USB新建新设备usb\_new\_device

首先声明，在此usb\_new\_device新建设备之前，usb\_alloc\_new\_device已经分配了新设备的存储地址以及某些相当基础的信息。

1 usb\_new\_device（USB设备），此函数只调用了usb\_setup\_device（）来创建设备，以及usb\_hub\_probe（）来探测新建的设备是不是集线器设备。两函数具体的功能如下。

2.1 usb\_setup\_device（USB设备，布尔变量read，父设备），这个布尔值永远为true。此函数的作用是，向已经分配好空间的USB设备写入他的详细信息，包括描述符等。该过程又分为两部分执行分别是usb\_prepare\_device（）准备USB设备，以及usb\_select\_config（）对设备进行更深层其的配置。

2.2 usb\_hub\_probe（USB设备，接口号），此函数下调用的usb\_hub\_check（）函数探测设备是否是集线器，如果不是集线器直接返回0，如果是集线器先调用函数usb\_hub\_configure（）对该集线器进行配置然后返回（配置不出错也是返回0）。

3.1 usb\_prepare\_device（USB设备，设备地址，布尔值read，父设备），通过usb\_setup\_descriptor（）来设置设备描述符中的最大传输数据包大小，通过usb\_set\_address（★）设置地址，本质就是调用usb\_control\_msg向下位机发送对应指令。

3.2 usb\_select\_config（USB设备），首先用get\_descriptor\_len（★）获取设备描述符，并用存储器复制memcpy将设备描述符从临时缓存区tmpbuf拷贝到dev->descriptor。用usb\_get\_configuration\_len（★）获取配置描述符长度，如果该长度大于0再利用usb\_get\_configuration\_no（★）获取配置描述符，并存储在临时缓存空间中。用usb\_parse\_config（）函数解析存储在数据缓存区中的配置描述符，解析后依次写入USB设备。用usb\_set\_maxpacket（）根据配置描述符设置每个端点最大传输数据包的大小。利用函数usb\_set\_configuration（★）向下位机发送对应命令更改其配置。最后清除残留的设备制造商信息、产品信息和产品序列号，重新写入上述信息。至此，当前USB设备握手成功。

3.3 usb\_hub\_check（USB设备，接口号），该函数识别USB设备是否是集线器，如果是集线器就返回0，不是集线器的话返回一个负值。判别方法用的是排除法，该接口的类iface->desc.bInterfaceClass如果不是集线器HUB类型的话排除，如果接口子类iface->desc.bInterfaceSubClass既不是0也不是1的话排除，如果该接口对应多个端点即iface->desc.bNumEndpoints不为一的话排除，如果该端点是输出端点的话排除，如果该端点不是中断端点的话排除，剩下的就是集线器了。

3.4 usb\_hub\_configure（USB设备），程序运行到此处已经能确定当前的USB设备是集线器了，所以我们要调用此函数对这个集线器进行配置。此配置分为以下三个方面：配置当前这个集线器相关信息，检查当前集线器配置信息，将此集线器加入扫描列表以扫描该集线器下挂载的更多USB设备。

首先，调用函数usb\_get\_hub\_device（）查看集线器设备数组hub\_dev[usb\_hub\_index]判断集线器的连接个数是否达到上限，如果没达到上限就分配一个设备空位给集线器设备hub，将当前集线器设备hub绑定其对应的USB设备。调用usb\_get\_hub\_descriptor（★）函数获取集线器设备描述符与其长度，将设备描述符指针descriptor指向数据缓存区buffer。在描述符叙述长度与描述符结构体实际长度间取最小值确定描述符实际长度，再次调用函数usb\_get\_hub\_descriptor（★）准确的获取设备描述符，并将获取到的设备描述符数据拷贝到hub->desc中。

接下来根据获取到的设备描述符配置集线器设备，如集线器特征，集线器位图，该集线器最大连接的子设备数量，配置集线器设备电源，判断该集线器是复合设备的一部分还是独立集线器，检测端口的过流保护，检测集线器设备描述符中的协议，配置集线器反应时间。

用函数usb\_get\_hub\_status（★）获取集线器状态，调用函数usb\_hub\_power\_on（）打开集线器电源。

用for循环根据此集线器子设备数量分配扫描设备空间，并添加到扫描设备列表。调用函数usb\_device\_list\_scan（）开始扫描集线器子设备。

4.1 usb\_setup\_descriptor（USB设备，布尔值），根据USB设备的速度来判断它是低速设备、高速设备还是全速设备。如果是低速和高速设备就按照其速度，直接配置其最大传输包的大小dev->descriptor.bMaxPacketSize0，并将其写入对应的输入输出端点dev->epmaxpacketin\out[0]中，最后对此大小进行编码，写入dev->maxpacketsize中方便计算。如果是全速设备就调用get\_descriptor\_len（★）函数获取最大数据包大小后设置。

4.2 usb\_parse\_config（USB设备，数据缓存区），解析位于缓冲区中的配置，并填充 dev->config 结构。先将数据缓存区全部的配置描述符数据拷贝到dev->config中，然后对数据缓存区的数据进行分段解析，如果发现是新的接口数据就将他拷贝到if\_desc，如果是新的端点数据就将他拷贝到if\_desc->ep\_desc[epno]。

4.3 usb\_set\_maxpacket（USB设备），设置给定配置中所有端点的最大打包值。两层for循环，第一层遍历该设备所有接口dev->config.desc.bNumInterfaces，第二层遍历对应接口下所有端点dev->config.if\_desc[i].desc.bNumEndpoints，多次调用函数usb\_set\_maxpacket\_ep（USB设备，接口号，端点号）。在该函数中，根据端点性质ep->bmAttributes计算出该端点是控制端点、输入端点还是输出端点，最后向相应位置赋值最大打包值。

4.4 usb\_hub\_power\_on（USB集线器设备），首先从USB集线器设备找到对应的USB设备。遍历所有子设备，用usb\_set\_port\_feature（★）函数使能子设备电源。设置延时时间值以便在查询设备之前其电源稳定工作，并设定开机超时时间值。

4.5 usb\_device\_list\_scan（无参数）扫描集线器设备下的子设备。循环进行如下过程：首先调用函数list\_empty（）查看当前集线器下的扫描设备列表是否为空，如果为空代表所有子设备遍历完成，可以结束对当前集线器的遍历。否则的话，先用list\_for\_each\_entry\_safe（）函数检查当前扫描设备列表入口是否是符合规范的，然后进一步用usb\_scan\_port（）开始对特定子设备开始扫描，直到扫描不到设备结束循环。

5 usb\_scan\_port（扫描设备），首先加载USB设备、集线器设备以及扫描端口。先用usb\_get\_port\_status（★）函数获取端口状态，如果获取失败就从扫描列表删除此设备。从上述端口状态读取端口连接变化信息与设备连接状态信息，如果端口连接无变化且无设备连接说明是空集线器，从扫描列表删除此设备。端口复位与端口超速就调用usb\_clear\_port\_feature（★）清除端口特性。

到此说明集线器下有子设备连接，且新设备已经准备就绪。调用usb\_hub\_port\_connect\_change（）函数改变端口连接。清除端口特性后检查有没有子设备连接，如果有的话再次调用usb\_hub\_port\_connect\_change（）函数。

最后检查状态：端口状态为暂停就清除端口特性，电流过载状态就清除端口特性并重新设置。如果未达到最大扫描计数，则返回而不从扫描列表中删除设备，这将重新发出新的扫描，扫描子设备的子设备。否则我们已经完成了这个子设备的扫描，所以让我们从扫描列表中删除这个设备并返回。

6 usb\_hub\_port\_connect\_change（USB设备，端口），这个函数是重中之重，就是靠它实现了集线器扫描的核心。先调用函数usb\_get\_port\_status（★）获取端口状态，再调用usb\_clear\_port\_feature（★）清除端口特性，根据端口特性判断该端口下是否有连接，如没有任何连接立即返回。

确认有连接后调用usb\_hub\_port\_reset（★）复位端口，根据端口状态配置通信速度。最后掉用函数usb\_alloc\_new\_device（**递归**）分配新设备空间，配置父设备子设备端口关系实现连接，调用函数usb\_new\_device（**递归**）新建设备。

1

2.1

2.2

3.1

3.2

3.3

3.4

4.1

4.2

4.3

4.4

4.5

5

6

★

★

★

★

★

★

★

★

**新建USB设备框架**

注：带★的表示调用底层函数