Universal Serial Bus Device Automount

1 Introduction

1.1 Context

Automount is a feature that allows the system to detect a USB device without having the user input commands into the terminal. Our project was to create a program that would not only automatically detect the USB, instead would provide a choice to the user to mount the USB or not.

We implemented the mount and unmount features using udev[9] rules and the GUI elements using zenity[7] and libnotify[8]. Shell scripting was used to call zenity[7] and libnotify[8] commands.

1.2 Problem Statement

One of the problems we have seen as a beginner Windows to Linux user is figuring out how to mount USB devices and where to find them.

When we began using Linux and we needed to backup some information into a usb device, it wasn't the most straightforward solution. Coming from windows, we expected to see a dialogue box that informed us whether a USB Device has been connected or not and also be able to see the folder in the C directory.

However, what should have been a simple solution became complicated. We had to perform research on how to mount USB Devices and figure out where the USB devices were located in order to make sure they had actually been recognized by the system. This is time consuming if you need to transfer files immediately. We then figured out we could also automount the USB, however, we had install additional packages (deconf) and make sure automount was enabled in nautilus . We also realized you are not notified if the USB has been mounted which can be confusing for beginners.

How we solved this problem is described in the following section.

1.3 Result

We decided to overcome the problem stated above by using udev[9] rules to automount and unmount the USB device and when it is recognized, provide a dialogue box (zenity) that allows the user to choose whether they want to mount the USB or not. If they choose 'Yes' we inform the user that the USB has been detected through a notification (libnotify) and if they choose 'No' , the user is notified that the system failed to detect the USB (libnotify). Once detected, a file with the USB name is automatically created in the media drive. We also notify the user when the USB is removed (libnotify).

This is elaborated in Section 3.

1.4 Outline

Section 2 presents background information relevant to this project.
Section 3 describes in detail the obtained result.

The result is evaluated in Section 4.
We conclude with Section 5.

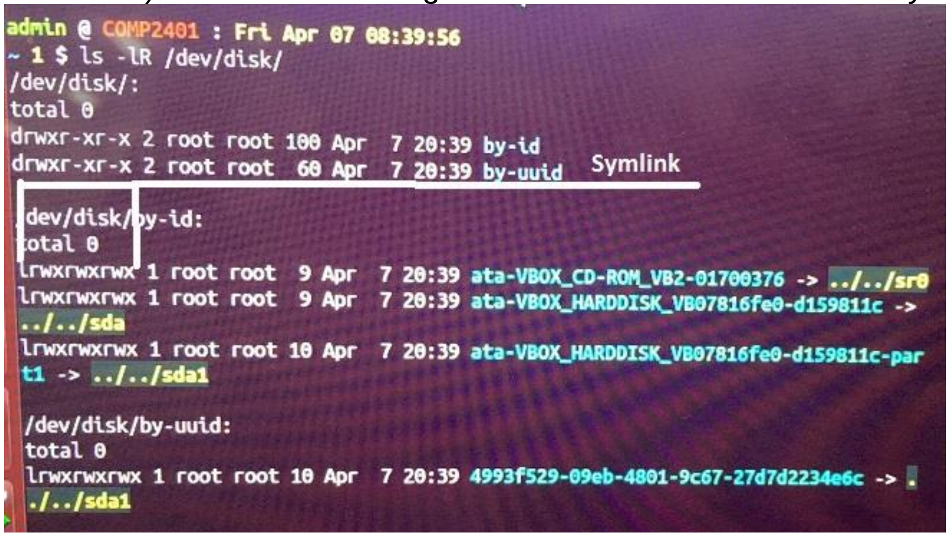
2 Background

To understand automounting implementation and capabilities the following concepts in operating systems need to be explored: device recognition, storage and udev[9], blkids [4], mounting and mount flags.

Device recognition, storage and udev

In the earlier versions of Linux, devices files for mounting devices were created manually by the system administrator. These were added to /dev directory which handles device files for device types such as block (e.g. USB drives, hard-drives)and character devices(e.g tapes). Shortly after, a script called MAKEDEV was embedded into the /dev directory that assisted in creating device files by assigning the conventional device names and numbers and diminished the need for lookup. However this task was laborious and MAKEDEV sometimes could not create the appropriate device files. In addition, the /dev directory statically stored device files for all devices mounted without organization or cleanup of outdated versions.

With the 2.6.2 version of the Linux kernel, a system script called udev[9] was introduced which worked alongside the sysfs system to dynamically allocate device nodes. This meant that device files could then be stored and removed from the /dev directory as they were mounted and unmounted. It also enabled device file storage inside /dev in a more organized fashion using symbolic links or symlinks (files that point to other files in the system as depicted in Figure 1, symlinks are identifiable with the letter ‘l’ at the beginning of the line in the long format of running the the ls command). This method of organization made it easier to identify the device.



```
admin @ COMP2401 : Fri Apr 07 08:39:56
~ 1 $ ls -lR /dev/disk/
/dev/disk/:
total 0
drwxr-xr-x 2 root root 100 Apr  7 20:39 by-id
drwxr-xr-x 2 root root  60 Apr  7 20:39 by-uuid

/dev/disk/by-id:
total 0
lrwxrwxrwx 1 root root  9 Apr  7 20:39 ata-VBOX_CD-ROM_VB2-01700376 -> ../../sr0
lrwxrwxrwx 1 root root  9 Apr  7 20:39 ata-VBOX_HARDDISK_VB07816fe0-d159811c ->
../../sda
lrwxrwxrwx 1 root root 10 Apr  7 20:39 ata-VBOX_HARDDISK_VB07816fe0-d159811c-par
t1 -> ../../sda1

/dev/disk/by-uuid:
total 0
lrwxrwxrwx 1 root root 10 Apr  7 20:39 4993f529-09eb-4801-9c67-27d7d2234e6c -> .
./../sda1
```

Figure 1: Result of listing the disk directory in /dev. Devices are organized and stored using symlinks

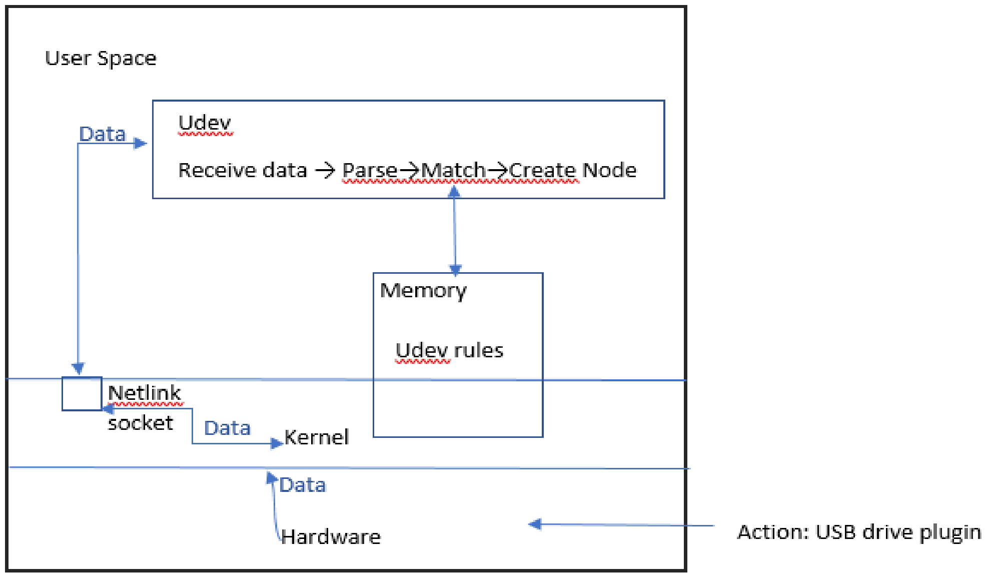
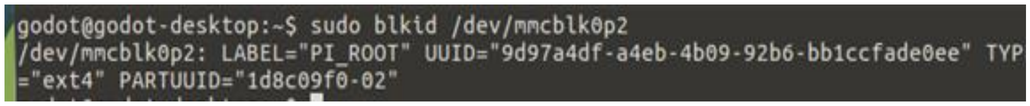


Figure 2: Udev interaction with kernel and memory on device mount

Figure 2 shows how udev[9] interacts with the kernel to store devices in memory. Udev, in the userspace listens to the netlink socket in the kernel for events. When a device such as a USB is plugged in, the kernel sends device data, such as serial numbers, through the netlink socket to the udev[. Udev will parse the data and match it with its rules, which are stored in memory, to create a device node and associating symlinks for the device based on the rules it has. It then uses modalias to load the appropriate driver for the device. If any changes are made to the device or rules, udev will update accordingly.

Blkid



```
godot@godot-desktop:~$ sudo blkid /dev/mmcblk0p2
/dev/mmcblk0p2: LABEL="PI_ROOT" UUID="9d97a4df-a4eb-4b09-92b6-bb1ccfade0ee" TYP
="ext4" PARTUUID="1d8c09f0-02"
```

Figure 3: Blkid print output

Block identification or block ids are a command line utility that is intended for locating and printing block device information such as the attributes of the device. In order to run the blkid[4] command, root privileges are required. Figure 3 shows the print output from running the blkid[4] command in terminal after device mounting. The significance of blkid[4] is that it can display attributes such as the name or label of the device, its location and universally unique identifier (UUID) which are also serialized in udev when device is loaded. Pertaining to automounting, this information can be parsed and sent in the mount command to udev[6] to load a specific device. While the both device label and UUID can be used for mounting, UUID is prefered as it is unique to the device, therefore, in the event that two of the same labelled devices are simultaneously connected there is no confusion.

Mount and mount flags

Mount identifies and places the filesystem of a block device in a directory at a mount point. It provides the kernel with data on an access method to the filesystem on the device (i.e. driver) and root directory access to the system. Prior to mounting the kernel should have the filesystem driver already configured into the device. The sys/mount.h header file also provides options on access privileges or environments conditions to mounting a block device. These are the mount flags and the can be used to specify whether access to the device is read-only (MS_RDONLY), available to most users (NO_SUID), forbids file execution (MS_NOEXEC) or updates (MS_MANLOCK).

3 Result

Our final result is more or less what we intended it to be. Our main goal when we began the project was to create the automount feature such that it is a lot simpler to use and understand for a user switching from windows to linux.

We made ours simpler to understand because unlike the conventional automount feature in Linux, users are actually notified and prompted whether they want to mount the usb or not.

Our program works is in the following way:

If the usb is plugged in without running our program, the user will notice that they are not prompted to mount the usb, nor the does a folder get created for the USB device.

Figure 1: When you don’t run our program, USB is not mounted

In order to run the program, you need to first install the following packages:

libudev.h[9], libnotify[8] and some operating systems may need zenity[4] and gcc as well.

Libudev[9], gcc, zenity[7]:
These can be acquired by running

```
$ sudo apt-get install libudev -dev
$ sudo apt-get install gcc
$ sudo apt-get install zenity
```

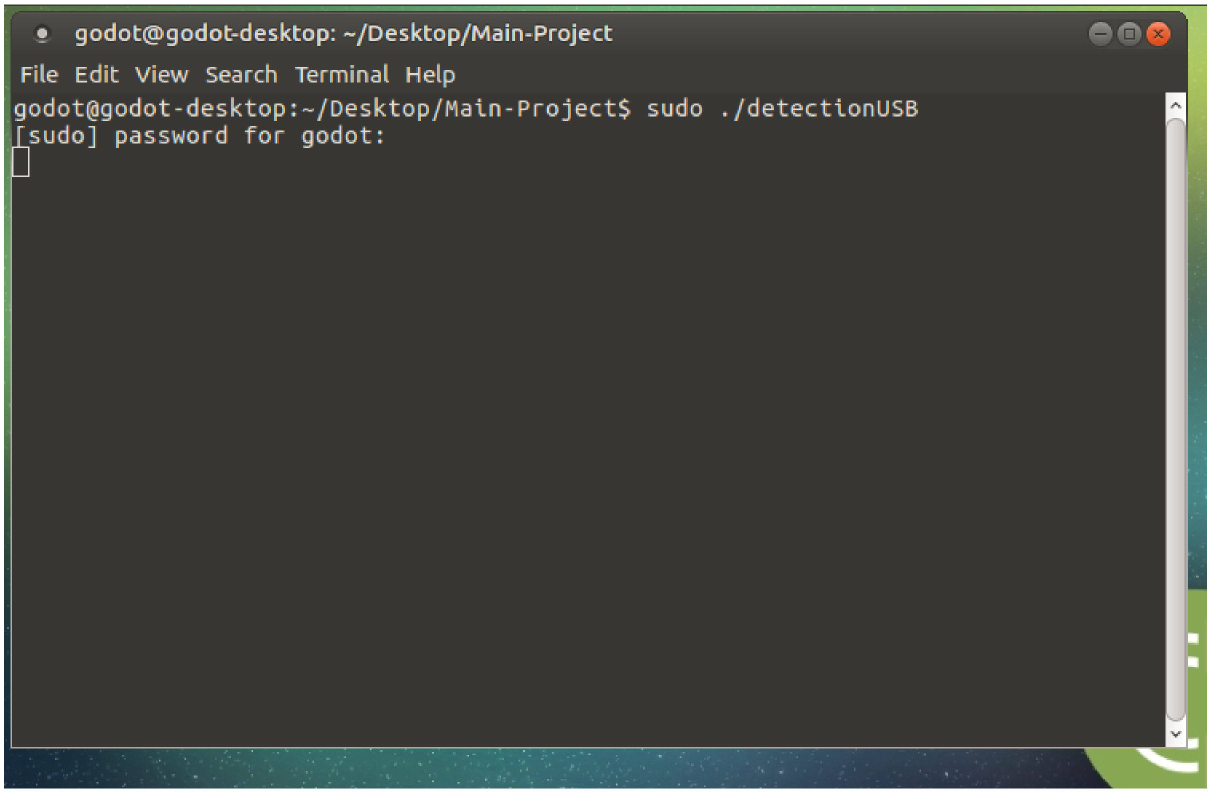
Libnotify[8]:

```
$ npm install libnotify
```

We then move into the correct directory (Main Project) and then run the program by typing in

```
$ ./detectionUSB
```

into the terminal.


godot@godot-desktop: ~/Desktop/Main-Project
File Edit View Search Terminal Help

```
godot@godot-desktop:~/Desktop/Main-Project$ sudo ./detectionUSB
[sudo] password for godot:
```

Fig2: Running the program

First, it blocks the usb and looks at the directory and checks if directory is valid and if udev is installed in the computer. If it isn’t installed, it doesn’t run and we receive an error message. If it is found, it writes the udev rules in /etc/udev.rules.d which is a location within udev. The program makes sure to reload the rules so that i can be run without rebooting the program every single time.

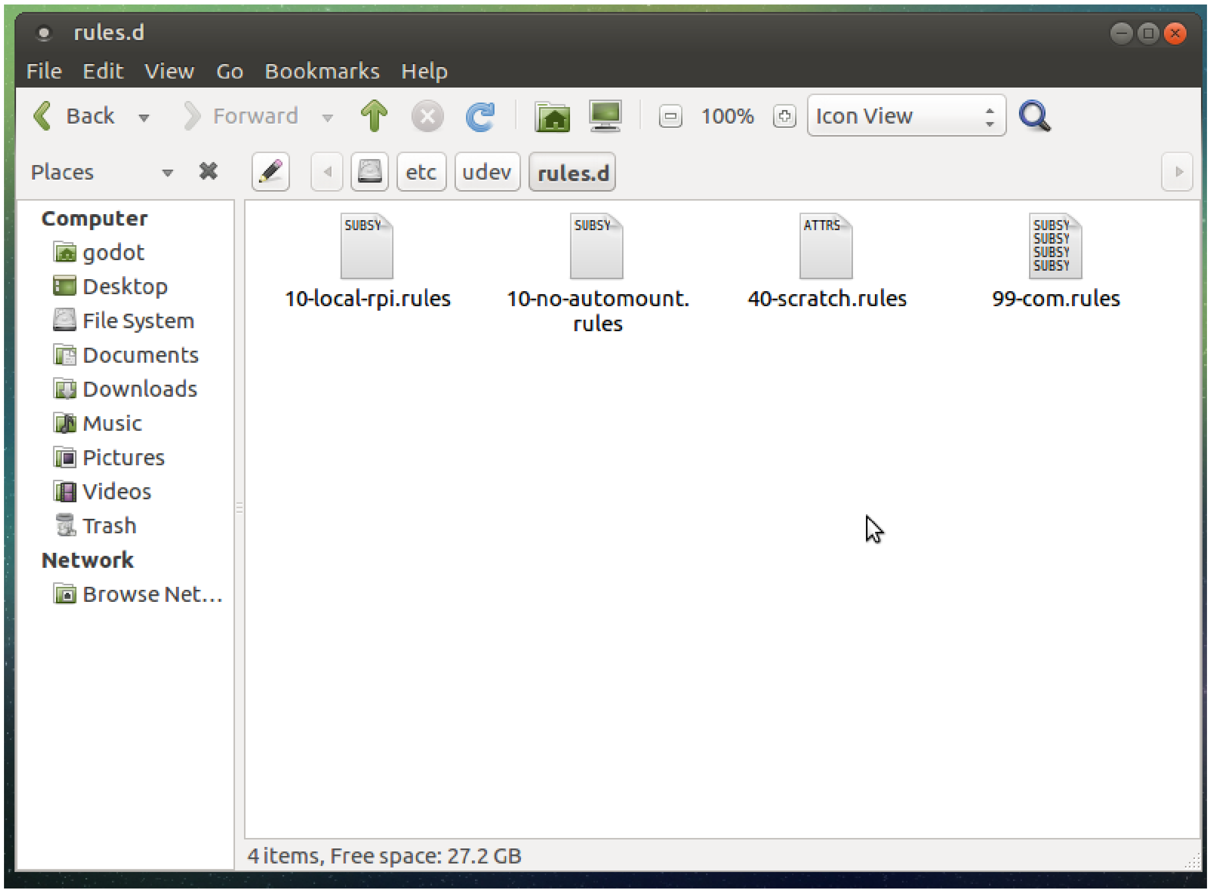
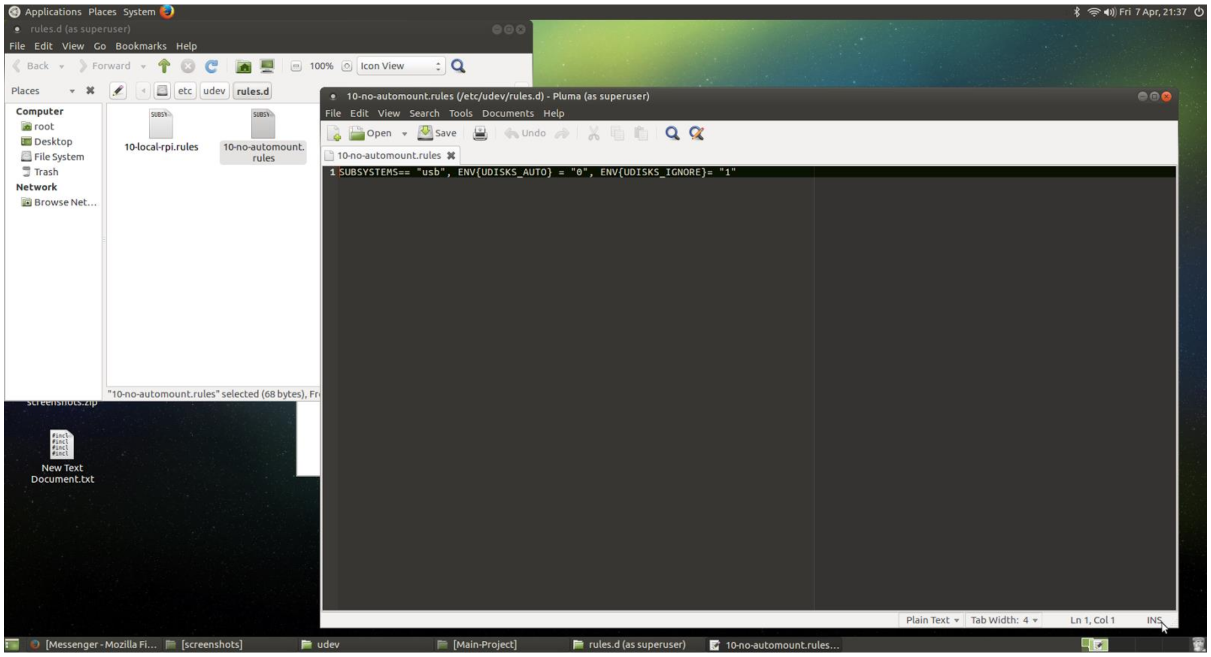


Fig3: Location of udev rules

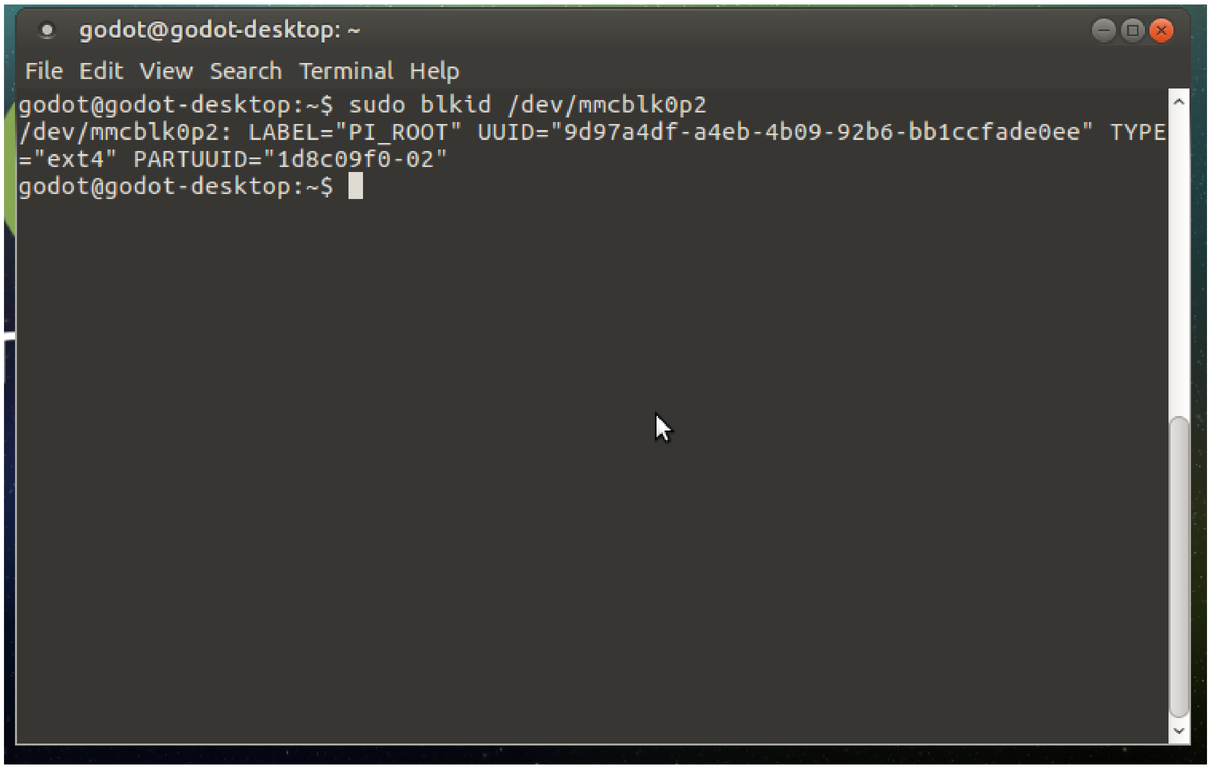

10-no-automount.rules (/etc/udev/rules.d) - Pluma (as superuser)
File Edit View Search Tools Documents Help
Open Save Undo
10-no-automount.rules
Plain Text Tab Width: 4 Ln 1, Col 1 INS

```
SUBSYSTEMS== "usb", ENV{UDISKS_AUTO} = "0", ENV{UDISKS_IGNORE}= "1"
```

Fig 4: udev rules written

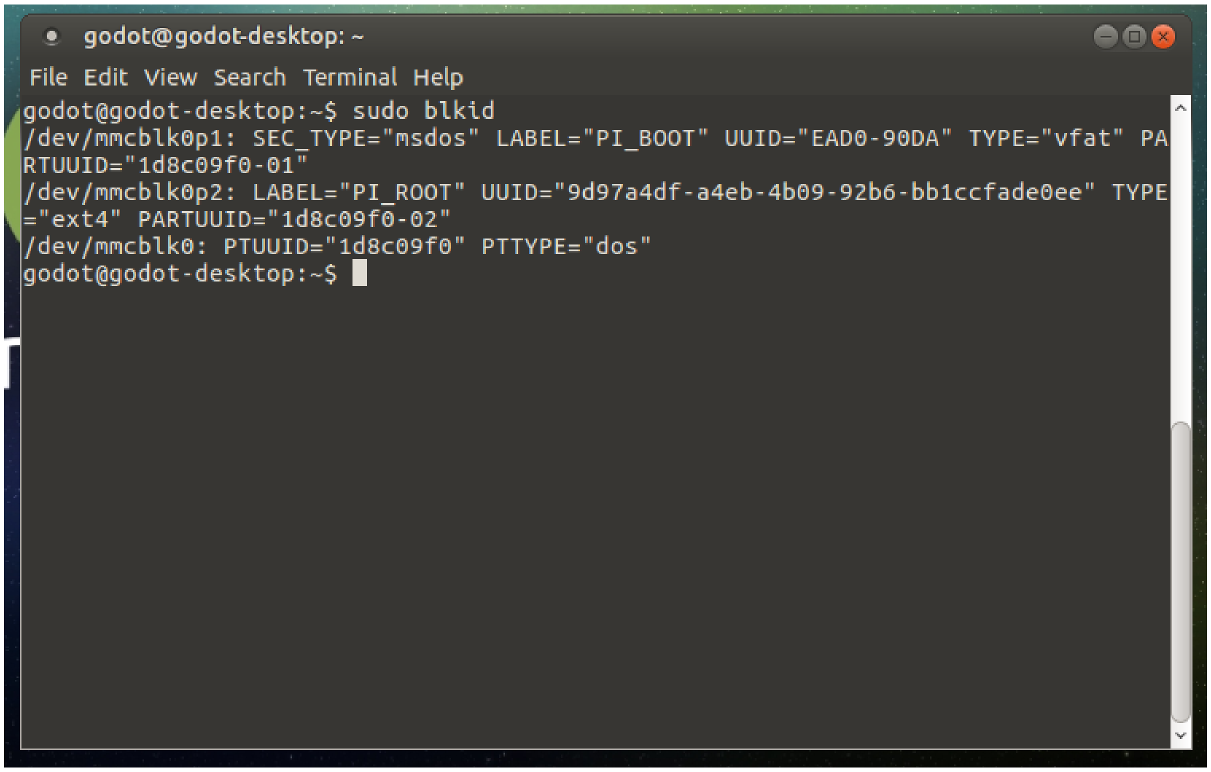
Next, memory is allocated for every place in the array (100). This is one of the reasons why our program would crash after 100 usb devices. How we intend to fix it is stated in Section 5.3.

We then use blkid[4] from the terminal which creates a text file called USB.txt with the information related to the USB drive (UUID, Label, etc.) The program then looks for the label, UUID, Type and Location where the usb folder will be created. Currently the USB drive is write protected by default, however we will need to remove that in the future. It then makes a directory if it doesn't already exist and if it does, then do nothing.



```
godot@godot-desktop:~$ sudo blkid /dev/mmcblk0p2
/dev/mmcblk0p2: LABEL="PI_ROOT" UUID="9d97a4df-a4eb-4b09-92b6-bb1ccfade0ee" TYPE
="ext4" PARTUUID="1d8c09f0-02"
godot@godot-desktop:~$
```

Fig 5: Calling blkid from the terminal. Shows UUID, Label and Type of the storage devices.

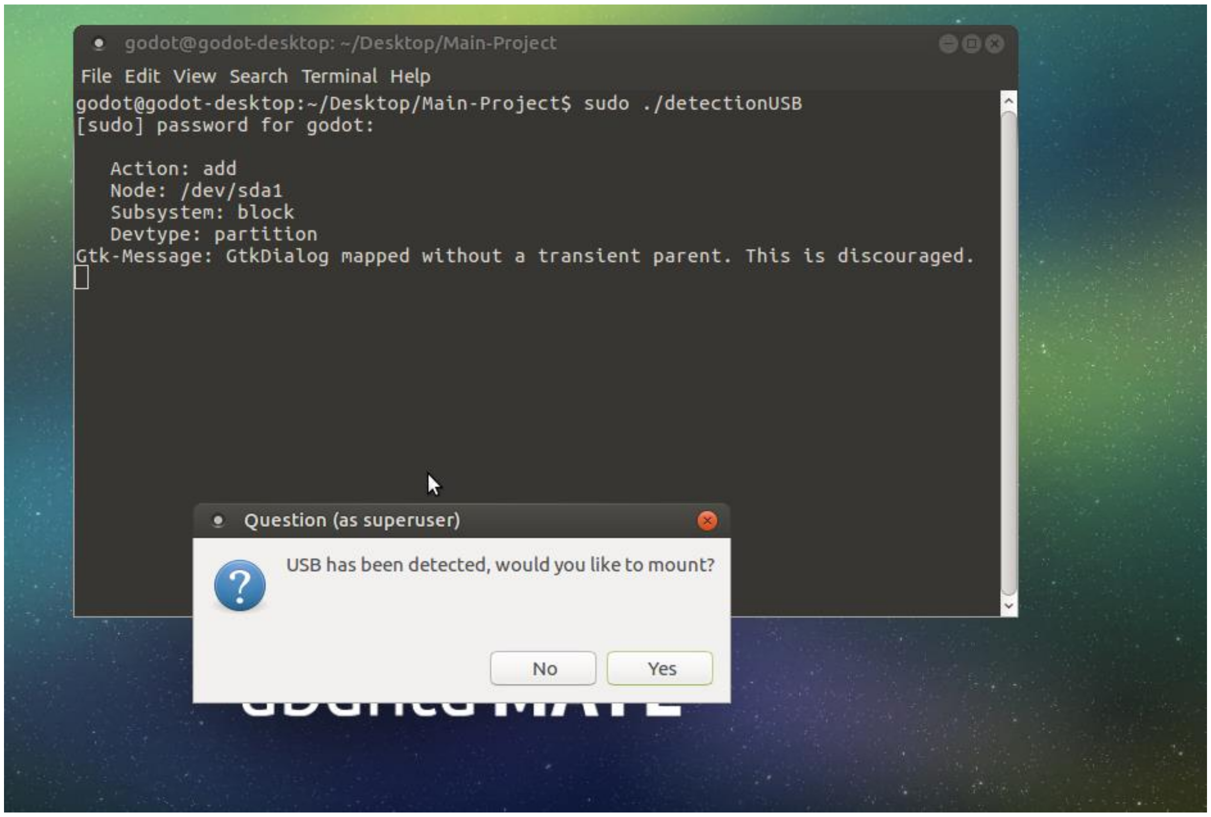


```
godot@godot-desktop:~$ sudo blkid
/dev/mmcblk0p1: SEC_TYPE="msdos" LABEL="PI_BOOT" UUID="EAD0-90DA" TYPE="vfat" PA
RTUUID="1d8c09f0-01"
/dev/mmcblk0p2: LABEL="PI_ROOT" UUID="9d97a4df-a4eb-4b09-92b6-bb1ccfade0ee" TYPE
="ext4" PARTUUID="1d8c09f0-02"
/dev/mmcblk0: PTUUID="1d8c09f0" PTTYPE="dos"
godot@godot-desktop:~$ 
```

Fig 6: USB connected, BLKID called again.

The usb is then mounted, the user sees the action in the terminal. While this is happening, the information related to USB (dev location and media location) is being added to the USB array.

The user is then prompted with a dialogue box that asks them if they want to mount the usb or not. If they click on yes, the folder is created and you can see it in the media directory. The user also receives a notification that informs them that the USB has been detected.

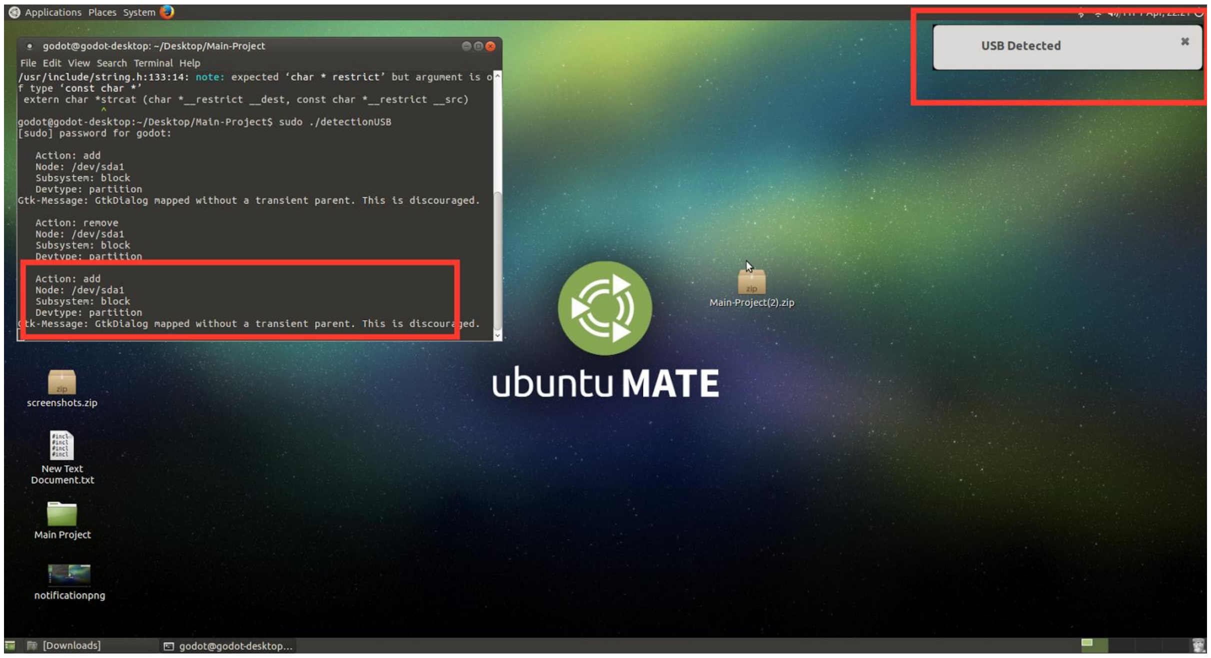

godot@godot-desktop: ~/Desktop/Main-Project
File Edit View Search Terminal Help
Question (as superuser)
No
Yes

```
godot@godot-desktop:~/Desktop/Main-Project$ sudo ./detectionUSB
[sudo] password for godot:

   Action: add
   Node: /dev/sda1
   Subsystem: block
   Devtype: partition
Gtk-Message: GtkDialog mapped without a transient parent. This is discouraged.
```

USB has been detected, would you like to mount?

Fig 7: Prompts the user



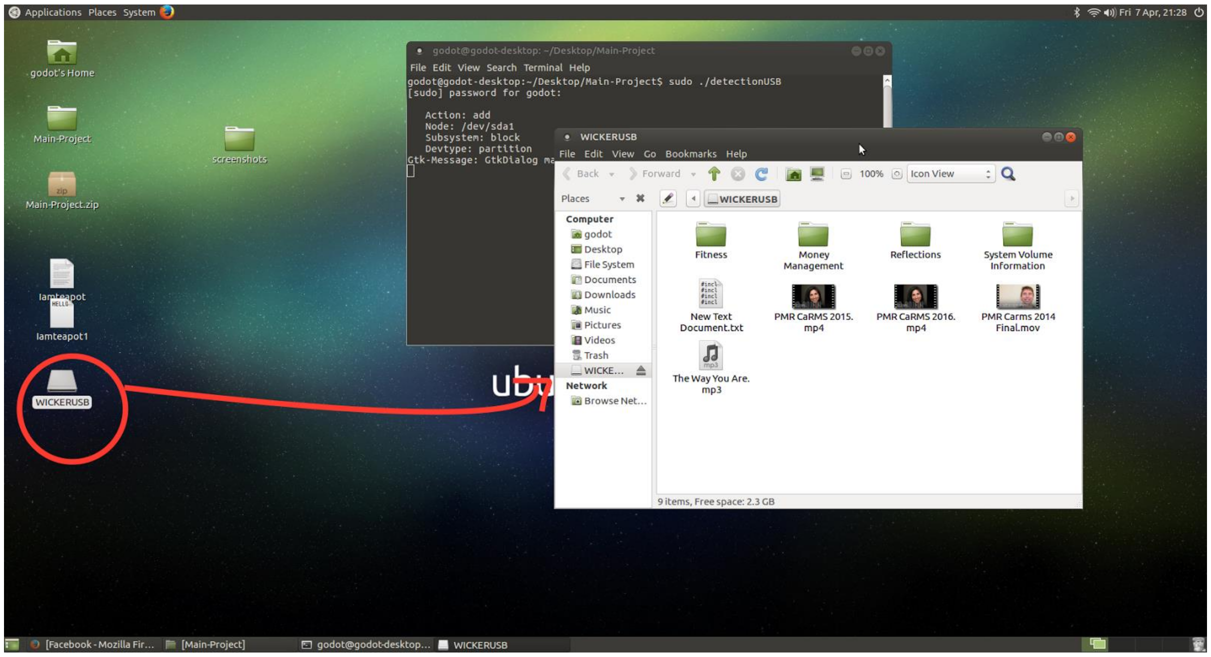
```
/usr/include/string.h:133:14: note: expected ‘char * restrict’ but argument is o
f type ‘const char *’
 extern char *strcat (char *__restrict __dest, const char *__restrict __src)
              ^
godot@godot-desktop:~/Desktop/Main-Project$ sudo ./detectionUSB
[sudo] password for godot:

   Action: add
   Node: /dev/sda1
   Subsystem: block
   Devtype: partition
Gtk-Message: GtkDialog mapped without a transient parent. This is discouraged.

   Action: remove
   Node: /dev/sda1
   Subsystem: block
   Devtype: partition

   Action: add
   Node: /dev/sda1
   Subsystem: block
   Devtype: partition
Gtk-Message: GtkDialog mapped without a transient parent. This is discouraged.
```

Fig 8: If selected Yes



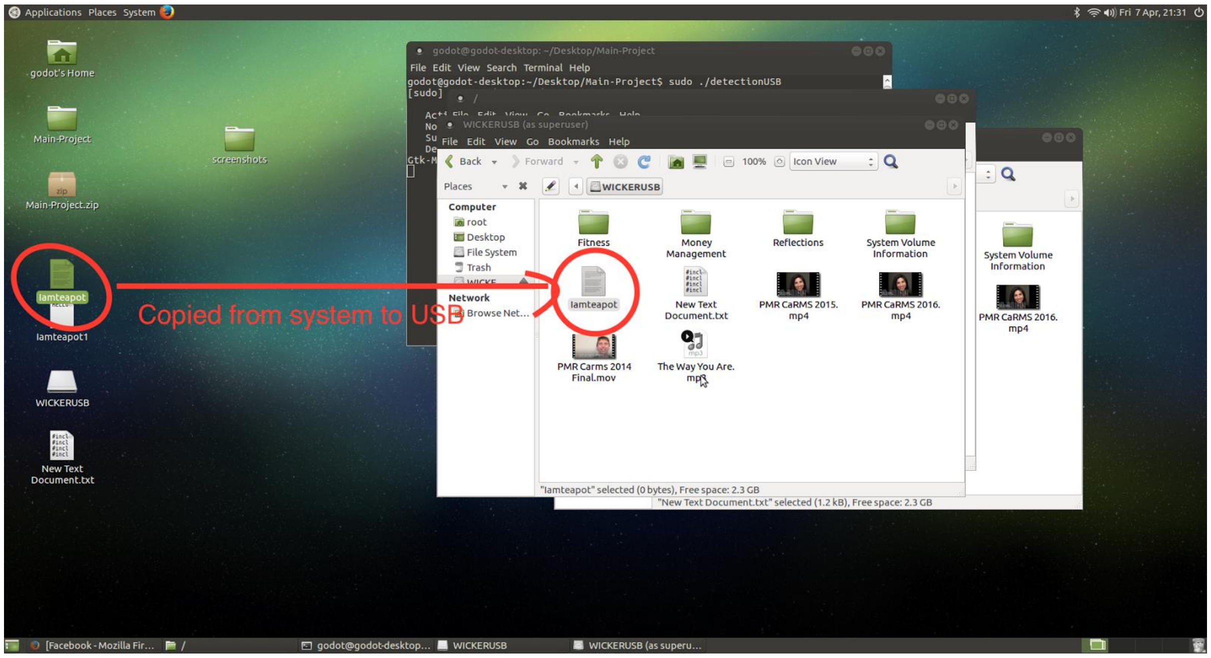
```
godot@godot-desktop:~/Desktop/Main-Project$ sudo ./detectionUSB
[sudo] password for godot:

   Action: add
   Node: /dev/sda1
   Subsystem: block
   Devtype: partition
Gtk-Message: GtkDialog ma
```

9 items, Free space: 2.3 GB

Fig 9: USB folder opened

You can transfer files from the USB to the system and copy into the USB. However, you will not be able to create folders in the USB. (write-protected). You will also need to be logged in as a superuser in order to be able to copy files into the usb device. If you're not a super user you will receive an error message when you try to copy a file into the usb.

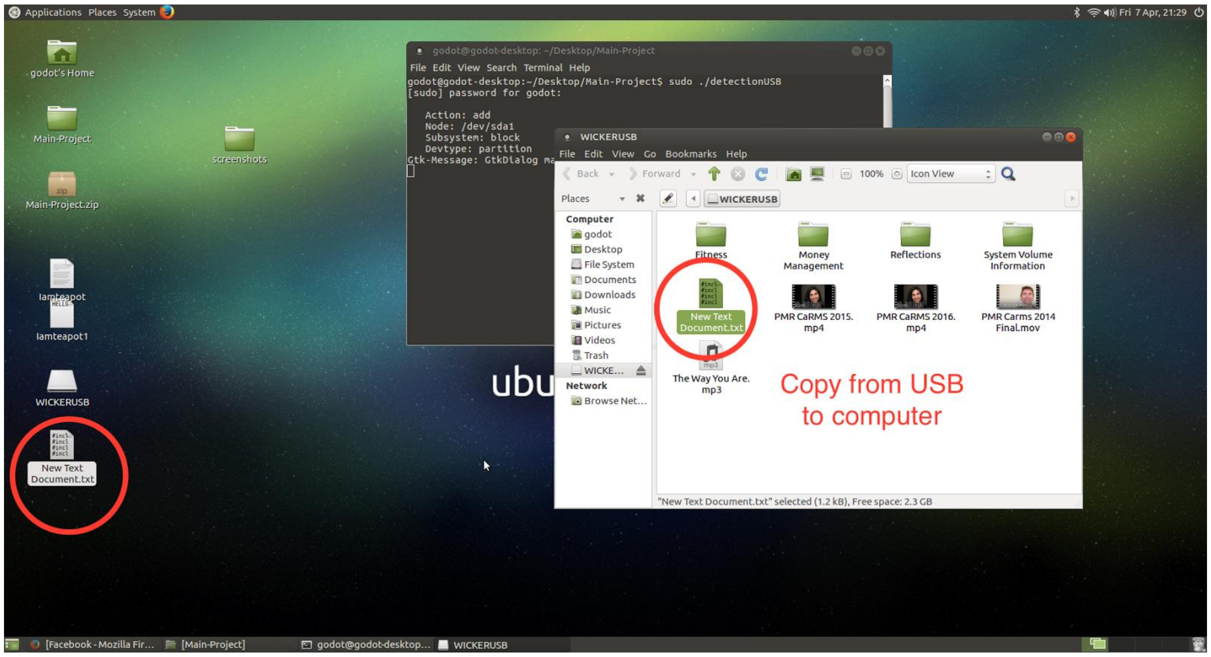


```
godot@godot-desktop:~/Desktop/Main-Project$ sudo ./detectionUSB
[sudo]
```

"Iamteapot" selected (0 bytes), Free space: 2.3 GB

"New Text Document.txt" selected (1.2 kB), Free space: 2.3 GB

Figure 10 : Logged in as superuser, copy from system to USB

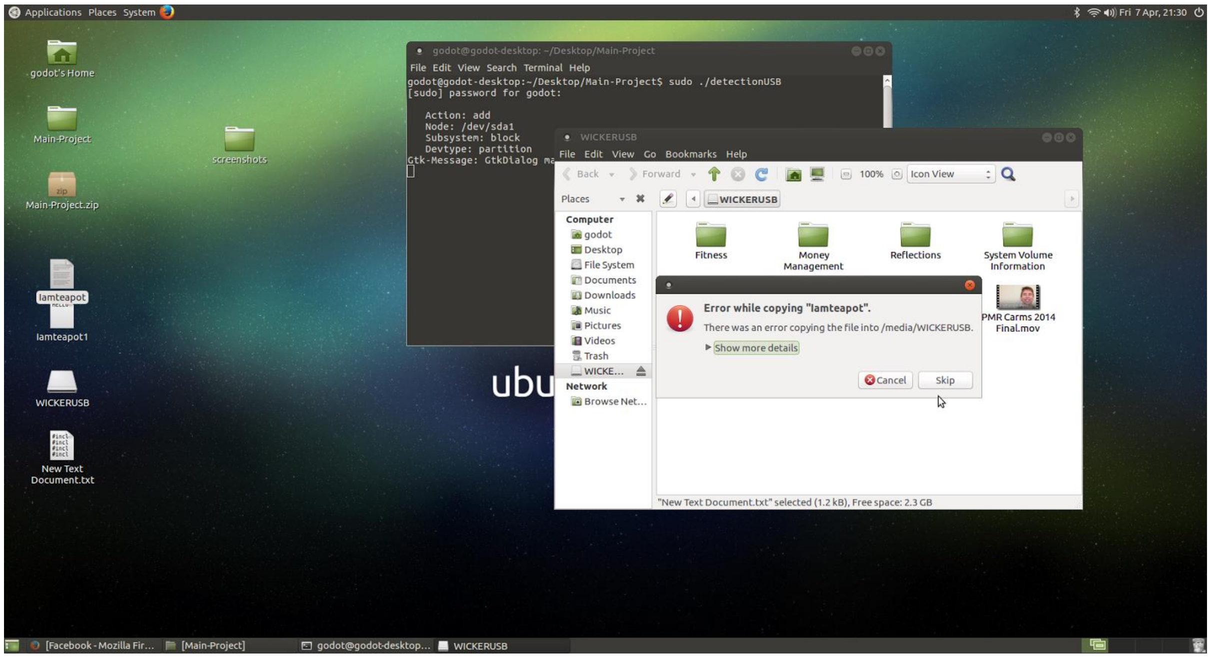


```
godot@godot-desktop:~/Desktop/Main-Project$ sudo ./detectionUSB
[sudo] password for godot:

   Action: add
   Node: /dev/sda1
   Subsystem: block
   Devtype: partition
Gtk-Message: GtkDialog ma
```

"New Text Document.txt" selected (1.2 kB), Free space: 2.3 GB

Fig 11: Copied from USB to system



```
godot@godot-desktop:~/Desktop/Main-Project$ sudo ./detectionUSB
[sudo] password for godot:

   Action: add
   Node: /dev/sda1
   Subsystem: block
   Devtype: partition
Gtk-Message: GtkDialog ma
```

Error while copying "Iamteapot".

There was an error copying the file into /media/WICKERUSB.

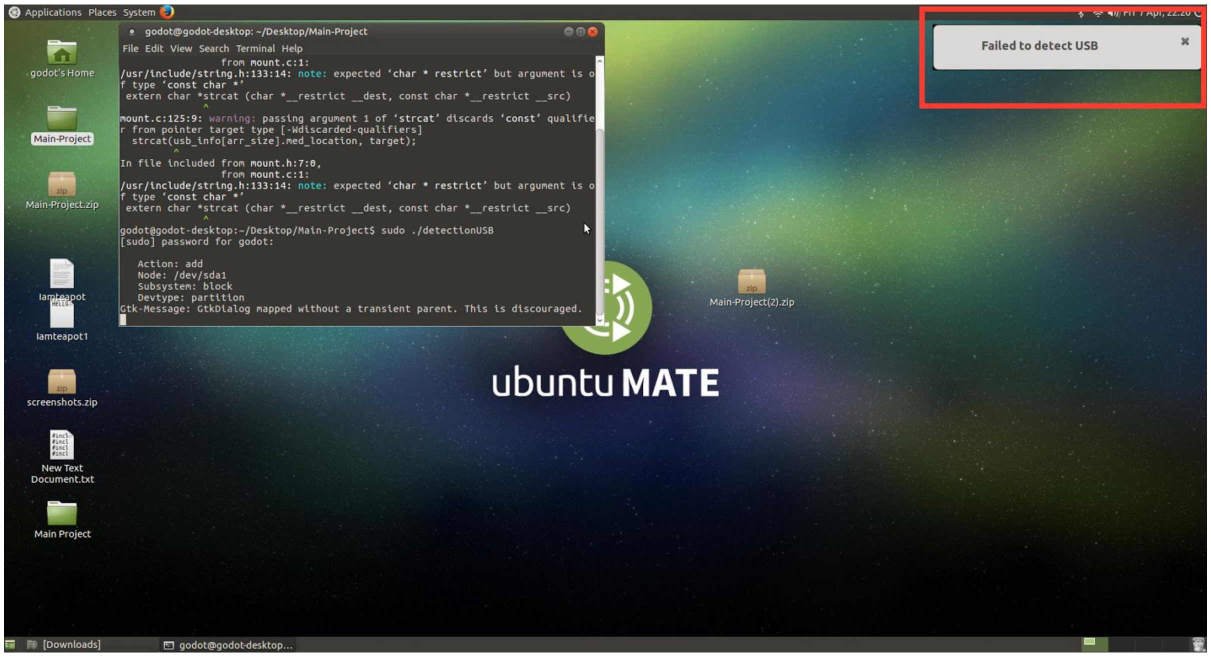
Show more details

Cancel Skip

"New Text Document.txt" selected (1.2 kB), Free space: 2.3 GB

Figure 12: Logged in as non-superuser, trying to copy from system to usb

If the user selected ‘No’, the folder does not get created and the user receives a notification that alerts them that the program failed to detect the USB.



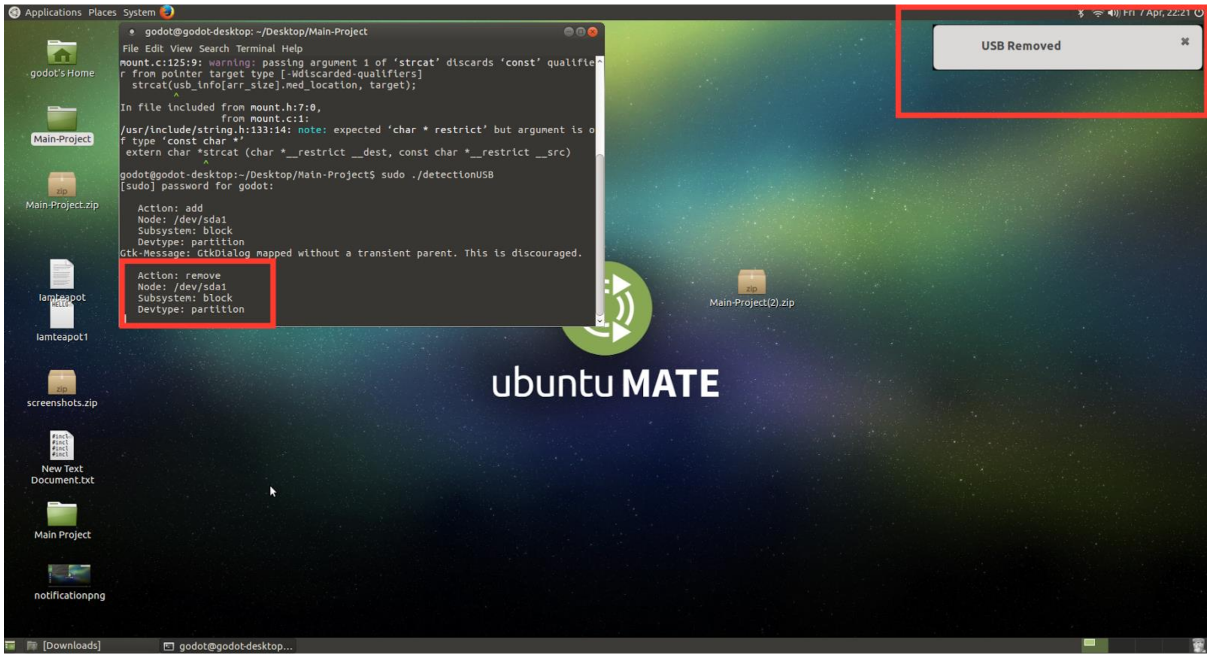
```
godot@godot-desktop: ~/Desktop/Main-Project
                 from mount.c:1:
/usr/include/string.h:133:14: note: expected ‘char * restrict’ but argument is o
f type ‘const char *’
 extern char *strcat (char *__restrict __dest, const char *__restrict __src)
              ^
mount.c:125:9: warning: passing argument 1 of ‘strcat’ discards ‘const’ qualifie
r from pointer target type [-Wdiscarded-qualifiers]
  strcat(usb_info[arr_size].med_location, target);
        ^
In file included from mount.h:7:0,
                 from mount.c:1:
/usr/include/string.h:133:14: note: expected ‘char * restrict’ but argument is o
f type ‘const char *’
 extern char *strcat (char *__restrict __dest, const char *__restrict __src)
              ^
godot@godot-desktop:~/Desktop/Main-Project$ sudo ./detectionUSB
[sudo] password for godot:

   Action: add
   Node: /dev/sda1
   Subsystem: block
   Devtype: partition
Gtk-Message: GtkDialog mapped without a transient parent. This is discouraged.
```

Failed to detect USB

Fig 13: If selected No

When the user removes the USB, the user can see information about the action in the terminal. The USB folder is deleted and they receive a notification that notifies them the USB device has been removed. The device is also dereferenced to save memory and prevent memory leaks.



```
godot@godot-desktop: ~/Desktop/Main-Project
mount.c:125:9: warning: passing argument 1 of ‘strcat’ discards ‘const’ qualifie
r from pointer target type [-Wdiscarded-qualifiers]
  strcat(usb_info[arr_size].med_location, target);
        ^
In file included from mount.h:7:0,
                 from mount.c:1:
/usr/include/string.h:133:14: note: expected ‘char * restrict’ but argument is o
f type ‘const char *’
 extern char *strcat (char *__restrict __dest, const char *__restrict __src)
              ^
godot@godot-desktop:~/Desktop/Main-Project$ sudo ./detectionUSB
[sudo] password for godot:

   Action: add
   Node: /dev/sda1
   Subsystem: block
   Devtype: partition
Gtk-Message: GtkDialog mapped without a transient parent. This is discouraged.

   Action: remove
   Node: /dev/sda1
   Subsystem: block
   Devtype: partition
```

USB Removed

Fig 14: USB Removed

4. Evaluation

The way we performed the evaluation of our program is by selecting 6 test subjects, each person with varying amounts of exposure to Linux. We then asked them to test our program. We provided a usb drive, a laptop where the program was already installed in a virtual box and the instructions to running our program and running the automount feature in ubuntu.

After the test subjects were done testing the project, they were provided with a questionnaire that asked them questions on how difficult/easy it was for them to use the regular, built-in automount feature in Linux and how difficult/easy it was for them to use our program to mount the usb.

Our results varied according to the level of exposure to Linux users. For users who have already used linux before and are well versed with it, didn’t feel much of a difference. They appreciated the GUI elements and the notification feature, however the level of difficulty/easy was not that different.

However, for the subjects who had zero to very limited/beginner level of exposure to the linux system felt the automount feature was similar to the one in Windows, therefore easier to navigate and use.

Therefore, we believe that we have succeeded what we wanted to do with our program which was create an automount feature similar to that of windows so that beginner Windows to Linux users will not have too much difficulty navigating to the usb drive.

Moreover, since we weren’t able to implement the security features but is something we will work on in the future, we also asked questions about how they would feel about having the security features in the usb, such as, password protecting the usb drive and limiting the types of files to be transferred.

It seemed that password protection is something everyone felt wasn’t extremely necessary however a very nice security feature that would be appreciated and used, however ideas on restriction of files to be transferred was returned with mixed reviews. Some people asked the question ‘What if I need to transfer such a file, would I be able to have a way to override the default system?’, which got us rethinking the feature. This will be talked about more in the following sections.

In conclusion, according to the questionnaires our automount program seems simpler to use than the built-in automount feature.

We also performed our own tests. Once parts of the program had been programmed, the program was tested in different virtual boxes to make sure it is scalable amongst Linux systems. We also tested using variously formatted USB devices (FAT 32, vFAT, exFAT and NTFS). We tested it in the Ubuntu Linux

System, the system provided by the Carleton University COMP 2401 course, the Ubuntu Mate System for Raspberry pi installed in a laptop and a raspberry pi where the Ubuntu Mate System was set up.

Input		Test (success = x)							
USB Name&Size	Format	USB Detection	Mount Dialog Box	Reject Mount	Accept Mount	File transfer from USB	File transfer to USB	File Transfer as Superuser	Unmount USB
Lexar 8Gb	exFAT	X	X	X					
Kingston 16Gb	exFAT	X	X	X					
Lexar 8Gb	FAT32	X	X	X	X		X	X	X
Lexar 8Gb	NTFS	X	X	X					
Gigastone 16Gb	vFAT	X	X	X	X		X	X	X
Sandisk 32GB	vFAT	X	X	X	X		X	X	X

Figure 1 : A brief overview of the black-box testing results. “X” indicates a successful outcome inline with the expected outcome

As you can see in the above figure, we were able to use vFat and FAT32 formatted USB with no problem, however, exFAT and NTFS were not compatible with our program. This is because we need to install specific packages for them to run. More information about it can be found in the ‘Results’ section.

Throughout the process of testing, we came across a few glitches related to the add and remove actions being called twice. However, this was not reproducible in the Ubuntu Mate System. After further testing, we figured out this only happened when the usb drive is bootable.

However, depending on the certain operating system, usleep() can help prevent double docking on different ubuntu operating systems due to the time waiting. Hence we integrated it into our code.

It is not fully compatible with other linux systems, due to the Ubuntu mate being a bare bones operating system. There might be other usb mount features that has not been fully disabled.

Apart from that, the program was tested multiple times throughout the development process to make sure there were no segmentation faults and that the usb drive was detected correctly and ejected correctly.

5 Conclusion

5.1 Summary

This a simple USB mount system with user control who chooses what happens to a USB when mounted. We were able to implement udev rules successfully and use system calls to the terminal to be able to mount certain formats of the device. We were also able to detect and inform the user in the event that the USB was unmounted.

Throughout the development process, our main goal of the project which was to create an automount feature similar to that of the windows system was fulfilled. However, there are a lot more features we had initially planned to add which are still in the process of being implemented. The extra features were related to security around usb devices, such as being able to enable password protection and restricting the types of files to be transferred. As a result of time restrictions and lack of research on subjects, we were not able to implement them but the future plan is to do more research and integrate them to the main program. More information about this can be found in Section 5.3.

Moreover, even though our automount feature works well, we still would like to further improve it by embedding it into the operating system so that the user does not need to call the executable from the terminal every time.

The work done in this project has enabled a better understanding for us on how a device works with the operating system and has built a foundation for us that can be worked upon to build better software for the end user.

5.2 Relevance

Operating systems provide end-users with an environment wherein their interaction with both the physical and sometimes non-physical elements enable them to accomplish daily tasks.

Enabling USB automounting relates to a lot of the topics in the course such as the use of kernel modules, storing device drivers and communication between devices and the kernel. The GUI component also deals with how the user mode is used to interact with the kernel and how it acts as a medium between the end user and the system. So the system performs the actions the user intended.

5.3 Future Work

Some of the future goals we have and how we plan to implement them are:

- **Embedding the automount feature into the operating system**

  - We need to do more research on this, however, from what we understand we need to integrated this into the kernel. From what we learnt in the course, we believe we might need to build a kernel module in the /dev folder so that it gets called whenever we mount the usb. However, this also means we will need to figure out a way to override the already built in automount feature.

Another way we could implement this is by creating a desktop icon that can be clicked on to run the software instead of having the user call the executable from the terminal.

- **Increasing the number of usb devices that can be mounted in one session**

  - Currently the software allows only 100 usbs to be mounted in one session. If a there we try plugging in another usb after mounting them 100 times, the program crashes. This is because we have a USB array of size 100. In order to fix this problem we would have to figure out how to maybe use another data structure that can be resized infinitely and efficiently without slowing down the process. Possibly dynamically allocated the memory instead of statically allocating it.

- **Enabling password protection for usb devices**

  - We believe that if you can password protect a usb, it will prevent strangers from accessing the files in the usb. We plan on implementing this possibly by creating a simple username/password program. However, to make it more secure to prevent hackers, we will have to look into encryption in more depth.

- **Restricting the types of files to be transferred from usb to the system**

  - Currently, we can't seem to add any sort of files to the usb, however we can transfer files from the usb to the system. However, we will need to look through our code thoroughly to figure out what is causing the restriction. To restrict the types of files to be transferred, we were initially planning on creating rules for it that would not let certain types of files to be transferred which would always be true. However, after performing our evaluation, we have decided to instead have the default behaviour that will restrict the file types, but the user will have the choice of enabling or disabling the function. We will have to do further research on how we could implement this but for now we believe we will have to use zenity[7] again for the GUI parts of it.

Contributions of Team Members

Anna Shi: editing, report writing, programming and testing (Implementation of research)

Zarish Owais: testing and report writing. (Research, Report)

Ameera Alam: report writing and programming (Zenity, Libnotify, Research, Report)

Appendix

/dev: location for devices

Blkid: block identification

Makedev: script file for loading device files

Udev: userspace /dev

USB: universal serial bus

UUID: unique universal identifier

References

[1] Nemeth, Evi, Garth Synder and Trent R Hein. Linux Administration Handbook. Upper Saddle River, 7th April 2017. PDF.

[2] UNNIKRISHNAN. Udev: Introduction to Device Management In Modern Linux System. 18 December 2009. https://www.linux.com/news/udev-introduction-device-management-modern-linux-system. 7th April 2017.

[3] 24.4.2. Using the blkid Command, Red Hat. n.d. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/s2-sysinfo-filesystems-blkid.html. 7th April 2017.

[4] Dilger, Andreas. blkid(8) - Linux man page. n.d. https://linux.die.net/man/8/blkid. 7th April 2017.

[5] 31.3.2 Mount, Unmount, Remount. n.d. https://www.gnu.org/software/libc/manual/html_node/Mount_002dUnmount_002dRemount.html. 7th April 2017.

[6] Kerrisk, Michael. Linux Programmers Manual Mount(2). n.d. http://man7.org/linux/man-pages/man2/mount.2.html. 7th April 2017.

[7] https://help.gnome.org/users/zenity/stable/

[8] https://developer.gnome.org/libnotify/

[9] https://www.linux.com/news/udev-introduction-device-management-modern-linux-system