嵌入式開發中的程序架構

STM32嵌入式開發 2023-01-09 17:00 發表於山東

前言

在嵌入式軟體開發，包括單片機開發中，軟體架構對於開發人員是一個必須認真考慮的問題。

軟體架構對於系統整體的穩定性和可靠性是非常重要的，一個合適的軟體架構不僅結構清晰，並且便於開發。

我相信在嵌入式或單片機軟體開發的初期大多數開發者採用的都是簡單的前後台順序執行架構（我就是這樣的）。 在嵌入式軟體開發中，程序架構主要分為三種，本篇文章將對這三種程序架構做出詳解。

軟體架構存在的意義

可以說一個好的程序架構，是一個有經驗的工程師和一個初學者的分水嶺。 軟體架構對於開發人員是友好的，你希望先執行什麼任務後執行什麼任務，或者這一個時間點執行什麼任務下一個執行什麼任務，又或者什麼事件會同步到某個任務等等，在不同的軟體架構下，解決上述問題的具體方法都是有所區別的。

軟體架構對開發者最大的説明是：幫助開發者掌控整個工程的框架，當你熟練使用其中某一個程序架構后，對於系統中出現的bug你一定能夠快速的定位並解決。 當然，我建議要根據需要選擇合適的軟體架構進行開發，具體原因在文章後面會進行介紹。

深入介紹三種不同的程序架構

三種常用的軟體架構有：順序執行的前後台系統、時間片輪詢系統和多任務作業系統。 為了讓大家有一個更為清晰的認識，我分別用三種軟體架構對一個實例進行介紹說明。 這個實例如下：它有4個任務，這4個任務為按鍵掃描、聲光報警、顯示幕刷新和超聲波測距。 這個實例的具體功能是通過按鍵設置測量距離的閾值，當測距距離低於設置的閾值時，觸發聲光報警並且將測量距離實時顯示在顯示幕上（這個應用是汽車倒車雷達的具體體現）。

1 順序執行的前後台系統

在順序執行的前後台系統中，我會把鍵盤掃描用查詢的方式放在while（1）中，而顯示幕刷新和超聲波測距使用中斷，在中斷服務函數中獲取測量距離後進行顯示，在主函數的迴圈中進行按鍵的檢測，聲光處理也放在主迴圈中。 這樣整個程式就以變數標誌的同步方式在主迴圈和後台中斷中執行，對應的程式代碼如圖所示：

```c
int main(void)
{
    key_init();    //系统初始化
    alarm_init(); //声光报警初始化
    lcd_init();    //显示屏初始化
    ultrasonic_init();//超声波初始化
    timer_init(); //定时器初始化

    while (1)
    {
        ...
        key_scan(); //按键扫描，设置测距阈值

        if (distance <= key_distance) //如果测量距离小于设置的阈值
        {
            alarm_open();  //启动声光报警
        }
        else
        {
            alarm_close(); //关闭声光报警
        }
        ...
    }

    return 0;
}
```

上面代碼是順序執行前後台系統的主函數。

```c
void InterruptTimer0() interrupt 1 //定时器中断服务函数
{
    ...
    times++;
    if (times == 30) //到达30ms定时
    {
        lcd_display(); //显示屏刷新
    }
    else if (times == 100)
    {
        times = 0;
        ultrasonic_measure(); //超声波测距
    }
}
```

如上代码，顺序执行前后台系统的中断服务函数。

这种架构的优点是使用简单易于理解，而缺点是每个任务所占的CPU时间过长的话，会导致程序的实时性能差，就比如按键的检测等。

2 时间片轮询系统和多任务操作系统

时间片轮询法实际上通常出现在操作系统中，也就是说他属于操作系统，但在这里所说的是基于前后台系统的时间片轮询。

时间片轮询法的实质其实就是选出一个定时器，每进一次定时中断对计数值进行自加，在主循环中根据这个计数值执行任务，这个计数值也就是任务轮询的时间片。

在这个实例中，如果采用时间片轮询系统的话，首先选用主控芯片的任一定时器，定时器定时时间周期由我们决定，为了保证实时性和运行效率，这个值通常取10ms、30ms、50ms等，我会将按键扫描轮值值设置为20ms，因为按键抖动的时长一般为20ms，这样处理既达到了消抖的目的，又不会漏掉按键的检测。

而显示屏的刷新设置为30ms，如果你觉得刷新反应慢了也可以修改这一轮询值得到改善；而超声波测距的轮询值设置为100ms，即每隔100模式触发测距一次，这个测距频率已经能够满足大多数的情况了。

程序代码如下：

```
int main(void)
{
    key_init();    //系统初始化
    alarm_init(); //声光报警初始化
    lcd_init();    //显示屏初始化
    ultrasonic_init();//超声波初始化
    timer_init(); //定时器初始化

    while (1)
    {
        if (times == 20)
        {
            key_scan(); //按键扫描
        }
        else if (times == 30)
        {
            lcd_display(); //显示屏刷新
        }
        else if(times == 100)
        {
            ultrasonic_measure(); //超声波测距
        }
    }

    return 0;
}
```

如上代码，时间片轮询系统的主函数。

```
void InterruptTimer0() interrupt 1 //定时器中断服务函数
{
    ...
    times++;
    if (times == 101) //定时到达101ms清零,之所以在101ms清零
                      //而不是100ms的原因是保证主循环中的超声波测距成功触发
    {
        times = 0;
    }

}
```

如上代码，时间片轮询系统的定时器中断函数。

可以看出时间片轮询法相比顺序执行还是有很大优势的，既有顺序执行法的优点，也有操作系统的部分优点。

3 多任务操作系统

操作系统的本身是一个比较复杂的东西，任务的管理和调度实现的底层是很复杂和困难的。

但是，我们一般都是把操作系统本身作为一个工具一个平台，我们的目的是使用它的功能而不是开发一个操作系统。

我使用过ucos和freertos小型的实时操作系统，也使用过Linux大型的操作系统，有了操作系统，不管是对于程序的稳定性和开发的效率都会好很多。

我们在使用操作系统的时候更多的需要去学习和理解它的一些调度和通信的方式。

实际上真正能使用操作系统的人并不多，反而是跑裸机的占大多数，这也和产品的具体要求有关，很多简单的系统只需要裸机即可满足。

在这里本我不过多的介绍操作系统本身，因为操作系统确实挺复杂的，相关文章：使用STM32CubeMx工具，写FreeRTOS的demo程序。下面图例中的代码是在freertos中创建按键控制LED亮灭的程序结构，大家可以对比一下：

```c
#include "mytask.h"
#include "init.h"

TaskHandle_t LED_TaskHandler;
TaskHandle_t KEY_TaskHandler;

int main()
{

    //系统初始化
    SYS_Init();

    xTaskCreate(LED_Task, "LED_Task", 256, (void*)0, 1, &LED_TaskHandler);

    //创建任务KEY_Task，名称 KEY_Task，分配内存 256Word，传入参数 空，优先级 2，任务句柄 KEY_TaskHandler
    xTaskCreate(KEY_Task, "KEY_Task", 256, NULL, 2, &KEY_TaskHandler);

    //开始任务调度
    vTaskStartScheduler();

    while (1); //开启任务调度之后程序永远不会执行到这一步
    return 0;
}
```

如上，freertos多任务系统中主函数。

```c
#include "mytask.h"
#include "init.h"

//LED任务：LED1闪烁，间隔 300ms
void LED_Task(void *pvParameters)
{
    while(1)                            //操作系统都有的无限循环
    {
        LED_Toggle(0);
        vTaskDelay(300);                //已经将系统心跳设置为1ms，因此300为300ms
    }
}

//KEY任务：按下KEY0，LED0闪烁一次
void KEY_Task(void *pvParameters)
{
    u8 key=1;
    while(1)
    {
        key = Get_KEY0();               //KEY0，低电平有效，按下为0
        vTaskDelay(20);                 //消抖
        key = Get_KEY0();

        if(key == 0)
        {
            LED_Toggle(1);
            vTaskDelay(500);            //保持当前状态0.5s
        }
    }
}
```

如上，freertos多任务操作系统中的任务回调函数。

如何选择合适的软件架构

我使用过多种不同MCU做项目开发，例如：STM32、STC51、新唐等，也接触过复杂的设计需求，例如：车载智能系统和智能家居，跑过操作系统ucos、freertos和Linux等等，在回到裸机开发时，就会不然而然的去思考完整系统的软件架构的设计问题，相信在读者中开发裸机的也占大多数。

我认为没有最好的软件架构(程序架构)，而是只有最合适的。因为在不同的应用场景中适合采用不同的程序设计，而单纯的去比较哪种程序架构是最好的没有什么实际的意义。

那接下来我们来对具体的应用场景进行分析：

- 在一些逻辑清晰功能单一的系统中就很适合选择顺序执行的前后台架构，这个软件架构往往能够满足我们大部分的需求，比如电饭煲、电磁炉和声控灯泡等；
- 在一些资源缺乏的单片机并且对系统可靠性要求较高的情况下非常适合，因为这种方法的系统耗费比较小，只是牺牲了一个定时器而已，但是选择此种程序架构需要我们对时间片进行深思熟虑的划分；
- 最后，在一些功能复杂，逻辑控制较为困难的系统中就适合选择多任务操作系统，比如视频监控系统、无人机等等应用场景。

作为嵌入式软件工程师，掌握这三种软件架构是非常有必要的，它们可以让我们在设计程序时拥有更多的选择和思考，而每一种不同的程序架构都具备它自己的优势与不足，这需要我们去用心实践方可体会到它的奥妙。