**#include <ESP8266\_Seniverse.h>**

**#include <Hash.h>**

**#include <stack>**

**#include "Alert.h"**

**#include "OLED.h"**

**#include "Tool.h"**

**#include "Universal.h"**

**#include "WebServer.h"**

**ESP8266WebServer server(ServerPort);**

**ALERT alert;**

**OLED oled;**

**HTTPClient http;**

**WiFiClient client;**

**TOOL tool;**

**Ticker TimeRefresh\_ticker;**

**Ticker System\_time;**

**Ticker Desktop\_ticker;**

**Ticker CMDControlPanel\_ticker;**

**Ticker WIFI\_Test;**

**bool Charging\_State = false;**

**bool WIFI\_State = false;**

**bool Developer\_Mode = false;**

**bool allowResponse = true;**

**bool allowDownloadMode = true;**

**bool freezeMode = false;**

**bool diskMode = false;**

**bool CMDCP\_State = false;**

**class FlashFileSystem {**

**private:**

**FSInfo Flash\_info;**

**Dir FileDirectory;**

**stack<String> WorkingDirectoryStack;**

**String WorkingDirectory = "**

**void deleteFolder(String path) {**

**Dir FileDirectory = LittleFS.openDir(path);**

**while (FileDirectory.next()) {**

**String entryPath = path + "**

**if (LittleFS.exists(entryPath)) {**

**deleteFolder(entryPath);**

**} else {**

**LittleFS.remove(entryPath);**

**}**

**}**

**LittleFS.rmdir(path);**

**}**

**public:**

**String getFlash\_info() {**

**LittleFS.begin();**

**LittleFS.info(Flash\_info);**

**Serial.print("totalBytes: ");**

**Serial.print(Flash\_info.totalBytes);**

**Serial.println(" Bytes");**

**Serial.print("usedBytes: ");**

**Serial.print(Flash\_info.usedBytes);**

**Serial.println(" Bytes");**

**Serial.print("Proportion: ");**

**Serial.print(Proportion);**

**Serial.println(" %");**

**Serial.print("maxPathLength: ");**

**Serial.println(Flash\_info.maxPathLength);**

**Serial.print("maxOpenFiles: ");**

**Serial.println(Flash\_info.maxOpenFiles);**

**Serial.print("blockSize: ");**

**Serial.println(Flash\_info.blockSize);**

**Serial.print("pageSize: ");**

**Serial.println(Flash\_info.pageSize);**

**\***

**unsigned char Proportion =**

**static\_cast<unsigned char>(round((static\_cast<float>(Flash\_info.usedBytes)**

**String ProportionBar = "[          ] ";**

**for (unsigned char i = 1; i < static\_cast<unsigned char>(0.1 \* Proportion); ++i) ProportionBar[i] = '=';**

**return "Flash Info (Bytes)\nTotal:" + String(Flash\_info.totalBytes) + "\nUsed: " + String(Flash\_info.usedBytes) + "\n" + ProportionBar +**

**String(Proportion) + "%\nMaxPathLength:" + String(Flash\_info.maxPathLength) + "\nMaxOpenFiles:" + String(Flash\_info.maxOpenFiles) +**

**"\nBlockSize:" + String(Flash\_info.blockSize) + "\nPageSize:" + String(Flash\_info.pageSize);**

**}**

**String getWorkDirectory() { return WorkingDirectory; }**

**void backDirectory() {**

**if (!WorkingDirectoryStack.empty()) {**

**WorkingDirectory = WorkingDirectoryStack.top();**

**WorkingDirectoryStack.pop();**

**}**

**}**

**void changeDirectory(String path) {**

**WorkingDirectoryStack.push(path);**

**WorkingDirectory = path;**

**}**

**String listDirectoryContents() {**

**String path = WorkingDirectory;**

**LittleFS.begin();**

**FileDirectory = LittleFS.openDir(path.c\_str());**

**String FlashlistDirectory = "";**

**while (FileDirectory.next()) {**

**FlashlistDirectory += path + FileDirectory.fileName() + "\n";**

**}**

**return FlashlistDirectory;**

**}**

**String readFile(String fileName, String filePath = "") {**

**String path;**

**if (filePath == "")**

**path = WorkingDirectory + fileName;**

**else**

**path = filePath;**

**LittleFS.begin();**

**File dataFile;**

**String File\_Info = "";**

**if (LittleFS.exists(path.c\_str())) {**

**File dataFile = LittleFS.open(path.c\_str(), "r");**

**while (dataFile.available()) {**

**File\_Info += (char)dataFile.read();**

**}**

**dataFile.close();**

**} else {**

**File\_Info = "[FLASH FILE NOT FOUND]: " + path;**

**}**

**return File\_Info;**

**}**

**void fileAppend(String text, String fileName, String filePath = "") {**

**String path;**

**if (filePath == "")**

**path = WorkingDirectory + fileName;**

**else**

**path = filePath;**

**LittleFS.begin();**

**File dataFile;**

**if (LittleFS.exists(path)) {**

**dataFile = LittleFS.open(path.c\_str(), "a");**

**} else {**

**dataFile = LittleFS.open(path.c\_str(), "w");**

**}**

**dataFile.print(text);**

**dataFile.close();**

**}**

**void fileCover(String text, String fileName, String filePath = "") {**

**String path;**

**if (filePath == "")**

**path = WorkingDirectory + fileName;**

**else**

**path = filePath;**

**LittleFS.begin();**

**File dataFile;**

**dataFile = LittleFS.open(path.c\_str(), "w");**

**dataFile.print(text);**

**dataFile.close();**

**}**

**void createFile(String fileName) {**

**String path = WorkingDirectory + fileName;**

**LittleFS.begin();**

**File dataFile;**

**dataFile = LittleFS.open(path.c\_str(), "w");**

**dataFile.println("");**

**dataFile.close();**

**}**

**void makeDirector(String dirName) {**

**String path = WorkingDirectory + dirName;**

**LittleFS.begin();**

**LittleFS.mkdir(path.c\_str());**

**}**

**bool removeFile(String fileName, String filePath = "") {**

**String path;**

**if (filePath == "")**

**path = WorkingDirectory + fileName;**

**else**

**path = filePath;**

**LittleFS.begin();**

**if (LittleFS.remove(path.c\_str())) {**

**return true;**

**} else {**

**return false;**

**}**

**}**

**void removeDirector(String dirName, String dirPath = "") {**

**String path;**

**if (dirPath == "")**

**path = WorkingDirectory + dirName;**

**else**

**path = dirPath;**

**LittleFS.begin();**

**deleteFolder(path);**

**}**

**void copyFile(String sourceFilePath, String targetFilePath, bool moveMode = false) {**

**LittleFS.begin();**

**File source = LittleFS.open(sourceFilePath, "r");**

**File target = LittleFS.open(targetFilePath, "w");**

**这段代码将从源文件source读取的数据写入目标文件target。**

**它通过使用 source.available() 方法来检查源文件是否还有未读取的数据，并使用 target.write(source.read()) 方法将读取的数据写入目标文件。**

**这段代码的前提是：源文件source和目标文件target已经打开。**

**\***

**if (source && target) {**

**while (source.available()) {**

**target.write(source.read());**

**}**

**}**

**source.close();**

**target.close();**

**if (moveMode) LittleFS.remove(sourceFilePath.c\_str());**

**}**

**void copyDir(String sourceDirPath, String targetDirPath, bool moveMode = false) {**

**LittleFS.begin();**

**if (!LittleFS.exists(sourceDirPath)) return;**

**if (!LittleFS.exists(targetDirPath)) LittleFS.mkdir(targetDirPath);**

**Dir sourceDir = LittleFS.openDir(sourceDirPath);**

**while (sourceDir.next()) {**

**String sourceFilePath = sourceDirPath + "**

**String targetFilePath = targetDirPath + "**

**if (sourceDir.isDirectory()) {**

**copyDir(sourceFilePath, targetFilePath, moveMode);**

**} else {**

**copyFile(sourceFilePath, targetFilePath, moveMode);**

**}**

**}**

**if (moveMode) deleteFolder(sourceDirPath);**

**}**

**这段代码是一个查找文件的函数。该函数接收两个参数：dirPath (目录路径) 和 fileName (待查找文件名称)。该函数执行以下步骤：**

**启动闪存文件系统**

**打开目录dirPath**

**循环读取目录中的所有文件**

**a. 如果读取的是目录，递归调用该函数，并将其结果加入foundFile字符串。**

**b. 如果读取的是文件：**

**i. 以"."分割该文件的文件名，以得到其扩展名。**

**ii. 比较待查找文件名和该文件的文件名：**

**如果待查找文件名为"."，说明查找所有文件，不进行筛选。**

**如果待查找文件名为"\*.txt"，说明查找所有扩展名为"txt"的文件，按扩展名筛选。**

**如果待查找文件名为"a.txt"，说明查找文件名为"a.txt"的文件，按文件名筛选。**

**返回找到的文件路径列表(存储在foundFile字符串中)**

**注：**

**LittleFS.begin()是闪存文件系统的初始化函数。**

**LittleFS.openDir(dirPath)是打开目录的函数。**

**dir.next()是读取下一个文件**

**\***

**String findFiles(String dirPath, String fileName) {**

**String foundFile = "";**

**vector<String> targetName = oled.strsplit(fileName, ".");**

**LittleFS.begin();**

**Dir dir = LittleFS.openDir(dirPath);**

**while (dir.next()) {**

**if (dir.isDirectory()) {**

**foundFile += findFiles(dirPath + dir.fileName() + "**

**} else {**

**String foundName = dir.fileName();**

**vector<String> foundName\_Split = oled.strsplit(foundName, ".");**

**if (targetName[0] == "\*" && targetName[1] == "\*") {**

**foundFile += dirPath + foundName + "\n";**

**} else if (targetName[0] == "\*" && targetName[1] != "\*") {**

**if (foundName\_Split[1] == targetName[1]) foundFile += dirPath + foundName + "\n";**

**} else if (targetName[0] != "\*" && targetName[1] != "\*") {**

**if (foundName == fileName) foundFile += dirPath + foundName + "\n";**

**}**

**}**

**}**

**return foundFile;**

**}**

**} FFileS;**

**class TimeRefresh {**

**public:**

**typedef struct SystemTime {**

**unsigned short year = 0;**

**unsigned char month = 0;**

**unsigned char day = 0;**

**unsigned char hour = 0;**

**unsigned char minute = 0;**

**unsigned char second = 0;**

**void updateTime() {**

**second++;**

**if (second >= 60) {**

**second = 0;**

**minute++;**

**}**

**if (minute >= 60) {**

**minute = 0;**

**hour++;**

**}**

**if (hour >= 24) {**

**hour = 0;**

**day++;**

**}**

**unsigned char daysInMonth = 31;**

**if (month == 2) {**

**daysInMonth = (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0)) ? 29 : 28;**

**} else if (month == 4 || month == 6 || month == 9 || month == 11) {**

**daysInMonth = 30;**

**}**

**if (day > daysInMonth) {**

**day = 1;**

**month++;**

**if (month > 12) {**

**month = 1;**

**year++;**

**}**

**}**

**}**

**void setSystemTime(String netWorkTimeStr) {**

**year = static\_cast<unsigned short>(netWorkTimeStr.substring(0, 4).toInt());**

**month = static\_cast<unsigned char>(netWorkTimeStr.substring(4, 6).toInt());**

**day = static\_cast<unsigned char>(netWorkTimeStr.substring(6, 8).toInt());**

**hour = static\_cast<unsigned char>(netWorkTimeStr.substring(8, 10).toInt());**

**minute = static\_cast<unsigned char>(netWorkTimeStr.substring(10, 12).toInt());**

**second = static\_cast<unsigned char>(netWorkTimeStr.substring(12, 14).toInt());**

**}**

**} SystemTime;**

**SystemTime sysTime;**

**String netWorkTimeStr = "";**

**bool allow = false;**

**void begin() {**

**TimeRefresh\_ticker.attach(600, [this](void) -> void {**

**if (freezeMode == true) TimeRefresh\_ticker.detach();**

**if (digitalRead(SENOUT) == HIGH) allow = true;**

**});**

**System\_time.attach(1, [this](void) -> void { sysTime.updateTime(); });**

**}**

**void getNetWorkTime() {**

**uint8 httpCode = http.GET();**

**if (httpCode > 0) {**

**if (httpCode == HTTP\_CODE\_OK) {**

**netWorkTimeStr = http.getString();**

**StaticJsonDocument<200> doc;**

**deserializeJson(doc, netWorkTimeStr);**

**netWorkTimeStr = doc["sysTime1"].as<String>();**

**sysTime.setSystemTime(netWorkTimeStr);**

**} else {**

**Serial.printf("[HTTP GET Failed] ErrorCode: %s\n", http.errorToString(httpCode).c\_str());**

**}**

**} else {**

**Serial.printf("[HTTP GET Failed] ErrorCode: %s\n", http.errorToString(httpCode).c\_str());**

**}**

**http.end();**

**}**

**String format(unsigned char timeInt) {**

**if (timeInt < 10) {**

**return "0" + String(timeInt);**

**} else {**

**return String(timeInt);**

**}**

**}**

**String timeRead(bool mode = true) {**

**if (mode == true) {**

**return format(sysTime.hour) + ":" + format(sysTime.minute);**

**} else {**

**return String(sysTime.year) + format(sysTime.month) + format(sysTime.day) + format(sysTime.hour) + format(sysTime.minute) +**

**format(sysTime.second);**

**}**

**}**

**} timeRef;**

**typedef struct ANIM\_INDEX {**

**unsigned short name;**

**unsigned short Duration;**

**unsigned short Begin;**

**unsigned short End;**

**unsigned char IMG\_Width;**

**unsigned char IMG\_Hight;**

**} ANIM\_INDEX;**

**class Animation {**

**private:**

**ANIM\_INDEX Index;**

**POINT Pos = {0, 0};**

**unsigned short Duration = UINT16\_MIN;**

**unsigned short Begin = UINT16\_MAX;**

**unsigned short End = UINT16\_MIN;**

**void AnimController() {**

**if (Duration == UINT16\_MIN) Duration = Index.Duration;**

**if (Begin == UINT16\_MAX) Begin = Index.Begin;**

**if (End == UINT16\_MIN) End = Index.End;**

**unsigned long Sleep\_ms = static\_cast<unsigned long>(static\_cast<float>(Duration)**

**if (Index.name == loading\_X16\_30F.name) {**

**for (unsigned short i = Begin; i < End; ++i) {**

**oled.OLED\_DrawBMP(Pos.x, Pos.y, Index.IMG\_Width, Index.IMG\_Hight, Loading\_X16\_30F[i]);**

**delay(Sleep\_ms);**

**}**

**}**

**if (Index.name == loading\_X16\_60F.name) {**

**for (unsigned short i = Begin; i < End; ++i) {**

**oled.OLED\_DrawBMP(Pos.x, Pos.y, Index.IMG\_Width, Index.IMG\_Hight, Loading\_X16\_60F[i]);**

**delay(Sleep\_ms);**

**}**

**}**

**if (Index.name == loadingBackForthBar\_60x8\_60F.name) {**

**for (unsigned short i = Begin; i < End; ++i) {**

**oled.OLED\_DrawBMP(Pos.x, Pos.y, Index.IMG\_Width, Index.IMG\_Hight, LoadingBackForthBar\_60x8\_60F[i]);**

**delay(Sleep\_ms);**

**}**

**}**

**if (Index.name == loadingBar\_60x8\_30F.name) {**

**for (unsigned short i = Begin; i < End; ++i) {**

**oled.OLED\_DrawBMP(Pos.x, Pos.y, Index.IMG\_Width, Index.IMG\_Hight, LoadingBar\_60x8\_30F[i]);**

**delay(Sleep\_ms);**

**}**

**}**

**if (Index.name == loadingBar\_60x8\_60F.name) {**

**for (unsigned short i = Begin; i < End; ++i) {**

**oled.OLED\_DrawBMP(Pos.x, Pos.y, Index.IMG\_Width, Index.IMG\_Hight, LoadingBar\_60x8\_60F[i]);**

**delay(Sleep\_ms);**

**}**

**}**

**}**

**public:**

**ANIM\_INDEX loading\_X16\_30F = {0, 250, 0, 30, 16, 16};**

**ANIM\_INDEX loading\_X16\_60F = {1, 500, 0, 60, 16, 16};**

**ANIM\_INDEX loadingBackForthBar\_60x8\_60F = {2, 500, 0, 60, 60, 8};**

**ANIM\_INDEX loadingBar\_60x8\_30F = {3, 300, 0, 30, 60, 8};**

**ANIM\_INDEX loadingBar\_60x8\_60F = {4, 500, 0, 60, 60, 8};**

**void setAnimation(u8 x, u8 y, u16 duration = UINT16\_MIN, u16 begin = UINT16\_MAX, u16 end = UINT16\_MIN) {**

**Pos = {x, y};**

**Duration = duration;**

**Begin = begin;**

**End = end;**

**}**

**void runAnimation(ANIM\_INDEX index) {**

**Index = index;**

**AnimController();**

**}**

**} anim;**

**void ShowFireWarning() {**

**oled.OLED\_DrawBMP(0, 0, 32, 64, FireWarning\_32x64[0]);**

**oled.OLED\_DrawBMP(32, 0, 32, 64, FireWarning\_32x64[1]);**

**oled.OLED\_DrawBMP(64, 0, 32, 64, FireWarning\_32x64[2]);**

**oled.OLED\_DrawBMP(96, 0, 32, 64, FireWarning\_32x64[3]);**

**}**

**class Desktop {**

**private:**

**typedef struct StatusBars\_Ranked {**

**unsigned char unit = 16;**

**bool Register\_State[8] = {false, false, false, false, false, false, false, false};**

**int StatusBars\_Pos[8] = {-16, -16, -16, -16, -16, -16, -16, -16};**

**unsigned char Clear\_Icon = 6;**

**unsigned char Charging = 0;**

**unsigned char WIFI = 1;**

**unsigned char ProgramDownload = 2;**

**unsigned char Disconnected = 3;**

**unsigned char Battery = 4;**

**unsigned char CMDCP = 5;**

**} StatusBars\_Ranked;**

**StatusBars\_Ranked SBR;**

**void Icon\_Register(unsigned char name, bool mode) {**

**if (mode == false && SBR.Register\_State[name] == true) {**

**SBR.Register\_State[name] = false;**

**for (auto& i : SBR.StatusBars\_Pos)**

**if (i > SBR.StatusBars\_Pos[name]) i -= SBR.unit;**

**for (unsigned char i = SBR.StatusBars\_Pos[name]; i <= 112; i += SBR.unit) oled.OLED\_DrawBMP(i, 6, 16, 16, StatusBars[SBR.Clear\_Icon]);**

**SBR.StatusBars\_Pos[name] = 0;**

**} else if (mode == true && SBR.Register\_State[name] == false) {**

**SBR.Register\_State[name] = true;**

**SBR.StatusBars\_Pos[name] = tool.findArrMax(SBR.StatusBars\_Pos, 8) + SBR.unit;**

**}**

**if (mode == true) {**

**oled.OLED\_DrawBMP(SBR.StatusBars\_Pos[name], 6, 16, 16, StatusBars[name]);**

**}**

**}**

**void StatusBars\_Render() {**

**if (Charging\_State == true) {**

**Icon\_Register(SBR.Charging, true);**

**} else {**

**Icon\_Register(SBR.Charging, false);**

**}**

**if (WIFI\_State == true) {**

**Icon\_Register(SBR.WIFI, true);**

**} else {**

**Icon\_Register(SBR.WIFI, false);**

**}**

**if (digitalRead(0) == LOW) {**

**Icon\_Register(SBR.ProgramDownload, true);**

**} else {**

**Icon\_Register(SBR.ProgramDownload, false);**

**}**

**if (WIFI\_State == false) {**

**Icon\_Register(SBR.Disconnected, true);**

**} else {**

**Icon\_Register(SBR.Disconnected, false);**

**}**

**if (Charging\_State == false) {**

**Icon\_Register(SBR.Battery, true);**

**} else {**

**Icon\_Register(SBR.Battery, false);**

**}**

**if (CMDCP\_State == true) {**

**Icon\_Register(SBR.CMDCP, true);**

**} else {**

**Icon\_Register(SBR.CMDCP, false);**

**}**

**}**

**public:**

**void begin() {**

**Desktop\_ticker.attach\_ms(500, [this](void) -> void {**

**if (freezeMode == true) Desktop\_ticker.detach();**

**if (digitalRead(SENOUT) == HIGH && Developer\_Mode == false) {**

**if (timeRef.sysTime.second == 0) oled.OLED\_ShowString(4, -1, timeRef.timeRead().c\_str(), 49);**

**StatusBars\_Render();**

**}**

**});**

**}**

**void Main\_Desktop() {**

**oled.OLED\_ShowString(4, -1, timeRef.timeRead().c\_str(), 49);**

**StatusBars\_Render();**

**}**

**} Desktop;**

**void Show\_Charging\_info() {**

**if (digitalRead(CHRG) == LOW && Charging\_State == false) {**

**Charging\_State = true;**

**anim.setAnimation(112, 6, 400);**

**for (u8 i = 0; i < 2; ++i) anim.runAnimation(anim.loading\_X16\_60F);**

**oled.OLED\_DrawBMP(112, 6, 16, 16, Loading\_X16\_60F[60]);**

**delay(1000);**

**oled.OLED\_DrawBMP(0, 0, 128, 64, Charging\_IMG);**

**delay(1000);**

**oled.OLED\_Clear();**

**Main\_Desktop();**

**\***

**} else if (digitalRead(CHRG) == HIGH && Charging\_State == true) {**

**Charging\_State = false;**

**}**

**}**

**void ProgramDownloadMode() {**

**if (digitalRead(Decoder\_C) == LOW && allowDownloadMode == true && freezeMode == false) {**

**bool allowReset = true;**

**oled.OLED\_DrawBMP(0, 0, 128, 48, DownloadMode\_IMG);**

**anim.setAnimation(112, 6, 400);**

**for (u8 i = 0; i < 6; ++i) {**

**anim.runAnimation(anim.loading\_X16\_60F);**

**if (digitalRead(0) == HIGH) {**

**oled.OLED\_DrawBMP(112, 6, 16, 16, Loading\_X16\_60F[60]);**

**allowReset = false;**

**break;**

**}**

**}**

**if (allowReset == true) {**

**oled.OLED\_Display\_Off();**

**digitalWrite(RST, LOW);**

**}**

**}**

**}**

**class SystemSleep {**

**private:**

**public:**

**String GPIO\_Read() {**

**int decoderC = digitalRead(Decoder\_C);**

**int decoderB = digitalRead(Decoder\_B);**

**int decoderA = digitalRead(Decoder\_A);**

**String decodedWith = "\nDecoded\_with=";**

**switch (decoderC << 2 | decoderB << 1 | decoderA) {**

**case 1:**

**decodedWith += "Buzzer\_Enable";**

**break;**

**case 2:**

**decodedWith += "RedLED\_Enable";**

**break;**

**case 3:**

**decodedWith += "GreenLED\_Enable";**

**break;**

**case 4:**

**decodedWith += "BlueLED\_Enable";**

**break;**

**case 6:**

**decodedWith += "Sensor\_and\_OLED\_disabled";**

**break;**

**default:**

**decodedWith += "NULL";**

**break;**

**}**

**String GPIO\_State = "RST=" + String(digitalRead(RST)) + "\nTXD=" + String(digitalRead(TXD)) + "\nRXD=" + String(digitalRead(RXD)) +**

**"\nSCL=" + String(digitalRead(SCL)) + "\nSDA=" + String(digitalRead(SDA)) + "\nCHRG=" + String(digitalRead(CHRG)) +**

**"\nLOWPOWER=" + String(digitalRead(LOWPOWER)) + "\nSENOUT=" + String(digitalRead(SENOUT)) + "\nDecoder\_C=" + String(decoderC) +**

**"\nDecoder\_B=" + String(decoderB) + "\nDecoder\_A=" + String(decoderA) + decodedWith;**

**return GPIO\_State;**

**}**

**String getSysModeAndStatus() {**

**return "Charging\_State=" + String(Charging\_State) + "\nWIFI\_State=" + String(WIFI\_State) + "\nDeveloper\_Mode=" + String(Developer\_Mode) +**

**"\nallowResponse=" + String(allowResponse) + "\nallowDownloadMode=" + String(allowDownloadMode) + "\nfreezeMode=" + String(freezeMode) +**

**"\ndiskMode=" + String(diskMode);**

**}**

**void removeSleepFile() {**

**if (LittleFS.exists("**

**}**

**void resumeFromDeepSleep() {**

**LittleFS.begin();**

**String SleepFile = "";**

**if (LittleFS.exists("**

**SleepFile = FFileS.readFile("", "**

**for (auto& i : oled.strsplit(SleepFile, "\n")) {**

**vector<String> GPIO\_Status = oled.strsplit(i, "=");**

**if (GPIO\_Status[0] == "Decoder\_C") digitalWrite(Decoder\_C, GPIO\_Status[1].toInt());**

**if (GPIO\_Status[0] == "Decoder\_B") digitalWrite(Decoder\_B, GPIO\_Status[1].toInt());**

**if (GPIO\_Status[0] == "Decoder\_A") digitalWrite(Decoder\_A, GPIO\_Status[1].toInt());**

**}**

**FFileS.removeFile("", "**

**}**

**if (LittleFS.exists("**

**SleepFile = FFileS.readFile("", "**

**for (auto& i : oled.strsplit(SleepFile, "\n")) {**

**vector<String> SysModeAndStatus = oled.strsplit(i, "=");**

**if (SysModeAndStatus[0] == "Developer\_Mode") Developer\_Mode = (SysModeAndStatus[1].toInt() != 0);**

**if (SysModeAndStatus[0] == "allowResponse") allowResponse = (SysModeAndStatus[1].toInt() != 0);**

**if (SysModeAndStatus[0] == "allowDownloadMode") allowDownloadMode = (SysModeAndStatus[1].toInt() != 0);**

**if (SysModeAndStatus[0] == "freezeMode") freezeMode = (SysModeAndStatus[1].toInt() != 0);**

**if (SysModeAndStatus[0] == "diskMode") diskMode = (SysModeAndStatus[1].toInt() != 0);**

**}**

**FFileS.removeFile("", "**

**}**

**diskMode = false;**

**}**

**void Sys\_freezeMode(bool Enable = true) {**

**if (Enable == true) {**

**freezeMode = true;**

**oled.OLED\_Display\_Off();**

**digitalWrite(Decoder\_C, HIGH);**

**digitalWrite(Decoder\_B, HIGH);**

**digitalWrite(Decoder\_A, LOW);**

**WiFi.setSleepMode(WIFI\_MODEM\_SLEEP);**

**} else if (Enable == false) {**

**WiFi.setSleepMode(WIFI\_NONE\_SLEEP);**

**freezeMode = false;**

**digitalWrite(Decoder\_C, HIGH);**

**digitalWrite(Decoder\_B, HIGH);**

**digitalWrite(Decoder\_A, HIGH);**

**for (uint8 i = 0; i < 10; ++i) {**

**delay(1000);**

**if (digitalRead(SENOUT) == HIGH) break;**

**}**

**oled.OLED\_Init();**

**oled.OLED\_ColorTurn(0);**

**oled.OLED\_DisplayTurn(0);**

**timeRef.getNetWorkTime();**

**oled.OLED\_Clear();**

**Desktop.Main\_Desktop();**

**timeRef.begin();**

**Desktop.begin();**

**}**

**}**

**void Sys\_diskMode(uint64\_t time\_us = 0) {**

**CMDCP\_State = false;**

**Developer\_Mode = false;**

**diskMode = true;**

**LittleFS.begin();**

**File GPIO\_StatusFile = LittleFS.open("**

**GPIO\_StatusFile.print(GPIO\_Read());**

**GPIO\_StatusFile.close();**

**File SysModeAndStatusFile = LittleFS.open("**

**SysModeAndStatusFile.print(getSysModeAndStatus());**

**SysModeAndStatusFile.close();**

**File PrintBoxFile = LittleFS.open("**

**for (auto& i : oled.getPrintBox()) {**

**PrintBoxFile.print(i + "\n");**

**}**

**PrintBoxFile.close();**

**ESP.deepSleep(time\_us);**

**}**

**} SysSleep;**

**class CMDControlPanel {**

**private:**

**String CMD = "";**

**String PassWord = "";**

**bool LockerState = false;**

**String CMDCP\_Online\_Response = "";**

**vector<String> clientLogedIP;**

**bool PassLocker() {**

**allowResponse = false;**

**CMD = "";**

**String PassWord\_Temp1 = "", PassWord\_Temp2 = "";**

**LittleFS.begin();**

**if (LittleFS.exists("**

**for (unsigned char j = 0; j < 3; ++j) {**

**CMDCP\_Response("Set password:");**

**server.send(200, "text**

**for (unsigned char i = 0; i < 60; ++i) {**

**server.handleClient();**

**if (CMD != "") {**

**PassWord\_Temp1 = CMD;**

**CMD = "";**

**break;**

**}**

**delay(500);**

**}**

**if (PassWord\_Temp1 == "") continue;**

**CMDCP\_Response("Re-enter password:");**

**server.send(200, "text**

**for (unsigned char i = 0; i < 60; ++i) {**

**server.handleClient();**

**if (CMD != "") {**

**PassWord\_Temp2 = CMD;**

**CMD = "";**

**break;**

**}**

**delay(500);**

**}**

**if (PassWord\_Temp1 == PassWord\_Temp2 && PassWord\_Temp1 != "") {**

**File dataFile = LittleFS.open("**

**dataFile.print(String(sha1(PassWord\_Temp1)));**

**dataFile.close();**

**CMDCP\_Response("Accepted");**

**server.send(200, "text**

**allowResponse = true;**

**CMDCP\_State = false;**

**return false;**

**}**

**}**

**} else {**

**PassWord = FFileS.readFile("", "**

**for (unsigned char j = 0; j < 3; ++j) {**

**CMDCP\_Response("Enter password:");**

**server.send(200, "text**

**for (unsigned char i = 0; i < 60; ++i) {**

**server.handleClient();**

**if (CMD != "") {**

**PassWord\_Temp1 = String(sha1(CMD));**

**CMD = "";**

**break;**

**}**

**delay(500);**

**}**

**if (PassWord == PassWord\_Temp1) {**

**CMDCP\_Response("Passed");**

**server.send(200, "text**

**allowResponse = true;**

**return true;**

**}**

**}**

**}**

**CMDCP\_State = false;**

**allowResponse = true;**

**return false;**

**}**

**void CMDCP\_Response(String Response) {**

**CMDCP\_Online\_Response = "";**

**if (Response != "") {**

**CMDCP\_Online\_Response = Response;**

**oled.print(Response);**

**Serial.println(Response);**

**}**

**}**

**void saveCmdHistory(String CMD, String clientIP) {**

**LittleFS.begin();**

**File cmdHistory = LittleFS.open("**

**cmdHistory.print(clientIP + "-" + timeRef.timeRead(false) + "-" + CMD + "\n");**

**cmdHistory.close();**

**}**

**public:**

**bool allow = false;**

**void begin() {**

**oled.setTextBox(0, 0, 128, 48);**

**}**

**String CMDControlPanelOnlinePortal(String CMDCP\_Online\_Message) {**

**if (CMDCP\_Online\_Message == "") return "";**

**CMD = CMDCP\_Online\_Message;**

**String clientLogingIP = server.client().remoteIP().toString();**

**bool LockerState = false;**

**for (auto& i : clientLogedIP) {**

**if (i == clientLogingIP) LockerState = true;**

**}**

**if (LockerState == false && (CMD == "CMD" || CMD == "cmd" || CMD == "login")) {**

**CMDCP\_State = true;**

**LockerState = PassLocker();**

**if (LockerState == true) {**

**if (clientLogedIP.empty() == true) {**

**Developer\_Mode = true;**

**oled.setTextBox(0, 0, 128, 64);**

**}**

**clientLogedIP.push\_back(clientLogingIP);**

**LittleFS.begin();**

**File cmdLoggedInfo = LittleFS.open("**

**cmdLoggedInfo.print(clientLogingIP + "-" + timeRef.timeRead(false) + "-login\n");**

**cmdLoggedInfo.close();**

**}**

**}**

**if (LockerState == true) {**

**saveCmdHistory(CMD, clientLogingIP);**

**commandIndexer();**

**}**

**allow = false;**

**return CMDCP\_Online\_Response;**

**}**

**void CMDControlPanelSerialPortal() {**

**SerialReceived();**

**if (LockerState == false && Serial.available() == false && (CMD == "CMD" || CMD == "cmd" || CMD == "login")) {**

**LockerState = PassLocker();**

**if (LockerState == true) {**

**Developer\_Mode = true;**

**oled.setTextBox(0, 0, 128, 64);**

**}**

**}**

**if (LockerState == true) commandIndexer();**

**allow = false;**

**}**

**void SerialReceived() {**

**CMD = "";**

**while (Serial.available()) CMD += static\_cast<char>(Serial.read());**

**}**

**void commandIndexer() {**

**oled.print("> " + CMD);**

**Serial.println("> " + CMD);**

**vector<String> CMD\_Index = oled.strsplit(CMD, " ");**

**if (CMD\_Index[0] == "help") {**

**CMDCP\_Response(CMDCP\_HELP);**

**}**

**if (CMD\_Index[0] == "pwd") {**

**CMDCP\_Response(FFileS.getWorkDirectory());**

**}**

**if (CMD\_Index[0] == "ls") {**

**String listDirectory = FFileS.listDirectoryContents();**

**CMDCP\_Response(listDirectory);**

**}**

**{切换当前工作目录(Change directory)}cd [dirName]**

**[cd ~][cd**

**[cd -] : 返回上一个打开的目录;**

**\***

**if (CMD\_Index[0] == "cd") {**

**if (CMD\_Index[1] == "~") {**

**FFileS.changeDirectory("**

**} else if (CMD\_Index[1] == "-") {**

**FFileS.backDirectory();**

**} else {**

**if (CMD\_Index[1].charAt(CMD\_Index[1].length() - 1) != '**

**FFileS.changeDirectory(CMD\_Index[1]);**

**}**

**CMDCP\_Response("");**

**}**

**if (CMD\_Index[0] == "cat") {**

**String File\_Info = FFileS.readFile(CMD\_Index[1]);**

**CMDCP\_Response(File\_Info);**

**}**

**if (CMD\_Index[0] == "touch") {**

**FFileS.createFile(CMD\_Index[1]);**

**CMDCP\_Response("");**

**}**

**if (CMD\_Index[0] == "mkdir") {**

**FFileS.makeDirector(CMD\_Index[1]);**

**CMDCP\_Response("");**

**}**

**echo [string] ：内容打印到控制台;**

**echo [string] > [fileName] ：将内容直接覆盖到工作目录的文件中;**

**echo [string] >> [fileName] ：将内容追加到工作目录的文件中;**

**\***

**if (CMD\_Index[0] == "echo") {**

**if (CMD\_Index[2] == ">") {**

**FFileS.fileCover(CMD\_Index[1], CMD\_Index[3]);**

**CMDCP\_Response("");**

**} else if (CMD\_Index[2] == ">>") {**

**FFileS.fileAppend(CMD\_Index[1], CMD\_Index[3]);**

**CMDCP\_Response("");**

**} else {**

**CMDCP\_Response(CMD\_Index[1]);**

**}**

**}**

**{删除一个文件或者目录(Remove)}**

**rm [fileName] : 删除工作目录下的文件;**

**rm -r [dirName] : 删除工作目录下的文件夹;**

**\***

**if (CMD\_Index[0] == "rm") {**

**if (CMD\_Index[1] == "-r") {**

**FFileS.removeDirector(CMD\_Index[2]);**

**} else {**

**FFileS.removeFile(CMD\_Index[1]);**

**}**

**CMDCP\_Response("");**

**}**

**{复制一个文件或目录(Copy file)}cp [-options] [sourcePath] [targetPath];**

**cp [源文件路径] [目标文件路径] : 复制一个文件到另一个文件;**

**cp -r [源目录路径] [目标目录路径] : 复制一个目录到另一个目录;**

**\***

**if (CMD\_Index[0] == "cp") {**

**if (CMD\_Index[1] == "-r") {**

**FFileS.copyDir(CMD\_Index[2], CMD\_Index[3]);**

**} else {**

**FFileS.copyFile(CMD\_Index[1], CMD\_Index[2]);**

**}**

**CMDCP\_Response("");**

**}**

**{文件或目录改名或将文件或目录移入其它位置(Move)}mv [-options] [sourcePath] [targetPath];**

**mv [源文件路径] [目标文件路径] : 移动一个文件到另一个文件;**

**mv -r [源目录路径] [目标目录路径] : 移动一个目录到另一个目录;**

**\***

**if (CMD\_Index[0] == "mv") {**

**if (CMD\_Index[1] == "-r") {**

**FFileS.copyDir(CMD\_Index[2], CMD\_Index[3], true);**

**} else {**

**FFileS.copyFile(CMD\_Index[1], CMD\_Index[2], true);**

**}**

**CMDCP\_Response("");**

**}**

**{查找指定目录下的文件(包含子目录的文件)}find [dirPath] [fileName] : 在dirPath目录下按文件名查找文件;**

**[fileName] = \*.\* : 查找所有文件;**

**[fileName] = \*.txt : 查找所有扩展名为txt的文件;**

**[fileName] = a.txt : 查找a.txt文件;**

**\***

**if (CMD\_Index[0] == "find") {**

**if (CMD\_Index[1].charAt(CMD\_Index[1].length() - 1) != '**

**String foundFile = FFileS.findFiles(CMD\_Index[1], CMD\_Index[2]);**

**CMDCP\_Response(foundFile);**

**}**

**if (CMD\_Index[0] == "osinfo") {**

**CMDCP\_Response(GSG3\_Os\_Info);**

**oled.OLED\_DrawBMP(0, 0, 128, 48, GasSensorGen3OS\_Info);**

**}**

**if (CMD\_Index[0] == "reboot") {**

**digitalWrite(RST, LOW);**

**CMDCP\_Response("");**

**}**

**if (CMD\_Index[0] == "pios") {**

**CMDCP\_Response(SysSleep.GPIO\_Read());**

**}**

**if (CMD\_Index[0] == "pss") {**

**CMDCP\_Response(SysSleep.getSysModeAndStatus());**

**}**

**{点亮板载的RGBLED}led [color] [state]**

**led r 1**

**led g 1**

**led b 1**

**led r 0**

**led g 0**

**led b 0**

**note: 点亮红灯和绿灯会触发下载模式进而导致复位, 因此我们要先禁用下载模式;**

**\***

**if (CMD\_Index[0] == "led") {**

**if (CMD\_Index[2] == "1" || CMD\_Index[2] == "true" || CMD\_Index[2] == "enable") {**

**if (CMD\_Index[1] == "r") {**

**allowDownloadMode = false;**

**alert.LED\_R\_Enable(0, true);**

**} else if (CMD\_Index[1] == "g") {**

**allowDownloadMode = false;**

**alert.LED\_G\_Enable(0, true);**

**} else if (CMD\_Index[1] == "b") {**

**alert.LED\_B\_Enable(0, true);**

**}**

**} else if (CMD\_Index[2] == "0" || CMD\_Index[2] == "false" || CMD\_Index[2] == "disable") {**

**if (CMD\_Index[1] == "r") {**

**alert.LED\_R\_Enable(0, false);**

**allowDownloadMode = true;**

**} else if (CMD\_Index[1] == "g") {**

**alert.LED\_G\_Enable(0, false);**

**allowDownloadMode = true;**

**} else if (CMD\_Index[1] == "b") {**

**alert.LED\_B\_Enable(0, false);**

**}**

**}**

**CMDCP\_Response("");**

**}**

**{打开蜂鸣器}buzz [state]**

**state = 1**

**state = 0**

**\***

**if (CMD\_Index[0] == "buzz") {**

**if (CMD\_Index[1] == "1" || CMD\_Index[1] == "true" || CMD\_Index[1] == "enable") {**

**alert.BUZZER\_Enable(0, true);**

**} else if (CMD\_Index[1] == "0" || CMD\_Index[1] == "false" || CMD\_Index[1] == "disable") {**

**alert.BUZZER\_Enable(0, false);**

**}**

**CMDCP\_Response("");**

**}**

**if (CMD\_Index[0] == "alertdis") {**

**alert.ALERT\_Disable();**

**CMDCP\_Response("");**

**}**

**{浅休眠模式}freeze [enable]**

**[休眠模式-freeze], 冻结I**

**freeze 1**

**freeze 0**

**\***

**if (CMD\_Index[0] == "freeze") {**

**if (CMD\_Index[1] == "1" || CMD\_Index[1] == "true" || CMD\_Index[1] == "enable") {**

**SysSleep.Sys\_freezeMode(true);**

**} else if (CMD\_Index[1] == "0" || CMD\_Index[1] == "false" || CMD\_Index[1] == "disable") {**

**SysSleep.Sys\_freezeMode(false);**

**}**

**CMDCP\_Response("");**

**}**

**{深度睡眠模式}disk [time\_us]**

**[深度休眠模式-disk] 运行状态(GPIO\_Status, 系统模式和状态, 文本框信息)数据存到Flash(醒来时恢复状态), 然后ESP12F进入深度睡眠;**

**time\_us(微秒) = 0 : 无限期进入深度睡眠, 只有手动按RST复位才能恢复;**

**\***

**if (CMD\_Index[0] == "disk") {**

**if (CMD\_Index[1].toInt() != 0) SysSleep.Sys\_diskMode(CMD\_Index[1].toInt());**

**CMDCP\_Response("");**

**}**

**{显示历史执行过的命令}history [-options]**

**history : 显示历史执行过的命令;**

**history -s : (history -sleep)显示深度睡眠前执行过的命令;**

**history -c : (history -clear)清空所有的命令历史记录;**

**\***

**if (CMD\_Index[0] == "history") {**

**if (CMD\_Index[1] == "-s") {**

**CMDCP\_Response(FFileS.readFile("", "**

**} else if (CMD\_Index[1] == "-c") {**

**FFileS.removeFile("", "**

**CMDCP\_Response("");**

**} else {**

**CMDCP\_Response(FFileS.readFile("", "**

**}**

**}**

**if (CMD\_Index[0] == "who") {**

**String who = "";**

**for (auto& i : clientLogedIP) who += (i + "\n");**

**CMDCP\_Response(who);**

**}**

**{查看所有系统登录记录}last [-options];**

**last : 查看所有系统登录记录;**

**last -c : 清空登录记录;**

**\***

**if (CMD\_Index[0] == "last") {**

**if (CMD\_Index[1] == "-c") {**

**FFileS.removeFile("", "**

**CMDCP\_Response("");**

**} else {**

**CMDCP\_Response(FFileS.readFile("", "**

**}**

**}**

**{显示或设置系统时间}date [-options] [timeStr];**

**date : 显示系统时间;**

**date -n : 同步网络时间;**

**date -s [timeStr] : 根据字符串设置(set)系统时间, timeStr = 20230203121601 (Year Month Day Hour Minute Second);**

**\***

**if (CMD\_Index[0] == "date") {**

**if (CMD\_Index[1] == "-n") {**

**timeRef.getNetWorkTime();**

**CMDCP\_Response("");**

**} else if (CMD\_Index[1] == "-s") {**

**timeRef.sysTime.setSystemTime(CMD\_Index[2]);**

**CMDCP\_Response("");**

**} else {**

**CMDCP\_Response(timeRef.timeRead(false));**

**}**

**}**

**if (CMD\_Index[0] == "clear") {**

**oled.clearTextBox();**

**CMDCP\_Response("");**

**}**

**if (CMD\_Index[0] == "df") {**

**String Flash\_Info = FFileS.getFlash\_info();**

**CMDCP\_Response(Flash\_Info);**

**}**

**if (CMD\_Index[0] == "free") {**

**String FreeHeap = "FreeRAM: " + String(ESP.getFreeHeap()) + " Byte";**

**CMDCP\_Response(FreeHeap);**

**}**

**if (CMD\_Index[0] == "wifi") {**

**FFileS.fileCover(CMD\_Index[1] + "<SSID**

**CMDCP\_Response("");**

**}**

**if (CMD\_Index[0] == "poweroff") {**

**SysSleep.Sys\_diskMode(0);**

**CMDCP\_Response("");**

**}**

**{登出和锁定CMDCP}logout [-options] [clientIP];**

**logout : 自己登出, 不会影响其他终端;**

**logout -k [clientIP] : 将指定IP地址的终端登出(kill);**

**logout -k other : 登出(kill)除自己外的其他终端;**

**logout -k all : 登出所有终端;**

**\***

**auto clientLogout = [this](String clientIP) -> void {**

**clientLogedIP.erase(remove(clientLogedIP.begin(), clientLogedIP.end(), clientIP), clientLogedIP.end());**

**File cmdLoggedInfo = LittleFS.open("**

**cmdLoggedInfo.print(clientIP + "-" + timeRef.timeRead(false) + "-logout\n");**

**cmdLoggedInfo.close();**

**};**

**if (CMD\_Index[0] == "logout") {**

**LittleFS.begin();**

**if (CMD\_Index[1] == "-k") {**

**if (CMD\_Index[2] == "other") {**

**String clientLogoutIP = server.client().remoteIP().toString();**

**vector<String> allClientIP = clientLogedIP;**

**for (auto& i : allClientIP)**

**if (i != clientLogoutIP) clientLogout(i);**

**} else if (CMD\_Index[2] == "all") {**

**vector<String> allClientIP = clientLogedIP;**

**for (auto& i : allClientIP) clientLogout(i);**

**} else {**

**clientLogout(CMD\_Index[2]);**

**}**

**} else {**

**String clientLogoutIP = server.client().remoteIP().toString();**

**clientLogout(clientLogoutIP);**

**}**

**if (clientLogedIP.empty() == true) {**

**CMDCP\_State = false;**

**LockerState = false;**

**Developer\_Mode = false;**

**oled.setTextBox(0, 0, 128, 48);**

**oled.OLED\_Clear();**

**Desktop.Main\_Desktop();**

**}**

**CMDCP\_Response("");**

**}**

**CMD = "";**

**}**

**} CMDCP;**

**class WebServer {**

**public:**

**void beginServer() {**

**server.begin();**

**server.on("**

**server.on("**

**String CMDCP\_Online\_Message = server.arg("message");**

**String CMDCP\_Send\_Message = CMDCP.CMDControlPanelOnlinePortal(CMDCP\_Online\_Message);**

**if (allowResponse == true) server.send(200, "text**

**});**

**}**

**void WiFi\_Connect() {**

**if (timeRef.timeRead() == "00:00")**

**anim.setAnimation(67, 6);**

**else**

**anim.setAnimation(112, 6);**

**if (WiFi.status() != WL\_CONNECTED) {**

**WIFI\_State = false;**

**vector<String> SSID\_PASSWD = oled.strsplit(FFileS.readFile("", "**

**WiFi.begin(SSID\_PASSWD[0], SSID\_PASSWD[1]);**

**for (unsigned char i = 0; i < 100; ++i) {**

**if (WiFi.status() == WL\_CONNECTED) {**

**Serial.print("IP Address: ");**

**Serial.print(WiFi.localIP());**

**Serial.println(":" + String(ServerPort));**

**WIFI\_State = true;**

**break;**

**} else {**

**if (timeRef.timeRead() == "00:00")**

**anim.runAnimation(anim.loadingBar\_60x8\_30F);**

**else**

**anim.runAnimation(anim.loading\_X16\_60F);**

**}**

**}**

**} else {**

**WIFI\_State = true;**

**}**

**}**

**} WebServer;**

**void setup(void) {**

**Serial.begin(115200);**

**LittleFS.begin();**

**alert.AlertInit();**

**pinMode(RST, OUTPUT);**

**digitalWrite(RST, HIGH);**

**pinMode(LOWPOWER, INPUT\_PULLUP);**

**pinMode(SENOUT, INPUT\_PULLUP);**

**pinMode(CHRG, INPUT\_PULLUP);**

**oled.OLED\_Init();**

**oled.OLED\_ColorTurn(0);**

**oled.OLED\_DisplayTurn(0);**

**oled.OLED\_DrawBMP(0, 0, 128, 64, RMSHE\_IMG);**

**WebServer.WiFi\_Connect();**

**http.setTimeout(TimeOut);**

**http.begin(client, GetSysTimeUrl);**

**timeRef.getNetWorkTime();**

**oled.OLED\_Clear();**

**Desktop.Main\_Desktop();**

**WIFI\_Test.attach(300, [](void) -> void {**

**if (digitalRead(SENOUT) == HIGH && WiFi.status() != WL\_CONNECTED) WIFI\_State = false;**

**});**

**timeRef.begin();**

**Desktop.begin();**

**CMDCP.begin();**

**WebServer.beginServer();**

**SysSleep.resumeFromDeepSleep();**

**}**

**void loop(void) {**

**if (digitalRead(SENOUT) == LOW && freezeMode == false) {**

**alert.flashWriteAlertLog("S" + timeRef.timeRead(false));**

**unsigned int WarningCycle = 0;**

**while (digitalRead(SENOUT) == LOW) {**

**alert.LED\_B\_Enable(100 \* cos(0.1 \* WarningCycle + 3.14) + 100);**

**oled.OLED\_DrawBMP(0, 0, 128, 64, Fire\_Warning);**

**alert.LED\_R\_Enable(100 \* cos(0.1 \* WarningCycle) + 200);**

**ShowFireWarning();**

**for (unsigned char i = 0; i < 30; ++i) {**

**alert.LED\_R\_Enable(1);**

**alert.LED\_G\_Enable(1);**

**alert.LED\_B\_Enable(1);**

**}**

**alert.BUZZER\_Enable(400 \* cos(0.1 \* WarningCycle) + 500);**

**++WarningCycle;**

**}**

**alert.flashWriteAlertLog("E" + timeRef.timeRead(false));**

**alert.ALERT\_Disable();**

**oled.OLED\_Clear();**

**Desktop.Main\_Desktop();**

**}**

**Show\_Charging\_info();**

**ProgramDownloadMode();**

**if (timeRef.allow == true) {**

**timeRef.getNetWorkTime();**

**timeRef.allow = false;**

**}**

**if (WIFI\_State == false) WebServer.WiFi\_Connect();**

**server.handleClient();**

**}**

**#include "OLED.h"**

**void OLED::I2C\_Init() {**

**SDA\_OUT;**

**SCL\_OUT;**

**I2C\_SDA\_H;**

**I2C\_SCL\_H;**

**}**

**void OLED::I2C\_Start() {**

**I2C\_SCL\_H;**

**I2C\_SDA\_H;**

**I2C\_SDA\_L;**

**I2C\_SCL\_L;**

**}**

**void OLED::I2C\_Stop() {**

**I2C\_SCL\_H;**

**I2C\_SDA\_L;**

**I2C\_SDA\_H;**

**}**

**void OLED::I2C\_Wait\_Ack() {**

**I2C\_SDA\_H;**

**I2C\_SCL\_H;**

**I2C\_SCL\_L;**

**}**

**void OLED::Write\_I2C\_Byte(unsigned char dat) {**

**unsigned char i;**

**for (i = 0; i < 8; i++) {**

**I2C\_SCL\_L;**

**if (dat & 0x80) {**

**I2C\_SDA\_H;**

**} else {**

**I2C\_SDA\_L;**

**}**

**dat <<= 1;**

**I2C\_SCL\_H;**

**}**

**I2C\_SCL\_L;**

**}**

**void OLED::OLED\_WR\_Byte(u8 dat, u8 mode) {**

**if (PreRendered == true && mode == OLED\_DATA) {**

**SET\_OLED\_GDDRAM\_CLONE(dat);**

**} else if (PreRendered == false) {**

**I2C\_Start();**

**Write\_I2C\_Byte(0x78);**

**I2C\_Wait\_Ack();**

**if (mode == OLED\_DATA) {**

**SET\_OLED\_GDDRAM\_CLONE(dat);**

**Write\_I2C\_Byte(0x40);**

**} else {**

**Write\_I2C\_Byte(0x00);**

**}**

**I2C\_Wait\_Ack();**

**Write\_I2C\_Byte(dat);**

**I2C\_Wait\_Ack();**

**I2C\_Stop();**

**}**

**}**

**void OLED::SET\_OLED\_GDDRAM\_CLONE(unsigned char BytesData) {**

**OLED\_GDDRAM\_CLONE[OLED\_Pos.y][OLED\_Pos.x] = BytesData;**

**}**

**void OLED::OLED\_ColorTurn(u8 i) {**

**if (!i)**

**OLED\_WR\_Byte(0xA6, OLED\_CMD);**

**else**

**OLED\_WR\_Byte(0xA7, OLED\_CMD);**

**}**

**void OLED::OLED\_DisplayTurn(u8 i) {**

**if (i == 0) {**

**OLED\_WR\_Byte(0xC8, OLED\_CMD);**

**OLED\_WR\_Byte(0xA1, OLED\_CMD);**

**} else {**

**OLED\_WR\_Byte(0xC0, OLED\_CMD);**

**OLED\_WR\_Byte(0xA0, OLED\_CMD);**

**}**

**}**

**void OLED::OLED\_Set\_Pos(u8 x, u8 y) {**

**OLED\_Pos = {x, y};**

**OLED\_WR\_Byte(0xb0 + y, OLED\_CMD);**

**OLED\_WR\_Byte(((x & 0xf0) >> 4) | 0x10, OLED\_CMD);**

**OLED\_WR\_Byte((x & 0x0f), OLED\_CMD);**

**}**

**void OLED::OLED\_Display\_On(void) {**

**OLED\_WR\_Byte(0X8D, OLED\_CMD);**

**OLED\_WR\_Byte(0X14, OLED\_CMD);**

**OLED\_WR\_Byte(0XAF, OLED\_CMD);**

**}**

**void OLED::OLED\_Display\_Off(void) {**

**OLED\_WR\_Byte(0X8D, OLED\_CMD);**

**OLED\_WR\_Byte(0X10, OLED\_CMD);**

**OLED\_WR\_Byte(0XAE, OLED\_CMD);**

**}**

**void OLED::OLED\_Clear(void) {**

**for (u8 i = 0; i < 8; ++i) {**

**OLED\_WR\_Byte(0xb0 + i, OLED\_CMD);**

**OLED\_WR\_Byte(0x00, OLED\_CMD);**

**OLED\_WR\_Byte(0x10, OLED\_CMD);**

**for (u8 n = 0; n < 128; ++n) OLED\_WR\_Byte(0, OLED\_DATA);**

**}**

**}**

**void OLED::OLED\_ShowChar(u8 x, u8 y, const u8 chr, u8 sizey) {**

**u8 c = 0, sizex = sizey**

**u16 i = 0, size1;**

**if (sizey == 8)**

**size1 = 5;**

**else**

**size1 = (sizey**

**c = chr - ' ';**

**OLED\_Set\_Pos(x, y);**

**for (i = 0; i < size1; i++) {**

**if (i % sizex == 0 && sizey != 8) OLED\_Set\_Pos(x, y++);**

**if (sizey == 8) {**

**temp = pgm\_read\_byte(&GSOS\_ASCII\_0805[c][i]);**

**OLED\_WR\_Byte(temp, OLED\_DATA);**

**} else if (sizey == 16) {**

**temp = pgm\_read\_byte(&asc2\_1608[c][i]);**

**OLED\_WR\_Byte(temp, OLED\_DATA);**

**} else if (sizey == 32) {**

**temp = pgm\_read\_byte(&asc2\_3216[c][i]);**

**OLED\_WR\_Byte(temp, OLED\_DATA);**

**} else if (sizey == 64) {**

**temp = pgm\_read\_byte(&asc2\_6432[c][i]);**

**OLED\_WR\_Byte(temp, OLED\_DATA);**

**} else if (sizey == 36) {**

**temp = pgm\_read\_byte(&asc2\_3618[c][i]);**

**OLED\_WR\_Byte(temp, OLED\_DATA);**

**} else if (sizey == 48) {**

**temp = pgm\_read\_byte(&asc2\_4824[c][i]);**

**OLED\_WR\_Byte(temp, OLED\_DATA);**

**} else if (sizey == 49) {**

**temp = pgm\_read\_byte(&asc2\_Digital\_2448[c][i]);**

**OLED\_WR\_Byte(temp, OLED\_DATA);**

**} else**

**return;**

**OLED\_Pos = {static\_cast<unsigned char>(static\_cast<unsigned short>(x) + i), y};**

**}**

**}**

**u32 OLED::oled\_pow(u8 m, u8 n) {**

**u32 result = 1;**

**while (n--) result \*= m;**

**return result;**

**}**

**void OLED::OLED\_ShowNum(u8 x, u8 y, u32 num, u8 len, u8 sizey) {**

**u8 t, temp, m = 0;**

**u8 enshow = 0;**

**if (sizey == 8) m = 2;**

**for (t = 0; t < len; t++) {**

**temp = (num**

**if (enshow == 0 && t < (len - 1)) {**

**if (temp == 0) {**

**OLED\_ShowChar(x + (sizey**

**continue;**

**} else**

**enshow = 1;**

**}**

**OLED\_ShowChar(x + (sizey**

**}**

**}**

**void OLED::OLED\_ShowString(u8 x, u8 y, const char \*chr, u8 sizey) {**

**u8 j = 0;**

**while (chr[j] != '\0') {**

**OLED\_ShowChar(x, y, chr[j++], sizey);**

**if (sizey == 8)**

**x += 6;**

**else**

**x += sizey**

**}**

**}**

**void OLED::OLED\_ShowChinese(u8 x, u8 y, const u8 no, u8 sizey) {**

**u16 i, size1 = (sizey**

**u8 temp;**

**for (i = 0; i < size1; i++) {**

**if (i % sizey == 0) OLED\_Set\_Pos(x, y++);**

**if (sizey == 32) {**

**temp = pgm\_read\_byte(&FireWarning\_32x32[no][i]);**

**OLED\_WR\_Byte(temp, OLED\_DATA);**

**} else if (sizey == 64) {**

**temp = pgm\_read\_byte(&FireWarning\_32x64[no][i]);**

**OLED\_WR\_Byte(temp, OLED\_DATA);**

**} else**

**return;**

**}**

**}**

**void OLED::OLED\_DrawBMP(u8 x, u8 y, u8 sizex, u8 sizey, const u8 BMP[]) {**

**u16 j = 0;**

**u8 i, m, temp;**

**sizey = sizey**

**for (i = 0; i < sizey; i++) {**

**OLED\_Set\_Pos(x, i + y);**

**for (m = 0; m < sizex; m++) {**

**temp = pgm\_read\_byte(&BMP[j++]);**

**OLED\_WR\_Byte(temp, OLED\_DATA);**

**}**

**}**

**}**

**void OLED::OLED\_Init(void) {**

**I2C\_Init();**

**OLED\_WR\_Byte(0xAE, OLED\_CMD);**

**OLED\_WR\_Byte(0x00, OLED\_CMD);**

**OLED\_WR\_Byte(0x10, OLED\_CMD);**

**OLED\_WR\_Byte(0x40, OLED\_CMD);**

**OLED\_WR\_Byte(0x81, OLED\_CMD);**

**OLED\_WR\_Byte(0xCF, OLED\_CMD);**

**OLED\_WR\_Byte(0xA1, OLED\_CMD);**

**OLED\_WR\_Byte(0xC8, OLED\_CMD);**

**OLED\_WR\_Byte(0xA6, OLED\_CMD);**

**OLED\_WR\_Byte(0xA8, OLED\_CMD);**

**OLED\_WR\_Byte(0x3f, OLED\_CMD);**

**OLED\_WR\_Byte(0xD3, OLED\_CMD);**

**OLED\_WR\_Byte(0x00, OLED\_CMD);**

**OLED\_WR\_Byte(0xd5, OLED\_CMD);**

**OLED\_WR\_Byte(0x80, OLED\_CMD);**

**OLED\_WR\_Byte(0xD9, OLED\_CMD);**

**OLED\_WR\_Byte(0xF1, OLED\_CMD);**

**OLED\_WR\_Byte(0xDA, OLED\_CMD);**

**OLED\_WR\_Byte(0x12, OLED\_CMD);**

**OLED\_WR\_Byte(0xDB, OLED\_CMD);**

**OLED\_WR\_Byte(0x40, OLED\_CMD);**

**OLED\_WR\_Byte(0x20, OLED\_CMD);**

**OLED\_WR\_Byte(0x02, OLED\_CMD);**

**OLED\_WR\_Byte(0x8D, OLED\_CMD);**

**OLED\_WR\_Byte(0x14, OLED\_CMD);**

**OLED\_WR\_Byte(0xA4, OLED\_CMD);**

**OLED\_WR\_Byte(0xA6, OLED\_CMD);**

**OLED\_Clear();**

**OLED\_WR\_Byte(0xAF, OLED\_CMD);**

**}**

**void OLED::OLED\_GDDRAM\_Refresh(u8 leftPixel, u8 topPixel, u8 rightPixel, u8 bottomPixel) {**

**unsigned char topPage = static\_cast<unsigned char>(topPixel >> 3);**

**unsigned char bottomPage = static\_cast<unsigned char>(bottomPixel >> 3);**

**for (u8 i = topPage; i < bottomPage; ++i) {**

**OLED\_WR\_Byte(0xb0 + i, OLED\_CMD);**

**OLED\_WR\_Byte(0x00, OLED\_CMD);**

**OLED\_WR\_Byte(0x10, OLED\_CMD);**

**for (u8 n = leftPixel; n < rightPixel; ++n) {**

**OLED\_WR\_Byte(OLED\_GDDRAM\_CLONE[i][n], OLED\_DATA);**

**}**

**}**

**}**

**void OLED::BeginBatchDraw() { PreRendered = true; }**

**void OLED::EndBatchDraw() { PreRendered = false; }**

**void OLED::putpixel(u8 xPixel, u8 yPixel, bool enable) {**

**if (xPixel > 127 || yPixel > 63) return;**

**unsigned char Byte\_data = static\_cast<unsigned char>(0x01 << (yPixel % 8));**

**unsigned char Page\_Pos = static\_cast<unsigned char>(yPixel >> 3);**

**if (enable == true) {**

**Byte\_data |= OLED\_GDDRAM\_CLONE[Page\_Pos][xPixel];**

**} else {**

**Byte\_data = OLED\_GDDRAM\_CLONE[Page\_Pos][xPixel] & (~Byte\_data);**

**}**

**if (Byte\_data == OLED\_GDDRAM\_CLONE[Page\_Pos][xPixel]) return;**

**OLED\_Set\_Pos(xPixel, Page\_Pos);**

**OLED\_WR\_Byte(static\_cast<unsigned char>(0xb0 + Page\_Pos), OLED\_CMD);**

**OLED\_WR\_Byte(static\_cast<unsigned char>(((xPixel & 0xf0) >> 4) | 0x10), OLED\_CMD);**

**OLED\_WR\_Byte(static\_cast<unsigned char>((xPixel & 0x0f) | 0x00), OLED\_CMD);**

**OLED\_WR\_Byte(Byte\_data, OLED\_DATA);**

**}**

**void OLED::line(u8 x1Pixel, u8 y1Pixel, u8 x2Pixel, u8 y2Pixel) {**

**unsigned char dx = abs(x2Pixel - x1Pixel);**

**unsigned char dy = abs(y2Pixel - y1Pixel);**

**short sx = (x1Pixel < x2Pixel) ? 1 : -1;**

**short sy = (y1Pixel < y2Pixel) ? 1 : -1;**

**short err = (dx > dy ? dx : -dy)**

**short e2;**

**while (true) {**

**putpixel(x1Pixel, y1Pixel);**

**if (x1Pixel == x2Pixel && y1Pixel == y2Pixel) break;**

**e2 = err;**

**if (e2 > -dx) {**

**err -= dy;**

**x1Pixel += sx;**

**}**

**if (e2 < dy) {**

**err += dx;**

**y1Pixel += sy;**

**}**

**}**

**}**

**void OLED::rectangle(u8 leftPixel, u8 topPixel, u8 rightPixel, u8 bottomPixel) {**

**line(leftPixel, topPixel, rightPixel, topPixel);**

**line(leftPixel, topPixel, leftPixel, bottomPixel);**

**line(rightPixel, topPixel, rightPixel, bottomPixel);**

**line(leftPixel, bottomPixel, rightPixel, bottomPixel);**

**}**

**void OLED::fillrectangle(u8 leftPixel, u8 topPixel, u8 rightPixel, u8 bottomPixel) {**

**line(leftPixel, topPixel, rightPixel, topPixel);**

**line(leftPixel, topPixel, leftPixel, bottomPixel);**

**line(rightPixel, topPixel, rightPixel, bottomPixel);**

**line(leftPixel, bottomPixel, rightPixel, bottomPixel);**

**for (unsigned char i = topPixel; i <= bottomPixel; i++) {**

**line(leftPixel, i, rightPixel, i);**

**}**

**}**

**void OLED::clearrectangle(u8 leftPixel, u8 topPixel, u8 rightPixel, u8 bottomPixel) {**

**for (unsigned char y = topPixel; y <= bottomPixel; ++y) {**

**for (unsigned char x = leftPixel; x <= rightPixel; ++x) {**

**putpixel(x, y, false);**

**}**

**}**

**}**

**vector<String> OLED::strsplit(String input, String separator) {**

**vector<String> vecStr;**

**while (input.indexOf(separator) != -1) {**

**unsigned int splitIndex = input.indexOf(separator);**

**String segment = input.substring(0, splitIndex);**

**vecStr.push\_back(segment);**

**input = input.substring(splitIndex + separator.length());**

**}**

**vecStr.push\_back(input);**

**return vecStr;**

**该函数的作用是将一个给定的字符串 "input" 根据另一个字符串 "separator" 进行分割，并将分割后的每一段字符串存储在一个 vector 容器中。**

**这个函数首先使用 while 循环来检查输入字符串中是否存在 "separator"，如果存在，则使用 "input.indexOf(separator)" 方法来找到 "separator"**

**第一次出现的位置，然后使用 "input.substring(0, splitIndex)" 方法来截取从 0 到 "splitIndex" 位置的字符串，将截取的字符串存储在 vector 容器 "vecStr"**

**中。接着，使用 "input = input.substring(splitIndex + 分割字符的长度);" 更新输入字符串，以便在下一次循环中继续查找 "separator"。**

**当循环结束后，将剩余的字符串 "input" 添加到 vector 容器 "vecStr" 中，并返回该容器。**

**\***

**}**

**void OLED::drawPrintBox() {**

**BeginBatchDraw();**

**sliderPos[0] = static\_cast<unsigned char>(8 \* Hight\_MaxNumChar \* (static\_cast<float>(First\_Line)**

**line(PrintRECT.right - 5, PrintRECT.top, PrintRECT.right - 5, PrintRECT.bottom);**

**clearrectangle(PrintRECT.right - 3, sliderPos[1], PrintRECT.right, sliderPos[1] + sliderHeight);**

**rectangle(PrintRECT.right - 3, sliderPos[0], PrintRECT.right - 1, sliderPos[0] + sliderHeight);**

**sliderPos[1] = sliderPos[0];**

**unsigned char y = PrintRECT.top;**

**for (unsigned int i = First\_Line; i < First\_Line + [](unsigned int PrintBox\_size, unsigned int Hight\_MaxNumChar) -> unsigned int {**

**if (PrintBox\_size < Hight\_MaxNumChar)**

**return PrintBox\_size;**

**else**

**return Hight\_MaxNumChar;**

**}(PrintBox.size(), Hight\_MaxNumChar);**

**++i) {**

**unsigned char x = PrintRECT.left;**

**for (auto &j : PrintBox[i]) {**

**OLED\_ShowChar(x, y, j, 8);**

**x += 5;**

**}**

**++y;**

**}**

**EndBatchDraw();**

**OLED\_GDDRAM\_Refresh(PrintRECT.left, PrintRECT.top, PrintRECT.right, PrintRECT.bottom);**

**unsigned char x = PrintRECT.left;**

**unsigned char y = PrintRECT.top;**

**unsigned char PrintWidth = static\_cast<unsigned char>(0.2 \* (PrintRECT.right - PrintRECT.left)) \* 5 - 5;**

**unsigned char PrintHight = static\_cast<unsigned char>(0.125 \* (PrintRECT.bottom - PrintRECT.top));**

**unsigned int Newline\_Pos = text.indexOf("\n", 0);**

**for (auto &i : text) {**

**unsigned int Text\_Pos = static\_cast<unsigned int>(std::distance(text.begin(), &i));**

**if (x >= PrintWidth || Text\_Pos == Newline\_Pos) {**

**Newline\_Pos = text.indexOf("\n", Newline\_Pos + 1);**

**x = PrintRECT.left;**

**++y;**

**if (y >= PrintHight) {**

**x = PrintRECT.left;**

**y = PrintRECT.top;**

**}**

**} else {**

**OLED\_ShowChar(x, y, i, 8);**

**x += 5;**

**}**

**}**

**\***

**}**

**void OLED::setTextBox(u8 leftPixel, u8 topPixel, u8 rightPixel, u8 bottomPixel) {**

**PrintRECT = {leftPixel, topPixel, rightPixel, bottomPixel};**

**First\_Line = 0;**

**if (PrintBox.empty() == false) {**

**PrintBox.clear();**

**PrintBox.shrink\_to\_fit();**

**}**

**}**

**void OLED::print(String text, bool autoScroll) {**

**BeginBatchDraw();**

**clearrectangle(PrintRECT.left, PrintRECT.top, PrintRECT.right, PrintRECT.bottom);**

**EndBatchDraw();**

**这里实现的是首先按照换行符换行, 换行符换行完成后检查是否有某行超出显示范围, 如果有则再对其换行.**

**这段代码主要的作用是将文本字符串根据换行符和一行最大字符数分割成若干行,**

**1.首先, 通过计算 PrintRECT 的宽度和高度，计算出最大字符数 Width\_MaxNumChar 和最大行数 Hight\_MaxNumChar。**

**2.然后，通过strsplit函数将文本按照'\n'分割成一个数组。**

**3.接下来，对于每一个分割后的字符串，使用 for 循环将其分割成若干段，每一段不会超过Width\_MaxNumChar个字符。**

**4.对于每一段，使用匿名函数计算出需要填充的空格数，并将这一段文本加上空格作为一个字符串插入到PrintBox中。**

**5.整个过程循环进行直到所有的文本都被分割完成。**

**'\n' 在代码中的作用是用来将文本分割成若干行, 这样就可以按照行来进行分割。**

**--------------------------------------------------------------------------------------------------------------------------------\***

**Width\_MaxNumChar = static\_cast<unsigned char>(0.2 \* (PrintRECT.right - PrintRECT.left)) - 1;**

**Hight\_MaxNumChar = static\_cast<unsigned char>(0.125 \* (PrintRECT.bottom - PrintRECT.top));**

**for (auto &i : strsplit(text, "\n")) {**

**for (unsigned int j = 0; j < i.length(); j += Width\_MaxNumChar) {**

**PrintBox.push\_back(i.substring(j, min(j + Width\_MaxNumChar, i.length())) + [](unsigned int length, unsigned char Width\_MaxNumChar) -> String {**

**String SpaceChar = "";**

**for (unsigned int k = length; k < Width\_MaxNumChar; ++k) SpaceChar += " ";**

**return SpaceChar;**

**}(i.length(), Width\_MaxNumChar));**

**}**

**}**

**sliderHeight = static\_cast<unsigned char>(8 \* Hight\_MaxNumChar \* (static\_cast<float>(Hight\_MaxNumChar)**

**if (autoScroll == true && PrintBox.size() > Hight\_MaxNumChar) First\_Line = PrintBox.size() - Hight\_MaxNumChar;**

**drawPrintBox();**

**}**

**void OLED::clearTextBox() {**

**PrintBox.clear();**

**PrintBox.shrink\_to\_fit();**

**First\_Line = 0;**

**};**

**vector<String> OLED::getPrintBox() { return PrintBox; }**

**void OLED::replacePrintBox(vector<String> newPrintBox) { PrintBox = newPrintBox; }**

**void OLED::moveScrollBar(bool direction) {**

**if (direction == true && (First\_Line + Hight\_MaxNumChar) < PrintBox.size()) {**

**++First\_Line;**

**} else if (direction == false && First\_Line > 0) {**

**--First\_Line;**

**}**

**drawPrintBox();**

**delay(1);**

**}**