### 基于 Linux 的智能灯具控制器软件设计

##### 摘 要：

针对如港口、厂房等对灯光照明有要求的场所，如何在保证经济效益的前提下，节能减排成为了一个可以切入的点，针对此，设计了基于嵌入式 Linux 的智能灯具控制器。尝试通过智能控制的技术，保证光照度满足作业要求的情况下，可以实时的动态调整灯具的亮度等级，以期达到节能的目的。系统采用单元化的设计准则，通过 ARM9 系列的工业级 ARM 处理器作为智能灯具控制器的核心处理器，外围扩展单元模块：LoRa 无线模块和末端的灯具控制器通讯，完成和智能灯具的双向通讯；4G 无线单元网络模块，完成和云平台双向的网络通讯；触摸显示屏单元模块，实现友好的人机访问。考虑到外围多模块，数据交互复杂的特点，采用基于嵌入式 Linux 操作系统的多线程编程思想。实际部署表明：系统具有快速稳定响应控制灯具动作的特点，友好的人机界面以及响应网络数据低延迟的优点，对大型港口、厂房等智能灯光控制系统的发展具有重要的实际意义。

#### 0 引 言

目前，国内港口码头的作业均采用全天侯工作模式，规模庞大的高杆灯群、岸桥灯组是保证港口码头夜间作业的必要设施。目前普遍使用的对灯具的控制还是采用传统的配电房控制器开关电，或者有些采用定时继电器来实现对灯具的开关灯处理。在目前来看，这两种处理措施存在这一定的弊端：远程配电房控制开关灯，需要明显的人为参与，控灯复杂，有时候还需要到灯杆下的控制器手动开电，并且大部分时候配电房距离控制灯杆存在一定的距离，开关灯具有明显的高人工的特点；而采用定时继电器控制灯具，则存在控制不灵活，开关灯时间范围需要修改的话，需要手动的去灯杆下面的控制柜修改定时继电器的配置。除了以上描述的几个缺点外，传统的灯具控制技术存在无法主动检测灯具状态是否正常、无法动态修改灯具的亮度等级以及因为没有接入网络而无法通过网络或者手机 APP 控制灯具的功能。  
本文在对已有传统灯具控制器技术分析研究基础上，采用基于嵌入式 Linux 的三层软件设计准则开发智能灯具控制器的软件。传统的两层软件设计准则为：软件直接和硬件模块通讯，不设计任何操作系统，目前这个设计方法适用在一般的小型嵌入式系统；而三层软件设计方法，则是在应用程序和硬件层中间又添加了一层内核，内核提供给应用程序控制底层硬件的方法。对于复杂的程序设计，操作系统可以屏蔽硬件的差异，并且同一版本的操作系统可以提供一致的访问硬件的方法给应用层。降低了更换硬件需要修改 APP 程序的复杂度。

#### 1 总体设计

嵌入式智能灯具控制器的目的在保证高可靠性的前提下减少人为对作业环境灯光控制的干预，实现自动化、智能化的灯光环境控制。既可以保证满足环境作业的灯光亮度要求，也可以尽量节省能耗，响应国家号召实现节能减排以及智能制造。系统的软件总体设计方案如图1所示，软件设计分为三层，底层是硬件、中间层是嵌入式 Linux 操作系统，最上层是 APP 服务程序。  
  
图 1  
硬件设备层连接了 spi 接口的 LoRa 模块， 串口的 4G 模块，以及串口的触摸显示屏模块。操作系统提供了访问 spi ， uart 这类标准通讯接口底层方法，通过标准的 read、write 和 ioctl 系统调用来和这些接口的外设通讯。

#### 2 软件设计

智能灯具控制器的软件设计基于嵌入式 Linux 操作系统开发。首先 Linux 是现代的类 UNIX 操作系统，具有高可靠性，开源并且没有许可限制的操作系统。根据公开的报道截止 2019 年 9 月份，世界上前 500 的超级计算机，99.6 % 的运行 Linux 操作系统。因为 Linux 具有如此重要的优势，不仅仅是在传统的 PC 工作站开发，在嵌入式设备上开发也可以很好的兼容目前任何基于 Linux 的发行版操作系统，所以围绕 Linux 嵌入式开发的资料和工具都很丰富。除了可以直接获取操作系统的系统的源码之外，还可以获取对应的交叉编译工具链的源码，目前有很多工具支持方便的编译对应的交叉编译工具链，比如：builroot、crosstool-NG。容易获取的开发环境也是社区或者厂商愿意选择 Linux 作为系统开发的操作系统的原因。基于 Linux 的如上特点，采取基于嵌入式 Linux 操作系统完成本次智能灯具控制器的软件设计。  
嵌入式设备相比传统 PC 具有一个明显特点就是存储部分，一般具有较小的存储空间和内存，很少有嵌入式设备需要外接硬盘来作为存储介质，本次嵌入式设备的硬件采用 128MB 的 NAND flash 为存储媒介，保存 U-Boot、内核和文件系统，并且带有大小为 128MB 的 DDR2 内存空间。其中考虑到长期擦写 NAND flash 同一块区域容易产生坏块的影响，为了保持磨损均衡采用 UBIFS 文件系统，根据实际部署的服务空间大小，将 flash 从 16MB ～ 80MB 这部分内存作为 rootfs，末端的 80MB ～ 128MB 作为辅助的一个分区自动挂载到 /opt 目录，为了尽量保持 rootfs 完全只读，而文件可以只在 opt 目录进行操作，这样处理可以降低长期运行导致 rootfs 损坏致使系统无法启动的风险。  
智能灯具控制器软件基于 Linux，采用多线程编程的设计方法，多线程软件相比单进程以及单线程具有更高的执行效率，通过将大量的任务分配给不同的线程，这样当一个任务阻塞的时候，可以继续执行其他任务，避免了导致 CPU 的浪费，特别是在多 CPU 的系统环境，可以更大化的提高 CPU 的利用率。同时，多线程和多进程相比，具有更加高效率的资源共享策略，因为同属一个进程的多线程具有相同的内存空间，而多进程之间具有各自独立的内存空间，这样在资源共享方面，多线程更加高效率。创建线程和线程切换的开销相比创建进程和进程切换更低，同时线程的启动速度相比进程的启动速度至少快 4 倍。多线程相比多进程在具有这些有点的同时，也有一些对应的缺点，比如线程之间的安全问题，要知道多线程具有相同的内存空间，也就是说同属一个进程的各个线程看到的虚拟内存是一致的，虽然目前对于 32 bit 给予 ARM 的操作系统可以看到 4GB 的虚拟内存空间，但是对于嵌入式设备实际的物理内存大小一般远比这个理论值小，如果过度使用虚拟内存在低物理内存的环境如果是 swap 交换分区那么会明显降低系统响应速度。所以这个时候就不太适用单进程多线程编程。针对那些内存占用较低，单元模块数据交互频繁 的场景可以使用多线程的方法来提高系统的响应，但是编程的时候要小心设计多线程的内存分配以及线程间通讯的方法，避免出现单线程阻塞导致整个服务进程完全卡住的现象出现。  
线程作为操作系统处理任务的最小单元，为了体现多线程的快速响应，将智能灯具控制器服务功能拆分为多个模块，分派给多个线程处理。  
从上图-1 可知，整个灯具控制器的软件采用三层开发模型，应用服务程序直接或者间接地通过系统调用到内核态，完成和 ARM 灯具控制器的硬件通讯。重点的开发工作是在应用程序这个层次。应用程序的软件框架可以展示为下图-2：  
  
从图-2 可知，整个灯具控制器软件的应用为单进程多线程架构。以港口岸桥这种场合为例，工作状态下，系统一共包含 10 个线程：具体线程负责的任务如表-1：

|  |  |
| --- | --- |
| 线程名 | 任务描述 |
| JLedSrv | 系统的 main 线程 |
| watchdog | 看门狗线程，负责监控系统工作是否异常，异常时重启系统 |
| screen | 监控串口触摸屏幕消息并派发消息以及负责屏幕显示内容的线程 |
| loraspi | 接收 lora 模块收到的消息的线程 |
| modbus | 通过 modbus 模块解析岸桥小车的位置信息，根据小车的位置信息完成自动亮度调节 |
| power | 扫描末端灯具状态并统计末端灯具的能耗 |
| plan | 处理本地预案的线程 |
| network | 接受并提取网络数据包消息的线程 |
| handle\_screen | 处理触摸屏幕消息的线程 |
| handle\_network | 处理平台的网络数据包消息的线程 |

当系统启动时，会首先到 JLedSrv 这个 main 线程，在该线程通过解析配置文件关于线程集合的定义，创建对应的线程，特别地，考虑到触摸屏命令解析、平台网络数据包解析以及和灯具的通讯这三大模块具有明显的异步性和并发的可能，为了提高系统的执行效率，采用了类 C-S 架构的处理方法。在处理触摸屏命令的时候，Client 端可以认为是线程 screen。这个线程单纯的倾听通过串口和 arm 核心控制链接的句柄，当有触摸事件发生时，该线程就可以及时获取到对应的数据，如果对数据教研通过，就会提取有用的信息，然后直接派发给处理该命令的线程 handle\_screen，handle\_screen 线程就类似于后台的服务端处理触摸屏事件，这样就可以实现在处理触摸屏事件的时候，不至于完全卡住无法处理下一个事件。关于平台网络的命令解析也是类似的，有作为类 Client 的 network 线程和类 Server 的 handle\_network 线程。关于通过 Lora 模块和末端灯具通讯的任务，考虑的多个线程都有可能触发和灯具控制事件：比如到达了预案定义的时间节点、平台有控灯动作或者本地有控灯动作，再加上和末端灯具的通讯，除了广播情况，单播都是要保证一发一收的可靠通讯，如果将发送和接收数据放在同一个线程，那么处理并发问题的时候，就不可避免的需要加锁来保证互斥。但是考虑到对 Lora 数据的接收具有明显的延迟性，所以如过锁加的时间太长，在以上列举的三个场合就有可能出现明显卡顿的现象，举例来说，如果当预案线程因为预设的时间到达，执行单灯控制指令，同时，平台端也触发的控灯的动作，这时候因为单灯控制会在读取 Lora 模块数据返回后才会释放对应的锁，所以平台端会一直卡在控灯的界面，这时候就会出现卡端的现象，这是优秀的产品不应该具有的效果。  
通过上述分析，为了保证系统可以快速响应以及数据处理的一致性，设计通过 Lora 和末端灯具的通讯方案时，采用了将数据接收的任务全部放在了线程 loraspi 完成，该线程完全负责接收 Lora 模块的数据，然后将数据保存到缓存区域。其他线程在获取 Lora 模块接收到数据的时候可以直接从缓冲区取数据。同时，在等待 Lora 模块返回数据时，添加了超时时间，loraspi 线程通过信号量的方式通知等待消息的线程。通过这两个措施，可以保证系统不会出现长时间卡住的现象。

#### 3 结束语

本文根据智能灯具控制器软件的系统要求，设计开发了基于 Linux 操作系统的灯具控制器应用服务程序。针对系统具有模块数量多，关联较为复杂的特点，采用了单进程多线程的框架设计。系统参数配置通过 cJSON 格式的文本文件确定，实现了配置和功能分离的功能，可以保证程序具有更好的适应性以及避免了修改配置需要重新变异文件的步骤，可以明显降低后续维护的负责性。该系统的实现方法和框架设计，具有明显的灵活性、和快速响应的特点，具有一定的实际意义和运用推广价值。

#### 参考文献：

[1] https://www.omgubuntu.co.uk/2017/11/linux-now-powers-100-worlds-top-500-supercomputers