTest/4315 close(fd: 3                                                           ) = 0
     0.088 ( 0.002 ms): EmptyTest/4315 access(filename: 0xdcc35d24                                           ) = -1 ENOENT No such file or directory
     0.095 ( 0.004 ms): EmptyTest/4315 open(filename: 0xdce3dd60, flags: CLOEXEC                             ) = 3
     0.101 ( 0.003 ms): EmptyTest/4315 read(fd: 3, buf: 0x7fff2f9b0188, count: 832                           ) = 832
     0.105 ( 0.002 ms): EmptyTest/4315 fstat(fd: 3, statbuf: 0x7fff2f9b0020                                  ) = 0
     0.109 ( 0.003 ms): EmptyTest/4315 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS            ) = 0x7fe1dce24000
     0.116 ( 0.005 ms): EmptyTest/4315 mmap(len: 3971488, prot: EXEC|READ, flags: PRIVATE|DENYWRITE, fd: 3   ) = 0x7fe1dc84c000
     0.123 ( 0.007 ms): EmptyTest/4315 mprotect(start: 0x7fe1dca0c000, len: 2097152                          ) = 0
     0.132 ( 0.005 ms): EmptyTest/4315 mmap(addr: 0x7fe1dcc0c000, len: 24576, prot: READ|WRITE, flags: PRIVATE|DENYWRITE|FIXED, fd: 3, off: 1835008) = 0x7fe1dcc0c000
     0.141 ( 0.003 ms): EmptyTest/4315 mmap(addr: 0x7fe1dcc12000, len: 14752, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS|FIXED) = 0x7fe1dcc12000
     0.151 ( 0.002 ms): EmptyTest/4315 close(fd: 3                                                           ) = 0
     0.163 ( 0.003 ms): EmptyTest/4315 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS            ) = 0x7fe1dce23000
     0.169 ( 0.002 ms): EmptyTest/4315 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS            ) = 0x7fe1dce22000
     0.174 ( 0.002 ms): EmptyTest/4315 arch_prctl(option: 4098, arg2: 140608050181888                        ) = 0
     0.221 ( 0.007 ms): EmptyTest/4315 mprotect(start: 0x7fe1dcc0c000, len: 16384, prot: READ                ) = 0
     0.231 ( 0.005 ms): EmptyTest/4315 mprotect(start: 0x600000, len: 4096, prot: READ                       ) = 0
     0.239 ( 0.005 ms): EmptyTest/4315 mprotect(start: 0x7fe1dce3b000, len: 4096, prot: READ                 ) = 0
     0.247 ( 0.012 ms): EmptyTest/4315 munmap(addr: 0x7fe1dce25000, len: 89958                               ) = 0
     0.297 (         ): EmptyTest/4315 exit_group(                                                           )
usertest@ubuntu:~/Desktop/Profiling tests/EmptyTest$ 
```

4. Run the commands

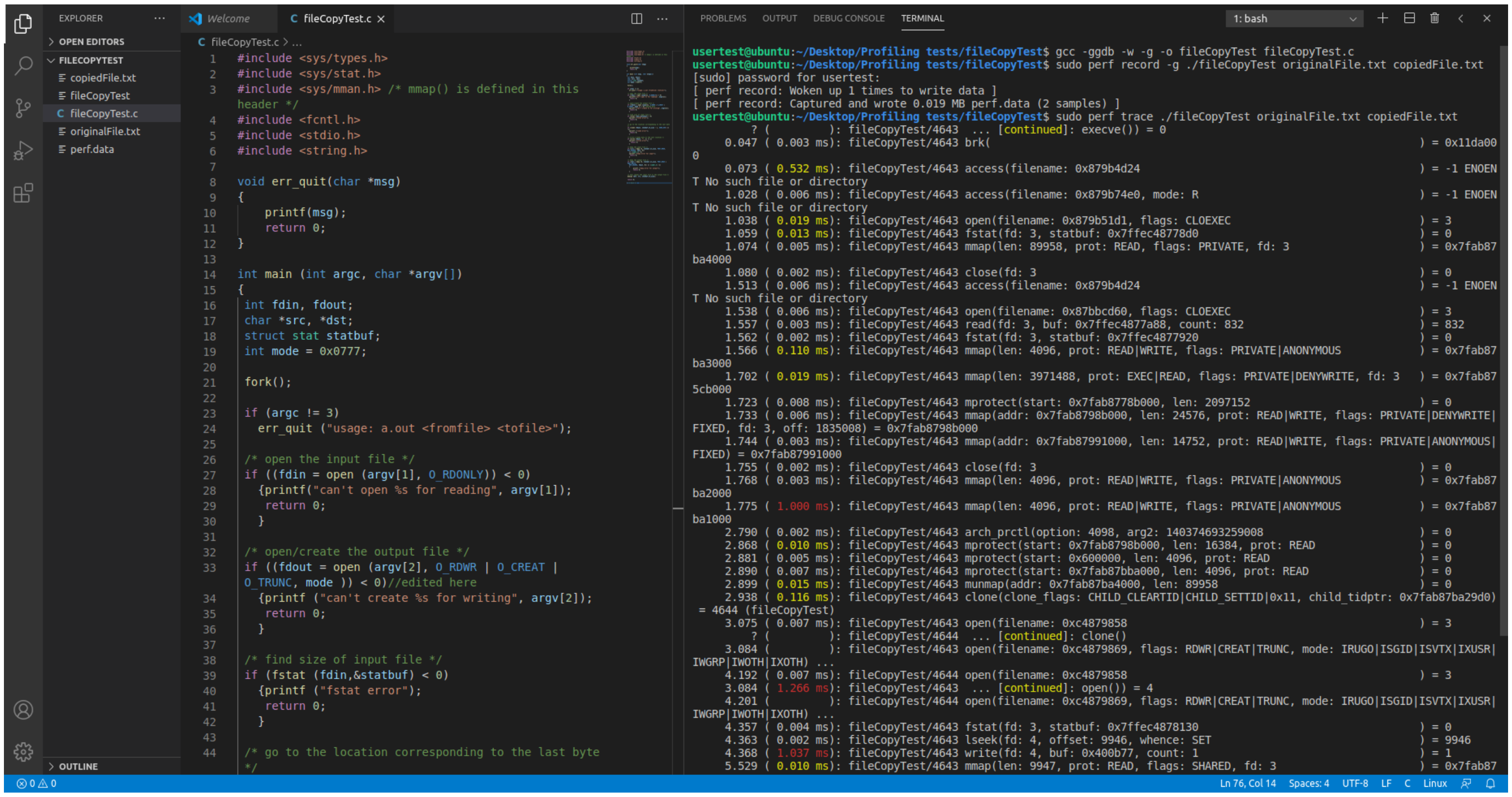
   $ gcc -ggdb -g -o emptyTest emptyTest.c

   $ sudo perf record -g ./emptyTest

   $ sudo perf trace ./emptyTest
5. Copy the result of the trace command, and save it in a text file (call it **emptyTest.txt**).

From the previous image, you can see that perf record generates several lines of code for an empty file. These lines are common in any profiled compiled file. We will show how to ignore them.

6. For **fileCopyTest**, create the following program and call it **fileCopyTest.c**



```c
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h> /* mmap() is defined in this
header */
#include <fcntl.h>
#include <stdio.h>
#include <string.h>

void err_quit(char *msg)
{
    printf(msg);
    return 0;
}

int main (int argc, char *argv[])
{
 int fdin, fdout;
 char *src, *dst;
 struct stat statbuf;
 int mode = 0x0777;

 fork();

 if (argc != 3)
   err_quit ("usage: a.out <fromfile> <tofile>");

 /* open the input file */
 if ((fdin = open (argv[1], O_RDONLY)) < 0)
   {printf("can't open %s for reading", argv[1]);
    return 0;
   }

 /* open/create the output file */
 if ((fdout = open (argv[2], O_RDWR | O_CREAT |
O_TRUNC, mode )) < 0)//edited here
   {printf ("can't create %s for writing", argv[2]);
    return 0;
   }

 /* find size of input file */
 if (fstat (fdin,&statbuf) < 0)
   {printf ("fstat error");
    return 0;
   }

 /* go to the location corresponding to the last byte
*/
```

```
usertest@ubuntu:~/Desktop/Profiling tests/fileCopyTest$ gcc -ggdb -w -g -o fileCopyTest fileCopyTest.c
usertest@ubuntu:~/Desktop/Profiling tests/fileCopyTest$ sudo perf record -g ./fileCopyTest originalFile.txt copiedFile.txt
[sudo] password for usertest:
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.019 MB perf.data (2 samples) ]
usertest@ubuntu:~/Desktop/Profiling tests/fileCopyTest$ sudo perf trace ./fileCopyTest originalFile.txt copiedFile.txt
         ? (         ): fileCopyTest/4643  ... [continued]: execve()) = 0
     0.047 ( 0.003 ms): fileCopyTest/4643 brk(                                                     ) = 0x11da00
0
     0.073 ( 0.532 ms): fileCopyTest/4643 access(filename: 0x879b4d24                              ) = -1 ENOEN
T No such file or directory
     1.028 ( 0.006 ms): fileCopyTest/4643 access(filename: 0x879b74e0, mode: R                     ) = -1 ENOEN
T No such file or directory
     1.038 ( 0.019 ms): fileCopyTest/4643 open(filename: 0x879b51d1, flags: CLOEXEC                ) = 3
     1.059 ( 0.013 ms): fileCopyTest/4643 fstat(fd: 3, statbuf: 0x7ffec48778d0                     ) = 0
     1.074 ( 0.005 ms): fileCopyTest/4643 mmap(len: 89958, prot: READ, flags: PRIVATE, fd: 3       ) = 0x7fab87
ba4000
     1.080 ( 0.002 ms): fileCopyTest/4643 close(fd: 3                                              ) = 0
     1.513 ( 0.006 ms): fileCopyTest/4643 access(filename: 0x879b4d24                              ) = -1 ENOEN
T No such file or directory
     1.538 ( 0.006 ms): fileCopyTest/4643 open(filename: 0x87bbcd60, flags: CLOEXEC                ) = 3
     1.557 ( 0.003 ms): fileCopyTest/4643 read(fd: 3, buf: 0x7ffec4877a88, count: 832              ) = 832
     1.562 ( 0.002 ms): fileCopyTest/4643 fstat(fd: 3, statbuf: 0x7ffec4877920                     ) = 0
     1.566 ( 0.110 ms): fileCopyTest/4643 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS         ) = 0x7fab87
ba3000
     1.702 ( 0.019 ms): fileCopyTest/4643 mmap(len: 3971488, prot: EXEC|READ, flags: PRIVATE|DENYWRITE, fd: 3   ) = 0x7fab87
5cb000
     1.723 ( 0.008 ms): fileCopyTest/4643 mprotect(start: 0x7fab8778b000, len: 2097152             ) = 0
     1.733 ( 0.006 ms): fileCopyTest/4643 mmap(addr: 0x7fab8798b000, len: 24576, prot: READ|WRITE, flags: PRIVATE|DENYWRITE|
FIXED, fd: 3, off: 1835008) = 0x7fab8798b000
     1.744 ( 0.003 ms): fileCopyTest/4643 mmap(addr: 0x7fab87991000, len: 14752, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS|
FIXED) = 0x7fab87991000
     1.755 ( 0.002 ms): fileCopyTest/4643 close(fd: 3                                              ) = 0
     1.768 ( 0.003 ms): fileCopyTest/4643 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS         ) = 0x7fab87
ba2000
     1.775 ( 1.000 ms): fileCopyTest/4643 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS         ) = 0x7fab87
ba1000
     2.790 ( 0.002 ms): fileCopyTest/4643 arch_prctl(option: 4098, arg2: 140374693259008          ) = 0
     2.868 ( 0.010 ms): fileCopyTest/4643 mprotect(start: 0x7fab8798b000, len: 16384, prot: READ   ) = 0
     2.881 ( 0.005 ms): fileCopyTest/4643 mprotect(start: 0x600000, len: 4096, prot: READ          ) = 0
     2.890 ( 0.007 ms): fileCopyTest/4643 mprotect(start: 0x7fab87bba000, len: 4096, prot: READ    ) = 0
     2.899 ( 0.015 ms): fileCopyTest/4643 munmap(addr: 0x7fab87ba4000, len: 89958                  ) = 0
     2.938 ( 0.116 ms): fileCopyTest/4643 clone(clone_flags: CHILD_CLEARTID|CHILD_SETTID|0x11, child_tidptr: 0x7fab87ba29d0)
 = 4644 (fileCopyTest)
     3.075 ( 0.007 ms): fileCopyTest/4643 open(filename: 0xc4879858                                ) = 3
         ? (         ): fileCopyTest/4644  ... [continued]: clone()
     3.084 (         ): fileCopyTest/4643 open(filename: 0xc4879869, flags: RDWR|CREAT|TRUNC, mode: IRUGO|ISGID|ISVTX|IXUSR|
IWGRP|IWOTH|IXOTH) ...
     4.192 ( 0.007 ms): fileCopyTest/4644 open(filename: 0xc4879858                                ) = 3
     3.084 ( 1.266 ms): fileCopyTest/4643  ... [continued]: open()) = 4
     4.201 (         ): fileCopyTest/4644 open(filename: 0xc4879869, flags: RDWR|CREAT|TRUNC, mode: IRUGO|ISGID|ISVTX|IXUSR|
IWGRP|IWOTH|IXOTH) ...
     4.357 ( 0.004 ms): fileCopyTest/4643 fstat(fd: 3, statbuf: 0x7ffec4878130                     ) = 0
     4.363 ( 0.002 ms): fileCopyTest/4643 lseek(fd: 4, offset: 9946, whence: SET                   ) = 9946
     4.368 ( 1.037 ms): fileCopyTest/4643 write(fd: 4, buf: 0x400b77, count: 1                     ) = 1
     5.529 ( 0.010 ms): fileCopyTest/4643 mmap(len: 9947, prot: READ, flags: SHARED, fd: 3         ) = 0x7fab87
```

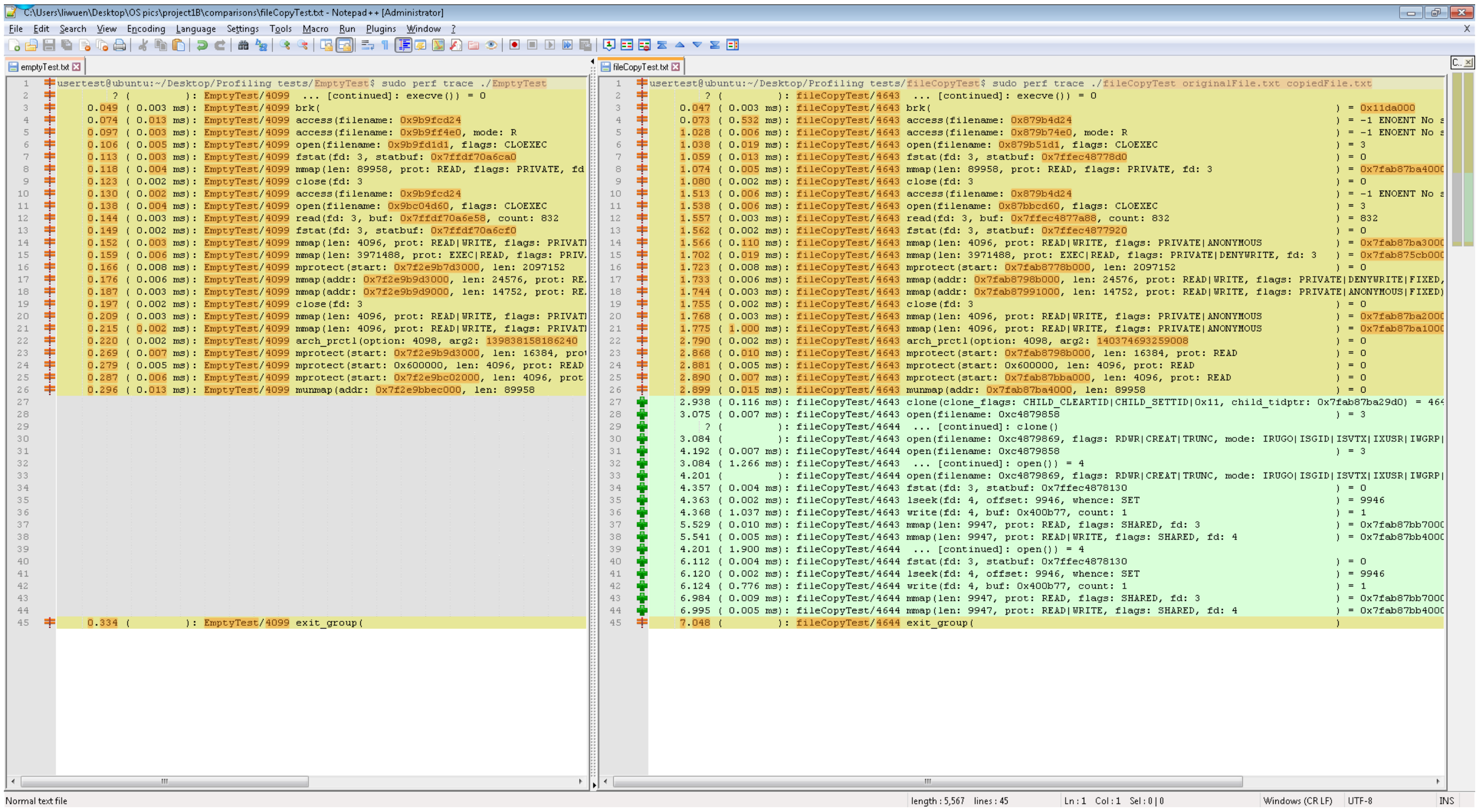
This program copies one text file into another file. It uses the functions **fork, mmap, write,** and **printf.**

7. Download the **originalFile.txt** from E3 and place it in the same folder as **fileCopyTest.c**.
8. Create a new text file and call it **copiedFile.txt.**
9. Run the following commands
   $ gcc -ggdb -w -g -o fileCopyTest fileCopyTest.c
   $ sudo perf record -g ./fileCopyTest originalFile.txt copiedFile.txt
   $ sudo perf trace ./fileCopyTest originalFile.txt copiedFile.txt

[Screenshot # 17 and #18: Create two screenshots showing these files and the trace results, as shown above. Your student ID must be visible.]

Note: In the previous screenshots, we used **visual code** to display the code and the execution results. You are encouraged to use it, but if you wish to use any other developing tool feel free to do so. Just be sure that your student ID is visible in the screenshots.

10. Using Notepad++, compare both files.


C:\Users\liwuen\Desktop\OS pics\project1B\comparisons\fileCopyTest.txt - Notepad++ [Administrator]

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
emptyTest.txt
fileCopyTest.txt

```
usertest@ubuntu:~/Desktop/Profiling tests/EmptyTest$ sudo perf trace ./EmptyTest
         ? (          ): EmptyTest/4099  ... [continued]: execve()) = 0
     0.049 ( 0.003 ms): EmptyTest/4099 brk(
     0.074 ( 0.013 ms): EmptyTest/4099 access(filename: 0x9b9fcd24
     0.097 ( 0.003 ms): EmptyTest/4099 access(filename: 0x9b9ff4e0, mode: R
     0.106 ( 0.005 ms): EmptyTest/4099 open(filename: 0x9b9fd1d1, flags: CLOEXEC
     0.113 ( 0.003 ms): EmptyTest/4099 fstat(fd: 3, statbuf: 0x7ffdf70a6ca0
     0.118 ( 0.004 ms): EmptyTest/4099 mmap(len: 89958, prot: READ, flags: PRIVATE, fd
     0.123 ( 0.002 ms): EmptyTest/4099 close(fd: 3
     0.130 ( 0.002 ms): EmptyTest/4099 access(filename: 0x9b9fcd24
     0.138 ( 0.004 ms): EmptyTest/4099 open(filename: 0x9bc04d60, flags: CLOEXEC
     0.144 ( 0.003 ms): EmptyTest/4099 read(fd: 3, buf: 0x7ffdf70a6e58, count: 832
     0.149 ( 0.002 ms): EmptyTest/4099 fstat(fd: 3, statbuf: 0x7ffdf70a6cf0
     0.152 ( 0.003 ms): EmptyTest/4099 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATI
     0.159 ( 0.006 ms): EmptyTest/4099 mmap(len: 3971488, prot: EXEC|READ, flags: PRIV.
     0.166 ( 0.008 ms): EmptyTest/4099 mprotect(start: 0x7f2e9b7d3000, len: 2097152
     0.176 ( 0.006 ms): EmptyTest/4099 mmap(addr: 0x7f2e9b9d3000, len: 24576, prot: RE.
     0.187 ( 0.003 ms): EmptyTest/4099 mmap(addr: 0x7f2e9b9d9000, len: 14752, prot: RE.
     0.197 ( 0.002 ms): EmptyTest/4099 close(fd: 3
     0.209 ( 0.003 ms): EmptyTest/4099 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATI
     0.215 ( 0.002 ms): EmptyTest/4099 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATI
     0.220 ( 0.002 ms): EmptyTest/4099 arch_prctl(option: 4098, arg2: 139838158186240
     0.269 ( 0.007 ms): EmptyTest/4099 mprotect(start: 0x7f2e9b9d3000, len: 16384, prot
     0.279 ( 0.005 ms): EmptyTest/4099 mprotect(start: 0x600000, len: 4096, prot: READ
     0.287 ( 0.006 ms): EmptyTest/4099 mprotect(start: 0x7f2e9bc02000, len: 4096, prot
     0.296 ( 0.013 ms): EmptyTest/4099 munmap(addr: 0x7f2e9bbec000, len: 89958
     0.334 (          ): EmptyTest/4099 exit_group(
```

```
usertest@ubuntu:~/Desktop/Profiling tests/fileCopyTest$ sudo perf trace ./fileCopyTest originalFile.txt copiedFile.txt
         ? (          ): fileCopyTest/4643  ... [continued]: execve()) = 0
     0.047 ( 0.003 ms): fileCopyTest/4643 brk(                                                   ) = 0x11da000
     0.073 ( 0.532 ms): fileCopyTest/4643 access(filename: 0x879b4d24                            ) = -1 ENOENT No s
     1.028 ( 0.006 ms): fileCopyTest/4643 access(filename: 0x879b74e0, mode: R                   ) = -1 ENOENT No s
     1.038 ( 0.019 ms): fileCopyTest/4643 open(filename: 0x879b51d1, flags: CLOEXEC              ) = 3
     1.059 ( 0.013 ms): fileCopyTest/4643 fstat(fd: 3, statbuf: 0x7ffec48778d0                   ) = 0
     1.074 ( 0.005 ms): fileCopyTest/4643 mmap(len: 89958, prot: READ, flags: PRIVATE, fd: 3     ) = 0x7fab87ba4000
     1.080 ( 0.002 ms): fileCopyTest/4643 close(fd: 3                                            ) = 0
     1.513 ( 0.006 ms): fileCopyTest/4643 access(filename: 0x879b4d24                            ) = -1 ENOENT No s
     1.538 ( 0.006 ms): fileCopyTest/4643 open(filename: 0x87bbcd60, flags: CLOEXEC              ) = 3
     1.557 ( 0.003 ms): fileCopyTest/4643 read(fd: 3, buf: 0x7ffec4877a88, count: 832            ) = 832
     1.562 ( 0.002 ms): fileCopyTest/4643 fstat(fd: 3, statbuf: 0x7ffec4877920                   ) = 0
     1.566 ( 0.110 ms): fileCopyTest/4643 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS ) = 0x7fab87ba3000
     1.702 ( 0.019 ms): fileCopyTest/4643 mmap(len: 3971488, prot: EXEC|READ, flags: PRIVATE|DENYWRITE, fd: 3 ) = 0x7fab875cb000
     1.723 ( 0.008 ms): fileCopyTest/4643 mprotect(start: 0x7fab8778b000, len: 2097152           ) = 0
     1.733 ( 0.006 ms): fileCopyTest/4643 mmap(addr: 0x7fab8798b000, len: 24576, prot: READ|WRITE, flags: PRIVATE|DENYWRITE|FIXED,
     1.744 ( 0.003 ms): fileCopyTest/4643 mmap(addr: 0x7fab87991000, len: 14752, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS|FIXED)
     1.755 ( 0.002 ms): fileCopyTest/4643 close(fd: 3                                            ) = 0
     1.768 ( 0.003 ms): fileCopyTest/4643 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS ) = 0x7fab87ba2000
     1.775 ( 1.000 ms): fileCopyTest/4643 mmap(len: 4096, prot: READ|WRITE, flags: PRIVATE|ANONYMOUS ) = 0x7fab87ba1000
     2.790 ( 0.002 ms): fileCopyTest/4643 arch_prctl(option: 4098, arg2: 140374693259008         ) = 0
     2.868 ( 0.010 ms): fileCopyTest/4643 mprotect(start: 0x7fab8798b000, len: 16384, prot: READ ) = 0
     2.881 ( 0.005 ms): fileCopyTest/4643 mprotect(start: 0x600000, len: 4096, prot: READ        ) = 0
     2.890 ( 0.007 ms): fileCopyTest/4643 mprotect(start: 0x7fab87bba000, len: 4096, prot: READ  ) = 0
     2.899 ( 0.015 ms): fileCopyTest/4643 munmap(addr: 0x7fab87ba4000, len: 89958                ) = 0
     2.938 ( 0.116 ms): fileCopyTest/4643 clone(clone_flags: CHILD_CLEARTID|CHILD_SETTID|0x11, child_tidptr: 0x7fab87ba29d0) = 464
     3.075 ( 0.007 ms): fileCopyTest/4643 open(filename: 0xc4879858                              ) = 3
         ? (          ): fileCopyTest/4644  ... [continued]: clone()
     3.084 (          ): fileCopyTest/4643 open(filename: 0xc4879869, flags: RDWR|CREAT|TRUNC, mode: IRUGO|ISGID|ISVTX|IXUSR|IWGRP|
     4.192 ( 0.007 ms): fileCopyTest/4644 open(filename: 0xc4879858                              ) = 3
     3.084 ( 1.266 ms): fileCopyTest/4643  ... [continued]: open()) = 4
     4.201 (          ): fileCopyTest/4644 open(filename: 0xc4879869, flags: RDWR|CREAT|TRUNC, mode: IRUGO|ISGID|ISVTX|IXUSR|IWGRP|
     4.357 ( 0.004 ms): fileCopyTest/4643 fstat(fd: 3, statbuf: 0x7ffec4878130                   ) = 0
     4.363 ( 0.002 ms): fileCopyTest/4643 lseek(fd: 4, offset: 9946, whence: SET                 ) = 9946
     4.368 ( 1.037 ms): fileCopyTest/4643 write(fd: 4, buf: 0x400b77, count: 1                   ) = 1
     5.529 ( 0.010 ms): fileCopyTest/4643 mmap(len: 9947, prot: READ, flags: SHARED, fd: 3       ) = 0x7fab87bb7000
     5.541 ( 0.005 ms): fileCopyTest/4643 mmap(len: 9947, prot: READ|WRITE, flags: SHARED, fd: 4 ) = 0x7fab87bb4000
     4.201 ( 1.900 ms): fileCopyTest/4644  ... [continued]: open()) = 4
     6.112 ( 0.004 ms): fileCopyTest/4644 fstat(fd: 3, statbuf: 0x7ffec4878130                   ) = 0
     6.120 ( 0.002 ms): fileCopyTest/4644 lseek(fd: 4, offset: 9946, whence: SET                 ) = 9946
     6.124 ( 0.776 ms): fileCopyTest/4644 write(fd: 4, buf: 0x400b77, count: 1                   ) = 1
     6.984 ( 0.009 ms): fileCopyTest/4644 mmap(len: 9947, prot: READ, flags: SHARED, fd: 3       ) = 0x7fab87bb7000
     6.995 ( 0.005 ms): fileCopyTest/4644 mmap(len: 9947, prot: READ|WRITE, flags: SHARED, fd: 4 ) = 0x7fab87bb4000
     7.048 (          ): fileCopyTest/4644 exit_group(                                           )
```

Normal text file   length : 5,567   lines : 45   Ln : 1   Col : 1   Sel : 0 | 0   Windows (CR LF)   UTF-8   INS

<u>Note:</u> In this step we used Notepad++ and its **compare** plugin. In case you cannot install Notepad++ or wish to use another tool that displays the differences between two text files, please feel free to do so, but be sure that the differences are clear enough.

If you want to use Notepad++, here it is explained how to turn on the compare plugin:

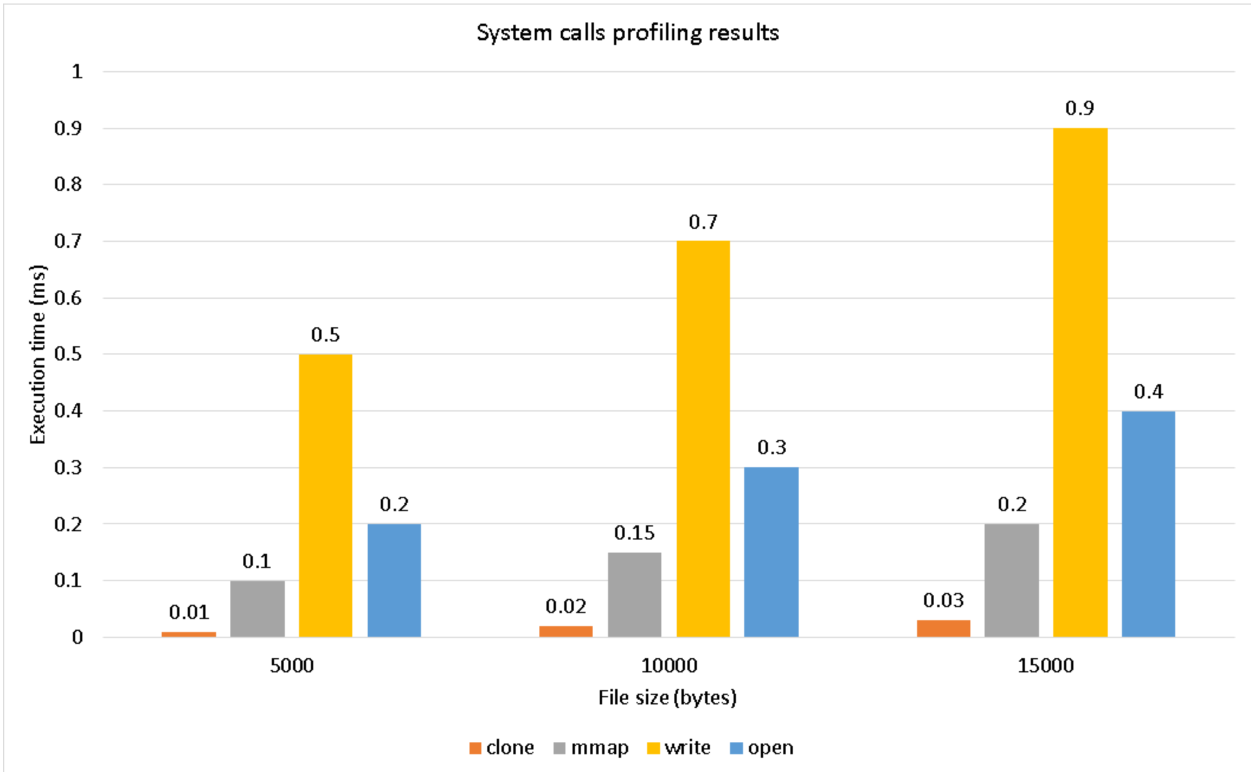
http://www.technicaloverload.com/compare-two-files-using-notepad/

<u>[Screenshot # 19: Create a screenshot showing the differences between these files.]</u>

We can see that **fileCopyTest.txt** has some extra lines which are invocations of the clone, mmap, write and open system calls. By using these lines we can calculate the average execution time of each function for this scenario.

Questions to answer (in both the report and video):

1. What are these functions: **clone**, **mmap**, **write** and **open?** **[Screenshot # 19]**
2. Why is there no **fork** system call? What is the difference between **fork** and **clone?** **[Screenshot # 19]**
3. Will the functions' execution time be longer if the file is bigger? **[Screenshot # 19]**
4. **Create a graph of file size (in bytes) vs. execution time (ms) of these four functions, using 3 different file sizes.**
   How is the behavior of each function? Sort them from slowest to fastest.
   (Example -from fastest to slowest- : clone, mmap, open, write).
   [Screenshot # 20]
   An example of the expected graph is shown below



(This is a dummy graph that only shows what is expected: **file size** vs. **execution time.** The real behavior of the functions is not reflected in this example.)

[Do it yourself] In the previous example we showed a comparisons between **fork**, **mmap**, **open,** and **write.**

Answer the following questions in both your report and video:

1. Perf also has the **report** command:
   $ **sudo perf report**
   Explain:
   a. What is it for? **[Screenshot # 21]**
   b. For **fileCopyTest**, show and interpret the results. **[Screenshot # 21]**
2. Perf has more commands (please refer to Section 2.1 step 3). Select another command (besides report, trace and record), explain what is it for and show how to use it. **[Screenshot # 22]**