```cpp
#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 sourceDirP
ath, String targetDirPath, bool mo
veMode = false) {
    LittleFS.begin();

    if (!LittleFS.exists(sourceDir
Path)) return;
    if (!LittleFS.exists(targetDirP
ath)) LittleFS.mkdir(targetDirPat
h);

    Dir sourceDir = LittleFS.openD
ir(sourceDirPath);

    while (sourceDir.next()) {

      String sourceFilePath = sou
rceDirPath + "
      String targetFilePath = targ
etDirPath + "

      if (sourceDir.isDirectory()
) {
        copyDir(sourceFilePath, t
argetFilePath, moveMode);
      } else {
        copyFile(sourceFilePath,
targetFilePath, moveMode);
      }
    }

    if (moveMode) deleteFolder(s
ourceDirPath);
  }
```

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(dirP
ath);

    while (dir.next()) {
      if (dir.isDirectory()) {
        foundFile += findFiles(dir
Path + dir.fileName() + "
      } else {
        String foundName = dir.file
Name();
        vector<String> foundNam
e_Split = oled.strsplit(foundName
, ".");

        if (targetName[0] == "*" &
& targetName[1] == "*") {
          foundFile += dirPath + fo
undName + "\n";

        } else if (targetName[0] ==
 "*" && targetName[1] != "*") {
          if (foundName_Split[1] =
= targetName[1]) foundFile += dir
Path + foundName + "\n";

        } else if (targetName[0] !=
"*" && targetName[1] != "*") {
          if (foundName == fileNam
e) foundFile += dirPath + foundNa
me + "\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 % 40
0 == 0 || (year % 4 == 0 && year % 1
00 != 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 net
WorkTimeStr) {
        year = static_cast<unsigne
d short>(netWorkTimeStr.substri
ng(0, 4).toInt());
        month = static_cast<unsigne
d char>(netWorkTimeStr.substrin
g(4, 6).toInt());
        day = static_cast<unsigned
char>(netWorkTimeStr.substring
(6, 8).toInt());
        hour = static_cast<unsigne
d char>(netWorkTimeStr.substrin
g(8, 10).toInt());
        minute = static_cast<unsign
ed char>(netWorkTimeStr.substri
ng(10, 12).toInt());
        second = static_cast<unsig
ned char>(netWorkTimeStr.subst
ring(12, 14).toInt());
      }

  } SystemTime;

  SystemTime sysTime;
  String netWorkTimeStr = "";

  bool allow = false;

  void begin() {

    TimeRefresh_ticker.attach(6
00, [this](void) -> void {

      if (freezeMode == true) Tim
eRefresh_ticker.detach();

      if (digitalRead(SENOUT) == H
IGH) allow = true;
    });

    System_time.attach(1, [this](v
oid) -> void { sysTime.updateTime(
); });
  }

  void getNetWorkTime() {
    uint8 httpCode = http.GET();
    if (httpCode > 0) {
      if (httpCode == HTTP_CODE_
OK) {
        netWorkTimeStr = http.get
String();

        StaticJsonDocument<200>
doc;
        deserializeJson(doc, netW
orkTimeStr);

        netWorkTimeStr = doc["sy
sTime1"].as<String>();

        sysTime.setSystemTime(ne
tWorkTimeStr);

      } else {
        Serial.printf("[HTTP GET F
ailed] ErrorCode: %s\n", http.err
orToString(httpCode).c_str());
      }
    } else {
      Serial.printf("[HTTP GET Fai
led] ErrorCode: %s\n", http.erro
rToString(httpCode).c_str());
    }
    http.end();
  }

  String format(unsigned char ti
meInt) {
    if (timeInt < 10) {
      return "0" + String(timeInt);
    } else {
      return String(timeInt);
    }
  }

  String timeRead(bool mode = tru
e) {
    if (mode == true) {
      return format(sysTime.hou
r) + ":" + format(sysTime.minute);
    } else {
      return String(sysTime.year
) + format(sysTime.month) + form
at(sysTime.day) + format(sysTim
e.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 = UINT
16_MIN;
  unsigned short Begin = UINT16_
MAX;
  unsigned short End = UINT16_MI
N;


  void AnimController() {

    if (Duration == UINT16_MIN) D
uration = Index.Duration;
    if (Begin == UINT16_MAX) Begi
n = Index.Begin;
    if (End == UINT16_MIN) End = I
ndex.End;


    unsigned long Sleep_ms = stat
ic_cast<unsigned long>(static_c
ast<float>(Duration)

    if (Index.name == loading_X16
_30F.name) {
      for (unsigned short i = Begi
n; 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 = Begi
n; 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 == loadingBack
ForthBar_60x8_60F.name) {
      for (unsigned short i = Begi
n; i < End; ++i) {
        oled.OLED_DrawBMP(Pos.x,
 Pos.y, Index.IMG_Width, Index.IMG
_Hight, LoadingBackForthBar_60
x8_60F[i]);
        delay(Sleep_ms);
      }
    }
    if (Index.name == loadingBar_
60x8_30F.name) {
      for (unsigned short i = Begi
n; 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 = Begi
n; 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 loadingBackForthB
ar_60x8_60F = {2, 500, 0, 60, 60, 8
};
  ANIM_INDEX loadingBar_60x8_3
0F = {3, 300, 0, 30, 60, 8};
  ANIM_INDEX loadingBar_60x8_6
0F = {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 i
ndex) {
    Index = index;
    AnimController();
  }
} anim;


void ShowFireWarning() {
  oled.OLED_DrawBMP(0, 0, 32, 64,
FireWarning_32x64[0]);
  oled.OLED_DrawBMP(32, 0, 32, 6
4, FireWarning_32x64[1]);
  oled.OLED_DrawBMP(64, 0, 32, 6
4, FireWarning_32x64[2]);
  oled.OLED_DrawBMP(96, 0, 32, 6
4, FireWarning_32x64[3]);
}


class Desktop {
 private:


  typedef struct StatusBars_Ra
nked {
    unsigned char unit = 16;


    bool Register_State[8] = {fal
se, false, false, false, false, fals
e, 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 ProgramDownl
oad = 2;
    unsigned char Disconnected
= 3;
    unsigned char Battery = 4;
    unsigned char CMDCP = 5;


  } StatusBars_Ranked;
  StatusBars_Ranked SBR;



  void Icon_Register(unsigned c
har name, bool mode) {
    if (mode == false && SBR.Regis
ter_State[name] == true) {
      SBR.Register_State[name] =
false;


      for (auto& i : SBR.StatusBar
s_Pos)
        if (i > SBR.StatusBars_Pos
[name]) i -= SBR.unit;


      for (unsigned char i = SBR.S
tatusBars_Pos[name]; i <= 112; i +
= SBR.unit) oled.OLED_DrawBMP(i,
6, 16, 16, StatusBars[SBR.Clear_I
con]);


      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.StatusBar
s_Pos, 8) + SBR.unit;
    }


    if (mode == true) {
      oled.OLED_DrawBMP(SBR.Sta
tusBars_Pos[name], 6, 16, 16, Stat
usBars[name]);
    }
  }


  void StatusBars_Render() {


    if (Charging_State == true) {
      Icon_Register(SBR.Chargin
g, true);
    } else {
      Icon_Register(SBR.Chargin
g, false);
    }


    if (WIFI_State == true) {
      Icon_Register(SBR.WIFI, tru
e);
    } else {
      Icon_Register(SBR.WIFI, fal
se);
    }


    if (digitalRead(0) == LOW) {
      Icon_Register(SBR.Program
Download, true);
    } else {
      Icon_Register(SBR.Program
Download, false);
    }


    if (WIFI_State == false) {
      Icon_Register(SBR.Disconn
ected, true);
    } else {
      Icon_Register(SBR.Disconn
ected, false);
    }


    if (Charging_State == false) {
      Icon_Register(SBR.Battery,
true);
    } else {
      Icon_Register(SBR.Battery,
false);
    }


    if (CMDCP_State == true) {
      Icon_Register(SBR.CMDCP, t
rue);
    } else {
      Icon_Register(SBR.CMDCP, f
alse);
    }
  }


 public:
  void begin() {
    Desktop_ticker.attach_ms(50
0, [this](void) -> void {


      if (freezeMode == true) Des
ktop_ticker.detach();


      if (digitalRead(SENOUT) == H
IGH && 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.run
Animation(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) == H
IGH && Charging_State == true) {
    Charging_State = false;
  }
}


void ProgramDownloadMode() {


  if (digitalRead(Decoder_C) == L
OW && 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.loadi
ng_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(De
coder_C);
    int decoderB = digitalRead(De
coder_B);
    int decoderA = digitalRead(De
coder_A);


    String decodedWith = "\nDeco
ded_with:";
    switch (decoderC << 2 | decod
erB << 1 | decoderA) {
      case 1:
        decodedWith += "Buzzer_
Enable";
        break;
      case 2:
        decodedWith += "RedLED_
Enable";
        break;
      case 3:
        decodedWith += "GreenLE
D_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)) + "\nTX
D=" + String(digitalRead(TXD)) +
"\nRXD=" + String(digitalRead(RX
D)) +
          "\nSCL=" + String(di
gitalRead(SCL)) + "\nSDA=" + Stri
ng(digitalRead(SDA)) + "\nCHRG=
" + String(digitalRead(CHRG)) +
          "\nLOWPOWER=" + Str
ing(digitalRead(LOWPOWER)) + "\n
SENOUT=" + String(digitalRead(SE
NOUT)) + "\nDecoder_C=" + Strin
g(decoderC) +
          "\nDecoder_B=" + St
ring(decoderB) + "\nDecoder_A
=" + String(decoderA) + decoded
With;
    return GPIO_State;
  }


  String getSysModeAndStatus()
{
    return "Charging_State=" + S
tring(Charging_State) + "\nWIFI
_State=" + String(WIFI_State) + "
\nDeveloper_Mode=" + String(De
veloper_Mode) +
        "\nallowResponse=" + Stri
ng(allowResponse) + "\nallowDo
wnloadMode=" + String(allowDown
loadMode) + "\nfreezeMode=" +
String(freezeMode) +
        "\ndiskMode=" + String(dis
kMode);
  }


  void removeSleepFile() {


    if (LittleFS.exists("
  }


  void resumeFromDeepSleep() {
    LittleFS.begin();
    String SleepFile = "";


    if (LittleFS.exists("

    SleepFile = FFileS.readFile("
", "


    for (auto& i : oled.strsplit(Sl
eepFile, "\n")) {
      vector<String> GPIO_Stat
us = oled.strsplit(i, "=");


      if (GPIO_Status[0] == "Dec
oder_C") digitalWrite(Decoder_C,
 GPIO_Status[1].toInt());
      if (GPIO_Status[0] == "Dec
oder_B") digitalWrite(Decoder_B,
 GPIO_Status[1].toInt());
      if (GPIO_Status[0] == "Dec
oder_A") digitalWrite(Decoder_A,
 GPIO_Status[1].toInt());
    }
    FFileS.removeFile("", "
  }


   if (LittleFS.exists("

    SleepFile = FFileS.readFile("
", "


    for (auto& i : oled.strsplit(Sl
eepFile, "\n")) {
      vector<String> SysModeA
ndStatus = oled.strsplit(i, "=");



      if (SysModeAndStatus[0] =
= "Developer_Mode") Developer
_Mode = (SysModeAndStatus[1].t
oInt() != 0);
      if (SysModeAndStatus[0] =
= "allowResponse") allowRespon
se = (SysModeAndStatus[1].toInt
() != 0);
      if (SysModeAndStatus[0] =
= "allowDownloadMode") allowDow
```

```
nloadMode = (SysModeAndStatus[
1].toInt() != 0);
        if (SysModeAndStatus[0] =
= "freezeMode") freezeMode = (
SysModeAndStatus[1].toInt() != 0
);
        if (SysModeAndStatus[0] =
= "diskMode") diskMode = (SysMod
eAndStatus[1].toInt() != 0);

      }
      FFileS.removeFile("", "
    }

    diskMode = false;
  }


  void Sys_freezeMode(bool Ena
ble = true) {
    if (Enable == true) {
      freezeMode = true;

      oled.OLED_Display_Off();


      digitalWrite(Decoder_C, HIG
H);
      digitalWrite(Decoder_B, HIG
H);
      digitalWrite(Decoder_A, LOW
);

      WiFi.setSleepMode(WIFI_MOD
EM_SLEEP);
    } else if (Enable == false) {
      WiFi.setSleepMode(WIFI_NON
E_SLEEP);

      freezeMode = false;


      digitalWrite(Decoder_C, HIG
H);
      digitalWrite(Decoder_B, HIG
H);
      digitalWrite(Decoder_A, HIG
H);
```

```


      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 tim
e_us = 0) {

    CMDCP_State = false;
    Developer_Mode = false;

    diskMode = true;
    LittleFS.begin();


    File GPIO_StatusFile = LittleFS
.open("
    GPIO_StatusFile.print(GPIO_R
ead());
    GPIO_StatusFile.close();



    File SysModeAndStatusFile = L
ittleFS.open("
    SysModeAndStatusFile.print(
getSysModeAndStatus());
    SysModeAndStatusFile.close(
);
```

```


    File PrintBoxFile = LittleFS.op
en("
    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 pa
ssword:");
        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 == Pa
ssWord_Temp2 && PassWord_Tem
p1 != "") {

          File dataFile = LittleFS.op
en("
          dataFile.print(String(sh
a1(PassWord_Temp1)));
          dataFile.close();


          CMDCP_Response("Acce
pted");
          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 = Str
ing(sha1(CMD));
            CMD = "";
            break;
          }
          delay(500);
        }


        if (PassWord == PassWord
_Temp1) {
          CMDCP_Response("Pass
ed");
          server.send(200, "text
          allowResponse = true;

          return true;
        }
      }
    }
```

```
    CMDCP_State = false;
    allowResponse = true;
    return false;
  }

  void CMDCP_Response(String R
esponse) {
    CMDCP_Online_Response = "";
    if (Response != "") {
      CMDCP_Online_Response = R
esponse;
      oled.print(Response);
      Serial.println(Response);
    }
  }


  void saveCmdHistory(String CM
D, String clientIP) {
    LittleFS.begin();

    File cmdHistory = LittleFS.ope
n("
    cmdHistory.print(clientIP + "-
" + timeRef.timeRead(false) + "-
" + CMD + "\n");
    cmdHistory.close();
  }

 public:
  bool allow = false;

  void begin() {
    oled.setTextBox(0, 0, 128, 48);


  }

  String CMDControlPanelOnlineP
ortal(String CMDCP_Online_Mess
age) {

    if (CMDCP_Online_Message ==
"") return "";

    CMD = CMDCP_Online_Message;
```

```
    String clientLogingIP = serve
r.client().remoteIP().toString();


    bool LockerState = false;
    for (auto& i : clientLogedIP) {

      if (i == clientLogingIP) Lock
erState = true;
    }


    if (LockerState == false && (C
MD == "CMD" || CMD == "cmd" || CMD =
= "login")) {
      CMDCP_State = true;

      LockerState = PassLocker(
);


      if (LockerState == true) {

        if (clientLogedIP.empty()
== true) {
          Developer_Mode = true;

          oled.setTextBox(0, 0, 12
8, 64);
        }

        clientLogedIP.push_back(
clientLogingIP);


        LittleFS.begin();

        File cmdLoggedInfo = Littl
eFS.open("
        cmdLoggedInfo.print(clie
ntLogingIP + "-
" + timeRef.timeRead(false) + "-
login\n");
        cmdLoggedInfo.close();
      }
    }


    if (LockerState == true) {
      saveCmdHistory(CMD, client
LogingIP);
      commandIndexer();

    }

    allow = false;
    return CMDCP_Online_Respon
se;
  }

  void CMDControlPanelSerialPor
tal() {
    SerialReceived();


    if (LockerState == false && Se
rial.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) comm
andIndexer();
    allow = false;
  }

  void SerialReceived() {
    CMD = "";

    while (Serial.available()) CMD
+= static_cast<char>(Serial.rea
d());
  }

  void commandIndexer() {
    oled.print("> " + CMD);
    Serial.println("> " + CMD);
```

```


    vector<String> CMD_Index = o
led.strsplit(CMD, " ");



    if (CMD_Index[0] == "help") {
      CMDCP_Response(CMDCP_HE
LP);
    }


    if (CMD_Index[0] == "pwd") {
      CMDCP_Response(FFileS.get
WorkDirectory());
    }


    if (CMD_Index[0] == "ls") {
      String listDirectory = FFileS.
listDirectoryContents();
      CMDCP_Response(listDirect
ory);
    }



    (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(C
MD_Index[1].length() - 1) != '
        FFileS.changeDirectory(C
MD_Index[1]);
      }
      CMDCP_Response("");
    }
```

```
    if (CMD_Index[0] == "cat") {
      String File_Info = FFileS.rea
dFile(CMD_Index[1]);
      CMDCP_Response(File_Info
);
    }


    if (CMD_Index[0] == "touch")
{
      FFileS.createFile(CMD_Inde
x[1]);
      CMDCP_Response("");
    }


    if (CMD_Index[0] == "mkdir") {
      FFileS.makeDirector(CMD_In
dex[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_Inde
x[1], CMD_Index[3]);
        CMDCP_Response("");
      } else if (CMD_Index[2] == ">
>") {
        FFileS.fileAppend(CMD_Ind
ex[1], CMD_Index[3]);
        CMDCP_Response("");
      } else {
        CMDCP_Response(CMD_Ind
ex[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(CM
D_Index[2]);
      } else {
        FFileS.removeFile(CMD_In
dex[1]);
      }
      CMDCP_Response("");
    }


    (复制一个文件或目录
(Copy file))cp [-
options] [sourcePath] [targetPat
h];
    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_Inde
x[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] [targetPat
h];
    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_Inde
x[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(CM
D_Index[1].length() - 1) != '

      String foundFile = FFileS.fin
dFiles(CMD_Index[1], CMD_Index[
2]);
      CMDCP_Response(foundFile
);
    }


    if (CMD_Index[0] == "osinfo")
{
      CMDCP_Response(GSG3_Os_
Info);

      oled.OLED_DrawBMP(0, 0, 12
8, 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" || CM
D_Index[2] == "true" || CMD_Index
[2] == "enable") {
      if (CMD_Index[1] == "r") {
        allowDownloadMode = fal
se;
        alert.LED_R_Enable(0, t
rue);
      } else if (CMD_Index[1] ==
"g") {
        allowDownloadMode = fal
se;
        alert.LED_G_Enable(0, t
rue);
      } else if (CMD_Index[1] ==
"b") {
```

```
        alert.LED_B_Enable(0, t
rue);
      }
    } else if (CMD_Index[2] == "
0" || CMD_Index[2] == "false" || CM
D_Index[2] == "disable") {
      if (CMD_Index[1] == "r") {
        alert.LED_R_Enable(0, f
alse);
        allowDownloadMode = tru
e;
      } else if (CMD_Index[1] ==
"g") {
        alert.LED_G_Enable(0, f
alse);
        allowDownloadMode = tru
e;
      } else if (CMD_Index[1] ==
"b") {
        alert.LED_B_Enable(0, f
alse);
      }
    }
    CMDCP_Response("");
  }


  (打开蜂鸣器)buzz [state]
  state = 1
  state = 0
  *
  if (CMD_Index[0] == "buzz") {
    if (CMD_Index[1] == "1" || CM
D_Index[1] == "true" || CMD_Index
[1] == "enable") {
      alert.BUZZER_Enable(0, tr
ue);
    } else if (CMD_Index[1] == "
0" || CMD_Index[1] == "false" || CM
D_Index[1] == "disable") {
      alert.BUZZER_Enable(0, fa
lse);
    }
    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" || CM
D_Index[1] == "true" || CMD_Index
[1] == "enable") {
        SysSleep.Sys_freezeMod
e(true);
      } else if (CMD_Index[1] == "
0" || CMD_Index[1] == "false" || CM
D_Index[1] == "disable") {
        SysSleep.Sys_freezeMod
e(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_In
dex[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.r
eadFile("", "
      } else if (CMD_Index[1] == "
-c") {
        FFileS.removeFile("", "
        CMDCP_Response("");
      } else {
        CMDCP_Response(FFileS.r
eadFile("", "
      }
    }


    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.r
eadFile("", "
      }
    }


    (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 (Ye
ar 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.setSystem
Time(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.g
etFlash_info();
      CMDCP_Response(Flash_Inf
o);
    }


    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] == "powerof
f") {
      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](Stri
ng clientIP) -> void {

      clientLogedIP.erase(remov
e(clientLogedIP.begin(), clientLo
gedIP.end(), clientIP), clientLoge
dIP.end());


      File cmdLoggedInfo = LittleF
S.open("
      cmdLoggedInfo.print(clientI
P + "-
" + timeRef.timeRead(false) + "-
logout\n");
      cmdLoggedInfo.close();

    };
    if (CMD_Index[0] == "logout")
{
      LittleFS.begin();
      if (CMD_Index[1] == "-k") {
```

```
        if (CMD_Index[2] == "othe
r") {
          String clientLogoutIP = s
erver.client().remoteIP().toStrin
g();
          vector<String> allClientI
P = clientLogedIP;

          for (auto& i : allClientIP)
            if (i != clientLogoutIP)
clientLogout(i);

        } else if (CMD_Index[2] ==
"all") {
          vector<String> allClientI
P = clientLogedIP;
          for (auto& i : allClientIP)
clientLogout(i);
        } else {
          clientLogout(CMD_Index
[2]);
        }
      } else {
        String clientLogoutIP = se
rver.client().remoteIP().toString
();
        clientLogout(clientLogout
IP);
      }

      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_Messa
ge = server.arg("message");

      String CMDCP_Send_Messag
e = CMDCP.CMDControlPanelOnline
Portal(CMDCP_Online_Message);


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


  void WiFi_Connect() {

    if (timeRef.timeRead() == "00:0
0")
      anim.setAnimation(67, 6);
    else
      anim.setAnimation(112, 6);


    if (WiFi.status() != WL_CONNEC
TED) {
      WIFI_State = false;


      vector<String> SSID_PASSW
D = oled.strsplit(FFileS.readFile(
"", "


      WiFi.begin(SSID_PASSWD[0],
SSID_PASSWD[1]);


      for (unsigned char i = 0; i < 1
00; ++i) {
```

```
        if (WiFi.status() == WL_CON
NECTED) {
          Serial.print("IP Address
: ");
          Serial.print(WiFi.localIP(
));
          Serial.println(":" + Strin
g(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_PULL
UP);
  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, 6
4, 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) == HIG
H && WiFi.status() != WL_CONNECTE
D) WIFI_State = false;
  });


  timeRef.begin();
  Desktop.begin();
  CMDCP.begin();
  WebServer.beginServer();


  SysSleep.resumeFromDeepSlee
p();
}


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 * c
os(0.1 * WarningCycle + 3.14) + 10
0);
      oled.OLED_DrawBMP(0, 0, 12
8, 64, Fire_Warning);
```

```
      alert.LED_R_Enable(100 * c
os(0.1 * WarningCycle) + 200);

      ShowFireWarning();


      for (unsigned char i = 0; i < 3
0; ++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) WebSer
ver.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(unsign
ed 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 && mod
e == OLED_DATA) {
    SET_OLED_GDDRAM_CLONE(da
t);
  } else if (PreRendered == false
) {
    I2C_Start();
    Write_I2C_Byte(0x78);
    I2C_Wait_Ack();
    if (mode == OLED_DATA) {
      SET_OLED_GDDRAM_CLONE(d
at);
      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_CL
ONE(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_CM
D);
  else
    OLED_WR_Byte(0xA7, OLED_CM
D);
}


void OLED::OLED_DisplayTurn(u8 i
) {
  if (i == 0) {
```

```
    OLED_WR_Byte(0xC8, OLED_CM
D);
    OLED_WR_Byte(0xA1, OLED_CM
D);
  } else {
    OLED_WR_Byte(0xC0, OLED_CM
D);
    OLED_WR_Byte(0xA0, OLED_CM
D);
  }
}


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_CM
D);
    OLED_WR_Byte(0x10, OLED_CM
D);
    for (u8 n = 0; n < 128; ++n) OLE
D_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(&GSO
5_ASCII_0805[c][i]);
      OLED_WR_Byte(temp, OLED_D
ATA);
    } else if (sizey == 16) {
      temp = pgm_read_byte(&asc
2_1608[c][i]);
      OLED_WR_Byte(temp, OLED_D
ATA);
    } else if (sizey == 32) {
      temp = pgm_read_byte(&asc
2_3216[c][i]);
      OLED_WR_Byte(temp, OLED_D
ATA);
    } else if (sizey == 64) {
      temp = pgm_read_byte(&asc
2_6432[c][i]);
```

```
      OLED_WR_Byte(temp, OLED_D
ATA);
    } else if (sizey == 36) {
      temp = pgm_read_byte(&asc
2_3618[c][i]);
      OLED_WR_Byte(temp, OLED_D
ATA);
    } else if (sizey == 48) {
      temp = pgm_read_byte(&asc
2_4824[c][i]);
      OLED_WR_Byte(temp, OLED_D
ATA);
    } else if (sizey == 49) {
      temp = pgm_read_byte(&asc
2_Digital_2448[c][i]);
      OLED_WR_Byte(temp, OLED_D
ATA);
    } else
      return;


    OLED_Pos = {static_cast<unsi
gned char>(static_cast<unsigne
d 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, u
8 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 + (size
y
        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_Po
s(x, y++);
    if (sizey == 32) {
      temp = pgm_read_byte(&Fir
eWarning_32x32[no][i]);
      OLED_WR_Byte(temp, OLED_D
ATA);
    } else if (sizey == 64) {
      temp = pgm_read_byte(&Fir
eWarning_32x64[no][i]);
      OLED_WR_Byte(temp, OLED_D
ATA);
    } else
      return;
  }
}
```

```
void OLED::OLED_DrawBMP(u8 x, u
8 y, u8 sizex, u8 sizey, const u8 B
MP[]) {
  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_D
ATA);
    }
  }
}



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 right
Pixel, u8 bottomPixel) {

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

```cpp
    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。

```cpp
    *
}

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。

```cpp
    ----------------------------------------------------------------------------------------------------------------------------*

    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.len
gth(); j += Width_MaxNumChar) {
        PrintBox.push_back(i.subst
ring(j, min(j + Width_MaxNumChar,
 i.length())) + [](unsigned int leng
th, unsigned char Width_MaxNumC
har) -> String {

          String SpaceChar = "";
          for (unsigned int k = lengt
h; k < Width_MaxNumChar; ++k) Spa
ceChar += " ";
          return SpaceChar;
        }(i.length(), Width_MaxNumC
har));
      }
   }



   sliderHeight = static_cast<unsi
gned char>(8 * Hight_MaxNumCha
r * (static_cast<float>(Hight_Ma
xNumChar)



   if (autoScroll == true && PrintB
ox.size() > Hight_MaxNumChar) Fi
rst_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(vect
or<String> newPrintBox) { PrintB
ox = newPrintBox; }
```

```
void OLED::moveScrollBar(bool di
rection) {
   if (direction == true && (First_L
ine + Hight_MaxNumChar) < PrintB
ox.size()) {
      ++First_Line;
   } else if (direction == false && F
irst_Line > 0) {
      --First_Line;
   }
   drawPrintBox();
   delay(1);
}
```