Assignment 6 – Device Driver

Description

Virtualized ACPI Device Driver for FANS. The device tree was custom written for this project, in which it provides ACPI standard methods for querying and altering a FAN#'s state (# being the FAN number). These methods include turning the fan _ON and _OFF with read() and write(), and various ioctl commands, such as querying the FAN#'s current _FST (Fan Status), returning a flattened _FPS (Fan Performance State) package, setting the current FLVL (Level of the Fan Performance State), and querying a FAN#'s entire set of possible _FPS packages. There are also open() and release() functions to handle a user specified fan_handle through a file pointer.

How to Build and Use this Module

```
git clone <this_repo>
```

Since we are using a virtualized device tree, you need to set up a VM to run this module. I chose QEMU, because it is easy to inject the virtualized device tree into it with -acpitable. I have provided my script as a working example.

Once you have your script set up, and have properly loaded the vm, you can double check that the device tree was actually loaded by installing acpica-tools and running:

```
ls /sys/firmware/acpi/tables/
```

If you see the SSDT, it was successfully loaded. To double check, you can:

```
cat /sys/firmware/acpi/tables/SSDT > ssdt.bin
iasl -d ssdt.bin
```

And you should see my signatures, "VIRTIO" and "FAKEFANS".

I have provided what a successful one looks like in /Test. You can also see a successfully decompiled ssdt.dsl tree.

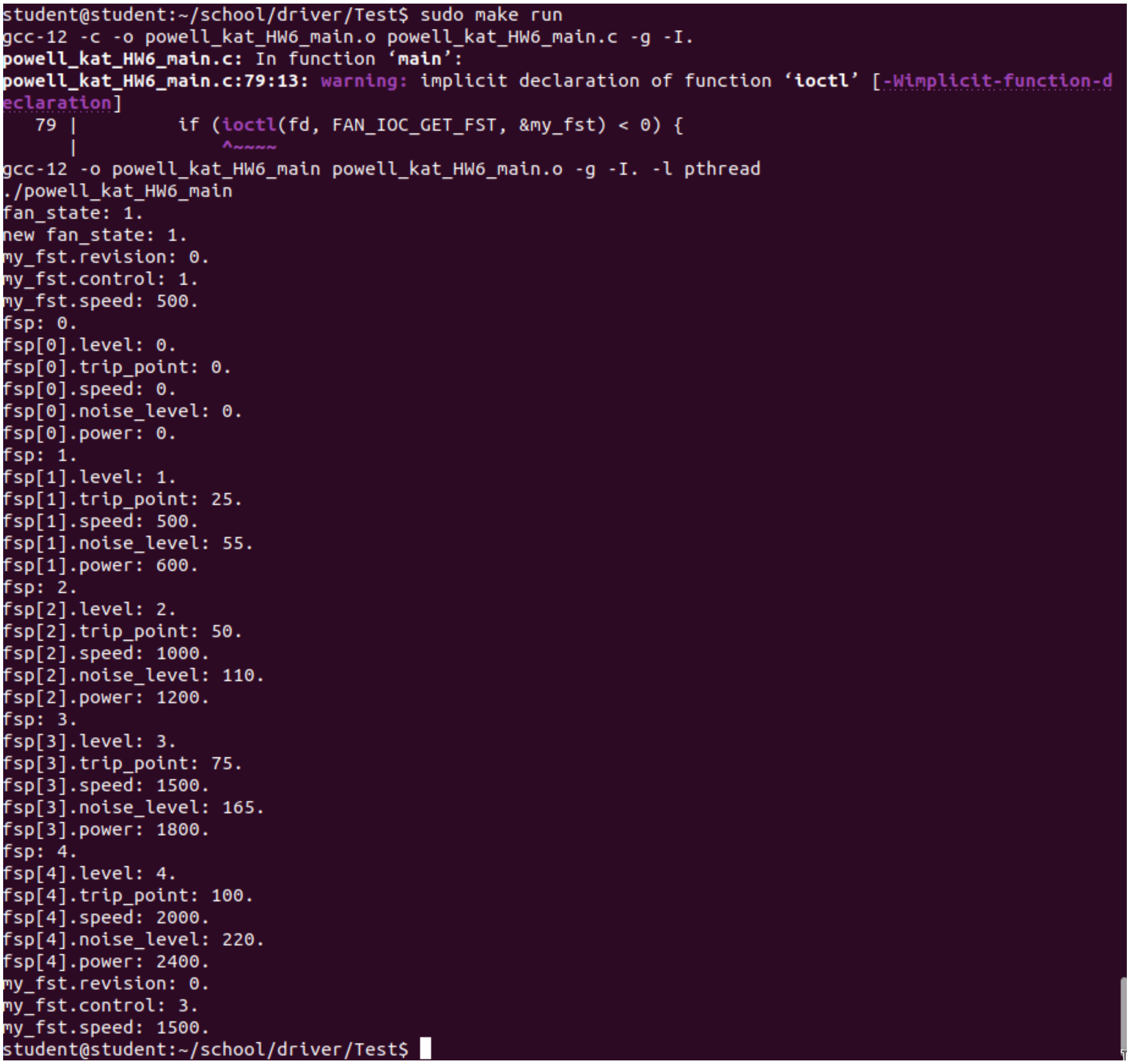
If it is not showing that, or not there, the table was not loaded correctly. Try again.

Now that it is loaded, you can proceed with testing the module.
Inside your VM:

```
git clone <this_repo>
```

```
cd Module
make clean
make
sudo insmod virtfan.ko
cd ../Test
sudo make clean
sudo make run
```

And we should see the user application print the state of the fan (ON or OFF), set it to ON, query the current FPS package, query the list of possible FPS packages, and change the fan state level to 3 (it prints the current FPS package after doing so).



```
student@student:~/school/driver/Test$ sudo make run
gcc-12 -c -o powell_kat_HW6_main.o powell_kat_HW6_main.c -g -I.
powell_kat_HW6_main.c: In function ‘main’:
powell_kat_HW6_main.c:79:13: warning: implicit declaration of function ‘ioctl’ [-Wimplicit-function-d
eclaration]
   79 |         if (ioctl(fd, FAN_IOC_GET_FST, &my_fst) < 0) {
      |             ^~~~~
gcc-12 -o powell_kat_HW6_main powell_kat_HW6_main.o -g -I. -l pthread
./powell_kat_HW6_main
fan_state: 1.
new fan_state: 1.
my_fst.revision: 0.
my_fst.control: 1.
my_fst.speed: 500.
fsp: 0.
fsp[0].level: 0.
fsp[0].trip_point: 0.
fsp[0].speed: 0.
fsp[0].noise_level: 0.
fsp[0].power: 0.
fsp: 1.
fsp[1].level: 1.
fsp[1].trip_point: 25.
fsp[1].speed: 500.
fsp[1].noise_level: 55.
fsp[1].power: 600.
fsp: 2.
fsp[2].level: 2.
fsp[2].trip_point: 50.
fsp[2].speed: 1000.
fsp[2].noise_level: 110.
fsp[2].power: 1200.
fsp: 3.
fsp[3].level: 3.
fsp[3].trip_point: 75.
fsp[3].speed: 1500.
fsp[3].noise_level: 165.
fsp[3].power: 1800.
fsp: 4.
fsp[4].level: 4.
fsp[4].trip_point: 100.
fsp[4].speed: 2000.
fsp[4].noise_level: 220.
fsp[4].power: 2400.
my_fst.revision: 0.
my_fst.control: 3.
my_fst.speed: 1500.
student@student:~/school/driver/Test$
```

Approach

Since I am doing an ACPI driver, I need a device tree. This device tree will have the nodes of the respective hardware that I want to manipulate. I want to make my device driver for fan management in ACPI. To do so, I will need to make a virtual ACPI fan device tree in ACPI source language (ASL). I then compile it with the AML, and then pass that through Qemu. From there, I can modify the ACPI fan device tree with my driver.

To make an ASL device tree

References

For this assignment, I will be mainly using Intel™'s ASL tutorial, along with referencing their ACPI(CA) manuals, UEFI's ACPI specification, and Linux source code for the necessary ACPICA calls.

Intel's ASL tutorial

UEFI's ACPI Specification

Linux Source Code

Overarching plan

What I want my device driver to do

Turn on a fan

1. Turn off a fan
2. Query the state of the fan

Assignment requires

1. Open the driver
2. Release the driver
3. Read - fan state
4. Write - to the fans state
5. At Least one Ioctl command

How I meet the requirements

- Open the driver
- Release the driver
- Read: Read the state of the fans
- Write: Change the state of the fans
- Ioctl: ?? (addressed later, not sure yet)

Stretch goals
Ideally, I would also like to incorporate a virtualized thermal device. With it, I could have the fan's state change based on what the temperature of the CPU is. But, the project scope may not provide enough time to accomplish both, so I am sticking with just the fans unless time allows.

The Plan

Virtualizing the device

The foundation is a definition block. Using Intel™'s ASL tutorial, we see It is structured like so:

```
DefinitionBlock (AMLFileName, TableSignature, ComplianceRevision,
OEMID, TableID, OEMRevision)
{
	TermList // A list of ASL terms
}
```

And it explains the parameters:

- AMLFileName —Name of the AML file (string). Can be a null string.
- TableSignature —Signature of the AML file (could be DSDT or SSDT) (4-character string)
- ComplianceRevision —A value of 2 or greater enables 64-bit arithmetic; a value of 1 or less enables 32-bit arithmetic (8 bit unsigned integer)
- OEMID —ID of the original equipment manufacturer (OEM) developing the ACPI table (6-character string)
- TableID —A specific identifier for the table (8-character string)
- OEMRevision —Revision number set by the OEM (32-bit number)

So, for my virtualized device tree, I want it to have:

`AMLFileName` - fakefans.aml
`TableSiganature` - `SSDT`.
The `SSDT` is loaded dynamically at run time, whereas the `DSDT` is loaded only at boot time. `SSDT` is a secondary, modifiable tree that adds to the `DSDT`. We do not want to overwrite the `DSDT`, simply supplement it with the new information.

`ComplianceRevision` - 2 as we are on 64-bit architecture.

Since we are virtualizing, the next three parameters are meaningless metadata. So, we will just use fake metadata here.

`OEMID` - VIRTIO	// 6 character string
`TableID` - FAKEFANS	// 8 character string
`OEMRevision` - 0x00000	// 32 bit number

The Term List

This is where the main implementation of our fan device will go. Here, we can give our device a name, methods, and structure.

Following ASL standard namespacing, I've created a fake device tree below. Some parameters are just metadata, and therefore are not real (i.e., "VIRTIO" for the OEMID). But where I could, I tried to mimic a real ASL tree by using ASL defined variables. For example:
SCOPE _SB: SB is ACPI reserved variable for system bus devices
DEVICE (FAN#) to mock real FAN names
_HID, EISAID("PNP0C0B"): _HID ACPI reserved variable for Hardware ID XXX
_UID, "VirtualFan0": _UID ACPI reserved variable for Unique ID of a device, giving it a pseudo unique identifier "VirtualFan0"

And so on…

Here is my initial planning of the Device Tree:

```
DefinitionBlock ("", "SSDT", 2, "VIRTIO", "FAKEFANS", 0x0)
{
      SCOPE (\_SB)
      {
            DEVICE (FAN0)
            {
                  Name (_HID, EISAID("PNP0C0B"))

                  Name (_UID, "VirtualFan0")

                  Name (_STA, 0x0F)     // To tell ACPI this device is

                                        // enabled

                  Name (FSTA, Zero)     // For tracking fan state

                  // To get this fan's package information

                  Method (_FIF, 0, NotSerialized)

                  {

                        Return (Package () {

                              0,        // Revision (0 is current)

                              0x00,     // Minimum level (0-100%)

                              0xC8,     // Maximum level (200%)
```

```
        0xA0,      // Step size (160)
        0x3C,      // Default level (60%)
        0x01,      // Flags (bit 0 == fine grained
                   // control)
        0x00       // Reserved
    })
}
// Return current power state
Method (_PSC, 0, NotSerialized)
{
    Return (FSTA)
}
// Simulate turning fan on
Method (_ON, 0, NotSerialized)
{
    Store (One, \FSTA)
}
// Simulate turning fan off
Method (_OFF, 0, NotSerialized)
{
    Store (Zero, \FSTA)
}
// Return power state
Method (_PSC, 0, NotSerialized)
{
```

```
                                Return (FSTA)

                          }

                     }
          }
}
```

Making the Device Driver

Our fake device tree has:

- `_FIF`, for getting the package information of a specified fan
- `_ON/_OFF`, for turning the fan ON/OFF
- `_PSC`, for querying the current power state

So, our driver can potentially:

- Get the package information of a specified fan, and print it
- Turn a specified fan ON/OFF
- Query the current power state

*Ideally, we would have the `_TZ` (Thermal Zone) tree, so we could query the thermal zones and change fan speed/state based on the thermal temperature of the computer. That is a limitation of this assignment due to time.

My Plan

I need to have some way of grabbing fan information and storing it in memory for the client. To do so, I will use a struct in memory to keep track of my driver's operations and state. My methods will include:

`load()`	to load the driver
`query(char *thing, int fans[])`	print `*thing` (i.e., `_PCS` or `_FIF)`
`on(int a[])`	`to turn on specified fans`
`off(int fans[])`	`to turn off specified fans`
`unload()`	`to cleanup the driver`

(EDITING KAT : THE ABOVE CHANGES, THIS IS AN INITIAL PLAN WITH LIMITED UNDERSTANDING.)

For ioctl, we need a character device structure. (With help from this cdev tutorial) Linux's cdev struct is:

```
struct cdev {
     struct kobject kobj;
```

```
    struct module *owner;
    const struct file_operations *ops;
    struct list_head list;
    dev_t dev;
    unsigned int count;
}
```

Specifically, we need to specify `file_operations`. This is where we can specify `open/read/write/release/etc.`, and thus a user can call them.

`open()`	Receives the index of the desired fan and stores it
`release()`	Removes the index of the desired fan
`read()`	Reads the _PCS state of the fan
`write()`	Writes the _PCS state of the fan
`ioctl()`	?? (I address this later in my issues and resolutions)

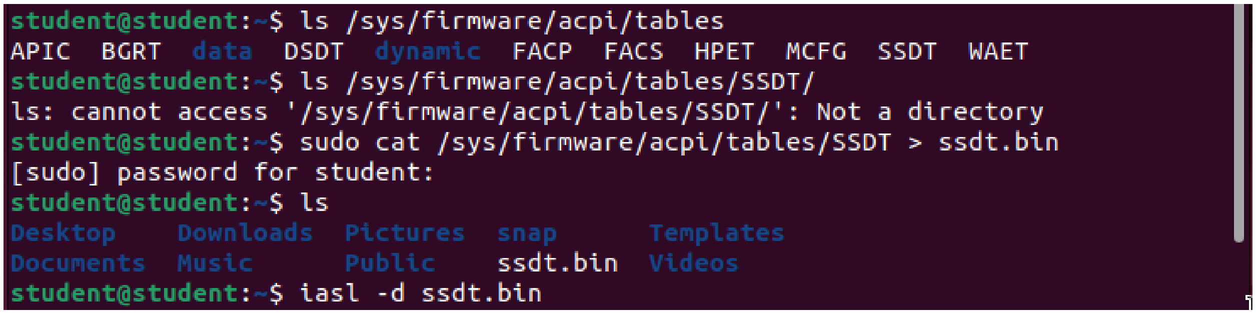
Issues and Resolutions:

Passing the Virtualized Tree Through QEMU

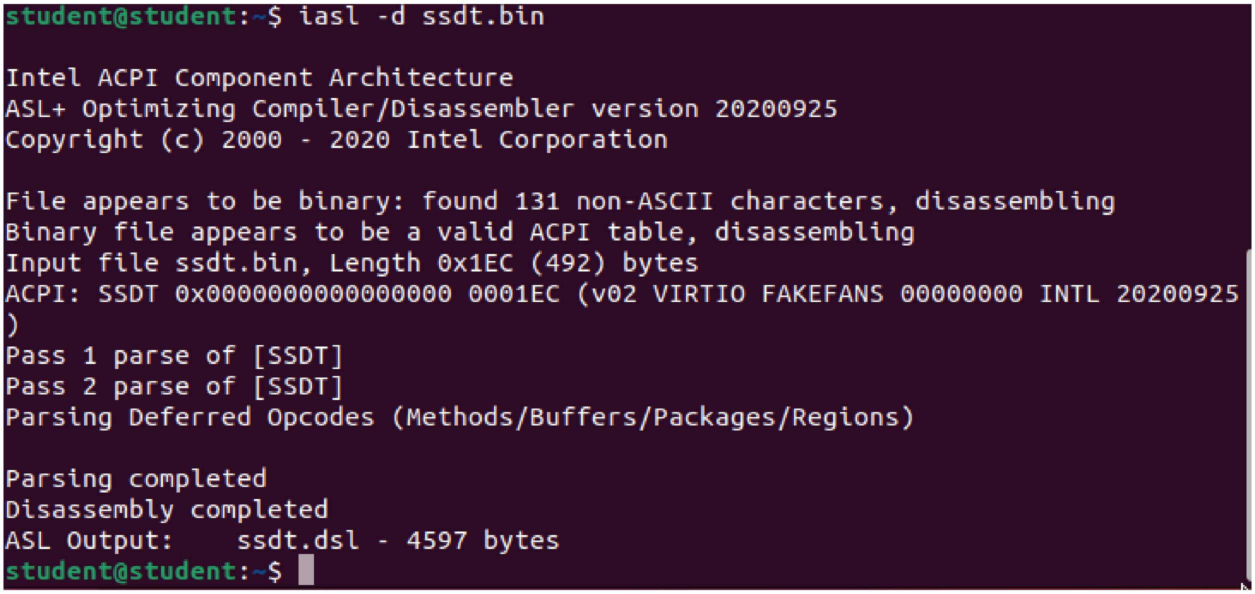
Now that we have our compiled device tree, we need to pass it through QEMU. We can do so by using the flag `-acpitable file=/home/bee/vm/csc415/fake_fans.aml`. To verify, we can look inside `/sys/firmware/acpi/tables`. Here, we should see `SSDT`, which is what we specified our device tree under. And, if we cat it to another file and decompile it, we should see the device tree signature. We do! We see:

`(v02 VIRTIO FAKEFANS 00000000 INTL 20200925)`

`v02`	Version number? (but I cannot confirm of what)
VIRTIO	The OEMID I gave my device tree
FAKEFANS	The TableID I gave my device tree
`00000000`	The 8-bit OEMRevision number I gave my device tree
`INTL 20200925`	Intel™, and its disassembler version



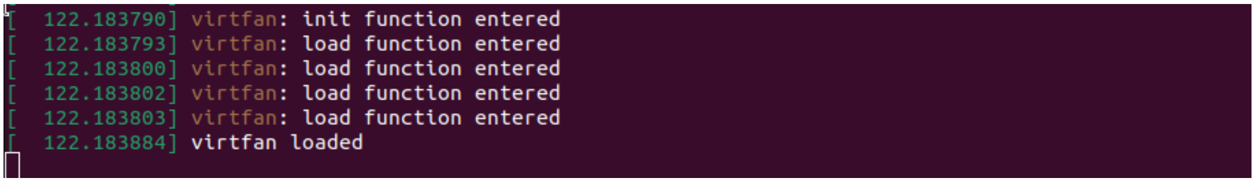
```
student@student:~$ ls /sys/firmware/acpi/tables
APIC  BGRT  data  DSDT  dynamic  FACP  FACS  HPET  MCFG  SSDT  WAET
student@student:~$ ls /sys/firmware/acpi/tables/SSDT/
ls: cannot access '/sys/firmware/acpi/tables/SSDT/': Not a directory
student@student:~$ sudo cat /sys/firmware/acpi/tables/SSDT > ssdt.bin
[sudo] password for student:
student@student:~$ ls
Desktop    Downloads  Pictures  snap      Templates
Documents  Music      Public    ssdt.bin  Videos
student@student:~$ iasl -d ssdt.bin
```



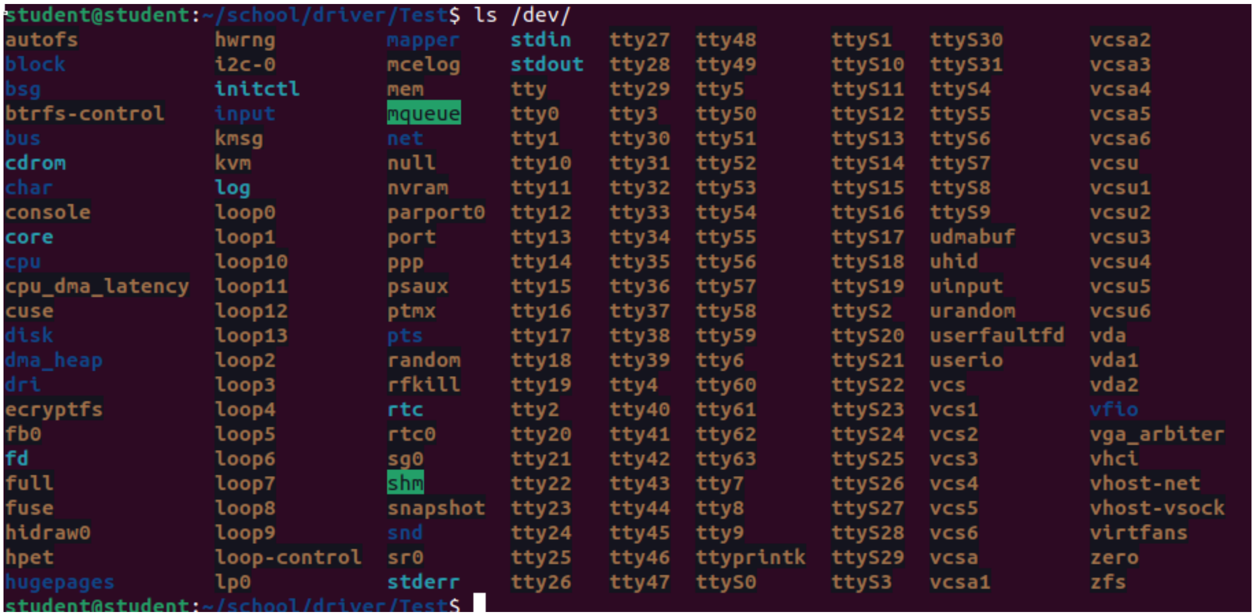
```
student@student:~$ iasl -d ssdt.bin

Intel ACPI Component Architecture
ASL+ Optimizing Compiler/Disassembler version 20200925
Copyright (c) 2000 - 2020 Intel Corporation

File appears to be binary: found 131 non-ASCII characters, disassembling
Binary file appears to be a valid ACPI table, disassembling
Input file ssdt.bin, Length 0x1EC (492) bytes
ACPI: SSDT 0x0000000000000000 0001EC (v02 VIRTIO FAKEFANS 00000000 INTL 20200925
)
Pass 1 parse of [SSDT]
Pass 2 parse of [SSDT]
Parsing Deferred Opcodes (Methods/Buffers/Packages/Regions)

Parsing completed
Disassembly completed
ASL Output:    ssdt.dsl - 4597 bytes
student@student:~$ 
```

Making the Device Driver

First successful device load, and we can see it live in /dev/!



```
[  122.183790] virtfan: init function entered
[  122.183793] virtfan: load function entered
[  122.183800] virtfan: load function entered
[  122.183802] virtfan: load function entered
[  122.183803] virtfan: load function entered
[  122.183884] virtfan loaded
```



```
student@student:~/school/driver/Test$ ls /dev/
autofs           hwrng         mapper    stdin   tty27  tty48      ttyS1   ttyS30       vcsa2
block            i2c-0         mcelog    stdout  tty28  tty49      ttyS10  ttyS31       vcsa3
bsg              initctl       mem       tty     tty29  tty5       ttyS11  ttyS4        vcsa4
btrfs-control    input         mqueue    tty0    tty3   tty50      ttyS12  ttyS5        vcsa5
bus              kmsg          net       tty1    tty30  tty51      ttyS13  ttyS6        vcsa6
cdrom            kvm           null      tty10   tty31  tty52      ttyS14  ttyS7        vcsu
char             log           nvram     tty11   tty32  tty53      ttyS15  ttyS8        vcsu1
console          loop0         parport0  tty12   tty33  tty54      ttyS16  ttyS9        vcsu2
core             loop1         port      tty13   tty34  tty55      ttyS17  udmabuf      vcsu3
cpu              loop10        ppp       tty14   tty35  tty56      ttyS18  uhid         vcsu4
cpu_dma_latency  loop11        psaux     tty15   tty36  tty57      ttyS19  uinput       vcsu5
cuse             loop12        ptmx      tty16   tty37  tty58      ttyS2   urandom      vcsu6
disk             loop13        pts       tty17   tty38  tty59      ttyS20  userfaultfd  vda
dma_heap         loop2         random    tty18   tty39  tty6       ttyS21  userio       vda1
dri              loop3         rfkill    tty19   tty4   tty60      ttyS22  vcs          vda2
ecryptfs         loop4         rtc       tty2    tty40  tty61      ttyS23  vcs1         vfio
fb0              loop5         rtc0      tty20   tty41  tty62      ttyS24  vcs2         vga_arbiter
fd               loop6         sg0       tty21   tty42  tty63      ttyS25  vcs3         vhci
full             loop7         shm       tty22   tty43  tty7       ttyS26  vcs4         vhost-net
fuse             loop8         snapshot  tty23   tty44  tty8       ttyS27  vcs5         vhost-vsock
hidraw0          loop9         snd       tty24   tty45  tty9       ttyS28  vcs6         virtfans
hpet             loop-control  sr0       tty25   tty46  ttyprintk  ttyS29  vcsa         zero
hugepages        lp0           stderr    tty26   tty47  ttyS0      ttyS3   vcsa1        zfs
student@student:~/school/driver/Test$ 
```

Incorrect scope

My device has scope “`\\_SB.FAN#`”, where # is the fan number. Well, when I was call `snprintf(...)`, I was only calling “`\\_SB.FAN#`”. This is because I thought the scope resolution was “`\_SB.FAN#`”, but this is incorrect. It needs another slash up front to account for root. Changing it to “`\\\\_SB.FAN#`” resolved this error.

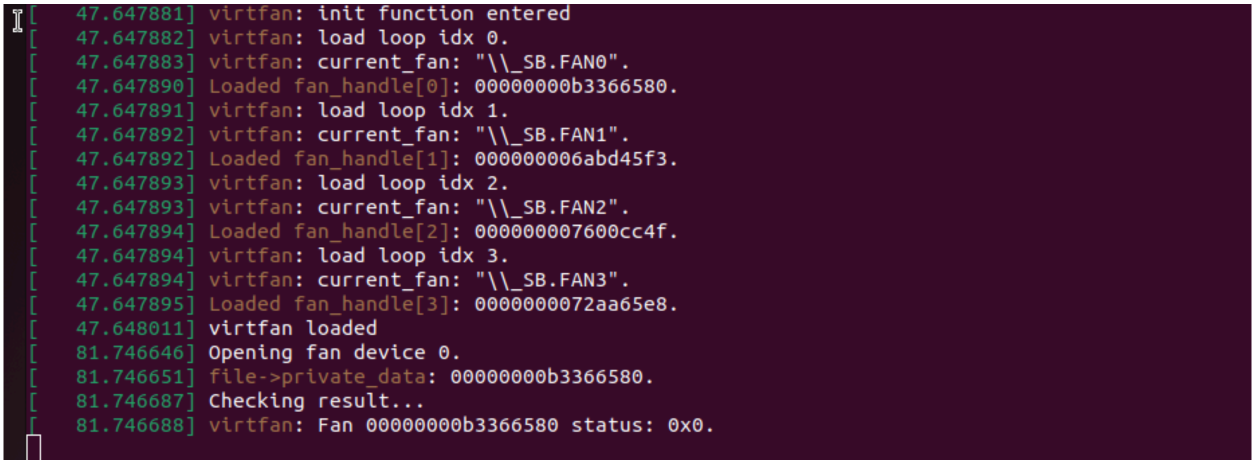
Incorrect size after fixing scope

I had forgotten that I had just added a character to a fix-size array. The resolution was to adjust the max size of my array.

AE_NOT_FOUND

Then, we were getting `AE_NOT_FOUND`, because I had mistakenly typed “`_PSC`” for the method call, rather than “`_PCS`”. Fixing the typo resolved that.

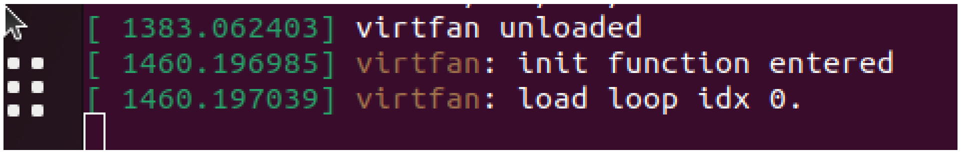
Finally reading!



```
[   47.647881] virtfan: init function entered
[   47.647882] virtfan: load loop idx 0.
[   47.647883] virtfan: current_fan: "\\_SB.FAN0".
[   47.647890] Loaded fan_handle[0]: 00000000b3366580.
[   47.647891] virtfan: load loop idx 1.
[   47.647892] virtfan: current_fan: "\\_SB.FAN1".
[   47.647892] Loaded fan_handle[1]: 000000006abd45f3.
[   47.647893] virtfan: load loop idx 2.
[   47.647893] virtfan: current_fan: "\\_SB.FAN2".
[   47.647894] Loaded fan_handle[2]: 000000007600cc4f.
[   47.647894] virtfan: load loop idx 3.
[   47.647894] virtfan: current_fan: "\\_SB.FAN3".
[   47.647895] Loaded fan_handle[3]: 0000000072aa65e8.
[   47.648011] virtfan loaded
[   81.746646] Opening fan device 0.
[   81.746651] file->private_data: 00000000b3366580.
[   81.746687] Checking result...
[   81.746688] virtfan: Fan 00000000b3366580 status: 0x0.
```

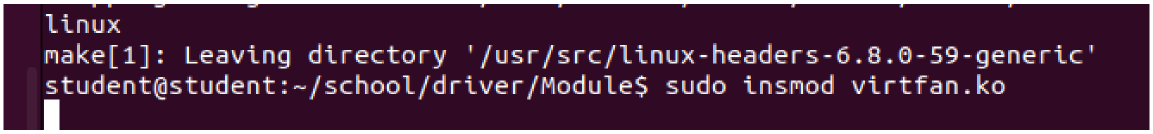
Unload and then load again, hangs, have to do full power off and reboot

Unload and then load again:



```
[ 1383.062403] virtfan unloaded
[ 1460.196985] virtfan: init function entered
[ 1460.197039] virtfan: load loop idx 0.
```

Hanging input:

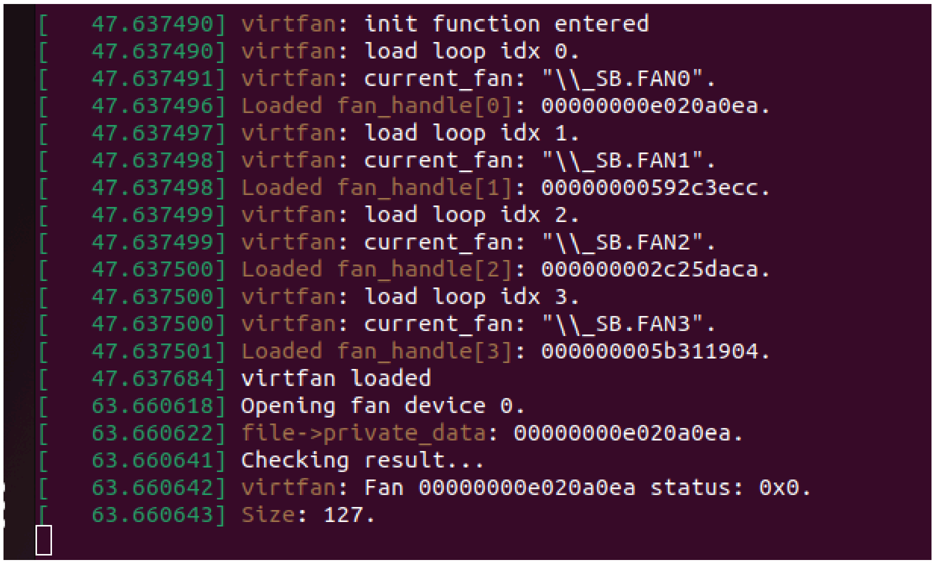


```
linux
make[1]: Leaving directory '/usr/src/linux-headers-6.8.0-59-generic'
student@student:~/school/driver/Module$ sudo insmod virtfan.ko
```

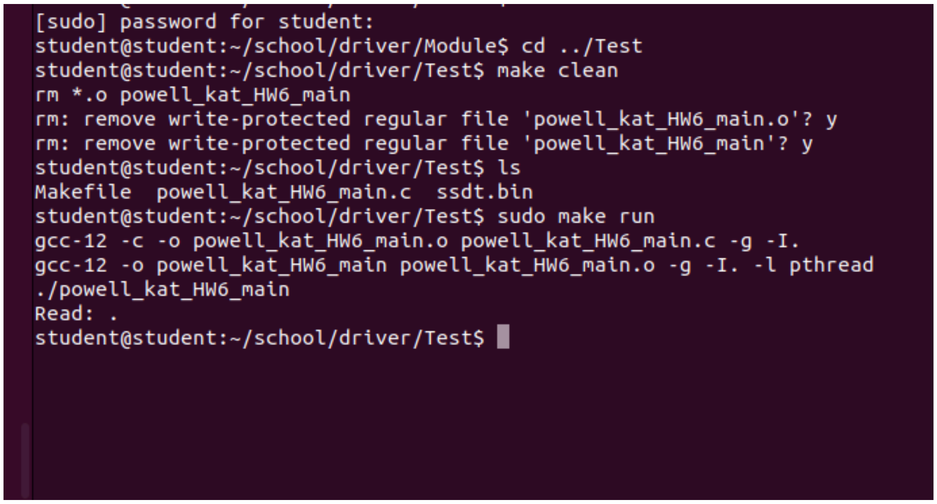
My new destructor routine for `fan_handles` was incorrectly iterating through the `fan_handles` and freeing them at each index. This is incorrect behavior: we only need to free the main structure. There is not allocated memory at each index. The resolution was to change my destructor routine to only free the top level `fan_handle` structure.

Incorrect success message

"Successful" read, but `copy_to_user` not working – was actually that I was copying into a `char *buf` instead of `uint64_t`! Resolution was to change the `char *buf` to `uint64_t`.

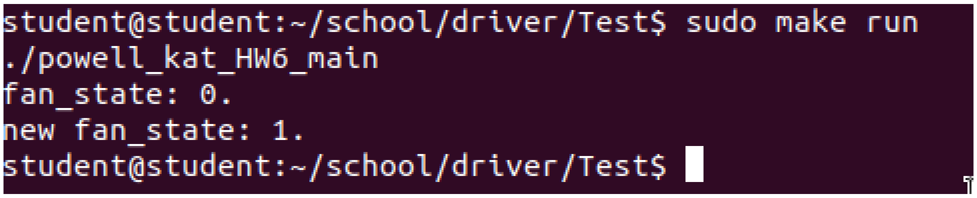


```
[   47.637490] virtfan: init function entered
[   47.637490] virtfan: load loop idx 0.
[   47.637491] virtfan: current_fan: "\\_SB.FAN0".
[   47.637496] Loaded fan_handle[0]: 00000000e020a0ea.
[   47.637497] virtfan: load loop idx 1.
[   47.637498] virtfan: current_fan: "\\_SB.FAN1".
[   47.637498] Loaded fan_handle[1]: 00000000592c3ecc.
[   47.637499] virtfan: load loop idx 2.
[   47.637499] virtfan: current_fan: "\\_SB.FAN2".
[   47.637500] Loaded fan_handle[2]: 000000002c25daca.
[   47.637500] virtfan: load loop idx 3.
[   47.637500] virtfan: current_fan: "\\_SB.FAN3".
[   47.637501] Loaded fan_handle[3]: 000000005b311904.
[   47.637684] virtfan loaded
[   63.660618] Opening fan device 0.
[   63.660622] file->private_data: 00000000e020a0ea.
[   63.660641] Checking result...
[   63.660642] virtfan: Fan 00000000e020a0ea status: 0x0.
[   63.660643] Size: 127.
```



```
[sudo] password for student:
student@student:~/school/driver/Module$ cd ../Test
student@student:~/school/driver/Test$ make clean
rm *.o powell_kat_HW6_main
rm: remove write-protected regular file 'powell_kat_HW6_main.o'? y
rm: remove write-protected regular file 'powell_kat_HW6_main'? y
student@student:~/school/driver/Test$ ls
Makefile  powell_kat_HW6_main.c  ssdt.bin
student@student:~/school/driver/Test$ sudo make run
gcc-12 -c -o powell_kat_HW6_main.o powell_kat_HW6_main.c -g -I.
gcc-12 -o powell_kat_HW6_main powell_kat_HW6_main.o -g -I. -l pthread
./powell_kat_HW6_main
Read: .
student@student:~/school/driver/Test$
```

Successful open, read and write functions!



```
student@student:~/school/driver/Test$ sudo make run
./powell_kat_HW6_main
fan_state: 0.
new fan_state: 1.
student@student:~/school/driver/Test$
```

Device tree changes

After I got read and write working, the next step was to get more functionality in my device tree for proper ioctl. I chose to implement:

_FPS	Fan Performance States, returns list of all available power states of the fan
_FST	Fan Status, returns a package of the current revision, control, and speed
_FSL	Fan Set Level, sets the Fan Performance State level

First, this required instantiating FPS packages. The structure of one is like so:

```
Package ()                 // Fan P-State
{
    Control,               // Integer DWORD
    TripPoint,             // Integer DWORD
    Speed,                 // Integer DWORD
    NoiseLevel,            // Integer DWORD
    Power                  // Integer DWORD
}
```

I made 5 pseudo packages with incremental increases in their performance states. I modelled it off `Noctua's NF-F12 IndustialPPC-2000PWM` model, which has a max RPM of 2000.

Now that we have performance states, we want a caller to be able to set them if so desired. `_FSL` takes an argument from the caller (the state they want), and applies that to `FLVL`.

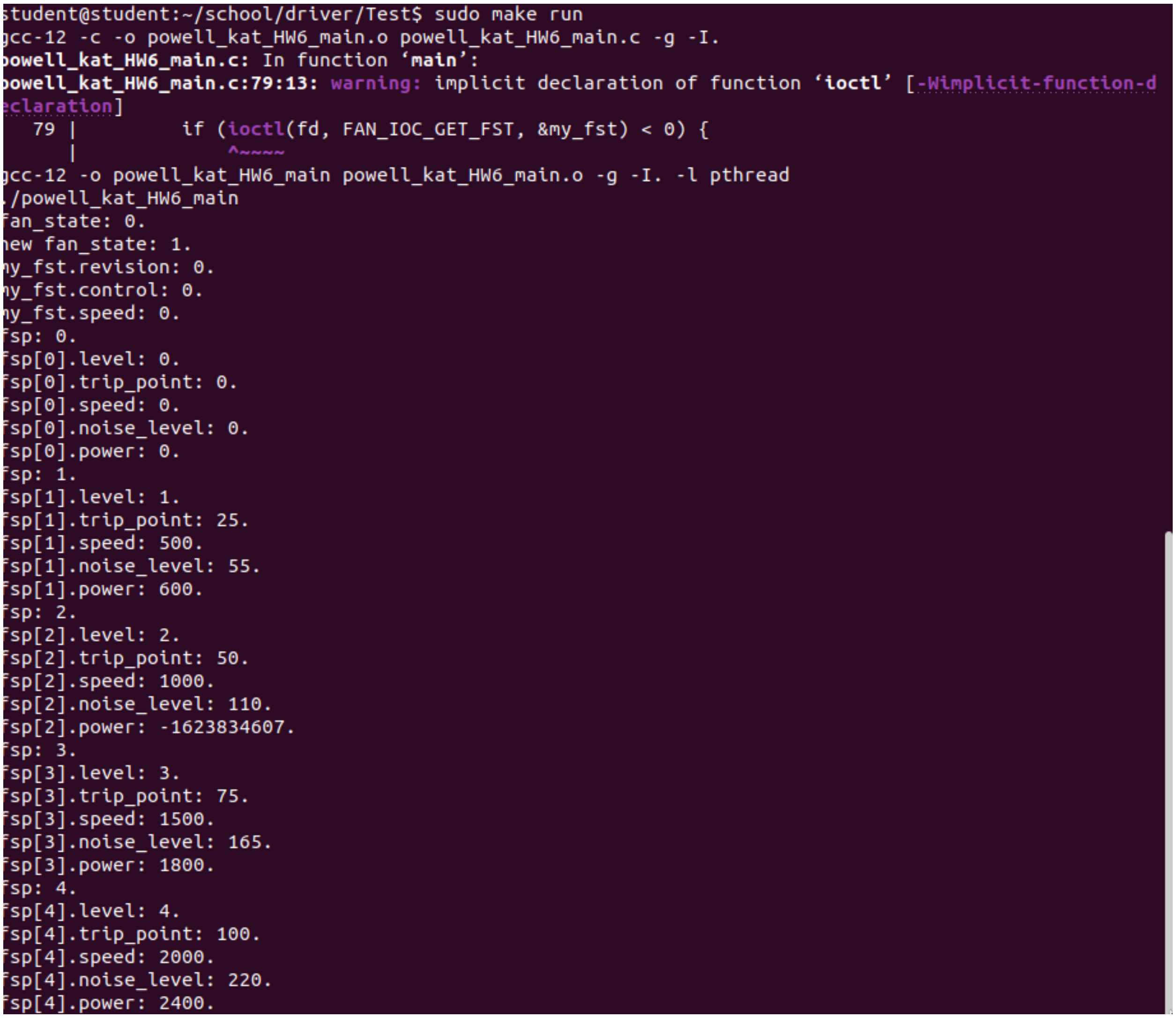
But, if a caller wants to set the state, won't they want to know what the available states are? So, I then implemented `_FPS`, which returns a package of the available `FPS` packages. But, this meant making that structure in the device. So, I had to implement an `FPSP` structure that points to each of the other packages.

Finally, what if a user wants to know the exact speed that the fan is currently on? `_FST` provides this to the user, so that was the last thing I had to implement. But, this required generating a package with only three parameters: revision, control, and speed. I could have simply hard coded it into the device namespace, but I wanted to dynamically build it if the user wants it. Now, this was slightly more difficult than I first expected. After reading multiple ASL tutorials, and facing cryptic compiler errors, I was surely confused. Well, I had yet to find this one piece of knowledge: if you are building a package inside a method, you need to first assign it to a local variable and then pass that local variable. You also need to define the local variable as a package variable (i.e., can't store a package inside a local variable that's not been assigned as a package).

After figuring that out, I finally had my new device tree compiling. Yay!

Debugging woes

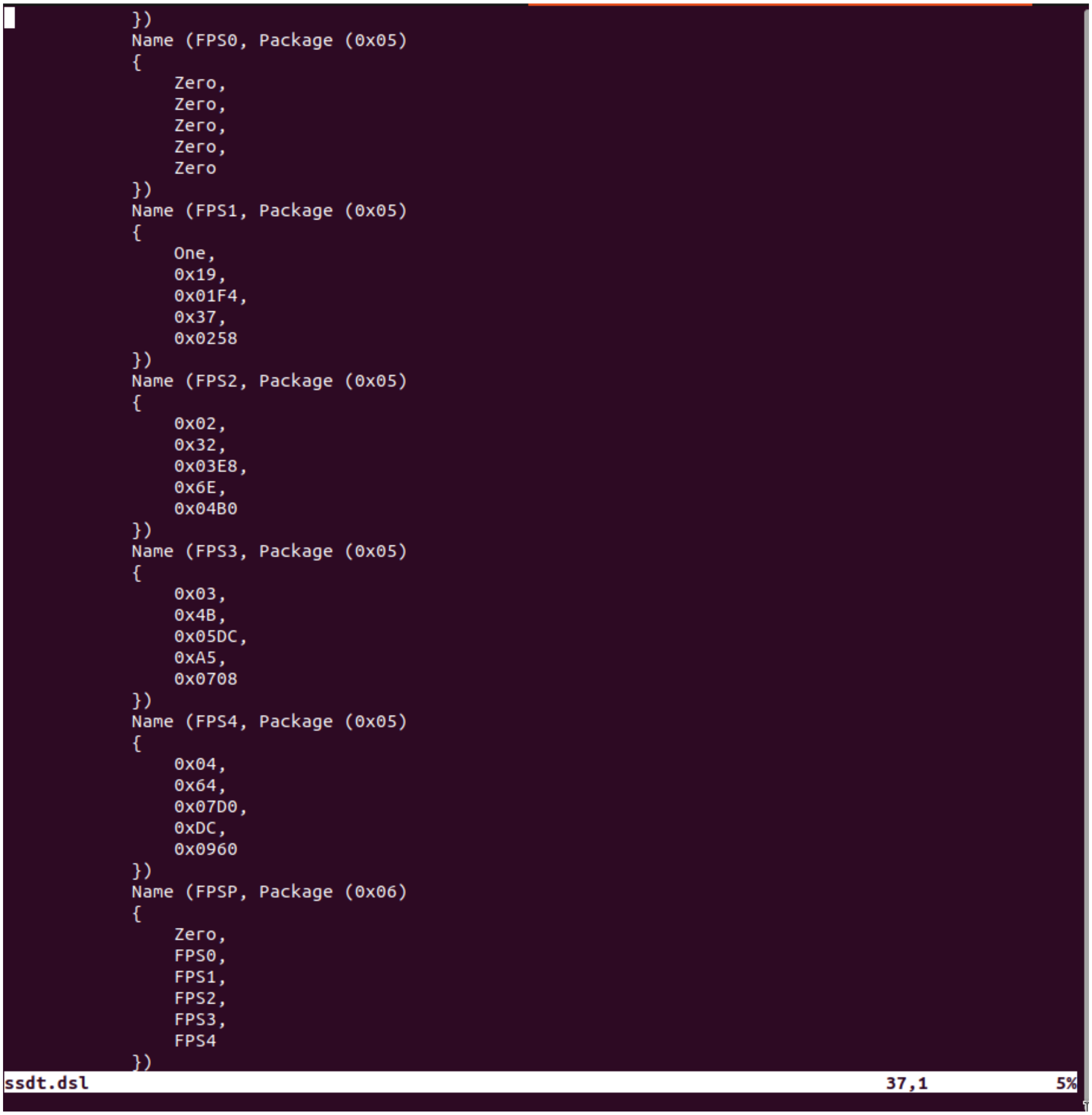
`fsp[2]` was printing a large, overflowing integer:



```
student@student:~/school/driver/Test$ sudo make run
gcc-12 -c -o powell_kat_HW6_main.o powell_kat_HW6_main.c -g -I.
powell_kat_HW6_main.c: In function ‘main’:
powell_kat_HW6_main.c:79:13: warning: implicit declaration of function ‘ioctl’ [-Wimplicit-function-d
eclaration]
   79 |         if (ioctl(fd, FAN_IOC_GET_FST, &my_fst) < 0) {
      |             ^~~~~
gcc-12 -o powell_kat_HW6_main powell_kat_HW6_main.o -g -I. -l pthread
./powell_kat_HW6_main
fan_state: 0.
new fan_state: 1.
my_fst.revision: 0.
my_fst.control: 0.
my_fst.speed: 0.
fsp: 0.
fsp[0].level: 0.
fsp[0].trip_point: 0.
fsp[0].speed: 0.
fsp[0].noise_level: 0.
fsp[0].power: 0.
fsp: 1.
fsp[1].level: 1.
fsp[1].trip_point: 25.
fsp[1].speed: 500.
fsp[1].noise_level: 55.
fsp[1].power: 600.
fsp: 2.
fsp[2].level: 2.
fsp[2].trip_point: 50.
fsp[2].speed: 1000.
fsp[2].noise_level: 110.
fsp[2].power: -1623834607.
fsp: 3.
fsp[3].level: 3.
fsp[3].trip_point: 75.
fsp[3].speed: 1500.
fsp[3].noise_level: 165.
fsp[3].power: 1800.
fsp: 4.
fsp[4].level: 4.
fsp[4].trip_point: 100.
fsp[4].speed: 2000.
fsp[4].noise_level: 220.
fsp[4].power: 2400.
```

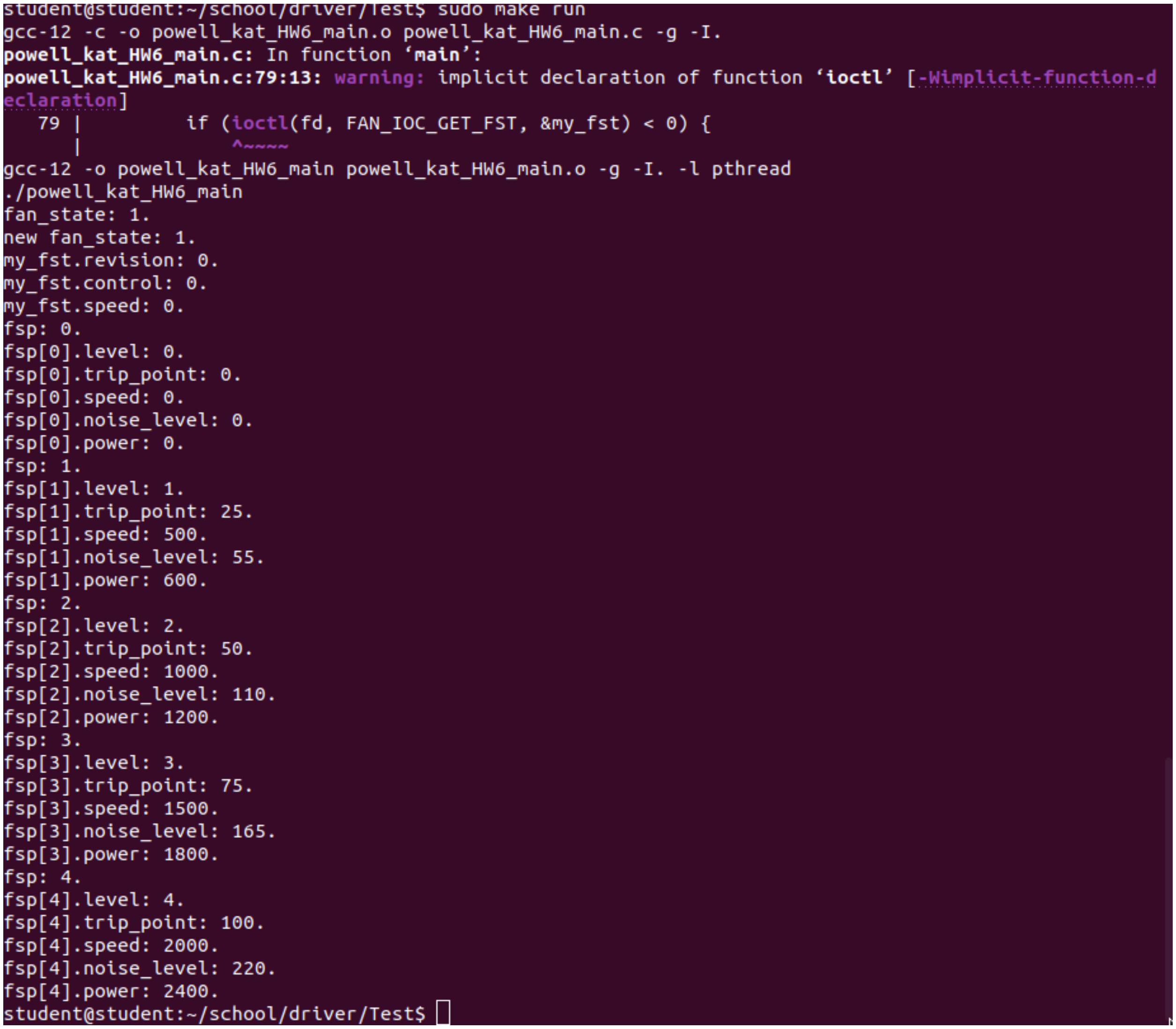
My immediate thought was that memory was corrupted somewhere. I first checked the `ACPI` SSDT tree to ensure it wasn’t corrupted. As we can see, `FPS2` still correctly has `0x04B0`

(1200).

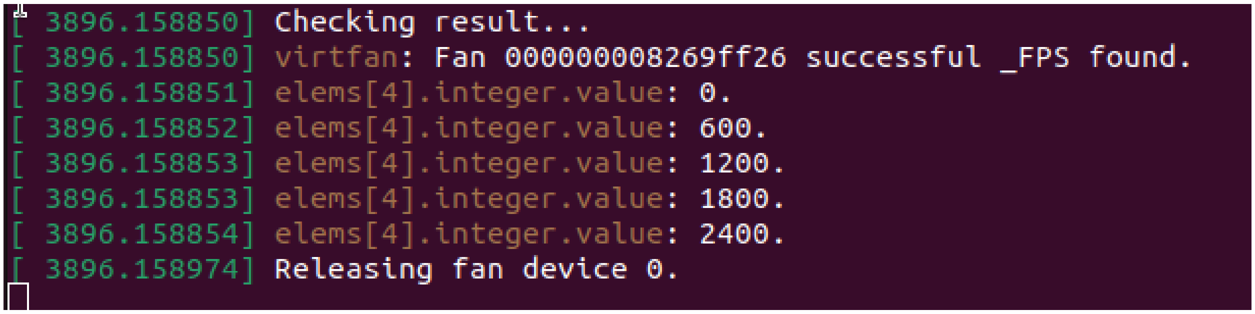


```
            })
            Name (FPS0, Package (0x05)
            {
                Zero,
                Zero,
                Zero,
                Zero,
                Zero
            })
            Name (FPS1, Package (0x05)
            {
                One,
                0x19,
                0x01F4,
                0x37,
                0x0258
            })
            Name (FPS2, Package (0x05)
            {
                0x02,
                0x32,
                0x03E8,
                0x6E,
                0x04B0
            })
            Name (FPS3, Package (0x05)
            {
                0x03,
                0x4B,
                0x05DC,
                0xA5,
                0x0708
            })
            Name (FPS4, Package (0x05)
            {
                0x04,
                0x64,
                0x07D0,
                0xDC,
                0x0960
            })
            Name (FPSP, Package (0x06)
            {
                Zero,
                FPS0,
                FPS1,
                FPS2,
                FPS3,
                FPS4
            })
ssdt.dsl                                                    37,1          5%
```

So, next I rebooted my VM, to make sure I didn't have weird garbage in my memory from previous issues. This did not fix it. Next, I checked both my module and my user test code at the same time. I made sure I had carbon-copies of their structures. I made sure both were indexing correctly at the right times. Then, I printed what that element was inside my ioctl function, and found that it was corrupted there first. This meant it was an issue with how I was handling memory inside my module. Starting from the top of my function, I scanned all potential memory leaks, and saw that I am using the same `acpi_buffer` structure for each ioctl call. Tracing each switch case, I found that I was freeing it too early in my `FAN_IOC_SET_FSPS` functions. The resolution was to move it to a later point (when I am actually done using it). This immediately fixed it:

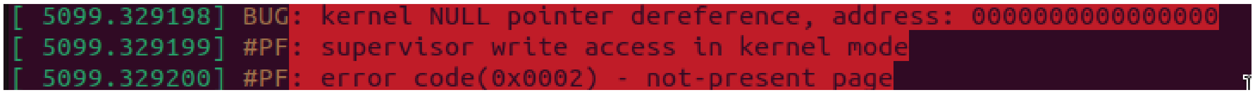


```
student@student:~/school/driver/Test$ sudo make run
gcc-12 -c -o powell_kat_HW6_main.o powell_kat_HW6_main.c -g -I.
powell_kat_HW6_main.c: In function ‘main’:
powell_kat_HW6_main.c:79:13: warning: implicit declaration of function ‘ioctl’ [-Wimplicit-function-d
eclaration]
   79 |         if (ioctl(fd, FAN_IOC_GET_FST, &my_fst) < 0) {
      |             ^~~~~
gcc-12 -o powell_kat_HW6_main powell_kat_HW6_main.o -g -I. -l pthread
./powell_kat_HW6_main
fan_state: 1.
new fan_state: 1.
my_fst.revision: 0.
my_fst.control: 0.
my_fst.speed: 0.
fsp: 0.
fsp[0].level: 0.
fsp[0].trip_point: 0.
fsp[0].speed: 0.
fsp[0].noise_level: 0.
fsp[0].power: 0.
fsp: 1.
fsp[1].level: 1.
fsp[1].trip_point: 25.
fsp[1].speed: 500.
fsp[1].noise_level: 55.
fsp[1].power: 600.
fsp: 2.
fsp[2].level: 2.
fsp[2].trip_point: 50.
fsp[2].speed: 1000.
fsp[2].noise_level: 110.
fsp[2].power: 1200.
fsp: 3.
fsp[3].level: 3.
fsp[3].trip_point: 75.
fsp[3].speed: 1500.
fsp[3].noise_level: 165.
fsp[3].power: 1800.
fsp: 4.
fsp[4].level: 4.
fsp[4].trip_point: 100.
fsp[4].speed: 2000.
fsp[4].noise_level: 220.
fsp[4].power: 2400.
student@student:~/school/driver/Test$
```



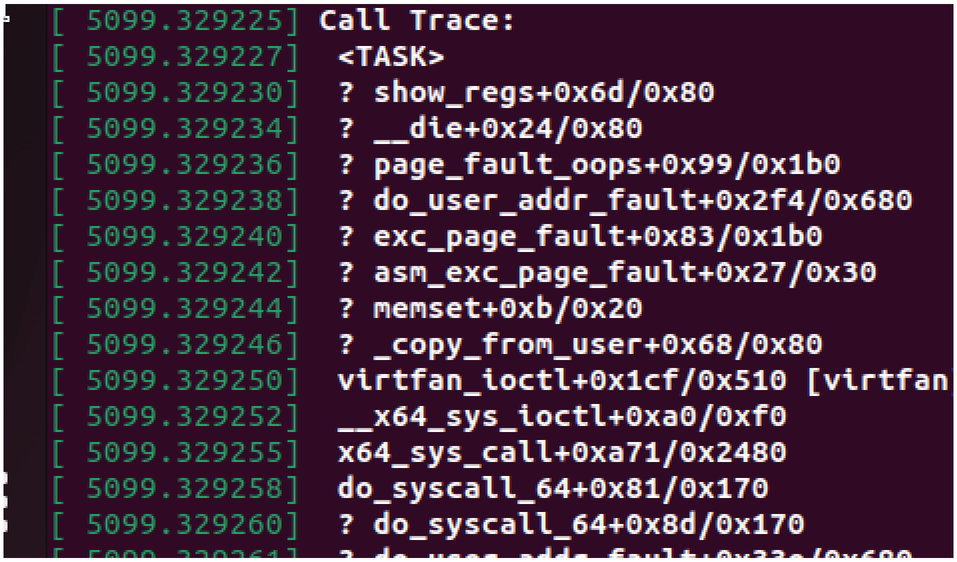
```
[ 3896.158850] Checking result...
[ 3896.158850] virtfan: Fan 000000008269ff26 successful _FPS found.
[ 3896.158851] elems[4].integer.value: 0.
[ 3896.158852] elems[4].integer.value: 600.
[ 3896.158853] elems[4].integer.value: 1200.
[ 3896.158853] elems[4].integer.value: 1800.
[ 3896.158854] elems[4].integer.value: 2400.
[ 3896.158974] Releasing fan device 0.
```

Next, I call FAN_IOC_SET_FSL. Uh-oh! A NULL pointer dereference in kernel memory…



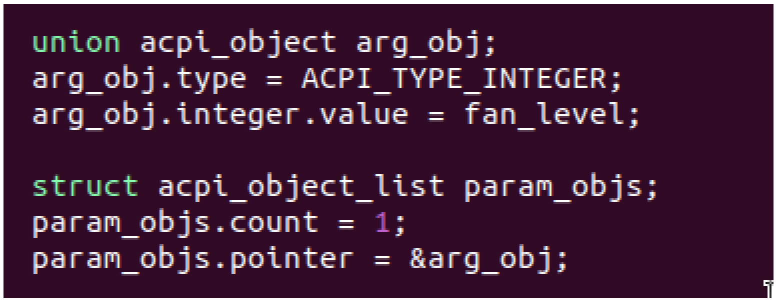
```
[ 5099.329198] BUG: kernel NULL pointer dereference, address: 0000000000000000
[ 5099.329199] #PF: supervisor write access in kernel mode
[ 5099.329200] #PF: error code(0x0002) - not-present page
```

To find out what's happening, we can see in our call trace that we are erroring after `copy_from_user`:



```
[ 5099.329225] Call Trace:
[ 5099.329227]  <TASK>
[ 5099.329230]  ? show_regs+0x6d/0x80
[ 5099.329234]  ? __die+0x24/0x80
[ 5099.329236]  ? page_fault_oops+0x99/0x1b0
[ 5099.329238]  ? do_user_addr_fault+0x2f4/0x680
[ 5099.329240]  ? exc_page_fault+0x83/0x1b0
[ 5099.329242]  ? asm_exc_page_fault+0x27/0x30
[ 5099.329244]  ? memset+0xb/0x20
[ 5099.329246]  ? _copy_from_user+0x68/0x80
[ 5099.329250]  virtfan_ioctl+0x1cf/0x510 [virtfan
[ 5099.329252]  __x64_sys_ioctl+0xa0/0xf0
[ 5099.329255]  x64_sys_call+0xa71/0x2480
[ 5099.329258]  do_syscall_64+0x81/0x170
[ 5099.329260]  ? do_syscall_64+0x8d/0x170
[ 5099.329261]  ? do_user_addr_fault+0x33e/0x680
```

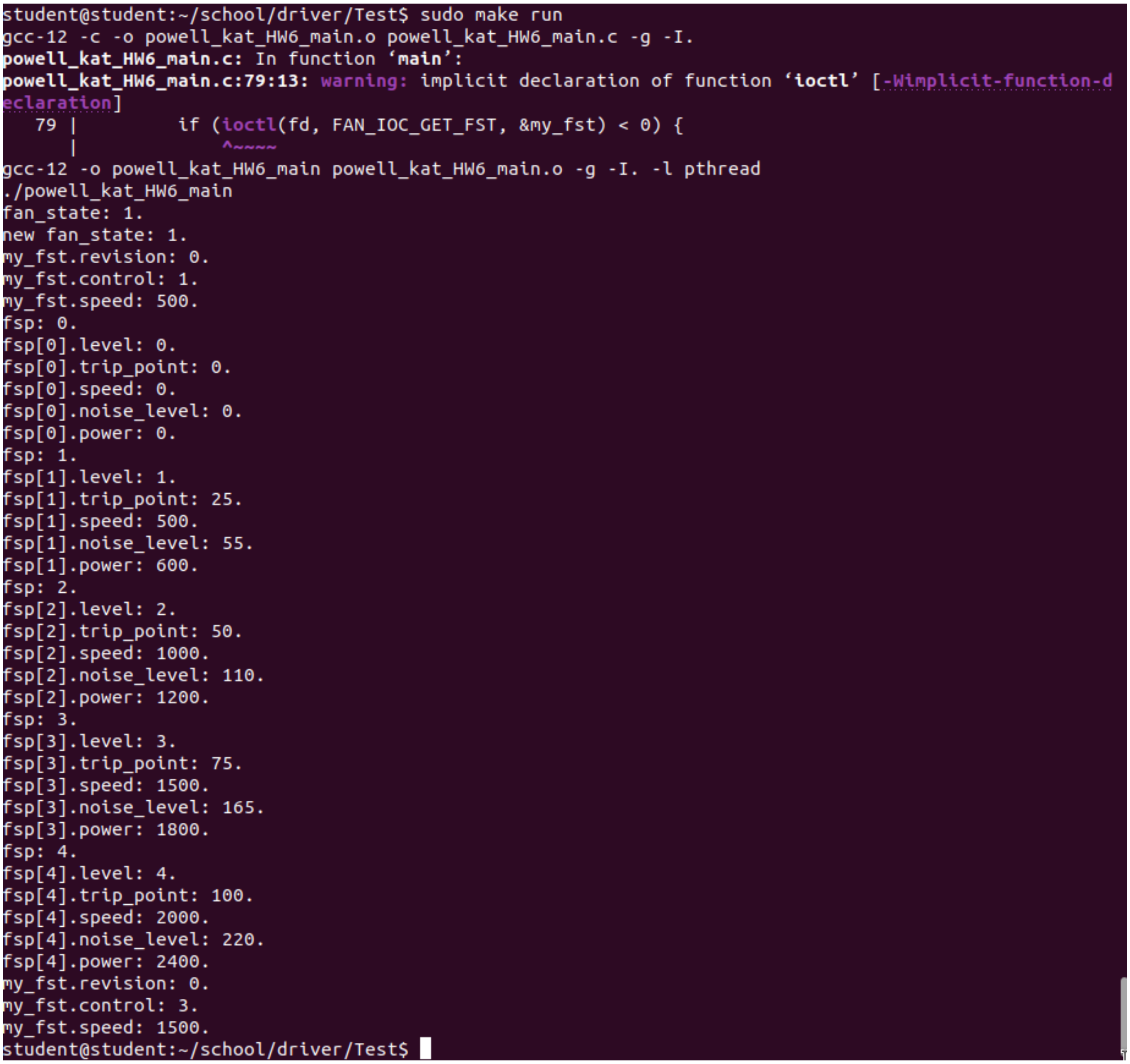
This was because I was incorrectly setting the acpi_object_list param_objs to just the address of the integer. Very wild mistake there. I needed to instantiate this way:



```c
union acpi_object arg_obj;
arg_obj.type = ACPI_TYPE_INTEGER;
arg_obj.integer.value = fan_level;

struct acpi_object_list param_objs;
param_objs.count = 1;
param_objs.pointer = &arg_obj;
```

And then pass in the address of `param_objs` to `acpi_evaluate_object()`.

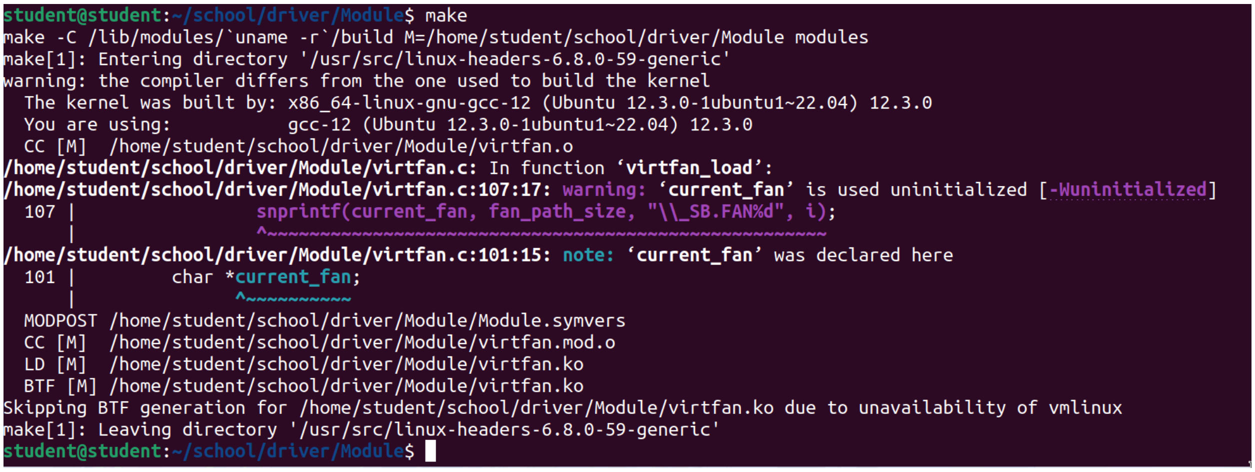
With this final fix, we now have all three ioctl functions working 😀



```
student@student:~/school/driver/Test$ sudo make run
gcc-12 -c -o powell_kat_HW6_main.o powell_kat_HW6_main.c -g -I.
powell_kat_HW6_main.c: In function ‘main’:
powell_kat_HW6_main.c:79:13: warning: implicit declaration of function ‘ioctl’ [-Wimplicit-function-d
eclaration]
   79 |         if (ioctl(fd, FAN_IOC_GET_FST, &my_fst) < 0) {
      |             ^~~~~
gcc-12 -o powell_kat_HW6_main powell_kat_HW6_main.o -g -I. -l pthread
./powell_kat_HW6_main
fan_state: 1.
new fan_state: 1.
my_fst.revision: 0.
my_fst.control: 1.
my_fst.speed: 500.
fsp: 0.
fsp[0].level: 0.
fsp[0].trip_point: 0.
fsp[0].speed: 0.
fsp[0].noise_level: 0.
fsp[0].power: 0.
fsp: 1.
fsp[1].level: 1.
fsp[1].trip_point: 25.
fsp[1].speed: 500.
fsp[1].noise_level: 55.
fsp[1].power: 600.
fsp: 2.
fsp[2].level: 2.
fsp[2].trip_point: 50.
fsp[2].speed: 1000.
fsp[2].noise_level: 110.
fsp[2].power: 1200.
fsp: 3.
fsp[3].level: 3.
fsp[3].trip_point: 75.
fsp[3].speed: 1500.
fsp[3].noise_level: 165.
fsp[3].power: 1800.
fsp: 4.
fsp[4].level: 4.
fsp[4].trip_point: 100.
fsp[4].speed: 2000.
fsp[4].noise_level: 220.
fsp[4].power: 2400.
my_fst.revision: 0.
my_fst.control: 3.
my_fst.speed: 1500.
student@student:~/school/driver/Test$ 
```

We see that we start with `my_fst.control` at `_FSL 1`. Later, we call `_FSL` and set it to 3. This is reflected at the bottom, where we need the `my_fst.control` is now 3, and `my_fst.speed` is 1500. This corresponds to our state of 1 and state of 3 respectively. Furthermore, we see that we are successfully grabbing our FSP tables, flattening them to memory, and passing them to userland. So, our `_FSL`, `_FST`, and FPSP ioctl functions successfully work.

Warnings

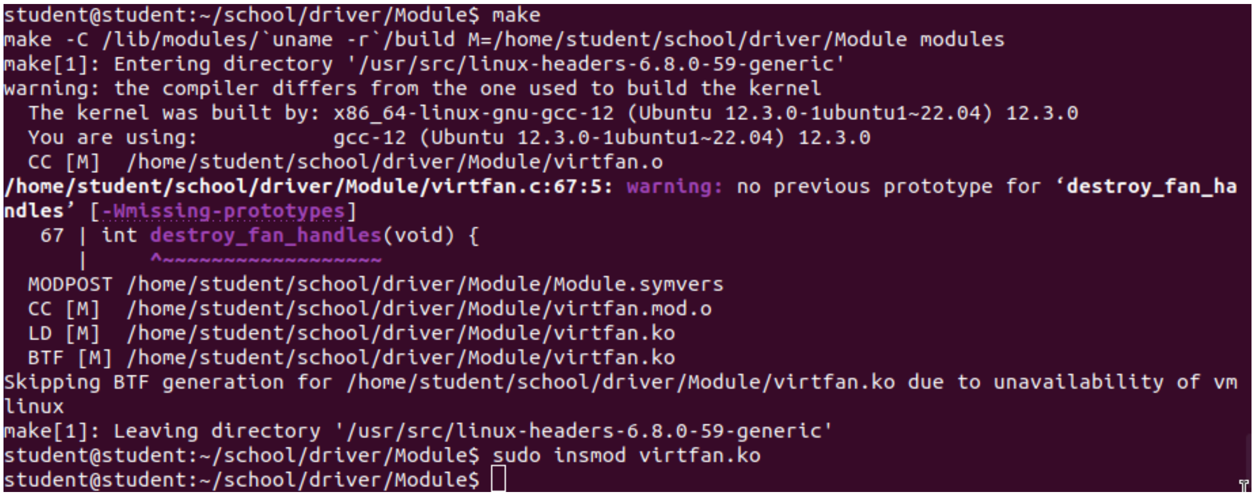


```
student@student:~/school/driver/Module$ make
make -C /lib/modules/`uname -r`/build M=/home/student/school/driver/Module modules
make[1]: Entering directory '/usr/src/linux-headers-6.8.0-59-generic'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: x86_64-linux-gnu-gcc-12 (Ubuntu 12.3.0-1ubuntu1~22.04) 12.3.0
  You are using:           gcc-12 (Ubuntu 12.3.0-1ubuntu1~22.04) 12.3.0
  CC [M]  /home/student/school/driver/Module/virtfan.o
/home/student/school/driver/Module/virtfan.c: In function ‘virtfan_load’:
/home/student/school/driver/Module/virtfan.c:107:17: warning: ‘current_fan’ is used uninitialized [-Wuninitialized]
  107 |                 snprintf(current_fan, fan_path_size, "\\_SB.FAN%d", i);
      |                 ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/home/student/school/driver/Module/virtfan.c:101:15: note: ‘current_fan’ was declared here
  101 |         char *current_fan;
      |               ^~~~~~~~~~~
  MODPOST /home/student/school/driver/Module/Module.symvers
  CC [M]  /home/student/school/driver/Module/virtfan.mod.o
  LD [M]  /home/student/school/driver/Module/virtfan.ko
  BTF [M] /home/student/school/driver/Module/virtfan.ko
Skipping BTF generation for /home/student/school/driver/Module/virtfan.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-6.8.0-59-generic'
student@student:~/school/driver/Module$
```

This warning can be ignored because it is a throw-away variable while calling acpi_get_handle().

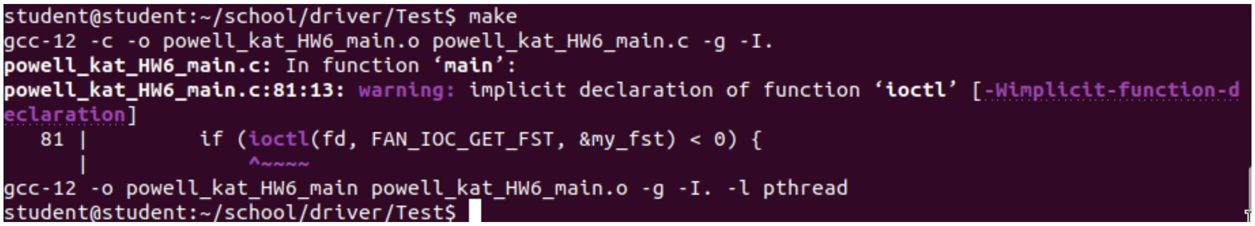
Screenshot of compilation:

Driver:



```
student@student:~/school/driver/Module$ make
make -C /lib/modules/`uname -r`/build M=/home/student/school/driver/Module modules
make[1]: Entering directory '/usr/src/linux-headers-6.8.0-59-generic'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: x86_64-linux-gnu-gcc-12 (Ubuntu 12.3.0-1ubuntu1~22.04) 12.3.0
  You are using:           gcc-12 (Ubuntu 12.3.0-1ubuntu1~22.04) 12.3.0
  CC [M]  /home/student/school/driver/Module/virtfan.o
/home/student/school/driver/Module/virtfan.c:67:5: warning: no previous prototype for ‘destroy_fan_ha
ndles’ [-Wmissing-prototypes]
   67 | int destroy_fan_handles(void) {
      |     ^~~~~~~~~~~~~~~~~~~
  MODPOST /home/student/school/driver/Module/Module.symvers
  CC [M]  /home/student/school/driver/Module/virtfan.mod.o
  LD [M]  /home/student/school/driver/Module/virtfan.ko
  BTF [M] /home/student/school/driver/Module/virtfan.ko
Skipping BTF generation for /home/student/school/driver/Module/virtfan.ko due to unavailability of vm
linux
make[1]: Leaving directory '/usr/src/linux-headers-6.8.0-59-generic'
student@student:~/school/driver/Module$ sudo insmod virtfan.ko
student@student:~/school/driver/Module$
```

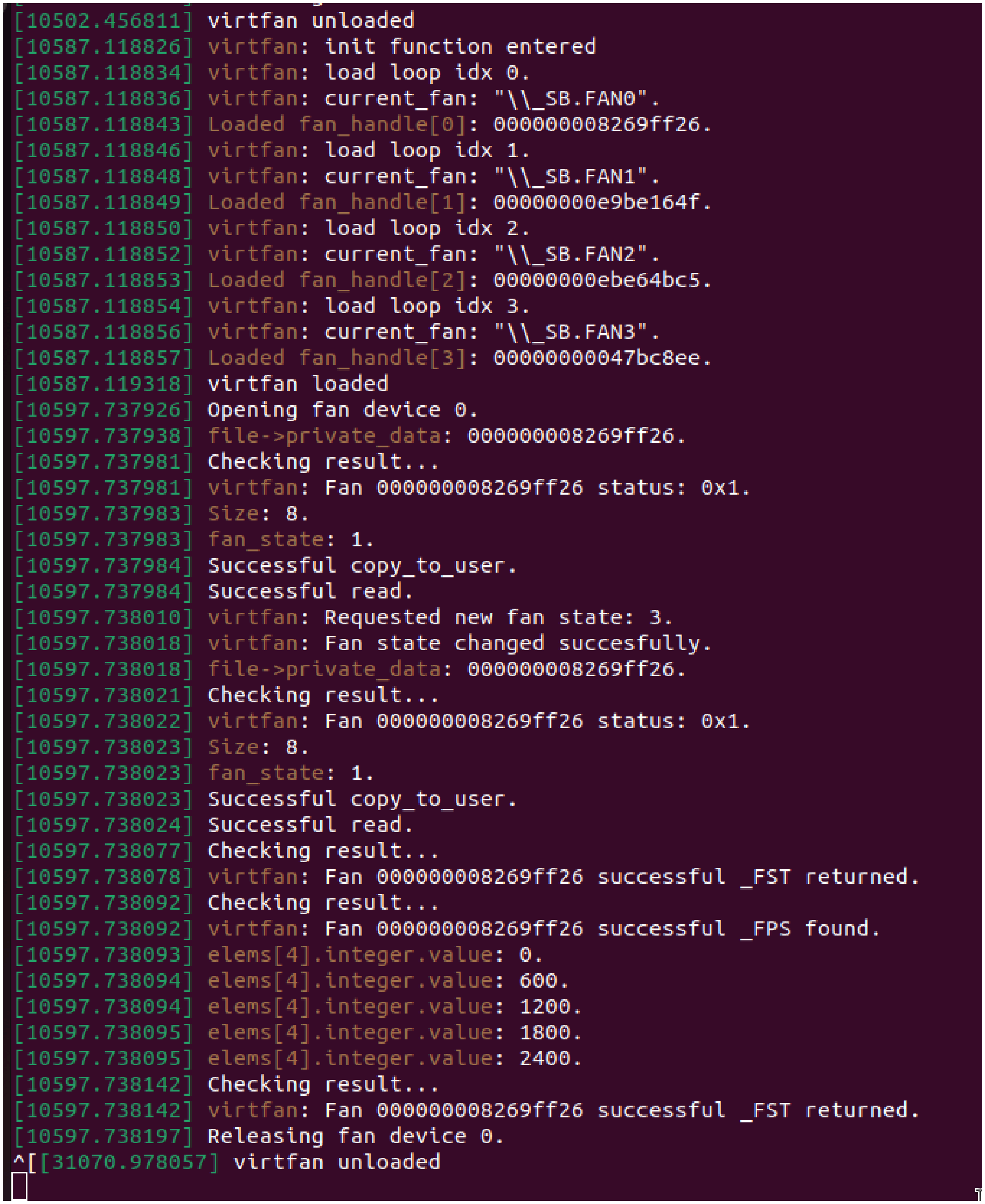
Test application:



```
student@student:~/school/driver/Test$ make
gcc-12 -c -o powell_kat_HW6_main.o powell_kat_HW6_main.c -g -I.
powell_kat_HW6_main.c: In function ‘main’:
powell_kat_HW6_main.c:81:13: warning: implicit declaration of function ‘ioctl’ [-Wimplicit-function-d
eclaration]
   81 |         if (ioctl(fd, FAN_IOC_GET_FST, &my_fst) < 0) {
      |             ^~~~~
gcc-12 -o powell_kat_HW6_main powell_kat_HW6_main.o -g -I. -l pthread
student@student:~/school/driver/Test$
```

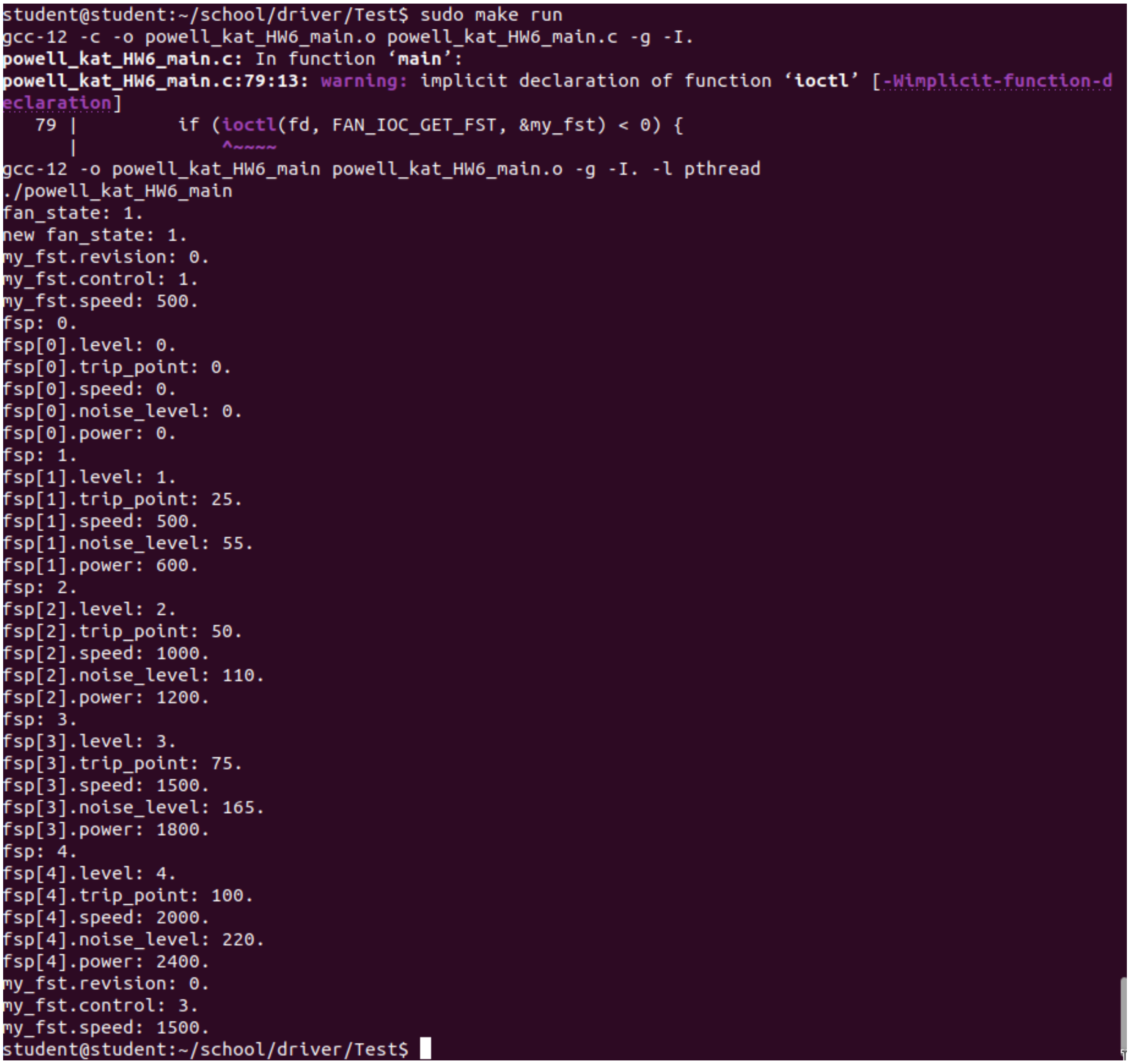
Screenshot(s) of the execution of the program:

Loading/unloading the driver:



```
[10502.456811] virtfan unloaded
[10587.118826] virtfan: init function entered
[10587.118834] virtfan: load loop idx 0.
[10587.118836] virtfan: current_fan: "\\_SB.FAN0".
[10587.118843] Loaded fan_handle[0]: 000000008269ff26.
[10587.118846] virtfan: load loop idx 1.
[10587.118848] virtfan: current_fan: "\\_SB.FAN1".
[10587.118849] Loaded fan_handle[1]: 00000000e9be164f.
[10587.118850] virtfan: load loop idx 2.
[10587.118852] virtfan: current_fan: "\\_SB.FAN2".
[10587.118853] Loaded fan_handle[2]: 00000000ebe64bc5.
[10587.118854] virtfan: load loop idx 3.
[10587.118856] virtfan: current_fan: "\\_SB.FAN3".
[10587.118857] Loaded fan_handle[3]: 00000000047bc8ee.
[10587.119318] virtfan loaded
[10597.737926] Opening fan device 0.
[10597.737938] file->private_data: 000000008269ff26.
[10597.737981] Checking result...
[10597.737981] virtfan: Fan 000000008269ff26 status: 0x1.
[10597.737983] Size: 8.
[10597.737983] fan_state: 1.
[10597.737984] Successful copy_to_user.
[10597.737984] Successful read.
[10597.738010] virtfan: Requested new fan state: 3.
[10597.738018] virtfan: Fan state changed succesfully.
[10597.738018] file->private_data: 000000008269ff26.
[10597.738021] Checking result...
[10597.738022] virtfan: Fan 000000008269ff26 status: 0x1.
[10597.738023] Size: 8.
[10597.738023] fan_state: 1.
[10597.738023] Successful copy_to_user.
[10597.738024] Successful read.
[10597.738077] Checking result...
[10597.738078] virtfan: Fan 000000008269ff26 successful _FST returned.
[10597.738092] Checking result...
[10597.738092] virtfan: Fan 000000008269ff26 successful _FPS found.
[10597.738093] elems[4].integer.value: 0.
[10597.738094] elems[4].integer.value: 600.
[10597.738094] elems[4].integer.value: 1200.
[10597.738095] elems[4].integer.value: 1800.
[10597.738095] elems[4].integer.value: 2400.
[10597.738142] Checking result...
[10597.738142] virtfan: Fan 000000008269ff26 successful _FST returned.
[10597.738197] Releasing fan device 0.
^[[31070.978057] virtfan unloaded
```

Running the test application:



```
student@student:~/school/driver/Test$ sudo make run
gcc-12 -c -o powell_kat_HW6_main.o powell_kat_HW6_main.c -g -I.
powell_kat_HW6_main.c: In function ‘main’:
powell_kat_HW6_main.c:79:13: warning: implicit declaration of function ‘ioctl’ [-Wimplicit-function-d
eclaration]
   79 |         if (ioctl(fd, FAN_IOC_GET_FST, &my_fst) < 0) {
      |             ^~~~~
gcc-12 -o powell_kat_HW6_main powell_kat_HW6_main.o -g -I. -l pthread
./powell_kat_HW6_main
fan_state: 1.
new fan_state: 1.
my_fst.revision: 0.
my_fst.control: 1.
my_fst.speed: 500.
fsp: 0.
fsp[0].level: 0.
fsp[0].trip_point: 0.
fsp[0].speed: 0.
fsp[0].noise_level: 0.
fsp[0].power: 0.
fsp: 1.
fsp[1].level: 1.
fsp[1].trip_point: 25.
fsp[1].speed: 500.
fsp[1].noise_level: 55.
fsp[1].power: 600.
fsp: 2.
fsp[2].level: 2.
fsp[2].trip_point: 50.
fsp[2].speed: 1000.
fsp[2].noise_level: 110.
fsp[2].power: 1200.
fsp: 3.
fsp[3].level: 3.
fsp[3].trip_point: 75.
fsp[3].speed: 1500.
fsp[3].noise_level: 165.
fsp[3].power: 1800.
fsp: 4.
fsp[4].level: 4.
fsp[4].trip_point: 100.
fsp[4].speed: 2000.
fsp[4].noise_level: 220.
fsp[4].power: 2400.
my_fst.revision: 0.
my_fst.control: 3.
my_fst.speed: 1500.
student@student:~/school/driver/Test$
```