# Documentation

Multifunctional Mouse Driver in Linux for Boosting User Productivity

## Introduction

Our project aims to develop a specialized device driver for the Linux operating system, tailored for a multifunctional mouse equipped with four additional buttons alongside standard left and right click functionalities. These extra buttons serve distinct purposes, including Morse code input, backspace functionality, and tab-switching. By serving as a crucial intermediary layer between the mouse hardware and the Linux kernel, our driver interprets input from these specialized buttons, translating them into corresponding actions within the operating system. With a focus on user-friendliness, our driver transforms a standard mouse into a powerful tool, catering to the needs of both professionals and the general public.

## A brief explanation of the logic behind the provided codes:

- **Multifunctional Mouse Device Driver:**

❖ The driver initializes and manages a USB mouse with additional buttons. It registers itself with the USB subsystem to handle plug and unplug events.

❖ When the mouse is plugged in, it allocates memory, sets up interrupt handling, and registers the mouse with the input subsystem. The driver interprets USB interrupt requests (URBs) to detect mouse movements and button clicks.

❖ It provides file operations for user-space interaction, allowing read and write operations on the mouse device file. When the mouse is unplugged, the driver cleans up resources and unregisters the device.

- **Morse Code Functionality:**

❖ The driver includes support for interpreting Morse code input from two additional buttons on the multifunctional mouse. Morse code input is translated into corresponding alphanumeric characters and special symbols.

❖ When Morse code input is detected, the driver converts it into English text using a dictionary, and writes the result to an output file.

❖ Users can utilize Morse code input for specialized interactions or communication purposes.

❖ The driver stops trying to detect Morse Code characters when the middle scroll button is pressed, and prints whatever has been written into the file so far onto the terminal window.

❖ Once the contents are printed onto the window, the file is flushed and we can start the procedure all over again.

❖ We have a third dedicated space button on the mouse to type out a space character to separate 2 distinct Morse code characters from each other.

- **Tab-Switching Functionality:**

❖ The driver includes support for tab-switching triggered by one of the additional buttons on the mouse.

- ❖ When the tab-switching button is pressed, the driver simulates keyboard shortcuts to switch between tabs or perform copy-paste actions in the operating system.
- ❖ This functionality enhances user productivity by providing quick access to tab-switching functionality directly from the mouse.
- ❖ When the left tab-switching button is pressed, it sends an event which changes the value of the Ctrl, Shift, and Tab keys momentarily to 1 in the input file and then sets them back to zero.
- ❖ The same procedure is executed but only for the Ctrl and Tab keys in case the right tab-switching button is pressed.

- **User Test Cases:**

- ❖ The test case allows users to interact with the multifunctional mouse driver from user space. Users can open the mouse device file and perform read and write operations on it.

- ❖ For read operations, the test case interprets USB input events and prints button states, including left, right, and middle mouse buttons. For write operations, users can input data to simulate interaction with the mouse.
- ❖ For this, we use the last element of the data array in our mouse struct. This element, a character value, contains the status of all the buttons present on our mouse.
- ❖ Using the hexadecimal masks 0x01, 0x02, 0x04, 0x08, and 0x10, we isolate the status of every individual button on the mouse and then print if that button has been pressed or not.
- ❖ This can be tested out by randomly pressing any of the buttons on the mouse.

## Integration and Usage:

- The driver translates hardware-specific actions into kernel-level operations, while the test case allows users to interact with the driver from user space.

- Users can compile and load the driver module into the Linux kernel using appropriate build commands.
- They can then use the provided user test case to interact with the driver and test its functionality with a multifunctional USB mouse.
- Can extend the Morse code functionality to support additional characters or symbols, or enhance the tab-switching functionality with additional keyboard shortcuts or actions.

Overall, the inclusion of Morse code and tab-switching functionalities enhances the versatility and usability of the multifunctional USB mouse driver, providing users with additional tools for efficient interaction and productivity in the Linux environment.

## Running the code:

- Go to the respective driver directory which contains the code for the driver. The Makefile needs to be run as root user so we need to use sudo.

- First compile the driver using the make command. This should create a driver module object which can then be used to communicate with the mouse

  -sudo make

- Then, in order to insert our mouse driver module, we first need to remove the standard driver module for usb mice (usbhid).

  -sudo rmmod usbhid

- This removes the linux mouse module

- Now, we can insert our driver to access all the functionality of the driver.

  -sudo insmod adv_mousedriver.ko

- Now that we have our device module ready in the kernel space to communicate with the hardware, we create a device file to which all the data received from the mouse is written to. For this, run the following command. Make sure the major number is correct by using sudo dmesg|tail.

  -sudo mknod /dev/mymousedev c 240 0

  -If said file has already been created, you can ignore this part.

- Once the device file is created, we need to give it the required permissons. For this run

  -sudo chmod 777 /dev/mymousedev

  -If the perms have already been given, you can ignore this part.

- Now we just need to compile and run our C program that handles the inputs

  -gcc -o exec usertest.c //for morsecode

  -sudo ./exec

- Once you are done using the driver, you can remove it and insert back the standard mouse driver module

  -sudo rmmod adv_mousedriver

  -sudo modprobe usbhid

Note run sudo make clean to get rid of the existing files