Arduino Project Summary: Smart Home Security System

Objective:

The objective of this Arduino-based project is to create a smart home security system using various components such as PIR motion sensor, LDR, buzzer, LED, home lock, and servo motor. The system aims to provide functionalities like system activation/deactivation, locking/unlocking, and alerting users to potential security breaches.

Components Used:

1. Arduino Uno Board
2. PIR Motion Sensor
3. LDR (Light Dependent Resistor)
4. Green, Red, and White LEDs
5. Home Lock
6. Servo Motor
7. Buzzer
8. Jumper Wires

Setup Steps and Documentation:

1. **Library Inclusion:**
    - The inclusion of the Servo library is essential to control the servo motor responsible for actuating the home lock mechanism.

- Download Arduino Servo library from Github.
- Extract it and add it to the Arduino libraries in Ubuntu Linux.
- Add the servo library in the USER_LIB_PATH and ARDUINO_LIB in Makefile.

```
#include "Servo.h"
```

2. **Pin and Component Constants:**
    - Define pin assignments for each component used in the project to ensure proper interfacing and control.
    - Assign meaningful names to each pin to enhance code readability and maintainability.
    - Specify digital and analog pins for sensors, LEDs, the buzzer, the servo motor, and any other components used in the project.

```
#define WHITE_LED 7
#define GREEN_LED 8
#define RED_LED 9
#define PIR_PIN 2
#define LDR_PIN A0
#define BUZZER_PIN 10
#define SERVO_PIN 3
#define BUTTON_PIN 4
```

3. **Setup():**

- **Component Initialization:**
    1. In the setup() function, the pinMode() function is used to configure the Arduino pins as either INPUT or OUTPUT.
    2. For output components like LEDs, the pinMode() function sets the pins as OUTPUT to allow the Arduino to control the component's state (ON/OFF).
    3. For input components like the PIR motion sensor and button, the pinMode() function sets the pins as INPUT to read external signals.
- **Servo Initialization:**
    1. The Servo library is initialized by creating an instance of the Servo class (myServo) and attaching it to the designated pin (SERVO_PIN).
    2. This step prepares the servo motor for precise angular control, allowing it to actuate the home lock mechanism.
- **Serial Communication Setup:**
    1. Serial communication is initialized at a baud rate of 9600 using Serial.begin(9600).
    2. This enables communication between the Arduino board and an external device such as a computer, facilitating debugging and user interaction via the serial monitor.
- **LED State Initialization:**

i. The initial state of the LEDs is configured, with the green LED (GREEN_LED) set to HIGH (ON) and the red LED (RED_LED) set to LOW (OFF).
ii. This establishes the default visual indication of the system's status, with the green LED indicating readiness and the red LED initially off.

4. **Loop():**

- **User Input and System State Management:**
    i. The loop() function continuously checks for user input via the serial monitor or a physical button connected to the Arduino (BUTTON_PIN).
    ii. Depending on the user's selection, the system state (systemState) is updated to reflect whether the security system is active (1) or inactive (0).
    iii. Additionally, the loop() function handles user input to lock or unlock the security system, adjusting the lock state (lockState) accordingly.
- **Sensor Reading and Event Handling:**
    i. Within the loop(), the PIR motion sensor (PIR_PIN) and LDR (LDR_PIN) are continuously monitored to detect motion and ambient light levels, respectively.

ii. If motion is detected by the PIR sensor, the system responds by activating the white LED spotlight (WHITE_LED) if it's nighttime (nightState = 1).

iii. The system also checks the light level detected by the LDR to determine whether it's daytime or nighttime, updating the nightState accordingly.

- **Buzzer and Servo Control:**
    i. Depending on the system's state and detected events, the loop() function triggers actions such as activating the buzzer (BUZZER_PIN) to emit audible alerts or controlling the servo motor (myServo) to lock/unlock the home lock mechanism.

    ii. For example, if the security system is locked (lockState = 1), the buzzer is activated, and the servo motor rotates to lock the home lock, deterring unauthorized access.

- **Serial Communication and User Interaction:**
    i. Serial.print() and Serial.println() functions are used within the loop() to provide status updates and prompts for user input via the serial monitor.

    ii. The loop() continuously monitors the serial input buffer for commands entered by the user, allowing them to interact with and control the security system remotely.

5. **User Input Handling / Serial Monitor Interaction:**

- Users can interact with the system via the Arduino's serial monitor, entering commands to control various aspects of the security system.
- The following user selections are available, each triggering different actions within the system:
- **1. System Off:**
    1. If the user selects "1" from the serial monitor, the system checks if it's already in the off state (systemState = 0).
    2. If the system is already off, a message indicating that the system is already down is printed to the serial monitor.
    3. If the system is currently on (systemState = 1), it transitions to the off state (systemState = 0), and all system components are deactivated (e.g., LEDs, buzzer, servo motor).
- **2. System On:**
    1. If the user selects "2" from the serial monitor, the system checks if it's already in the on state (systemState = 1).
    2. If the system is already on, a message indicating that the system is already up is printed to the serial monitor.
    3. If the system is currently off (systemState = 0), it transitions to the on state (systemState = 1), allowing the system to monitor sensors and respond to events.
- **3. Lock:**
    1. If the user selects "3" from the serial monitor, the system checks if it's currently on (systemState = 1).

ii. If the system is off, a message indicating that the system needs to be turned on first is printed to the serial monitor.
iii. If the system is on, the system checks if it's already locked (lockState = 1).
iv. If the system is already locked, a message indicating that the system is already locked is printed to the serial monitor.
v. If the system is unlocked (lockState = 0), the system locks the home lock mechanism by rotating the servo motor to the locked position (180 degrees) and activates the buzzer to alert of the lock status.

- **4. Unlock:**
    i. If the user selects "4" from the serial monitor, the system checks if it's currently on (systemState = 1).
    ii. If the system is off, a message indicating that the system needs to be turned on first is printed to the serial monitor.
    iii. If the system is on, the system checks if it's currently locked (lockState = 1).
    iv. If the system is already unlocked (lockState = 0), a message indicating that the system is already unlocked is printed to the serial monitor.
    v. If the system is locked, the system prompts the user to enter the password via the serial monitor.

vi. Upon entering the correct password, the system unlocks the home lock mechanism by rotating the servo motor to the unlocked position (0 degrees) and deactivates the buzzer.

Challenges encountered:

During the development of the Smart Home Security System, several challenges were encountered, each requiring careful troubleshooting and problem-solving to overcome. Some of the key challenges faced include:

1. **Interfacing with Multiple Sensors:**
    - Integrating multiple sensors (e.g., PIR motion sensor, LDR) into the system posed a challenge in ensuring proper wiring and data acquisition.
    - Understanding the sensor data formats and calibration methods was crucial to accurately interpret sensor readings and trigger appropriate system responses.
2. **Servo Motor Calibration:**
    - Calibrating the servo motor to accurately control the home lock mechanism required experimentation and fine-tuning.
    - Ensuring that the servo motor rotated precisely to the desired angles (0 degrees for unlocked and 180 degrees for locked) was essential for reliable system operation.
3. **User Input Handling:**

- Implementing robust user input handling via the serial monitor involved addressing potential edge cases and error scenarios.
- Ensuring that the system responded correctly to user commands and provided clear feedback required thorough testing and validation.

4. **Password Verification:**
    - Implementing password verification for unlocking the system introduced complexity in handling user input and validating passwords.
    - Ensuring that the system accurately verified passwords while maintaining security and user privacy was a critical consideration.
5. **Real-time Feedback Mechanisms:**
    - Implementing real-time feedback mechanisms, such as LED indicators and the buzzer, required careful synchronization and coordination with sensor data.
    - Ensuring that the feedback provided clear and intuitive information to users about the system's status and alerts was essential for user experience.
6. **Integration with External Hardware:**
    - Integrating external hardware components, such as the home lock mechanism and servo motor, required compatibility testing and proper wiring.
    - Ensuring that the Arduino board could effectively control and communicate with these components was essential for the overall functionality of the system.

7. **Software Installation and Configuration:**
    - Installing and configuring the necessary software, such as the Servo library for controlling the servo motor, required familiarity with Arduino IDE and library management.
    - Ensuring that the libraries were correctly installed and included in the Arduino project was crucial for compiling and uploading code without errors.
8. **Integration with External Systems:**
    - Establishing communication between the Arduino board and external systems, such as Windows Subsystem for Linux (WSL) via USB/IP, required understanding network protocols and configuration.
    - Ensuring reliable communication and data exchange between the Arduino board and external systems was essential for system integration and functionality.

By addressing these challenges through systematic troubleshooting, experimentation, and adaptation of solutions, the Smart Home Security System was successfully developed, demonstrating effective problem-solving skills and resilience in overcoming technical obstacles.

Reflection:

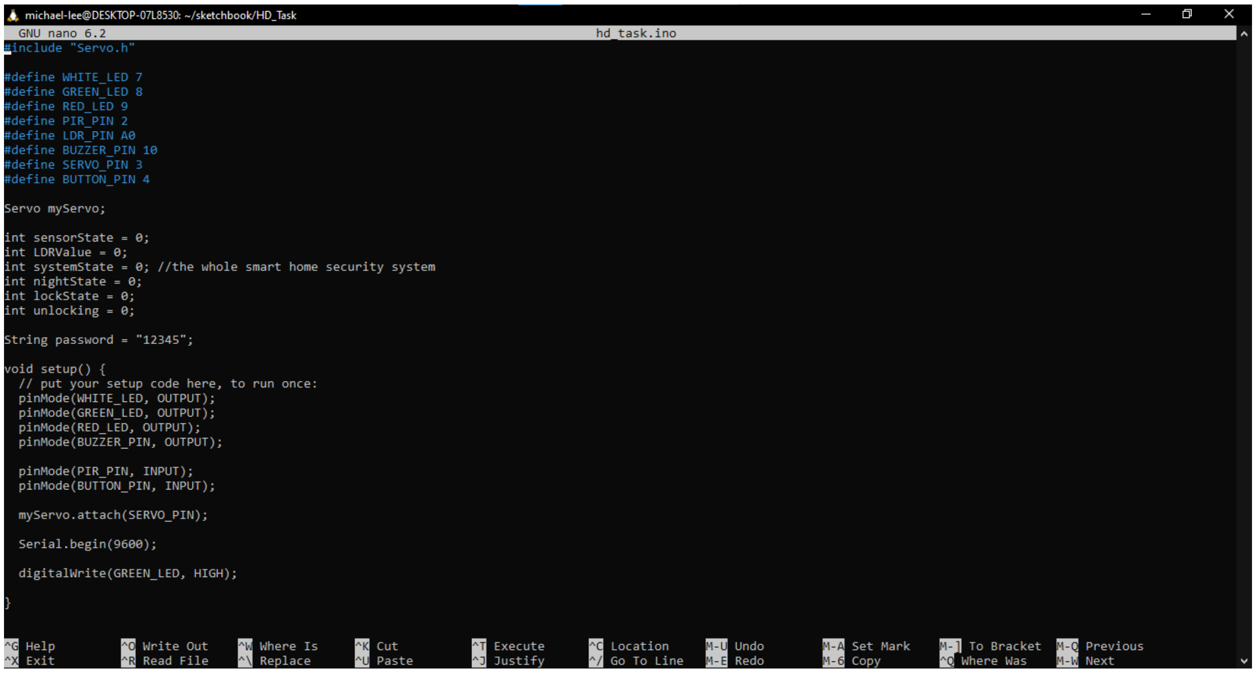
Building the smart home security system was an enlightening journey that not only solidified my understanding of Arduino programming and hardware integration but also provided valuable insights into project management and documentation practices. As I navigated through challenges such as library installation and interfacing with external hardware, I realized the importance of thorough research, experimentation, and perseverance in problem-solving.

Moreover, this project sparked my curiosity about the broader applications of embedded systems and IoT in enhancing everyday life. Exploring concepts like sensor fusion, data processing, and remote monitoring opened up a world of possibilities for future projects and innovations. I'm eager to delve deeper into these areas and explore how technology can be leveraged to address real-world challenges and improve quality of life.

Looking ahead, I envision leveraging the skills and knowledge gained from this project to tackle more complex projects and contribute meaningfully to the field of electronics and programming. Whether it's designing home automation solutions, developing IoT devices, or exploring emerging technologies, I am excited to continue my journey of exploration and innovation in the realm of embedded systems and beyond.

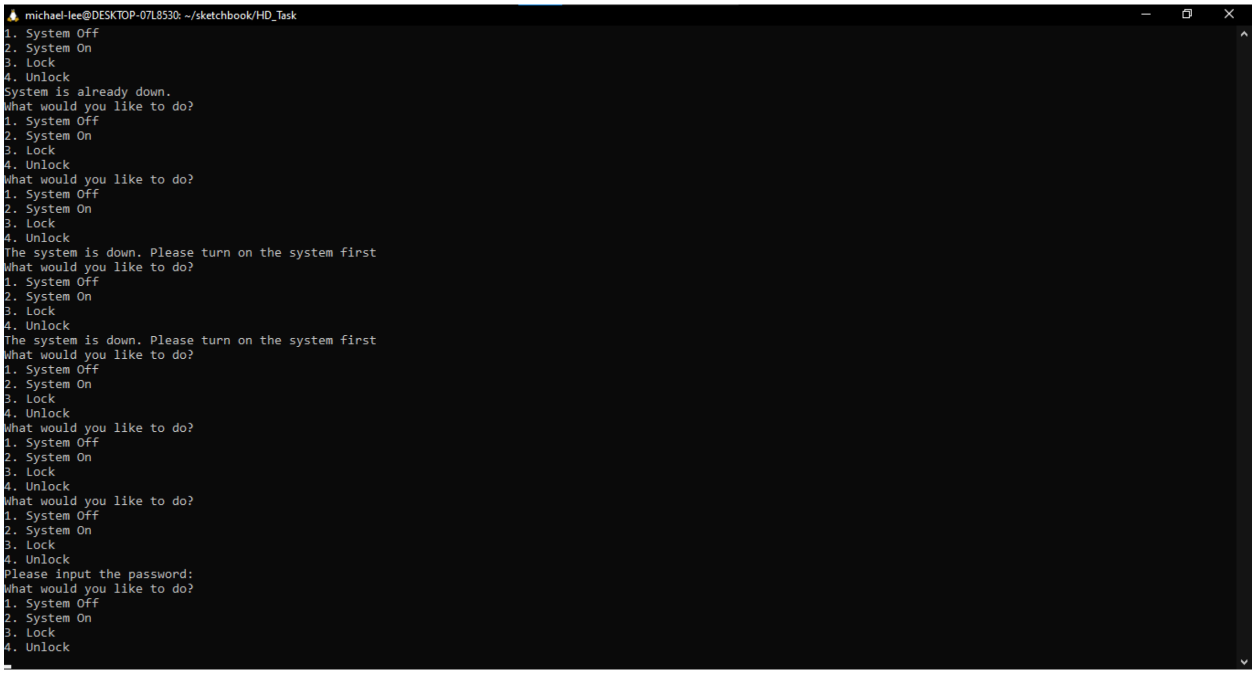
Screenshots:

1. **Arduino sketch in linux (using linux command eg nano hd_task.ino)**


michael-lee@DESKTOP-07L8530: ~/sketchbook/HD_Task

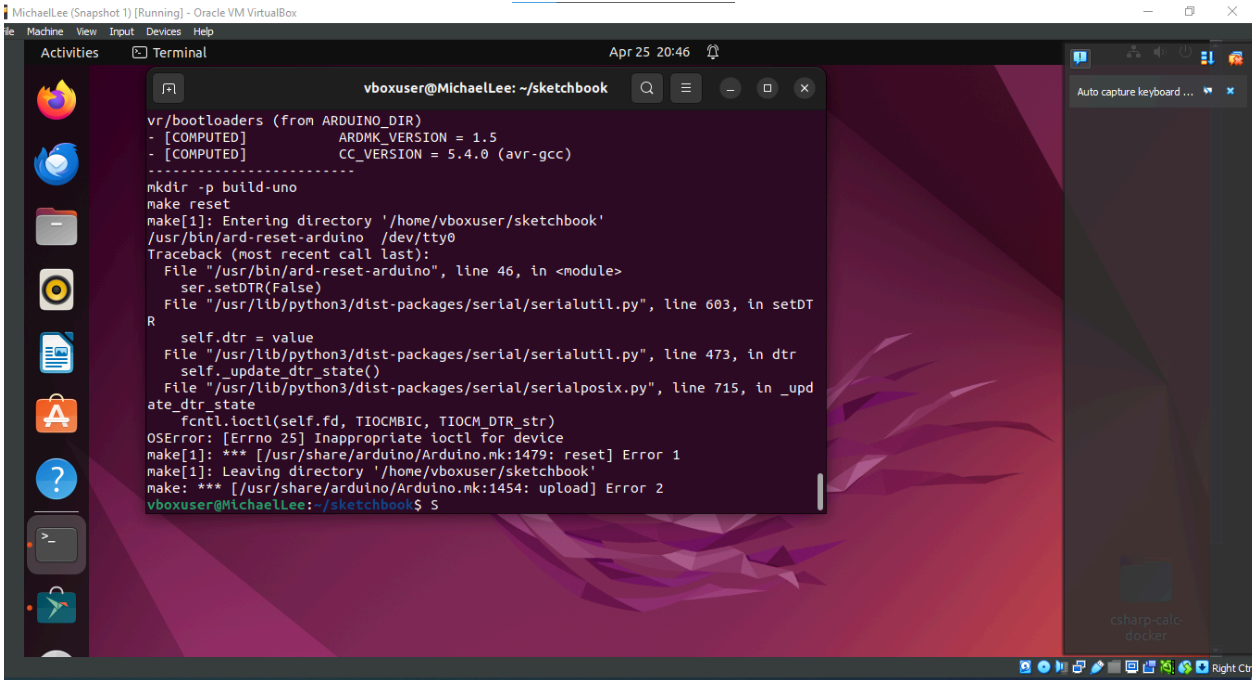
```
  GNU nano 6.2                                                                    hd_task.ino
#include "Servo.h"

#define WHITE_LED 7
#define GREEN_LED 8
#define RED_LED 9
#define PIR_PIN 2
#define LDR_PIN A0
#define BUZZER_PIN 10
#define SERVO_PIN 3
#define BUTTON_PIN 4

Servo myServo;

int sensorState = 0;
int LDRValue = 0;
int systemState = 0; //the whole smart home security system
int nightState = 0;
int lockState = 0;
int unlocking = 0;

String password = "12345";

void setup() {
  // put your setup code here, to run once:
  pinMode(WHITE_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);
  pinMode(RED_LED, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);

  pinMode(PIR_PIN, INPUT);
  pinMode(BUTTON_PIN, INPUT);

  myServo.attach(SERVO_PIN);

  Serial.begin(9600);

  digitalWrite(GREEN_LED, HIGH);

}

^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark  M-] To Bracket M-Q Previous
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo      M-6 Copy      ^Q Where Was  M-W Next
```

2. **“Sudo apt-get install screen“, and then use command eg “make upload monitor clean” to open a screen for user interaction and screen monitor**

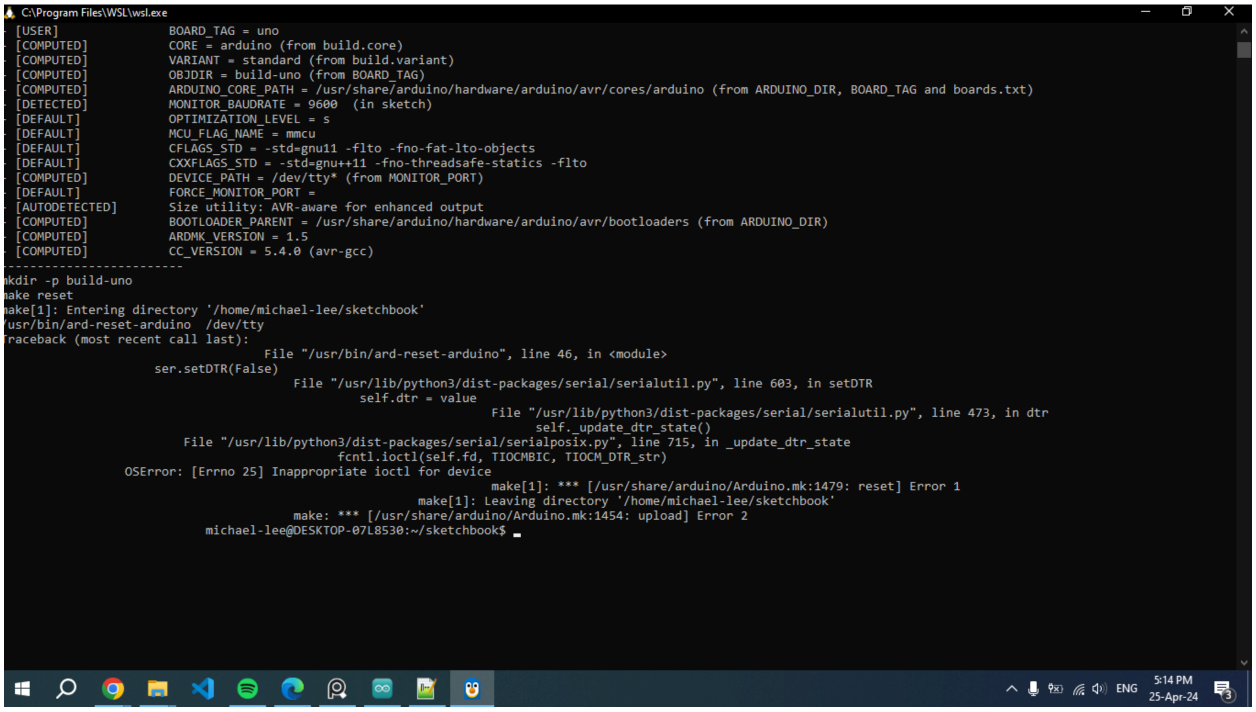

michael-lee@DESKTOP-07L8530: ~/sketchbook/HD_Task

```
1. System Off
2. System On
3. Lock
4. Unlock
System is already down.
What would you like to do?
1. System Off
2. System On
3. Lock
4. Unlock
What would you like to do?
1. System Off
2. System On
3. Lock
4. Unlock
The system is down. Please turn on the system first
What would you like to do?
1. System Off
2. System On
3. Lock
4. Unlock
The system is down. Please turn on the system first
What would you like to do?
1. System Off
2. System On
3. Lock
4. Unlock
What would you like to do?
1. System Off
2. System On
3. Lock
4. Unlock
What would you like to do?
1. System Off
2. System On
3. Lock
4. Unlock
Please input the password:
What would you like to do?
1. System Off
2. System On
3. Lock
4. Unlock
```

3. WSL/ USB Port Error

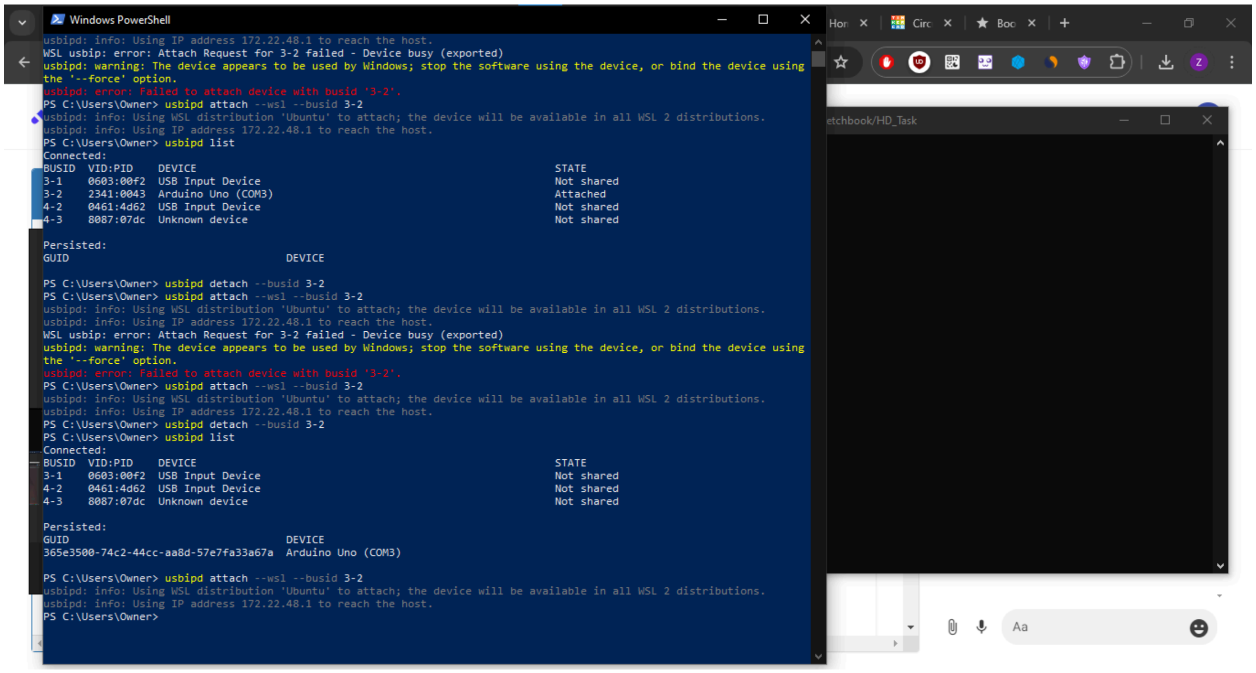


```
vr/bootloaders (from ARDUINO_DIR)
- [COMPUTED]           ARDMK_VERSION = 1.5
- [COMPUTED]           CC_VERSION = 5.4.0 (avr-gcc)
-------------------------
mkdir -p build-uno
make reset
make[1]: Entering directory '/home/vboxuser/sketchbook'
/usr/bin/ard-reset-arduino  /dev/tty0
Traceback (most recent call last):
  File "/usr/bin/ard-reset-arduino", line 46, in <module>
    ser.setDTR(False)
  File "/usr/lib/python3/dist-packages/serial/serialutil.py", line 603, in setDT
R
    self.dtr = value
  File "/usr/lib/python3/dist-packages/serial/serialutil.py", line 473, in dtr
    self._update_dtr_state()
  File "/usr/lib/python3/dist-packages/serial/serialposix.py", line 715, in _upd
ate_dtr_state
    fcntl.ioctl(self.fd, TIOCMBIC, TIOCM_DTR_str)
OSError: [Errno 25] Inappropriate ioctl for device
make[1]: *** [/usr/share/arduino/Arduino.mk:1479: reset] Error 1
make[1]: Leaving directory '/home/vboxuser/sketchbook'
make: *** [/usr/share/arduino/Arduino.mk:1454: upload] Error 2
vboxuser@MichaelLee:~/sketchbook$ S
```



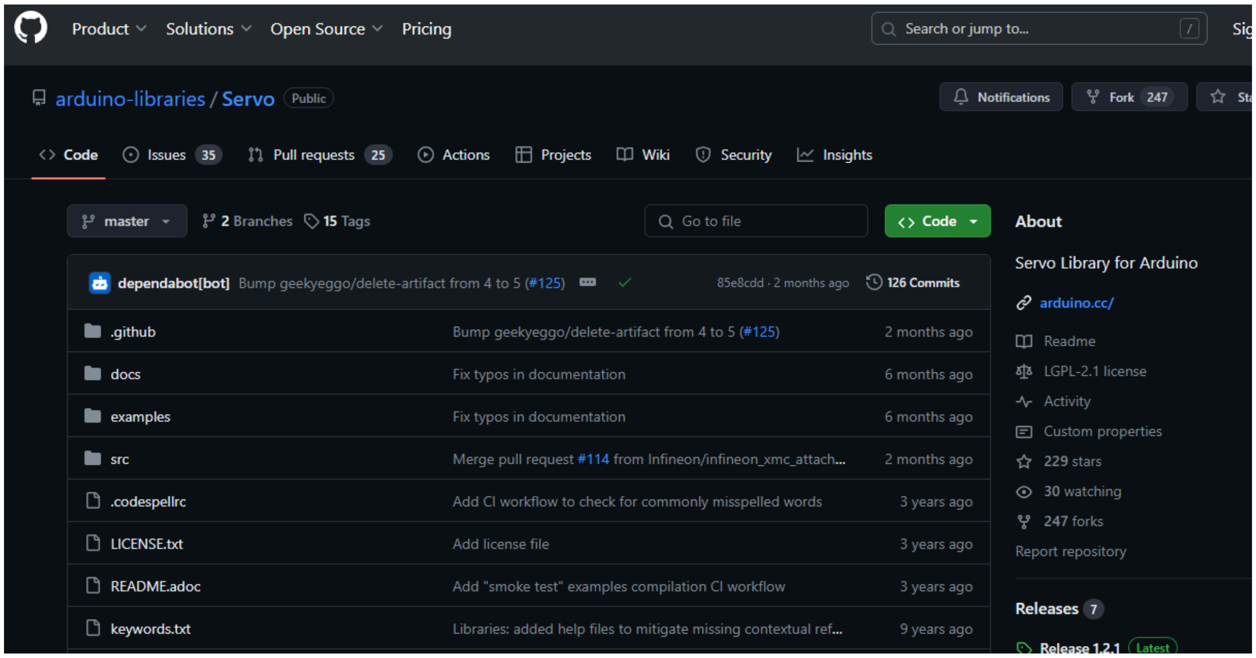
```
- [USER]                BOARD_TAG = uno
- [COMPUTED]            CORE = arduino (from build.core)
- [COMPUTED]            VARIANT = standard (from build.variant)
- [COMPUTED]            OBJDIR = build-uno (from BOARD_TAG)
- [COMPUTED]            ARDUINO_CORE_PATH = /usr/share/arduino/hardware/arduino/avr/cores/arduino (from ARDUINO_DIR, BOARD_TAG and boards.txt)
- [DETECTED]            MONITOR_BAUDRATE = 9600  (in sketch)
- [DEFAULT]             OPTIMIZATION_LEVEL = s
- [DEFAULT]             MCU_FLAG_NAME = mmcu
- [DEFAULT]             CFLAGS_STD = -std=gnu11 -flto -fno-fat-lto-objects
- [DEFAULT]             CXXFLAGS_STD = -std=gnu++11 -fno-threadsafe-statics -flto
- [COMPUTED]            DEVICE_PATH = /dev/tty* (from MONITOR_PORT)
- [DEFAULT]             FORCE_MONITOR_PORT =
- [AUTODETECTED]        Size utility: AVR-aware for enhanced output
- [COMPUTED]            BOOTLOADER_PARENT = /usr/share/arduino/hardware/arduino/avr/bootloaders (from ARDUINO_DIR)
- [COMPUTED]            ARDMK_VERSION = 1.5
- [COMPUTED]            CC_VERSION = 5.4.0 (avr-gcc)
-------------------------
nkdir -p build-uno
nake reset
nake[1]: Entering directory '/home/michael-lee/sketchbook'
/usr/bin/ard-reset-arduino  /dev/tty
Traceback (most recent call last):
                                   File "/usr/bin/ard-reset-arduino", line 46, in <module>
                     ser.setDTR(False)
                                       File "/usr/lib/python3/dist-packages/serial/serialutil.py", line 603, in setDTR
                                                self.dtr = value
                                                                File "/usr/lib/python3/dist-packages/serial/serialutil.py", line 473, in dtr
                                                                      self._update_dtr_state()
                         File "/usr/lib/python3/dist-packages/serial/serialposix.py", line 715, in _update_dtr_state
                                             fcntl.ioctl(self.fd, TIOCMBIC, TIOCM_DTR_str)
                 OSError: [Errno 25] Inappropriate ioctl for device
                                                                  make[1]: *** [/usr/share/arduino/Arduino.mk:1479: reset] Error 1
                                                        make[1]: Leaving directory '/home/michael-lee/sketchbook'
                                       make: *** [/usr/share/arduino/Arduino.mk:1454: upload] Error 2
                           michael-lee@DESKTOP-07L8530:~/sketchbook$ _
```

4. WSL/ USB Port solved by attaching usb port to wsl in windows powershell



```
usbipd: info: Using IP address 172.22.48.1 to reach the host.
WSL usbip: error: Attach Request for 3-2 failed - Device busy (exported)
usbipd: warning: The device appears to be used by Windows; stop the software using the device, or bind the device using
the '--force' option.
usbipd: error: Failed to attach device with busid '3-2'.
PS C:\Users\Owner> usbipd attach --wsl --busid 3-2
usbipd: info: Using WSL distribution 'Ubuntu' to attach; the device will be available in all WSL 2 distributions.
usbipd: info: Using IP address 172.22.48.1 to reach the host.
PS C:\Users\Owner> usbipd list
Connected:
BUSID  VID:PID    DEVICE                                                        STATE
3-1    0603:00f2  USB Input Device                                              Not shared
3-2    2341:0043  Arduino Uno (COM3)                                            Attached
4-2    0461:4d62  USB Input Device                                              Not shared
4-3    8087:07dc  Unknown device                                                Not shared

Persisted:
GUID                                  DEVICE

PS C:\Users\Owner> usbipd detach --busid 3-2
PS C:\Users\Owner> usbipd attach --wsl --busid 3-2
usbipd: info: Using WSL distribution 'Ubuntu' to attach; the device will be available in all WSL 2 distributions.
usbipd: info: Using IP address 172.22.48.1 to reach the host.
WSL usbip: error: Attach Request for 3-2 failed - Device busy (exported)
usbipd: warning: The device appears to be used by Windows; stop the software using the device, or bind the device using
the '--force' option.
usbipd: error: Failed to attach device with busid '3-2'.
PS C:\Users\Owner> usbipd attach --wsl --busid 3-2
usbipd: info: Using WSL distribution 'Ubuntu' to attach; the device will be available in all WSL 2 distributions.
usbipd: info: Using IP address 172.22.48.1 to reach the host.
PS C:\Users\Owner> usbipd detach --busid 3-2
PS C:\Users\Owner> usbipd list
Connected:
BUSID  VID:PID    DEVICE                                                        STATE
3-1    0603:00f2  USB Input Device                                              Not shared
4-2    0461:4d62  USB Input Device                                              Not shared
4-3    8087:07dc  Unknown device                                                Not shared

Persisted:
GUID                                  DEVICE
365e3500-74c2-44cc-aa8d-57e7fa33a67a  Arduino Uno (COM3)

PS C:\Users\Owner> usbipd attach --wsl --busid 3-2
usbipd: info: Using WSL distribution 'Ubuntu' to attach; the device will be available in all WSL 2 distributions.
usbipd: info: Using IP address 172.22.48.1 to reach the host.
PS C:\Users\Owner>
```

5. Servo library installation



arduino-libraries / Servo Public

dependabot[bot] Bump geekyeggo/delete-artifact from 4 to 5 (#125) ✓ 85e8cdd · 2 months ago 126 Commits

.github	Bump geekyeggo/delete-artifact from 4 to 5 (#125)	2 months ago
docs	Fix typos in documentation	6 months ago
examples	Fix typos in documentation	6 months ago
src	Merge pull request #114 from Infineon/infineon_xmc_attach...	2 months ago
.codespellrc	Add CI workflow to check for commonly misspelled words	3 years ago
LICENSE.txt	Add license file	3 years ago
README.adoc	Add "smoke test" examples compilation CI workflow	3 years ago
keywords.txt	Libraries: added help files to mitigate missing contextual ref...	9 years ago

About

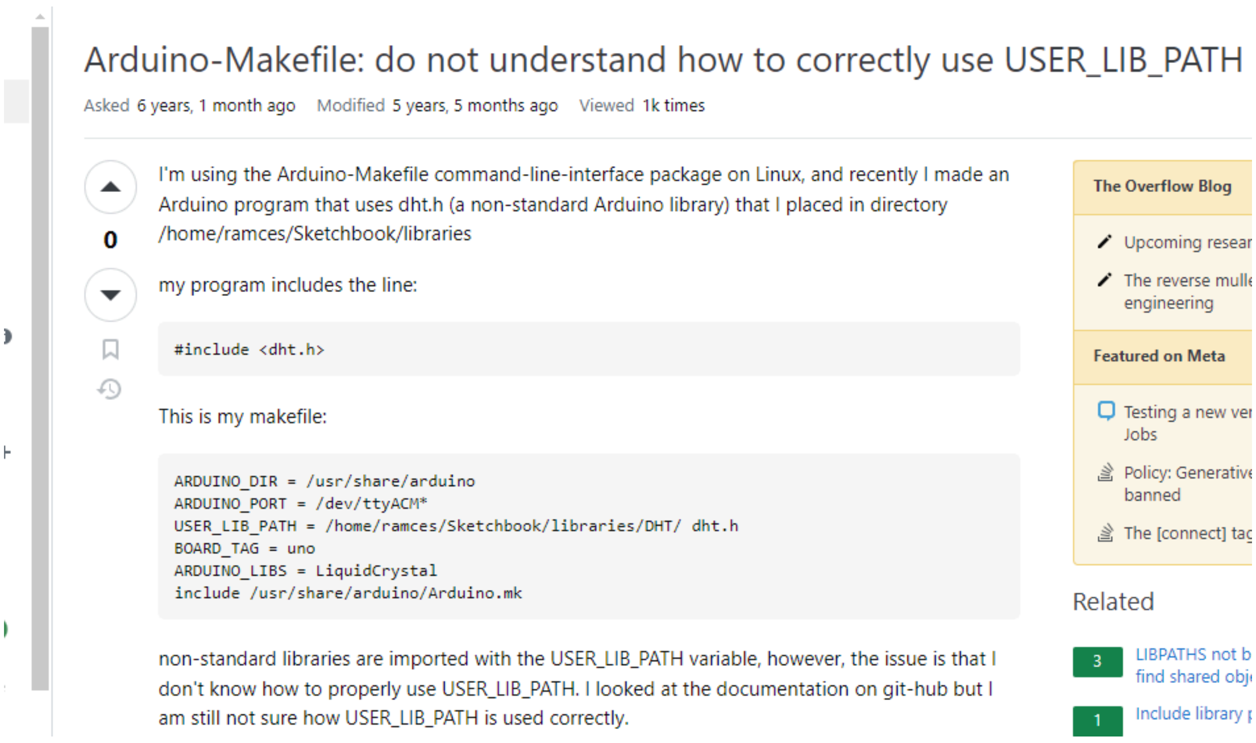
Servo Library for Arduino

arduino.cc/

Readme
LGPL-2.1 license
Activity
Custom properties
229 stars
30 watching
247 forks
Report repository

Releases 7

Release 1.2.1 Latest



Arduino-Makefile: do not understand how to correctly use USER_LIB_PATH

Asked 6 years, 1 month ago   Modified 5 years, 5 months ago   Viewed 1k times

0

I'm using the Arduino-Makefile command-line-interface package on Linux, and recently I made an Arduino program that uses dht.h (a non-standard Arduino library) that I placed in directory /home/ramces/Sketchbook/libraries

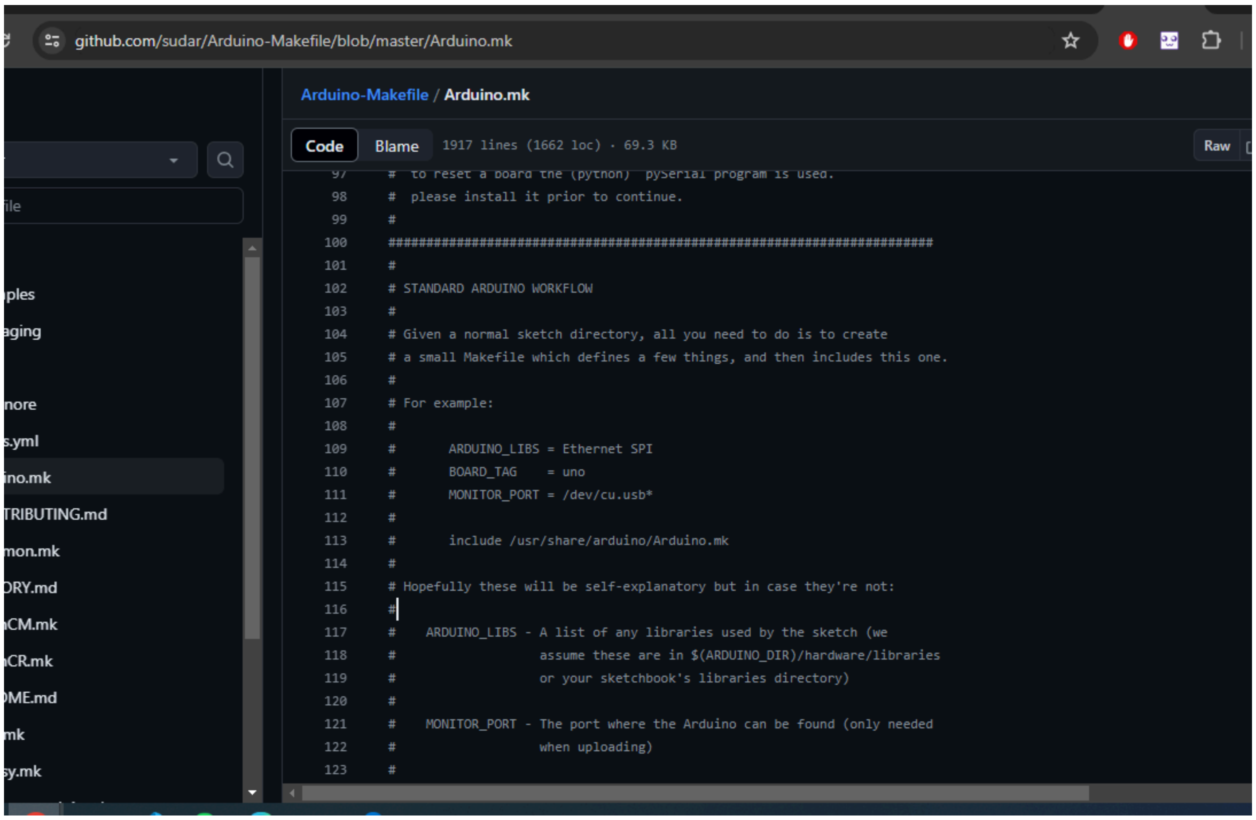
my program includes the line:

```
#include <dht.h>
```

This is my makefile:

```
ARDUINO_DIR = /usr/share/arduino
ARDUINO_PORT = /dev/ttyACM*
USER_LIB_PATH = /home/ramces/Sketchbook/libraries/DHT/ dht.h
BOARD_TAG = uno
ARDUINO_LIBS = LiquidCrystal
include /usr/share/arduino/Arduino.mk
```

non-standard libraries are imported with the USER_LIB_PATH variable, however, the issue is that I don't know how to properly use USER_LIB_PATH. I looked at the documentation on git-hub but I am still not sure how USER_LIB_PATH is used correctly.

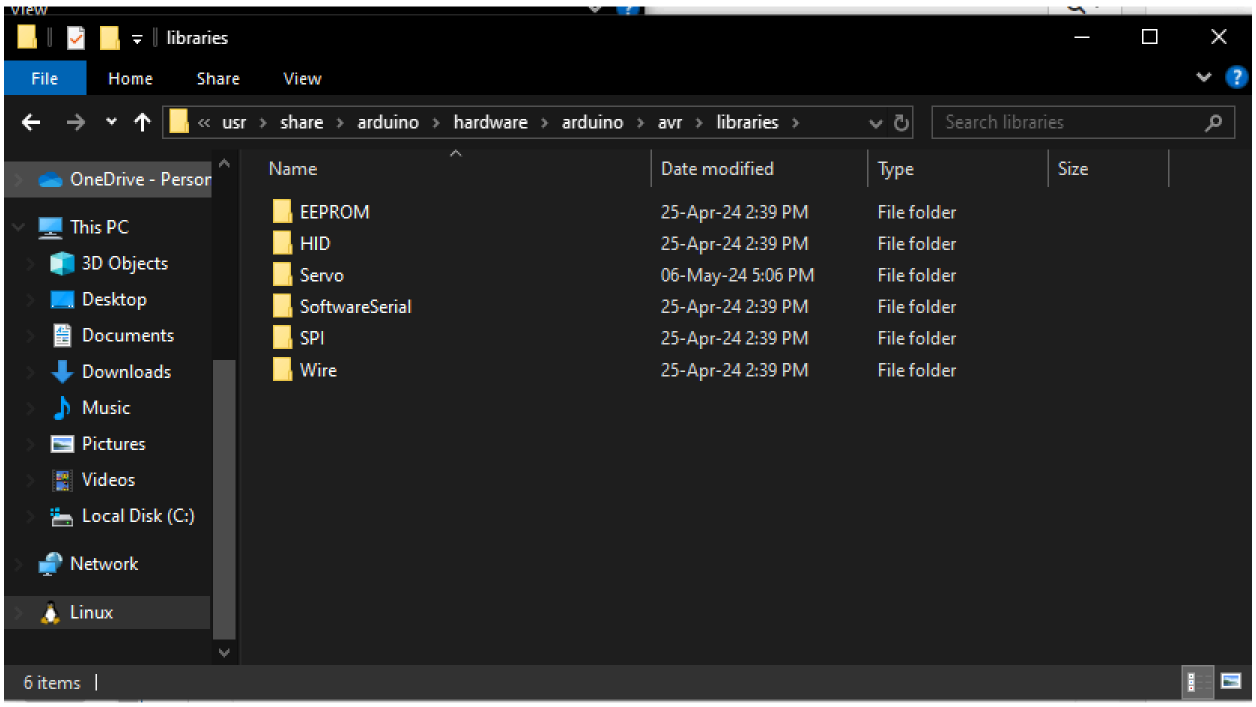


github.com/sudar/Arduino-Makefile/blob/master/Arduino.mk

Arduino-Makefile / Arduino.mk

Code   Blame   1917 lines (1662 loc) · 69.3 KB   Raw

```
#  to reset a board the (python)  pySerial program is used.
#  please install it prior to continue.
#
########################################################################
#
# STANDARD ARDUINO WORKFLOW
#
# Given a normal sketch directory, all you need to do is to create
# a small Makefile which defines a few things, and then includes this one.
#
# For example:
#
#       ARDUINO_LIBS = Ethernet SPI
#       BOARD_TAG    = uno
#       MONITOR_PORT = /dev/cu.usb*
#
#       include /usr/share/arduino/Arduino.mk
#
# Hopefully these will be self-explanatory but in case they're not:
#
#    ARDUINO_LIBS - A list of any libraries used by the sketch (we
#                   assume these are in $(ARDUINO_DIR)/hardware/libraries
#                   or your sketchbook's libraries directory)
#
#    MONITOR_PORT - The port where the Arduino can be found (only needed
#                   when uploading)
#
```



libraries

File Home Share View

« usr › share › arduino › hardware › arduino › avr › libraries ›

Name	Date modified	Type	Size
EEPROM	25-Apr-24 2:39 PM	File folder	
HID	25-Apr-24 2:39 PM	File folder	
Servo	06-May-24 5:06 PM	File folder	
SoftwareSerial	25-Apr-24 2:39 PM	File folder	
SPI	25-Apr-24 2:39 PM	File folder	
Wire	25-Apr-24 2:39 PM	File folder	

6 items

Code

```
#include "Servo.h"

#define WHITE_LED 7
#define GREEN_LED 8
#define RED_LED 9
#define PIR_PIN 2
#define LDR_PIN A0
#define BUZZER_PIN 10
#define SERVO_PIN 3
#define BUTTON_PIN 4

Servo myServo;

int sensorState = 0;
int LDRValue = 0;
int systemState = 0; //the whole smart home security system
int nightState = 0;
int lockState = 0;
int unlocking = 0;

String password = "12345";

void setup() {
  // put your setup code here, to run once:
  pinMode(WHITE_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);
  pinMode(RED_LED, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);

  pinMode(PIR_PIN, INPUT);
  pinMode(BUTTON_PIN, INPUT);

  myServo.attach(SERVO_PIN);

  Serial.begin(9600);

  digitalWrite(GREEN_LED, HIGH);

}

void loop() {

  while(unlocking == 1){
    while(Serial.available() == 0){
      //Serial.println("waiting for pass");
    }

    String pass = Serial.readString();
    pass.trim();
    if(pass == password){
      //unlock the system
      lockState = 0;
      myServo.write(0);
      noTone(BUZZER_PIN);
      unlocking = 0;

      digitalWrite(GREEN_LED, HIGH);
      digitalWrite(RED_LED, LOW);
    }
    else{
      Serial.println("Password incorrect");
    }
  }

  Serial.println("What would you like to do?");
  Serial.println("1. System Off");
  Serial.println("2. System On");
  Serial.println("3. Lock");
  Serial.println("4. Unlock");

  while(Serial.available() == 0){

    if(digitalRead(BUTTON_PIN) == HIGH){
      digitalWrite(RED_LED, HIGH);
      digitalWrite(GREEN_LED, LOW);
    }

    //system is on
    if(systemState == 1){

      //LDR
      LDRValue = analogRead(LDR_PIN);
      //Serial.println(LDRValue);

      if(LDRValue < 500){
        //night is coming, turn on the spotlight
        nightState = 1;

      }else{
        //night is ending, turn off the spotlight to save power
        //only turn off the spotlight not the movement sensor
        //because there are thieves in the day time and you need to
continue detecting in day time
        nightState = 0;

        //always set to low
        digitalWrite(WHITE_LED, LOW);
      }


      // put your main code here, to run repeatedly:
      sensorState = digitalRead(PIR_PIN);

      if(sensorState == HIGH){

        if(nightState == 1){
          digitalWrite(WHITE_LED, HIGH);
        }
      }
      else{
        digitalWrite(WHITE_LED, LOW);
      }

    }

  }

  int input = Serial.parseInt();

  switch(input){
    case 1:
      if(systemState == 0){

        //system already down, no need set again
        Serial.println("System is already down.");
        //return;

      }
      else{

        //system is on, turn off system and sensors
        systemState = 0;
        lockState = 0;

        digitalWrite(WHITE_LED, LOW);
        myServo.write(0);
```

```
        noTone(BUZZER_PIN);

      }
      break;
    case 2:
      if(systemState == 1){

        //system already down, no need set again
        Serial.println("System is already up.");
        //return;

      }
      else{

        //system is on, turn off system
        systemState = 1;

      }
      break;
    case 3:
      if(systemState == 0){
        Serial.println("The system is down. Please turn on the system
first");
      }
      else{

        if(lockState == 0){

          //lock the thieves inside the house, and turn on the
siren(buzzer) to alert the security
          lockState = 1;
          myServo.write(180);
          tone(BUZZER_PIN, 1000);

        }
        else{
          Serial.println("The system is already locked.");
        }

      }
      break;
    case 4:

      if(systemState == 0){
        Serial.println("The system is down. Please turn on the system
first");
      }
      else{

        if(lockState == 1){

          Serial.println("Please input the password: ");
          unlocking = 1;

        }
        else{
          Serial.println("The system is already unlocked.");
        }

      }

      break;
  }

  delay(1000);

}
```

Video:

I uploaded the video to my dummy youtube channel “zackchong2778” - Link