AirMouse

OS PROJECT

Ashiq Muhammed
Ronnel Davis
Vrushab Jambhulkar

Android Application

```
touch_area.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View view, MotionEvent motionEvent) {

        x = motionEvent.getX();
        y = motionEvent.getY();

        dx = (-1) * Math.round((x - lastX));
        dy = Math.round(y - lastY);

        //printing change in x and y
        Log.d(MouseControllerActivity.TAG, dx + " " + dy + " " + key);

        lastX = x;
        lastY = y;
        return true;
    }
});
```

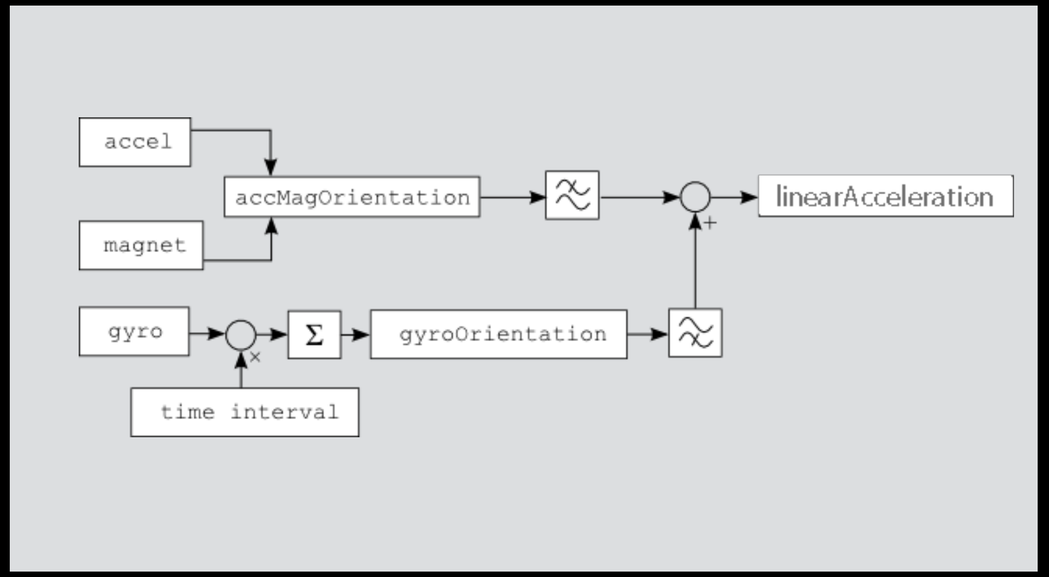
This methods reads the current touch X and Y coordinates, we take difference of the previous and latest touch coordinates and then prints them on the logcat.

```java
@Override
public boolean onTouch(View view, MotionEvent motionEvent) {
    if (view.getId() == R.id.button_left_touch
            || view.getId() == R.id.button_right_touch) {
        if (motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
            if (view.getId() == R.id.button_left_touch) {
                key = 3;
            } else if (view.getId() == R.id.button_right_touch) {
                key = 1;
            }
            Log.d(MouseControllerActivity.TAG, dx + " " + dy + " " + key);
        } else if (motionEvent.getAction() == MotionEvent.ACTION_UP) {
            key = 0;
        }
    }
    return true;
}
```

On pressing left or right buttons the key value becomes 3 or 1 respectively, otherwise key value is 0.

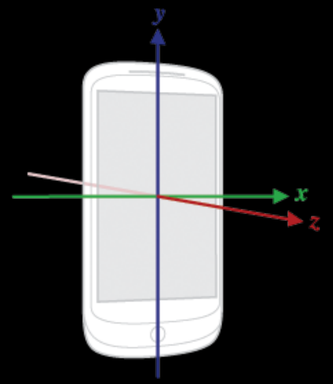
```
11-09 22:37:51.993 11128-11128/com.vrjco.v.os_project D/OSP: 1 -2 3
11-09 22:37:52.013 11128-11128/com.vrjco.v.os_project D/OSP: 1 -2 3
11-09 22:37:52.043 11128-11128/com.vrjco.v.os_project D/OSP: 0 -1 3
11-09 22:37:52.103 11128-11128/com.vrjco.v.os_project D/OSP: -1 2 3
11-09 22:37:52.133 11128-11128/com.vrjco.v.os_project D/OSP: -1 2 3
11-09 22:37:52.183 11128-11128/com.vrjco.v.os_project D/OSP: 1 -4 3
11-09 22:37:52.183 11128-11128/com.vrjco.v.os_project D/OSP: 0 0 3
```

On the logcat, it prints as “OSP: <x> <y> <key> “



For the Gyro Mouse, We use Sensor Manager in android to read the sensor values.

We read Acceleration, Magnetic Sensor and Gyro Sensor values, and combine Acceleration and Magnetic Sensor values to compute accMagOrientation which is then filtered using a low-pass gyro filter to get the linear acceleration.



And similar to the touch values use print the X-axis and Z-axis values along with the key to the logcat.

Shell Script

```
#!/bin/sh

#  Script.sh
#
#  Created on 11/7/16.
#  Copyright © 2016. All rights reserved.

PKG_OK=$(dpkg-query -W --showformat='${Status}\n'
android-tools-adb|grep "install ok installed")
if [ "" == "$PKG_OK" ]; then
    echo "====================================="
    echo "ADB not installed, installing ADB now"
    echo "====================================="

    sudo add-apt-repository ppa:phablet-team/
    tools
    sudo apt-get update
    sudo apt-get --force-yes --yes install
    android-tools-adb
fi
```

- Initially we have to check whether android adb tool is already installed on your computer. So we query all the packages currently installed on the system and checks whether any of the packages is of android adb.
- If the android adb packages are already installed we move on to the next step otherwise we start the shell scripting with add-apt-repository ppa:phablet-team/tools which finds the respective repository from the online directory and adds it to the local machine.
- Then we run apt-get update to update the contents of the local repo with the latest build.
- Finally running apt-get install android-tools-adb installs the android adb tools required for the android device to communicate with the computer.

- Now again as the same as before we have to check if the linux headers required for the kernel program are installed.
- If the linux headers are installed we move on to the next step otherwise we install the required linux headers using the command apt-get install linus-headers-$(uname -r)
- Other than linux headers we also require some additional headers to be installed. It can be installed using the following command :
  apt-get install build-essential

```
PKG_OK=$(dpkg-query -W --showformat='${Status}\n'
linus-headers-$(uname -r)|grep "install ok
installed")
if [ "" == "$PKG_OK" ]; then
    echo “=======================================”
    echo "Linux headers not installed, installing
    linux headers now"
    echo “=======================================”
    sudo apt-get install linus-headers-$(uname -r)
fi

PKG_OK=$(dpkg-query -W --showformat='${Status}\n'
build-essential|grep "install ok installed")
if [ "" == "$PKG_OK" ]; then
    echo "========================="
    echo "Building essential headers"
    echo "========================="

    sudo apt-get install build-essential
fi
```

```
echo "============="
echo "Making module"
echo "============="

sudo rmmod vms1.ko
make
sudo insmod vms1.ko

echo "================"
echo "Compiling client"
echo "================"

gcc coord.c
adb logcat -s "OSP" | sudo ./a.out
```

- Now we run rmmod vsm1.ko to remove any old copies of the kernel module file from the Operating System.
- Then we run the command make which runs the MakeFile which does the work of creating the kernel module file vms1.ko from the c program vms1.c
- Finally we can insert the newly compiled kernel module file into the Operating System using the command insmod vms1.ko

Client Application

```
#include<unistd.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
#include<sys/stat.h>
#include<fcntl.h>

int main(int argc, char *argv[])
{
    char str[200];
    int sim_fd;
    int x, y, key;
    char buffer[10];

  x = 0;
  y = 0;
  const int sensitivity = 1;
  // Open the sysfs coordinate node
  sim_fd = open("/sys/devices/platform/virmouse/vmevent",
  O_RDWR);

    if (sim_fd < 0) {
        perror("Couldn't open vms coordinate file\n");
        exit(-1);
    }
```

- This is the C program that controls the flow of data from the Android Device into the compiled linux kernel module.
- The variables ‘x’ and ‘y’ will be used to store the values passed via adb logical from the mobile device. Initially they are set to 0.
- We also have a multiplier value to increase the sensitivity of the mouse controller.
- When we run the compiled kernel module vms1.ko it creates an event file called vmevent whereon we pass on the x-y coordinates from the device.
- We also run a check to see whether the required vmevent file is created by the kernel module before we move any further.

- Now we run a while loop to continuously keep on accepting x-y coordinates received via adb logcat and passing it on to the kernel module.
- We use memmove function to cut short each line passed by the adb logcat to remove the garbage values preceding the actual x-y coordinates.
- sscanf reads the three values ‘x’, ’y’, ‘key’ which are used to store the x coordinate, y coordinate and button event clicks respectively.
- Finally we multiply the x-y coordinates with the sensitivity value and write it into the vmevent file.

```
    while (1) {
        x = 0;
        y = 0;

        gets(str);
        memmove(str, str+18, strlen(str));

        sscanf(str, "%d %d %d", &x, &y, &key);

        // Convey simulated coordinates to the virtual
mouse driver

      x *= -sensitivity;
      y *= sensitivity;

      sprintf(buffer, "%d %d %d", x, y, key);
      write(sim_fd, buffer, strlen(buffer));
      fsync(sim_fd);
    }

    close(sim_fd);
}
```

Kernel Module

```
#include <linux/fs.h>
#include <asm/uaccess.h>
#include <linux/pci.h>
#include <linux/input.h>
#include <linux/platform_device.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

struct input_dev *virmouse_input_dev;
static struct platform_device *virmouse_dev; /* Device
structure */

/* Sysfs method to input simulated coordinates */
static ssize_t write_virmouse(struct device *dev,
                              struct device_attribute *attr,
                              const char *buffer, size_t
count)
{
    int x, y, key;

    /* parsing input data */
    sscanf(buffer, "%d%d%d", &x, &y, &key);
```

- We declare all the headers required to run the kernel
- Initialise the structure for the virtual mouse that we are gonna use.
- Now we initialize the variables to store the x-y coordinates and also the variable to register the left click and the right click functions.
- The values passed on from the client side(coord.c) restored in these variables as required.

- The values stored in x and y are passed on to the function input_report_rel as one of its attributes respectively. If needed these values can be displayed on the screen.
- Now the third variable key used to store values of mouse button clicks are to be passed on to its respective function.
- The variable key can take 3 values : 1 for left button click, 2 for middle button click and finally 3 for right mouse button click.
- All the events are finally stored in the structure virmouse_input_dev which is later on passed on to the function input_sync

```
/* Report relative coordinates */
    input_report_rel(virmouse_input_dev, REL_X, x);
    input_report_rel(virmouse_input_dev, REL_Y, y);

    printk ("virmouse_event: X:%d Y:%d %d\n", x, y, key);

    /* Report key event */
    if (key>0) {
        if (key==1)
            input_report_key(virmouse_input_dev, BTN_LEFT,
1);
        else if (key==2)
            input_report_key(virmouse_input_dev, BTN_MIDDLE,
1);
        else if (key==3)
            input_report_key(virmouse_input_dev, BTN_RIGHT,
1);
    }

    input_sync(virmouse_input_dev);

    return count;

}
```

```c
/* Attach the sysfs write method */
DEVICE_ATTR(vmevent, 0644, NULL, write_virmouse);

/* Attribute Descriptor */
static struct attribute *virmouse_attrs[] = {
    &dev_attr_vmevent.attr,
    NULL
};

/* Attribute group */
static struct attribute_group virmouse_attr_group = {
    .attrs = virmouse_attrs,
};

/* Driver Initializing */
int __init virmouse_init(void)
{
    /* Register a platform device */
    virmouse_dev =
platform_device_register_simple("virmouse", -1, NULL, 0);
    if (IS_ERR(virmouse_dev)){
        printk ("virmouse_init: error\n");
        return PTR_ERR(virmouse_dev);
    }
```

- Now the sysfs write method is attached via the DEVICE_ATTR function.
- After that is done we have to go on and further create two more structures for attribute descriptor and attribute group.
- Now we can start with writing the actual function responsible for initialization of the Device Driver called __init virmouse_init
- This will register itself as a virtual mouse platform. After this checks are performed to see if the virtual mouse initialization was successful

- We have to create a sysfs node to read the simulated coordinates that have been passed on to the kernel module.
- Memory has to be allocated to the input data device data structure virmouse_input_dev
- Now we have to announce that the new virtual mouse will be generating the relative coordinates for the mouse and the respective values are passed on.
- Other device driver information like the bus type, vendor, product and version number of the kernel module are also set.

```
/* Create a sysfs node to read simulated coordinates */
    sysfs_create_group(&virmouse_dev->dev.kobj,
&virmouse_attr_group);

    /* Allocate an input device data structure */
    virmouse_input_dev = input_allocate_device();
    if (!virmouse_input_dev) {
        printk("Bad input_allocate_device()\n");
        return -ENOMEM;
    }

    /* Announce that the virtual mouse will generate relative
coordinates */
    set_bit(EV_REL, virmouse_input_dev->evbit);
    set_bit(REL_X, virmouse_input_dev->relbit);
    set_bit(REL_Y, virmouse_input_dev->relbit);

    virmouse_input_dev->name = "Virtual Mouse";
  virmouse_input_dev->phys = "vmd/input0"; // "vmd" is the
driver's name
  virmouse_input_dev->id.bustype = BUS_VIRTUAL;
  virmouse_input_dev->id.vendor  = 0x0000;
  virmouse_input_dev->id.product = 0x0000;
  virmouse_input_dev->id.version = 0x0000;
```

```
virmouse_input_dev->evbit[0] = BIT_MASK(EV_KEY) |
BIT_MASK(EV_REL);
  virmouse_input_dev->keybit[BIT_WORD(BTN_MOUSE)] =
BIT_MASK(BTN_LEFT) | BIT_MASK(BTN_RIGHT) |
BIT_MASK(BTN_MIDDLE);
  virmouse_input_dev->relbit[0] = BIT_MASK(REL_X) |
BIT_MASK(REL_Y);
  virmouse_input_dev->keybit[BIT_WORD(BTN_MOUSE)] |=
BIT_MASK(BTN_SIDE) | BIT_MASK(BTN_EXTRA);
  virmouse_input_dev->relbit[0] |= BIT_MASK(REL_WHEEL);

    /* Announce key event */
    set_bit(EV_KEY, virmouse_input_dev->evbit);
    set_bit(BTN_LEFT, virmouse_input_dev->keybit);
    set_bit(BTN_MIDDLE, virmouse_input_dev->keybit);
    set_bit(BTN_RIGHT, virmouse_input_dev->keybit);

    /* Register with the input subsystem */
    input_register_device(virmouse_input_dev);

    /* print messages in the dmesg */
    printk("Virtual Mouse Driver Initialized.\n");

    return 0;
}
```

- Now that we have successfully declared that there will be new relative coordinates we can pass on the x-y coordinates from the adb logcat on behalf of the actual values by masking it. This step is done for button click events as well.
- Now it has to be declared that all the coordinates are event clicks have been successfully updated.
- Finally input_register_device function is called to register the new driver with the linux input subsystem and Success message is displayed on the screen.

- The last part in the kernel module is to write the function for uninitializing the currently initialized virtual mouse driver.
- To do this we start with calling the input_unregister_device function, then removing the sysfs node and then finally unregistering the driver by calling the function platform_device_unregister

```
/* Driver Uninitializing */
void virmouse_uninit(void)
{
    /* Unregister from the input subsystem */
    input_unregister_device(virmouse_input_dev);

    /* Remove sysfs node */
    sysfs_remove_group(&virmouse_dev->dev.kobj,
&virmouse_attr_group);

    /* Unregister driver */
    platform_device_unregister(virmouse_dev);

    return;
}

module_init(virmouse_init);
module_exit(virmouse_uninit);

MODULE_DESCRIPTION("AIR MOUSE");
```