Table of Contents

Table of Figures

1 Introduction

1.1. A brief introduction on computer systems and cooling systems

In the contemporary landscape of technology, computer systems have become indispensable tools for individuals, businesses, and organizations alike. These systems include a wide array of hardware and software components, working together to perform various tasks efficiently. At their core, computer systems consist of CPUs (Central Processing Units), memory modules, storage devices, I/O (input/output) peripherals, and networking components. The evolution of computer systems has been marked by advancements in processing power, memory capacity, and connectivity, leading to transformative changes across industries and societies.

Thermals play a crucial role in the performance and longevity of computer systems. The electronic components inside a computer generate heat due to the flow of electrical currents. All these computer parts need a way to dissipate that heat through active or passive solutions, otherwise they will throttle, or even worse, get damaged.

1.2 Motives for using additional cooling

The problem with new computer systems is that manufacturers have begun to add more and more cores, increased the frequency of the processor units and this affects the performance of the computer in the long term. The heat of the components reduces the lifespan of the computer accentuated and robs the user from extracting all the performance from his components. This is accentuated by modern computer cases that became larger and more restrictive than previous designs. Also, the current thermal management of computer systems is limited by the hardware that’s implemented by the motherboard manufactures.

A trend that can be seen on the market is that the motherboards are getting even more expensive without giving enough features for the current user. Motherboards are coming with less connectors for PCI-E, SATA, RAM, USB interfaces and the number of fan headers to connect computers fans are coming to a very low number. The majority of affordable motherboards are only coming with two or three fan headers, of which one is exclusively for the processor. So, the actual number of usable number of headers is one or two, and that’s not enough for the modern airflow restrictive current computer cases. Modern computer cases are also getting into a trend of being surrounded by glass. This trend is esthetically pleasing, but it restricts the air supply for the components.

The combination of increasing TDP of CPUs, reduced numbers of computer fan ports and newly built computer cases are straining the thermal capacity of the system to be capable of sustaining a continuous load.

1.3 Objectives of the paper

The concept behind this whole idea is to have an alternative solution for a user to add more cooling performance to a computer. The system is a separate system that's managed by the Linux kernel module. This module entails an expansion of the critical function of cooling to enhance performance and scalability. Adopting this type of system offers thermal headroom to the computer system. This system also has a benefit for the computer builder, by giving him more precision and flexibility to implement the best thermal and acoustic configuration of airflow.

1.4 Structure of the paper

The project scope is to create an inexpensive hub that's connected to the motherboard and has the capability to individually control for each computer fan connected to the hub. The document can be summarized into 9 chapters:

- Introduction
- State of the art
- Requirement analysis
- Design
- Implementation
- Validation
- Contributions to the field of computer cooling
- Conclusion
- Further Work

2 State of the art

2.1 State of the Art

Since the introduction of the integrated circuits in the 1940s, the integrated circuit have evolved very fast in the last century. A chip, also called integrated circuit, is a small object constructed of semiconductor material that contains transistors, capacitors and resistors. The most common material for these chips is silicon and it has physical properties between an insulator and a conductor.

In the 1960's, the Moore's Law [1] effect was proposed (Fig. 2.1). It is important to note that this law is not a physical or natural law, but rather an observation of historical trends in semiconductor manufacturing. At the same time, the market increased its demand of electronic devices and moved towards decreasing the size of such devices. However, the miniaturization of electronic devices pushed to higher heat flux density.

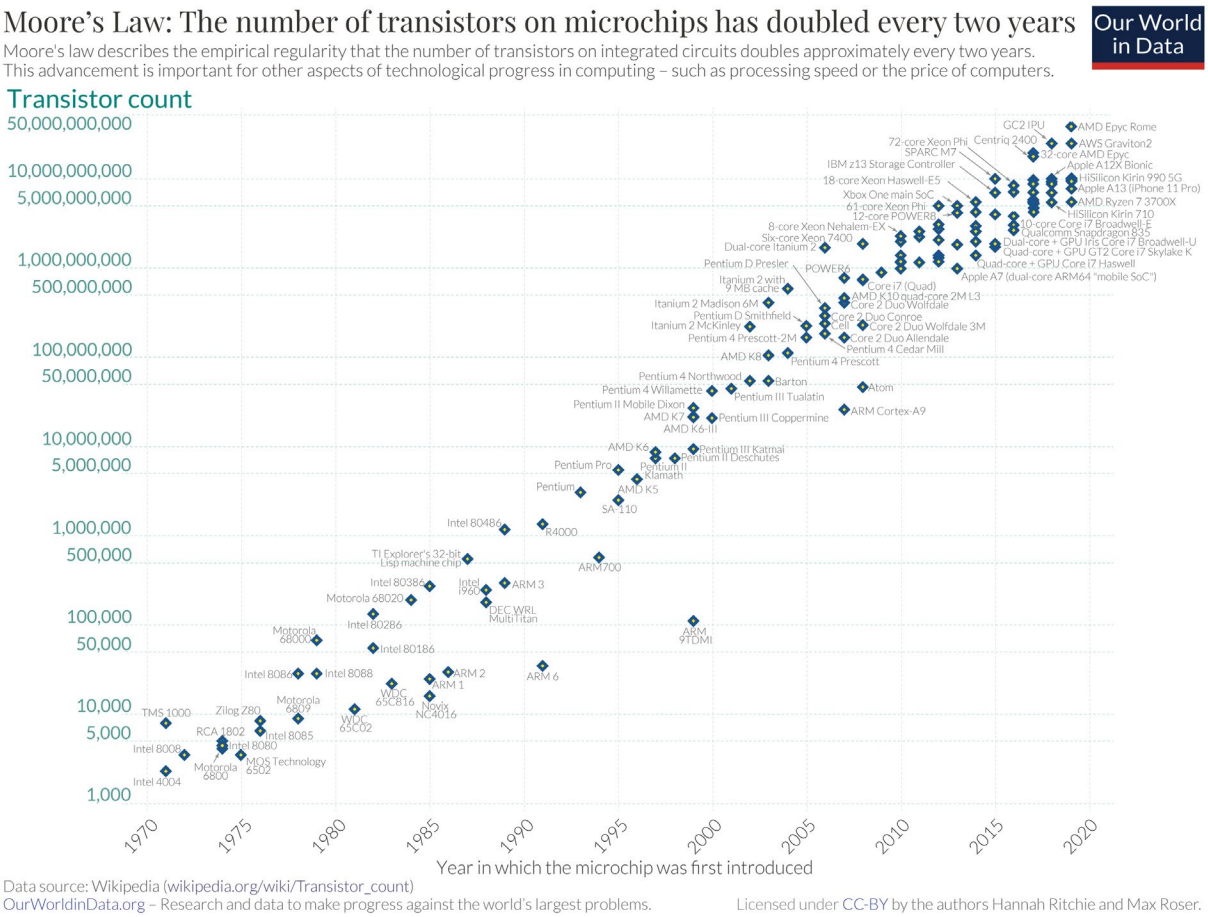

Fig. 2.1 – Evolution of Moore's Law [2]

2.1.1 Computer Cooling Systems

As mentioned before, the continuous improvement of manufacturing increased the heat generation of integrated chips. Consequently, the computer hardware manufacturers increased the investment and complexity of thermal solutions for the desktop market. In the early 1990s, early thermal solutions relied on aluminum heatsinks (Fig. 2.2) to dissipate CPU heat. Newer solutions added a large copper slug surrounded by aluminum fins (Fig. 2.3) with a computer fan on top. The next generation of computer coolers used copper heat pipes (Fig. 2.4) to quickly transfer heat energy from the CPU and spread it over an aluminum fin stacks. This allows for higher surface area and combined with forced air cooling gives the best cost-efficient solution for modern CPUs, GPUs and NPUs. All these improvements in air-cooling can be associated the increased thermal density of products from chip manufacturers. [3]

Fig. 2.2 – Aluminum heatsink

Fig. 2.3 – Aluminum heatsink with a copper slug

Fig. 2.4 – Aluminum fin stack with cooper heat pipes

The other method of cooling computer chips is represented by passing liquids that absorb heat through a copper base plate with slits on top (Fig. 2.5). This liquid is then pumped into a radiator (Fig. 2.6). The radiator that works exactly like in a vehicle, hot coolant flows through a network of narrow tubes, which are surrounded by metal fins. These fins get heated and radiate into the surrounding air. [4]

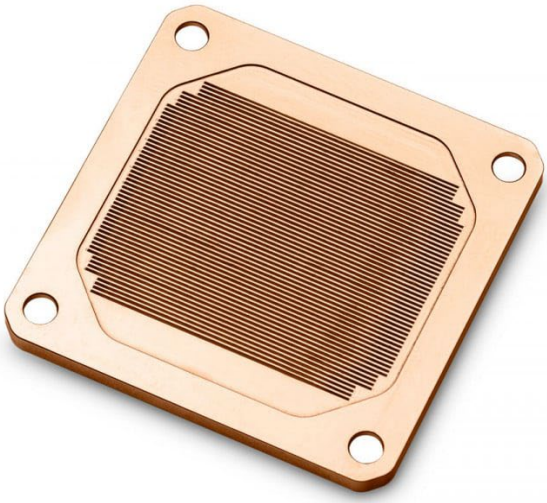

Fig. 2.5 – Cooper base plate

Fig. 2.6 – Aluminum radiator

In short, thermal solutions can be categorized into two parts: coolers that rely on air to pass through fin stacks and coolers that rely on liquid medium – like water – to transmit away heat from the CPU. All of these thermal solutions require forced airflow through metal surfaces. It becomes obvious that the computer needs access to fresh cool air to keep it as cool as possible.

So, fans are required to get air from outside the computer case to replace the inside volume of air.

Computer motherboard manufactures have offered less and less features as time progressed and this can be observed in the number of internal and external USB ports, computer fan ports, SATA ports and PCI-E ports. These new technologies are to the detriment of future upgradability of the platform. This effect can be seen in the table 2.1 from below.

Name of product	Gigabyte GA-F2A88X-D3H	ASUS PRIME A520M-A II	ASUS PRIME B760-PLUS D4
No. USB Ports (internal and external connectors)	12	12	13
No. Fan ports	4	3	5
No. SATA ports	8	4	4
No. PCI-E ports	7	3	4
Launch date	September 27, 2013	July 1, 2021	January 3, 2023
Price	90 $ as of January 2017 on newegg.com	104 $ as of June 2024 on newegg.com	260 $ as of June 2024 on newegg.com

Table 2.1 – Comparison between older and newer motherboards

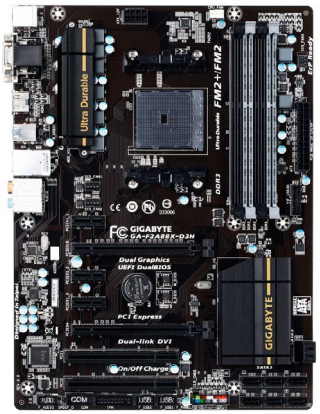

Fig. 2.7 – Gigabyte GA-F2A88X-D3H

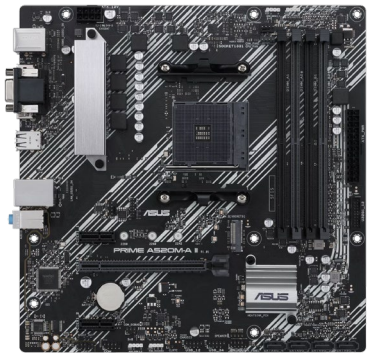

Fig. 2.8 – ASUS PRIME A520M-A II

Fig. 2.9 – ASUS PRIME B760-PLUS D4

If it is accounted for the inflation of the dollar from 2017 to 2024, the price of the first presented motherboard is in the price range of the second motherboard. It can be seen that the number of connectors has decreased and is required more upfront investment from the user to get same feature set as seven years ago.

2.1.2 Computer case airflow and the physics behind it [4]

Airflow is a fluid flow respecting the fluid dynamics' laws. These fluid laws are a summary of principle that apply in the real world, like Newton's law and conservation of energy. In physics, air is considered a fluid, and so, fluid dynamics' law applies to it. An important principle for fluid dynamics is Bernoulli's principle [5], which states that potential energy of a fluid decreases and its speed increases. This applies also in reverse. This principle is relevant because air moves from high pressure to low pressure.

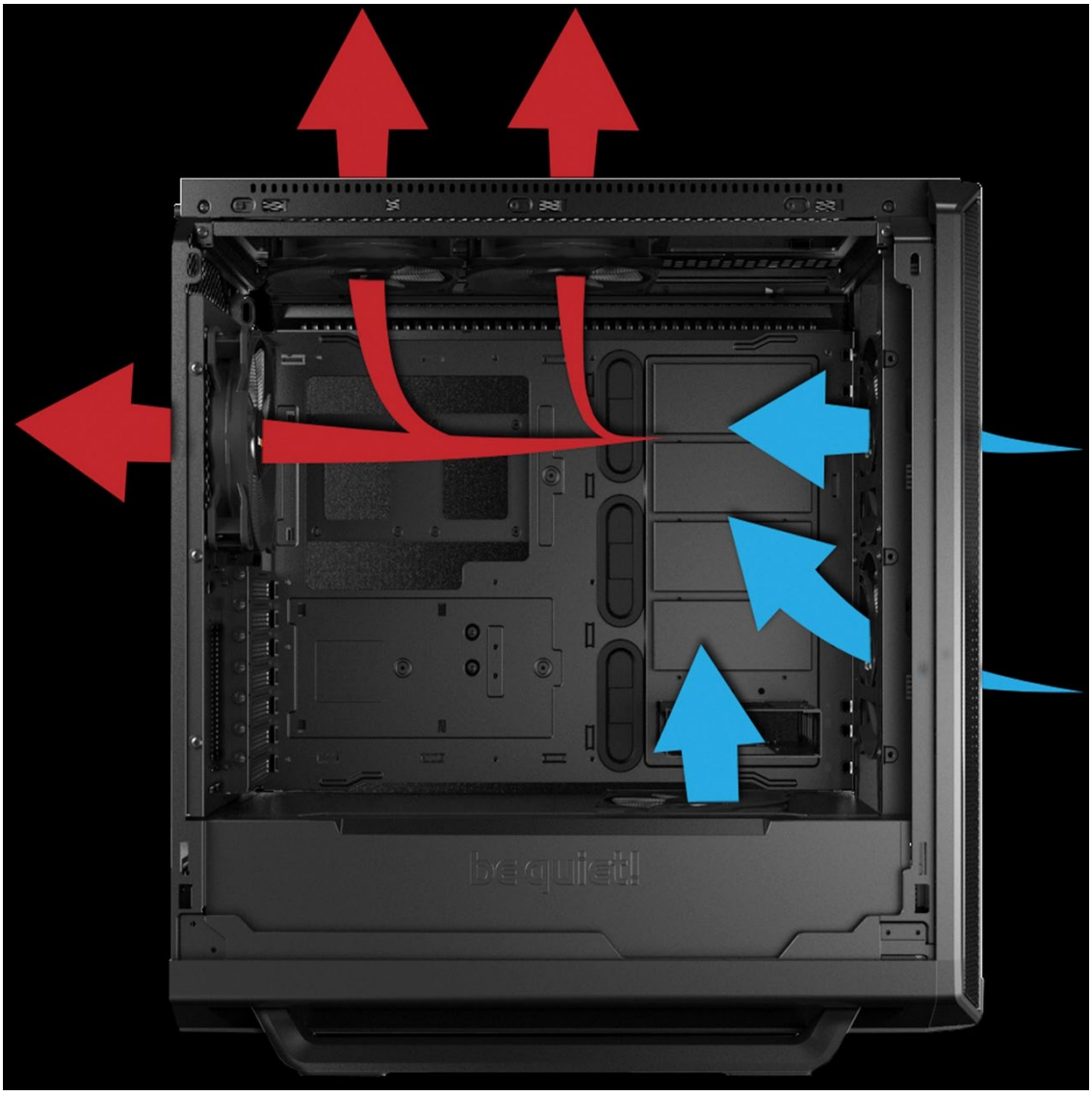

Fig. 2.10 – Conventional set-up of airflow in a computer case [6]

Computer airflow can be categorized as positive pressure system and negative pressure system. A positive pressure system is a system in which input fans enter more air than output air can exhaust. If fans have the same CMM (Cubic Meters per Minute) rating, a computer system will have a positive air pressure if the number of input fans is bigger than output fans. In a negative pressure system, output computer fans are in a larger quantity than input computer fans. [7]

The idea of having more computer fans that blow air inside a computer case is not a new concept for PC builders. This positive pressure is achieved through pushing more air inside the case and this results in a higher density of air. The positive pressure system pushes all air from within the case to escape through all the holes of the computer case.

At the opposite end, a system with negative pressure will pull air from around the case into each hole of it. This results in a lower density of air in the computer case and affects negatively the temperature of computer's components. So, if Bernoulli principles are applied in this case, the higher density air from around the case is sucked in the lower density space that is the computer.

The preferred scenario is the one in which the computer system has a positive pressure, due to the fact that particles and dust problem can be managed. This problem also requires that the computer case is compatible with air filters to catch dust. In this case, the system is pulling cool air towards its inside fans and output fans manage to redirect inside air through the computer case.

In a computer without fans, heat will rise up because of its lower density and is will not manage to cool itself just with the created convection flow. The convection flow can be easily overmatched by the power of computer fans. The direction of airflow will push heat in the same course as the fans push air. [8]

In addition, colder air requires more power from fans to push than warmer air. This effect happens because cool air is more dense than warm air for a fixed volume. When air heats up, the air expands and particles spread out from the its vibration.

The positive pressure and negative pressure systems are very similar. The reason for this is cases are large and full of holes that require powerful computer fans to overcome. It's important that computer are supplied with air that is cool, with a high density and dust-free.

2.1.3 Linux Kernel and Kernel Modules

Linux is an open-source operating system started by Linus Torvalds in the 1990s. At that time, this Finnish computer science student took on as a personal project to create an operating system kernel as an alternative to Unix. Torvalds' vision was to develop a Unix-like system that would be freely available and customizable by anyone. In 1991, he released the first version of the Linux kernel. [9]

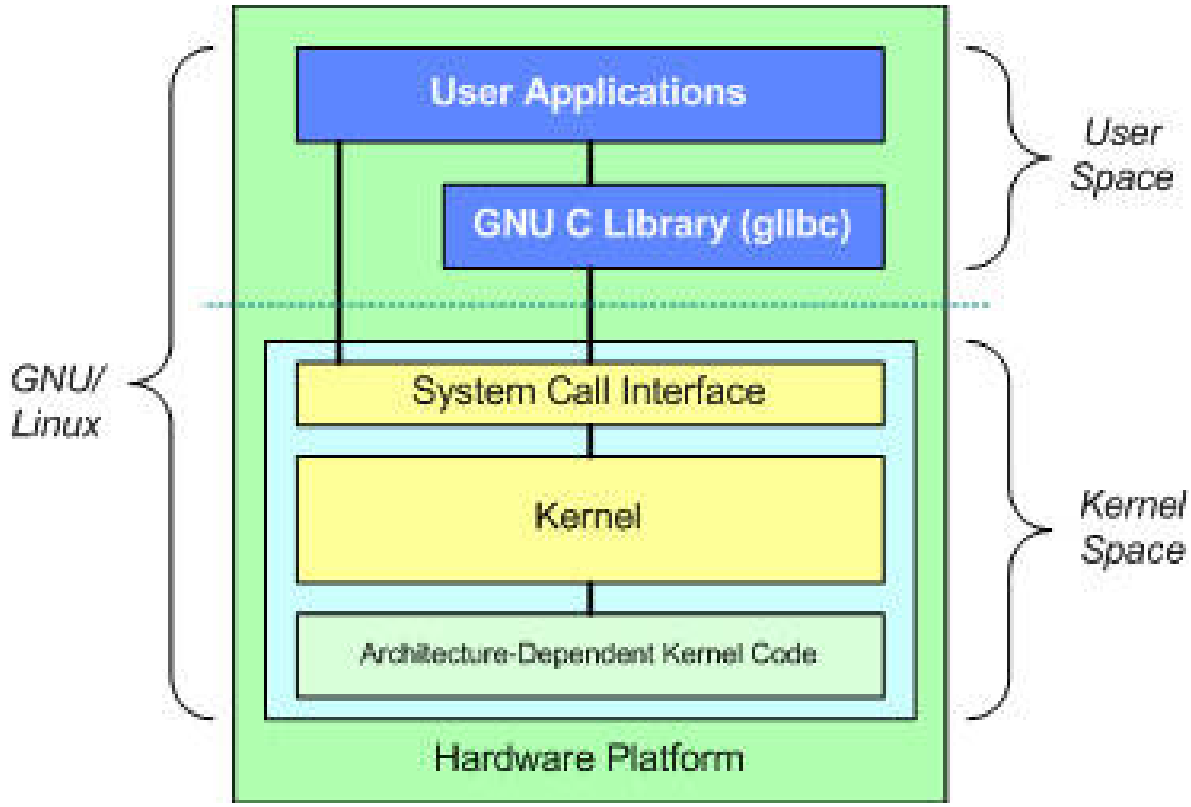

Fig. 2.11 – Architecture of GNU/Linux operating system [10]

At the heart of Linux's development is the spirit of open source — a collaborative approach to software development that emphasizes transparency, peer review, and community involvement. The Linux project demonstrated over time that it’s security, resource efficiency, privacy, robustness and reliability are worthy for commercial and personal use. This is why it is found in a wide range of industries, for which there are countless distributions, such as Ubuntu and Red Hat.

At the core of the Linux operating system lies the Linux kernel, a fundamental component responsible for managing system resources and facilitating communication between hardware and software. The kernel's functionality is enhanced by modules — small, specialized pieces of code that can be dynamically loaded and unloaded at runtime to extend the kernel's capabilities. [11]

Modules can serve a variety of purposes, from device drivers and filesystems to network protocols and security enhancements. By encapsulating specific functionalities in modular form, Linux achieves flexibility. Furthermore, the modular approach allows developers to optimize system resources by loading only the modules necessary for a given task, reducing memory footprint and enhancing overall performance. [12]

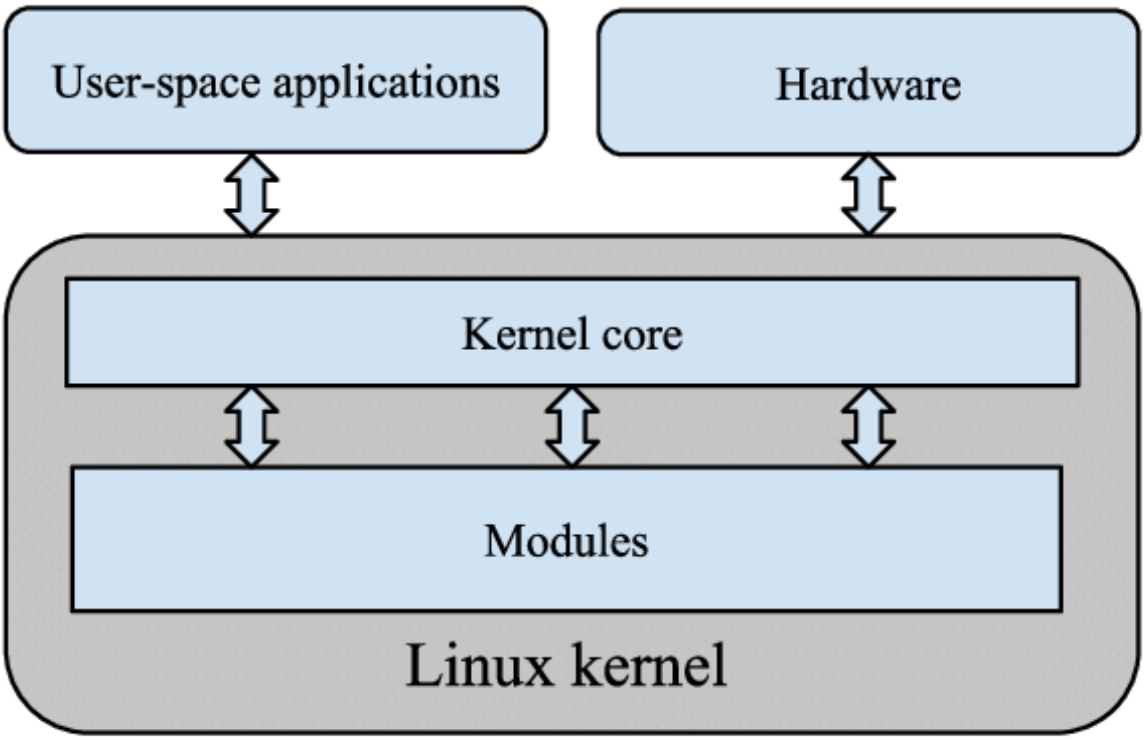

Fig. 2.12 – Linux kernel architecture [13]

2.2 Related Work

In today's market, the landscape of products catering to cooling solutions for computer systems presents a variety of options, each with its own set of advantages and limitations. While some products may lack control or affordability, others offer features designed to meet the needs of users. The commercial solutions available on the market are:

- Fan Splitter
- PSU (Power Supply Unit) Adaptor to computer fan
- Dedicated-powered Fan Hub

- Dedicated-powered Fan Controller

2.2.1 Fan Splitter

A fan splitter is a simple yet effective device that allows multiple fans to be connected to a single motherboard fan header. This solution is particularly useful for users who wish to expand their cooling setup without the need for additional fan headers. This device is built with a plug on one end and one or more sockets of the same type on the other end. Fan splitters come in various configurations, accommodating different fan numbers and offering users a simple way to add airflow within a system. However, one limitation of fan splitters is that they do not provide individual control over each fan's speed or performance. Another disadvantage of such product is that the power must be divided between the fans and can lower the maximum fan speed of the pair.

Fig. 2.13 – Fan Splitter

2.2.2 PSU Adaptor to computer fan

A PSU Adaptor to computer fan connectors is a device that was more common in the 2000s computers. This type of adaptor is built with a MOLEX connector on one side and fan connectors on the other side. The MOLEX connector is connected to the power supply unit and bypasses the control from the motherboard. The PSU ensures a steady and reliable voltage to the fans, but that means the fans lack the control provided by the motherboard-based fan control systems.

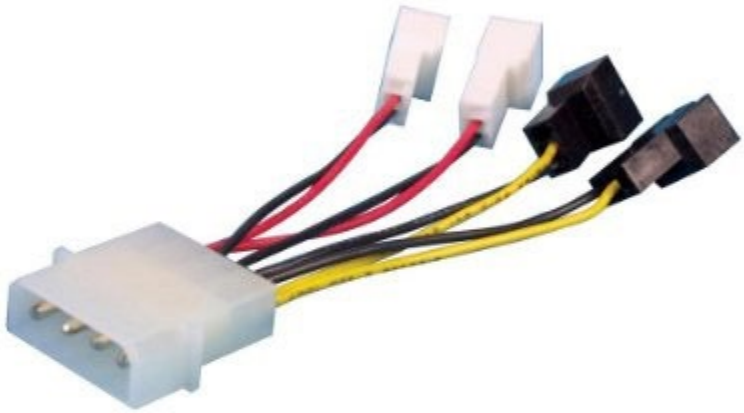

Fig. 2.14 – PSU Adaptor to computer fan

2.2.3 Dedicated-powered Fan Hub

A dedicated-powered fan hub serves as a centralized control unit for managing multiple fans within a computer system. These hubs connect to a motherboard's fan header for control, enabling all connected fan to synchronize all fan speeds through hardware controls. This type of hub requires more power than the fan controller from the motherboard offers, so an additional MOLEX or SATA power connector is needed to supply that additional power. The drawback of this system is that neither the operating system, neither the computer system does recognize that

many more fans were connected. That results that the computer fans are also not separately controlled by the computer system.

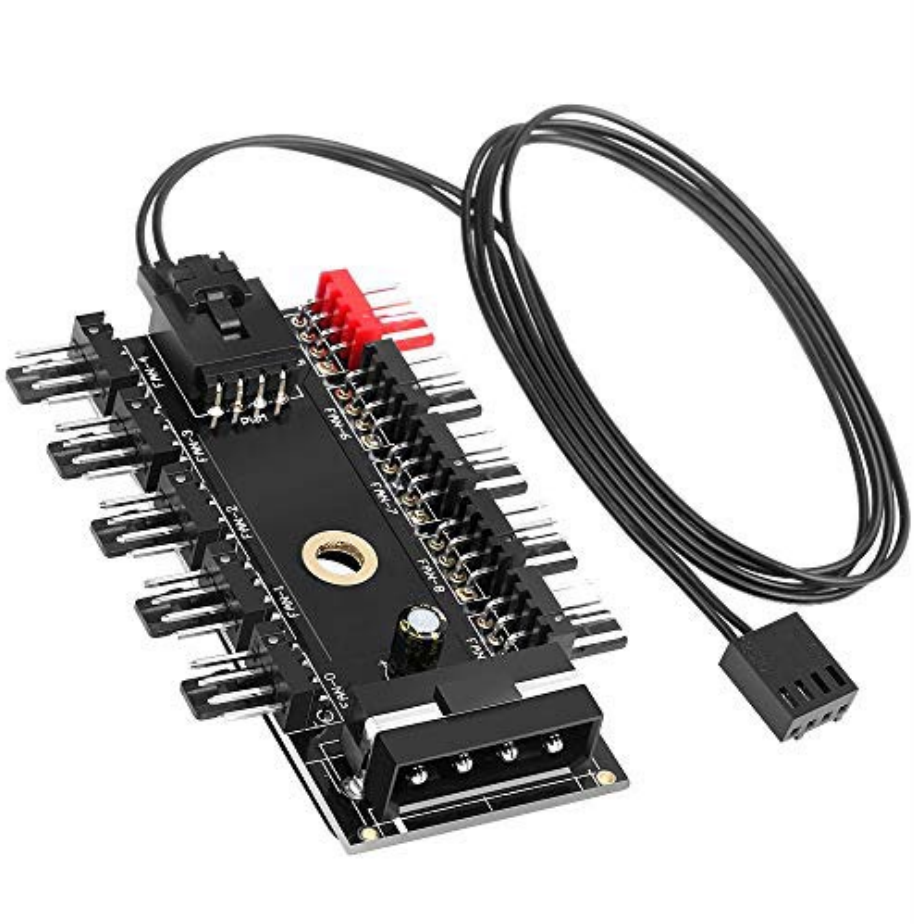

Fig. 2.15 – Dedicated-powered Fan Hub

2.2.4 Dedicated-powered Fan Controller

A dedicated-powered fan controller provides all the advantages of a dedicated-powered fan hub and more, such as the capability to connect more fans and to have them independently controlled by the user. The construction of such device is the most complex of them all, having circuits to let the user to manually set the speeds of each individual fan. This type of system does not have the capability to synchronize the computer fans connected to the motherboard and the fans connected to the fan controller. A user would set the speed of the fans through physical or software interfaces, depending on the fan controller. A physical interface for such device is controlled through knobs and is located in the place where floppy drives or disc drives would sit in computer cases. A software interface for a fan controller is created as an application in the application layer of the operating system. It offers an interface where the user can set manually the speed of the computer fans or configure profile based on the temperature of the central processing unit. A problem of such implementation is that the

application does not take in consideration the load and continuously adjusts the fans when is a background program in the operating system. If the application fails to start at the start-up of the operating system, then the fans would continue to keep the previous fan speed.

Fig. 2.16 – Fan Controller with physical interface

Fig. 2.17 – Fan Controller with software interface

2.3 Summary and comparison of existing approaches

The shortcoming of current modern computers is comprised of computer cases with a need of airflow that can't be matched by the affordable motherboards of the market. The need of motherboards that have useful capabilities is being made by more and more restrictive cases that require premium priced motherboards of which features were found on older generations. All existing solutions on the market try to mitigate this problem by adding more computer fans. The caveat of current solutions is that the user does not have the capability to modulate the speed in a simple and refined way to have precise control over the thermal system.

	Supports different fan speeds	Manually controlled by the user	Automatic control without the need of inputs from the user	Individual control over computer fans	Integration with thermal management of computer
Fan Splitter	✓	✗	✓	✗	✓
PSU Adaptor to computer fan	✗	✗	✗	✗	✗
Dedicated-powered Fan Hub	✓	✗	✓	✗	✓
Dedicated-powered Fan Controller	✓	✓	✓	✓	✗

Table 2.2 – Comparison between existing solutions on the market

The results expected out of this research paper is that the proposed system would offer a way for the user to have the added system conjoined with the existing cooling management of the motherboard. All the existing or preconfigured thermal management would apply automatically to the added system to the computer.

A use-case for the proposed solution is for servers in custom enclosed cases that require additional airflow to keep the thermal problem manageable. The proposed system would expand

the server's capability to have more control over its own airflow based on case temperature and central processing unit load.

Another use-case is for SBCs (Single Board Computer) that usually are built to run at processor temperatures of more than 90° Celsius at room temperature. These SBCs don't usually come with integrated circuits to manage active cooling solutions.

These are just some of use-cases in which the proposed system would be a benefit for the users who are not satisfied with current solutions. The solutions found on the market don't have a way to control the additional fans at the individual level at the kernel space level of an operating system and some users need to sacrifice acoustics and thermals.

3 Requirements analysis

3.1 Functional requirements

In this system, the computer user is the only actor in the system. The user has the capability to read and set manually the speed of the computer fans. The inputs from the actor are done through the CLI (Command Line Interface).

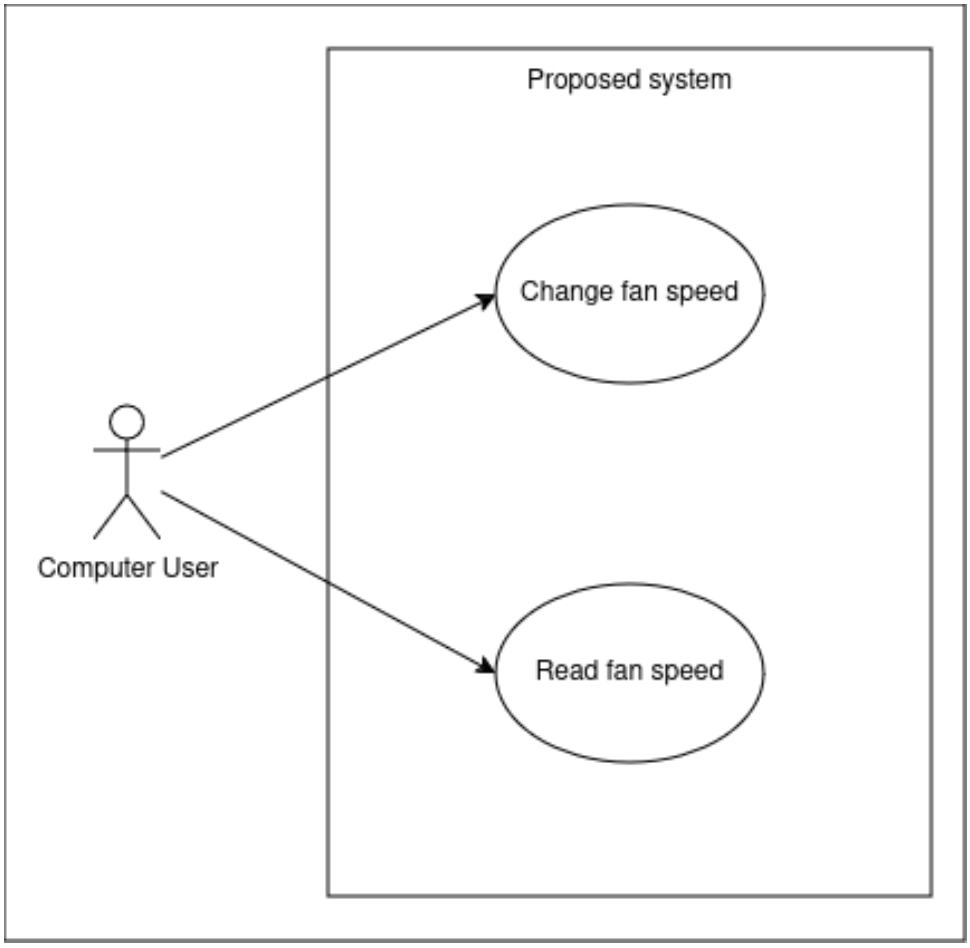

Fig. 3.1 – Software Use Case Diagram

The Use Case functions of the system are:

- Change fan speed – UC1
- Read fan speed – UC2

Use Case Name and ID:	Change fan speed – UC1
Description:	Users sets up the speed of a computer fan.
Actor:	Computer User
Preconditions:	The Operating System is booted up and running and user is in the Command Line Interface.
Postconditions:	The speed of a computer fan is changed.

Main scenario:

Actor	System
1. type command to change fan speed	
	2. system returns speed value of the fan to kernel

Table 3.1 – Change fan speed Use Case Descriptions

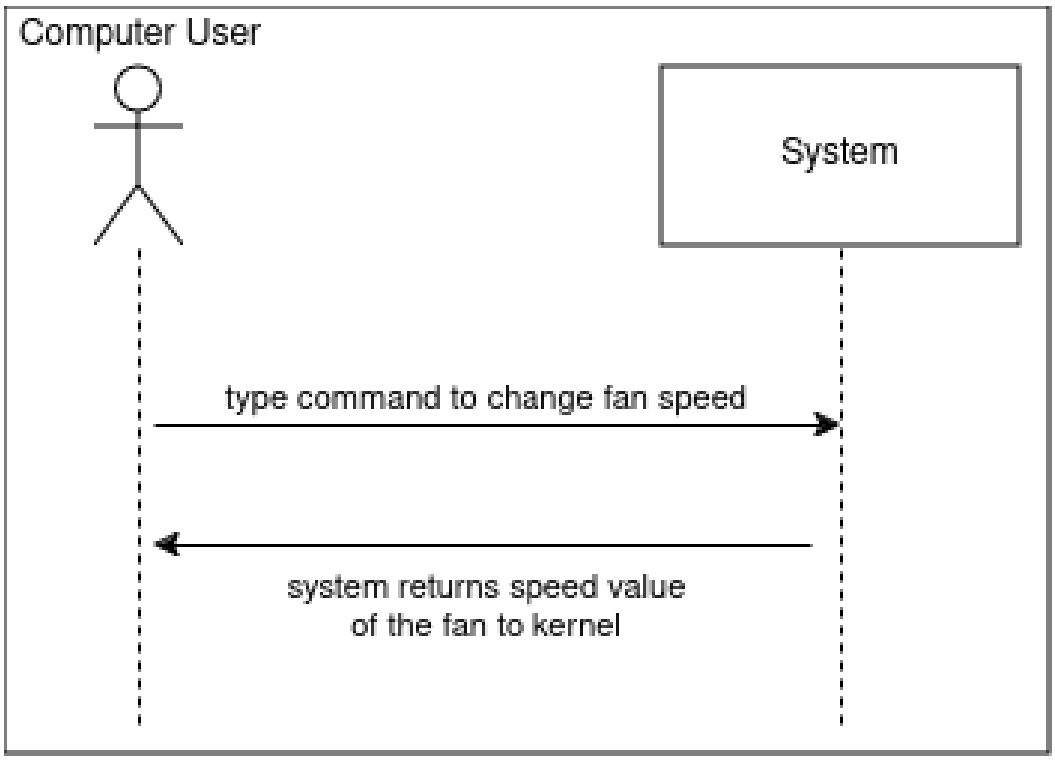

Fig. 3.2 – Change Fan Speed Sequence Diagram

<table>
<tr><td>Use Case Name and ID:</td><td>Read Fan Speed – UC2</td></tr>
<tr><td>Description:</td><td>Users reads the speed of a computer fan.</td></tr>
<tr><td>Actor:</td><td>Computer User</td></tr>
<tr><td>Preconditions:</td><td>The Operating System is booted up and running and user is in the Command Line Interface.</td></tr>
<tr><td>Postconditions:</td><td>The fan speed is printed in terminal.</td></tr>
<tr><td>Main scenario:</td><td>
<table>
<tr><td>Actor</td><td>System</td></tr>
<tr><td>1. type command to read fan speed</td><td></td></tr>
<tr><td></td><td>2. kernel prints in terminal speed value of the fan</td></tr>
</table>
</td></tr>
</table>

Table 3.2 – Read fan speed Use Case Descriptions

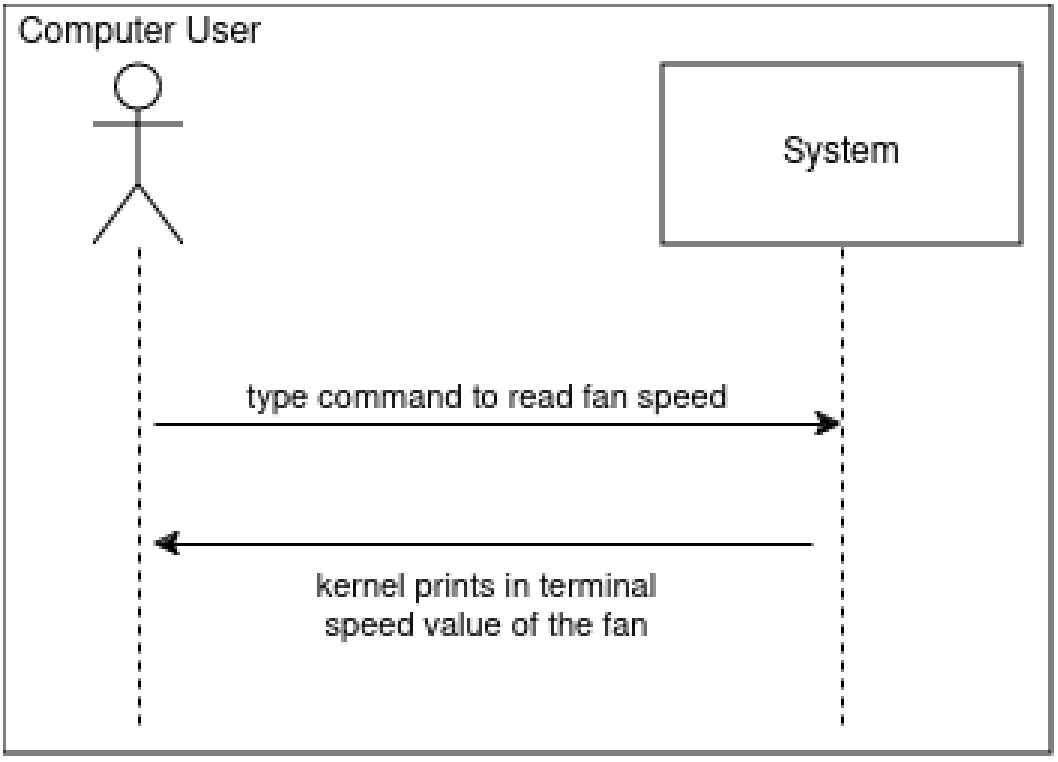

Fig. 3.3 – Read Fan Speed

3.2 Non-functional requirements

- Reliability: System has a MTBF (Mean Time Between Failures) of 90 days.
- Fault tolerance: System has integrated fail-save state, in which the computer fans will continue to supply airflow to the computer.
- Open source: It’s available on Github.

4 Design

4.1 Kernel Module Driver in a Monolithic Architecture

A loadable kernel module called a device driver controls data transfers between an OS device and device. At boot time or upon request, loadable modules are loaded, and unloaded upon request. A device driver is a group of C functions and structures that other kernel modules can use. Entry points, which are common interfaces, must be used by these procedures. The calling modules are protected from the driver's internal details by means of entry points. [14]

A part of the proposed system is a loadable kernel module that integrated in the Linux Monolithic Kernel Architecture. It expands over included HID (Human Interface Driver) driver and uses the Linux Hardware Monitoring kernel API to conjoin both motherboard and proposed system capabilities into one. This results in one common interface for the active cooling system through which the user can use all types of graphical or command line applications to modify or apply custom profiles. These profiles are customized by the user or computer generated and can take into consideration the CPU's temperature, the load of the CPU and the temperature of the air from inside the computer case. All of this information about the computer system is found directly in the Linux kernel through access to "/proc" filesystem.

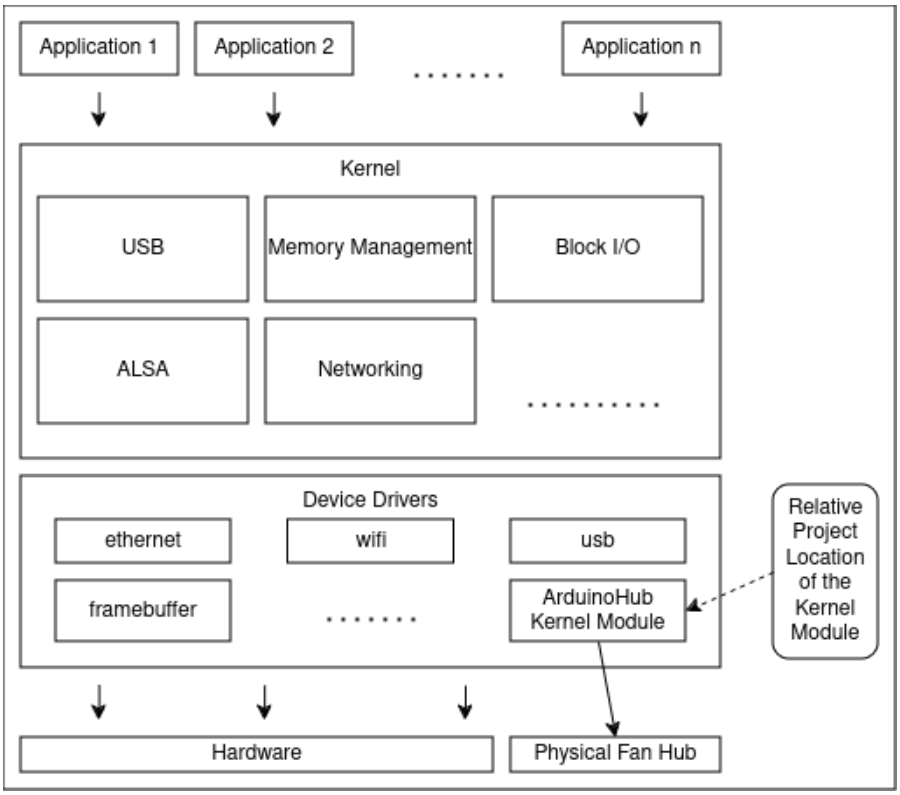

Fig. 4.1 – Location of the proposed system in the Linux Architecture

4.2 Designed Protocol for Computer to Microcontroller Communication

It's required to have a protocol for the computer system and microcontroller system to communicate. This protocol needs to take into consideration the limits of USB and to fit into 64 bytes packets.

A packet contains:

1. Start byte
2. Byte that represents the type of command
3. Data length of the sent data
4. Checksum
5. Data

The protocol is separated into two parts: commands and acknowledgements. The commands are sent by the computer and the microcontroller receiving them. The acknowledgements are sent by the microcontroller and the computer knows that previous command was executed successfully.

The data length that is being sent is the length of exactly the last part of the packet which contains the data and doesn't take into consideration the whole packet length.

The checksum is added to the protocol to detect errors that may be introduced during transmission. The chosen algorithm for this task is CRC8. CRC8 operates by dividing the data by a fixed polynomial, using binary arithmetic, and then the remainder is return to be used into the protocol. This type of algorithm is sufficient enough to ensure the integrity of sent packet.

The last part of the packet is data that contains information about fan speed, the index of the fan, version of the system or PWM duty rate. It's also capable to send a combination of all enumerated capabilities.

4.3 Electrical circuit design

The computer fans that are compatible with the project use a 4 pins female connector. These types of fans were selected for the capability of receiving PWM signal.

One of the pins of the fan gives a tachometer output signal with the following characteristics:

- Open collector output
- Two cycles per revolution
- Maximum current of 5mA [15]

A square wave signal is generated by the RPM speed signal and is required a pull-up resistor to be able to be read by a microcontroller. The pull-up resistor is a fixed-value resistor that is connected between the voltage supply of the microcontroller and the pin that reads the signal. This always ensures a known state when read.

A square wave type PWM signal needs to be supplied to the PWM pin of the fan and it also needs to conform to:

- duty-cycle range between 0% and 100%
- total voltage of 5.25V
- total current of 5mA
- frequency of 25kHz, accepted range of 21kHz to 28kHz [16]

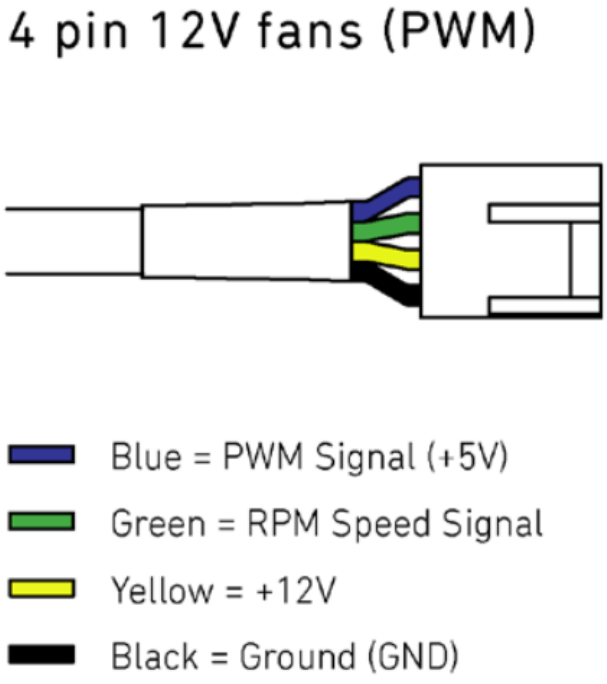

Fig. 4.2 – Computer Fan Pin Configuration

The circuit for the proposed project needs to have acces to USB interface, to microcontrollers PWM pins and to the 12V and Ground line from the PSU. The USB interface will be connected to the microcontroller. The 12V Line from the PSU supplies current to the computer fan and the PWM pin of the fan is connected to a pin capable of PWM of the microcontroller.

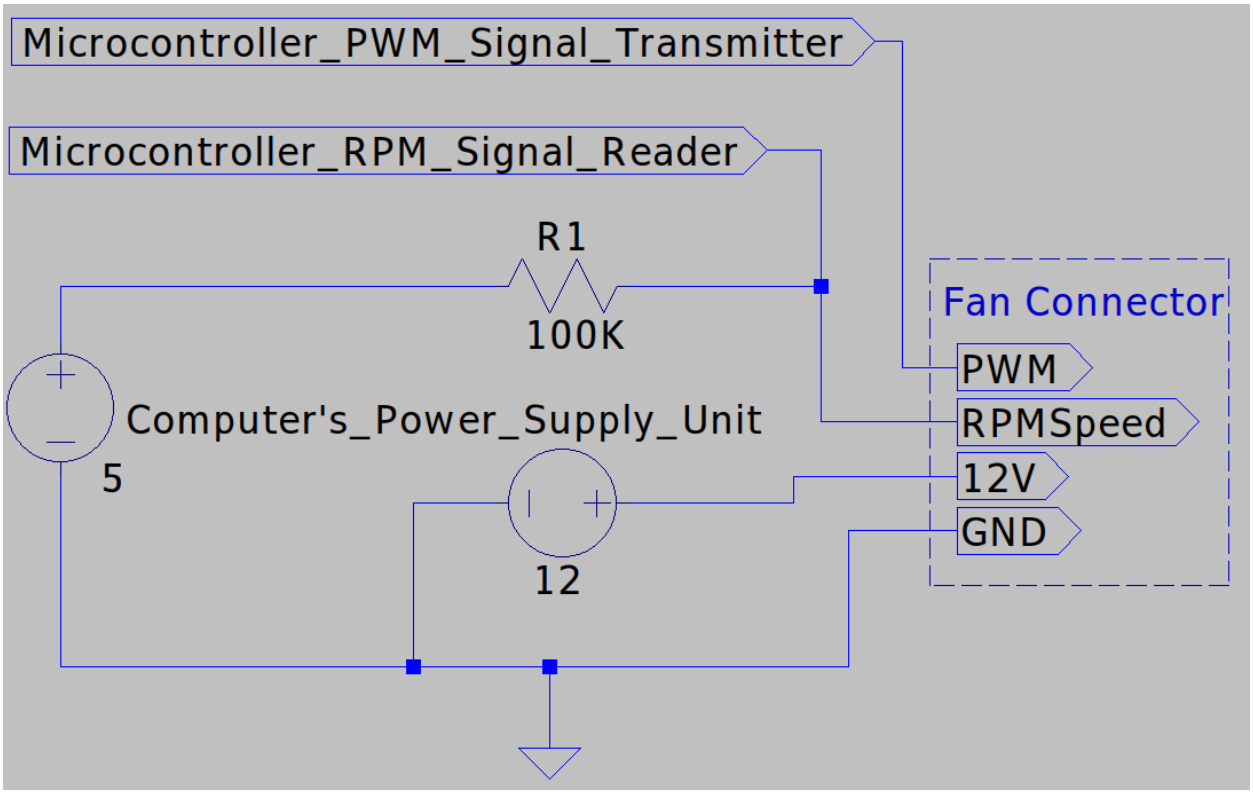

Fig. 4.3 – Circuit between Arduino and fan header

4.4 Detailed Diagrams

The kernel driver has a monolithic architecture. This architecture is chosen because of the simplicity of having one code base and gives fast results for such a small code base.

As can be seen in the figure 4.4, the driver has some key elements:

- Main Class: it contains various methods for packet transmission, device probing, raw events, and hardware monitoring operations
- Protocol: it can hold either a command or an acknowledgment
- Packet: it is structured as it was designed previously
- Device structure: representation of the Arduino as a HID device

- Hardware Monitor System structures:
    - hwmon_ops: operations for hardware monitoring
    - hwmon_chip_info: contains operations and channel information
    - hwmon_channel_info: defines hardware monitoring channels
    - hid_device_id: Defines a USB HID device
    - hid_driver: Defines a HID driver with various operations

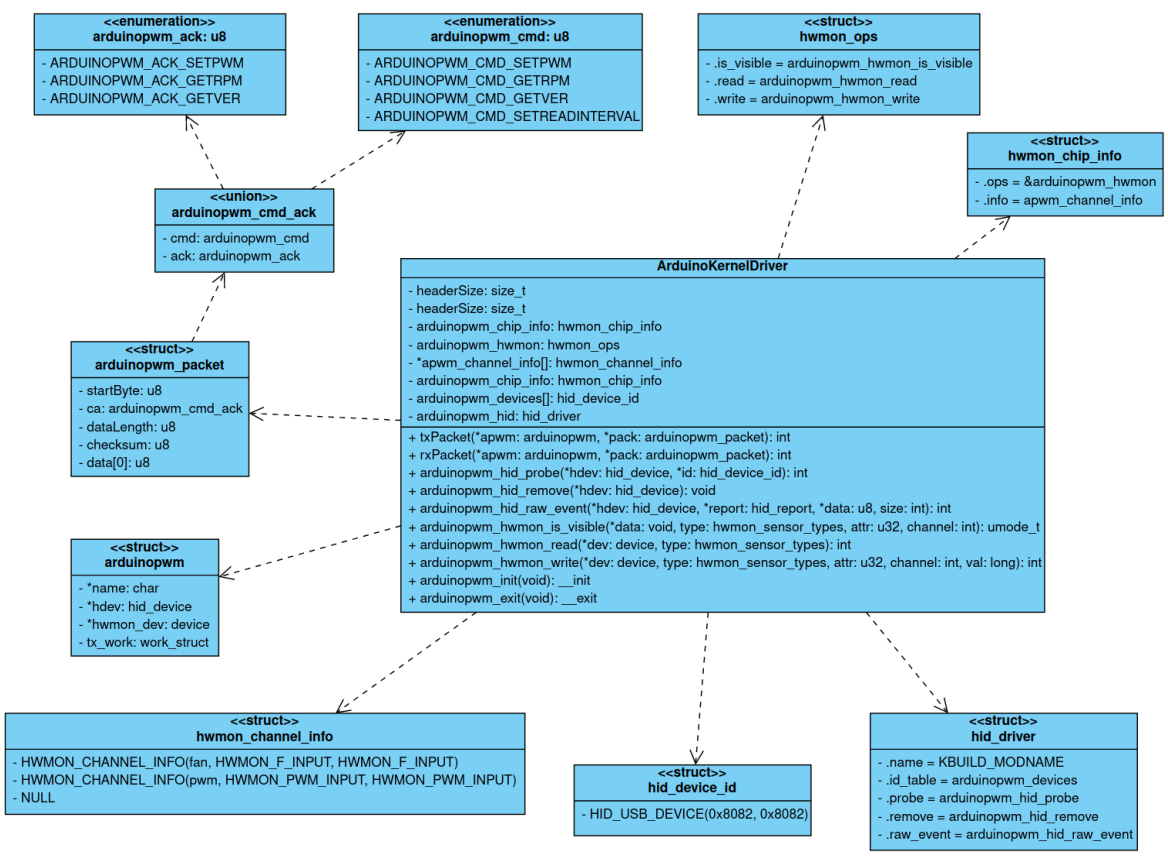

Fig. 4.4 – Arduino Fan Hub Kernel Module Driver Software Class Diagram

The key elements for the Arduino Firmware are:

- Main Class: methods to handle various operations related to PWM, RPM, version control, sending and receiving packets
- Protocol: it can hold either a command or an acknowledgment
- Packet: it is structured as it was designed previously

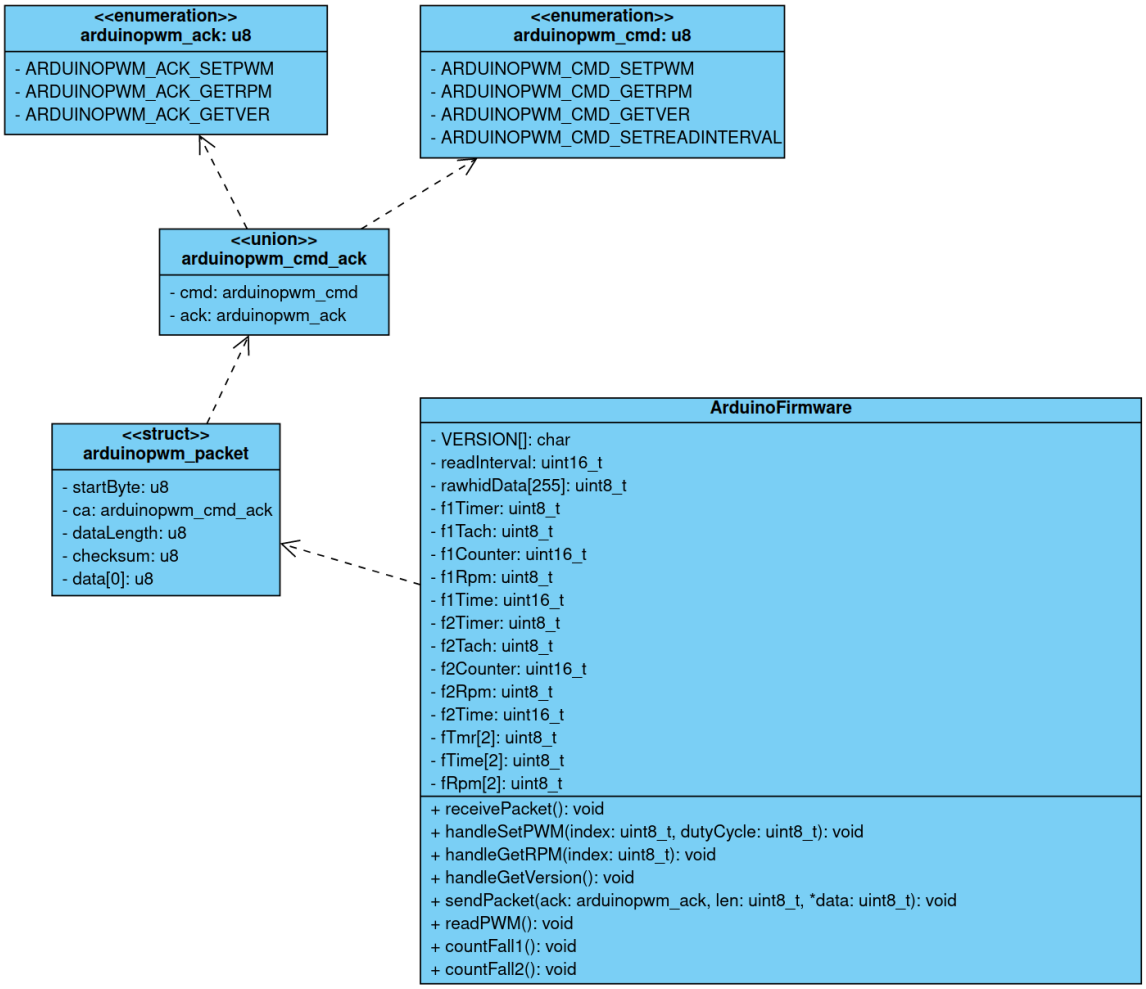

Fig. 4.5 – Arduino Fan Hub Firmware Software Class Diagram

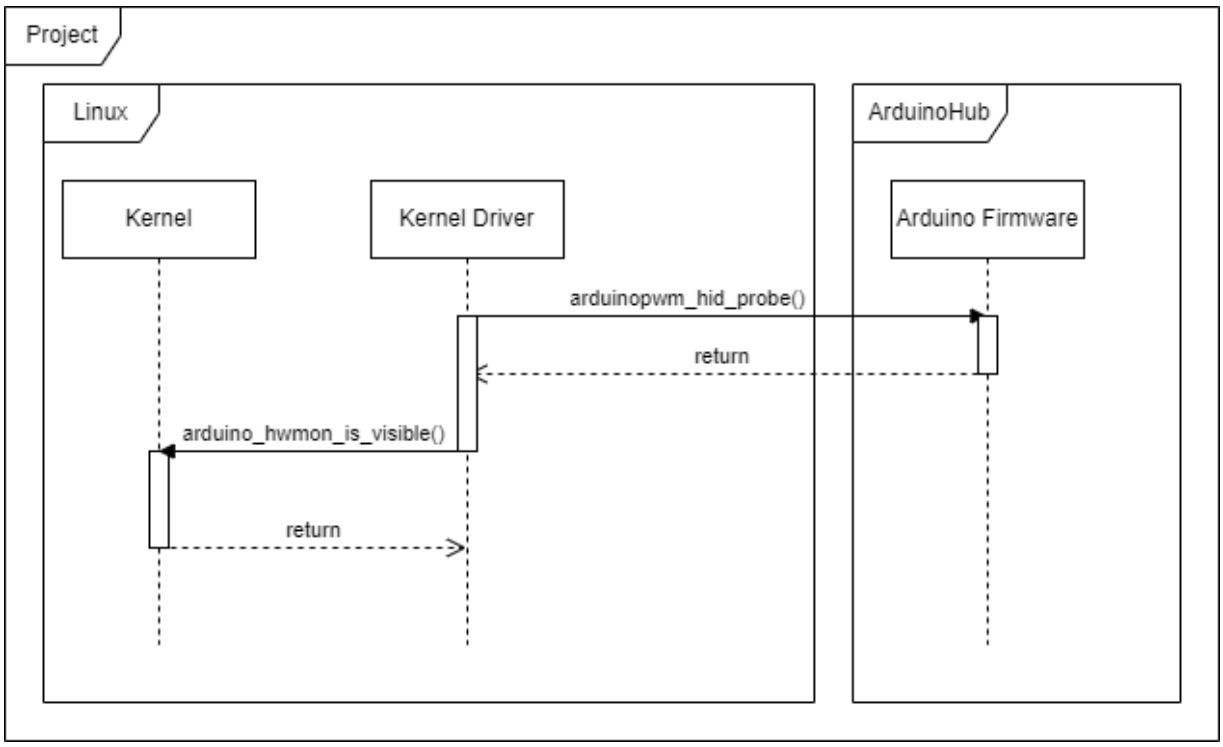

Fig. 4.6 – Connection Attempt Sequence Diagram

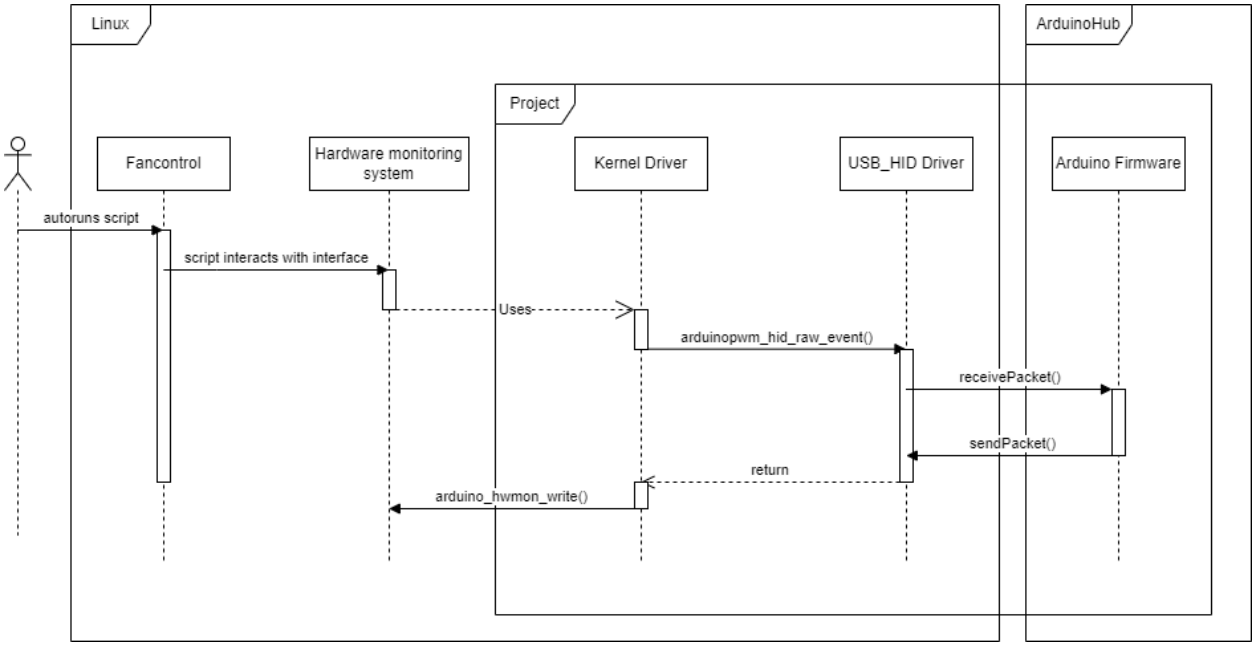

Fig. 4.7 – UC1 Sequence Diagram

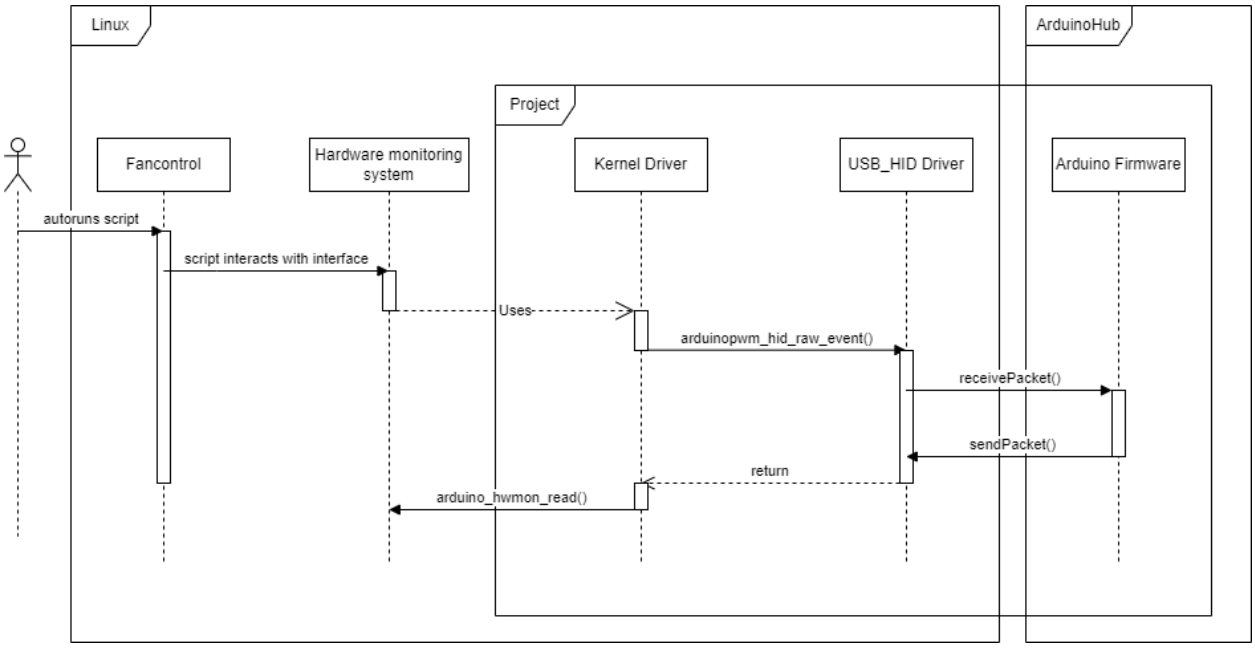

Fig. 4.8 – UC2 Sequence Diagram

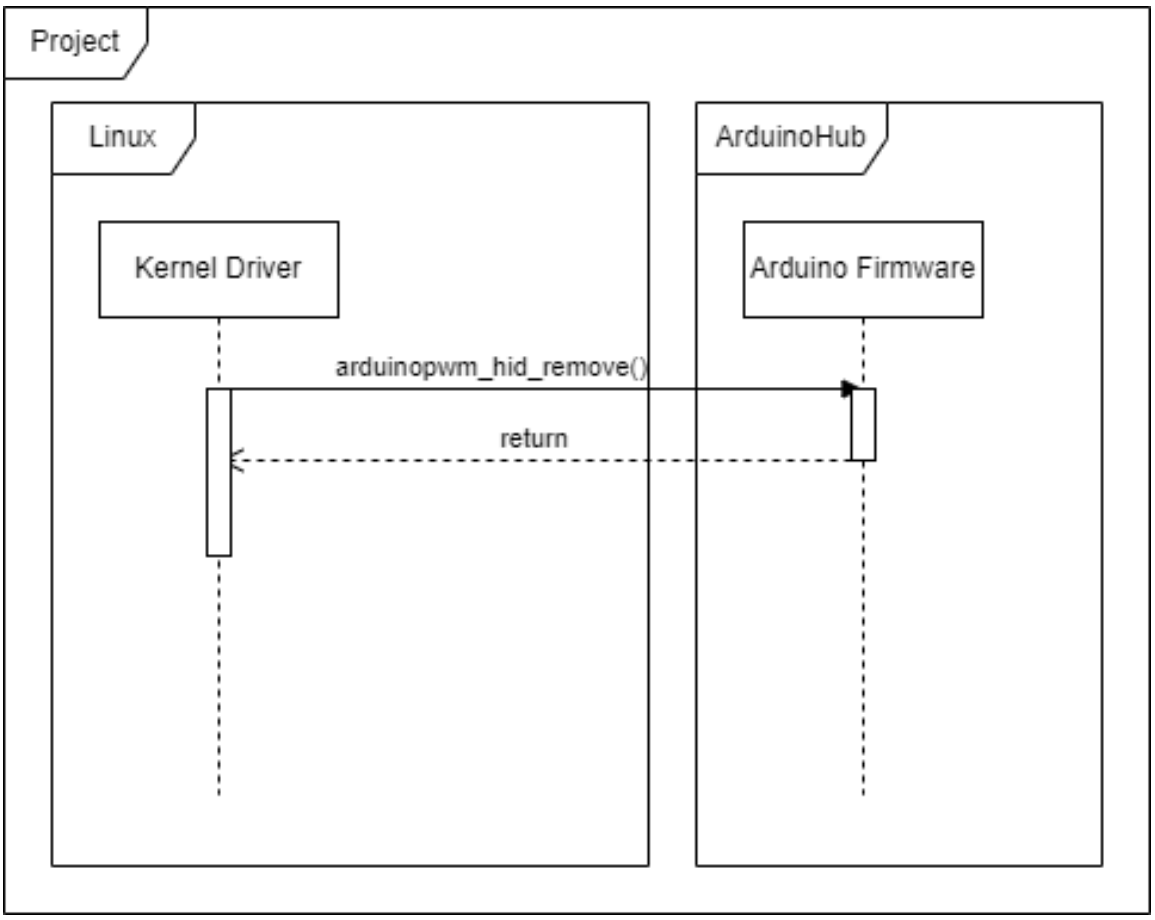

Fig. 4.9 – Disconnect Hub from Kernel Sequence Diagram

4.5 Overview of the proposed system design

The system is composed of three main components:

- The kernel module driver
- The microcontroller firmware
- Hardware implementation of the fan hub

Computer fans are connected to the fan hub, and the communication between the hub and the kernel is made with the help of the previously described protocol. The scope of the kernel driver is to be an interface for the hardware to the kernel.

5 Implementation

5.1 Technologies used

The project is developed by using Linux packages called *base-devel* and *linux-header*, GCC (GNU Compiler Collection), RawHID Arduino library, VSC with PlatformIO plugin and an Arduino microcontroller.

In the Linux ecosystem, the *base-devel* and *linux-header* packages are essential components to Linux Kernel Module development. The *base-devel* package group includes a collection of fundamental development tools and libraries necessary for compiling and building software from source. This typically encompasses tools like GCC, make, autoconf, and other utilities vital for development tasks. On the other hand, the *linux-header* packages contain the header files for the Linux kernel, which are crucial for building and compiling kernel modules and software that interacts closely with the kernel. [17] These headers define the interface between the kernel and user-space applications, enabling developers to write programs that can communicate effectively with the underlying hardware.

The GNU Compiler Collection is a powerful suite of compilers developed by the GNU Project for various programming languages, including C, C++, Fortran, Ada, and more. GCC supports a wide array of hardware architectures and operating systems and is used extensively in academic research, commercial software development, and operating system projects, including major Linux distributions.

PlatformIO is a robust open-source ecosystem designed to enhance the development of embedded systems and IoT applications. It provides a cross-platform build system, a unified debugger, and a comprehensive library, all integrated within the very popular code editor, Visual Studio Code. [18]

An Arduino microcontroller is a compact, open-sourced, single-board microcontroller designed for ease of use in electronics projects and prototyping. The chosen Arduino-compatible board for this project is a copy of a SparkFun Pro Micro; the version that is running at 16 GHz and 5V. This board is selected for the it’s specific controller, the ATmega32U4. The Atmega32U4 has an integrated USB interface and does not require like an Arduino Uno a separate chip to handle this task.

5.2 Circuit Implementation

The circuit was initially built on a breadboard and was implemented based the design phase circuit. In this circuit are used:

- 2 resistors of 10 kΩ
- some jumper wires
- Arduino Pro Micro
- computer fan
- breadboard power supply
- breadboard

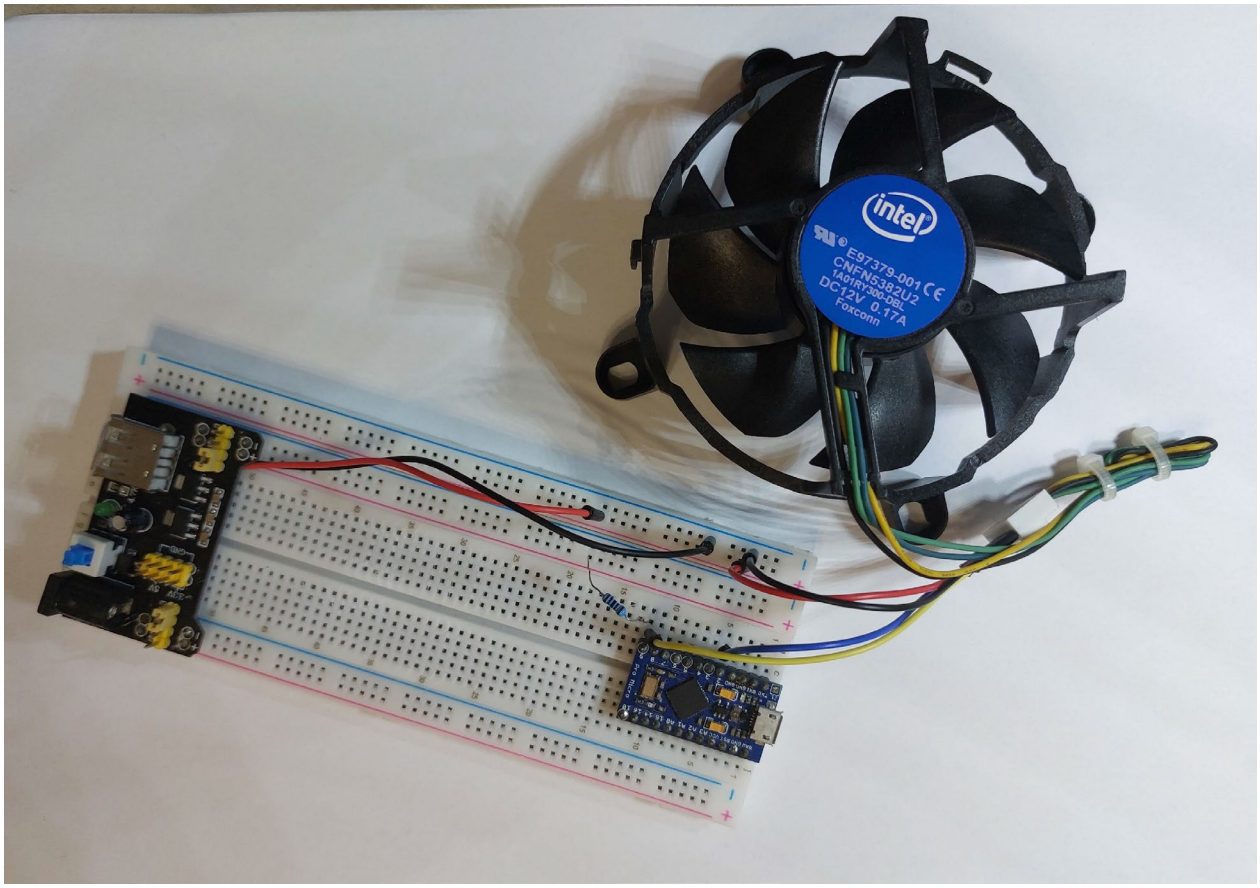

Fig. 5.1 – Prototype on a breadboard

It was necessary to write some code to test the prototype. The expected results from this test are to have the capability to set the duty cycle and read the fan speed from the Serial Monitor of the Arduino IDE. The code snippets for this test are found below.

```
1  void setup() {
2    Serial.begin(9600);
3    pinMode(PWMPIN, OUTPUT);
4    TCCR1A = (1 << COM1A1);
5    TCCR1A |= (1 << COM1B1);
6    TCCR1A |= (1 << WGM10);
7    TCCR1B = (1 << WGM12);
8    TCCR1B |= (1 << CS11);
9    OCR1A = 0;
10   attachInterrupt(digitalPinToInterrupt(TACHPIN), countFall,
FALLING);
11 }
12
13 void countFall() {
14   fanSpeedCounter++;
15 }
```

Code Snippet 5.1 – Arduino Setup

In the code snippet above, the “PWMPIN” is set to the specific requirements for computer fans. At line 4 and 5, the registers OC1A/OC1B are set to clear on compare match and to set OC1A/OC1B at BOTTOM. Line 6 and 7 are required to set the internal counter of the Arduino to Fast PWM, 8-bit mode. Line 8 is used to set the prescaler of the counter to a frequency of $clk_{I/O}/8$.

At line 10, an interrupt is attaches to the pin “TACHPIN”. The “digitalPinToInterrupt” function returns the pin number if it is an appropriate interrupt pin. The “countFall” is the name of the interrupt service routine (ISR) that will be called when a falling edge is detected on “TACHPIN”. This gives the Arduino the capability to count each time when the Hall effect sensor passes.

```
1   if(millis() - timeMeas > 5000UL) {
2     analogWrite(PWMPIN, duty * 255 / 100);
3     Serial.println(duty);
4     timeMeas = millis();
5   }
6
7   if(millis() - timeReadFan > 1000UL) {
8     Serial.print("RPM: ");
9     Serial.println(fanSpeedCounter * 60 / 2);
10    fanSpeedCounter = 0;
11    timeReadFan = millis();

12  }
```

Code Snippet 5.2 – Read and set up fan speed

In this code snippet, the fan speed and duty rate are set. The duty rate is set at an interval of 5 seconds using the command “analogWrite” at line 2. The fan speed is read each second and is printed in the Serial Monitor of Arduino IDE.

The results from this test were correct, as the fan speed was between 1000 – 2400 RPM and the speed of the fan increased linearly to the duty rate.

The next step for the prototype was to be built on a perfboard. The used components are:

- 1x perfboard
- 5x jumper cable
- 2x 10 kΩ resistor
- 2x 4 pin male header
- 2x 12 pin female header
- 1x Molex female electrical connector

The previous test was conducted again to check if mistakes were made during construction. The perfboard variant of the project passed the test as well.

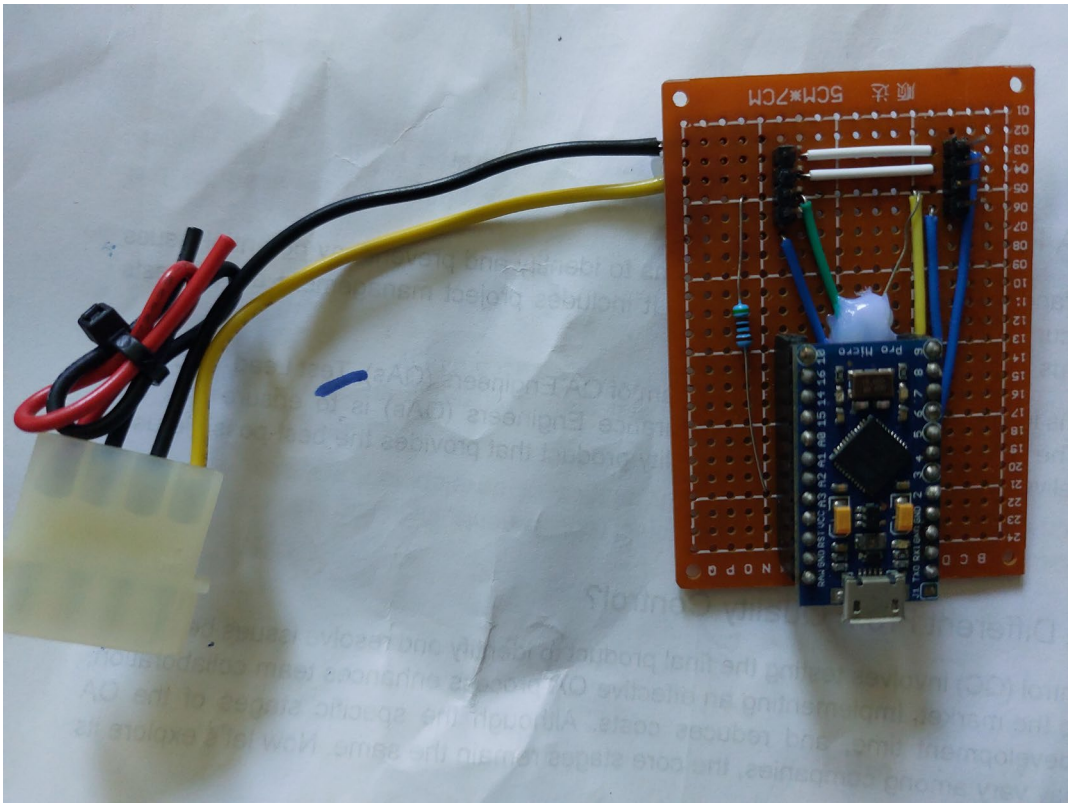

Fig 5.2 – Picture from above of the Arduino Hub

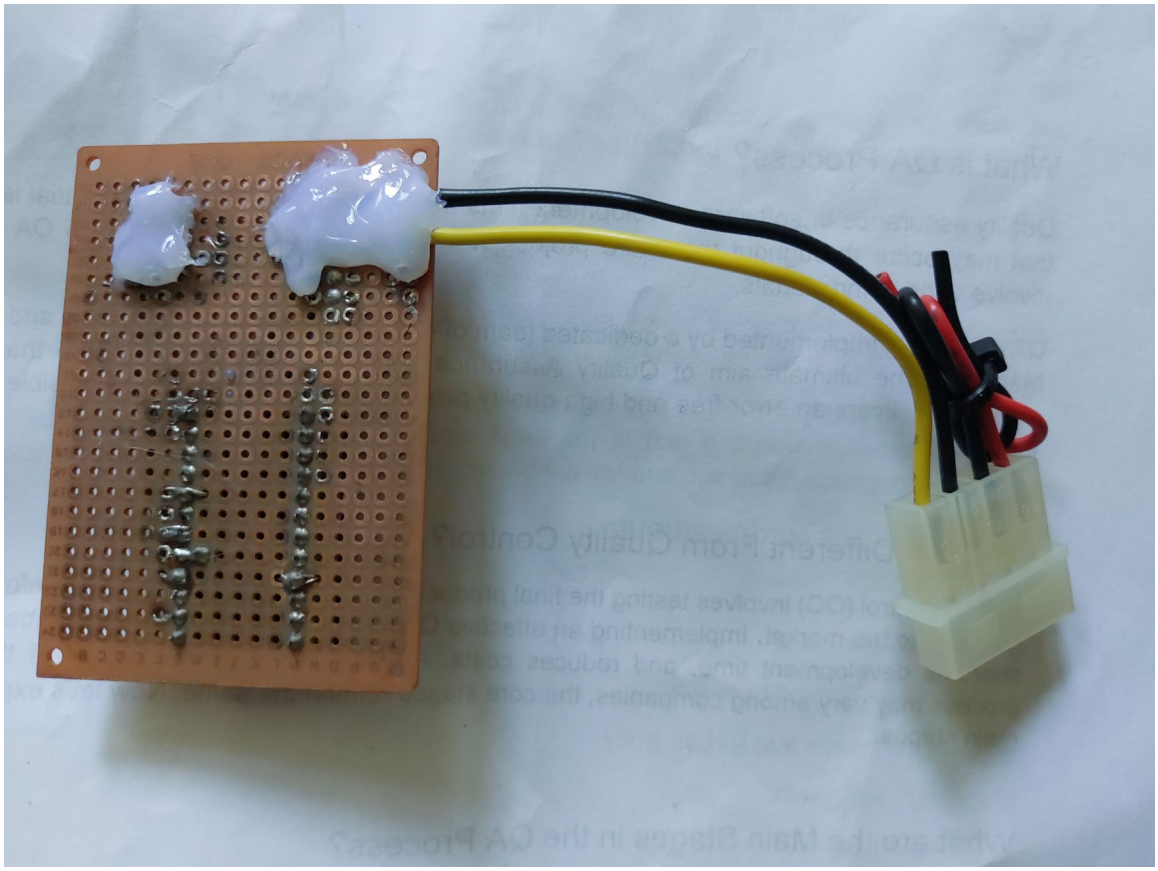

Fig 5.3 – Picture from below of the Arduino Hub

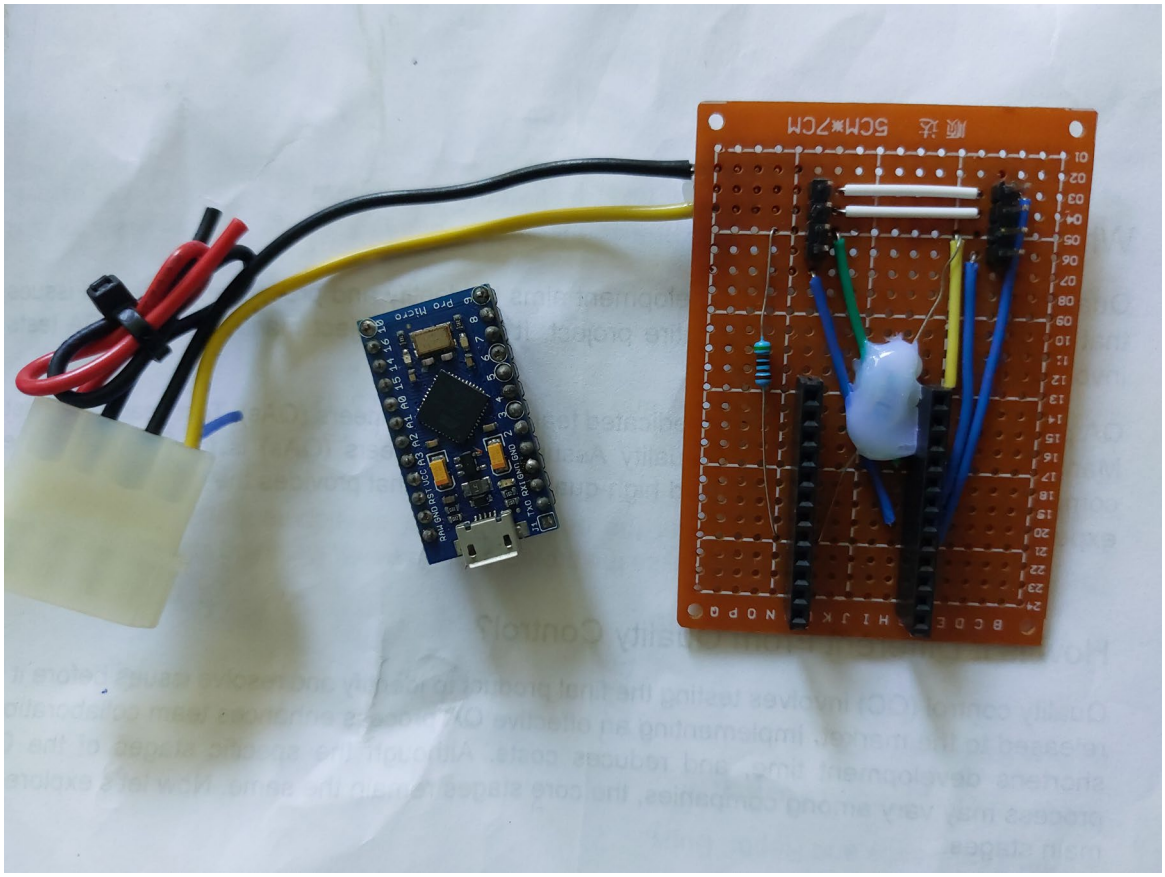

Fig 5.4 – Picture from above of the Hub without the Arduino in the socket

5.3 Arduino Firmware

The final version of the firmware was written with PlatformIO plugin from Visual Studio Code and Arduino IDE was abandoned. For this version, an identical setup was created from the test Arduino code prototype. The only difference is the addition of "RawHID" communication setup.

An enumeration is used to declare all commands and a different enumeration is used for acknowledgements. Both enumerations are of byte type and they are both hold in a union class type. In this code snippet can also be found the required returned data for the handle functions.

```
1  enum arduinopwm_cmd : u8 {
2     ARDUINOPWM_CMD_SETPWM=0x01, /* data: index u8, pwm value u8
*/
3    ARDUINOPWM_CMD_GETRPM=0x02, /* data: index u8 */
4    ARDUINOPWM_CMD_GETVER=0x10, /* no data */
5    ARDUINOPWM_CMD_SETREADINTERVAL=0x11, /* data: time in ms u16
*/
6  };
7  enum arduinopwm_ack : u8 {
8     ARDUINOPWM_ACK_SETPWM=0x01, /* data: index u8, pwm value u8
*/
9    ARDUINOPWM_ACK_GETRPM=0x02, /* data: index u8, rpm value u16
*/
10   ARDUINOPWM_ACK_GETVER=0x10, /* data: version u32 */
11 };
12 union arduinopwm_cmd_ack{
13   enum arduinopwm_cmd cmd;
14   enum arduinopwm_ack ack;
15 };
```

Code Snippet 5.3 – Structure of all commands and acknowledgements

"receivePacket" function does read the bytes received over the USB and categorizes the bytes into the structure of the protocol. This happens from line 2 to line 13. From line 14 to line 25, the command byte is compared with existing values in the program and calls a specific function to the requested action.

```
1  void receivePacket() {
2    arduinopwm_cmd cmd;
3    uint8_t dataLen;
4    uint8_t dataChecksum;
5    static uint8_t dataBuffer[64];
6    auto bytesAvailable = RawHID.available();
7    if(bytesAvailable > 4 && RawHID.read() == 0x00) {
8      cmd = RawHID.read();
9      dataLen = RawHID.read();
10     dataChecksum = RawHID.read();
```

```
11          if(RawHID.readBytes(dataBuffer, dataLen) != dataLen)
return;
12    }
13    else return;
14    switch(cmd) {
15      case ARDUINOPWM_CMD_SETPWM:
16        handleSetPWM(dataBuffer[0], dataBuffer[1]); break;
17      case ARDUINOPWM_CMD_GETRPM:
18        handleGetRPM(dataBuffer[0]); break;
19      case ARDUINOPWM_CMD_GETVER:
20        handleGetVersion(); break;
21      case ARDUINOPWM_CMD_SETREADINTERVAL:
22        readInterval = dataBuffer[0]; break;
23      default: break;
24    }
25 }
```

Code Snippet 5.4 – Receive packet from computer function

In the next code snippet, it will be shown a previous implementation for sending a packet to the computer. Again, the structure of the packet is recreated using the protocol and all data transmission is handled by the “RawHID” library.

```
1    void sendPacket(arduinopwm_ack ack, uint8_t len, uint8_t*
data)
2  {
3    static uint8_t buffer[64];
4    auto& packet = *(arduinopwm_packet*)&buffer;
5
6    packet.startByte = 0x00;
7    packet.ca.ack = ack;
8    packet.dataLength = len;
9    packet.checksum = 0x42;
10
11   memcpy(packet.data, data, len);
12
13   RawHID.write(buffer, sizeof(arduinopwm_packet) + len);
14 }
```

Code Snippet 5.5 – Send packet to computer function

And the last code snippet shown in this chapter will be a handle function to get RPM value. It’s seen that an array that contains the index and the RPM value of the fan is created and is sent to the function found above.

```
1 void handleGetRPM(uint8_t index) {
2   uint8_t retData[]={index, fRpm[index]};
3   sendPacket(ARDUINOPWM_ACK_GETRPM, sizeof(retData), retData);
4 }
```

Code Snippet 5.6 – Handle Function for sending the RPM value

The rest of handle function are built the same, only that the returned data will be different based on the type of command.

5.4 Kernel Module Driver

This function initializes the device, allocates memory for the device context, starts HID hardware, and registers the hardware monitoring device.

```
1  static int arduinopwm_hid_probe(struct hid_device *hdev, const struct
hid_device_id *id) {
2       struct arduinopwm *apwm;
3       int ret;
4       apwm = devm_kzalloc(&hdev->dev, sizeof(*apwm), GFP_KERNEL);
5       if (!apwm) return -ENOMEM;
6       ret = hid_parse(hdev);
7       if (ret) return ret;
8       ret = hid_hw_start(hdev, HID_CONNECT_HIDRAW);
9       if (ret) return ret;
10      ret = hid_hw_open(hdev);
11      if (ret) goto out_hw_stop;
12      apwm->hdev = hdev;
13      hid_set_drvdata(hdev, apwm);
14      mutex_init(&apwm->mutex);
15      hid_device_io_start(hdev);
16      apwm->hwmon_dev = hwmon_device_register_with_info(&hdev->dev,
                                                          KBUILD_MODNAME,
                                                          apwm,
                                                          &arduinopwm_chip_info,
                                                          0);
17      if (IS_ERR(apwm->hwmon_dev)) {
18          ret = PTR_ERR(apwm->hwmon_dev);
19          goto out_hw_close;
20      }
21      return 0;
22 out_hw_close:
23      hid_hw_close(hdev);
24 out_hw_stop:
25      hid_hw_stop(hdev);
26      return ret;
27 }
```

Code Snippet 5.7 – Initialize HID

The following function is called when the device is removed. It unregisters the hardware monitoring device and stops HID hardware.

```
1  static void arduinopwm_hid_remove(struct hid_device *hdev) {
2      struct arduinopwm *apwm = hid_get_drvdata(hdev);
3      hwmon_device_unregister(apwm->hwmon_dev);
4      hid_hw_close(hdev);
5      hid_hw_stop(hdev);
6  }
```

Code Snippet 5.8 – Remove HID

This next code snippet handles raw HID events by copying received data into the receive buffer and processing it with rxPacket.

```
1  static int arduinopwm_hid_raw_event(struct hid_device *hdev, struct hid_report
*report, u8 *data, int size) {
2      struct arduinopwm *apwm = hid_get_drvdata(hdev)
3      memcpy(&apwm->rxbuf, data, min(ARDUINOPWM_SIZE_RXBUF, size));
4      struct arduinopwm_packet* rxPack = (struct arduinopwm_packet*)&apwm-
>rxbuf;
5      rxPacket(apwm, rxPack);
6      return 0;
7  }
```

Code Snippet 5.9 – Read HID packet

The following function determines the file mode for each supported attribute by the HID device.

```
1  static umode_t arudinopwm_hwmon_is_visible(const void *data, enum
hwmon_sensor_types type, u32 attr, int channel) {
2      const struct arduinopwm *apwm = data;
3      if(channel < 0 || channel >= ARDUINOPWM_FANS) return 0;
4      if(type == hwmon_fan && attr == hwmon_fan_input) return 0444;
5      if(type == hwmon_pwm && attr == hwmon_pwm_input) return 0644;
6      return 0;
7  };
```

Code Snippet 5.10 – Driver determines compatible HID attributes

This function handles reading hardware monitoring attributes such as fan RPM and PWM values. It sends a request to the Arduino if necessary and returns the latest known values.

```
1  static int arduinopwm_hwmon_read(struct device *dev, enum hwmon_sensor_types
type, u32 attr, int channel, long *val) {
2      struct arduinopwm *apwm = dev_get_drvdata(dev);
3      if(channel<0 || channel >= ARDUINOPWM_FANS) return -ENOTSUPP;
```

```
4       if(type==hwmon_fan && attr == hwmon_fan_input) {
5           struct arduinopwm_packet *rpmRequest = (void*)apwm->txbuf;
6           rpmRequest->startByte=0x00;
7           rpmRequest->ca.cmd=ARDUINOPWM_CMD_GETRPM;
8           rpmRequest->dataLength=1;
9           rpmRequest->data[0] = channel;
10          txPacket(apwm, rpmRequest);
11          *val = apwm->fans[channel].lastRPM;
12          return 0;
13      }
14      if(type==hwmon_pwm && attr == hwmon_pwm_input) {
15          *val = apwm->fans[channel].lastPWM;
16          return 0;
17      }
18      return -ENOTSUPP;
19  }
```

Code Snippet 5.12 – Reads fan speed

This function handles writing hardware monitoring attributes such as setting PWM values for fans.

```
1   static int arduinopwm_hwmon_write(struct device *dev, enum hwmon_sensor_types
type, u32 attr, int channel, long val) {
2       struct arduinopwm *apwm = dev_get_drvdata(dev);
3       if(channel<0 || channel >= ARDUINOPWM_FANS) return -ENOTSUPP;
4       if(val<0 || val > 255) return -EINVAL;
5       if(type == hwmon_pwm && attr == hwmon_pwm_input) {
6           struct arduinopwm_packet *pwmSet = (void*)apwm->txbuf;
7           pwmSet->startByte=0x00;
8           pwmSet->ca.cmd=ARDUINOPWM_CMD_GETRPM;
9           pwmSet->dataLength=0;
10          pwmSet->data[0] = channel;
11          txPacket(apwm, pwmSet);
12          return 0;
13      }
14      return -EOPNOTSUPP;
15  }
```

Code Snippet 5.13 – Sets up fan speed

6 Validation

The testing methodology for the entire system is to place in an ideal position the fan, to run the fancontrol script from lm_sensors library and to measure the temperature with and without the system running. The GPU temperature was measured and it was read from the system at an interval of one second.

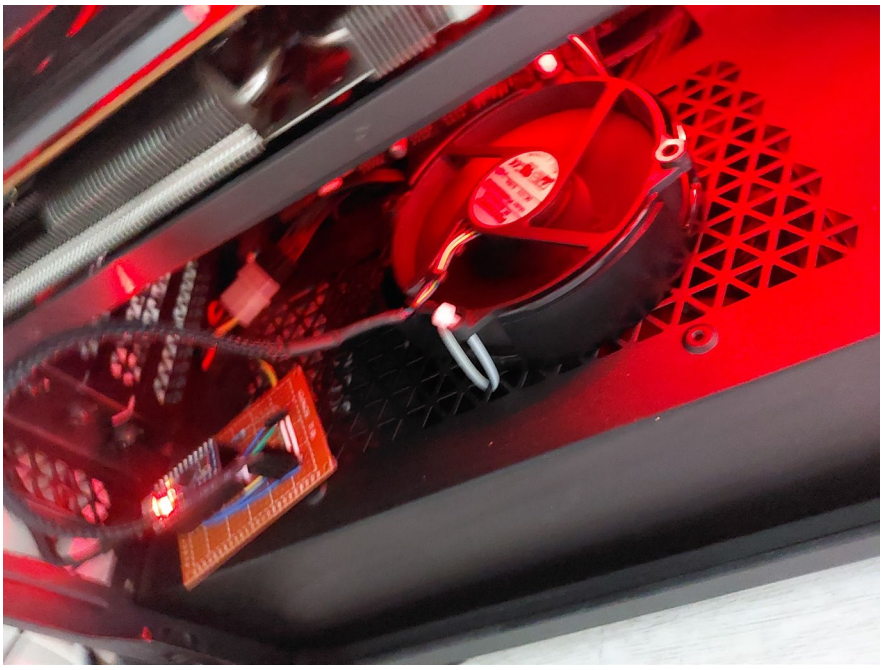

Fig 6.1 – Picture during validation phase

The computer was let to heat soak and afterwards the test started. The results from this test are:

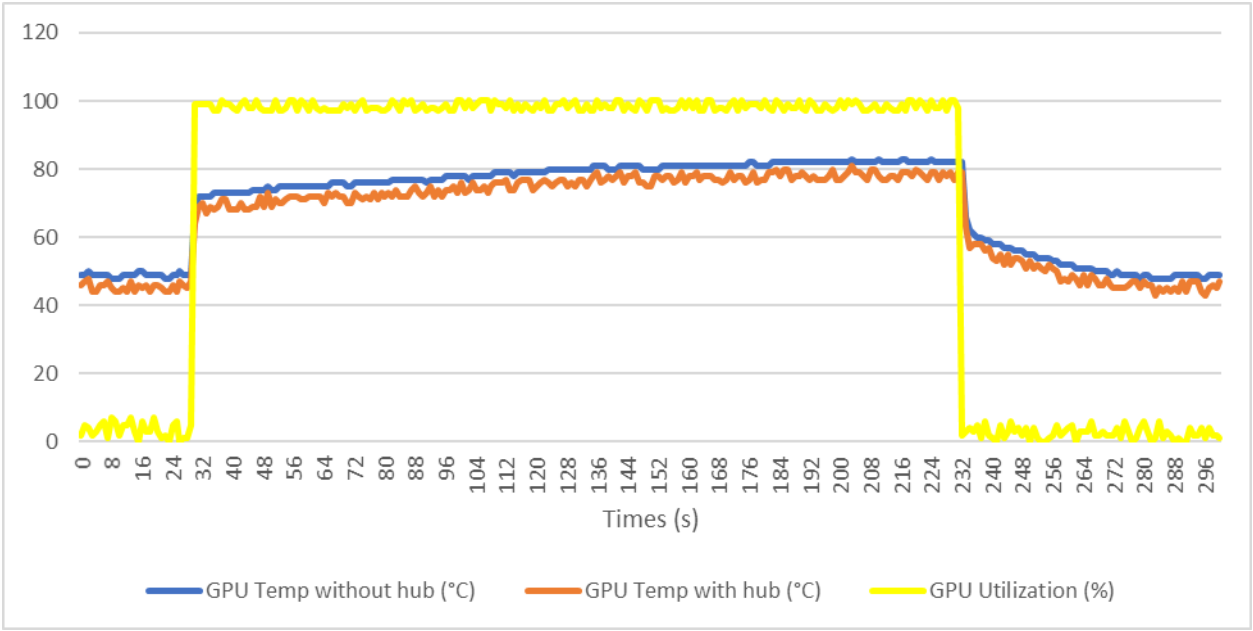

Fig. 6.2 – Obtained results after the test was concluded

The test results indicate that adding the Arduino hub to the computer effectively reduces the GPU temperature. Over the duration of the test, the mean reduction in temperature was 3.6 degrees Celsius when the fan hub was utilized. The graph clearly shows that during the period of high GPU utilization, the temperatures with the fan hub were consistently lower than without the hub. This demonstrates palpable results for the fan hub in maintaining lower GPU temperatures under load.

7 Contributions to the field of computer cooling

The contributions of this paper to the field of computer cooling are impactful, addressing a critical need in enhancing computer performance while maintaining system stability. This project focused on increasing the capacity of computers to add fans and thereby reducing both sound and temperature, introduces a sophisticated solution that bridges the gap between existing hardware and software cooling solutions.

At the heart of paper is the development of a unique device that integrates directly with the operating system at the kernel level. This integration is particularly noteworthy as it distinguishes this solution from the majority of current market offerings, which are often limited to application-level connections. By interfacing with the kernel, the developed device provides more direct control over the cooling mechanisms, ensuring that the operating system is fully aware of the additional fans. This level of integration not only enhances the system's cooling capabilities but also ensures that the presence of additional cooling solutions is seamlessly recognized and managed by the computer's existing system.

This paper also addresses a significant gap in the market where most available products are hubs that the computer does not recognize as part of its cooling system. By making the operating system aware of the additional fans, it ensures that more airflow is given to the system and that it can offer an effective reduction in temperature and noise levels. This is particularly beneficial for users who require high-performance computing environments, such as power users and those utilizing non-conventional computer cases or server-grade hardware in desktop setups.

A key advantage of the solution is its compatibility with Linux systems. However, it simplifies the user experience by eliminating the need for separate user applications to manage different cooling systems. Users can control the entire cooling setup through the operating system, providing a unified and convenient interface for managing fan speeds and profiles. This integration not only enhances usability but also ensures that the system operates more efficiently and effectively.

Conclusion

The proposed solution in this paper comprises a microcontroller that connects to both the computer and the added fans within the computer case. A kernel module facilitates communication between the microcontroller and the computer, while a dedicated program enables the microcontroller to control the fans. This design ensures that it is both cost-effective and easy to implement, making it accessible for users seeking to enhance their system's cooling performance without incurring significant expenses.

The primary objective of this paper is to reduce the temperature and noise levels within the computer by adding more fans. The anticipated outcome wasn’t a noticeable decrease in the temperature of the computer's components, because of the placement of the added computer fan to the computer case.

Furthermore, it offers control over individual fans directly from the operating system, eliminating the need for multiple applications and providing a streamlined power user experience. This is particularly useful for computers facing cooling challenges that cannot accommodate additional fans within the current setup. Power users, in particular, will benefit from this solution, especially those utilizing high-powered hardware in unconventional or modified cases, such as servers housed in desktop cases or computers used for cryptocurrency mining.

Further Work

While the current implementation of the computer cooling project demonstrates significant improvements in system cooling and user convenience, there are several areas where further development could enhance the quality, reliability, and overall user experience. This section outlines potential future work to refine and expand the capabilities of the cooling solution.

Possible improvements to the system are:

- A more capable microcontroller
- An actual PCB design
- Higher quality soldering
- Code refactoring and modularity
- Ensure no memory leaks
- Ensure security of the entire system
- Improved logging
- Debug features

Some of the improvements are discussed below.

One of the primary areas for improvement is the selection of the microcontroller. The current microcontroller effectively manages the communication between the fans and the operating system; however, upgrading to a more powerful and feature-rich microcontroller could provide additional benefits. A better microcontroller would offer an enhanced peripheral support.

Transitioning from a prototype to a real product requires the design and fabrication of a dedicated printed circuit board (PCB). An actual PCB design would improve the reliability and durability of the hardware [19]. A custom PCB can be optimized for space, ensuring that the cooling solution fits into more various computer cases.

The software that drives the cooling solution is a critical component of its functionality. Refactoring the code will improve its structure, readability, and maintainability is an essential

step in further development. This process involves optimizing the existing codebase to eliminate redundancies, and improve readability.

Security is a critical consideration, particularly as the cooling solution interfaces directly with the operating system at the kernel level. Ensuring that the system is secure from potential vulnerabilities is very important. Usually, USB devices are treated by operating systems with trust and this can be weak link. [20] That’s why is important to accord attention to the implementation of devices who use USB and HID technologies. The Linux Foundation is a leader in security and quality and this must be shown also in Linux projects.

Enhancing the logging and debugging features of the software will facilitate easier troubleshooting and maintenance. Clearer and more comprehensive logging will provide insight into the system's operation, making it easier to identify and resolve issues.

References

[1] Moore, G. (1998). Cramming more components onto integrated circuits. Proceedings of the IEEE, 86(1), 82–85. https://doi.org/10.1109/jproc.1998.658762

[2] Roser, M., Ritchie, H., & Mathieu, E. (2024, February 1). What is Moore's Law? Our World in Data. Retrieved March 10, 2024, from https://ourworldindata.org/moores-law

[3] Hirasawa, T., Kawabata, K., & Oomi M. (2005). Evolution of the Heatsink Technology. Furukawa Review No. 27. Retrieved March 12, 2024, from https://www.furukawa.co.jp/review/fr027/fr27_06.pdf

[4] Thorén, J., & Widell, A. (2011). Development of liquid cooling for PCs. https://odr.chalmers.se/bitstream/20.500.12380/153096/1/153096.pdf

[5] Mungan, C. E. (2011). The Bernoulli equation in a moving reference frame. European Journal of Physics, 32(2), 517–520. https://doi.org/10.1088/0143-0807/32/2/022

[6] be quiet!. (2021, July 17). Inside be quiet!. Retrieved March 13, 2024, from https://www.bequiet.com/en/insidebequiet/3357

[7] Chang, J. Y., Yu, C. W., & Webb, R. L. (2000). Identification of minimum air flow design for a desktop computer using CFD modeling. Journal of Electronic Packaging, 123(3), 225–231. https://doi.org/10.1115/1.1348012

[8] Linton, R. L., & Agonafer, D. (1994). Thermal model of a PC. Journal of Electronic Packaging, 116(2), 134–137. https://doi.org/10.1115/1.2905501

[9] Bokhari, S. (1995). The Linux operating system. Computer, 28(8), 74–79. https://doi.org/10.1109/2.402081

[10] Jones, M. (2007, June 5). Anatomy of Linux kernel. Retrieved March 13, 2024, from https://developer.ibm.com/articles/l-linux-kernel/

[11] Bovet, D. P., Cassetti, M., & Oram, A. (2000). Understanding the Linux kernel. http://ermak.cs.nstu.ru/Understanding.Linux.Kernel.pdf

[12] De Goyeneche, J., & De Sousa, E. (1999). Loadable kernel modules. IEEE Software, 16(1), 65–71. https://doi.org/10.1109/52.744571

[13] Khoroshilov, A., Mutilin, V., Novikov, E., & Zakharov, I. (2015). Modeling environment for static verification of Linux kernel modules. In Lecture notes in computer science (pp. 400–414). https://doi.org/10.1007/978-3-662-46823-4_32

[14] Salzman, P. J., Burian, M., & Pomerantz, O. (2009). The Linux Kernel Module Programming Guide. CreateSpace.

[15] Noctua. Noctua PWM specifications white paper. Retrieved April 8, 2024, from https://noctua.at/pub/media/wysiwyg/Noctua_PWM_specifications_white_paper.pdf

[16] Intel Corporation. (2005, September). 4-Wire Pulse Width Modulation (PWM) Controlled Fans, revision 1.3. Retrieved April 8, 2024, from https://www.intel.com/content/dam/support/us/en/documents/intel-nuc/intel-4wire-pwm-fans-specs.pdf

[17] The Linux Kernel. Building External Modules. Retrieved May 5, 2024, from https://docs.kernel.org/kbuild/modules.html

[18] PlatformIO. What is PlatformIO. Retrieved May 5, 2024, from https://docs.kernel.org/kbuild/modules.html

[19] Dababneh, A., & Ozbolat I. T. (2015) Predictive reliability and lifetime methodologies for circuit boards, Journal of Manufacturing Systems, 37(1), 141-148. https://doi.org/10.1016/j.jmsy.2015.08.001

[20] Nicho, M., & Sabry I. (2023). Bypassing Multiple Security Layers Using Malicious USB Human Interface Device Retrieved June 1, 2024, from https://zuscholars.zu.ac.ae/cgi/viewcontent.cgi?article=6782&context=works

Glossary

A	**Ampere**
API	**Application Programming Interface**
CLI	**Command Line Interface**
CMM	**Cubic Meter per Minute**
CPU	**Central Processing Unit**
CRC	**Cyclic redundancy check**
GCC	**GNU Compiler Collection**
GND	**Ground**
GPU	**Graphical Processing Unit**
HID	**Human Interface Device**
IDE	**Integrated Development Environment**
I/O	**Input/Output**
LKM	**Loadable Kernel Module**
MTBF	**Mean Time Between Failures**
NPU	**Neural Processing Unit**
PC	**Personal Computer**
PCB	**Printed Circuit Board**
PCI-E	**Peripheral Component Interconnect Express**
PSU	**Power Supply Unit**
PWM	**Pulse Width Modulation**
RAM	**Random Access Memory**
RPM	**Rotations per Minute**

SATA	**Serial Advanced Technology Attachment**
SBC	**Single Board Computer**
TDP	**Thermal Design Power**
UC	**Use Case**
USB	**Universal Serial Bus**
V	**Volt**
VSC	**Visual Studio Code**