به نام خدا

USB و درایو نویسی

نویسنده:

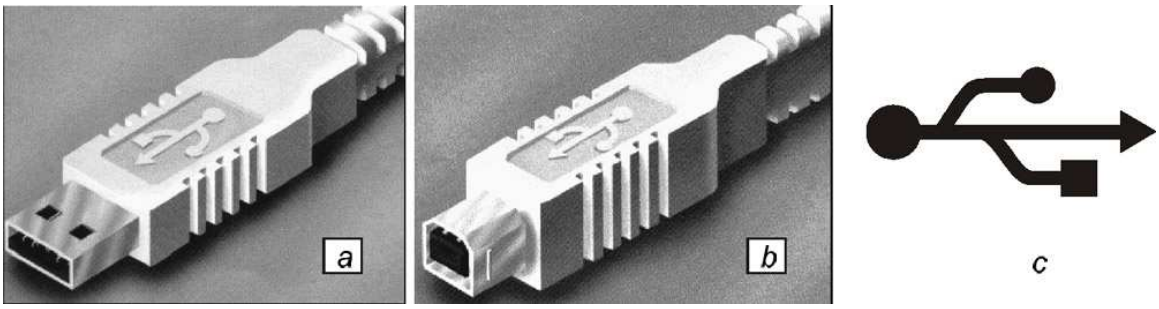
حسین پایمرد

کلمات کلیدی:

پورت **USB** –درایو نویسی.

۰-۱- مقدمه

استانداردها درکامپیوترهای جدید برمبنای کاهش استفاده از اسلاتها و پورتهای سری وموازی قرارگرفته است.به همین دلیل پورت جدیدی به نام USB(Universal Serial Bus) ساخته شده است.علامت USB درزیر نشان داده شده است و بر هر دستگاهی دیده شود نشان دهنده پشتیبانی از USB است.



شکل ۱-۱- علامت USB

از مزایای این پورت می توان به موارد زیر اشاره کرد.

۱. اهداف استاندارد های جدید را کاملا در بر می گیرد.
۲. دستگاههای جانبی را وقتی که کامپیوتر یا دستگاه مجهز به پورت USB روشن است می توان وصل و یا قطع کرد.
۳. امکان استفاده از ۱۲۷ دستگاه جانبی
۴. عدم نیاز به وقفه ها یا جمپر هابرای دستگاه های جانبی
۵. فشردگی ومتمرکزوکوچک بودن یکی از قابلیتهای مهم دستگاه هایUSB است.
۶. USB کلاسهای بسیاری را به سیستمهایی که از آن استفاده می کنند افزوده است.
۷. دستگاههایی که بر اساس استانداردهای پایین USB ساخته شده اند با نگارش های بالاتر سازگارند.
۸. امکان دریافت تنظیم های مختلفی از سوی کامپیوترکار را ساده کرده است.
۹. پروتکل USB بسیار انعطاف پذیر است و به راحتی می توان حالت انتقال داده را در آن تغییر داد.
۱۰. دستگاه های USB نسبت به سرعت بسیار بالایی که دارند ارزان قیمت هستند.
۱۱. انتقال دستگاههای جانبی از یک کامپیوتر به کامپیوتر دیگر بسیار ساده است.
۱۲. استفاده از آن برای کاربر بسیار آسان است.

همچنینUSB دارای ویژگی های زیر نیز می باشد:

پیکربندی خودکار

هنگامی که کاربر وسیله جانبی USB را به سیستم خود متصل می کند.ویندوز به طور خودکار آن را پیدا کرده و راه انداز مربوط به آن را بارگذاری می کند.

آزاد کردن منابع سخت افزاری برای وسایل جانبی دیگر

استفاده از USB این امکان را فراهم می کند که خطوط IRQ برای دیگر وسایل جانبی که نیاز به استفاده از آن دارند آزاد شود.در کامپیوتر تعداد خطوط IRQ محدود است و عدم اجبار به تخصیص یک خط به وسیله جانبی خاص خود می تواند یک دلیل کافی برای استفاده ازUSB باشد.

عدم احتیاج به منبع تغذیه

رابط USB شامل سیم های زمین و تغذیه نیز می باشد .وسایل جانبی که حداکثر به ۵۰۰ mA جریان نیاز دارند می توانند از این سیم جریان مورد نیاز خود را دریافت کنند بدون آنکه به منبع تغذیه مجزا نیاز داشته باشند.

جدول زیر بعضی از ویژگی های USB را با دیگر رابطهای متعارف مقایسه می کند.

Interface	Format	Number of Devices (maximum)	Distance (maximum, feet)	Speed (maximum, bits/sec.)	Typical Use
USB	asynchronous serial	127	16 (up to 96 ft. with 5 hubs)	1.5M, 12M, 480M	Mouse, keyboard, drive, audio, printer, other standard and custom peripherals
Ethernet	serial	1024	1600	10G	General network communications
IEEE-1394b (FireWire 800)	serial	64	300	3.2G	Video, mass storage
IEEE-488 (GPIB)	parallel	15	60	8M	Instrumentation
IrDA	asynchronous serial infrared	2	6	16M	Printers, hand-held computers
I^2C	synchronous serial	40	18	3.4M	Microcontroller communications
Microwire	synchronous serial	8	10	2M	Microcontroller communications
MIDI	serial current loop	2 (more with flow-through mode)	50	31.5k	Music, show control
Parallel Printer Port	parallel	2 (8 with daisy-chain support)	10–30	8M	Printers, scanners, disk drives
RS-232 (EIA/TIA-232)	asynchronous serial	2	50-100	20k (115k with some hardware)	Modem, mouse, instrumentation
RS-485 (TIA/EIA-485)	asynchronous serial	32 unit loads (up to 256 devices with some hardware)	4000	10M	Data acquisition and control systems
SPI	synchronous serial	8	10	2.1M	Microcontroller communications

جدول ۱-۱- مقایسه USB با انواع رابط های استاندارد

تاریخچه

نسخه ۱.۰۰ مرجع خصوصیات USB است و در سال ۱۹۹۶ ارایه شد. نسخه بعدی نگارش ۱.۱ بود که در سال ۱۹۹۸ ارایه شد.در این نسخه مشکلاتی که در تعاریف نسخه قبلی بود اصلاح شد و یک روش انتقال جدید به نام وقفه ای در این نسخه به پروتکل USB اضافه گردید.در آوریل ۲۰۰۰ جدید ترین نسخه USB با ارایه سرعت بالا به جهان معرفی شد. ساخت این نگارش از USB یک گام بزرگ در توسعه USB بود. در این نسخه سرعتی ارایه شد که ۴۰ برابر از سرعتهای قبلی بالاتر بود. این سرعت نظر سازندگان دستگاه هایی ما نند اسکنردرایورها و دوربین های دیجیتالی را به USB جلب کرد.

درجدول زیرنگارش های مختلف این پورت را به همراه تاریخ سا ختشا ن می بینید.

Revision	Issue Date	Comments
0.7	November 11, 1994	Supersedes 0.6e.
0.8	December 30, 1994	Revisions to Chapters 3-8, 10, and 11. Added appendixes.
0.9	April 13, 1995	Revisions to all the chapters.
0.99	August 25, 1995	Revisions to all the chapters.
1.0 FDR	November 13, 1995	Revisions to Chapters 1, 2, 5-11.
1.0	January 15, 1996	Edits to Chapters 5, 6, 7, 8, 9, 10, and 11 for consistency.
1.1	September 23, 1998	Updates to all chapters to fix problems identified.

جدول ۱-۲- نگارش های مختلف پورت USB

1-1- حالات کاری USB

USB درسه نوع حالت کاری فعالیت می کند. مهمترین تفاوت آنها در سرعتشان است. بنابراین نام این حالت ها بر اساس سرعت انتقال اطلاعات نامگذاری شده اند. درادامه این حالات کاری توضیح داده شده اند.

1-1-1- سرعت پایین (Low_Speed)

از این حا لت بیشتر دردستگاه های واکنشی (interactive) استفاده می شود.سرعت انتقال اطلاعات دراین حالت از ۱۰ Kb/s تا ۱۰۰ Kb/s متغییر است. از مهمترین خصوصیات حا لت سرعت پایین ارزان قیمتی و راحتی برای استفاده کاربر است. علاوه بر آن عملیات الحا ق و جدا سازی را به صورت پویا انجام می دهد(Dynamic Attach-Detach) حالت سرعت پایین برای دستگاه هایی مانند صفحه کلید,ماوس, قلم نوری ,دستگاه های بازی و دستگاه های واقعیت مجازی (Virtual Reality) بسیار مناسب است.

1-1-2- سرعت بالا (Full_Speed)

از این حا لت کاری بیشتر در دستگاه ها یی که با صوت و تصاویر ویدیویی کار می کنندااستفاده می شود. سرعت انتقا ل داده ها درسرعت‌بالا از500 Kb/s تا 10 Mb/s متغییر است. از خصوصیا ت بارز این حا لت علاوه بر مزایای موجود در حا لت سرعت پایین می توان به زمان عکس العمل وپهنای باند تضمین شده آن اشاره کرد. از این حالت در دستگاه ها یی مانند POTS , Broadband , دستگا ه های صوتی و میکروفون ها استفاده می شود.

1-1-3- سرعت نهایی (High_Speed)

ازاین حالت در دستگاه های ویدیویی وذخیره سازی اطلاعات استفاده می شود.سرعت انتقال اطلاعات درآن از 25 Mb/s تا 400 Mb/s متغییراست. این حالت علاوه برمزایای سرعت بالا و حالت سرعت پایین دارای پهنای باند بسیار وسیعی است که آن را ازبقیه حالات متمایز می کند. از سرعت نهایی در دستگاه های تصویربرداری , دوربین های ویدیویی دیسکهای فلش استفاده می شود.

2-1- تعدادی از اصطلاحات USB

- **نقطه پایا نی (Endpoint)**

بخش آدرس پذیری از دستگاه های USB است که منبع و یا گیرنده اطلاعا ت در یک جریان ارتباطی ازآن استفاده می کنند.

- **آدرس دستگاه (Device Address)**

یک مقدار هفت بیتی ا ست که آدرس یک دستگاه USB را مشخص می کند. مقدار پیش فرض هر آدرس صفراست که هنگام ریست شدن و یا اتصا ل منبع تغذیه از این آدرس استفاده می شود.

- **لوله (Pipe)**

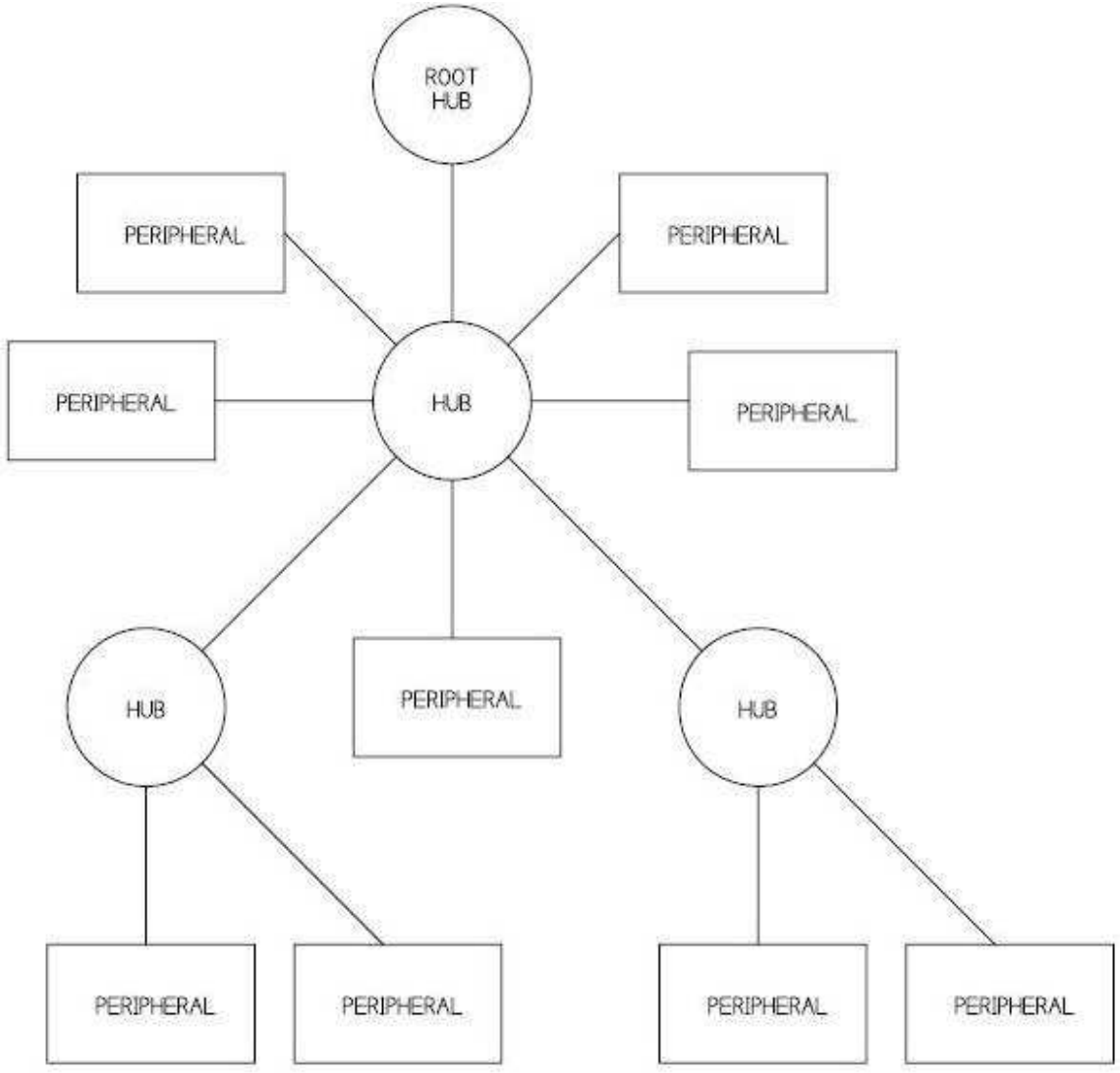
مسیر انتقال داده ها از یک نقطه پا یا نی به نرم افزار میزبان و با لعکس

- **مشخصه پاکت (PID/Packet ID)**

نام یک فیلد در USB است. این فیلد نوع بسته اطلا عاتی را مشخص می کند و قالب آن به صورتی است که ایجاد هر گونه خطا را مشخص می کند .

3-1- ساختار USB

ساختار یا نحوه اتصا لات باس به صورت ردیفهای ستاره ای است در مرکز هر ستاره یک هاب وجود دارد.در هر طرف از ستاره یک دستگاه نصب می شود که به یک پورت از هاب متصل می گردد.دستگاه ممکن است یک هاب دیگر یا وسیله جانبی باشد.تعداد نقاط هر یک از ستاره ها می تواند با توجه به نوع هاب متفاوت باشد. هاب های موجود دارای دو, چهار, هفت پورت هستند.وقتی که چندین ها ب پشت سر هم باشند می توانید تصور کنید که در یک ردیف پشت سر هم قرار گرفته اند.



شکل ۱-۲- ساختار USB

هنگام ارتباط با دستگاه USB نه میزبان و نه دستگاه متوجه نمی شوند که ارتباط به واسطه یک هاب انجام شده یا بیشتر.هاب ها به صورت خودکار این موضوع را مدیریت می کنند.

نویسنده : حسین

پایمرد

حداکثر تعداد هابی که بین یک دستگاه USB و هاب ریشه می تواند قرار بگیرد پنج عدد است. بعضی از دستگاه های USB خود دارای دو ردیف هستند که به آنها دستگاه های مرکب گفته می شودو در صورتی که از این دستگاه ها در ردیف هفتم (هفتمین هاب متوالی) استفاده شود فعال نمی گردد.

1-4- میزبان USB

درهر سیستمی که بر اساس USB کار می کند, فقط یک میزبان باید وجود داشته باشد . این قسمت را به نام کنترلر میزبان (Host Controller) معرفی کرده اند. یک میزبان USB از دو قسمت نرم افزار و سخت افزار تشکیل شده و مسئول بخش های زیر است:

۱. شناسایی اضافه ویا حذف شدن یک دستگاه USB
۲. مدیریت کنترل بین میزبان و دستگاه USB
۳. مدیریت جریان اطلاعات بین میزبان و دستگاه USB
۴. جمع آوری آمار وضعییت و فعالیت های تمام دستگاه های موجود در هرم USB
۵. آماده سازی جریان مورد نیاز برای دستگاه USB

نرم افزار میزبان , فعل وانفعالات بین دستگاه و میزبان را مدیریت می کند. این فعل وانفعالات شامل بخش های زیر می باشند.

۱. سرشماری دستگاه و تنظیم آ ن
۲. ارسال داده ها در فواصل زمانی معین (Isochronous)
۳. ارسال اطلاعات در زمان های مختلف (Asynchronous)
۴. مدیریت منبع تغذیه
۵. مدیریت اطلاعات دستگاه USB و پورت آ ن

1-5- دستگاه های USB

یک دستگاه USB باید توانایی های زیر را برای ایجاد ارتباط با USB داشته باشد.

۱. توانایی دریافت اطلاعات بر اساس استانداردهای USB

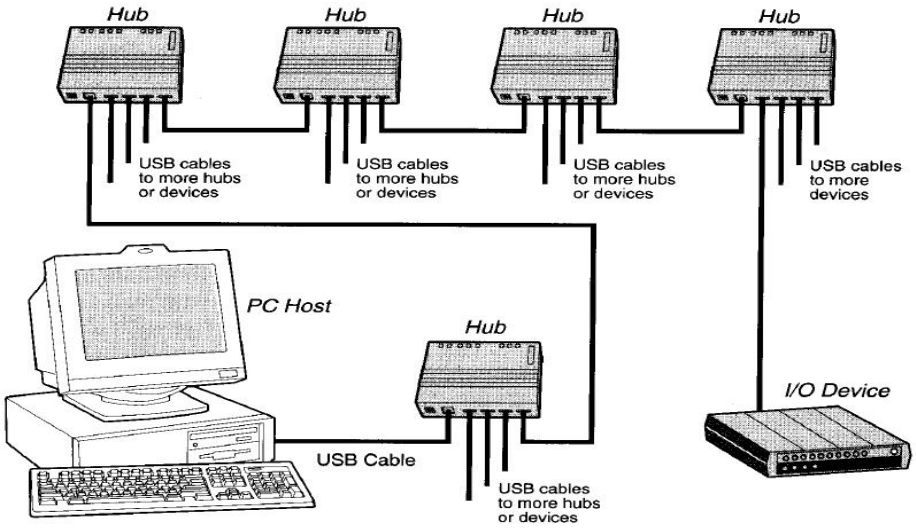
۲. توانایی آماده سازی و ارسال اطلاعات بر اساس استانداردهای USB مانند پیکربندی داده ها

۳. توانایی تشریح و تجزیه و ارسال اطلاعات USB

با توجه به موارد فوق دستگاه های USB به دو دسته زیر تقسیم می شوند:

1-5-1- هاب (Hub)

برای تشکیل و یا افزایش گره ها در هرم ازآنها استفاده می شود.تمام ها ب ها دارای کلید هایی برای تنظیم معماری USB هستند.



شکل ۱-۳- هاب

یک ها ب امکان اتصال قطعات USB را به آسانی و ارزانی در اختیار کاربران قرار می دهد. در حقیقت هاب ها امکان اتصال چندین دستگاه USB را به خود مهیا می کنند. نقاطی که این دستگاه ها به هاب وصل می شوند, همان پورت های USB هستند. یکی از وظایف بسیار مهم یک هاب ترکیب سیگنال های دستگاه های USB و ایجاد یک سیگنال ترکیبی برای انتقال اطلاعات است.هر هاب دارای دو نوع پورت است , یک نوع آن پورت سر چشمه(Upstream) نامیده می شود و کابلها را به سوی میزبان هدایت می کند.نوع دیگر پورت پایین رو (Downstream) است, که برای اتصال به دستگاه USB در نظر گرفته شده است. هاب ها می توانند, تعدادی از پورت های پایین رو را غیر فعال کنند و تمرکز را بر روی بقیه پورت ها برقرار کنند.

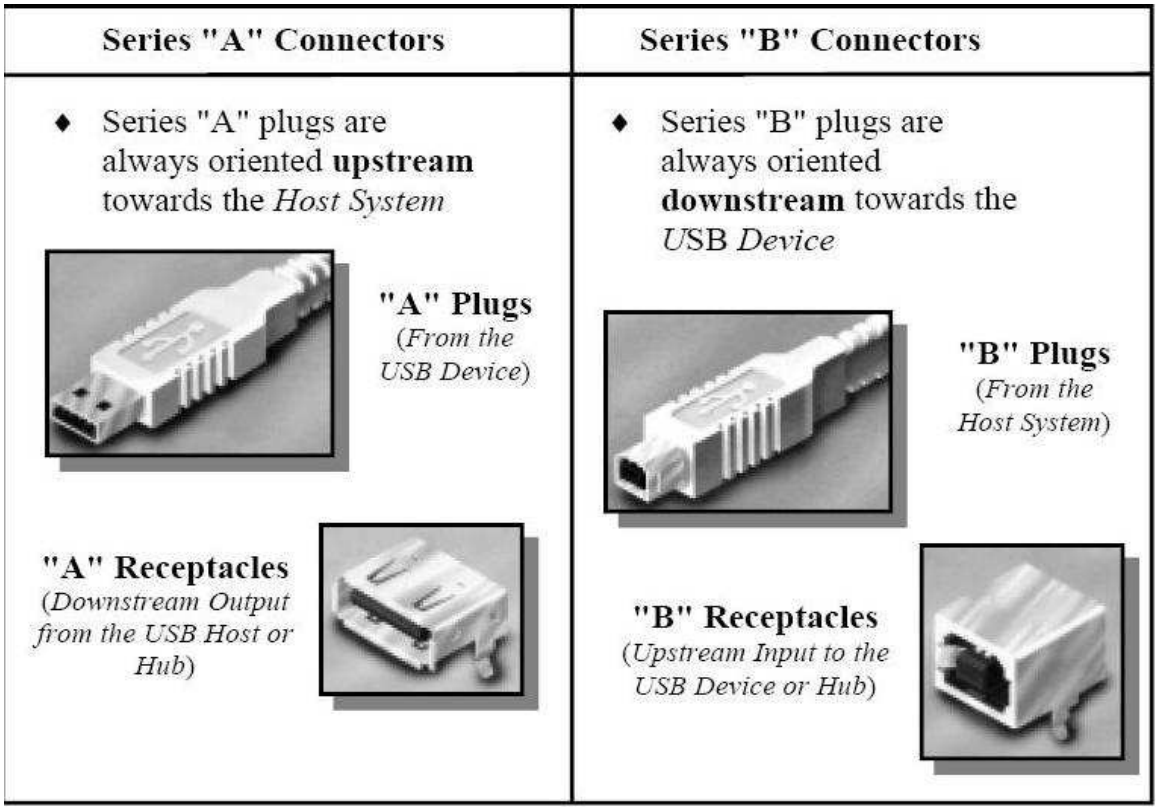
1-5-2- عامل ها (Functions)

عامل ها کار های اصلی سیستم را انجام می دهند, مانند اتصال های ISDN , دسته های بازی (Joystick) وبلندگوهای کامپیوتر (Speaker) عامل در حقیقت یک دستگاه USB است, که توانایی ارسال و دریافت اطلاعات و همچنین کنترل را از طریق USB دارد.یک عامل, به عنوان یک دستگاه جانبی به پورت متصل می شود. بعضی از عامل ها می توانندچندین کار را انجام داده و آنها را در یک بسته ترکیبی از یک کابل USB انتقال دهند. همانطور که گفته شد به این عامل ها دستگاه مرکب گفته می شود. این دستگاه ها دارای یک هاب در داخل خود هستند.

هر عامل شامل اطلاعاتی برای پیکر بندی (Configuration Information) است, که توانایی هاومنابع موردنیاز آن را مشخص می کند, همچنین این اطلاعات قبل از استفاده باید توسط میزبان تنظیم شوند.(این اطلاعات شامل پهنای باند و خصوصیات عامل هستند)

1-5-3- اتصالگرهای USB

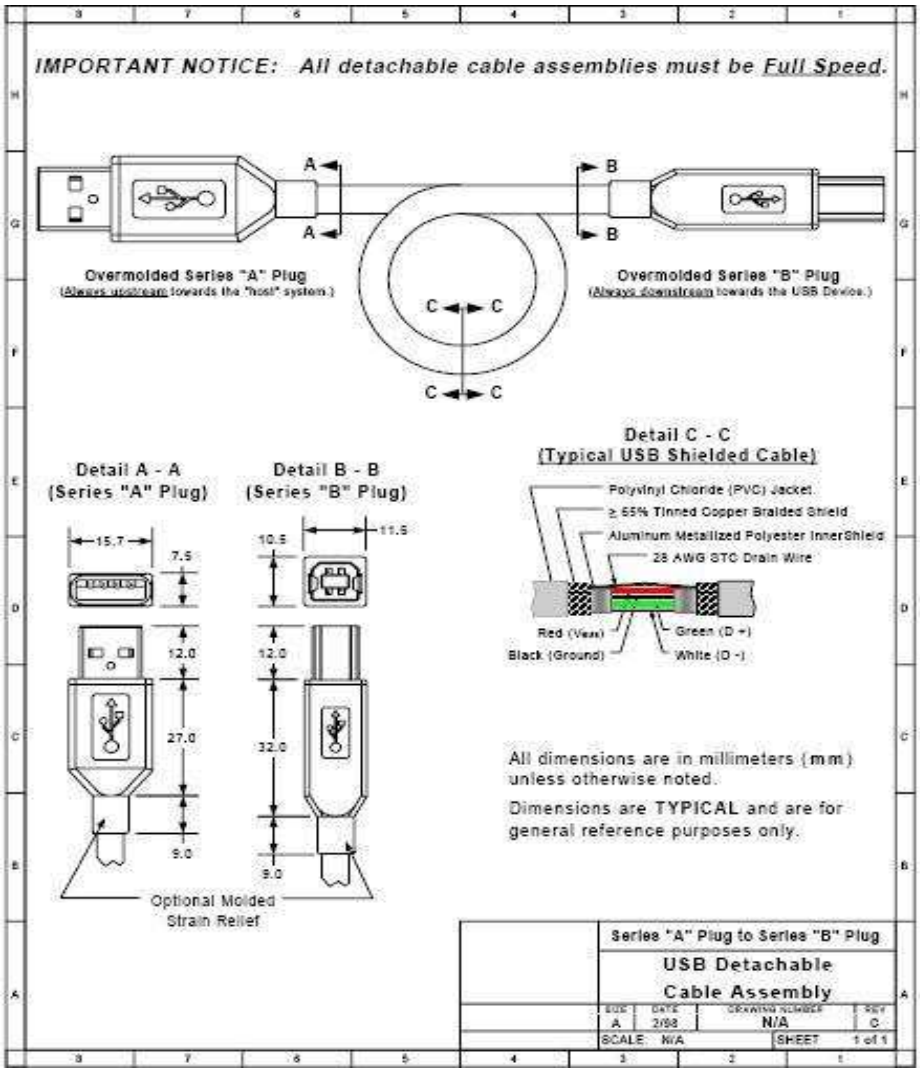
رابطهای USB همگی دارای اتصالگرهای کاملا استانداردی هستند و در دو نوع دیده می شوند.



شکل ۱-۴- اتصال گرهای USB

با توجه به دونوع بودن اتصالگرهای USB روشن است که یک کابل استاندارد USB باید در دو سر خود از دو نوع متفاوت از اتصالگرها استفاده کند , یک قسمت نوع A وقسمت دیگر نوع B .شکل صفحه بعد یک کابل استاندارد USB را نشان می دهد. دراین کابل از چهار سیم

برای انتقال اطلاعات استفاده می شود. این چهار سیم در رنگ های مختلف داده ها را بر اساس پروتکل USB جابجا می کنند. هر رنگ در این سیم ها دارای مفهوم خاصی است, که در جدول مشخص شده اند.در برخی مواقع برای کاهش نویز در دستگاه های سرعت بالا از یک سیم دیگر به نام شیلد (Shield) نیز استفاده می شود.



شکل ۱-۵- ساختار کابل USB

Contact Number	Signal Name	Typical Wiring Assignment
1	VBUS	Red
2	D-	White
3	D+	Green
4	GND	Black
Shell	Shield	Drain Wire

جدول ۱-۳- اجزاء رابط USB

1-5-4- تغذیه دستگاه های مجهز به USB

دستگاه ها یی که به USB مجهز هستند, می توانند دارای یکی از انواع تغذیه زیر باشند.

1-5-4-1- تغذیه سطح بالا(High Level)

این روش برای تغذیه دستگاه های جانبی بسیار معمول و متداول است . دراین روش , دستگاه ها از تغذیه های مستقل بهره می گیرند.

1-5-4-2- تغذیه BUS Powered

دراین روش, دستگاه جانبی از خط VBus , برای تغذیه خود استفاده می کند.جریانی که از این دستگاه می توان گرفت, بسته به نوع درایور و برنامه ریزی سیستم تا500 mA متغییر است.

1-6- تنظیم های هرم USB

یکی از قابلیت های مهم USB , خذف و اضافه کردن دستگاه ها بدون نیاز به ریست کردن سیستم است. بنابراین USB باید تغییرات هرم USB رادر هر لحظه کنترل کند که این تغییرات شامل حذف و اضافه کردن یک دستگاه USB است .

1-6-1 الحاق یک دستگاه به هرم USB

تمام دستگاه های USB فقط قادرند که به یک هاب متصل شوند.حتی آنهایی که مستقیما به کامپیوتر اتصال پیدا کرده اند در حقیقت به هاب ریشه وصل شده اند. پس در صورتی که یک دستگاه USB , به هاب متصل و یا از آن

جدا شود, وظیفه هاب است که آن را به میزبان USB گزارش دهد, زیرا میزبان دائما این تغییرات را از هاب ها درخواست می کند.در هنگام افزودن یک دستگاه , با گزارش هاب مربوط به آن, میزبان مجبور به ارایه پورت و آدرسی برای آن می شود.

آدرسی که به دستگاه الحاقی نسبت داده می شود, کاملا یکتا (Unique) است و بدین وسیله از مشکلات احتمالی پیش گیری می شود. بعد از ارایه آدرس, میزبان آن را به عنوان یک دستگاه جدید تعریف می کند و تشخیص می دهد که این دستگاه هاب است و یا یک عامل. در صورتی که آن دستگاه یک هاب باشد و عاملی به آن متصل شده باشد , عملکرد فوق دوباره تکرار می شود.

همچنین در صورتی که دستگاه مورد نظر یک عامل باشد , میزبان کنترل آن را به دست نرم افزار میزبان می دهد.

2-6-1- حذف یک دستگاه از هرم USB

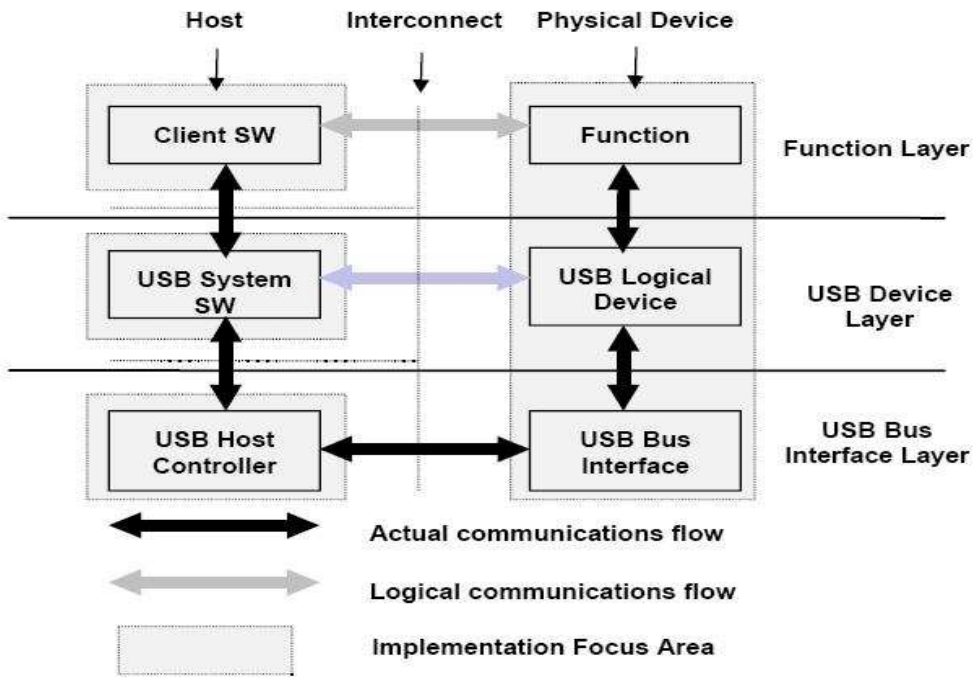
زمانی که ارتباط یک دستگاه USB , با هاب قطع می شود هاب , پورت مربوطه را غیر فعال کرده و حذف آن را به میزبان گزارش می دهد. در این هنگام میزبان ارتباط نرم افزار آن دستگاه را قطع می کند .

7-1- چگونگی جریان اطلاعات درUSB

در این بخش ساختار ولایه های انتقال اطلاعات از طریق USB بررسی خواهند شد.

1-7-1 بررسی لایه های انتقال اطلاعات

USB ارتباط بین میزبان و دستگاه متصل به آن را بر قرار می کند.شکل زیرلایه های مختلف انتقال اطلاعات را در USB نشان می دهد.



شکل ۱-۶- لایه های انتقال اطلاعا ت در USB

همان طور که در شکل نیز مشخص است , آن ها را به چهار بخش زیر می توان تقسیم کرد.

۱) **لایه فیزیکی دستگاه USB**

قسمتی از سخت افزار در انتهای کابل USB است , که قسمتی از کار های کاربر را انجام می دهد.

۲) **نرم افزار مشتری**

این نرم افزار همان برنامه نهایی است, که بوسیله یکی از زبان های برنامه نویسی نوشته و اجرا می شود.

۳) **نرم افزار سیستم USB**

نرم افزاری است که در قسمتی از سیستم عامل قرارمی گیرد و از USB پشتیبانی می کند . این نرم افزار ممکن است , بخشی از سیستم عامل و یا نرم افزار مشتری باشد.

۴) **کنترلر میزبان USB**

نرم افزار و سخت افزاری است , که در هنگام اتصال دستگاه به پورت USB آن را پذیرفته و به میزبان معرفی می کند.

2-7-1- توپولوژی انتقال اطلاعات

این توپولوژی از چهار بخش اساسی تشکیل شده است .

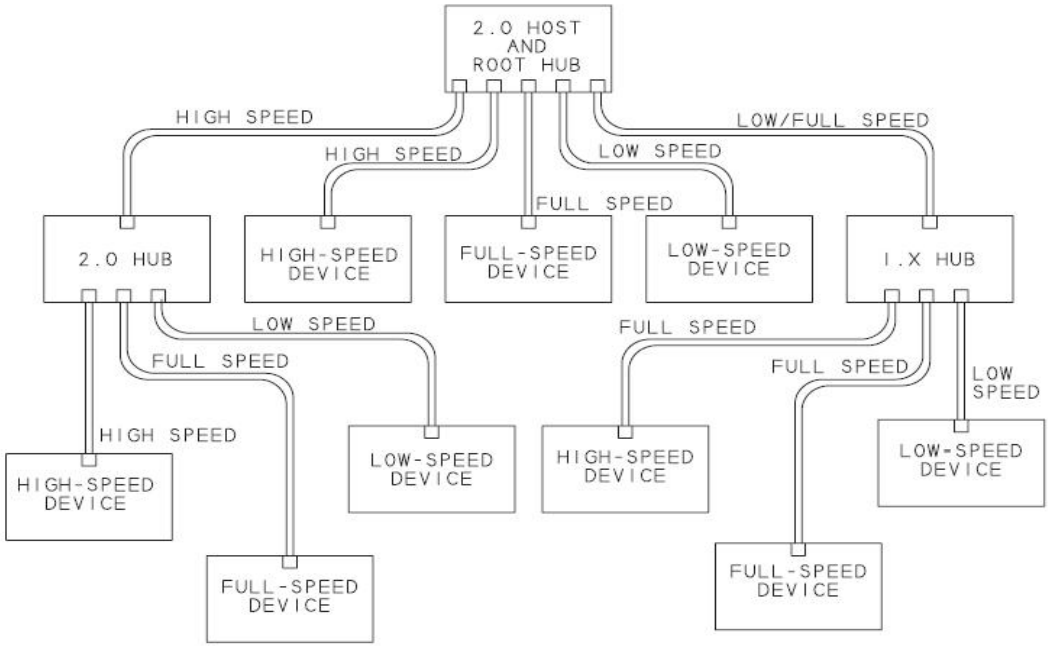
I. **میزبان و دستگاه**

اولین جزء سیستم USB را تشکیل می دهند.در حقیقت میزبان USB به عنوان یک هماهنگ کننده انحصاری در سیستم USB عمل می کند. یکی از وظایف بسیار مهم این بخش پاسخ گویی به دستگاه های جدیدی است که به هرم USB متصل می شوند.

میزبان USB تمام دسترسی ها را کنترل می کند , در نتیجه یک دستگاه زمانی می تواند به خط اطلاعات دسترسی پیدا کند که میزبان به آن اجازه داده باشد. علاوه بر موارد فوق , میزبان وظیفه کنترل توپولوژی USB را نیز بر عهده دارد.

II. **توپولوژی فیزیکی**

شامل عناصری است که به USB متصل شده اند.همان طور که در بخش های قبلی نیز گفته شد, دستگاه ها در سیستم USB بر اساس توپولوژی ستاره به میزبان متصل می شوند. دستگاه ها به نقاطی در این هرم متصل می شوند, که به آنها هاب گفته می شود. میزبان نیز دارای یک هاب ریشه است که مانند سایر هاب ها عمل می کند. هر یک از هاب ها می توانند در یک یا چند حالت کاری کار کنند. شکل زیر عملکردشان را در حالات و سرعت های مختلف نشان می دهد.

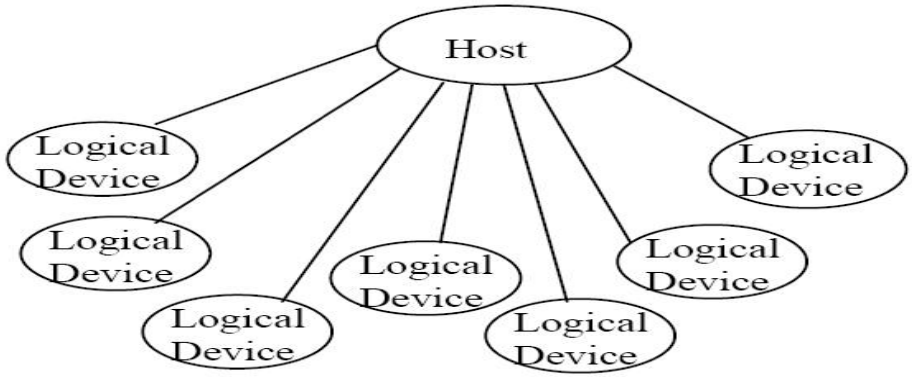


شکل ۱-۷- توپولوژی فیزیکی انتقال اطلا عات در USB

همان طور که در این شکل مشخص است , دستگاه های FS وLS (Low Speed,Full Speed) می توانند به هابی که از نگارش USB 1.1 پشتیبانی می کند متصل شوند. درصورتی که برای اتصال یک دستگاه HS (High Speed) باید یک هاب HS , را به عنوان واسط قرار داد.

III. توپولوژی منطقی

شامل قوانین و پروتکل های USB است و باعث می شود که رفتار ها و پاسخ ها با نظارت USB کنترل شوند.



شکل ۱-۸ – توپولوژی منطقی انتقال اطلا عات در USB

تا زمانی که هر دستگاهی به صورت فیزیکی به هرم USB متصل است , میزبان با تمام دستگاه های منطقی که مستقیما به پورت USB متصل هستند, ارتباط خود را حفظ می کند. هاب ها نیز دستگاه های منطقی هستند که

باعث اتصال دستگاه های منطقی بیشتری به میزبان می شوند و با قطع هاب تمام دستگاه های متصل به آن قطع می شوند.

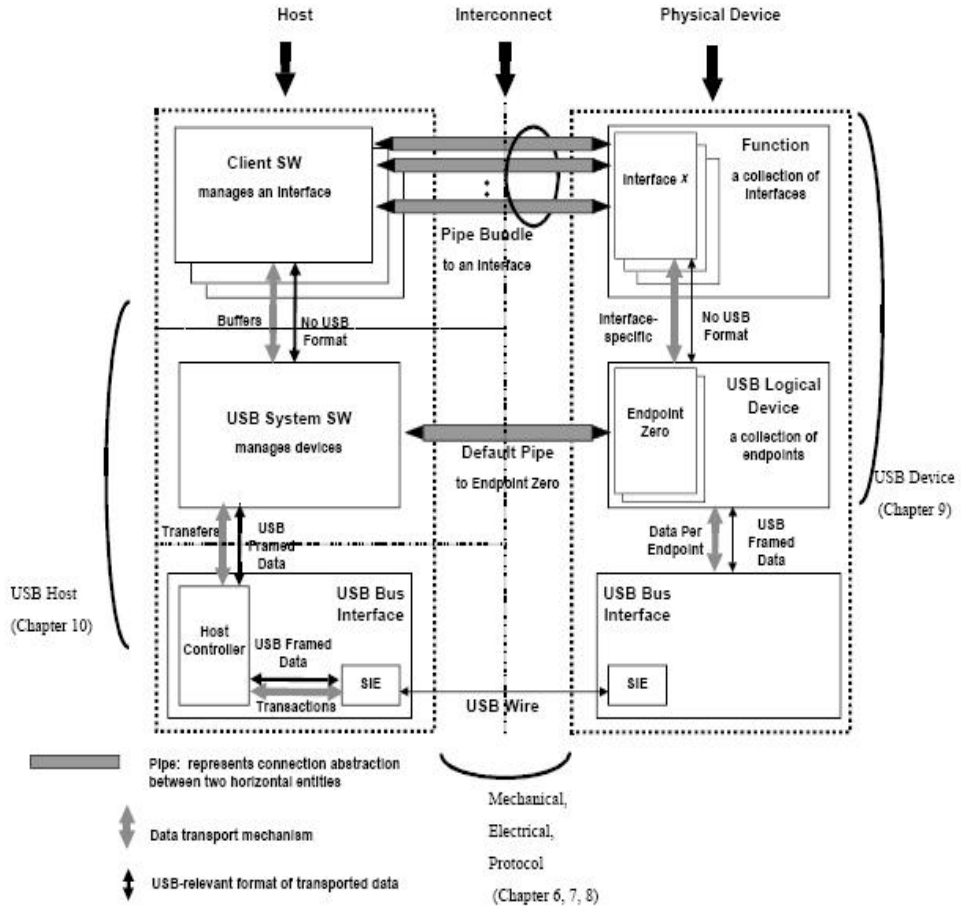
IV. رابطه نرم افزار مشتری با دستگاه USB

نشان می دهد نرم افزار و دستگاه چگونه یکدیگر را می بینند و رابطه برقرار می کنند. پس از بررسی توپولوژی منطقی و توپولوژی فیزیکی به نقطه اشتراک هر دو که نرم افزار مشتری است خواهیم رسید. این نرم افزار ها برای عامل USB خاص و به وسیله یکی از زبان های برنامه نویسی تحت ویندوز نوشته می شوند.

یک نرم افزار مشتری , باید در طول اجرا از دستگاه های دیگری که ممکن است , به میزبان متصل شوند, مستقل باشد وفقط با دستگاه مربوطه خود ارتباط برقرار کند. این امر باعث می شود که طراحان نرم افزار و سخت افزار بتوانند روی جزییات بیشتری کار کنند.

3-7-1- جریان اطلاعات در USB

مهمترین وظیفه USB آماده سازی و انتقال اطلاعات بین دستگاه USB ونرم افزار است.دستگاه های USB به جریان اطلاعاتی (Communication Flow) متفاوتی نیاز دارند, تابانرم افزار ارتباط برقرار کنند. USB روش خوبی برای جداسازی جریان های اطلاعاتی در دستگاه های مختلف دارد , به نحوی که هر یک از آنها به راحتی مسیر خود راازدستگاه USB تا نرم افزار طی می کنند. تمام این جریان ها , به وسیله یک نقطه پایانی که در هر دستگاهی وجوددارد, خاتمه می یابند. در لغت نامه USB به این نقاط , نقطه پایانی (Endpoint) گفته می شود که مهمترین کاربردآن ها , شناسایی تمام جریان های اطلاعاتی است. شکل زیر جزییات بیشتری را در رابطه با انتقال اطلاعات در USB نشان می دهد.

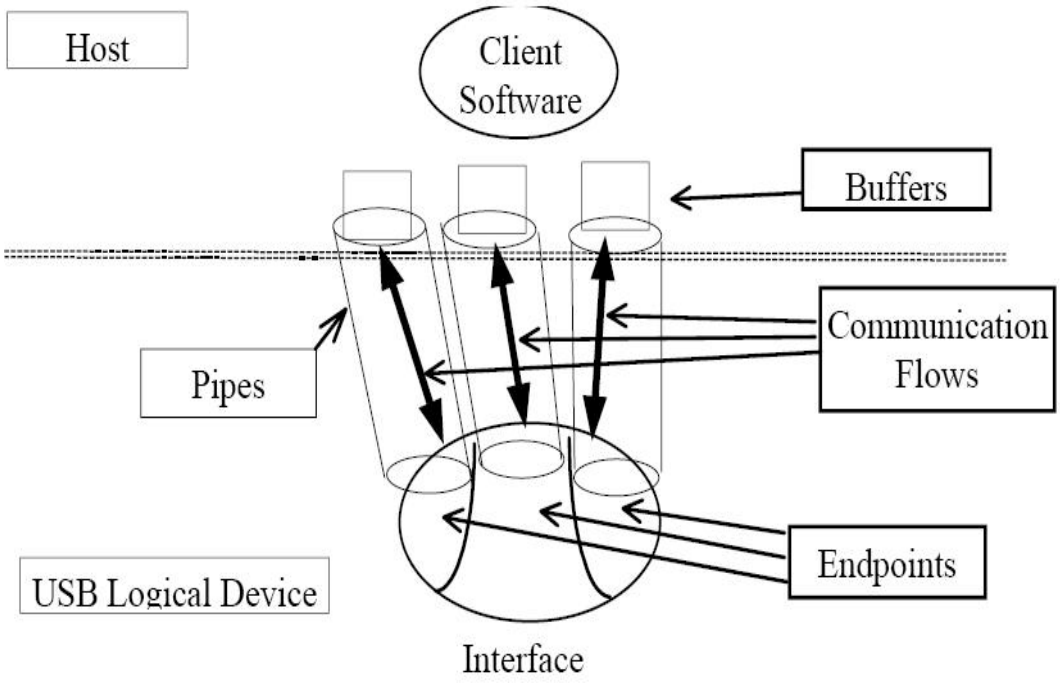


شکل ۱-۹- جریان اطلاعا ت در USB

این شکل نحوه واقعی ارتباط دستگاه های منطقی و لایه های ارتباطی را نشان می دهد. در حقیقت این جریان های اطلاعاتی ازرابط های مختلفی عبور می کنند تا به مقصد خودبرسند.

یک دستگاه منطقی , در سیستم USB به صورت مجموعه ای از نقاط پایانی مشخص می شود. نقاط پایانی نیز به صورت گروهی جمع شده واز رابط ها عبور می کنندو رابط ها نیز درگاه اتصال به دستگاه هستند. نرم افزار مشتری نیز دستگاه را به کنترل پیش فرض لوله هدایت می کند. همچنین این نرم افزار رابطهاراتنظیم می کندتا لوله ها را دسته بندی کنند(این کار نیز به وسیله نقاط پایانی انجام می شود.)

نرم افزار مشتری از USB درخواست می کند, که اطلاعات بین بافر میزبان و نقطه پایانی موجود در دستگاه جابجا شوند. کنترلر میزبان نیز اطلاعات USB رابسته بندی می کند. این قسمت , همچنین زمانی را که خط داده , اطلاعات را به USB تحویل می دهد هماهنگ می کند. شکل صفحه بعد نشان می دهد که چگونه اطلاعات بین نقاط پایانی و میزبان به وسیله لوله ها جابجا می شود.



شکل ۱-۱۰- انتقال اطلاعا ت بین نقا ط پایانی ومیزبان

نرم افزاری که در میزبان است , از طریق جریان های اطلاعاتی با دستگاه USB ارتباط برقرار می کند. مجموعه ای از این جریان ها از طرف طراحان نرم افزار و سخت افزار برای معرفی خصوصیات دستگاه در نظر گرفته شده اند.

8-1- نقاط پایانی یک دستگاه USB

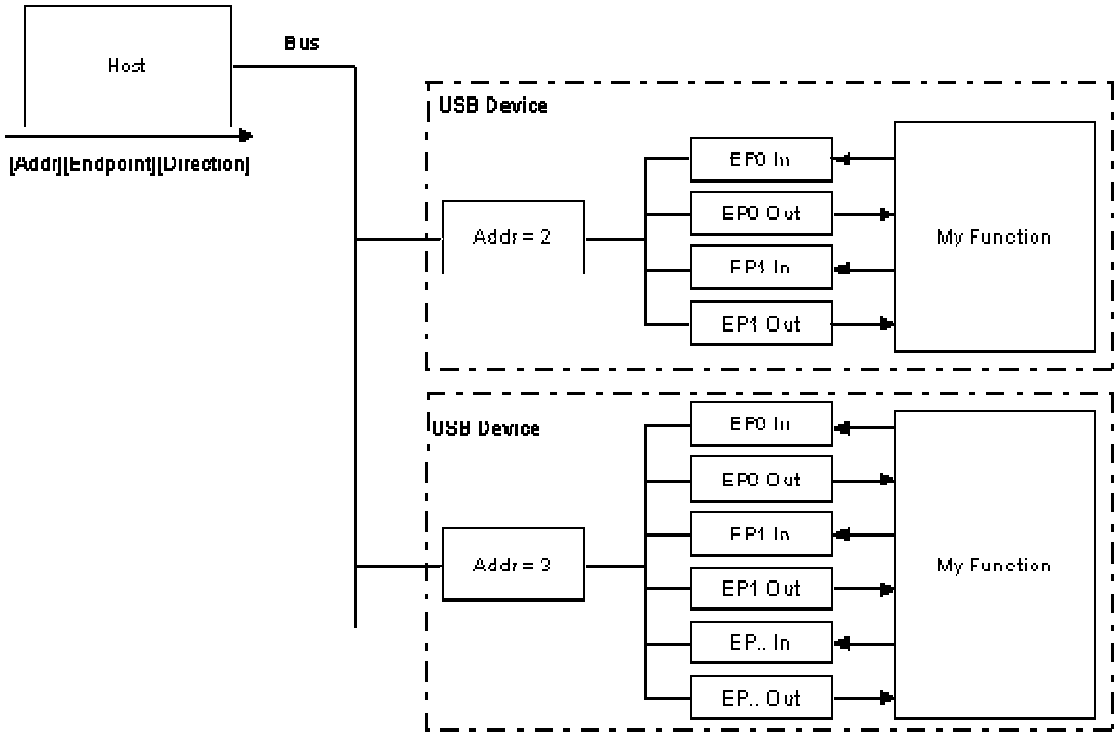
نقطه پایانی, قسمت منحصر به فردی از دستگاه USB است , که پایان جریان اطلاعاتی را بین دستگاه و میزبان مشخص می کند. تمام دستگاه های منطقی شامل مجموعه ای از نقاط پایانی مستقل از هم هستند. علاوه بر آن , این دستگاه هادارای آدرس های یکتایی هستند, که در زمان اتصال به USB سیستم به آن ها ارایه می کند. برخی از نقاط پایانی که به یک دستگاه در زمان طراحی و ساخت داده می شوند, یکتا هستند, مانند Endpoint Number,Endpoint Address . تمام نقاط پایانی با جریان اطلاعاتی یک طرفه خوانده می شوندو میزبان قادر نیست مقادیری را به آنها ارسال کند. همچنین هر کدام از نقاط پایانی دارای خصوصیات ویژه ای هستند , که نوع انتقال بین نقطه پایانی و میزبان را مشخص می کنند. یک نقطه پایانی خودش را به وسیله مواردی توصیف می کند که در صفحه بعد لیست آن ها را می بینید.

- پهنای باند مورد نیاز
- شماره نقطه پایانی
- رفتار های زمان خطا
- حداکثر اندازه یک بسته اطلاعات که نقطه پایانی می تواند آن را حمل کند
- نوع ارسال نقطه پایانی
- جهت ارسال اطلاعات بین میزبان و نقطه پایانی

یک نقطه پایانی خاص , با نام نقطه پایانی صفر (Endpoint Zero) وجود دارد که به عنوان پیش فرض در دستگاه ها قبل از تنظیم ها استفاده می شود. در حقیقت این نقطه پایانی یک حالت ناشناخته است و میزبان نمی تواند ازآن استفاده کند. پرسشی که در این قسمت مطرح است این است که چرا از نقطه پایانی صفر استفاده می شود؟

جواب مسئله در یک مشکل مهم مطرح می شودو آن زمانی است که دستگاه USB به یک مقدار پیش فرض برای کنترل پیش فرض نیاز دارد. در این زمان از این نقطه پایانی به صورت دوطرفه استفاده می شود. نرم افزار میزبان نیز از این پیش فرض برای تنظیم های اولیه دستگاه ها استفاده می کند. کنترل پیش فرض یک دستگاه , آن را برای تنظیم ها و دسترسی های بیشتر آماده می کند. به غیر از مواقعی که دستگاه USB تازه به هرم USB متصل شده است از این نقطه پایانی در زمان قطع منبع تغذیه و یا زمان ریست کردن نیز استفاده می شود.

عامل های USB, دارای نقاط پایانی به اضافه تری هستند. عامل هایی که در حالت کاری سرعت پایین کار می کنند, به دو نقطه پایانی تنظیمی محدود شده اند. دستگاه های سرعت بالا , نیز دارای تعدادی نقطه پایانی برای محدود سازی تعاریف پروتکل هستند. نقاط پایانی دیگر برای پردازش پیکربندی دستگاه USB بکار می روند.



شکل ۱-۱۱- ارتباط میزبان با عامل ها توسط انواع مختلف اندپوینت خروجی و ورودی

1-9- انواع لوله ها

در حقیقت لوله, پیوندی بین نقطه پایانی دستگاه و نرم افزارمیزبان است . یک لوله , نمایش دهنده توانایی انتقال اطلاعات بین نرم افزار از طریق حافظه بافر و نقطه پایانی است. در USB دو حالت برای ارتباط انحصاری لوله وجود دارد:

1-9-1- لوله جریان (Stream Pipe)

در این حالت , اطلاعات در ساختاری غیر از USB انتقال می یابند. این نوع از لوله ها دارای جهت یک سویه هستند و توانایی جریان پیدا کردن در یک جهت را دارند. نرم افزار سیستم USB نیازی به آماده سازی و تنظیم همزمانی (Synchronization) و هماهنگی بین چند دستگاه USB را که از یک لوله جریان مشابه استفاده کند, ندارد.زیرا زمانی که جریان اطلاعات در این لوله برقرار می شود , USB یقین دارد که تنها یک دستگاه USB موجوداست. ارایه اطلاعات به لوله جریان , دائما بین first-in و first-out متغییر است. یک لوله جریان به دستگاه USB و به یک شماره نقطه پایانی در یک جهت اختصاصی محدود است . شماره نقطه پایانی یک دستگاه برای انتقال در جهت دیگر باید از لوله جریان دیگری استفاده کند. در این نوع لوله, هرگونه اطلاعاتی, ارسال ودریافت می شوند.

این نوع اطلاعات , دائما در حال انتقال هستندو همچنین همانطور که گفته شد, دارای یک جهت از پیش تعیین شده می باشند. این روش در ارسال هایی از نوع Bulk-Transfer , Interrupt-Transfer , Isochronous-Transfer استفاده می شود.

2-9-1- لوله پیغام (Message Pipe)

این نوع نیز دارای قالب USB است, و بر اساس کنترل کنندگی کامپیوتر عمل می کند. این بدین معنی است که دستگاه توسط اطلاعات در خواستی و از طرف کامپیوتر کنترل و تنظیم می شود, سپس از مسیر در خواستی انتقال می یابد. لوله های پیغام , انتقال اطلاعات را به صورت دو طرفه انجام می دهند, با این حال فقط در Control-Transfer استفاده می شوند.

این نوع از لوله ها در مقابل نقاط پایانی به روش دیگری عمل می کنند. در ابتدا در خواستی از طرف میزبان به دستگاه USB ارسال می شود. این در خواست از جهت تعیین شده تبعیت می کند وپس از آن ,اطلاعات مربوط به وضعیت دستگاه ارسال می شود. در حقیقت این کار در سه مرحله انتقال درخواست (Request) , اطلاعات (Data) و وضعیت (Status) انجام می شود. لوله پیغام , به صورتی در جریان اطلاعات تاثیر می گذارد که فرامین بتوانند انتقال یابند و شناسایی شوند. این لوله اجازه جریان یافتن داده ها را به صورت دو طرفه می دهد. به همین علت است که کنترل کننده لوله, پیش فرض را لوله پیغام قرار داده است .

سیستم USB به صورتی عمل می کند که چندین در خواست همزمان , به لوله پیغام ارسال نشود. یک دستگاه USB فقط در هر لوله پیغام به سرویس دهی یک در خواست نیاز دارد. نرم افزار های مشتری چند گانه (Multiple Software Clients) که در میزبان قرار دارند , ممکن است که در خواست هایی را از لوله پیش فرض ارسال کند. در چنین مواقعی نرم افزار ها در خواست را به صورت first-in و first-out ارسال می کنند. یک دستگاه USB می تواند جریان اطلاعات رادر هر سه مرحله ذکر شده کنترل کند.

تا زمانی که پیغام های قبلی در دستگاه در حال پردازش هستند, از یک لوله پیغام جدید نمی توان استفاده کرد. با این حال زمانی که خطایی پیش آمده باشد , ارسال پیغام جاری متوقف شده و پیام جدید منتقل می شود.

با توجه به مطالب فوق تعیین نوع یک لوله به عوامل زیر بستگی دارد:

- ادعای USB در مورد دسترسی به خط اطلاعات و پهنای باند
- نوع ارسال داده که در بخش بعدی بررسی می شود

- خصوصیات نقطه پایانی مانند جهت, بیشترین اندازه اطلاعاتی که می تواند منتقل شود (Payload)

لوله ها شامل دو نقطه پایانی به همراه یک نقطه پایانی صفر هستند. لوله هایی که شامل نقطه پایانی صفر هستند, به عنوان پیش فرض و در مواقع قطع تغذیه و یا ریست کردن , به کار می روند. سایر لوله ها زمانی توسط سیستم USB استفاده می شوند که دستگاه USB تنظیم شده باشد.

1-10- پروتکل USB

درایور USB پس از نصب , جزیی از سیستم عامل شده و در زمان های منظم سرکشی (Pooling) می کند. هر پاسخ محیطی که دستگاه خارجی به کامپیوتر بدهدواطلاعات را بخواهد ارسال کند , کامپیوتر آن را در کمتر از 1ms دریافت و شناسایی می کند. به همین علت دستگاه هایی که به USB متصل می شوند بلافاصله شناسایی می شوند. کامپیوتر این کار را برای ماکزیمم ۱۲۷ دستگاه می تواند انجام دهدو اطلاعات آن ها را, مالتی پلکس کند.

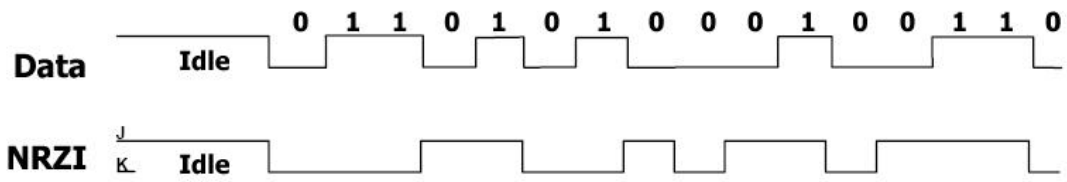
در این پورت اطلاعات در بسته هایی ارسال و دریافت می شوند , که آن ها راتراکنش (Transaction) می نامند و ازچندین فریم اطلاعاتی تشکیل شده اند. این فریم ها با فاصله 1ms از هم جدا می شوند. در USB های با سرعت 1.5 Mb/s هربیت در 667ns ارسال می شود, ولی USB های 12Mb/s هر بیت رادر فاصله 83.3ns ارسال می کنند. پورت USB خود را با فریم Start همزمان می کند. همچنین بافر داخلی USB به طور کلی نیاز دارد تا جریان ارسال و دریافت اطلاعات ثابت باقی بماند.

1-10-1- انتقال اطلاعات

در پروتکل USB دو سیم برای انتقال اطلاعات در نظر گرفته شده است که با نام های D+,D- شناسایی می شوند . این دو اطلاعات را به صورت سریال مقایسه ای انتقال می دهند . به این صورت که دو پایه D+,D- با هم مقایسه می شوند و در صورتی که $V_{D+}>V_{D-}$ مقدار انتقالی یک است و در غیر این صورت عدد موردنظر برابر صفر است .

1-10-2- کد گذاری اطلاعات ارسالی ودریافتی

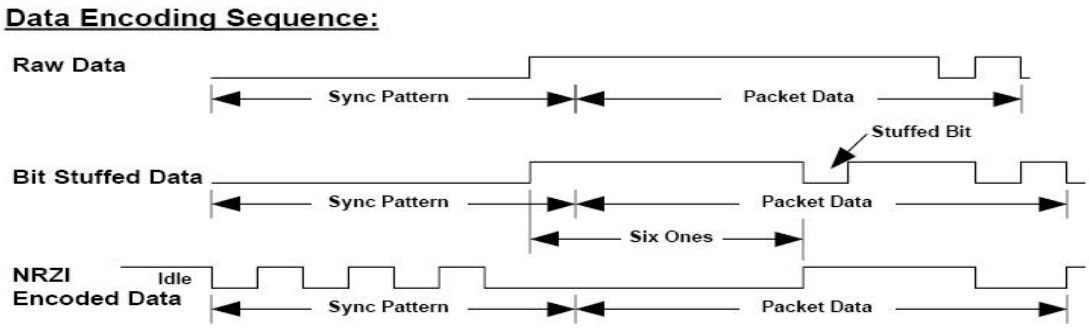
USB از کد گذاری NRZI (Non Return Zero-Inverted) برای انتقال پاکت های اطلاعاتی استفاده می کند. در کد گذاری NRZI , سطح منطقی یک با هیچ تغییری در سطح و سطح منطقی صفر با تغییر در سطح نشان داده می شود. شکل زیر یک جریان اطلا عات و معادل کدگذاری NRZI آن را نشان می دهد.



شکل ۱-۱۲- NRZI Coding

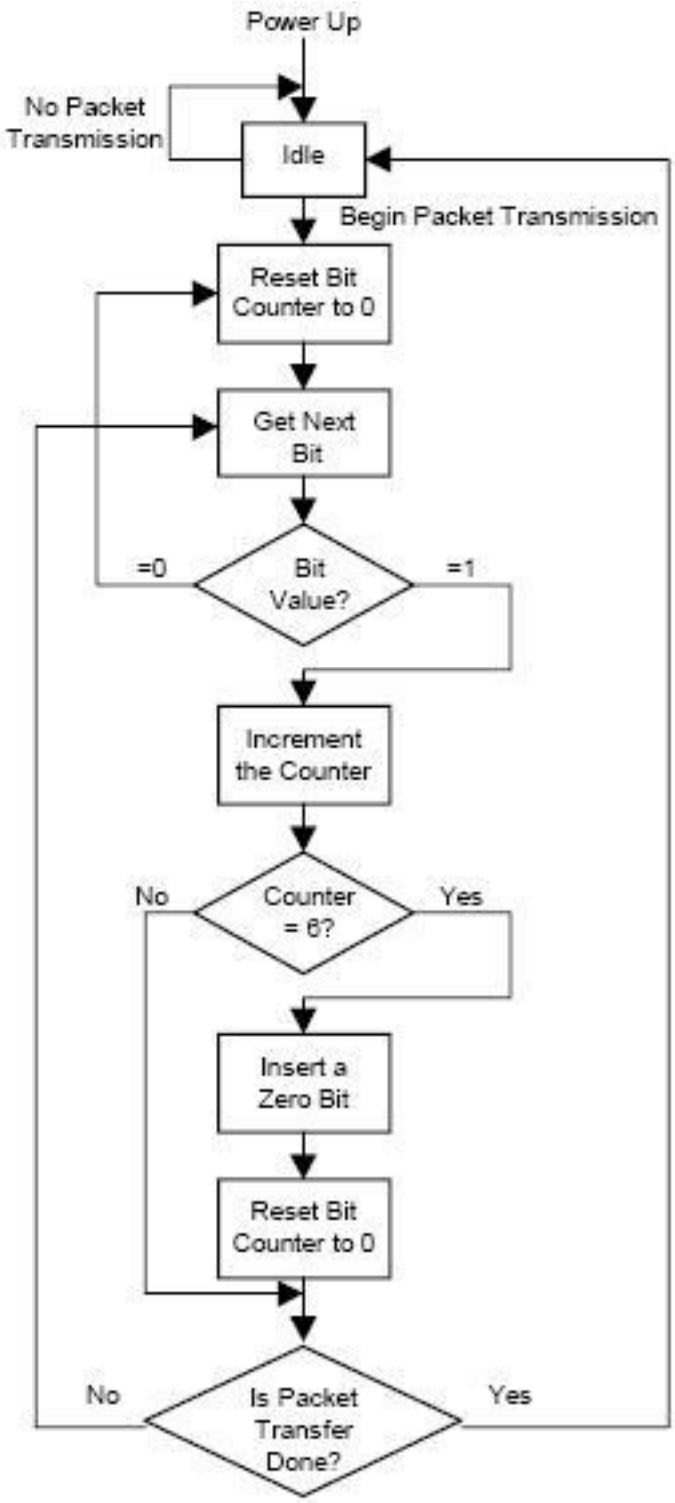
3-10-1- بیت گذاری (Bit Stuffing)

به منظور اطمینان از سلامت داده ارسالی یا دریافتی از این تکنیک استفاده می شود.bit stuffing توسط دستگاه ارسال کننده داده اعمال می شود و به این صورت است که پس از شش یک پشت سر هم یک صفردر اطلاعات خام قبل از کد بندی NRZI قرارمی گیرد این صفر زاید بوده و به منظور تغییر سطح در کد بندی NRZI و اطمینان از صحت اطلاعات و clock lock می باشد.



شکل ۱-۱۳- Bit Stuffing

در طرف گیرنده نیز باید کنترلر دریافت کننده وجود این بیت اضافه را تشخیص داده و آن را از اطلا عات اصلی جدا کند در زیر فلو چارتی را مشاهده می نمایید که چگونگی انجام عمل bit stuffing را روی اطلاعات در طرف فرستنده نشان می دهد.

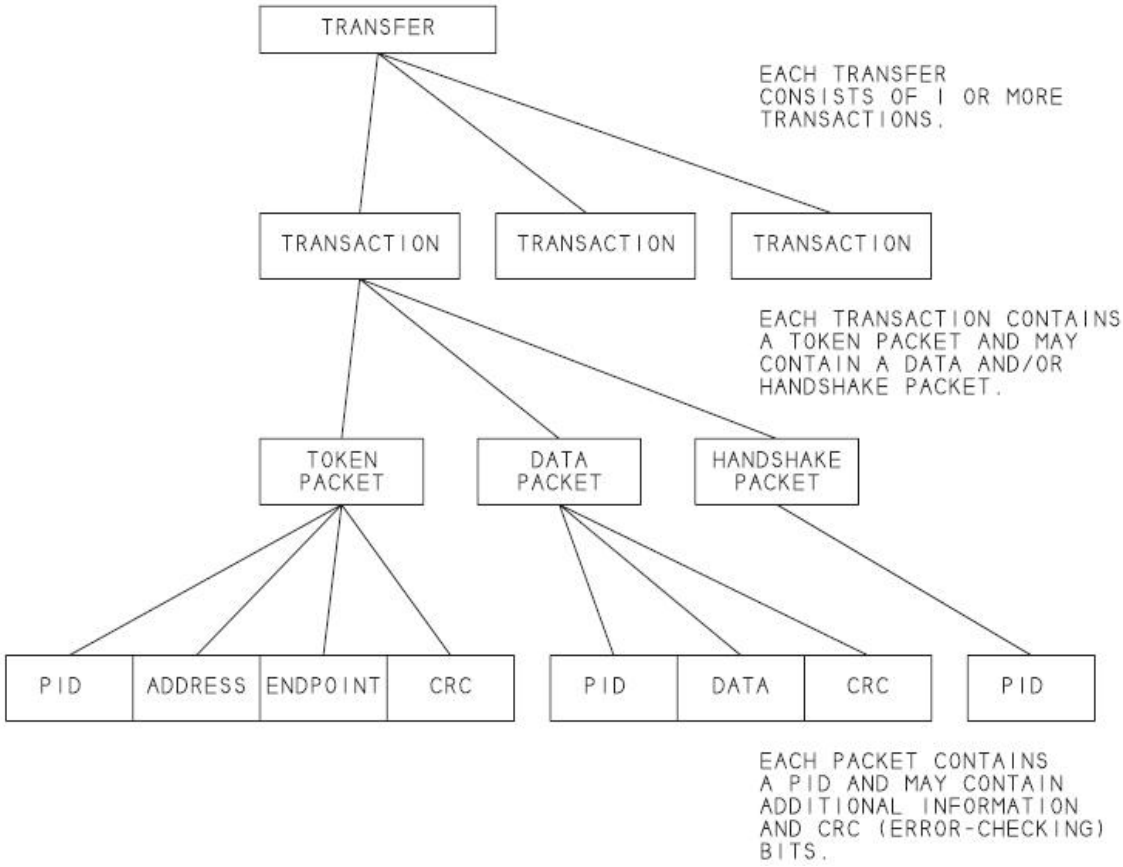


شکل ۱-۱۴- فلوچارت اعمال Bit Stuffing

1-11- اجزاء ترنزکشن

هر ترنزکشن به اجزای کوچکتری به نام بسته (Packet)یا فاز (Phase) تقسیم می شود که به صورت زیر است :

- بسته توکن (Token Packet)
- بسته داده (Data Packet)
- بسته دست دهی (HandShake Packet)



شکل ۱-۱۵- اجزاء ترنزکشن

هر بسته نیز به اجزای کوچکتری به نام فیلد (Field) تقسیم می شود.

میزبان خواسته خود را در بسته توکن به سمت دستگاه ارسال می کند. مشخصه بسته (PID) نوع ترنزکشن را مشخص می کند , که می تواند ورودی , خروجی یا Setup باشد.

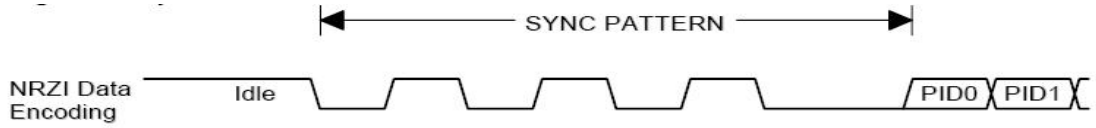
بسته داده نیز اطلاعاتی که باید از میزبان به دستگاه و یا برعکس انتقال یابد رادرخودجای می دهد. پس از ارسال بسته داده , میزبان یادستگاه جانبی , وضعیت خودرا توسط بسته دست دهی به اطلاع می رساند که می تواند یکی از وضعیت های ACK,NAK ,STALL

NYET و یا ERR باشد.

1-11-1- فیلدها در بسته انتقال

1-1-11-1- فیلد sync

هر بسته بااین فیلدشروع می شود, که برای سرعت های پایین به صورت هشت بیتی ودربقیه سرعت ها به صورت ۳۲بیتی است.درزیر معادل کد بندی NRZI این فیلد برای سرعت پایین نشان داده شده است.



شکل ۱-۱۶- فیلد sync

هدف این فیلد سنکرون کردن است . چنانچه فیلد sync دریافت شده از اندازه تعریف شده کمتر باشد خطایی اتفاق نمی افتد.

1-1-11-2- فیلد PID

فیلد مشخصه بسته (PID) به صورت هشت بیتی است . به طوری که بیت های صفر تا سه, نوع بسته را مشخص کرده و بیت های چهار تا هفت مکمل یک بیت های صفر تا سه هستند. که برای بررسی خطا بکار می رود. شکل کلی این فیلد در زیر آمده است .

PID_0	PID_1	PID_2	PID_3	$nPID_0$	$nPID_1$	$nPID_2$	$nPID_3$

شکل ۱-۱۷- فیلد PID

هدف این فیلد تعیین نوع بسته می باشد که مطابق جدول صفحه بعد می باشد.

Group	PID Value	Packet Identifier
Token	0001	OUT Token
	1001	IN Token
	0101	SOF Token
	1101	SETUP Token
Data	0011	DATA0
	1011	DATA1
	0111	DATA2
	1111	MDATA
Handshake	0010	ACK Handshake
	1010	NAK Handshake
	1110	STALL Handshake
	0110	NYET (No Response Yet)
Special	1100	PREamble
	1100	ERR
	1000	Split
	0100	Ping

جدول ۱- ۴ - فیلد PID

1-11-3-1- فیلد ADDR

فیلد آدرس دارای هفت بیت است و آدرس دستگاهی که بسته اطلاعاتی باید به سمت آن ارسال شود رادرخودجای می دهد. در نتیجه تا ۱۲۷ دستگاه را می توان آدرس دهی کرد. البته آدرس صفر , یک آدرس پیش فرض بوده و دستگاهی که هنوز آدرس دهی نشده است باید به آدرس صفر پاسخ دهد.

1-11-1-4- فیلد ENDP

فیلد اندپوینت دارای چهاربیت است که شماره اندپوینتی که ارسال به سمت آن است را تعیین می کند. در دستگاه های سرعت پایین تعداد اندپوینت ها نمی تواند بیشتر از سه عدد باشد زیرا حداکثر سه لوله در یک دستگاه می تواند استفاده شود, که یک لوله برای اندپوینت صفر و دو لوله ی دیگر برای سایر اندپوینت ها درنظرگرفته می شود.

1-11-1-5- فیلد DATA

فیلدداده می تواند اطلاعاتی راازصفر تا ۱۰۲۴ بایت ارسال کند که این مقدار به نوع ارسال و مقدارداده بستگی دارد.

1-11-1-6- فیلد CRC

از فیلد CRC (Cyclic Redundancy Checks) برای بررسی خطا استفاده نموده و به صورت سخت افزاری قابل اجرا است. در حقیقت از CRC ها برای حفاظت از داده های سایر فیلدها استفاده می شود. PID در بسته هایی که از CRC استفاده می کنندکاربردی ندارد. تمام CRC ها توسط فیلد های مربوط به خود تولید شده و قبل از اینکه داده های مربوطه آن به کار روند, ارسال می شود. در صورتی که گیرنده بادریافت CRC خطایی را متوجه شود, بسته مربوطه رادرنظرنمی گیرد.

برای تولیدویاچک کردن CRC , مانند انواع متشابه, از Shift Register استفاده می شود. بدین صورت که بیت ارزش بالای باقیمانده با تمام بیت اطلاعات XOR می شوند, سپس بیت ها به اندازه یک بیت به چپ منتقل شده و بیت های پایین تر صفر می شوند. این کارادامه می یابدودرصورتی که نتیجه XOR برابر یک باشد, بدین معنی است که باقیمانده با جمله تولید کننده کد , XOR, شده است.

زمانی که آخرین بیت فیلد فیلد چک شده ارسال شد, تولید کننده CRC (CRC Generator) معکوس می شود و به سمت چک کننده MSB اول ارسال می شود. زمانی که آخرین بیت CRC توسط چک کننده دریافت شدوهیچ خطایی پیش نیاید , باقیمانده (Remainder) برابر مقدارتولیدی پس ماند است. یک خطای CRC زمانی تشخیص داده می شود , که باقیمانده آخر بسته با مقدار پس ماند مساوی نباشد. درصورتی که تمام شش بیت CRC یک باشد, باید مقدار بیت آخر صفر قرار داده شود.

در دوبسته از CRC استفاده می شود. به همین دلیل CRC دارای دو نوع متفاوت است که درزیربررسی شده اند:

I. **Token CRC** این فیلد شامل پنج بیت است که توسط فیلدهای ADDR و ENDP پوشش داده می شوند. فرمول تولید این CRC به صورت زیر است :

$G(x)=X^5+X^2+1$

الگوی باینری آن به صورت 00101B است واگرتمام بیتهای دریافت شده بدون خطاباشند مقدارپسماند 01100B می شود.

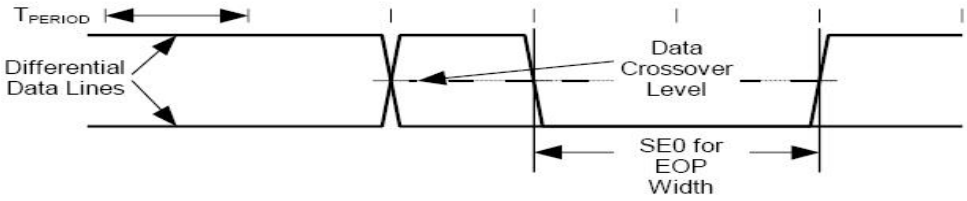
II. **DATA CRC** این CRC دارای ساختمان شانزده بیتی است و در تمام بسته های داده قرار داده می شود. جمله تولید کننده آن به صورت زیر است:

$G(x)=X^{16}+X^{15}+X^2+1$

الگوی باینری این CRC مقدار 1000000000000101B است واگرتمام بیت های دریافتی بدون خطا باشند, مقدار پس ماند باید 10000000000001101B باشد.

1-11-1-7- فیلد EOP (End of Packet)

این رشته مشخص کننده پایان بسته است و توسط سیگنال انتهایی صفر یا SE0 علامت گذاری می شود.



شکل ۱-۱۸- فیلد EOP

درحالت SE0 هر دو خط داده D+,D- به طول حداقل یک بیت تا دو بیت به سطح منطقی صفر می روند.

1-11-2- انواع بسته ها در USB

همان طور که در بخش قبلی گفته شد هرترنزکشن به اجزاء کوچکتری به نام بسته (Packet) تقسیم می شود که در این بخش به بررسی آن ها خواهیم پرداخت .

1-11-2-1- بسته نشانه (Token Packet)

قالب این بسته در زیر نشان داده شده است . بسته های نشانه به سه دسته اصلی زیر تقسیم می شوند:

۱. In به دستگاه جانبی می گوید که کامپیوتر مایل به خواندن اطلاعات از آن می باشد.

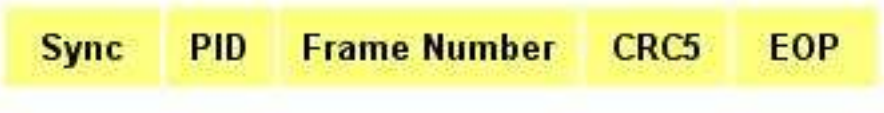
۲. Out به دستگاه جانبی می گوید که کامپیوتر مایل به ارسال اطلاعات به آن است .

۳. Setup برای شروع ارسال کنترلی استفاده می شود.

شکل ۱-۱۹- بسته نشانه

یک بسته نشانه از فیلد های اصلی ENDP,ADDR,PID تشکیل شده است. سایر فیلد ها نیز مانند EOF,Sync در آن گنجانده شده است.برای نوع OUT , Setup فیلدهای ENDP,ADDR منحصربه فردهستندوفقط میزبان می تواند یک بسته نشانه را صادر کند یک بسته نشانه از نوع In یک Data Transaction راازعامل به میزبان تعریف می کند. همچنین PID های Out و Setup , Data Transaction راازمیزبان به عامل تعریف می کنند.

این بسته حاوی یک CRC پنج بیتی است که آدرس و نقطه پایانی را پوشش می دهدوهمچنین نسبت به فیلد PID کاری راانجام نمی دهد, زیرا PID حاوی چهاربیت برای چک کردن اطلاعات خوداست. بسته توکن یک کاربرددیگرنیزدارد. این بسته ممکن است نشانه شروع فریم (SOF) را نیز ارسال کند. این بسته توسط میزبان و به صورت شکل زیر ارسال می شود.



شکل ۱-۲۰- بسته نشا نه شروع فریم

این بسته هر 1ms برای سرعت های بالاوهر 125µs برای سرعت نهایی توسط میزبان فرستاده می شود. دستگاه های سرعت پایین این بسته را دریافت نمی کنند. این بسته شامل شماره فریمی به صورت یازده بیتی است که به صورت شمارنده عمل می کند و تعداد فریم ها را مشخص می کند.صحت وارایه این بسته ضمانت نشده است و اجباری برای ارسال آن نیست .

1-1-2-2- بسته داده (Data Packet)

این بسته اطلاعاتی که قراراست از میزبان به دستگاه و یا بر عکس ارسال شود را منتقل می کندو به صورت شکل زیر است.

شکل ۱-۲۱- بسته داده

نویسنده : حسین

پایمرد

بسته داده بر اساسPID به چهارنوع تقسیم می شود:

۱.DATA0 ۲. DATA1 ۳. DATA2 ۴. MDATA

بیت اتصال داده (DATA10)

درارسال هایی که از چندین ترنزکشن استفاده می نمایید, ممکن است اطلاعات ارسالی به دلایلی از دست برود. برای جلوگیری از چنین حالتی از بیت اتصال داده استفاده می شود. این بیت درون فیلد PID قرار می گیرد.

با توجه به جدول انواع PID ها DATA0 دارای مقدار۰۰۱۱ , DATA1 دارای مقدار ۱۰۱۱ می باشد. به هنگام تبادل اطلاعات بین میزبان و دستگاه , هردوآخرین وضعیت این بیت را ذخیره می کنند. در حالت اولیه بیت DATA0 انتخاب می شود. هنگامی که گیرنده , ترنزکشن اطلاعاتی رادریافت نمود بیت اتصال داده رسیده را با بیت اتصال خود که قبلا ذخیره کرده مقایسه می کند, چنانچه این دوبیت با هم برابر بودند گیرنده بیت اتصال خودراتغییر می دهد. به طور مثال اگر بیت اتصال DATA0 بود به DATA1 تغییر پیدا می کند. پس از این مرحله گیرنده بسته تایید متقابل (ACK) رابرای فرستنده ارسال می کند.

درارسال بعدی , فرستنده بیت اتصال خودرابه DATA1 تغییر می دهد تابا بیت اتصال گیرنده برابر شود.گیرنده نیز پس از هر بار دریافت بیت اتصال و در صورت برابری با بیت اتصال خود, بسته تاییدمتقابل (ACK) ودرصورت مشغول بودن پاسخ (NAK) رابر می گرداند.

این روند تا هنگامی که تمام اطلاعات ارسال شود ادامه پیدا می کند.با توجه به مطلب فوق چنانچه به هنگام ارسال , اطلاعاتی همراه با خطا دریافت شد , گیرنده در پاسخ هیچ مقداری را نمی فرستد و بیت اتصال خود را تغییر نمی دهد.

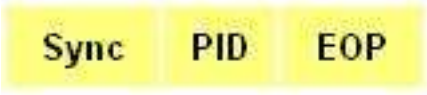
همزمان سازی بین فرستنده و گیرنده

چنانچه گیرنده بسته تایید متقابل (ACK) رابفرستد امافرستنده به دلیلی آن را نبیند, فرستنده دوباره اطلاعات را با همان بیت اتصال قبلی ارسال می کند, دراین حالت به دلیل آنکه گیرنده بیت اتصال خود را تغییر داده است , از داده رسیده صرف نظر کرده و بسته تایید متقابل را برای فرستنده ارسال می کند . این عمل باعث همزمان سازی بین فرستنده و گیرنده خواهد شد.

در ارسال های همزمان و سرعت های خیلی بالا از PID های MDATA,DATA2 نیز استفاده می شود.

1-1-2-11-3- بسته دست تکانی (Handshake Packet)

برای آنکه مطمئن شویم که همه ارسال ها به درستی انجام شده اند , USB از بسته دست تکانی استفاده می کند . این بسته از سه فیلد تشکیل شده است.



شکل ۱-۲۲- بسته دست تکانی

از این بسته جهت گزارش وضعیت ارسال داده استفاده می شود , که به طورکلی شش نوع خواهد بود.

۱. **ACK** تایید کننده (ACK) نشان می دهد که میزبان یادستگاه, اطلاعات را بدون خطا دریافت نموده است .

۲. **NAK** عدم تایید (NAK) نشان دهنده این است که دستگاه مشغول بوده و آماده دریافت نیست. اگر میزبان داده ای را در زمانی که دستگاه مشغول است بفرستد, دستگاه در بسته دست دهی , NAK رابرخواهد گرداند.میزبان هرگز نمی تواند NAK را بفرستد.

۳. **STALL** توسط دستگاه بر می گردد و نشان می دهد که خواسته کنترلی توسط دستگاه پشتیبانی نمی شود ویاخواسته اشتباه است . همچنین ممکن است دستگاه قادر به دریافت و ارسال اطلاعات نباشد که در این صورت STALL را به میزبان باز می گرداند.

چنانچه دستگاه به دلایلی نتواند از خواسته میزبان پشتیبانی کند STALL راباز می گرداند , که در اصطلاح به آن Protocol Stall می گویند.

همچنین اگر هنگام پاسخ به خواسته ای , خاصیت قطع (Halt) در اندپوینت فعال باشد, در این حالت به STALL بازگشتی از طرف دستگاه در اصطلاح Function Stall می گویند.

میزبان در هیچ شرایطی نمی تواند STALL را بر گرداند.

۴. **NYET** این دست تکانی تنها در سرعت های خیلی بالا استفاده می شود و نشان می دهد که دستگاه, بسته داده را بدون خطا دریافت نموده , اما هنوز برای دریافت بسته بعدی آمادگی ندارد. همچنین از آن برای جداسازی ترنزکشن ها در هاب ها نیز استفاده می شود.

۵. **ERR** دست تکانی ERR تنها در سرعت های خیلی بالا کاربرد داردوتوسط هاب ارسال می شودکه نشان دهنده تولید یک خطا است. همچنین از آن برای جداسازی ترنزکشن ها در هاب نیز استفاده می گردد.

۶. **بدون پاسخ** این نوع دست تکانی هنگامی به کار می رود که میزبان یا دستگاه در ارسال یادریافت دچار خطاشده است , که در این حالت ارسال دوباره باید صورت بگیرد.

12-1- بررسی پاسخهای دستگاه ومیزبان درمرحله دست تکانی(Handshake Responses)

در این بخش سعی داریم تا با استفاده از جدول ها و براساس مرجع خصوصیات USB 1.1 مروری کوتاه بر رفتارهای میزبان(Host PC) ودستگاه جانبی (Function) در مرحله دست تکانی (Handshaking) داشته باشیم.

جواب دستگاه به ترنزکشن ورودی

جدول زیرپاسخ های ممکن یک دستگاه جانبی به یک بسته نشانه ورودی را نشان می دهد.

Token Received Corrupted	Function Tx Endpoint Halt Feature	Function Can Transmit Data	Action Taken
Yes	Don't care	Don't care	Return no response
No	Set	Don't care	Issue STALL handshake
No	Not set	No	Issue NAK handshake
No	Not set	Yes	Issue data packet

جدول ۱-۵-پاسخ های ممکن عامل به بسته نشانه ورودی

درزیر جواب میزبان را به دیتا ورودی از دستگاه درقالب بسته دست دهی در حالت های مختلف رامی بینید.

Data Packet Corrupted	Host Can Accept Data	Handshake Returned by Host
Yes	N/A	Discard data, return no response
No	No	Discard data, return no response
No	Yes	Accept data, issue ACK

جدول ۱-۶- پاسخ های ممکن میزبان به داده ورودی

در زیرجواب های ممکن دستگاه به میزبان پس ازدریافت دیتاآورده شده است.

Data Packet Corrupted	Receiver Halt Feature	Sequence Bits Match	Function Can Accept Data	Handshake Returned by Function
Yes	N/A	N/A	N/A	None
No	Set	N/A	N/A	STALL
No	Not set	No	N/A	ACK
No	Not set	Yes	Yes	ACK
No	Not set	Yes	No	NAK

جدول ۱-۷-پاسخ های ممکن عامل پس از دریافت داده

درضمن دستگاه در هنگام دریافت بسته نشانه Setup همیشه ودرهرحالت باید ACK را در پاسخ بفرستد.

13-1- انواع ارسال درUSB

به طورکلی درUSB چهار نوع روش انتقال اطلاعات وجوددارد:

۱. ارسال کنترلی (Control Transfer)
۲. ارسال وقفه ای (Interrupt Transfer)
۳. ارسال توده ای (Bulk Transfer)
۴. ارسال همزمان (Isochronous Transfer)

در شکل صفحه بعد خصوصیات اصلی هر نوع ارسال را می بینید.

Transfer Type	Control	Bulk	Interrupt	Isochronous
Typical Use	Identification and configuration	Printer, scanner, drive	Mouse, keyboard	Streaming audio, video
Required?	yes	no	no	no
Low speed allowed?	yes	no	yes	no
Data bytes/millisecond per transfer, maximum possible per pipe (high speed).*	15,872 (thirty-one 64-byte transactions/ microframe)	53,248 (thirteen 512-byte transactions/ microframe)	24,576 (three 1024-byte transactions/ microframe)	24,576 (three 1024-byte transactions/ microframe)
Data bytes/millisecond per transfer, maximum possible per pipe (full speed).*	832 (thirteen 64-byte transactions/ frame)	1216 (nineteen 64-byte transactions/ frame)	64 (one 64-byte transaction/ frame)	1023 (one 1023-byte transaction/ frame)
Data bytes/millisecond per transfer, maximum possible per pipe (low speed).*	24 (three 8-byte transactions)	not allowed	0.8 (8 bytes per 10 milliseconds)	not allowed
Direction of data flow	IN and OUT	IN or OUT	IN or OUT (USB 1.0 sup-ports IN only)	IN or OUT
Reserved bandwidth for all transfers of the type (percent)	10 at low/full speed, 20 at high speed (minimum)	none	90 at low/full speed, 80 at high speed (isochronous & interrupt combined, maximum)	
Error correction?	yes	yes	yes	no
Message or Stream data?	message	stream	stream	stream
Guaranteed delivery rate?	no	no	no	yes
Guaranteed latency (maximum time between transfers)?	no	no	yes	yes
*Assumes transfers use maximum packet size.				

جدول ۱-۸- خصوصیات انواع مختلف ارسال در USB

هر یک از انواع ارسال (Transfer) دارای خصوصیات مختلف برای ایجاد جریان اطلاعات هستند که در زیر ذکر شده اند:

- قالب داده هایی که USB می خواهد, آن ها را جابجا کند.
- جهت جریان اطلاعات

- اندازه بسته هایی که داده ها را جابجا می کنند
- دسترسی به خط اطلاعات
- مرحله درخواست اطلاعات
- برخورد با خطا های پیش آمده

طراحان دستگاه USB تواناییها و قابلیت های نقطه پایانی دستگاه را انتخاب می کنند. زمانی که لوله بر اساس نقطه پایانی ساخته می شود , در اغلب خصوصیات انواع ارسال , لوله تعریف شده و تا آخر باقی می ماند. درشکل زیر ماکزیمم نرخ تبادل داده هر اندپوینت را برحسب مقدارpayload مجاز در هر نوع ارسال می بینید.

Transfer Type	Maximum data-transfer rate per endpoint (kilobytes/sec. with data payload/transfer = maximum packet size allowed for the speed)		
	Low Speed	Full Speed	High Speed
Control	24	832	15,872
Interrupt	0.8	64	24,576
Bulk	not allowed	1216	53,248
Isochronous		1023	24,576

جدول ۱-۹- ماکزیمم نرخ تبادل داده هر اندپوینت در هرنوع ارسال

1-13-1- ارسال کنترلی

این نوع ارسال, در سیستم USB دارای وظایف خاصی است . به طوری که به کمک ارسال میزبان می تواند دستگاه جانبی را پیکربندی کندواطلاعات موردنیاز دستگاه رادریافت نماید. این نوع ارسال تنها نوع ارسالی است که همه دستگاه های جانبی باید آن را پشتیبانی کنند.

در واقع هر دستگاهی باید ارسال کنترلی را از طریق لوله پیش فرض (لوله پیغام) و اندپوینت صفر(نقطه پایانی به شماره صفر که برای این نوع ارسال در نظر گرفته شده است و قابل تغییر نیست) پشتیبانی کند. یک دستگاه ممکن است دارای لوله های بیشتری برای ارسال کنترلی باشد ولی در واقع نیازی به آنها نیست , حتی اگر دستگاه نیاز به ارسال مقدارزیادی درخواست های کنترلی باشد, زیرا میزبان ها پهنای باند لازم را برحسب سایزوشماره درخواست ها در اختیارتبادل کنترلی قرار می دهندو نه بر حسب شماره نقطه پایانی کنترلی , در نتیجه تعداد بیشتر اندپوینت کنترلی مزیتی ندارد .

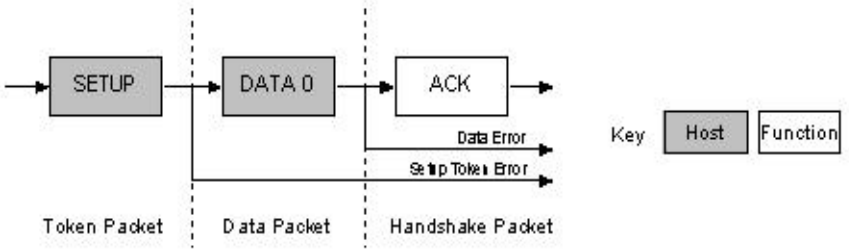
ارسال کنترلی دارای دو یا سه ترنزکشن است :

- **ترنزکشن Setup**
- **ترنزکشن داده**
- **ترنزکشن وضعیت**

از بین این سه مرحله, ترنزکشن Setup و وضعیت, اجباری و ترنزکشن داده اختیاری است. از آن جا که ارسال کنترلی تنهاازطریق اندپوینت صفر (اندپوینت کنترلی) انجام می شودو همه دستگاه ها نیز حداقل دارای این اندپوینت می باشند, بنابراین ازطریق ارسال کنترلی و ترنزکشن داده می توان اطلاعات موردنیاز رانیز منتقل نمود, امادر صورتی که دستگاه دارای اندپوینت های دیگری به غیر از اندپوینت صفربود می توان اطلاعات رااز طریق آن ها منتقل نمود. به همین دلیل ترنزکشن داده درارسال کنترلی اختیاری است. همان طور که قبلا نیز اشاره شد, ارسال کنترلی تنهاارسالی است که از لوله پیغام استفاده می کند و مابقی ارسال ها از طریق لوله جریان منتقل می شوند.

1-13-1-1- ترنزکشن Setup

دراین مرحله میزبان با فرستادن خواسته و اطلاعات مورد نیاز , دستگاه را پیکربندی می کند. تمامی دستگاه ها باید به این مرحله پاسخ دهند. شکل زیرترنزکشن Setup در ارسال کنترلی را نشان می دهد.



شکل ۱-۲۳-ترنزکشن Setup در ارسال کنترلی

با توجه به شکل زیر مشخص می شود که میزبان باارسال بسته نشانه که ازنوع ترنزکشن Setup می باشدارسال را آغاز می کند. دربسته نشانه , اطلاعات مربوط به آدرس دستگاه و اندپوینت وجوددارد و CRC آن نیز از نوع پنج بیتی است.

1. Setup Token	Sync	PID	ADDR	ENDP	CRC5	EOP	Address & Endpoint Number
2. Data0 Packet	Sync	PID	Data0		CRC16	EOP	Device Descriptor Request
3. Ack Handshake	Sync	PID	EOP				Device Ack. Setup Packet

شکل ۱-۲۴- آغاز ارسال کنترلی توسط میزبان

نویسنده : حسین

پایمرد

در مرحله بعد, میزبان بسته داده راارسال می کند , که به وسیله آن یک خواسته و اطلاعات مربوط به آن ارسال می شود. CRC آن از نوع شانزده بیتی است . در بسته داده مشخصه بسته DATA0 است و هشت بایت اطلاعات در پنج فیلد به صورت شکل زیر ارسال می شود.

bmRequest Type	bRequest	wValue	wIndex	wLength

شکل ۱-۲۵- بسته داده

- **bmRequest Type**

این فیلد به صورت هشت بیتی می باشد و خصوصیات یک خواسته معین را مشخص می کند.

بویژه این فیلد جهت تبادل داده را در فاز دوم تبادل داده در ارسال کنترلی مشخص می کند. ولی در صورتی که فیلد wLength برابر صفر باشد از حالت بیت جهت در این فیلد صرف نظر می شود زیرا این نشان دهنده عدم وجود دیتا است.

مرجع خصوصیات USB یک سری از خواسته های استاندارد را مشخص نموده که همه دستگاه های جانبی باید آن ها را پشتیبانی کنند که در بخش های بعدی توضیح داده خواهند شد .یک فروشنده دستگاه جانبی همچنین خواسته های مخصوص پشتیبانی شده توسط دستگاهش(Vendor Requests) را مشخص می کند.

این فیلد همچنین جزء گیرنده درخواست را نیز مشخص می کند, گیرنده در خواست ممکن است دستگاه , یک واسط روی دستگاه , یا یک نقطه پایانی مشخص دستگاه و یا اجزاء دیگر دستگاه باشد.

هنگامی که یک اندپوینت و یایک مدار واسط به عنوان گیرنده درخواست معین شوند , فیلد wIndex اندپوینت و یا مدار واسط مقصد را معین می کند.

- **bRequest**

این فیلد به صورت هشت بیتی است و خواسته را مشخص می کند. در صورتی که در فیلد bmRequest Type بیت های پنج و شش به صورت ۰۰ تنظیم شده باشند, فیلد bRequest باید شامل یکی از یازده خواسته استاندارد USB باشد. چنانچه بیت های پنج و شش به صورت ۰۱ تنظیم شود, فیلد bmRequest باید شامل خواسته ای که برای کلاس دستگاه تعریف شده است , باشدو اگر بیت ها روی ۱۰ تنظیم شوند bmRequest یک خواسته مشخص شده توسط فروشنده است .

- **wValue**

این فیلد به صورت شانزده بیتی است که از آن برای ارسال داده به دستگاه و یاارسال آدرس دستگاه استفاده می نماییم.

- **wIndex**

این فیلد شانزده بیتی بوده و با توجه به نوع خواسته کاربرد های مختلفی دارد . اما بیشترین کاربرد آن برای مشخص کردن اندپوینت و واسط است .

در صورتی که از آن برای مشخص کردن اندپوینت استفاده شود, بیت های صفر تا سه , شماره اندپوینت را مشخص می کند و بیت هفتم جهت را مشخص می کند که صفر به صورت خروجی و یک به صورت ورودی قابل تعیین است .جهت ورودی به سمت میزبان و جهت خروجی به سمت دستگاه جانبی است .در صورتی که از این فیلد برای مشخص کردن واسط استفاده شود , بیت های صفرتاهفت شماره واسط را مشخص می کند.

- **wLength**

این فیلد به صورت شانزده بیتی می باشدو طول اطلاعات ارسالی درترنزکشن داده را مشخص می کند. جهت ارسال نیز توسط بیت هفتم در فیلد bmRequest Type تعیین می شود.

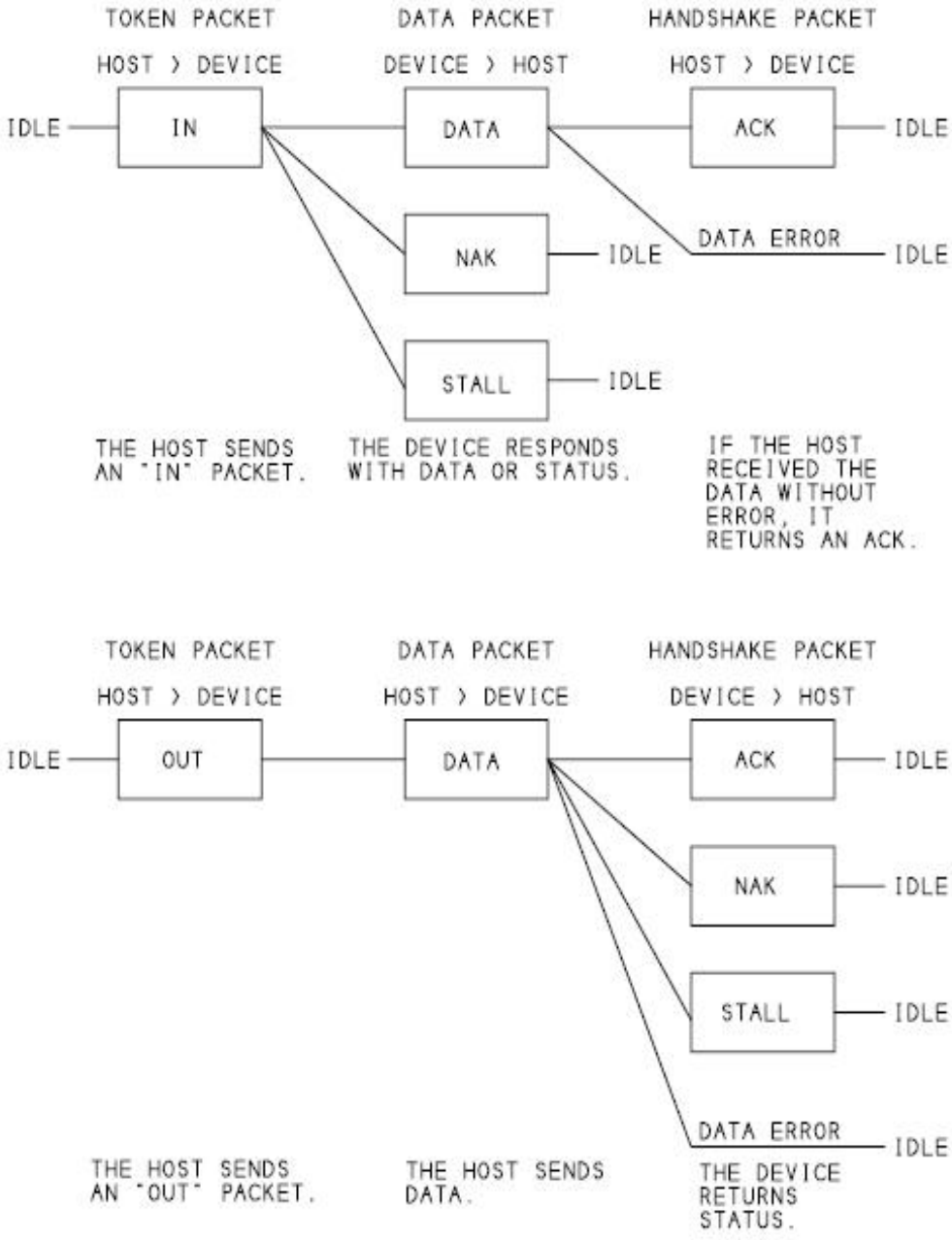
Offset	Field	Size	Value	Description
0	bmRequestType	1	Bit-Map	**D7 Data Phase Transfer Direction** 0 = Host to Device 1 = Device to Host **D6..5 Type** 0 = Standard 1 = Class 2 = Vendor 3 = Reserved **D4..0 Recipient** 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4..31 = Reserved
1	bRequest	1	Value	Request
2	wValue	2	Value	Value
4	wIndex	2	Index or Offset	Index
6	wLength	2	Count	Number of bytes to transfer if there is a data phase

جدول ۱-۱۰- بسته داده

میزبان پس از ارسال بسته داده منتظر دریافت بسته دست دهی از طرف دستگاه می ماند.

1-13-1-2- ترنزکشن داده

همان طور که قبلا اشاره شد ارسال کنترلی می تواند شامل ترنزکشن داده باشد. اطلاعات ارسالی در جهتی که به وسیله ترنزکشن Setup تعیین شده است , منتقل می شود. شکل زیر ترنزکشن داده در ارسال کنترلی را نشان می دهد.



شکل ۱-۲۶- ترنزکشن داده

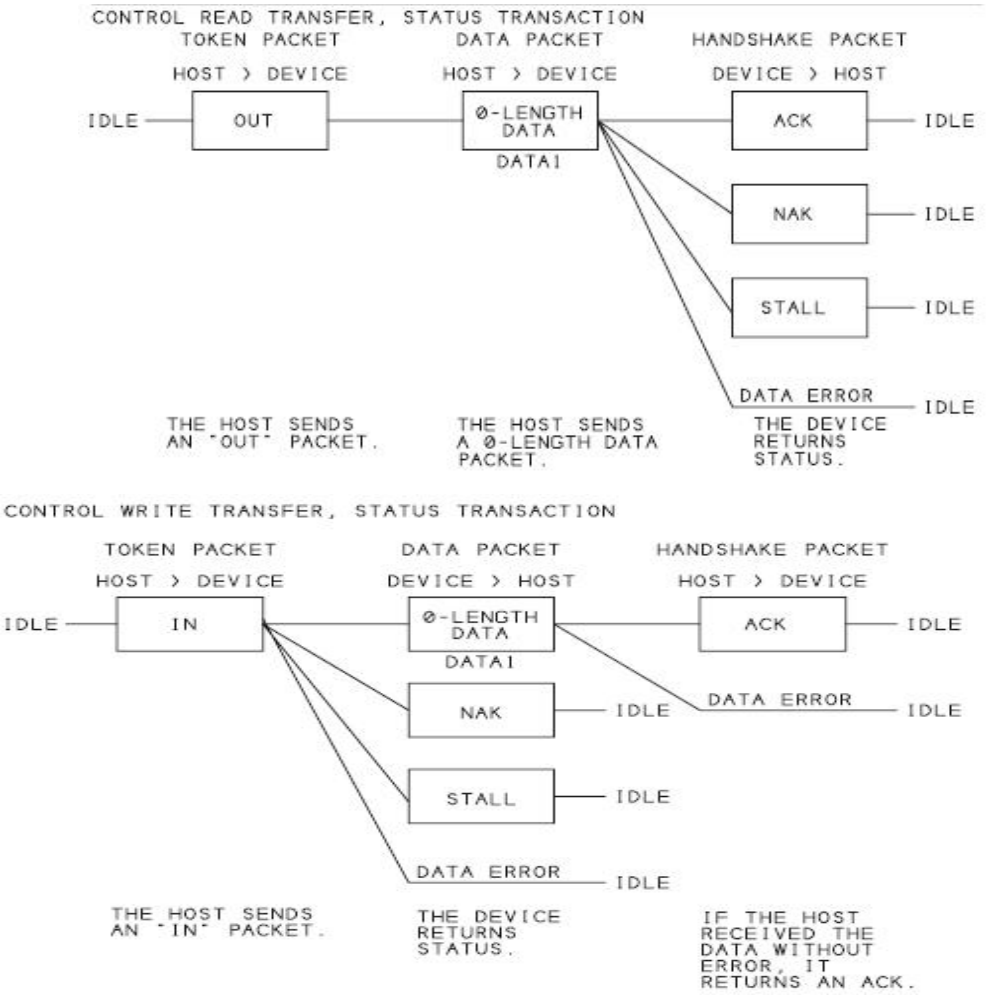
با توجه به شکل صفحه بعد هر ترنزکشن داده از نوع ورودی یا خروجی, شامل بسته های نشانه (ورودی, خروجی) , بسته داده و دست دهی است . چنانچه طول داده ها که در فیلد wLength در ترنزکشن Setup مشخص شده است از طول حداکثر اندازه اندپوینت بیشتر باشد , داده ها را در چندترنزکشن ارسال می کنند. به طور مثال چنانچه حداکثر طول اندپوینت موردنظر هشت بیت باشد و طول داده ارسالی دوازده بیت باشد ارسال به دو ترنزکشن نیاز دارد که یکی به طول هشت بیت و دیگری به طول چهاربیت ارسال خواهد شد. چنانچه در شکل زیر نیز مشخص است در اولین بسته داده , مشخصه بسته DATA1 است و در بسته داده بعدی به طور متناوب DATA0و DATA1 خواهد شد.

1. In Token	Sync	PID	ADDR	ENDP	CRC5	EOP	Address & Endpoint Number
2. Data1 Packet	Sync	PID	Data1		CRC16	EOP	First 8 Bytes of Device Descriptor
3. Ack Handshake	Sync	PID	EOP				Host Acknowledges Packet
1. In Token	Sync	PID	ADDR	ENDP	CRC5	EOP	Address & Endpoint Number
2. Data0 Packet	Sync	PID	Data0		CRC16	EOP	Last 4 bytes + Padding
3. Ack Handshake	Sync	PID	EOP				Host Acknowledges Packet

شکل ۱-۲۷- ارسال بسته داده

1-13-1-3- ترنزکشن وضعیت

پس از ارسال ترنزکشن های Setup وداده, ترنزکشن وضعیت , اطلاعات مربوط به وضعیت دستگاه را در مورد درستی یا نادرستی درمرحله قبل به اطلاع میزبان می رساند . شکل زیر ترنزکشن وضعیت در ارسال کنترلی را نشان می دهد.



شکل ۱-۲۸- ترنزکشن وضعیت

با توجه به شکل زیر ترنزکشن وضعیت نیز مانند دو ترنزکشن Setup و داده شامل بسته های نشانه (ورودی یا خروجی) , بسته داده و دست دهی می باشد. در بسته نشانه آدرس دستگاه, شماره اندپوینت و جهت بسته داده در ترنزکشن وضعیت مشخص می شود. باید توجه داشت جهت انتقال بسته داده در ترنزکشن وضعیت مخالف ترنزکشن داده است .

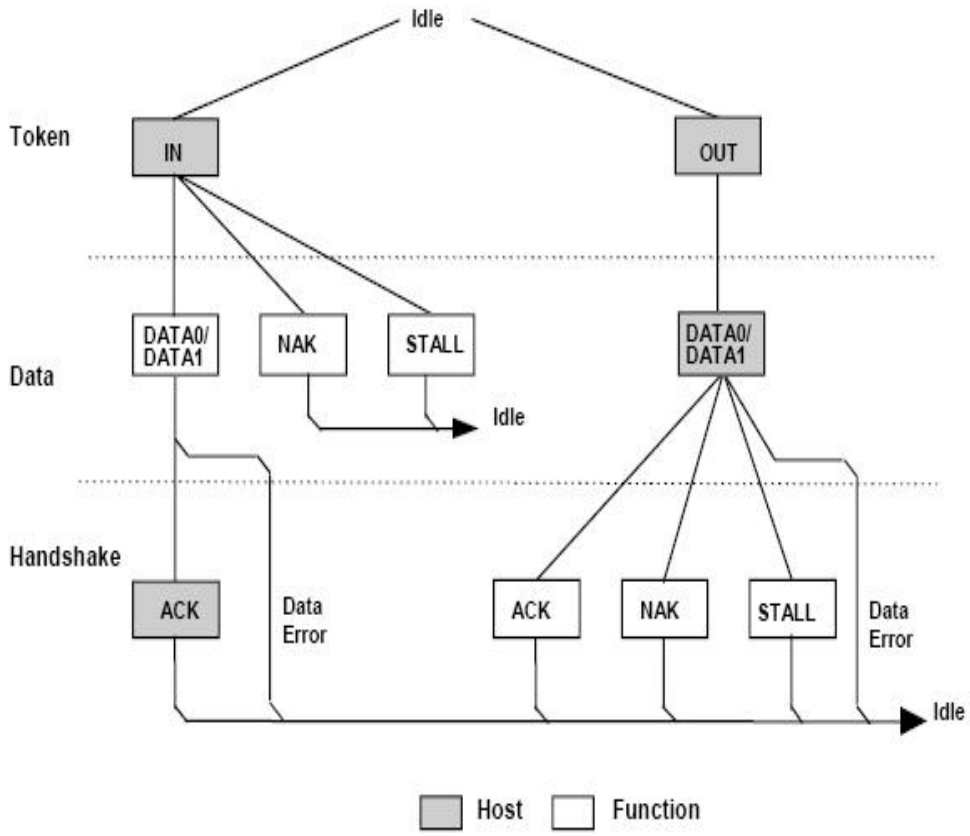
1. Out Token	Sync	PID	ADDR	ENDP	CRC5	EOP	Address & Endpoint Number
2. Data1 Packet	Sync	PID	Data1		CRC16	EOP	Zero Length Packet
3. Ack Handshake	Sync	PID	EOP				Device Ack. Entire Transaction

شکل ۱-۲۹- ارسال بسته وضعیت

نکته: بسته داده در ترنزکشن وضعیت دارای مشخصه DATA1و طول داده صفر است .

1-13-2- ارسال توده ای

این نوع ارسال معمولادر جا هایی استفاده می شود که بخواهیم اطلاعاتی با حجم بالا رادرزمان های مستقل ارسال کنیم .به طور مثال در پرینترواسکنر, این روش ارسال تنهادر دستگاه های سرعت بالا و سرعت نهایی به کار می رود. ارسال توده ای تشکیل شده است از چند ترنزکشن که از نوع ورودی یا خروجی می باشد. ارسال توده ای از لوله جریان و به صورت یک طرفه استفاده می کند.



شکل ۱-۳۰ - ارسال توده ای

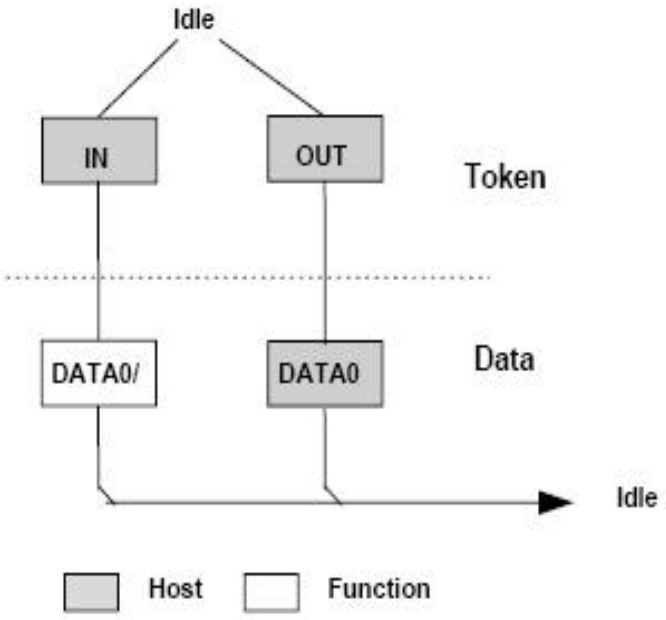
ارسال توده ای امکانات زیر را برای دستگاه درخواست دهنده , ایجاد می کند.

- دسترسی به خطوط اطلاعات در پهنای باند
- سعی دوباره برای دریافت بسته اطلاعات در هنگام بروز خطا
- تحویل اطلاعات بدون ضمانتی برای پهنای باند

در سرعت های بالا حداکثر طول داده می تواند به اندازه 32,16,8 یا ۶۴ بایت باشد و در سرعت نهایی حداکثر ۵۱۲ بایت است. باید توجه داشت که دستگاه های سرعت پایین قادر به استفاده ازارسال توده ای نیستند.

1-13-3- ارسال همزمان

این روش برای ارسال اطلاعات با حجم بالا استفاده می شود. در این روش زمان و سرعت ضمانت شده ولی خطایابی وجود ندارد و معمولا برای ارسال فایل های صوتی از آن بهره می گیرند. تنها دستگاه های سرعت بالا و سرعت نهایی می توانند از این ارسال استفاده کنند.ارسال همزمان شامل یک ترنزکشن ورودی یا خروجی است و از لوله جریان به صورت یک طرفه استفاده می کند. حداکثر اندازه بسته برای سرعت بالا برابر ۱۰۲۳ بایت و در سرعت های خیلی بالا حداکثر تا ۱۰۲۴ بایت خواهد بود. شکل زیر ارسال همزمان را نشان می دهد.



شکل ۱-۳۱- ارسال همزمان

درمحیط هایی که از ساختار USB پیروی نمی کنند, ارسال همزمان در سرعت ثابت می تواند به همراه خطایابی انجام گیرد. ولی در محیط هایی که بر اساس قوانینUSB کار می کنند, درخواست راارسال همزمان درخواست دهنده در شرایط زیر آماده می کند:

- ضمانت برای دسترسی به پهنای باند USB دردوره عکس العمل محدودومعین.
- ضمانت سرعت ثابت انتقال اطلاعات در میان لوله تا مدتی که داده ها در آن آماده می شوند.
- در هنگام پیش آمد خطا , بسته های اطلاعاتی استفاده نمی شوند.

ارسال همزمان از هیچ ساختار خاصی برای جریان اطلاعات پیروی نمی کند.

1-13-4- ارسال وقفه ای

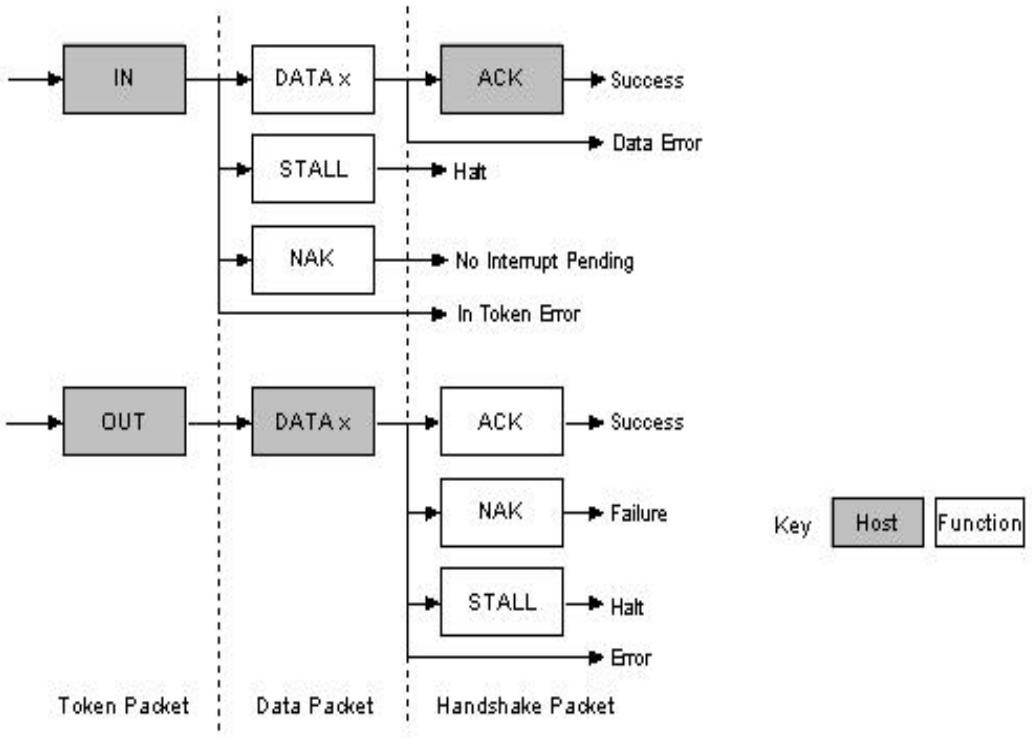
از این روش ارسال معمولا توسط دستگاه های با سرعت پایین مانند ماوس و صفحه کلید استفاده می شود, زیرا این دستگاه ها اطلاعات کمی را با سرعت زیاد دریافت و ارسال می کنند.

ارسال وقفه ای امکانات زیر را برای درخواست دهنده اش مهیا می کند:

- ضمانت بیشترین دوره سرویس برای لوله
- تلاش دوباره برای ارسال , درزمان پیش آمد خطا

این نوع ارسال از ساختار خاصی برای داده ها استفاده نمی کند و کاملا یک جهته است , زیرا از لوله جریان استفاده می کند. برای انتقال دو طرفه باید از دو ارسال وقفه ای استفاده کرد.

شکل زیر ارسال وقفه ای را نشان می دهد.



شکل ۱-۳۲- ارسال وقفه ای

اندازه بسته های اطلاعات در این نوع ارسال نیز مانند سایر آن ها , توسط نقاط پایانی مشخص می شوند. این اندازه ها در سرعت بالا ۶۴ بایت ودر سرعت نهایی ۱۰۲۴ بایت است . تعداد بایت های یک بسته در سرعت پایین نیز نباید از ۸ بایت تجاوز کند.

جدول زیرترنزکشن ها و پاکت های مورد نیاز در هر نوع ارسال رانشان می دهد.

Transfer Type		Transactions	Phases (packets). Each downstream, low-speed packet is also preceded by a PRE packet.
Control	Setup Stage	One transaction	Token
			Data
			Handshake
	Data Stage	Zero or more transactions (IN or OUT)	Token
			Data
			Handshake
	Status Stage	One transaction (opposite direction of transaction(s) in the Data stage or IN if there is no Data stage)	Token
			Data
			Handshake
Bulk		One or more transactions (IN or OUT)	Token
			Data
			Handshake
Interrupt		One or more transactions (IN or OUT)	Token
			Data
			Handshake
Isochronous		One or more transactions (IN or OUT)	Token
			Data

جدول ۱-۱۱- ترنزکشن ها و پاکت ها در هرنوع ارسا ل

14-1- توصیف گرها

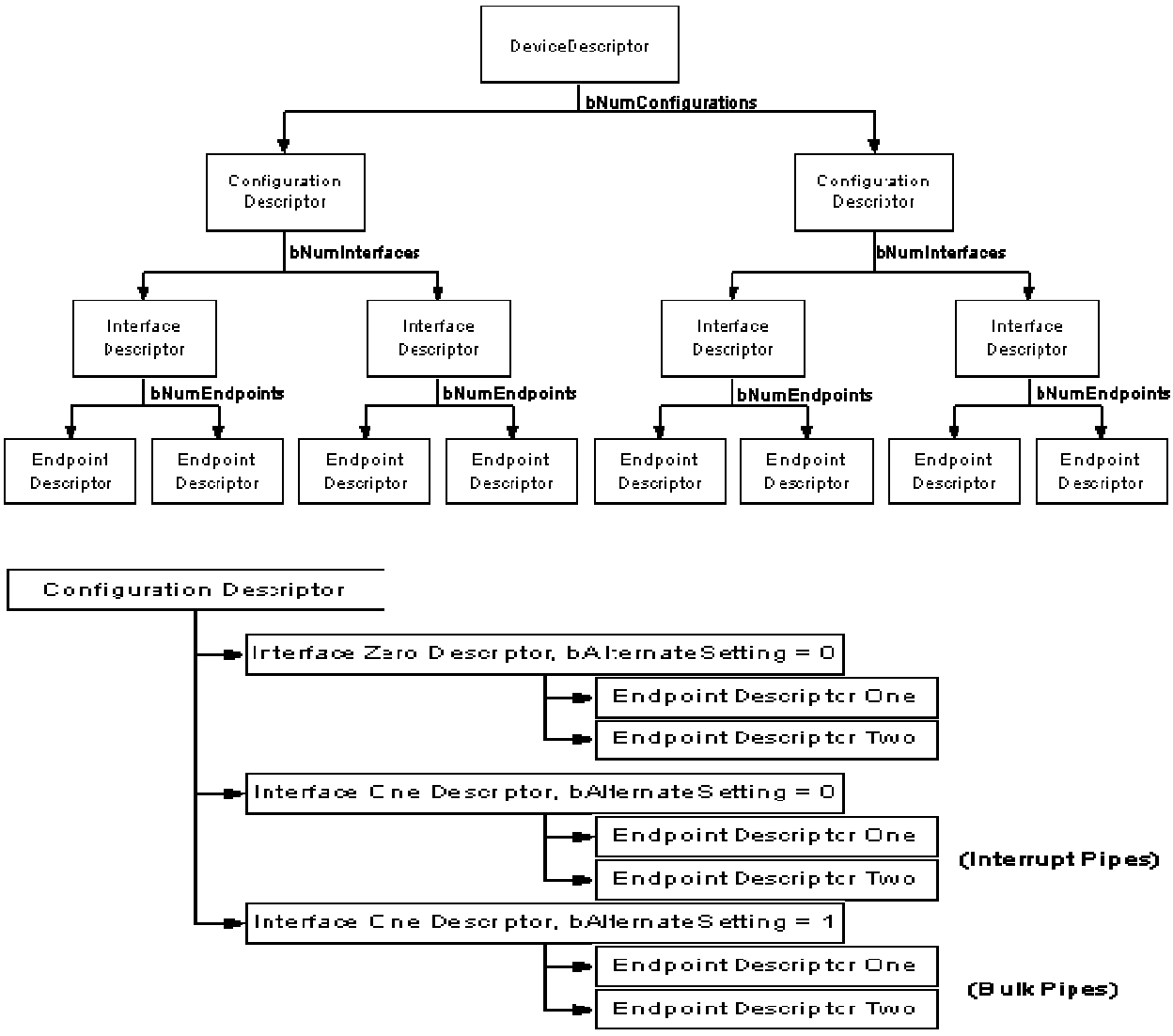
1-14-1- انواع توصیف گر ها در USB

توصیف گرها , اطلاعاتی با قالب بندی مشخص می باشند که میزبان را قادر به شناختن خصوصیات دستگاه می کنند. از جمله آن اطلاعات می توان به نام شرکت سازنده , تعداد روش های ارسال , تعداد اندپوینت و غیره اشاره نمود. هردستگاه USB باید دارای یک توصیف گر باشد و به خواسته های میزبان پاسخ دهد. انواع توصیف گرها در USB به صورت زیر است:

- توصیف گر دستگاه (Device Descriptor)
- توصیف گر پیکربندی (Configuration Descriptor)

- توصیف گر واسط (Interface Descriptor)
- توصیف گر اندپوینت (Endpoint Descriptor)
- توصیف گررشته (String Descriptor)
- توصیف گر کلاس HID (در صورت استفاده از کلاس HID)

ساختار این توصیف گرها در شکل های زیرنشان داده شده است.



شکل ۱-۳۳- ساختار توصیف گرها

سا ختار این توصیف گرهابرای هر دستگاه باید یکتا باشد تا کامپیوترراز اینکه چه دستگاهی به سخت افزار وصل است, آگاه کند.

نویسنده : حسین

پایمرد

یکی از کار های مهمی که توصیف گر پیکربندی, انجام می دهد , ارسال عددی است که نشان دهنده مقدار تغذیه لازم است . مثلا آیا دستگاه دارای تغذیه است و یااز باس تغذیه می شود. وقتی که دستگاه توسط کامپیوتر شناسایی می شود, کا مپیوتر توصیفگر دستگاه را می خواند و با توجه به اطلا عات خوانده شده مشخص می کند, که کدام تنظیم ها فعال شوند. این کار در هر مرحله فقط یک بار انجام می شود.

به عنوان مثال اگر دستگاه دارای تغذیه باس باشد و به کا مپیوتر متصل شود و یااگر دارای تغذیه سطح بالا باشد , کامپیوتر تنظیم می کند که تغذیه سطح بالا برای دستگاه خارجی فعال شود.

2-14-1- ساختار توصیف گرها

تمام توصیف گر ها دارای یک قالب مشخص هستند که این قالب را در جدول زیر می بینید.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes
1	bDescriptionType	1	Constant	DescriptorType
2	...	n		Start of parameters for descriptor

جدول ۱-۱۲- ساختار کلی توصیف گرها

اولین بایت مشخص کننده طول توصیف گر است. بایت دوم نیز مشخص کننده نوع آن بوده ووقتی که طول توصیف گر از طول تعیین شده کمتر باشد کامپیوتر آن را نادیده می گیرد. در صورتیکه بایت از اندازه تعیین شده بزرگتر بود, کامپیوتر اضافی آن را در نظر نمی گیرد و بقیه اطلاعات رااز پایان واقعی بایت مورد نظر می خواند.

1-2-14-1- ساختار توصیف گر دستگاه (Device Descriptor)

این توصیف گر معرف کلیات و خصوصیات دستگاه بوده و کاملا منحصربه فرد است.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of the Descriptor in Bytes (18 bytes)
1	bDescriptorType	1	Constant	Device Descriptor (0x01)
2	bcdUSB	2	BCD	USB Specification Number which device complies too.
4	bDeviceClass	1	Class	Class Code (Assigned by USB Org) If equal to Zero, each interface specifies it's own class code If equal to 0xFF, the class code is vendor specified. Otherwise field is valid Class Code.
5	bDeviceSubClass	1	SubClass	Subclass Code (Assigned by USB Org)
6	bDeviceProtocol	1	Protocol	Protocol Code (Assigned by USB Org)
7	bMaxPacketSize	1	Number	Maximum Packet Size for Zero Endpoint. Valid Sizes are 8, 16, 32, 64
8	idVendor	2	ID	Vendor ID (Assigned by USB Org)
10	idProduct	2	ID	Product ID (Assigned by Manufacturer)
12	bcdDevice	2	BCD	Device Release Number
14	iManufacturer	1	Index	Index of Manufacturer String Descriptor
15	iProduct	1	Index	Index of Product String Descriptor
16	iSerialNumber	1	Index	Index of Serial Number String Descriptor
17	bNumConfigurations	1	Integer	Number of Possible Configurations

جدول ۱-۱۳- ساختار توصیف گر دستگاه

این توصیف گر بیان کننده بعضی کلیات واطلاعات مهمی از دستگاه مانند نسخه USB , بیشترین طول بسته, کدشناسایی دستگاه وسازنده و تعداد تنظیمات ممکن است . قالب کلی آن در جدول بالا نشان داده شده است . همان طور که در جدول بالا نیز مشخص شده است فیلدهای مهم این توصیف گر به شرح زیر است.

۱. **bLength**

این فیلد اندازه توصیف گر دستگاه را مشخص می کند که برابر ۱۸ بایت است .

۲. **bDescriptorType**

این فیلد به صورت یک بایتی بوده و کد شناسایی توصیف گر دستگاه که برابر 01H است را مشخص می کند.

۳. **bcdUSB**

این فیلد نسخه USB که دستگاه باآن تطابق دارد رامشخص می کند. bcdUSB به صورت دو بایتی عمل می کند و مقدار آن به صورت کد BCD است. روش تعیین نسخه به این صورت است که بایت بالا قسمت صحیح و در بایت پایین, چهار بیت پایین تر قسمت صدم و چهار بیت بالاتر قسمت دهم را مشخص می کند. به طور مثال برای تعیین نسخه USB 1.1 , فیلد bcdUSB به صورت 0110H مقداردهی می شودو یا برای نسخه USB 2.0 به صورت 0200H مشخص می شود.

۴. **bDeviceClass**

دراین فیلدنام کلاسی که دستگاه متعلق به آن است قرارمی گیرد, bDeviceClass توسط سیستم عامل میزبان برای تشخیص درایور مربوط به دستگاه مورد استفاده قرار می گیرد, مقدار این فیلد می تواند از ۱ تا FFH باشد , مقدار 00H نشان دهنده آن است که دستگاه از نوع چندکلاسی است و مقدار FFH نشان دهنده آن است که دستگاه از نوع کلاسی است که توسط فروشنده نوشته شده است .

۵. **bDevice SubClass**

این فیلد نیز مانند فیلدقبلی است و چنانچه مقدارفیلد bDeviceClass صفر باشد, این فیلد نیز باید صفر شود.

۶. **bDeviceProtocol**

این فیلد پروتکل مشخصی از کلاس یازیر کلاس را تعیین می کند و مقدار آن می تواند بین ۱ تا FFH باشد.

۷. **bMaxPacketSize**

حداکثر اندازه پاکت برای اندپوینت صفر. میزبان از این مقدار برای خواسته های بعدی استفاده می کند . این عدد در دستگاه های سرعت پایین باید ۸ باشد. در دستگاه های سرعت بالا می تواند 32,16,8 یا ۶۴ باشد. دستگاه های سرعت نهایی باید از ۶۴ استفاده کنند.

۸. **idVendor**

اعضای گروه ابزار آلات USB و کسانی که مبلغ تعیین شده را پرداخته اند این اجازه را دارند که از یک شماره مشخصه فروشنده منحصر بفرد استفاده کنند.

همه دستگاه های USB تجاری باید این شماره مشخصه راداشته باشند. میزبان نیز ممکن است فایل INF ی داشته باشد که حاوی این مقداراست ودر این صورت این مقدار به ویندوز برای پیداکردن راه انداز مناسب و بارگذاری کردن آن کمک خواهد کرد.

۹. **idProduct**

هرکارخانه به ساخته خود شماره مشخصه محصولی برای مشخص کردن دستگاه نسبت می دهد.هردوتوضیح دهنده دستگاه و فایل INF میزبان این مقداررادر خود دارند. در این حالت ویندوز برای پیداکردن و بارگذاری راه انداز مناسب از این مقدار استفاده خواهد کرد.

۱۰. **bcdDevice**

شماره نسخه دستگاه با قالب بندی BCD که توسط کارخانه به آن نسبت داده می شودواختیاری است. این مقدارنیزمی تواند در پیداکردن راه انداز کمک کند.

۱۱. **iManufacture**

یک اشاره گر به توضیح دهنده رشته ای که کارخانه را توضیح می دهد و اختیاری است . در صورت عدم استفاده صفر می باشد.

۱۲. **iProduct**

اشاره گری است به یک توضیح دهنده رشته که محصول را توضیح می دهدو اختیاری است . در صورت عدم استفاده صفر می باشد.

۱۳. **iSerialNumber**

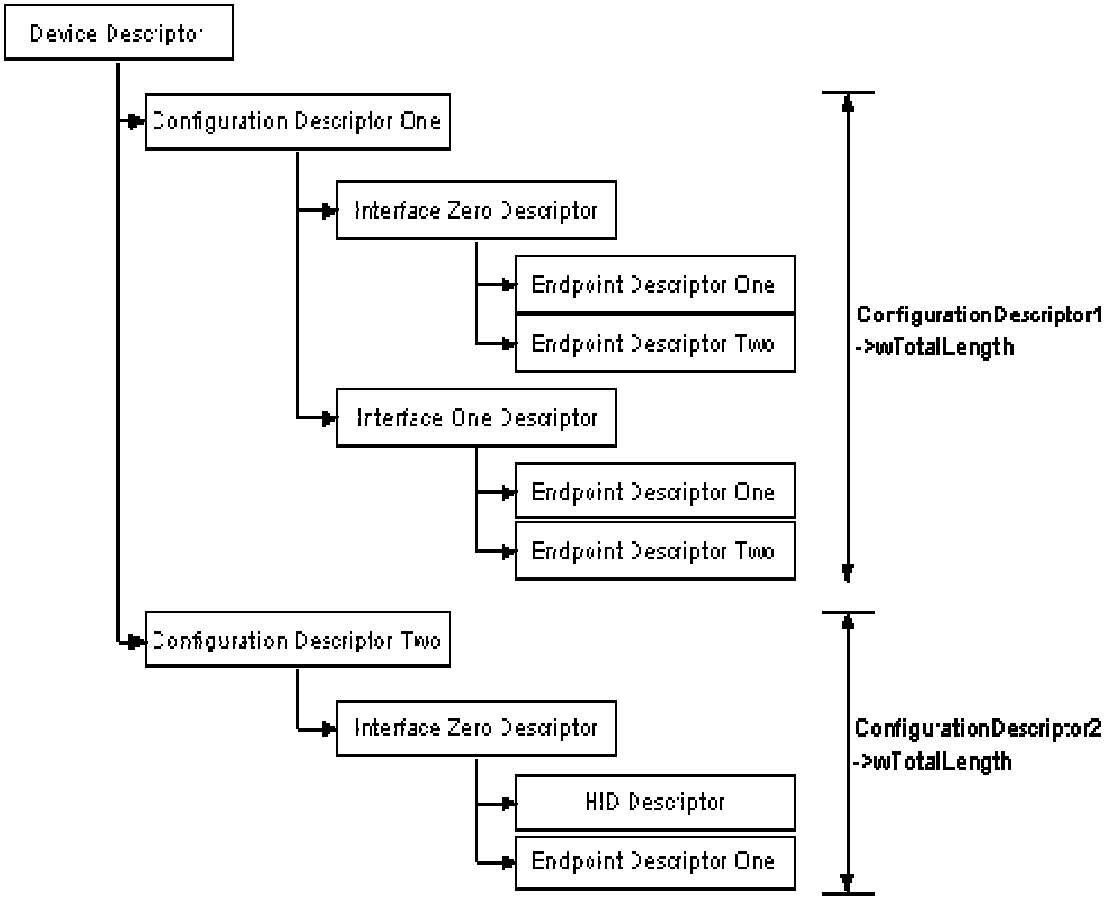
اشاره گری که به رشته ای که حاوی شماره سریال دستگاه است اشاره می کند. اختیاری است در صورت عدم استفاده صفر می باشد. وقتی که کاربر چند دستگاه از یک نوع بر روی باس داشته باشد شماره سریال مفید می باشد که بوسیله آن میزبان دستگاه مورد نظر خودرا مشخص کند.

۱۴. **bNumConfigurations**

تعداد پیکر بندی هایی که دستگاه پشتیبانی می کند.

1-14-2-2- ساختار توصیف گر پیکربندی (Configuration Descriptor)

پس از دریافت توضیح دهنده دستگاه , میزبان می تواند شروع به بازیابی توضیح دهنده های پیکربندی, مدار, واسط واندپوینت کند. هر دستگاه حداقل باید یک توضیح دهنده پیکربندی داشته باشد که تواناییها و ویژگی های دستگاه را توضیح می دهد.معمولا یک پیکر بندی کافی است . اما دستگاهی که چند ین کاربرد دارد و یااز مدهای فعالیت متفاوتی استفاده می کند می تواند از چندین پیکر بندی پشتیبانی کند. درهرلحظه فقط یک پیکربندی فعال است. هر پیکربندی احتیاج به یک توضیح دهنده دارد . توضیح دهنده پیکر بندی حاوی اطلاعاتی از قبیل مصرف انرژی دستگاه و تعداد مدارهای واسطی است که پشتیبانی می شود.هر توضیح دهنده پیکر بندی , چندین توضیح دهنده زیر شاخه خود دارد که شامل یک یا چند توضیح دهنده مدارواسط و احتمالا توضیح دهنده های اندپوینت می باشند.



شکل ۱-۳۴- توصیف گر پیکر بندی

میزبان با استفاده از خواسته Set_Configuration پیکربندی خاصی را انتخاب می کند و توسط خواسته Get_Configuration پیکر بندی جاری را می خواند.

توضیح دهنده شامل ۸ فیلد است که در جدول صفحه بعد لیست شده اند. فیلد ها شامل اطلاعاتی در مورد خود توضیح دهنده , پیکربندی و مصرف انرژِی دستگاه در پیکربندی می باشند .

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes
1	bDescriptorType	1	Constant	Configuration Descriptor (0x02)
2	wTotalLength	2	Number	Total length in bytes of data returned
4	bNumInterfaces	1	Number	Number of Interfaces
5	bConfigurationValue	1	Number	Value to use as an argument to select this configuration
6	iConfiguration	1	Index	Index of String Descriptor describing this configuration
7	bmAttributes	1	Bitmap	D7 Reserved, set to 1. (USB 1.0 Bus Powered) D6 Self Powered D5 Remote Wakeup D4..0 Reserved, set to 0.
8	bMaxPower	1	mA	Maximum Power Consumption in 2mA units

جدول ۱-۱۴- توصیف گر پیکربندی

فیلد های این توصیف گر به شرح زیر هستند:

۱. **bLength**

طول توضیح دهنده بر حسب بایت

۲. **bDescriptorType**

در توضیح دهنده پیکر بندی ثابت 02H است.

۳. **wTotalLength**

تعداد بایت های داده ای که دستگاه باز می گرداند و شامل مجموعه بایت های همه توضیح دهنده های مداراسط و اندپوینت ها می باشد.

۴. **bConfigurationValue**

شماره پیکربندی که توسط خواسته های Get_configurationو Set_Configuration استفاده می شود.

خواسته Set_Configuration با مقدار صفر باعث می شود که دستگاه از حالت پیکربندی خارج شود.

۵. **iConfiguration**

اشاره گر به یک رشته که پیکربندی را توضیح می دهد و اختیاری است .

۶. **bNumInterface**

تعداد مدارهای واسطی که پیکربندی پشتیبانی می کند و حداقل می تواند یک باشد.

۷. **bmAttributes**

بیت ششم برابر یک اگر دستگاه خود توان باشد. بیت پنجم برابر یک اگر دستگاه ازویژگیremote wakeup پشتیبانی کند. این خصوصیات, دستگاه هایی را که وارد حالت بیکاری شده اند قادر می سازد که به میزبان اطلاع دهند که می خواهند ارتباط بر قرار کنند. یک دستگاه USB درصورتی که به مدت ۳ میلی ثانیه فعالیتی روی باس نداشته باشد وارد حالت بیکاری می شود. حال اگر رویدادی در دستگاه بیکار احتیاج به فعالیت میزبان داشته باشد, دستگاهی که ویژگی remote wakeup را پشتیبانی می کند قادر است از میزبان بخواهد ارتباط را دوباره آغاز کند.

بقیه بیت ها بی استفاده هستند. بیت های صفر تا چهار باید صفر و بیت هفت باید یک باشد

۸. **MaxPower**

تعیین می کند که دستگاه چقدر جریان لازم دارد. مقدار MaxPower برحسب میلی آمپر , برابر نیمی از جریانی است که دستگاه احتیاج دارد . اگر دستگاه به ۲۰۰ میلی آمپر احتیاج داشته باشد MaxPower=100 خواهد بود. حداکثر جریانی که مجاز می باشد ۵۰۰ میلی آمپر می باشد. اگر میزبان در یابد که جریان خواسته شده موجود نیست, از پیکر بندی دستگاه صرف نظر می کند.

3-2-14-1- ساختار توصیف گرمدار واسط (Interface descriptor)

در کنار دستگاه و توضیح دهنده ها , مدارواسط به معنای مجموعه ای از اندپوینت هاست که برای انجام وظایف دستگاه استفاده می شوند. توضیح دهنده مدارواسط مربوط به یک پیکربندی حاوی اطلاعاتی در مورد اندپوینت هایی است که مدار واسط پشتیبانی می کند.

هر پیکربندی باید حداقل از یک مدارواسط پشتیبانی کند, ودر بیشتر دستگاه ها همین یک مدارواسط کافی است . اما یک پیکربندی می تواند چندین مدارواسط داشته باشد که در یک زمان فعال هستند, هر مدار واسط شامل توضیح دهنده مدارواسط و توضیح دهنده های اندپوینت زیر شاخه آن می باشد.

دستگاهی که پیکربندی ای باچندین مدارواسط داردوهمه آنها دریک زمان فعالند رادستگاه ترکیبی می نامند و میزبان برای هر مدار واسط راه اندازی را بارگذاری می کند. اگر دستگاهی چندین کاربرداشته باشد, به جای استفاده

ازچندین پیکربندی از یک پیکربندی با چندین مدارواسط تناوبی و انحصاری استفاده می شود چون تغییر مدار واسط ساده تر از تغییر پیکربندی است . میزبان شماره واسط را از طریق خواسته Set_Interface تنظیم می کند و توسط خواسته Get_Interface شماره مدار واسط جاری را می خواند. هر مدار واسط حاوی توضیح دهنده خود و توضیح دهنده های زیر شاخه خود می باشد.

توضیح دهنده مدارواسط شامل ۹ بایت است که در جدول زیر می بینید.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes (9 Bytes)
1	bDescriptorType	1	Constant	Interface Descriptor (0x04)
2	bInterfaceNumber	1	Number	Number of Interface
3	bAlternateSetting	1	Number	Value used to select alternative setting
4	bNumEndpoints	1	Number	Number of Endpoints used for this interface
5	bInterfaceClass	1	Class	Class Code (Assigned by USB Org)
6	bInterfaceSubClass	1	SubClass	Subclass Code (Assigned by USB Org)
7	bInterfaceProtocol	1	Protocol	Protocol Code (Assigned by USB Org)
8	iInterface	1	Index	Index of String Descriptor Describing this interface

جدول ۱-۱۵- توصیف گر مدار واسط

بسیاری از دستگاه ها احتیاج به همه این فیلد ها ندارند, توضیحات زیر این اطلاعات رابراساس کاربرد آن ها گروه بندی می کنند.

۱. **bLength**

طول توضیح دهنده بر حسب بایت

۲. **bDescriptorType**

برای توضیح دهنده مدار واسط ثابت 04H است.

۳. **iInterface**

اشاره گری به یک رشته که مدار واسط را توضیح می دهد.

۴. **bInterfaceNumber**

مدار واسط را مشخص می کند. در دستگاه های ترکیبی, هر پیکر بندی چندین مدار واسط دارد که در یک لحظه فعالند. هر مدار واسط باید توضیح دهنده ای با مقدار منحصر به فرد در این فیلد باشد . مقدار پیش فرض صفر است .

۵. **bAlternateSetting**

وقتی که پیکربندی از چندین مدار واسط تناوبی و انحصاری پشتیبانی می کند همه توضیح دهنده ها باید مقادیر یکسانی در bInterfaceNumber داشته باشندو مقادیر منحصر بفرد ی را در این فیلد مشخص کنند. توسط خواسته Get_Interface تنظیمات جاری این فیلد را بازیابی می کنیم وخواسته Set_Interface تنظیمی را که می خواهد استفاده کند مشخص می نماید.

۶. **bNumEndpoints**

تعداد اندپوینت هایی که مدارواسط علاوه بر اند پوینت صفر پشتیبانی می کند. در دستگاه هایی که فقط اندپوینت صفر دارند این مقدار باید صفر باشد.

۷. **bInterfaceClass**

مشابه با فیلد DeviceClass مربوط به توضیح دهنده دستگاه است , اما برای دستگاه هایی که کلاسی ویژه برای مدار واسط خود دارند به کار می رود. مقادیر بین 01H تا FEH برای کلاس های USB رزروهستند.
کدکلاس 03H,HID است .FFH کلاس تعریفی فروشنده را مشخص می کند. مقدار صفر رزرو است .
در شکل صفحه بعد جدول مقادیر رزرو شده را می بینید.

۸. **bInterfaceSubClass**

همانند فیلد bDeviceSubClass مر بوط به توضیح دهنده دستگاه است. اما برای دستگاه هایی که کلاسی ویژه برای مدارواسط خود دارند به کار می رود. درمدارهای واسطی که متعلق به کلاس خاصی هستند, این فیلدممکن است زیر کلاس آن رامشخص کند.اگر bInterfaceClass صفر باشد این فیلد نیز باید صفر باشد. اما اگر bInterfaceClass بین ۱تا FEH است , این فیلد باید کدی باشد که در مرجع خصوصیات تعریف شده است .
مقدار FFH نشان دهنده زیر کلاس تعریفی فروشنده می باشد.

۹. **bInterfaceProtocol**

همانند bDeviceprotocol مربوط به توضیح دهنده دستگاه است , اما برای دستگاه هایی که کلاسی ویژه برای مدار واسط خود دارند به کار می رود. اگر bInterface بین ۱ تا FEH باشد این فیلد باید شامل کدی باشد که در مرجع خصوصیات USB تعریف شده است .

Class Code (hexadecimal)	Description
01	Audio
02	(Communication Device Class) Communication Interface
03	Human Interface Device
05	Physical
06	Image
07	Printer
08	Mass storage
09	Hub
0A	(Communication Device Class) Data Interface
0B	Smart Card
0D	Content Security
0E	Video
DC	Diagnostic device (can also be declared at the device level) bInterfaceSubClass = 1 for Reprogrammable Diagnostic Device with bInterfaceProtocol = 1 for USB2 Compliance Device
E0	Wireless controller (can also be declared at device level) bInterfaceSubClass = 1 for RF Controller with bInterfaceProtocol = 1 for Bluetooth Programming Interface
FE	Application specific bInterfaceSubClass = 1 for Device Firmware Update bInterfaceSubClass = 2 for IrDA Bridge bInterfaceSubClass = 3 for Test and Measurement
FF	Vendor specific (can also be declared at the device level)

جدول ۱-۱۶- مقادیر فیلد bInterfaceProtocol

1-14-2-4- ساختار توصیف گراندپوینت (Endpoint Descriptor)

هر اندپوینتی که در توضیح دهنده مدار واسط مشخص شده است دارای یک توضیح دهنده اندپوینت است . اندپوینت صفر هیچگاه توضیح دهنده ندارد چون همه دستگاه ها بایداندپوینت صفررا پشتیبانی کنند. در توضیح دهنده دستگاه نیز حداکثراندازه پاکت آن , ذکر می شود. جدول صفحه بعد شش فیلد توضیح دهنده اندپوینت را لیست کرده است .

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes (7 bytes)
1	bDescriptorType	1	Constant	Endpoint Descriptor (0x05)
2	bEndpointAddress	1	Endpoint	Endpoint Address Bits 0..3b Endpoint Number. Bits 4..6b Reserved. Set to Zero Bits 7 Direction 0 = Out, 1 = In (Ignored for Control Endpoints)
3	bmAttributes	1	Bitmap	Bits 0..1 Transfer Type 00 = Control 01 = Isochronous 10 = Bulk 11 = Interrupt Bits 2..7 are reserved. If Isochronous endpoint, Bits 3..2 = Synchronisation Type (Iso Mode) 00 = No Synchonisation 01 = Asynchronous 10 = Adaptive 11 = Synchronous Bits 5..4 = Usage Type (Iso Mode) 00 = Data Endpoint 01 = Feedback Endpoint 10 = Explicit Feedback Data Endpoint 11 = Reserved
4	wMaxPacketSize	2	Number	Maximum Packet Size this endpoint is capable of sending or receiving
6	bInterval	1	Number	Interval for polling endpoint data transfers. Value in frame counts. Ignored for Bulk & Control Endpoints. Isochronous must equal 1 and field may range from 1 to 255 for interrupt endpoints.

جدول ۱-۱۷- توصیف گراندپوینت

۱. **bLength**

طول توضیح دهنده بر حسب بایت

۲. **bDescriptorType**

در توضیح دهنده اند پوینت ثابت 05H است .

۳. **bEndpointAddress**

حاوی شماره اندپوینت و جهت آن است . بیت های صفر تا سه شماره اند پوینت را مشخص می کنند . دستگاه های سرعت پایین می توانند حداکثر سه اند پوینت داشته باشند در حالی که دستگاه های سرعت بالا و نهایی قادرند شامل ۱۶ اندپوینت باشند. بیت هفت جهت اند پوینت را نشان می دهد:خروجی=۰ و ورودی=۱ . بیت های 5,4 و۶ بی استفاده هستندو باید صفر باشند.

۴. **bmAttibutes**

بیت های ۰و۱ نوع انتقالی که اندپوینت را پشتیبانی می کند مشخص می کند. ۰۰=کنترلی , ۰۱=همزمان , ۱۰=توده ای و ۱۱=وقفه ای . برای اندپوینت صفر این انتقال به صورت پیش فرض کنترلی است . در USB نسخه ۱.۱ بیت های ۲ تا۷ رزرو هستند. در USB نسخه ۲.۰ بیت های ۲ تا ۵ برای اندپوینت های همزمان سرعت بالا و نهایی استفاده می شوند. بیت های ۲و۳ نوع سنکرون را نشان می دهند. ۰۰=بدون سنکرون کردن , ۰۱=همزمان , ۱۰=انطباقی , برای اندپوینت های بدون سنکرون, بیت های ۲ تا ۵ باید صفر باشند. در همه اندپوینت ها بیت های ۶ و۷ باید صفر باشند.

۵. **bMaxPacketSize**

حداکثر تعداد بایتی که اندپوینت می تواند در یک ترنزکشن انتقال دهد. مقادیر مجاز بسته به سرعت دستگاه و نوع انتقال متفاوت است .

بیت های ده تا صفر حداکثر اندازه پاکت را بین صفر تا۱۰۲۴ تعیین می کنند. درUSB2.0 بیت های ۲و ۱۱ تعداد ترنزکشن های اضافی در هر میکروفریم که اندپوینت سرعت نهایی پشتیبانی می کند را مشخص می کنند:۰۰=هیچ ترنزکشن اضافی, ۰۱=یک ترنزکشن اضافی, ۱۰=دوترنزکشن اضافی, ۱۱=رزرو. در USB1.x این بیت ها باید صفر باشند. بیت های ۱۳ تا ۱۵ نیز رزرو هستندو باید صفر باشند.

۶. **bInterval**

حداکثر زمان تاخیر اندپوینت های وقفه ای یا فاصله زمانی برای اندپوینت های همزمان یا حداکثر نرخ NAK برای اندپوینت های کنترلی یا توده ای خروجی سرعت نهایی را مشخص می کند. دامنه مجاز و چگونگی استفاده از آن بسته به سرعت دستگاه , نوع انتقال و اینکه دستگاه USB2.0 را پشتیبانی می کند یا نه, متفاوت هستند.

در اندپوینت های وقفه ای سرعت پایین , حداکثر زمان تاخیر برابر bInterval بر حسب میلی ثانیه می باشد که می تواند بین ۱۰ تا ۲۵۵ متغییر باشد.

برای همه اندپوینت های وقفه ای سرعت بالا و اندپوینت های همزمان سرعت بالا منطبق با نسخه 1.x فاصله زمانی برابر bInterval برحسب میلی ثانیه است و می تواند مقداری بین ۱ تا۲۵۵ را اختیار کند. در اندپوینت های همزمان دستگاه 1.x مقدار باید ۱ باشد. در اندپوینت های همزمان دستگاه ۲.۰ مقدار بین ۱ تا ۱۶ مجاز است و فاصله زمانی

توسط فرمول $2^{bInterval-1}$ محاسبه می شود. این فرمول محدوده ای بین ۱ میلی ثانیه تا ۳۲.۷۶۸ ثانیه را ممکن می سازد.

در انتقال های کنترلی و توده ای سرعت بالا از این مقدار صرف نظر می شود.

در اندپوینت های سرعت نهایی , مقدار در واحد های ۱۲۵ میکروثانیه ای است که عرض میکروفریم می باشد. این مقدار برای اندپوینت های وقفه ای و همزمان می تواند بین ۱ تا ۱۶ باشد که به این ترتیب فاصله زمانی توسط فرمول $2^{bInterval-1}$ محاسبه می شود و محدوده ای بین ۱۲۵ میکروثانیه تا ۴.۰۹۶ ثانیه را ممکن می سازد. در اندپوینت های کنترلی و توده ای خروجی سرعت نهایی , این مقدار حداکثر نرخ NAK را مشخص می کند. این مقدار مربوط به وقتی است که دستگاه داده را دریافت می کند و با ACK پاسخ می دهد اما میزبان هنوز داده هایی برای فرستادن دارد با بازگرداندن ACK دستگاه می گوید که انتظار دارد که بتواند داده ترنزکشن بعدی را قبول کند. اگر پاکت داده بعدی رسید و به دلایلی دستگاه نتوانست آن را بپذیرد , اندپوینت NAK باز می گرداند. مقدار bInterval می گوید که اندپوینت بیشتر از یک NAK در هر دوره مشخص شده در bInterval نخواهد فرستاد. این مقدار می تواند بین ۰ تا ۲۵۵ میکروفریم باشد. مقدار صفر به معنای آن است که اندپوینت هیچگاه NAK نمی فرستد.

5-2-14-1- ساختار توصیف گر رشته (String Descriptor)

توصیف گر رشته حاوی یک متن توضیحی است . مرجع خصوصیات رشته هایی برای کارخانه, محصول, شماره سریال , پیکر بندی و مدار واسط تعریف می کند. همچنین دستگاه ممکن است توضیح دهنده های رشته دیگری نیز داشته باشد. استفاده از این توضیح دهنده ها اختیاری است.

جدول صفحه بعد نشان دهنده توصیف گرهای رشته ای است که معرف زبان مورد استفاده هستند.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes
1	bDescriptorType	1	Constant	String Descriptor (0x03)
2	wLANGID[0]	2	number	Supported Language Code Zero (e.g. 0x0409 English - United States)
4	wLANGID[1]	2	number	Supported Language Code One (e.g. 0x0c09 English - Australian)
n	wLANGID[x]	2	number	Supported Language Code x (e.g. 0x0407 German - Standard)

جدول ۱-۱۸- توصیف گر رشته

۱. **bLength**

طول توضیح دهنده بر حسب بایت

۲. **bDescriptorType**

در توضیح دهنده اند پوینت ثابت 03H است .

- **رشته**

هر رشته یک اشاره گر دارد. رشته صفر دارای کاربردی ویژه است و شماره مشخصه زبان را مشخص می کند در حالی که دیگر رشته ها می توانند حاوی هر متنی باشند.

۳. **wLANGID[0....n]**

فقط در توضیح دهنده رشته صفر استفاده می شود. توضیح دهنده رشته صفر شامل یک یا چند کد شماره مشخصه زبان ۱۶ بیتی است که زبانی که رشته ها به وسیله آن ارایه می شوند را مشخص می کند. این کد برای زبان انگلیسی 0009H و زیرکد برای U.S English مقدار0004H می باشد. به نظر می رسد که این کد , تنها کد مجاز در ویندوز ۹۸ انگلیسی است . این مقدار باید برای همه رشته ها مجاز باشد. در دستگاهی که توضیح دهنده رشته وجود ندارد , نباید آرایه شماره مشخصه زبان باز گردانده شود. در سایت مجمع ابزارالات USB , فهرستی از شماره مشخصه زبان های تعریف شده موجود می باشد.

bString : دررشته های شماره یک یا بیشتر فیلد String شامل یک رشته باقالب بندی Unicode است .

Unicode از ۱۶ بیت برای مشخص کردن هر کاراکتر استفاده می کند. با کمی تفاوت کاراکترهای با قالب بندی ANSI با کدهای 00H تا 7FH مربوط به کاراکترهای با قالب بندی Unicode بین 0000H تا می باشند. مثلا رشته محصول برای ساخته ای که "Gizmo" نامیده می شود باید حاوی پنج Unicode شانزده بیتی باشد 0047 0069 007A 006D 006F .

در جدول زیر قالب اصلی این توصیف گر نشان داده شده است .

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes
1	bDescriptorType	1	Constant	String Descriptor (0x03)
2	bString	n	Unicode	Unicode Encoded String

جدول ۱-۱۹- ساختار رشته در توصیف گر رشته

15-1- خواسته های استاندارد USB

در پروتکل USB , یازده خواسته استاندارد وجود دارد که در بخش ۹.۴ از مرجع خصوصیات USB در مورد دستگاه استاندارد در مورد آن ها توضیح داده شده است. این خواسته ها به طور کلی شامل خواسته های استاندارد دستگاه , واسط و اندپوینت می شوند. همه دستگاه ها باید به این خواسته ها پاسخ دهند. دستگاه باید در صورت پشتیبانی نکردن از برخی خواسته ها , وضعیت STALL را باز گردند.

1-15-1- خواسته های استاندارد دستگاه

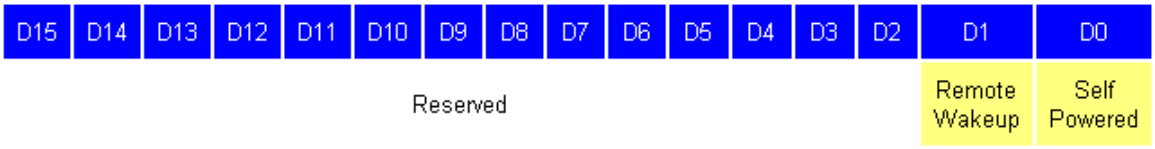
در جدول زیر صورت کلی این خواسته ها را می بینید.برای آگاهی از فیلد ها به قسمت ارسال کنترلی بخش خواسته بر گردید.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
1000 0000b	GET_STATUS (0x00)	Zero	Zero	Two	Device Status
0000 0000b	CLEAR_FEATURE (0x01)	Feature Selector	Zero	Zero	None
0000 0000b	SET_FEATURE (0x03)	Feature Selector	Zero	Zero	None
0000 0000b	SET_ADDRESS (0x05)	Device Address	Zero	Zero	None
1000 0000b	GET_DESCRIPTOR (0x06)	Descriptor Type & Index	Zero or Language ID	Descriptor Length	Descriptor
0000 0000b	SET_DESCRIPTOR (0x07)	Descriptor Type & Index	Zero or Language ID	Descriptor Length	Descriptor
1000 0000b	GET_CONFIGURATION (0x08)	Zero	Zero	1	Configuration Value
0000 0000b	SET_CONFIGURATION (0x09)	Configuration Value	Zero	Zero	None

جدول ۱-۲۰- خواسته های استاندارد دستگاه

- **Get_Status**

این خواسته از دستگاه درخواست ارسال ۲ بایت اطلاعات را در ترنزکشن داده می کند. قالب این اطلاعات به صورت زیر می باشد.



شکل ۱-۳۵- جواب خواسته Get_Status دستگاه

پس از ارسال این خواسته, دستگاه کلمه بالا را باز می گرداند که در آن بیت صفرم بیت خود توان (Self Powered) نامیده می شودو در صورت یک بودن مشخص می شود که دستگاه از نظر تغذیه ولتاژ مستقل از میزبان است و در صورت صفر بودن دستگاه , گذرگاه توان (Bus Powered) عمل می کند.
بیت یکم به نام Remote Wakeup نام دارد. اگر دستگاهی به مدت حدود 3ms فعالیتی بر روی گذرگاه نداشته باشد وارد مد بیکاری (IDEL) خواهد شد و چنانچه دستگاه دارای قابلیت Remote Wakeup باشد می تواند از میزبان درخواست کند تا ارتباط دوباره شروع شود.

- **Set_Feature**

این خواسته برای فعال کردن یک خاصیت ویژه در دستگاه یا واسط و یا اندپوینت استفاده می شود. از خاصیت های ویژه می توان به DEVICE_REMOTE_WAKEUP(01H) که برای فعال کردن خاصیت Remote Wakeup دستگاه و ENDPOINT_HALT (00H) که برای فعال کردن حالت توقف اندپوینت به کار می رود, اشاره نمود.

- **Clear_Feature**

این خواسته عکس Set_Feature عمل می کند.و باعث غیر فعال شدن ویژگی مورد نظر خواهد شد.

- **Set_Address**

این خواسته آدرس دستگاه را تنظیم می کند. باید توجه داشت که قبل از ارسال این خواسته , میزبان با آدرس پیش فرض صفر و اندپوینت صفر با دستگاه ارتباط برقرار می کند و میزبان با فرستادن خواسته Set_Address یک آدرس منحصر بفرد به دستگاه می دهد. دستگاه نیز پس از دریافت , آدرس آن را ذخیره نموده تا در ارتباط های بعدی از آن استفاده نماید.

- **Get_Descriptor**

از این خواسته برای گرفتن یک توصیف گر از دستگاه استفاده می شود. نوع توصیف گر نیز می تواند از نوع دستگاه , پیکربندی, واسط, اندپوینت و رشته باشد. در توصیف گر رشته , بایت پایین فیلد مقدار شماره رشته و فیلد شاخص شماره مشخصه زبان است که برای زبان انگلیسی آمریکایی 0409H می باشد.

برای مثال چنانچه میزبان خواسته Get_descriptor را برای خواندن توصیف گر پیکر بندی بفرستد, دستگاه در پاسخ به این خواسته می تواند اطلاعات زیر را بر گرداند:

09 02 00 20 01 01 00 80 64

bLength=09H اندازه داده به طول ۹ بایت است.

bDescriptorType=02H کد شناسایی توصیف گر پیکر بندی است.

wTotalLength=0020H اندازه مجموع بایت های توصیف گر های پیکربندی واسط و اندپوینت که برابر ۳۲ بایت (20H) است.

bNumInterface=01H تعداد واسط ها برابر یک است.

bConfigurationValue=01H شماره پیکربندی برابر 01H است.

Iconfiguration=00H

bmAttributes=80H از این فیلد می توان فهمید دستگاه, گذرگاه توان بوده و دارای ویژگی Remote Wakeup نیست.

MaxPower=64H از این فیلد می توان فهمید حداکثر جریان مجاز دستگاه برابر 200mA است.

- **Set_Descriptor**

از این خواسته برای به روز کردن یک توصیف گر در دستگاه استفاده می شود. چنانچه دستگاهی از این خواسته پشتیبانی نکند وضعیت STALL را باز می گرداند, بیشتر دستگاه ها از این خواسته پشتیبانی نمی کنند . قالب بندی این خواسته مانند خواسته Get_Descriptor است با این تفاوت که در فیلد نوع خواسته مقدار ۰۰ و در فیلد خواسته مقدار 07H قرار خواهد گرفت و بسته داده در ترنزکشن داده , اطلاعاتی که قرار است در مورد توصیف گر به دستگاه ارسال شود قرار می گیرد.

- **Set_Configuration**

این خواسته پیکربندی دستگاه را انجام می دهد. چنانچه دستگاه از این خواسته پشتیبانی نکند, وضعیت STALL را باز می گرداند.

- **Get_Configuration**

این خواسته مقادیر پیکر بندی دستگاه را گرفته و در صورتی که مقدار آن صفر باشد, نشان دهنده آن است که دستگاه پیکربندی نشده است.

2-15-1- خواسته های استاندارد واسط

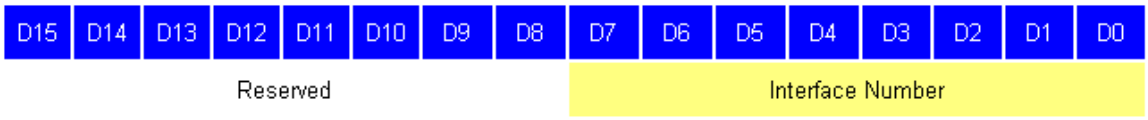
مرجع خصوصیت USB , تعداد پنج خواسته استاندارد را مطابق جدول زیر برای واسط ها در نظر گرفته است.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
1000 0001b	GET_STATUS (0x00)	Zero	Interface	Two	Interface Status
0000 0001b	CLEAR_FEATURE (0x01)	Feature Selector	Interface	Zero	None
0000 0001b	SET_FEATURE (0x03)	Feature Selector	Interface	Zero	None
1000 0001b	GET_INTERFACE (0x0A)	Zero	Interface	One	Alternate Interface
0000 0001b	SET_INTERFACE (0x11)	Alternative Setting	Interface	Zero	None

جدول ۱-۲۱- خواسته های استاندارد واسط

- **wIndex**

این فیلد برای معین کردن واسط مورد نظر خواسته به کار برده می شود که به فرم کلی زیر است.



شکل ۱-۳۶- فیلد wIndex خواسته واسط

- **Get_Status**

برای باز گرداندن وضعیت واسط به کار می رود. یک واسط باید 0000H را بر گرداند.

- **Clear_Feature / Set_Feature**

مانند خواسته های دستگاه برای فعال یا غیر فعال کردن یک خاصیت ویژه در واسط به کار می رود.

- **Get_Interface**

از این خواسته جهت باز گرداندن تنظیمات واسط انتخاب شده , استفاده می گردد.

- **Set_Interface**

این خواسته با توجه به فیلد bAlternateSetting در توصیف گر واسط کاربرد پیدا می کند . چنانچه واسط از تنظیم تناوبی استفاده کند با استفاده از این خواسته می توان تنظیم تناوبی مورد نظر را مشخص کرد.

3-15-1- خواسته های استاندارد اندپوینت

خواسته های استاندارد اندپوینت شامل چهار خواسته که در جدول بعد آمده است , می شود.

bmRequestType	bRequest	wValue	Windex	wLength	Data
1000 0010b	GET_STATUS (0x00)	Zero	Endpoint	Two	Endpoint Status
0000 0010b	CLEAR_FEATURE (0x01)	Feature Selector	Endpoint	Zero	None
0000 0010b	SET_FEATURE (0x03)	Feature Selector	Endpoint	Zero	None
1000 0010b	SYNCH_FRAME (0x12)	Zero	Endpoint	Two	FrameNumber

جدول ۱-۲۲- خواسته های استاندارد اندپوینت

- **wIndex**

این فیلد برای معین کردن اندپوینت مورد نظر خواسته و همچنین جهت خواسته های یک اندپوینت مانند شکل زیر به کار می رود.

شکل ۱-۳۷- فیلد wIndex خواسته اندپوینت

- **Get_Status**

این خواسته دو بایت را که نشان دهنده وضعیت اندپوینت (Halted / Stalled) است بر می گرداند.

شکل ۱-۳۸- پاسخ به خواسته Get_Status اندپوینت

- **Set_Feature / Clear_Feature**

این خواسته برای وضع کردن یا پاک کردن یک خصیصه در اندپوینت به کار می رود. مرجع خصوصیات USB تنها یک خصیصه با عنوان ENDPOINT-HALT با مقدار 00H برای این خواسته در نظر گرفته است که به میزبان اجازه می دهد یک اندپوینت را بی استفاده یا پاک کند.

- **SYNCH_FRAME**

برای انتشار یک فریم همزمانی به کار می رود.

16-1- سر شماری

قبل از اینکه برنامه کاربردی با دستگاه ارتباط بر قرار کند , میزبان باید دستگاه را بشناسد و راه انداز آن را بار گذاری کند. سرشماری, اولین تبادل اطلاعاتی است که انجام می شود. این پروسه , شامل نسبت دادن یک آدرس به دستگاه , خواندن ساختار داده ها از دستگاه, نسبت دادن و بارگذاری کردن راه انداز دستگاه و انتخاب یک پیکربندی با استفاده از داده های بازیابی شده می باشد. بعد از طی این مرحله دستگاه پیکربندی شده و آماده انتقال داده با استفاده از اندپوینت های تعریف شده در پیکربندی خواهد بود.

1-16-1- پردازش

یکی از وظایف هاب, تشخیص اتصال یا جداشدن دستگاه ها ست. هر هاب یک مسیر ارتباطی وقفه ای ورودی برای گزارش این گونه رخدادها به میزبان دارد. در هنگام بالا آمدن سیستم, میزبان هاب ریشه را برای تشخیص دستگاه هایی که به آن متصل هستند جستجو می کند. این جستجو شامل هاب های خارجی و دستگاه هایی که به آن ها وصل هستند نیز می شود. پس از بالا آمدن نیز میزبان همواره آماده است که اتصال دستگاه های جدید یا جدا شدن بعضی دستگاه ها را تشخیص دهد.

پس از تشخیص دستگاه جدید, میزبان یک سری نیازمندی ها را به هاب دستگاه می فرستد که باعث می شود هاب مسیر ارتباطی بین میزبان و دستگاه را برقرار کند. سپس میزبان, توسط فرستادن خواسته های کنترلی استاندارد USB به اندپوینت پیش فرض صفر تلاش می کند تا دستگاه را سرشماری کند. همه دستگاه های USB باید از انتقال کنترلی , خواسته های استاندارد و اندپوینت صفر پشتیبانی کنند. برای یک سرشماری موفق , دستگاه باید با بازگرداندن اطلاعات خواسته شده و انجام عملیات مناسب به هر خواسته پاسخ دهد.

در یک وسیله جانبی عمومی , کد برنامه دستگاه باید حاوی اطلاعاتی که میزبان درخواست می کند و پاسخ های مناسب به خواسته ها باشد. در سمت میزبان, تحت ویندوز احتیاجی به نوشتن کدی برای سر شماری نیست زیرا ویندوز به صورت خودکار این کار را انجام می دهد. ویندوز به دنبال فایل های متنی خاصی که INF نامیده می شوند می گردد که وظیفه آن ها معرفی راه انداز دستگاه می باشد.

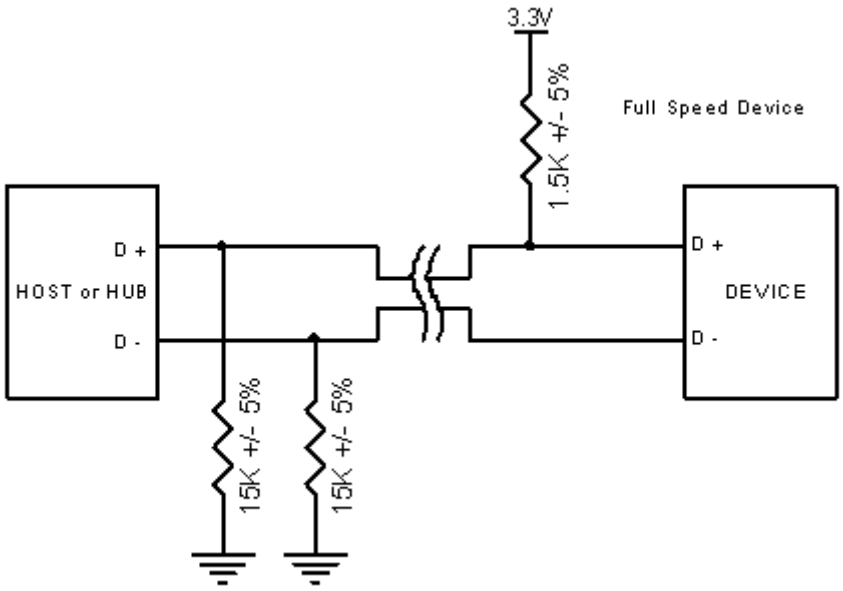
2-16-1- مراحل سر شماری

در طول مرحله سرشماری , دستگاه چهارحالت از شش حالت تعریف شده در مرجع خصوصیات راطی می کند که این حالت ها عبارتند از: روشن شدن, پیش فرض , آدرس و پیکربندی (دوحالت دیگر عبارتند از حالت های اتصال و بیکاری) . در هر حالت , دستگاه رفتار ها و قابلیت های تعریف شده متفاوتی دارد.

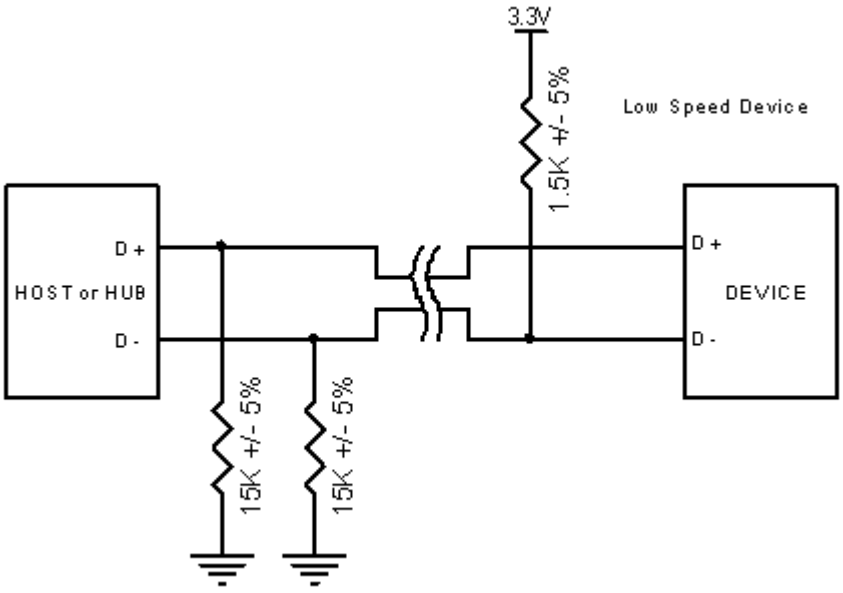
گام های زیر توالی احتمالی اتفاقاتی است که در طول مرحله سرشماری در ویندوز طی می شود. درهر حالت برنامه تراشه دستگاه نباید فرض کند که خواسته های سرشماری و رویداد ها به صورت مشخص و ویژه ای اتفاق می افتند. دستگاه باید همیشه آماده دریافت و پاسخ دادن به خواسته های کنترلی باشد.

۱. کاربر دستگاه را به پورت USB متصل می کند – یا اینکه کامپیوتر با دستگاهی که به پورت متصل است روشن می شود. این پورت ممکن است به هاب ریشه وصل شده باشد یا از طریق هاب های واسطی به میزبان متصل گردد. هاب , انرژی مورد نیاز دستگاه را تامین می کند و دستگاه وارد مرحله روشن شدن می شود.

۲. هاب دستگاه را تشخیص می دهد- هاب بر روی هر خط سیگنال (D- , D+) دارای یک مقاومت $15k\Omega$ است و دستگاه نیز بر روی خط D+ در دستگاه های سرعت بالا و بر روی D- در دستگاه های سرعت پایین دارای یک مقاومت $1.5\ k\Omega$ است. دستگاه های سرعت نهایی به عنوان دستگاه سرعت بالا متصل می شوند. وقتی که دستگاه به پورت اتصال می یابد , این مقاومت ها باعث می شوند ولتاژ خط دستگاه , یک شود و به این ترتیب هاب قادر شود دستگاه را تشخیص دهد. پس از تشخیص دستگاه , هنوز هاب انرژی مورد نیاز دستگاه را تامین می کند اما هیچ انتقالی از طریق USB صورت نمی گیرد , چون دستگاه آماده دریافت داده نمی باشد.



شکل ۱-۳۹- مدار اتصال USB در سرعت بالا



شکل ۱-۴۰- مدار اتصال USB در سرعت پایین

۳. میزبان از وجود دستگاه جدید آگاه می شود- هر هاب از یک مسیر ارتباطی وقفه ای برای گزارش رویدادهای هاب به میزبان استفاده می کند. وقتی که میزبان رویداد را دریافت کرد , از خواسته وضعیتی Get_Port_Status برای گرفتن اطلاعات بیشتر استفاده می کند. Get_Port_Status و خواسته های دیگری که در اینجا از آن ها نام برده می شود از خواسته های استاندارد

کلاس هاب هستند که همه هاب ها آن ها را می فهمند. اطلاعات باز گردانده شده مشخص می کند که چه وقت دستگاه جدید متصل شده است.

۴. برا ی هاب مشخص می شود که دستگاه سرعت بالا ست یا پایین – درست قبل از ریست کردن دستگاه , هاب با امتحان کردن ولتاژ موجود بر روی خط سیگنال تشخیص می دهد که دستگاه سرعت بالاست یا پایین . هاب این کار را با تعیین اینکه در هنگام بیکاری کدام خط سیگنال ولتاژ بیشتری دارد انجام می دهد. هاب این اطلاعات را در پاسخ به خواسته Get_Port_Status بعدی به میزبان می فرستد. USB1.x به هاب این اجازه را می دهد که بعد از ریست شدن , سرعت دستگاه را تشخیص دهد. اما در USB2.0 تعیین سرعت باید قبل از ریست شدن انجام شود.

۵. هاب دستگاه را ریست می کند – وقتی که میزبان دستگاه جدید را شناخت , کنترل کننده میزبان خواسته Set_Port_Feature را می فرستد که از هاب می خواهد پورت را ریست کند. هاب نیز خطوط داده USB دستگاه را حداقل برای مدت 10ms در حالت ریست قرار می دهد.ریست یک وضعیت ویژه است که در آن هر دو خط D- , D+ به صورت منطقی صفر هستند (در حالت عادی خطوط دارای حالت های منطقی متضاد هستند) هاب ریست را فقط به دستگاه های جدید می فرستد و دیگر دستگاه ها یی که روی باس هستند آن را نمی بینند

۶. در این مرحله میزبان از اینکه آیا دستگاه سرعت بالا , سرعت نهایی را هم پشتیبانی می کند یا نه آگاه می شود – تشخیص اینکه دستگاه از سرعت نهایی پشتیبانی می کند, از طریق دو حالت خاص برای سیگنال ها می باشد. در حالت J فقط خط D+ معادل یک منطقی است و در حالت K فقط خط D- یک منطقی می باشد. در طول مرحله ریست , دستگاهی که سرعت نهایی را پشتیبانی می کند, یک حالت K می فرستد. هاب سرعت نهایی این حالت را تشخیص داده و با یک سری از Kها وJ ها پاسخ ان را می دهد. هنگامی که دستگاه توالی KJKJKJ رادریافت کرد باید از حالت سرعت بالا خارج شود و بقیه ارتباط ها را با سرعت نهایی انجام دهد. اما اگر هاب هیچ پاسخی به حالت K فرستاده شده, نداد, دستگاه متوجه می شود که باید هنوز با همان سرعت بالا به ارتباط خود ادامه دهد. همه دستگاه های سرعت نهایی باید توانایی این را داشته باشند که به خواسته مرحله سر شماری با سرعت بالا پاسخ دهند.

۷. هاب مسیر سیگنال بین دستگاه و باس را ایجاد می کند

– میزبان با فرستادن خواسته Get_Port_Status متوجه می شود که دستگاه از حالت ریست خارج شده است . در داده ها, بیتی وجود دارد که وضعیت ریست شدن را مشخص می کند. در صورت لزوم, میزبان درخواست خود را تا موقعی که دستگاه از حالت ریست خارج شود تکرار می کند. وقتی که دستگاه از ریست خارج شد, دستگاه به حالت پیش فرض وارد می شود. در این زمان , رجیسترهای دستگاه USB در حالت ریست خود هستند و آماده اند که از طریق مسیر ارتباطی پیش فرض اندپوینت صفر به انتقال کنترلی بپردازند. حال دستگاه می تواند از طریق آدرس 00H خود ارتباط با میزبان را آغاز کند. دستگاه این امکان را دارد که 100mA از باس جریان بکشد.

۸. میزبان خواسته Get_Descriptor را به منظور

دریافت حداکثر اندازه پاکت مسیر ارتباطی پیش فرض می فرستد – میزبان خواسته را به آدرس صفر و اندپوینت صفر می فرستد . چون میزبان در هر لحظه تنها قادر است یک دستگاه را سر شماری کند, فقط یک دستگاه به این آدرس ارتباطی پاسخ خواهد داد حتی اگر چندین دستگاه به هاب متصل باشند. هشتمین بایت از توضیح دهنده دستگاه حاوی اندازه پاکت ماکزیممی است که توسط اندپوینت صفر پشتیبانی می شود. میزبان تحت ویندوز , خواستار ۶۴ بایت داده می شود اما پس از دریافت یک پاکت مرحله وضعیت را شروع می کند. پس از تکمیل مرحله وضعیت, میزبان تحت ویندوز از هاب می خواهد که دوباره دستگاه را ریست کند. مرجع خصوصیات ریست شدن در این مرحله را مشخص نکرده است , چون دستگاه همیشه باید آماده پاسخ به انتقال کنترلی و پاسخ به پاکت Setup بعدی باشد. اما ریست شدن دوباره , راهی مفید برای اطمینان از قرار گرفتن دستگاه در حالتی شناخته شده می باشد.

۹. میزبان آدرسی را به دستگاه نسبت می دهد – کنترلر

میزبان با فرستادن خواسته Set_Address آدرس منحصر بفردی را به دستگاه نسبت می دهد. دستگاه نیز خواسته را خوانده , تایید متقابل را ارسال کرده و ادرس جدید را ذخیره می نماید. هم اکنون دستگاه در حالت آدرس می باشد. از این لحظه به بعد همه ارتباطات از آدرس جدید استفاده خواهند کرد. این ادرس تا وقتی که دستگاه جدا شود یا ریست گردد یا سیستم خاموش شود فعال خواهد بود.

۱۰. میزبان از توانایی های دستگاه آگاه می شود – میزبان

خواسته Get_Descriptor را به آدرس جدید می فرستد تا این بار همه توضیح دهنده دستگاه را دریافت کند, توضیح دهنده یک ساختار داده ای است که حاوی ماکزیمم اندازه پاکت برای اندپوینت صفر , تعداد پیکربندی هایی

که دستگاه پشتیبانی می کند و اطلاعات اساسی دیگر درباره دستگاه می باشد که قبلا توضیح داده شده اند. میزبان دریافت اطلاعات از دستگاه را با فرستادن درخواست یک یاچند توضیح دهنده پیکربندی که در توضیح دهنده دستگاه تعیین شده اند , ادامه می دهد. دستگاه نیز در پاسخ به خواسته فرستاده شده این توضیح دهنده را به همراه همه توضیح دهنده های زیر شاخه اش انتقال می دهد. اما یک میزبان تحت ویندوز در دفعه اول فقط از ۹ بایت توضیح دهنده پیکر بندی استفاده می کند که حاوی طول مجموع همه توضیح دهنده های پیکربندی و توضیح دهنده های زیر شاخه آن می باشد. سپس ویندوز

دوباره درخواست توضیح دهنده پیکر بندی را تکرار می کند , این بار میزبان مجموع طول همه توضیح دهنده هایی را که فرستاده خواهد شد را دارد. حداکثر اندازه مجاز مربوط به طول توضیح دهنده ها FFH می باشد. به این ترتیب , دستگاه , توضیح دهنده پیکربندی را با توضیح دهنده های مدار واسط برای هر پیکر بندی , و به دنبال آن توضیح دهنده های اندپوینت مربوط به هر مدار واسط را خواهد فرستاد. اگر مجموع طول توضیح دهنده ها از FFH بایت بیشتر باشد, ویندوز ادامه توضیح دهنده ها را از طریق سومین درخواست بازیابی می کند. هر توضیح دهنده با طول و نوع آن آغاز می شود, تا میزبان را قادر سازد تا اجزای مختلف آن را تجزیه کند.

۱۱. میزبان راه انداز دستگاه را تشخیص داده و بارگذاری می کند – پس از اینکه میزبان اطلاعات موجود در توضیح دهنده ها را گرفت , به دنبال بهترین انتخاب برای راه انداز دستگاه خواهد گشت تا ارتباطات دستگاه را رهبری کند برای انتخاب راه انداز, ویندوز سعی می کند که اطلاعات ذخیره شده در فایل های INF سیستم را با شماره مشخصه های فروشنده و محصول که از دستگاه بازیابی شده اند تطبیق دهد. اگر راه انداز مناسبی پیدا نشد , ویندوز به دنبال انتخابی بین راه انداز های کلاس , زیر کلاس و پروتکل که از دستگاه دریافت کرده است خواهد گشت. پس از آنکه سیستم عامل راه انداز را مشخص کرده و بار گذاری نمود معمولا راه انداز از دستگاه خواهد خواست که دوباره توضیح دهنده ها یا توضیح دهنده های ویژه کلاس را بفرستد.

۱۲. راه انداز دستگاه یک پیکر بندی را انتخاب می کند – پس از شناختن دستگاه از طریق توضیح دهنده ها , راه انداز دستگاه با فرستادن خواسته Set_Configuration درخواست شماره پیکر بندی خاصی را می کند. بسیاری از دستگاه ها فقط از یک پیکر بندی پشتیبانی می کنند. اگر دستگاه چندین پیکر بندی داشته باشد , راه انداز بر اساس اطلاعاتی که از دستگاه دارد و یا کاری که کاربر می خواهد انجام دهد, پیکربندی خاصی را انتخاب خواهد کرد. دستگاه نیز این پیکر بندی را خوانده و خود را با آن تنظیم می کند. هم اکنون دستگاه در حالت پیکر بندی قرار دارد و مدار های واسط دستگاه فعالند.

هم اکنون دستگاه آماده استفاده است.

دو حالت دیگر دستگاه , حالت اتصال و حالت بیکاری , ممکن است در هر زمانی اتفاق بیفتند.

- **حالت اتصال**

اگر هاب , توان پورت را تامین نکند, دستگاه در حالت اتصال قرار خواهد گرفت . این اتفاق ممکن است در مواقعی که هاب , کشیده شدن جریان بیش از اندازه را تشخیص دهد یا اینکه میزبان از هاب بخواهد که به پورت توان ندهد , رخ دهد. در این حالت دستگاه و میزبان نمی توانند ارتباط برقرار نمایند.

- **حالت بیکاری**

اگر به مدت ۳ میلی ثانیه هیچ فعالیتی روی باس از جمله علا مت های شروع فریم وجود نداشته باشد, دستگاه وارد حالت بیکاری می شود, در این حالت دستگاه باید کمترین انرژی را از باس بگیرد . هر دو دسته دستگاه های پیکر بندی شده و نشده باید این حالت را پشتیبانی کنند.

1-16-3- سرشماری یک هاب

هاب ها نیز جزء دستگاه های USB به حساب می آیند و میزبان هاب های جدید متصل شده را به روشی مشابه با سرشماری دستگاه ها , شناسایی می کند. اگر به هاب دستگاهی دیگر متصل شده باشد, میزبان پس از هاب , همه دستگاه ها را سر شماری می کند.

1-16-4- جداکردن دستگاه

وقتی کاربر دستگاه را از باس جدا می سازد, هاب پورت دستگاه را غیر فعال می کند. میزبان از جداشدن دستگاه از طریق مقاومت موجود بر روی خطوط سیگنال آگاه می شود و خواسته Get_Port_Status را برای تشخیص رویداد رخ داده شده می فرستد . ویندوز سپس دستگاه را از لیست نمایش رهبری دستگاه حذف خواهد کرد و آدرس آن برای دستگاه های جدید قابل استفاده خواهد بود.

۲-۰- مقدمه

این آموزش سعی درتوضیح چگونگی نوشتن یک درایورساده دستگاه در محیط WinNt را دارد.منابع و آموزش های بسیاری جهت نوشتن درایور دستگاه وجود دارد بهر حال آنها قدری برای نوشتن و کار در محیط گرافیکی Windows کمیاب هستند که جستجوی اطلاعات برای آغاز ساخت درایور را قدری دشوار می کنند.شاید تصور کنید که در صورت وجود یک آموزش چرا به آموزش های بسیاری نیاز هست؟ پاسخ اینست که معمولا اطلاعات بیشتر مخصوصا در هنگام آغاز درک یک مفهوم خیلی مثمر ثمر هستند.معمولا مطالعه اطلاعات یکسان از مناظر مختلف خیلی مفید است افراد متفاوت می نویسند و اطلاعات معین را بصورت های مختلف شرح میدهند که به میزان آشنایی آنها به اطلاعات و همچنین تصور آنها از یک منظر ولزوم چگونگی شرح آن بستگی دارد..بعضی مواقد ایراد و موردهای محذوف ، بعضی مواقع مواردی انجام میشوند که ضروری نیستند و بعضی مواقع هم اطلاعات غلط و یا فقط ناقص هستند موضوع اینست که کسانی که سعی در یادگیری نوشتن درایور ها دارند نباید همین جا و جاهای دیگرمتوقف شوند بلکه همیشه در جستجو مثال ها و خرده کدهای مفید و تحقیق در زمینه موارد مختلف باشند.این آموزش چگونگی نوشتن یک درایورساده و بارگذاری و تخلیه بار بصورت پویا وسرانجام تغییرات بصورت دلخواه کاربر را آموزش خواهد داد.

۲-۱- ساخت یک درایور ساده دستگاه

۲-۱-۱- زیر سیستم (Sub System)چیست؟

پیش ازشرح چگونگی نوشتن درایور احتیاج به تعریف اولین زمینه داریم.نکته آغازین این مقاله کامپایلر خواهد بود ، کامپایلر و پیوند دهنده ، باینری را در قالبی تولید میکنند که سیستم عامل توانایی درک آن را داشته باشد.در Windows این قالب "PE" مخفف "Portable Excutable" میباشد.در این قالب مفهومی هست که زیر سیستم نامیده می شود.یک زیر سیستم بهمراه اختیارات دیگری که دراطلاعات PE Header مشخص شده چگونگی بارگذاری یک اجراشونده که خود نیز شامل یک اشاره گر ورود به باینری است را توصیف میکند.افراد بسیاری از VC++ IDE برای ساده کردن ساخت یک پروژه با پیش تنظیمات خط دستور کامپایلر استفاده می کنند.این دلیل اینست که چرا بسیاری از افراد با این مفهوم آشنایی ندارند حتی با اینکه قبلا در نوشتن برنامه های کاربردی از آنها استفاده کرده بودند.آیا تا بحال بورد کاربردی نوشته اید؟ آیا تا بحال برنامه کاربردی با رابط گرافیکی در Windowsنوشته اید؟ تمامی اینها زیر سیستم مختلف در Windows میباشند.هر دو اینها PE Binary را با استفاده از اطلاعات زیر سیستم تولیدخواهند نمود.این همچنین دلیل بر اینست که چرا بورد کاربردی از "Main"استفاده میکند در جایی که برنامه کاربردی Windows از "WinMain" استفاده میکند.وقتی که شما این پروژه ها را انتخاب میکنید،VC++ بسادگی با /SUBSYSTEM:CONSOLE یا /SUBSYSTEM:WINDOWS یک پروژه را میسازد.اگر تصادفی پروژه اشتباه (Console or Windows)را انتخاب کردید میتوانید بجای ایجاد دوباره پروژه براحتی از منوی پیوند دهنده (Linker) تنظیمات درست را انتخاب کنید. یک نکته در تمامی این موارد وجود دارد :درایوری که از زیر سیستم متمایزی استفاده کند "Native" خوانده میشود.

۲-۱-۲- "Main" درایور:

بعد از اینکه کامپایلر با تنظیمات مناسب راه اندازی شد، بهترین کار فکر کردن درمورد نقطه ورودی (Entry Point) درایور میباشد.درابتدا ممکن است قدری زیر سیستم طول بکشد. "Native"همچنین میتواند بعنوان اجرا برنامه کاربردی در مد کاربر استفاده شود که یکه نقطه ورود تعریف میکند که "NtProcessStartup" خوانده میشود.این نوع "Default" اجرا پذیر است که در هنگام تعیین بعنوان "Native" در یک طریق "WinMain"یا "Main" که در هنگام ساخت یک برنامه کاربردی توسط پیوند دهنده پیدا میشوند ساخته میشود.شما میتوانید نقطه ورودی قراردادی را به دلخواه براحتی از طریق بکارگیری-" entry:DriverEntry" از منوی پیوند دهنده تغییر دهید،اگر بدانیم که برای درایور به این تغییرات احتیاج

داریم ، نقطه ورودی را مینویسیم که حاوی لیست پارامترها باشد و انواع تطابق درایور را بازگرداند.این پروسه درایور را پس از نصب بارگذاری کرده و بعنوان درایور به سیستم میشناسند.

اسمی را که استفاده می کنیم هر چیزی می تواند باشد.اگر بخواهیم می توانیم BufferFly() را انتخاب کنیم،"DriverEntry" کلمه ای است که معمولا توسط درایور نویسان و مایکروسافت بعنوان نقط ورودی استفاده میشود در واقع ما عبارت "-entry:DriverEntry" را به خط دستور پیوند دهنده اضافه می کنیم ، اگر از DDK استفاده می کنید هنگامی که"Driver" را بعنوان اجراشونده برای ساخت تعیین کنید این تنظیمات بصورت خودکار انجام می شود. DDK دارای محیطی است که تنظیمات پیش فرض را در دایرکتوری Make دارد، که سبب می شود به سادگی برنامه کاربردی با تنظیمات پیشفرض بسازید. درایور نویس واقعی می تواند به سادگی تنظیمات را در صورت نیاز از دایرکتوری Make دوباره تغغیر دهد و یا چشم پوشی کند، این دلیلی است که چرا "Driver Entry" تبدیل به اسمی نسبتا "رسمی" برای نقطه ورودی درایور می شود.

به خاطر داشته باشید ، DLL های کامپایل شده "Windows" را بعنوان زیر سیستم تعیین می کنند، اما آنها همچنین یک گزینه اضافی (Additional Switch) دارند که DLL/خوانده می شوند. یک گزینه دیگر نیز برای درایور ها وجود دارد: DRIVER:WDM/(که همچنین بعنوان Native در پس زمینه حاضر می شود)، همچنین DRIVER:UP/ به این معنی است که این درایور توانایی بارگذاری در سیستم های چندین پردازنده را ندارد.

پیوند دهنده آخرین باینری را می سازد ، و طبق تنظیمات PE Header و تلاش باینری برای بارگذاری (اجرا بعنوان EXE از طریق بارگذار، بارگذاری توسط LoadLibrary یا تلاش برای بارگذاری بعنوان درایور) رفتار سیستم بارگذار را مشخص خواهد کرد.سیستم بارگذاری تلاش می کند برخی از سطح های شناسایی را انجام دهد ، نمونه ای بارگذاری می شود تا سیستم مواردی را که درایور انتظار بارگذاری شدن آنها را در این حالت دارد شناسایی کند. بطور مثال : در برخی موارد بعید است که کد آغازین به باینری، قبل از اینکه به نقطه ورودی دست یابیم اضافه شود.(WinMainCRTStartup ، WinMain را فرخوانی کند ، بطور مثال برای راه اندازی CRT)کار شما نوشتن کاربرد بر پایه این که چگونه می خواهید بارگذاری شود و سپس تنظیم، تنظیمات صحیح در پیوند دهنده است بنابراین پیوند دهنده می داند چگونه باینری را درست بسازد.منابع مختلفی در مورد قالب PEو جزئیات آن وجود دارد که اگر شما علاقمند به کار بیشتر در این زمینه هستید باید بررسی کنید.

تنظیمی که ما برای پیوند دهنده تنظیم خواهیم کرد با این عبارت خاتمه می پذیرد:

```
/SUBSYSTEM:NATIVE /DRIVER:WDM -entry:DriverEntry
```

۲-۱-۳- قبل از ساخت "DriverEntry"

مواردی هست که قبل از نوشتن "DriverEntry" لازم است مرور کنیم. من میدانم که بیشتر افراد دوست دارند سریع سراغ نوشتن درایور بروند بعد هم کارکردن آن را ببینند،معمول ترین کار برنامه نویسی این است که فقط کدی را برداشته ، تغییراتی را اعمال کنیم ، کامپایل کرده و تست کنیم ، اگر زمانی را که تازه برنامه نویسی ویندوز را یاد می گرفتید را بخاطر بیاورید ، آن زمان هم دقیقا همین کار را انجام می دادید.برنامه شما درست کار نمیکرد ، قفل و یا اصلا ناپدید می شد، سرگرمی جالبی بود و مسلما چیزهای زیادی را هم یاد گرفتید ،اما باید بدانید که داستان در مورد درایور یک کم فرق می کند،ندانستن اینکه چکار کنین ممکنه سیستم را با نمایش صفحه آبی خاتمه بده و اگر درایور در حالت بوت(راه اندازی اولیه سیستم) بارگذاری شده باشد و کدها اجرا شده باشند شما یک مشکل جدید خواهید داشت، درخوشبینانه ترین حالت،شما میتوانید سیستم را در مد امن(Safe Mode) راه اندازی کنید و یا تنظیمات سخت افزاری را به حالت اول برگردانید،مواردی کمی هست که قبل از نوشتن درایور بهتر است مرور کنیم به جای اینکه یاد بگیریم بعد از انجام کار چه کارهایی انجام دهیم.

اولین قانون کلی : درایوری را برندارید و با تنظیمات دلخواه خودتان کامپایل نکنید.اگر کار درایور را درک نمی کنید و یا نمی دانید چطور به درستی برنامه ریزی می شود احتمالا دچار مشکلی می شوید.درایورها میتوانند کل یک سیستم را دچار مشکل کنند، معمولا ایراداتی دارند که در اغلب موارد ظاهر نمی شوند به غیر ازپیشامدهایی که به ندرت رخ میدهند ، برنامه های کاربردی نیز میتوانند ایرادات یکسانی داشته باشند اما نه علت های ریشه ای ، بعنوان مثال : مواقعی پیش می آید که امکان دسترسی به حافظه قابل دسترسی نیست،اگر طریقه کار حافظه مجازی را بدانید ، شما می دانید که سیستم عامل مقداری از حافظه را حذف می کند تا به جایی که نیاز هست برساند و این دقیقا دلیل آن است که چگونه برنامه ها با حجم حافظه مورد نیاز بیشتر ازحافظه موجود ، توانایی اجرا پیدا می کنند.در مواردی که قسمتی از حافظه توانایی فراخوانی از دیسک را ندارد ، آن دسته درایورهایی که با حافظه کار می کنند فقط حق استفاده از این دسته حافظه ها را دارند .

کجا را برای این اختصاص دهیم ؟ خوب ! اگر به درایوری که تحت این محدودیت کار می کند اجازه دسترسی به حافظه قابل دسترسی دهیم ،احتمال دچار مشکل شدن به خاطر اینکه سیستم عامل معمولا درصدد هست که تمامی حافظه ها را در دسترسی نگه دارد خیلی ضعیف هست.اگر کاربردی که در حال اجرا است را ببندیم ، این امکان وجود دارد که بازهم در حافظه بماند.برای مثال:این دلیلی است برای اینکه ایراداتی مثل این معمولا ناشناخته می مانند (حتی با اینکه تلاش بسیاری می کنید مثل: بازبین کننده درایور) و در بعضی مواقع هم

ممکن است شناسایی شوند.مواقعی هم که اتفاق می افتد ، اگر مفاهیم پایه مثل این موضوع را ندانید دچار سردرگمی می شوید که مشکل چیست و چگونه بر طرفش کنید؟

مفاهیم بسیار زیادی که توضیح داده خواهند شد در این مقاله وجود دارند ، در مورد IRQL فقط در حدود بیست مقاله وجود دارد که می توانید در MSDN آنها را پیدا کنید ، همچنین مقاله بزرگی در مورد IRP وجود دارد.من قصد توضیح تک تک جزئیات را ندارم ، هدف توضیح خلاصه ای در مورد مسایل پایه و همچنین متوجه کردن شما به جاهایی است که می توانید اطلاعات بیشتری بدست آورید.دانستن حداقل بعضی از مفاهیم و همچنین آشنایی با بعضی معانی مرتبط با آنها ، قبل از نوشتن درایور ضروری است.

IRQL -4-1-2 چیست؟

IRQL را به عنوان "**I**nterrupts **R**e**Q**uests **L**evel" می شناسند.پردازنده کد را در یک رشته در یک IRQL خاص اجرا می کند.IRQL پردازنده ، ضرورتا در تعیین چگونگی وقفه یک رشته کمک می کند.رشته فقط می تواند توسط کدی که نیاز به IRQL بالاتر با یک پردازنده یکسان جهت اجرا دارد ، دچار وقفه شود .وقفه هایی که به IRQL یکسان یا پایین تر احتیاج دارند پنهان می مانند بنابراین تنها وقفه هایی که به IRQL بالاتر احتیاج دارند قابل پردازش اند.در سیستم های چند پردازنده ، هر پردازنده در IRQL خود مستقلانه کار می کند.

IRQL چهار سطح دارد که معمولا با آنها سر و کار داریم که عبارتند از:"Dispatch","APC","Passive"و "DIRQL" . هسته ثبت شده API در MSDN دارای یک یادداشت است که سطح مورد نیاز IRQL را برای اجرا ، به جای استفاده از API مشخص می کند.هر چه سطح IRQLمورد استفاده بالاتر باشد API کمتری مورد استفاده قرار می گیرد. اسناد MSDN سطح IRQL پردازنده را برای اجرا ، هنگامی که نقطه ورود خاصی خوانده می شود تعریف می کنند. "DriverEntry" برای مثال در سطح PASSIVE_LEVEL خوانده می شود.

PASSIVE_LEVEL-1-4-1-2

پایین ترین سطح IRQL می باشد،هیچ وقفه ای پنهان نمی ماند و سطحی است که یک رشته هنگامی که مد کاربری در حال اجرا است ، اجرا می شود.حافظه فراخوانده شده ،قابل دسترسی است.

APC_LEVEL -2-4-1-2

در پردازنده ای که در این سطح کار می کند، تنها وقفه های APC پنهان می مانند.سطحی است که در فراخوانی های رویه ناهمگام(Asynchronous Procedure Calls) رخ می دهد.حافظه فراخوانی شده همچنان قابل دسترسی است.هنگامی که APC رخ دهد ، پردازنده ارتقاء سطح به APC می دهد و همچنین از رخداد سایر APC ها جلوگیری می کند.یک درایور می توانند IRQLخود را تغییر سطح به APC (و یا سایر سطوح)دهد به جای اینکه خود را با APC ها منطبق کند.برای مثال:APC نمی تواند پاسخ دهد اگر شما همچنین در سطح APC باشید.بعضی از API ها هستند که بعلت غیر فعال بودن APC نمی توانند در سطح APC فراخوانده شوند.که در بهترین حالت ممکن تکمیل بعضی ازAPC's I/O را غیر فعال کند.

DISPATCH_LEVEL -3-4-1-2

پردازنده ای که در این سطح کار می کند سطح DPC و پوشش کمتری دارد.حافظه فراخوانی شده قابل دسترسی نیست وتنها حافظه هایی قابل دسترسی اند که فراخوانی نشده باشند.اگر شما در سطح Dispatch کار می کنید API که استفاده می کنید به مقدار زیادی کاهش پیدا می کند و به این خاطرست که فقط از حافظه های فراخوانی نشده می توانید استفاده کنید.

DIRQL(Device IRQL) -4-4-1-2

معمولا درایورهای سطح بالا نمی توانند با این سطح IRQL کار کنند ، اما تمامی وقفه ها در این سطح و یا پایین تر پنهان شده و آشکار نمی شوند.این دقیقا محدوده IRQL ها می باشد واین روشی برای تعیین اولویت دستگاه ها نسبت به یکدیگراست.

در این درایور،اساسا ما بر روی PASSIVE_LEVEL کار می کنیم،بنابراین نگران مسئله ای نیستیم .بهرحال مهم است که بدانید IRQL چیست؟ اگر هنوز قصد نوشتن درایور دارید.

IRP -5-1-2 چیست؟

"IRP"مخفف "I/O Request Packet" است و در پشته درایور از هر درایور به درایور دیگری عبور می کند.این ساختار داده به درایورها اجازه ارتباط با یکدیگر را و همچنین تقاضای انجام کار توسط درایور را می دهد.مدیریت I/O و یا درایوری دیگری IRP را ساخته و آن را به درایور شما می رسانند.IRP شامل اطلاعاتی از عملیات تقاضا شده است.

شرح و طریقه استفاده از IRP از سادگی به پیچیدگی شروع می شود.بنابراین ما فقط بصورت کلی توضیح خواهیم داد ، IRP چه معنایی برای شما خواهد داشت.

نویسنده : حسین

پایمرد

IRP همچنین شامل یک لیست از "sub-requests" است که با نام "IRP Stack Location" مشخص می شوند. هر درایور در پشته مخصوص به دستگاهش معمولا "sub-request" خود را دارد که وقفه IRP را مشخص می کند.این ساختار داده "IO_STACK_LOCATION" می باشد و در MSDN شرح داده شده است.

مقایسه IRP با "IO_STACK_LOCATION" ، شاید سه دسته افراد باشند که کاری متفاوت دارند مانند نجار، لوله کش و جوشکار ، اگر قصد ساختن یک خانه را داشته باشند.همگی لباس کار و جعبه ابزاری که شامل مته برقی و سایر ابزار است را دارند.همه این ابزار و لباس کار برای ساخت یک خانه می توانند IRP باشند.هر یک ، کار منحصر به فردی دارند که سبب انجام کار اصلی می شود.بطور مثال لوله کش باید تصمیم بگیرد که چه مکان هایی لوله قرار دهد، چه مقدار لوله دارد و غیره. کار هر قسمت در IO_STACK_LOCATION تفسیر می شود مثلا لوله کشی برای لوله کش مشخص می شود. نجار چارچوبهای خانه را می سازد و جزئیات کارش در IO_STACK_LOCATION می باشد.بنابراین موقعی که کل IRP تقاضای ساخت خانه می باشد هر فرد کار مشخص در پشته افراد دارد که توسط IO_STACK_LOCATION تعیین می شود که این عمل انجام شود.هنگامی که هر فرد کار مربوط به خود را انجام داد کارIRP انجام می شود.

درایور دستگاهی که ما قصد ساخت آن را داریم به این پیچیدگی نیست بلکه اساسا یک درایور در پشته میباشد.

6-1-2- روال ساخت The DriverEntry

مسایل زیادی برای توضیح دادن هست. بهرحال ، فکر کنم وقت آن رسیده که ، نوشتن درایور را آغاز کنیم و چگونگی کار را توضیح دهیم.توضیح و تفسیر تئوری کار و یا حتی چگونگی کار کدها بدون انجام کاری مشکل است.شما بقدری تلاش و تجربه برای محقق کردن این هدف احتیاج دارید.

نمونه از یک DriverEntry :

```
NTSTATUS DriverEntry ( PDRIVER_OBJECT pDriverObject,
PUNICODE_STRING pRegistryPath ) ;
```

DRIVER_OBJECT یک ساختار داده است که این درایور را معرفی می کند.روال نقطه ورودی درایور ، برای بررسی سایر تقاضای I/O آن را با بقیه نقطه ورودی های درایور مشغول می کند.این شیئ همچنین به DEVICE_OBJECT که یک ساختار داده می باشد و یک دستگاه خاص را معرفی می کند، اشاره می کند.یک درایور تنها ممکن است خود به تنهایی با دستگاه های بسیاری کار کند و به همین دلیل

DRIVER_OBJECT یک لیست پیوندی نگهداری می کند که به دستگاههایی که با این درایور کار می کند اشاره دارد.به سادگی ما تنها یک دستگاه خواهیم ساخت.

مسیر رجیستری "Registry Path" یک رشته است که به مکانی در رجیستری که اطلاعات مربوط به هر درایور ذخیره شده است ، اشاره دارد. درایور میتواند از این مکان برای ذخیره اطلاعات خاص مربوط به درایور استفاده کند.

قسمت بعدی قرار دادن کارهای بسیاری در روال نقطه ورودی درایور می باشد.اولین کاری که انجام خواهیم داد ساختن دستگاه می باشد.شاید تعجب کنید که چگونه دستگاه بسازیم و چه نوع دستگاهی باید بسازیم.این مساله به این خاطر است که معمولا درایور ها به یک سخت افزار وابسته هستند اما مسئله این نیست.رایور های متنوعی هستند که در سطح های مختلف کار میکنند ، همه درایور ها با سخت افزارمستقیما کار نمی کنند و یا درگیر نیستند.معمولا، شما از یک پشته از درایورها نگهداری می کنید که هر یک کار خاصی انجام می دهند.درایور با بالاترین سطح ، درایوری است که با مد کاربری در ارتباط است و سایر درایورها با سطح پایین ، درایورهایی هستند که فقط با سایر درایورها و سخت افزار در ارتباط هستند.درایورهای بسیاری مانند درایور شبکه ، درایور نمایش ، درایور فایل های سیستم و غیره وجود دارد و هریک پشته درایور خاص خود را دارند.هر مکان در پشته به درخواست های ساده و عمومی برای کار درایور سطح پایین مجزا می شود.درایور های سطح بالا خودشان با مد کاربری در ارتباط هستند مگرآنهایی که دستگاه خاصی با بدنه مخصوصی هستند(مانند درایورهای نمایش).آنها مانند سایر درایورها رفتار می کنند و فقط در تکمیل عملیات مختلفی درگیر می شوند.

بعنوان مثال ، درایو هارد دیسک را در نظر بگیرید ، درایوری که با مد کاربری در ارتباط است مستقیما با سخت افزار در تماس نیست.درایورهای سطح بالا حقیقتا فایل های سیستم و محل قرار گیری کارها را مدیریت می کنند.سپس به درایور سطح پایین که ممکن است با سخت افزار در ارتباط باشد جایی از دیسک سخت را که باید خوانده و یا نوشته شود معرفی می کند.لایه دیگر وجود دارد که محل فیزیکی که باید خوانده و یا نوشته شود را به درایوری که واقعا با سخت افزار در ارتباط است معین کرده و جواب درخواست را به سطح بالاتر بر می گرداند.سطح بالا آنها را بصورت داده های فایل تفسیر می کند ولی سطح پایین بی تفاوت فقط تقاضا ها را بررسی می کند ، تا آنجا که چه موقع باید سکتوری خوانده شود ، هد خواندن/نوشتن در کجای هارد دیسک قرار دارد.و تنها می تواند تصمیم بگیرد که چه سکتوری خوانده شود هرچند هیچ تفسیری در مورد اینکه چه نوع داده می باشد ندارد.

بهتراست نگاهی به اولین قسمت DriverEntry بیندازیم:

```
NTSTATUS DriverEntry(PDRIVER_OBJECT  pDriverObject, PUNICODE_STRING  pRegistryPath)
{   NTSTATUS NtStatus = STATUS_SUCCESS;
```

```
UINT uiIndex = 0;
PDEVICE_OBJECT pDeviceObject = NULL;
UNICODE_STRING usDriverName, usDosDeviceName;
DbgPrint("DriverEntry Called \r\n;("
RtlInitUnicodeString(&usDriverName, L"\\Device\\Example;("
RtlInitUnicodeString(&usDosDeviceName, L"\\DosDevices\\Example ;("
NtStatus = IoCreateDevice(pDriverObject,&  usDriverName ,
                FILE_DEVICE_UNKNOWN,
                FILE_DEVICE_SECURE_OPEN ,
                FALSE, &pDeviceObject)
```

اولین موردی که باید توجه کنید تابع DbgPrint است.مانند "printf" کار میکند و پیغامی را در Debugger و یا پنجره خروجی Debug ظاهر می کند.شما می توانید ابزاری را بنام "DBGVIEW" و تمامی اطلاعاتی را که در آن پیغام ظاهر می شود را از طریق این آدرس پیدا کنید:

http://www.sysinternals.com

همچنین توجه خواهید کرد که تابعی بکار بردیم به نام "RtlInitUnicodeString" که معمولا UNICODE_STRING ساختار داده را راه اندازی می کند.این ساختار داده معمولا شامل سه ورودی می باشد.اولین ورودی اندازه رشته unicode می باشد دومی حداکثر اندازه که رشته unicode می تواند باشد و سومی اشاره گری به رشته unicode است.این برای شرح رشته unicode ومعمولا در درایورها استفاده می شود.یک مساله که راجع UNICODE_STRING لازم است بدانیم این است که احتیاجی نیست که با NULL خاتمه یابد تا وقتی که پارامتر اندازه ای در ساختار باشد.معمولا این مساله برای افرادی که تازه درایور نویسی می کنند و می پندارند که با NULL خاتمه می یابد مشکل ساز خواهد بود و صفحه آبی درایور . اغلب رشته های unicode معمولا با NULL خاتمه نمی یابند.پس بهتر است متوجه این مسئله باشید.

دستگاه ها نیز مثل سایر چیزها نام دارند.آنها معمولا \Device\<somename> و این رشته است که در IoCreateDevice تعیین می کنیم.رشته دوم ،"\DosDevice\Example" قبل از اینکه در درایور استفاده شود ، قرار خواهیم داد.در IoCreateDevice ، در اشاره گر به رشته unicode که می خواهیم درایور را فراخواند خود درایور را تعیین می کنیم.و در نوعی از درایور "UNKNOWN" که با هیچ نوع خاص از دستگاه در ارتباط نیست را تعیین می کنیم و همچنین در اشاره گری که دستگاه جدید ساخته شده را دریافت

می کند تعیین می کنیم.تمامی پارامترها در خود IoCreateDevice با جزئیات بیشتر توضیح داده شده است.

دومین پارامتر که ۰ قرار می دهیم برای تعیین تعداد بایتی است که برای بسط دستگاه استفاده می کنیم ، در اصل این یک ساختار داده می باشد که درایور نویس برای هر دستگاه قرار می دهد.از این طریق در عوض ذخیره فوری اطلاعات می توان اطلاعات تعیین شده در دستگاه را توسعه و اضافه نمود و مفهوم دستگاه را بسازیم.

حالا که \Device\Example درایور دستگاه را با موفقیت ساختیم.احتیاج داریم که Driver Object را راه اندازی کنیم تا در هنگام ساخت یک تقاضا قطعی به داخل درایور فراخوانی شود،این تقاضا ها درخواست های بزرگ IRP هستند همچنین درخواست های خرد نیز هستند که در واقع درخواست های تابع می باشند و درمکان پشته IRP یافته خواهند شد.

کد زیر که حاوی درخواست های زیادی است:

```
for(uiIndex = 0; uiIndex < IRP_MJ_MAXIMUM_FUNCTION; uiIndex(++

        pDriverObject->MajorFunction[uiIndex] =
Example_UnSupportedFunction;

        pDriverObject->MajorFunction[IRP_MJ_CLOSE]          =
Example_Close;

      pDriverObject->MajorFunction[IRP_MJ_CREATE]          =
Example_Create;

      pDriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL]   =
Example_IoControl;

      pDriverObject->MajorFunction[IRP_MJ_READ]          =
Example_Read;

      pDriverObject->MajorFunction[IRP_MJ_WRITE]          =
USE_WRITE_FUNCTION;
```

از دستورات Create,Close,IoControl,Read و Write استفاده کردیم.اما به کجا رجوع می کنند؟ هنگام ارتباط با یک برنامه مد کاربری API مسیر خاصی را به درایور فرا میخواند و پارامترها را تععین می کند.

- `CreateFile -> IRP_MJ_CREATE`
- `CloseHandle -> IRP_MJ_CLEANUP & IRP_MJ_CLOSE`

- `WriteFile` -> `IRP_MJ_WRITE`
- `ReadFile`-> `IRP_MJ_READ`
- `DeviceIoControl` -> `IRP_MJ_DEVICE_CONTROL`

برای توضیح ، یکی از تفاوت ها IRP_MJ_CLOSE است که در مفهومی که در حال پردازش است فراخوانی نمی شود.اگر به پردازشی که با مرتب شدن در ارتباط است نیاز دارید باید IRP_MJ_CLEAN_UP را اجرا کنید.

پس همان گونه که مشاهده نمودید هنگامی که یک برنامه مد کاربری از این توابع استفاده کند به داخل درایور فراخوانده می شود.شاید تعجب کنید که چرا API مد کاربری آن را "File" می داند در حالی که هیچ معنایی به عنوان "File" ندارد.درواقع این API ها توانایی ارتباط با هر دستگاهی را که در مد کاربری باشد را دارند. آنه تنها برای دسترسی به فایل نیستند درپایان این مقاله مد کاربری را نوشته ایم که با درایور در ارتباط است و به سادگی CreateFile, WriteFile, CloseHandle انجام می
دهند.USE_WRITE_FUNCTION ثابتی است که در ادامه توضیح خواهیم داد.

کد بعدی که قصد معرفی داریم ، تابع بی بار کننده درایور می باشد:

```
pDriverObject->DriverUnload =  Example_Unload;
```

شما می توانید این تابع را حذف کنید اما اگر قصد خالی کردن بار درایور را دارید باید تعیین شده باشد و در صورتی که این تابع تعیین نشده باشد پس از یک بار بارگذاری درایور سیستم اجازه خالی کردن بار نمی دهد.

کدی بعدی مستقیما DEVICE_OBJECT را به استفاده می گیرد نه DRIVER_OBJECT هردوی این ساختار داده ها به لحاظ اینکه با حرف "D" شروع می شوند و به کلمه "OBJECT" خاتمه می یابند گیج کننده اند و معمولا سبب اشکال می شوند.

```
pDeviceObject->Flags |= IO_TYPE;

pDeviceObject->Flags &= (~DO_DEVICE_INITIALIZING);
```

بسادگی flag را تنظیم کردیم ،"IO_TYPE" در واقع یک ثابت است که نوع I/O را که ما می خواهیم انجام دهیم را تعیین می کند.(من در example.h تعیین کردم)

"DO_DEVICE_INITALIZING" به مدیریت I/O اعلام میکند که دستگاه راه اندازی شده است و درخواست I/O به درایور نفرستد.برای دستگاه هایی که در مفهوم "DriverEntry" ساخته شده اند احتیاجی به این نیست تا زمانی که مدیریت I/O ، "DriverEntry" Flag که انجام شده است را حذف کند.به هر حال

اگر هر دستگاهی را با هر نوع تابعی خارج از "DriverEntry" بسازید باید flag را برای هر دستگاهی که با IoCreateDevice می سازیم ، حذف کنیم.این flag معمولا توسط تابع IoCreateDevice معین می شود.ما در اینجا آن را جهت آموزش حذف کردیم نه به این دلیل که به آن نیاز نداریم.آخرین قسمت از درایور ما از هر دو رشته unicode که در بالا تعریف کردیم استفاده می کند. "\Device\Example" و "\DosDevice\Example"

```
IoCreateSymbolicLink(&usDosDeviceName, &usDriverName);
```

"IoCreateSymbolicLink" همان طور که از اسم تابع مشخص است ، یک "Symbolic Link" در مدیریت شیئ می سازد.برای مشاهده مدیریت شیئ احتیاج به ابزار "WINOBJ" دارید که از آدرسی که قبلا نیز معرفی کردم می توانید دانلود کنید.یک پیوند نمادی "Dos Device Name" را بصورت "NT Device Name" ترسیم می کند.

فروشنده های مختلف درایور های مختلفی دارند و هر درایور نیز اسمی دارد.شما نمی توانید دو درایو با یک اسم دستگاه Nt یکسان داشته باشید.به فرض ، شما یک حافظه قابل حمل دارید که خود را با یک حرف جدید قابل استفاده مثلا E: به صورت درایو به سیستم می شناساند.اگر شما این حافظه را جدا کنید و حرف E: را برای درایو شبکه استفاده کنید هیچ فرقی نمی کند و سیستم همان طور که با حافظه قابل حمل صحبت می کرد با درایو قابل حمل نیز صحبت می کند ، برای آنها فرقی نمی کن که E: دیسک فلاپی ، درایو CD-ROM ، حافظه قابل حمل و یا درایو شبکه است.چگونه ممکن است؟ خوب ، درایور تنها نیاز دارد که تقاضاها را بتواند تفسیر کند و فرقی نمی کند که تغییر مسیر دهنده شبکه باشد و یا اینکه به سخت افزار خاص آنها را ارجاع دهد.این قضیه به واسطه پیوند نمادی (Symbolic Link) انجام می شود.:E خود یک پیوند نمادی است.برای مثال : شبکه ممکن است که درایو E: را به \Device\NetworkRedirector ترسیم کند و حافظه قابل حمل نیز ممکن است E: را به \Device\FujiMemoryStick ترسیم کند .

این نشان می دهد که چگونه برنامه ها می توانند از ، اسم های معمول تعریف شده ای که مجزا شده اند برای اشاره به درایور هر دستگاه و انجام تقاضاها ، استفاده کنند . قانونی وجود ندارد ، ما می توانیم \Device\Example را به E: ترسیم کنیم .می توانیم هر آنچه می خواهیم انجام دهیم ، اما در پایان ، به هرحال برنامه طوری تلاش می کند دستگاه را بکار گیرد که درایور پاسخ دهد و کار کند.این بدان معناست که IOCTL معمولا به وسیله آن دسته از دستگاه ها استفاده می شود به طوری که برنامه ها آنها را بکار گیرند.COM1 وCOM2 و غیره همگی نمونه ای از این هستند.COM1 یک اسم داس می باشد که ترسیم شده اسم دستگاه NT درایوری است ، که درخواست های پیاپی(serial) دارد.احتیاجی نیست که ، یک پورت سریال فیزیکی باشد.

بنابراین ما "Example" را به عنوان یک دستگاه داس تعریف کردیم که به "\Device\Example" اشاره دارد.

- **ساخت روال تخلیه بار**

کد بعدی که خواهیم دید روال تخلیه بار است.این امر به جای اینکه درایور دستگاه را به صورت پویا تخلیه کنیم ، لازم است. این قسمت کم خواهد بود و مطلب زیادی جهت توضیح وجود ندارد. شما هر آنچه بخواهید می توانید در روال تخلیه بار انجام دهید.این روال بسیار ساده می باشد ، این روال فقط پیوند نمادی را حذف کرده و تنها بود.\Device\Exampleدستگاهی را که ساخته بودیم را حذف می کند که

2-1-7- ساخت IRP_MJ_WRITE

اگر از WriteFile و ReadFile استفاده کرده باشید ، می دانید که یک بافر داده ، برای نوشتن در دستگاه و یا خواندن از یک دستگاه ، تعیین می کنید.این پارامترها به دستگاه در IRP فرستاده می شود همان طور که قبلا توضیح داده شد.سه روش وجود دارد که مدیریت I/O برای سازمان دهی داده ها قبل از تحویل IRP به درایور ، انجام می دهد.همچنین این معنا را می دهد که چگونه اطلاعات سازمان داده شده ، چگونه توابع خواندن و نوشتن احتیاج به تفسیر داده دارند.

سه روش به این شرح است: "Direct I/O"، "Buffered I/O" و "Neither"

```
#ifdef __USE_DIRECT
#define IO_TYPE DO_DIRECT_IO
#define USE_WRITE_FUNCTION  Example_WriteDirectIO
#endif
#ifdef __USE_BUFFERED__
#define IO_TYPE DO_BUFFERED_IO
#define USE_WRITE_FUNCTION  Example_WriteBufferedIO
#endif
#ifndef IO_TYPE
#define IO_TYPE 0
#define USE_WRITE_FUNCTION  Example_WriteNeither
```

```
#endif
```

کد نوشته شده ، بنابراین اگر شما "__USE_DIRECT__" را در header تعریف کنید ، سپس IO_TYPE حالا DO_DIRECT_IO است و USE_WRITE_FUNCTION حالا Example_WriteDirectIO است.اگر "__USE_BUFFERED__" را در header تعریف کنید سپس IO_TYPE حالا DO_BUFFERED_IO است و USE_WRITE_FUNCTIONحالا Example_WriteBufferedIO است.اگر __USE_DIRECT__ یا __USE_BUFFERED__ را تعریف نکنید.سپس IO_TYPE به عنوان 0 (هیچیک) تعریف می شود و تابع نوشتن Example_WriteNeither است.حال هر نوع از I/O را بررسی می کنیم.

Direct I/O -1-7-1-2

اولین کاری که انجام می دهم نمایش کدی است برای کار با direct I/O

```
NTSTATUS Example_WriteDirectIO(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  NTSTATUS NtStatus = STATUS_SUCCESS;
  PIO_STACK_LOCATION pIoStackIrp = NULL;
  PCHAR pWriteDataBuffer;
  DbgPrint("Example_WriteDirectIO Called \r");
*/
*   Each time the IRP is passed down
*   the driver stack a new stack location is added
*   specifying certain parameters for the IRP to the driver.
/*
  pIoStackIrp = IoGetCurrentIrpStackLocation(Irp);
    if (pIoStackIrp)
{
```

```
    pWriteDataBuffer =
      MmGetSystemAddressForMdlSafe(Irp->MdlAddress, NormalPagePriority);
       if(pWriteDataBuffer(
    {
*/
*         We need to verify that the string
*         is NULL terminated. Bad things can happen
*         if we access memory not valid while in the Kernel.
/*
      if(Example_IsStringTerminated(pWriteDataBuffer ,
        pIoStackIrp->Parameters.Write.Length))
    {
         DbgPrint(pWriteDataBuffer);
} } }
  return NtStatus;
}
```

نقطه ورود در واقع device object را برای دستگاهی ، که این تقاضا برای آن ارسال شده ، فراهم می کند . اگر دوباره فراخوانید ، یک درایور تنها می تواند دستگاه های چند گانه را بسازد حتی با اینکه ما تنها یک دستگاه داریم و سایر پارامترها طوری که قبل از اینکه IRP باشند ، ذکر می شده بودند ، هستند.

اولین چیزی که ما انجام می دهیم "IoGetCurrentIrpStackLocation" و برای ما همراه با IO_STACK_LOCATION در تمرینمان فراهم می شود.تنها پارامتری که ما احتیاج داریم طول بافری است که برای درایور فراهم می شود که در Parameters.Write.Length هست.

روشی که I/O بافر شده کار می کند این است که با یک "MdlAddress" فراهم می شود که یک "Memory Decrptor List" است.این شرحی در مورد آدرس های مد کاربری و چگونگی ترسیم به آدرس

های فیزیکی ، است.تابع را ما فرا می خوانیم سپس "MmGetSystemAddressForMdlSafe" می شود.و برای انجام این کار ، از Irp->MdlAddress استفاده می کنیم .این عملیات سپس به ما آدرس مجازی سیستم را می دهد که جهت خواندن از حافظه به کار می گیریم.

استدلالی که در پس این موضوع هست ، این است که بعضی از درایور ها معمولا یک تقاضا مد کاربری را در زمینه یک رشته یا پردازشی که قبلا نیز گفتیم ، پردازش نمی کنند.اگر شما یک تقاضا را در رشته های مختلفی پردازش کنید، که در حال اجرا در پردازش دیگری هستند ، شما نمی توانید در حین پردازش به حافظه مد کاربری دست پیدا کنید ، شما حتما این مورد را از قبل می دانستید که هنگام اجرا دو برنامه ، بدون پشتیبانی سیستم عامل، نمی توانید از یکدیگر بخوانید یا بنویسید .

بنابراین،حافظه فیزیکی که توسط پردازش مد کاربری مورد استفاده قرار می گرفته ، در حافظه سیستم ترسیم می شود.ما می توانیم از آدرسی بازگشتی برای دسترسی به بافری که از مد کاربری عبور کرده ، استفاده کنیم.

این روش معمولا برای بافرهای بزرگتر تا زمانی که احتیاج به حافظه برای کپی شدن ندارند ، مورد استفاده قرار می گیرد.بافرهای مد کاربری در حافظه قفل می شوند تا زمانی که IRP کامل شده و در حال اتمام استفاده از direct I/O می باشد.این فقط افت می باشد و دلیلی بر آن است که چرا ، معمولا پر کابردترین برای بافرهای بزرگتر است.

Buffered I/O -2-7-1-2

اولین کاری که انجام می دهم نمایش کدی است برای کار با Buffered I/O

```
NTSTATUS Example_WriteBufferedIO(PDEVICE_OBJECT DeviceObject,
PIRP Irp)
{
  NTSTATUS NtStatus = STATUS_SUCCESS;
  PIO_STACK_LOCATION pIoStackIrp = NULL;
  PCHAR pWriteDataBuffer;
  DbgPrint("Example_WriteBufferedIO Called \r\n");
*/
*   Each time the IRP is passed down
```

```
*     the driver stack a new stack location is added
*     specifying certain parameters for the IRP to the driver.
/*
   pIoStackIrp = IoGetCurrentIrpStackLocation(Irp);
   if(pIoStackIrp(
   {
      pWriteDataBuffer = (PCHAR)Irp->AssociatedIrp.SystemBuffer;
      if(pWriteDataBuffer(
      {
*/
*           We need to verify that the string
*           is NULL terminated. Bad things can happen
*           if we access memory not valid while in the Kernel.
/*
        if(Example_IsStringTerminated(pWriteDataBuffer ,
             pIoStackIrp->Parameters.Write.Length((
}
           DbgPrint(pWriteDataBuffer);
      }}}
   return NtStatus;
}
```

همان طور که در بالا ذکرشده ، منظور آن است که ، داده را به درایور که قابل دسترسی از هر نوع مفهومی می باشد مانند رشته دیگری در پردازش دیگر ، تحویل دهیم . علت دیگر ترسیم حافظه به جایی که فراخوانی نشده و درایور همچنین آن را در سطح های افزایش یافته IRQL می خواند.

دلیل آن که در بعضی مواقع احتییاج به دسترسی به حافظه ، در بیرون از پردازش فعلی دارید این است که بعضی از درایور ها رشته ها را در پردازش سیستم می سازند.سپس انجام پردازش را همزمان و یا غیر همزمان به تعویق می اندازند.درایور در سطوح بالا ممکن است این کار را از طریق درایور شما انجام دهند و یا ممکن است درایور شما این کار را خود انجام دهد.

افت استفاده "Buffered I/O" این است که حافظه های فراخوانی نشده را تعیین کرده و یک کپی از آنها می گیرد.هر خواندن و نوشتن در درایور سبب سنگینی در پردازش می شود.این دلیلی برای استفاده در بافرهای کوچک است.کل فراخوانی مد کاربری ، نیاز به قفل شدن در حافظه مانند Direct I/O ندارد.این مسئله ویژگی این نوع می باشد.یکی از مشکلات این نوع با بافرهای بزرگ در تعیین حافظه های فراخوانی نشده ، تعیین یک بلوک بزرگ و پشت سرهم از حافظه های فراخوانی نشده است.

3-7-1-2- نوع غیر از Buffered , Direct

اولین کاری که انجام می دهم نمایش کدی برای کار با این نوع است.

```
NTSTATUS Example_WriteNeither(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
   NTSTATUS NtStatus = STATUS_SUCCESS;
   PIO_STACK_LOCATION pIoStackIrp = NULL;
   PCHAR pWriteDataBuffer;
   DbgPrint("Example_WriteNeither Called \r\n");
*/
*    Each time the IRP is passed down
*    the driver stack a new stack location is added
*    specifying certain parameters for the IRP to the driver.
/*
```

```
    pIoStackIrp = IoGetCurrentIrpStackLocation(Irp;(
        if(pIoStackIrp(
    {
*/
*       We need this in an exception handler or else we could trap.
/*
__      try {
                ProbeForRead(Irp->UserBuffer ,
            pIoStackIrp->Parameters.Write.Length ,
            TYPE_ALIGNMENT(char;((
            pWriteDataBuffer = Irp->UserBuffer;
                if(pWriteDataBuffer)
            {
*/
*               We need to verify that the string
*               is NULL terminated. Bad things can happen
*               if we access memory not valid while in the Kernel.
/*
            if(Example_IsStringTerminated(pWriteDataBuffer ,
                pIoStackIrp->Parameters.Write.Length))
{               DbgPrint(pWriteDataBuffer);
            {{
{               except( EXCEPTION_EXECUTE_HANDLER} (
        NtStatus = GetExceptionCode();
```

```
}}
    return NtStatus;
}
```

در این روش ، درایور مستقیما به آدرس مد کاربری دسترسی پیدا می کند.مدیریت I/O داده ها را کپی نمی کند ، فراخوانی های مد کاربری را در حافظه قفل نمی کند، به درایور، بافر آدرس مد کاربری می دهد.

مزایا این که: داده ای کپی نمی شود ، حافظه ای معیین نمی شود و هیچ فراخوانی در حافظه قفل نمی شود.عیب این نوع : شما باید این تقاضا را در مفهومی که رشته را می خواند پردازش کنید بنابراین به فضای آدرس مد کاربری پردازش صحیح ، دسترسی خواهید داشت.عیب دیگر این نوع این است که پردازش خود می تواند دسترسی به صفحه های حافظه ، حافظه های آزاد و غیره در سایر رشته ها را تغییر دهد.معمولا این مسئله دلیل این است که می خواهیم از توابع "ProbeForRead" و"ProbeForWrite" همراه با اجراگرهای استثنایی (کدهایی که در هنگام بروز مشکلات خاص وارد عمل می شوند)استفاده کنیم.هیچ تضمین وجود ندارد که فراخوانی هادر هر زمانی مجاز باشند و باید قبل از تلاش برای نوشتن یا خواندن ، از قابل استفاده بودن آنها اطمینان حاصل کنیم.این بافر در Irp- >UserBuffer ذخیره می شود.

تمامی این رویه های که بررسی کردید ، به پیوند دهنده اجازه میدهد که بداند کدها را در کدام قسمت قرار دهد و چه طور صفحه های حافظه را به چیند.برای مثال "DriverEntry" به عنوان "INIT" ، صفحه ای از حافظه که اهمیت ندارد ، تنظیم می شود ، این قضیه به این دلیل است که شما به آن تابع در حین راه اندازی نیازمندید.

2-1-8- بارگذاری و تخلیه بار درایور به صورت پویا

یک API مد کاربری وجود دارد که شما می توانید برای بار گذاری و تخلیه بار درایور،بدون نیاز به انجام کاری به کار برید.

این چیزی است به کار خواهیم برد:

```
int _cdecl main(void)
{
   HANDLE hSCManager;
   HANDLE hService;
   SERVICE_STATUS ss;
   hSCManager = OpenSCManager(NULL, NULL,
SC_MANAGER_CREATE_SERVICE);
      printf("Load Driver\n;("
   if(hSCManager(
   {
      printf("Create Service\n;("
      hService = CreateService(hSCManager, "Example ,"
"                        Example Driver ,"
                        SERVICE_START | DELETE | SERVICE_STOP ,
                        SERVICE_KERNEL_DRIVER,
                        SERVICE_DEMAND_START ,
                        SERVICE_ERROR_IGNORE ,
"                        C:\\example.sys ,"
```

```
                        NULL, NULL, NULL, NULL, NULL);
    if(!hService){
        hService = OpenService(hSCManager, "Example ,"
                SERVICE_START | DELETE | SERVICE_STOP);
    }
    if(hService){
        printf("Start Service\n");
        StartService(hService, 0, NULL);
        printf("Press Enter to close service\r\n;("
        getchar();
        ControlService(hService, SERVICE_CONTROL_STOP, &ss);
        DeleteService(hService);
        CloseServiceHandle(hService);
      }
    CloseServiceHandle(hSCManager);
}
    return 0;
}
```

این کد درایور را بارگذاری کرده و اجرا می کند.درایور را با "SERVICE_DEMAND_START" بار گذاری می کنیم که به این معناست که این درایور باید فیزیکی راه اندازی شود. درایور به صورت خود بخود در بوت اجرا نخواهد شد. روشی اس برای تست کردن و اگر صفحه آبی ظاهر شود بدون نیاز به راه اندازی در حالت امن می توان درایور را اصلاح کرد.

برنامه به سهولت متوقف خواهد شد.سپس می توانید برنامه ای را که در پنجره ای دیگر در حال کار می باشد را اجرا کنید.باید از کدهای بالا فهمیده باشید که برای اجرا درایور باید آنها را در مسیر C:\example.sys کپی

کنید.اگر سیستم نتواند آن را بسازد ، حتما ساخته شده است بنابراین آن را باز می کند. سپس کار را شروع می کنیم و متوقف می سازیم . با فشردن یکبار کلید Enter ، کار متوقف می شود و کار از لیست خدمات حذف شده و خارج می شود.این کدی است بسیار ساده که می توانید بستگی به مقاصدتان آن را اصلاح کنید.

- **ارتباط با دستگاه درایور:**

کد زیر ، کدی برای ارتباط با درایور است.

```
int _cdecl main(void)
{
   HANDLE hFile;
   DWORD dwReturn;
   hFile = CreateFile("\\\\.\\Example ,"
         GENERIC_READ | GENERIC_WRITE, 0, NULL ,
         OPEN_EXISTING, 0, NULL);
   if(hFile(
   {
      WriteFile(hFile, "Hello from user mode ,"!
            sizeof("Hello from user mode!"), &dwReturn, NULL);
      CloseHandle(hFile);
}
   return 0;}
```

مسلما ساده تر از تصورتان است.اگر درایور را با سه روش مختلف I/O ، کامپایل کنید ، پیغام از مد کاربری در DBGVIEW ثبت می شود ، همان طور که توجه کردید ، باید نام دستگاه داس را از طریق به کاربردن \\.\<DosName>_ باز کنید . همچنین باید \Device\<Nt Device Name> را با کار گیری روش قبل باز کنید.سپس می توانید با دستگاه کار کنید و همچنین می توانید WriteFile,ReadFile,CloseHandle<DeviceIoControl را فراخوانید.اگر کنجکاو هستید ، کار را انجام دهید و با استفاده از DbgPrint از کدهای اجرا شده در درایورتان مطلع گردید.

2-2- کار با IOCTLs

2-2-0- مقدمه

این مقاله دومین آموزش از سری مقاله های درایورنویسی دستگاه می باشد.موارد جالب زیادی در این مقاله خواهید یافت بنابراین هر آنچه از آموزش باقی مانده باشد را بدست خواهید آورد ، تمرکز اصلی این مقاله ها بر فراگیری تدریجی دانشی است که برای نوشتن درایور نیاز دارید. در این مقاله بر روی کدهایی که در قسمت اول ذکر شد کار خواهیم کرد و کدها را توسعه خواهیم داد تا کاربرد خواندن را اضافه کنیم ، با کنترل ورودی / خروجی که به عنوان IOCTLs می شناسیم ، کار کنیم و قدری بیشتر راجع IRP بدانیم.

2-2-1- اجرا ReadFile

همان طور که قبلا اشاره کردیم سه نوع متقاوت I/O داریم که شامل Direct , Bufferd و Neither می باشد. با هر سه نوع در مثال درایور کار کردیم حال تفاوت این است که به جای خواندن از حافظه در حافظه می نویسیم ، طوری توضیح نخواهم داد که همگی یکسان هستند ، چیزی که توضیح خواهم داد کاربرد جدیدی است که اضافه کردم:return values

در کار با WriteFile ، نیازی به نگرانی برای return values نیست. اجرا صحیح همیشه باید به برنامه مد کاربری میزان اطلاعاتی را که نوشته شده است را اطلاع دهد.به هرحال ، از توضیح جزئیات فعلا صربف نظر می کنیم.این مورد درباره "ReadFile" ضروری است و حتی در صورت صحت اجرا باید به مدیریت I/O به درستی اطلاع دهد.

چگونه کار می کند برای مثال،بافر حافظه در مکانی دیگر ساخته می شود "Buffered I/O"اگر بیاد بیاورید که باید بداند که چه مقدار I/Oاوحافظه مد کاربری کپی می شود.اگر بخواهیم از درایور داده بخوانیم ، مدیریت حافظه را باید از بافر موقت در مکان واقعی حافظه مد کاربری کپی کند . اگر این عمل را انجام ندهیم ، حافظه ای کپی نخواهد شد و برنامه مد کاربری داده ای دریافت نخواهد کرد.

```
NTSTATUS Example_ReadDirectIO(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  NTSTATUS NtStatus = STATUS_BUFFER_TOO_SMALL;
  PIO_STACK_LOCATION pIoStackIrp = NULL;
  PCHAR pReturnData = "Example_ReadDirectIO - Hello from the Kernel");
```

```
   UINT dwDataSize = sizeof("Example_ReadDirectIO - Hello from the Kernel”);
   UINT dwDataRead = 0;
   PCHAR pReadDataBuffer;
   DbgPrint("Example_ReadDirectIO Called \r\n”);
*/
*     Each time the IRP is passed down the driver stack a
*     new stack location is added
*     specifying certain parameters for the IRP to the
*     driver.
/*
   pIoStackIrp = IoGetCurrentIrpStackLocation(Irp;(
   if(pIoStackIrp && Irp->MdlAddress) {
      pReadDataBuffer = MmGetSystemAddressForMdlSafe(Irp->MdlAddres , NormalPagePriority);
      if(pReadDataBuffer  && pIoStackIrp->Parameters.Read.Length >= dwDataSize){
*/
*           We use "RtlCopyMemory" in the kernel instead
*           of memcpy.
*           RtlCopyMemory *IS* memcpy, however it's best
*           to use the
*           wrapper in case this changes in the future.
/*
         RtlCopyMemory(pReadDataBuffer, pReturnData ,dwDataSize);
         dwDataRead = dwDataSize;
         NtStatus = STATUS_SUCCESS;
```

```
}
```

2-2-2- اجرا Return Value

Return value اجرا شده ، IO_STATUS_BLOCK ، IRP را به کار می گیرد.این قسمت شامل تعدادی بخش از داده ها است که کاربرد خودشان را بستگی به اجرا تابع اصلی ، تغییر می دهند.در اجرا تابع اصلی ، "Status" برابر است با کد برگشتی و "Information" شامل تعداد بایت خواندن / نوشتن است.با مشاده کد جدید ، درمی یابیم که حال "IoCompleteRequest" را فراخواندیم.تمامی این فراخوانی به چه منظور است؟

IoCompleteRequest معمولابعد از تکمبل IRP توسط درایور فراخوانده می شود.دلیل اینکه در مثال قبلی همه چیز مناسب بود این است که مدیریت I/O خود در بیشتر مورد ها این کار را انجام می داد.به هرحال ، برای درایور مناسب تر است که IRP را در هر جا نیاز شد درست کند.این مکان شامل اطلاعاتی است در "IRP Handling" که می تواند اطلاعات بیشتری را فراهم کند.

```
Irp->IoStatus.Status = NtStatus;
   Irp->IoStatus.Information = dwDataRead;
   IoCompleteRequest(Irp, IO_NO_INCREMENT);
   return NtStatus;
}
```

پارامتر دوم IoCompleteRequest ، اولویت در حال ارتقاء را جهت تحویل به ترد در حال انتظار برای تکمیل IRP ، تعیین می کند.به عنوان مثال،شاید ترد مدت زمان طولانی را برای عملکرد شبکه منتظر می ماند.این ارتقاء به زمان بند کمک می کند که ترد را سریع تر از زمانی که بدون ارتقاء باید در صف بماند ، اجرا می کند. این معمولا یک کمک دهنده است که برای مطلع ساختن زمان بند برای اجرا یک ترد در حال انتظار این I/O استفاده می شود.

2-2-3- معتبرسازی پارامترهای محض و بررسی خطاها

هم اکنون کد با تعداد خطای کمتر و معتبر سازی پارامتر، نسبت به قبل اجرا می شود. یک مسئله که راجع درایور باید مطمئن شوید این است که برنامه مد کاربری نباید مکان حافظه نادرست و غیره را ارسال کند و منجر به نمایش صفحه آبی در سیستم شود.اجرای درایور باید برروی خطاهایی که به درایور مد کاربری بر می گرداند در عوض نمایش "STATUS_SUCCESS" در همه مواقع، بهتر کار کند.ما باید پرازش مد کاربری را در هنگامی

که ، داده های زیادی را ارسال می کند و یا در تعیین دقیق زمانی که دچار اشکال می شود ،مطلع سازیم. یکی از ویژیگی های API این است که می توانید GetLastError را فرابخوانید برای مشاهده علت خراب شدن و یا استفاده از ارزش برگشتی برای تعیین اینکه چطور کد تان را اصلاح کنید.اگر درایورتان همیشه "failed" یا بهتر "Success" را برگرداند ، مطابق کردن برنامه برای درست کار کردن با درایور مشکل تر می شود.

۲-۲-۴- کنترل ورودی/خروجی (IOCTL)

IOCTL به عنوان ارتباط بین درایور و برنامه بجای خواندن و نوشتن ساده داده ها ، بیشتر استفاده می شود. معمولا درایور تعداد زیادی IOCTL را ارسال می کند و ساختار داده مورد نیاز برای ارتباط را تعیین می کند.تمامی داده ها باید شامل بلوک یکسانی باشند .اگر قصد ساخت اشاره گر ها را دارید , کارهایی مثل : ساخت افست در بلوک داده ها پیش از پایان داده ثابت ، بنابراین درایور می تواند به راحتی این اطلاعات را پیدا کند.اگر بیاد بیاورید ، درایور این توانایی را دارد که داده مد کاربری را تا وقتی که در زمینه پردازش است بخواند.بنابراین ، این امکان وجود دارد که ابزار اشاره گر به حافظه و درایور به کپی کردن و یا قفل کردن صفحه های حافظه ، نیاز پیدا خواهند کرد (اجرا اساسا بافر یا direct I/O از خود درایور که می توانند انجام شده باشند) .پردازه مد کاربری از "DeviceIoControl" API برای انجام این ارتباط ، استفاده خواهد کرد.

۲-۲-۴-۱- تعریف IOCTL

اولین نکته که باید توضیح داده شود این است که کد IOCTL بین درایور و برنامه مورد استفاده قرار می گیرد.در ابتدا برای ربط دادن IOCTL به چیزی در مد کاربری ، شاید فکر کنید که آن یک پنجره پیغام است.در واقع یک ارزش که توسط درایور برای انجام تابع خواسته شده با ارزش ورودی / خروجی از پیش تعریف شده ، است.به هر حال این درایور بیشتر از یک پنجره پیغام است . .IOCTL دسترسی مورد نیاز را به جای اینکه روش انتقال مورد استفاده بین درایور و برنامه تعریف شود ، تعیین می کند.

IOCTL یک عدد ۳۲ بیتی است .اولین دو بیت کم ارزش "نوع انتقال" را تعیین می کنند که می توانند METHOD_BUFFERED , METHOD_IN_DIRECT , METHOD_OUT_DIRECT یا METHOD_NEITHER باشند.

سری بعدی رقم ها از ۲ تا ۱۳ "Function Code" را تعیین می کند. بیت پر ارزش به "Custom Bit" اشاره دارد.این رقم برای تعیین IOCTL که توسط کاربر تعیین شده در مقابل تعریف سیستم می باشد.این به این معناست که کدهای تابع ۰۸۰۰* و بالاتر انتخابی مانند WM_USER که برای پنجره پیغام کار می کند ، تعریف می شوند.

دو رقم بعدی برای دسترسی مورد نیاز به IOCTL است.این نشان می دهد که چگونه در خواست های IOCTL از طرف مدیریت I/O در صورتی که از طریق ورودی صحیح باز نشده باشد ، رد می شود.برای مثال انواع ورودی شامل FILE_READ_DATA و FILE_WRITE_DATA هستند.

آخرین بیت نوع دستگاه را که IOCTL برای آن نوشته شده است را بیان می کند. رقم پر ارزش ارزشهای تعریف شده کاربر را بیان می کند.

یک ماکرو وجود دارد که می توانیم IOCTL را سریعا تعریف کنیم و آن "CTL_CODE" است.در کد زیر از آن برای تعریف چهار IOCTL که روش انتقال دسترسی از نوع های متفاوت را انجام می دهد.

```
/*
* IOCTL's are defined by the following bit layout.
* [Common |Device Type|Required Access|Custom|Function Code|Transfer Type]
* 31    30      16 15         14 13  12         2 1         0
*
* Common         - 1 bit.  This is set for user-defined
*                  device types.
* Device Type    - This is the type of device the IOCTL
*                  belongs to.  This can be user defined
*                  (Common bit set).  This must match the
*                  device type of the device object.
* Required Access - FILE_READ_DATA, FILE_WRITE_DATA, etc.
*                  This is the required access for the
*                  device.
* Custom         - 1 bit.  This is set for user-defined
*                  IOCTL's.  This is used in the same
*                  manner as "WM_USER".
* Function Code  - This is the function code that the
*                  system or the user defined (custom
*                  bit set)
* Transfer Type  - METHOD_IN_DIRECT, METHOD_OUT_DIRECT,
```

```
*                   METHOD_NEITHER, METHOD_BUFFERED, This
*                   the data transfer method to be used.
*/
#define IOCTL_EXAMPLE_SAMPLE_DIRECT_IN_IO    \
        CTL_CODE(FILE_DEVICE_UNKNOWN,       \
                0x800,                  \
                METHOD_IN_DIRECT,           \
                FILE_READ_DATA | FILE_WRITE_DATA)
#define IOCTL_EXAMPLE_SAMPLE_DIRECT_OUT_IO   \
        CTL_CODE(FILE_DEVICE_UNKNOWN,       \
                0x801,                  \
                METHOD_OUT_DIRECT,          \
                FILE_READ_DATA | FILE_WRITE_DATA)
#define IOCTL_EXAMPLE_SAMPLE_BUFFERED_IO     \
        CTL_CODE(FILE_DEVICE_UNKNOWN,       \
                0x802,                  \
                METHOD_BUFFERED,            \
                FILE_READ_DATA | FILE_WRITE_DATA)
#define IOCTL_EXAMPLE_SAMPLE_NEITHER_IO      \
        CTL_CODE(FILE_DEVICE_UNKNOWN,       \
                0x803,                  \
                METHOD_NEITHER,             \
                FILE_READ_DATA | FILE_WRITE_DATA
```

در این کد طریقه تعریف IOCTL را مشاهده کردیم.

۲-۲-۴-۲- اجرا IOCTL

اولین چیزی که که نیاز به بروز دارد یک Switch Statement است که IOCTL را برای اجرا درست توزیع می کند.در اصل چیزی است که رویه ویندوز نیز برای ارسال سریع پنجره پیغام از آن استفاده می کند. همچنین مفهوم دیگری به عنوان "def IOCTL proc" وجود ندارد.

"Parameters.DeviceIoControl.IoControlCode" در IO_STACK_LOCATION نیز حاوی کد IOCTL است که فراخوانده می شود.

```
NTSTATUS Example_IoControl(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
   NTSTATUS NtStatus = STATUS_NOT_SUPPORTED;
   PIO_STACK_LOCATION pIoStackIrp = NULL;
   UINT dwDataWritten = 0;
   DbgPrint("Example_IoControl Called \r\n");
   pIoStackIrp = IoGetCurrentIrpStackLocation(Irp);
   if(pIoStackIrp) /* Should Never Be NULL! */
   {
      switch(pIoStackIrp->Parameters.DeviceIoControl.IoControlCode)
      {
         case IOCTL_EXAMPLE_SAMPLE_DIRECT_IN_IO:
            NtStatus = Example_HandleSampleIoctl_DirectInIo(Irp,
                        pIoStackIrp, &dwDataWritten);
            break;
         case IOCTL_EXAMPLE_SAMPLE_DIRECT_OUT_IO:
            NtStatus = Example_HandleSampleIoctl_DirectOutIo(Irp,
                        pIoStackIrp, &dwDataWritten);
            break;
         case IOCTL_EXAMPLE_SAMPLE_BUFFERED_IO:
            NtStatus = Example_HandleSampleIoctl_BufferedIo(Irp,
                        pIoStackIrp, &dwDataWritten);
            break;
         case IOCTL_EXAMPLE_SAMPLE_NEITHER_IO:
```

```
            NtStatus = Example_HandleSampleIoctl_NeitherIo(Irp,
                        pIoStackIrp, &dwDataWritten);
            break;
        }
    }
    Irp->IoStatus.Status = NtStatus;
    Irp->IoStatus.Information = dwDataWritten;
    IoCompleteRequest(Irp, IO_NO_INCREMENT);
    return NtStatus;
}
```

اگر اجرا ReadFile و WriteFile را درک کرده باشید , هر دو در یک فراخوانی اجرا می شوند.این موضوع دقیقا قضیه مورد بحث نیست. IOCTL فقط می تواند جهت خواندن داده ، نوشتن داده به کار رود و در کل نمی تواند داده ای را ارسال کند ولی درایور را برای اجرای یک عملکرد مطلع می سازد.

METHOD_x_DIRECT -3-4-2-2

METHOD_IN_DIRECT و METHOD_OUT_DIRECT هر دو را می توان در یک توضیح داد. اساسا هر دو یکسان هستند. بافر Input برای استفاده در اجرای "BUFFERED" تعیین می شود و Output بافر جهت استفاده در MdlAddress طبق توضیحی که در اجرا خواندن / نوشتن داده شد.تفاوت در میان "IN" و "OUT" در "IN" است.شما می توانید از بافر خروجی برای گذر به داده استفاده کنید."OUT" تنها جهت بازگشت داده مورد استفاده قرار می گیرد.در مثالی که داریم از "IN" برای گذر به داده ها استفاده نکردیم و اساسا عملکرد "OUT" و "IN" در مثال یکسان است .

```
NTSTATUS Example_HandleSampleIoctl_DirectOutIo(PIRP Irp,
    PIO_STACK_LOCATION pIoStackIrp, UINT *pdwDataWritten)
{
    NTSTATUS NtStatus = STATUS_UNSUCCESSFUL;
    PCHAR pInputBuffer;
    PCHAR pOutputBuffer;
```

```
    UINT dwDataRead = 0, dwDataWritten = 0;
    PCHAR pReturnData = "IOCTL - Direct Out I/O From Kernel!";
    UINT dwDataSize = sizeof("IOCTL - Direct Out I/O From Kernel!");
    DbgPrint("Example_HandleSampleIoctl_DirectOutIo Called \r\n");
    /*
     * METHOD_OUT_DIRECT
     *    Input Buffer = Irp->AssociatedIrp.SystemBuffer
     *    Ouput Buffer = Irp->MdlAddress
     *    Input Size   =  Parameters.DeviceIoControl.InputBufferLength
     *    Output Size  =  Parameters.DeviceIoControl.OutputBufferLength
     * What's the difference between METHOD_IN_DIRECT && METHOD_OUT_DIRECT?
     * The function which we implemented METHOD_IN_DIRECT
     * is actually *WRONG*!!!!  We are using the output buffer
     * as an output buffer!  The difference is that METHOD_IN_DIRECT creates
     * an MDL for the outputbuffer with
     * *READ* access so the user mode application
     * can send large amounts of data to the driver for reading.
     * METHOD_OUT_DIRECT creates an MDL
     * for the outputbuffer with *WRITE* access so the user mode
     * application can recieve large amounts of data from the driver!
         * In both cases, the Input buffer is in the same place,
     * the SystemBuffer.  There is a lot
     * of consfusion as people do think that
     * the MdlAddress contains the input buffer and this
     * is not true in either case.
```

```
    */
    pInputBuffer = Irp->AssociatedIrp.SystemBuffer;
    pOutputBuffer = NULL;
    if(Irp->MdlAddress)
    {
        pOutputBuffer =
          MmGetSystemAddressForMdlSafe(Irp->MdlAddress,
          NormalPagePriority);
    }
    if(pInputBuffer && pOutputBuffer)
    {
        /*
         * We need to verify that the string
         * is NULL terminated. Bad things can happen
         * if we access memory not valid while in the Kernel.
         */
       if(Example_IsStringTerminated(pInputBuffer,
         pIoStackIrp->Parameters.DeviceIoControl.InputBufferLength,
         &dwDataRead)) {
            DbgPrint("UserModeMessage = '%s'", pInputBuffer);
            DbgPrint("%i >= %i",
              pIoStackIrp->Parameters.DeviceIoControl.OutputBufferLength,
              dwDataSize);
            if(pIoStackIrp->
              Parameters.DeviceIoControl.OutputBufferLength >= dwDataSize)
            {
```

```
            /*
             * We use "RtlCopyMemory" in the kernel instead of memcpy.
             * RtlCopyMemory *IS* memcpy, however it's best to use the
             * wrapper in case this changes in the future.
             */
            RtlCopyMemory(pOutputBuffer, pReturnData, dwDataSize);
            *pdwDataWritten = dwDataSize;
            NtStatus = STATUS_SUCCESS;
        }
        else
        {
            *pdwDataWritten = dwDataSize;
            NtStatus = STATUS_BUFFER_TOO_SMALL;
        } } }
    return NtStatus;
}
```

METHOD_BUFFERED -4-4-2-2

عملکرد METHOD_BUFFERED در واقع با عملکرد خواندن و نوشتن یکسان است.یک بافر تخصیص یافته است و داده از این بافر کپی می شوند.بافر از لحاظ بزرگی دارای دو اندازه ، ورودی یا خروجی می باشد. سپس بافر خواندن در بافر جدید کپی می شود ، قبل از بازگشت ، داده برگشتی در بافر یکسان کپی می شود. ارزش بازگشتی در IO_STATUS_BLOCK قرار می گیرد و مدیریت I/O داده را در بافر خروجی کپی می کند.

```
NTSTATUS Example_HandleSampleIoctl_BufferedIo(PIRP Irp,
        PIO_STACK_LOCATION pIoStackIrp, UINT *pdwDataWritten)
{
    NTSTATUS NtStatus = STATUS_UNSUCCESSFUL;
```

```
    PCHAR pInputBuffer;
    PCHAR pOutputBuffer;
    UINT dwDataRead = 0, dwDataWritten = 0;
    PCHAR pReturnData = "IOCTL - Buffered I/O From Kernel!";
    UINT dwDataSize = sizeof("IOCTL - Buffered I/O From Kernel!");
    DbgPrint("Example_HandleSampleIoctl_BufferedIo Called \r\n");
    /*
     * METHOD_BUFFERED
     *
     *    Input Buffer = Irp->AssociatedIrp.SystemBuffer
     *    Ouput Buffer = Irp->AssociatedIrp.SystemBuffer
     *
     *    Input Size   =  Parameters.DeviceIoControl.InputBufferLength
     *    Output Size  =  Parameters.DeviceIoControl.OutputBufferLength
     *
     *    Since they both use the same location
     *    so the "buffer" allocated by the I/O
     *    manager is the size of the larger value (Output vs. Input)
     */
    pInputBuffer = Irp->AssociatedIrp.SystemBuffer;
    pOutputBuffer = Irp->AssociatedIrp.SystemBuffer;
    if(pInputBuffer && pOutputBuffer)
    {
        /*
         * We need to verify that the string
         * is NULL terminated. Bad things can happen
```

```
        * if we access memory not valid while in the Kernel.
        */
       if(Example_IsStringTerminated(pInputBuffer,
        pIoStackIrp->Parameters.DeviceIoControl.InputBufferLength,
        &dwDataRead)) {
          DbgPrint("UserModeMessage = '%s'", pInputBuffer);
          DbgPrint("%i >= %i",
            pIoStackIrp->Parameters.DeviceIoControl.OutputBufferLength,
            dwDataSize);
          if(pIoStackIrp->Parameters.DeviceIoControl.OutputBufferLength
                                                >= dwDataSize)
          {
             /*
              * We use "RtlCopyMemory" in the kernel instead of memcpy.
              * RtlCopyMemory *IS* memcpy, however it's best to use the
              * wrapper in case this changes in the future.
              */
             RtlCopyMemory(pOutputBuffer, pReturnData, dwDataSize);
             *pdwDataWritten = dwDataSize;
             NtStatus = STATUS_SUCCESS;
          }
          else
          {
             *pdwDataWritten = dwDataSize;
             NtStatus = STATUS_BUFFER_TOO_SMALL;
          }  }}
```

```
    return NtStatus;
}
```

MTHEOD_NEITHER -5-4-2-2

این روش همانند عملکرد neither I/O است .

```
NTSTATUS Example_HandleSampleIoctl_NeitherIo(PIRP Irp,
      PIO_STACK_LOCATION pIoStackIrp, UINT *pdwDataWritten)
{
    NTSTATUS NtStatus = STATUS_UNSUCCESSFUL;
    PCHAR pInputBuffer;
    PCHAR pOutputBuffer;
    UINT dwDataRead = 0, dwDataWritten = 0;
    PCHAR pReturnData = "IOCTL - Neither I/O From Kernel!";
    UINT dwDataSize = sizeof("IOCTL - Neither I/O From Kernel!");
    DbgPrint("Example_HandleSampleIoctl_NeitherIo Called \r\n");
    /*
     * METHOD_NEITHER
     *
     *    Input Buffer = Parameters.DeviceIoControl.Type3InputBuffer
     *    Ouput Buffer = Irp->UserBuffer
     *
     *    Input Size   =  Parameters.DeviceIoControl.InputBufferLength
     *    Output Size  =  Parameters.DeviceIoControl.OutputBufferLength
     *
     */
   pInputBuffer = pIoStackIrp-
>Parameters.DeviceIoControl.Type3InputBuffer;
```

```
    pOutputBuffer = Irp->UserBuffer;
    if(pInputBuffer && pOutputBuffer)
    {
        /*
         * We need this in an exception handler or else we could trap.
         */
        __try {
                ProbeForRead(pInputBuffer,
                    pIoStackIrp->Parameters.DeviceIoControl.InputBufferLength,
                    TYPE_ALIGNMENT(char));
                /*
                 * We need to verify that the string
                 * is NULL terminated. Bad things can happen
                 * if we access memory not valid while in the Kernel.
                 */
                if(Example_IsStringTerminated(pInputBuffer,
                    pIoStackIrp->Parameters.DeviceIoControl.InputBufferLength,
                    &dwDataRead))
                {
                    DbgPrint("UserModeMessage = '%s'", pInputBuffer);
                    ProbeForWrite(pOutputBuffer,
                    pIoStackIrp-
>Parameters.DeviceIoControl.OutputBufferLength,
                    TYPE_ALIGNMENT(char));
                    if(pIoStackIrp->
                    Parameters.DeviceIoControl.OutputBufferLength
                    >= dwDataSize)
```

```
                {
                /*
                 * We use "RtlCopyMemory"
                 * in the kernel instead of memcpy.
                 * RtlCopyMemory *IS* memcpy,
                 * however it's best to use the
                 * wrapper in case this changes in the future.
                 */
                RtlCopyMemory(pOutputBuffer,
                              pReturnData,
                              dwDataSize);
                *pdwDataWritten = dwDataSize;
                NtStatus = STATUS_SUCCESS;
                }
              else
                {
                *pdwDataWritten = dwDataSize;
                NtStatus = STATUS_BUFFER_TOO_SMALL;
                }
            }
      } __except( EXCEPTION_EXECUTE_HANDLER ) {
            NtStatus = GetExceptionCode();
      } }
  return NtStatus;
}
```

6-4-2-2- فراخوانی DeviceIoControl

عملکرد بسیار ساده ای است.

```
ZeroMemory(szTemp, sizeof(szTemp));
        DeviceIoControl(hFile,
                        IOCTL_EXAMPLE_SAMPLE_DIRECT_IN_IO,
                        "** Hello from User Mode Direct IN I/O",
                        sizeof("** Hello from User Mode Direct IN I/O"),
                        szTemp,
                        sizeof(szTemp),
                        &dwReturn,
                        NULL);
        printf(szTemp);
        printf("\n");
        ZeroMemory(szTemp, sizeof(szTemp));
        DeviceIoControl(hFile,
                         IOCTL_EXAMPLE_SAMPLE_DIRECT_OUT_IO,
                         "** Hello from User Mode Direct OUT I/O",
                         sizeof("** Hello from User Mode Direct OUT I/O"),
                         szTemp,
                         sizeof(szTemp),
                         &dwReturn,
                         NULL);
        printf(szTemp);
        printf("\n");
        ZeroMemory(szTemp, sizeof(szTemp));
        DeviceIoControl(hFile,
                        IOCTL_EXAMPLE_SAMPLE_BUFFERED_IO,
```

```
                "** Hello from User Mode Buffered I/O",
                sizeof("** Hello from User Mode Buffered I/O"),
                szTemp,
                sizeof(szTemp),
                &dwReturn, NULL);
        printf(szTemp);
        printf("\n");
        ZeroMemory(szTemp, sizeof(szTemp));
        DeviceIoControl(hFile,
                IOCTL_EXAMPLE_SAMPLE_NEITHER_IO,
                "** Hello from User Mode Neither I/O",
                sizeof("** Hello from User Mode Neither I/O"),
                szTemp,
                sizeof(szTemp),
                &dwReturn,
                NULL);
        printf(szTemp);
        printf("\n");
```

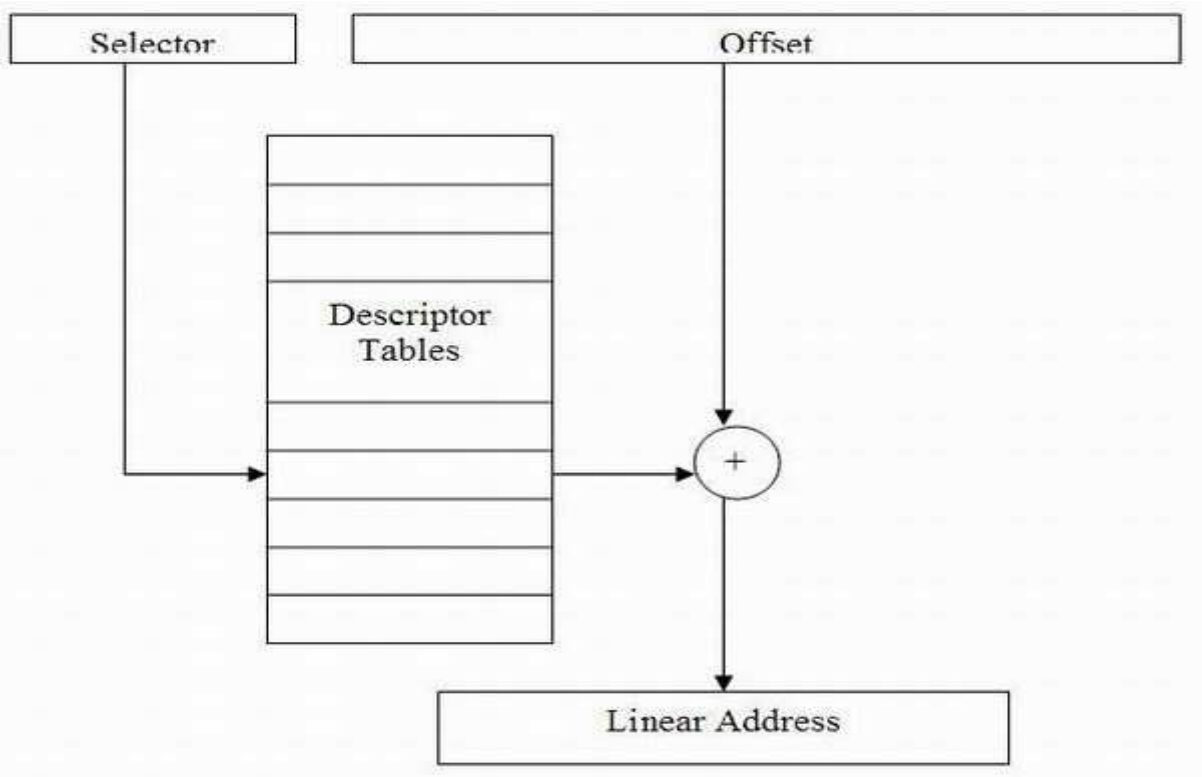
2-2-5- طرح حافظه سیستم

وقت مناسبی است که به آرایش و طرح حافظه ویندوز نگاهی بیندازیم.جهت درک کارکرد اول باید ببینیم که پردازنده های اینتل چه طور حافظه مجازی را اجرا می کنند.اجرا اصلی را با اینکه تفاوت هایی در اجرا هست ، توضیح خواهم داد.معمولا به عنوان "Virtual Address Translation" خوانده می شود.

Virtual Address Translation -1-5-2-2

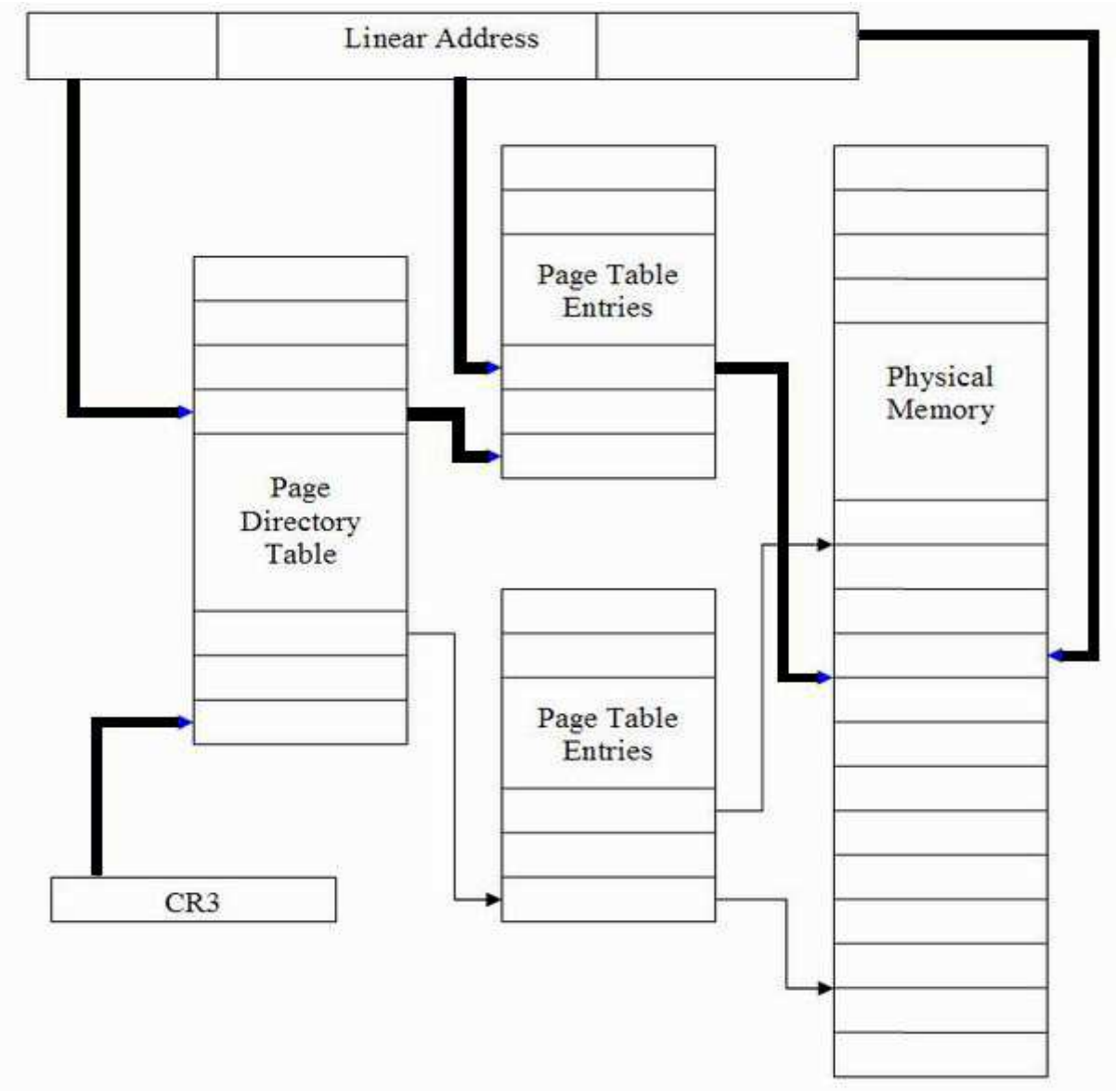
تمام قسمتهای ثبات ها "selectors" در مد حفاظت شده می شوند.برای آشنایی بیشتر با چگونگی کار کردن x86 مکانیزم پیجینگ را بطور کلی مرور می کنیم. ثبات های دیگری در CPU هستند که به "descriptor tables" اشاره دارند.این جداول مشخصات معین سیستم را تعریف می کنند که از توضیح بیش تر صرف نظر

می کنیم.در عوض ، چگونگی تبدیل آدرس مجازی را به آدرس فیزیکی توضیح خواهیم داد.جداول توصیف گر (descriptor tables) توانایی تعریف افستی را که به آدرس مجازی اضافه شده است ، دارند.اگر پیجینگ فعال نباشد یکبار که دو آدرس را اضافه کنید ، آدرس فیزیکی را بدست می آورید.اگر پیجینگ فعال باشد ، در عوض آدرس خطی بدست می آورید که به آدرس فیزیکی تبدیل می شود که از جداول پیج شده استفاده می کند.



شکل ۲-۱- جداول توصیف گر

یک نوع مکانیسم پیجینگ است که به عنوان "Page Address Extension" خوانده می شود و در اصل برای پردازنده های پنتیوم معرفی شده است.این مکانیسم به جداول پیج شده این اجازه را می دهد که به آدرس هایی تا ۳۲ رقم رجوع کنند.هرچند ، افست ها ۳۲ رقم هستند بنابراین در زمانی که به رم فیزیکی تا ۳۲ رقم دسترسی داشته باشید فقط به 4GB در یک زمان بدون بارگذاری مجدد جداول پیج شده ، می توانید دسترسی پیدا کنید.این مکانیسم پیجینگ دقیقا چیزی نیست که قصد داریم توضیح دهیم اما خیلی نزدیک و شبیه است.



شکل ۲-۲- مکانیزم پیجینگ

پیجینگ ۳۲ رقمی عادی به این صورت می باشد. یک ثبات CPU وجود دارد که به پایه دایرکتوری جدول پیج اشاره دارد و CR3 خوانده می شود. طرح زیر طریقه کارکرد مکانیزم پیجینگ را نشان می دهد. توجه داشته باشید که مکان صفحه فیزیکی ، نیازی نیست که با آدرس مجازی یا حتی با ورودی جدول قبلی ، خطی باشد.خطوط ضخیم جزء برگردان (translating) در مثال هستند و خطوط باریک در مثال های پیشین چگونگی راه اندازی جداول پیج هستند.

نویسنده : حسین

پایمرد

"Page Directory Table" ورودی هایی دارد که هریک به یک ساختار "Page Directory Table" اشاره دارد.ورودی های مذکور به ابتدای صفحه در RAM فیزیکی اشاره دارد.زمانی که ویندوز و اکثر سیستم عامل ها 4K صفحه استفاده می کنند , CPU در حقیقت می تواند 4k و 2MB صفحه را پشتیبانی کند.

در صورتی که صفحات 4K تعریف شوند کل پردازش به صورت زیر لیست می شود.

۱-انتخابگر به ورودی جدول توصیف گر اشاره می کند.

۲-"Base Offset" ورودی چدول توصیف به افست آدرس مجازی اضافه شده ، آدرس خطی می سازد.

۳-رقم های 22-31 آدرس خطی در "Page Directory Table" که توسط CR3 اشاره شده است ، اندیس گذاری می شود.

۴-ورودی در "Page Directory Table" به اساس "Page Entry Table" که توسط بیت های 12-21 اندیس گذاری شده است اشاره دارد که برای بازیافتن یک "Page Directory Table" به این جدول اندیس گذاری می شود .

۵-"Page Table Entry" صرف نظر از اطلاعات در مورد اینکه آدرس در دیسک پیج شده یا نه بخ مکان پایه صفحه در حافظه فیزیکی اشاره می کند.

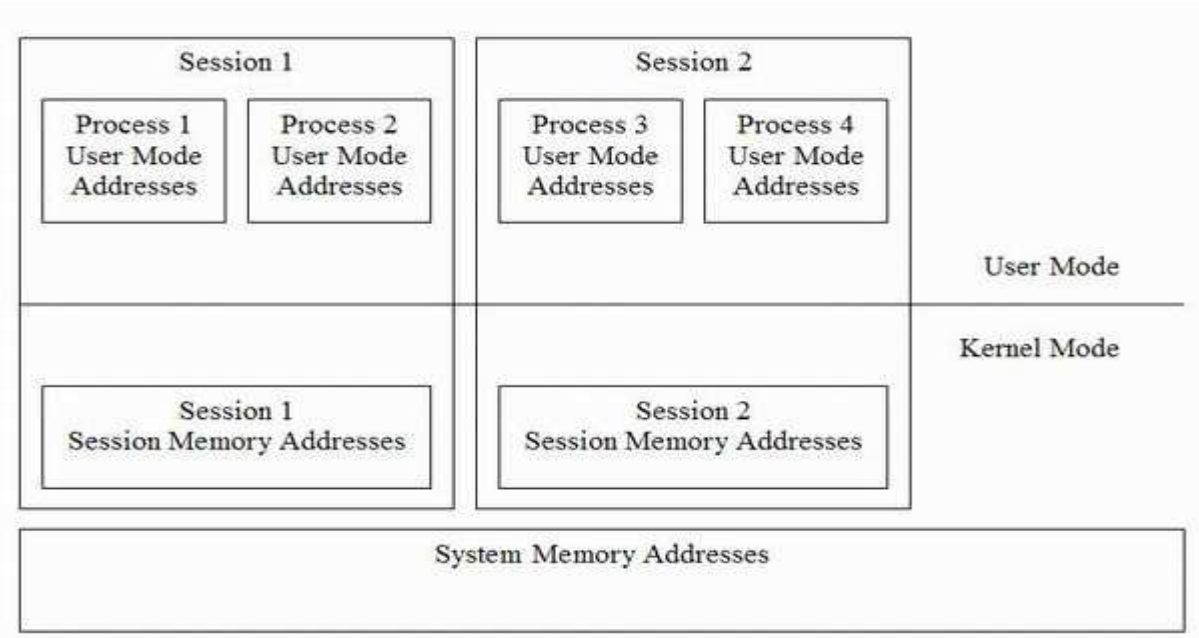
۶-رقم های باقی مانده آدرس خطی , رقم های 0-11 به ابتدای صفحه فیزیکی برای ساخت آدرس فیزیکی نهایی اضافه می شوند.

6-2-2 - عملکرد ویندوز

اگر معمولا اجرای جداول توصیف گر را رد کنید.ترجمه آدرس جهت دنبال کردن باید کاملا ساده باشد.آدرس به قسمت هایی تقسیم می شود که سبب تسهیل اندیس گذاری جدول حافظه ای که در نهایت به مکان یک صفحه فیزیکی اشاره دارد ، می شود. آخرین اندیس در پیج فیزیکی اندیس گذاری می شود.

ویندوز اساسا سه لایه مجزا ازمحدوده آدرس مجازی را انجام می دهد. اولی ، آدرس مد کاربری خواهد بود. این آدرس ها برای هر پردازش یکسان هستند.به این معنا که هر پروسه آدرس حافظه مختص به خود را در این محدوده دارد ، مسلما ، بهینه سازی وجود دارد مانند اینکه جداول صفحه متفاوت به مکان یکسان یک حافظه فیزیکی اشاره می کنند به جای اینکه کد را به اشتراک گذارند و حافظه ای که ثابت است تکثیر نکنند.

دومین محدوده از آدرس ها آن هایی هستند که در "Session Space" خواهند بود.اگر "Fast User Switching" یا "Terminal Services" را بکار برده باشید.می دانید که هر کاربر ضرورتا دسکتاپ برای خود دارد.درایورهای معینی هستند که در "Session Space" حافظه ای که برای هر Session تک است ، اجرا می شوند.در این حافظه چیزهای مثل درایور نمایش ، win32k.sys ، و بعضی از درایورهای چاپگر است.این دلیلی است بر آن که چرا ویندوز تعداد session ها را اندازه نمی گیرد. به این معنا که شما نمی توانید "FindWindow" را انجام دهید و یک پنجره در دسکتاپ کاربر دیگری مشاهده کنید. آخرین محدوده آدرس ها که به عنوان "System Space" شناخته شده است.حافظه ای است که در سرتا سر کل سیستم و در هر جا قابل دسترسی می باشد.



شکل ۲-۳- محدوده حافظه

هر زمانی که یک ترد سوئیچ می شود ، CR3 با یک اشاره گر مشخص که به جدول صفحه ای که قابل دسترسی برای آن ترد باشد ، اشاره می کند.اجرا این است که هر پردازش اشاره گر دایرکتوری صفحه خود را دارد و در CR3 بارگذاری می شود.این دقیقا به این معناست که چگئنه ویندوز پردازش ها را از یک دیگر مجزا می کند.همگی اشاره گر دایرکتوری خاص خود را دارند.این اشاره گر دایرکتوری طوری اجرا می شود که پردازش ها در جلسه یکسان ، فضای جلسه یکسان را ترسیم کنند و تمامی پردازش ها در سیستم حافظه سیستم را ترسیم کنند. محدوده حافظه ، خاص برای هر پردازشی که محدوده آدرس مد کاربری است ، اجرا می شود.

2-2-6-1- سوئیچ PAE/

PAE به عنوان "Physical Address Extension" خوانده می شود. اساسا به این معناست که سیستم عامل می تواند ۳۶ رقم حافظه فیزیکی را در ۳۲ رقم ترسیم کند و این طور نیست که شما می توانید به 4GB از حافظه در یک زمان دسترسی پیدا کنید.به این صورت است که آدرس های حافظه بالاتر میتوانند در ۳۲ رقم ترسیم شوند یعنی این که قابل دسترسی هستند همچنین به این معناست که سیستم عامل می تواند با این قابلیت از ماشین با 4 GB حافظه فیزیکی استفاده کند.بنابراین زمانی که یک پردازش امکان دسترسی به 4GB را ندارد , سیستم عامل می تواند حافظه را طوری مدیریت کند که صفحه حافظه های بیشتری را در یک زمان در دسترس نگه دارد. همچنین API وجود دارد که برنامه می تواند برای مدریت حافظه و به کار گیری 4GB از حافظه مورد استفاده قرار دهد. این ها "AWE" یا Address Windows Extensions نامیده می شوند.

2-2-6-2- سوئیچ 3GB/

سوئیچی وجود دارد که ممکن است در مورد آن شنیده باشید که 3GB/ خوانده می شود. به کاربر این اجازه را می دهد که 3 GB فضای آدرس داشته باشد.معمولا , محدوده 4GB به دو قسمت تقسیم می شود.2GB فضای آدرس برای مد کاربر و 2GB برای مد هسته است. به این معناست که آدرس های مد کاربری رقم پر ارزش (رقم ۳۱) را در زمانی که مد هسته رقم ۳۱ را داشته باشد نمی تواند داشته باشد.این به این معناست که 0*78000000 یک آدرس مد کاربر است وقتی 0*82000000 یک آدرس مد هسته است.تنظیم 3GB/ سوئیچ به پردازش مد کاربر این اجازه را خواهد داد که حافظه بیشتری را نگه دارد ام هسته حافظه کمتری خواهد داشت. معایب و محاسنی برای این سوئیچ جود دارد.

اصلی ترین محاسن استفاده :

برنامه هایی که به مقدار زیادی حافظه احتیاج دارند بهتر کار خواهند کرد در صورتی که از فوائد این روش مطلع باشند.اگر از حافظه مد کاربر برای ذخیره داده استفاده کنید جابجایی کمتری در دیسک وجود خواهد داشت.

اصلی ترین ایراد استفاده:

۱-حافظه مد کاربر زیادی در دسترس وجود ندارد بنابراین برنامه ها و عملیاتی که به حافظه هسته زیادی احتیاج دارند توانایی اجرا پیدا نخواهند کرد.

۲-برنامه ها و درایورهایی که رقم پرارزش را بررسی میکنند و برای تعیین حافظه مد هسته در مقابل حافظه مد کاربر از این بررسی استفاده می کنند به درستی کار نخواهند کرد.

۲-۳- درایورنویسی USB

۲-۳-۰- مقدمه

نکته اصلی USB آسودگی خیال کاربران است. ایده اصلی Plug and Play (PnP) فرایند نصب سخت افزار های مشخص مانند دستگاه جانبی واسط USB را بر روی کامپیوترها ساده کرده است.اگر چه , موضوع های پیکربندی باعث دردسر بسیار کاربران دستگاه هایی چون کیبورد و ماوس و وسایل جانبی سریال و موازی است ولی در دسترس بودن پورت یکی از عامل های اصلی است که از گذشته باعث محدود کردن ازدیاد وسایل جانبی , از قبیل مودم ها, اسکنرها , دستگاه های پاسخگو (Answering Machine) و دستیاران دیجیتال (Personal Digital Assistants)شده است. USB بوسیله تهیه یک روش متحدالشکل از اتصال بالقوه تعداد زیادی دستگاه خود تشخیص (self_identifying) از طریق تنها یک پورت کامپیوتر این مشکلات را حل کرده است.

شما می توانید دستگاه USB را وصل و یا قطع نمایید بدون اینکه از بابت باز یا بسته کردن برنامه های کاربردیی که از آن وسیله استفاده می کنند و یا از صدمات الکتریکی نگرانی داشته باشید.

درایورنویسی USB دردو بخش عمده پوشش داده می شود:

- **معماری USB**

  این بخش شامل چندین قسمت از معماری USB شامل ترنزکشن ها , فریم ها , پاکت ها و چهار روش مختلف انتقال می شوند که قبلا توضیح داده شده اند.

- **خصوصیات ویژه مدل درایور ویندوز (Windows Driver Model) برای درایورنویسی USB**

  به جای ارتباط مستقیم با سخت افزار از طریق فراخوانی تابع های لایه انتزاع سخت افزار (HAL) , یک درایور USB به طور چشمگیری متکی بر درایور باس و یک کتابخانه از مد هسته به نام USBD.SYS است. درایور USB برای ارسال درخواست به دستگاهش , یک بلوک درخواست (URB) تهیه می کند و به درایور باس ارایه می کند. درایور باس به نوبت در زمانبندی های منظم و بنابر پهنای باند لازم و در دسترس , خواسته ها را به باس تحویل می دهد. در این قسمت که در ادامه بحث و بررسی می شود , موارد پیشرفته و تخصصی برنامه نویسی درایور مانند مدیریت توان و چند سرنخی مورد توجه قرار نگرفته است . در این بخش با اسباب مورد نیاز برای درایور نویسی از نوع USB آشنا می شوید و اصول اصلی پیکربندی دستگاه USB خود از طریق درایور را فرا می گیرید.

در این بخش از پایان نامه سعی شده است تا مطا لب به صورت مرجع گونه بیان شود . مطالعه این بخش مستلزم اطلاعاتی از پروتکل USB و اصول درایور نویسی می باشد. در این بخش فرض بر این است که خواننده با درایورباس که همان کنترلر میزبان می باشد و DDK (نرم افزاری برای ساخت و کامپایل درایورها, شامل کتابخانه های مفید و ماکروهای مورد نیاز) و همچنین نحوه عمل IRP ها (بسته های درخواست ورودی / خروجی , که وظیفه اصلی تبادل درخواست ها و اطلاعات را در سطح هسته دارند) و برنامه نویسی C آشنایی کافی را دارد.

۲-۳-۱- معماری برنامه نویسی (Programming Architecture)

2-3-1-1- USB و ارتباط با درایورباس

در مقایسه با درایور های دستگاه هایی که به گذرگاه های تعبیه شده کامپیوتر متصل می شوند مانند PCI , یک درایور دستگاه USB هیچ گاه به طور مستقیم به سخت افزارش متصل نمی شود.در عوض , درایوریک نمونه از ساختار داده را تهیه می بیند که به عنوان بلوک درخواست USB (Urb) شناخته می شودو آنها را به درایور های والد ارایه می کند . شما Urb ها را با استفاده از یک پاکت درخواست ورودی, خروجی (IRP) با کد تابع اصلی IRP_MJ_INTERNAL_DEVICE_CONTROL به درایور مولد ارایه می کنید. در بعضی مواقع شما می توانید با استفاده از واسط فراخوان مستقیم (direct-call) درایور مولد یک تابع را صدا بزنید. درایور مولد به نوبت در زمان های منظم در فریم ها ,کد بندی را در Urbانجام می دهد.

در این قسمت ما مکانیسم کار با درایور های مولد را برای اجرای فرامین یک درایور دستگاه USB را توصیف می کنیم. در ابتدا چگونگی ساخت و تحویل یک Urb را بحث می کنیم , سپس در مورد مکانیسم پیکر بندی و یا دوباره پیکر بندی دستگاه بحث می کنیم . ودر پایان به صورت مختصر در موردچگونگی مدیریت درایور دستگاه USB بر چهار نوع انتقال از لوله های ارتباطی توضیح می دهیم.

2-3-1-1-1- شروع درخواست (Initiating Requests)

با ساختن یک Urb , شما فضایی از حافظه را به ساختار Urb تخصیص می دهید و درخواست شروع یک روتین را برای نوشتن فیلد های مختص ارتباط مورد نظر خود می نمایید. برای مثال فرض کنید که شما در حال آغاز پیکر بندی دستگاه در پاسخ به درخواست IRP_MN_START_DEVICE هستید. یکی از اولین کار های شما خواندن توصیف گر ها است . شما باید کد مختصر زیر را برای انجام این وظیفه به کار برید.

```
USB_DEVICE_DESCRIPTOR dd;
URB urb;
UsbBuildGetDescriptorRequest(&urb,
```

```
sizeof(_URB_CONTROL_DESCRIPTOR_REQUEST),
USB_DEVICE_DESCRIPTOR_TYPE, 0, 0, &dd, NULL,sizeof(dd),
NULL);
```

شما در ابتدا یک متغییر محلی (urb) برای نگهداری ساختار URB تعریف می کنید. URB در فایل سرآیند (USBDI.H) به عنوان یک یونیون از چندین زیر ساختار که هرکدام برای درخواست های دستگاه USB لازم هستند تعریف شده است. ما قصد داریم از زیر ساختار UrbControldescriptorRequest از یونیون URB , که به عنوان یک نمونه از ساختار _URB_CONTROL_DESCRIPTOR_REQUEST تعریف شده است استفاده کنیم .استفاده از یک متغییر اتوماتیک مانند این در صورتی مناسب است که شما از وجود فضای لازم در پشته برای نگهداری تعداد زیادی از URB های ممکن مطمئن باشید و همچنین تا کامل شدن هر URB صبر کنید قبل از اینکه متغییر را از بین ببرید. شما همچنین می توانید به صورت پویا حافظه ای را از قسمت heap برای URB ها در نظر بگیرید اگر مایل باشید:

```
PURB urb = (PURB) ExAllocatePool(NonPagedPool,
  sizeof(_URB_CONTROL_DESCRIPTOR_REQUEST));
if (!urb)
  return STATUS_INSUFFICIENT_RESOURCES;
UsbBuildGetDescriptorRequest(urb, ...);

ExFreePool(urb);
```

UsbBuildGetDescriptorRequest شبیه یک سرویس روتین معمولی به کار می رود در حالی که در حقیقت آن یک ماکرو تعریف شده در USBDLIB.H است که بنیان درون برنامه ای را برای مقداردهی اولیه فیلد های دریافت زیرساختارهای درخواست توصیف گر تولید می کند. فایل های سرآمد (Driver Development Kit) DDK یکی از این ماکروها را برای اغلب انواع URB هایی که می خواهید بسازید معین می کند.(جدول زیر را ببینید). به عنوان یک اصل کلی در ماکرو های پیش پردازنده شما باید از استفاده عبارت هایی که بر روی آرگومان های این ماکروها تاثیر می گذارند اجتناب کنید.

Helper Macro	Type of Transaction
UsbBuildInterruptOrBulkTransferRequest	Input or output to an interrupt or bulk endpoint
UsbBuildGetDescriptorRequest	*GET_DESCRIPTOR* control request for endpoint 0
UsbBuildGetStatusRequest	*GET_STATUS* request for a device, an interface, or an endpoint
UsbBuildFeatureRequest	*SET_FEATURE* or *CLEAR_FEATURE* request for a device, an interface, or an endpoint
UsbBuildSelectConfigurationRequest	*SET_CONFIGURATION*
UsbBuildSelectInterfaceRequest	*SET_INTERFACE*
UsbBuildVendorRequest	Any vendor-defined control request

جدول۲-۱ ماکروهای کمکی برای ساخت URB ها

در کد برنامه های قبلی , ما قصد داشتیم اطلاعات توصیف گر های دستگاه را در متغییر محلی (dd) با آدرس و طولی که ما تایین کرده بودیم بدست آوریم . URB های درگیر تبادل داده, به شما اجازه می دهند به دو روش یک بافر دیتا nonpaged را تعریف کنید. شما می توانید یک طول وآدرس مجازی از بافر تعیین کنید (کاری که در کدهای قبلی انجام شد). همچنین شما می توانید یک لیست توصیف گر حافظه (MDL) را پس از انجام مرحله probe-and-lock , با فراخوانی MmProbeAndLockPages تولید کنید.

2-1-1-3-2- ارسال یک URB (Sending a URB)

با داشتن یک URB , شما احتیاج به خلق و ارسال یک درخواست درونی IOCTL به درایور مولد دارید (IOCTL درخواست هایی که در سلسله مراتب پایین تری از درایورها قرار می گیرند). در بیشتر مواقع شما باید تا ارسال پاسخ دستگاه منتظر بمانید. در روتین کمکی زیر به این نیاز ها پاسخ داده شده است.

```
NTSTATUS SendAwaitUrb(PDEVICE_OBJECT fdo, PURB urb)
  {
  PDEVICE_EXTENSION pdx =
    (PDEVICE_EXTENSION) fdo->DeviceExtension;
  KEVENT event;
  KeInitializeEvent(&event, NotificationEvent, FALSE);
  IO_STATUS_BLOCK iostatus;
  PIRP Irp = IoBuildDeviceIoControlRequest
    (IOCTL_INTERNAL_USB_SUBMIT_URB, pdx-
>LowerDeviceObject,
    NULL, 0, NULL, 0, TRUE, &event, &iostatus);
  PIO_STACK_LOCATION stack = IoGetNextIrpStackLocation(I
rp);
  stack->Parameters.Others.Argument1 = (PVOID) urb;
  NTSTATUS status = IoCallDriver(pdx-
>LowerDeviceObject, Irp);
  if (status  == STATUS_PENDING)
    {
    KeWaitForSingleObject(&event, Executive, KernelMode,
      FALSE, NULL);
    status = iostatus.Status;
    }
  return status;
  }
```

این یک مثال ساده از خلق وارسال یک IRP سنکرون به یک درایور دیگر بود. تنها سختی کار, روش دقیق قراردادن حروف URB داخل پاکت INTERNAL_DEVICE_CONTROL , با قراردادن فیلد Parameters.Others.Argument1 ازپشته به اشاره گر URB انجام پذیراست.

2-3-1-1-3- وضعیت برگشتی ازURB ها (Status Returns from URBs)

هنگامی که شما یک URB را به درایور باس ارسال می کنید, نتیجتا شما یک کد NTSTATUS دریافت می کنید که نتیجه عمل انجام شده را توصیف می کند. درایور باس به طور درونی , گونه دیگری از کدهای وضعیت را با نوع مشخصه USBD_STATUS استفاده می کند. این کدها از نوع کدها ی NTSTATUS نیستند.

هنگامی که یک درایور مولد یک URB را کامل می کند , آن فیلد UrbHeader.Status ی URB , رابا یکی از مقدارهای USBD_STATUS مقداردهی می نماید. شما می توانید این مقدار را در درون درایورتان تست کنید تا اطلاعاتی در مورد URB خود بدست آورید.

ماکرو URB_STATUS درون DDK این فرآیند را آسان کرده است:

```
NTSTATUS status = SendAwaitUrb(fdo, &urb);
USBD_STATUS ustatus = URB_STATUS(&urb);
```

اگر چه پروتکل مشخصی برای نگهداری این وضعیت و پس فرستادن آن به برنامه کاربردی وجود ندارد ولی شما در کار کردن با آن کاملا آزاد هستید.

2-3-2- پیکربندی (Configuration)

درایور باس USB به صورت خودکار اتصال یک دستگاه USB را تشخیص می دهد. آن سپس شروع به خواندن توصیف گر های آن دستگاه می نماید تا از نوع دستگاه اطلاع یابد.فیلدهای مشخصه محصول و فروشنده توصیف گربه اضافه سایر توصیف گرها , مشخص می کنند که کدام درایور(راه انداز) باید بار گذاری شود.

مدیر پیکربندی تابع AddDevice درایور رادر حالت های معمول فراخوانی می کند.این تابع تمام کارهارا انجام می دهد از قبیل: خلق کردن یک Device Object , ارتباط این Device Object به درون سلسله مراتب درایور, و سایر وظایف. سرانجام مدیر پیکر بندی یک درخواست Plug and Play با عنوان IRP_MN_START_DEVICE به درایور می فرستد. شما باید با فراخوانی یک تابع کمکی به نام StartDevice با آرگومان های توصیف شده به عنوان منبع تخصیص بازگشتی یا غیر بازگشتی برای دستگاه , آن خواسته را کنترل و با آن رفتار کنید. شما نیازی نیست که از بابت منابع ورودی یا خروجی در درایورهای USB نگرانی ای داشته باشید, چون آن ها در این قسم از درایور ها وجود ندارند.

بنابراین شما می توانید یک تابع کمکی StartDevice را با اسکلت بندی زیر بنویسید:

```
NTSTATUS StartDevice(PDEVICE_OBJECT fdo)
  {
  PDEVICE_EXTENSION pdx =
    (PDEVICE_EXTENSION) fdo->DeviceExtension;
  <configure device>
  return STATUS_SUCCESS;

  }
```

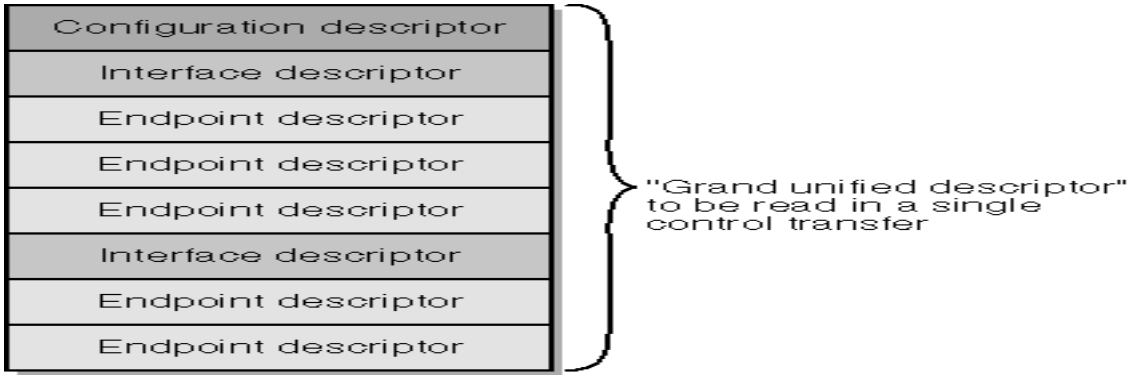
در مرحله پیکر بندی وسیله, شما نیاز به نوشتن مقدار زیادی کد برای پیکربندی سخت افزار دارید . اما همان طور که گفته شد شما نگرانی ای از بابت پورت های ورودی , خروجی و وقفه ها ,آبجکت های وفق دهنده دستیابی مستقیم به حافظه (DMA) یا هرعنصر resource-oriented دیگری ندارید. یک دید اجمالی از آنچه که شما احتیاج دارید در StartDevice انجام دهید , به صورت زیر است. در ابتدا شما یک پیکر بندی برای دستگاه انتخاب می کنید.

در اکثر دستگاه ها فقط یک نوع پیکربندی پشتیبانی می شود. در حالی که شما پیکربندی را انتخاب می کنید, شما یک یا چند واسط را که جزیی از پیکربندی است انتخاب می کنید. پشتیبانی کردن از چند واسط برای یک دستگاه , غیر رایج نیست. بعد از انتخاب پیکربندی و مجموعه واسط ها , شما یک URB پیکربندی انتخابی را به درایور باس می فرستید. درایورباس به نوبت به دستگاه برای فعال کردن پیکربندی و واسط ها دستور می دهد.

درایور باس لوله هایی را تهیه می نماید که به شما اجازه می دهد با اندپوینت های واسط انتخابی ارتباط برقرار کنید و دستگیره هایی را تولید می کند که شما می توانید بوسیله آن ها به لوله ها دست یابی داشته باشید.آن همچنین دستگیره هایی برای پیکربندی و واسط ها خلق می کند. شما دستگیره ها رااز URB های کامل شده استخراج می نمایید و آن ها را برای استفاده های بعدی حفظ می کنید.

1-2-3-2- خواندن یک توصیف گر پیکربندی (Reading a Configuration Descriptor)

فکرکردن به یک توصیف گر پیکربندی با طول ثابت در سر آمد یک ساختار با طول متغیر که پیکربندی , همه واسط هایش و همه اندپوینت های واسط ها یش را توصیف می کند بهترین ایده است.



شکل ۲-۴- توصیف گر پیکربندی

شما باید معنی تمام ساختار , باطول متغییر یک محیط پیوسته از حافظه را بدانید, زیرا سخت افزار به شما اجازه دستیابی مستقیم به توصیف گر های واسط و اندپوینت را نمی دهد. متاسفانه در آغاز شما اطلاعی از طول ساختار های ترکیبی ندارید . کد ذکر شده در زیر به شما نشان می دهد که چگونه می توانید از دو URB برای خواندن توصیف گر پیکربندی استفاده کنید:

```
ULONG iconfig = 0;
URB urb;
USB_CONFIGURATION_DESCRIPTOR tcd;
UsbBuildGetDescriptorRequest(&urb,
  sizeof(_URB_CONTROL_DESCRIPTOR_REQUEST),
  USB_CONFIGURATION_DESCRIPTOR_TYPE,
  iconfig, 0, &tcd, NULL, sizeof(tcd), NULL);
SendAwaitUrb(fdo, &urb);
ULONG size = tcd.wTotalLength;
PUSB_CONFIGURATION_DESCRIPTOR pcd =
  (PUSB_CONFIGURATION_DESCRIPTOR) ExAllocatePool(
  NonPagedPool, size);
UsbBuildGetDescriptorRequest(&urb,
  sizeof(_URB_CONTROL_DESCRIPTOR_REQUEST),
  USB_CONFIGURATION_DESCRIPTOR_TYPE,
  iconfig, 0, pcd, NULL, size, NULL);
SendAwaitUrb(fdo, &urb);
.
ExFreePool(pcd);
```

دراین قطعه کد, ما یک URB را برای خواندن یک توصیفگر پیکربندی داخل یک فضا ی توصیف گر موقت به نام tcd درنظرگرفتیم.ایندکس صفربرای اولی به کار رفته است. این توصیفگرحاوی طول ساختار ترکیبی است (wTotalLength) که شامل توصیف گر های پیکربندی , واسط و اندپوینت می شود. ما این فضا را تخصیص می دهیم و URB دوم را برای خواندن کامل توصیف گر استفاده می کنیم. در پایان پروسس متغیر pcd به تمام آن چه ما نیاز داریم اشاره می کند.

2-2-3-2- انتخاب پیکربندی (Selecting Configuration)

سرانجام شما باید یک پیکربندی را با فرستادن یک سری دستورات کنترلی به دستگاه انتخاب کنید و واسط دلخواه را فعال سازید. ما از تابع USBD_CreateConfigurationRequestEx برای خلق URB این سری دستورات استفاده می نماییم. یکی از آرگومان های این تابع یک بردار از اشاره گر ها به توصیف گر های واسطی است که ما قصد فعال کردن آن ها را داریم. مرحله بعدی پیکر بندی بعد از معین کردن پیکربندی مورد استفاده , مهیا کردن این بردار است.

- **خواندن یک توصیف گر رشته**

برای گزارش یا قصد های دیگر ممکن است شما بخواهید چند تا از توصیف گرهای رشته ای را که دستگاه شما تهیه دیده است بازیابی کنید. در زیر کدهای تابع کمکی برای خواندن این گونه توصیف گرها آورده شده است.بخش جالب و جدید این تابع قالب بندی URB برای واکشی یک توصیف گر رشته است. علاوه برتهیه شاخص رشته دلخواه, ما نیاز به یک مشخصه زبان به نام LANGID داریم که مشابه شاخص های زبان برنامه های ویندوز است . دستگاه ها می توانند رشته هایی در چندین زبان تهیه کنند و توصیف گر رشته ۰ شامل یک لیست ازشاخص زبان های پشتیبانی شده می باشد.من در ابتدا اجرای این روتین این رشته را می خوانم وبطور دلخواه اولین زبان را انتخاب می کنم. درایور مولد این شاخصه را همراه با شاخص رشته به عنوان یک پارامتر برای گرفتن درخواست توصیف گری که به دستگاه فرستاده , عبور می دهد. دستگاه خود مسئول تصمیم گیری در مورد رشته بازگشتی است.

خروجی تابع GetStringDescriptor به صورت یک رشته یونیکد است و در پایان شما RtlFreeUnicodeString را فراخوان می کنید تا باعث آزاد سازی بافر رشته شود.

```
NTSTATUS GetStringDescriptor(PDEVICE_OBJECT fdo, UCHAR i
string,
  PUNICODE_STRING s)
  {
  NTSTATUS status;
```

```
  PDEVICE_EXTENSION pdx =
    (PDEVICE_EXTENSION) fdo->DeviceExtension;
  URB urb;

  UCHAR data[256];

  if (!pdx->langid)
    {
    UsbBuildGetDescriptorRequest(&urb,
      sizeof(_URB_CONTROL_DESCRIPTOR_REQUEST),
      USB_STRING_DESCRIPTOR_TYPE,
      0, 0, data, NULL, sizeof(data), NULL);
    status = SendAwaitUrb(fdo, &urb);
    if (!NT_SUCCESS(status))
      return status;
    pdx->langid = *(LANGID*)(data + 2);
    }

  UsbBuildGetDescriptorRequest(&urb,
    sizeof(_URB_CONTROL_DESCRIPTOR_REQUEST),
    USB_STRING_DESCRIPTOR_TYPE,
    istring, pdx-
>langid, data, NULL, sizeof(data), NULL);
  status = SendAwaitUrb(fdo, &urb);
  if (!NT_SUCCESS(status))
    return status;

  ULONG nchars = (data[0] -
 sizeof(WCHAR)) / sizeof(WCHAR);
  if (nchars > 257)
    nchars = 257;
  PWSTR p = (PWSTR) ExAllocatePool(PagedPool,
    (nchars + 1) * sizeof(WCHAR));
  if (!p)
    return STATUS_INSUFFICIENT_RESOURCES;
  RtlCopyMemory(p, data + 2, nchars * sizeof(WCHAR));
  p[nchars] = 0;
  s->Length = (USHORT) (sizeof(WCHAR) * nchars);
  s->MaximumLength = (USHORT) ((sizeof(WCHAR) * nchars)
    + sizeof(WCHAR));
  s->Buffer = p;

  return STATUS_SUCCESS;
  }
```

یاد آوری می شود که وقتی ما توصیف گر پیکربندی را می خوانیم ما همچنین تمام توصیف گر های واسط درون حافظه مجاور را نیز می خوانیم. این حافظه شامل یک مجموعه از توصیف گر ها است: یک توصیف گر پیکربندی, یک توصیف گرواسط به همراه تمام اندپوینت هایش و دیگر توصیف گر های واسط به همراه اندپوینت هایش و غیره . یکی از راه های انتخاب واسط ها تجزیه و تحلیل این مجموعه از توصیف گرها و یادآوری آدرس های توصیف گرهای واسط دلخواه می باشد.

درایور باس یک روتین به نام USBD_ParseConfigurationDescriptorEx برای آسان سازی این فرآیند تهیه کرده است.

```
PUSB_INTERFACE_DESCRIPTOR pid;
pid = USBD_ParseConfigurationDescriptorEx(pcd, StartPosi
tion,
  InterfaceNumber, AlternateSetting, InterfaceClass,
  InterfaceSubclass, InterfaceProtocol);
```

در این تابع , pcd آدرس توصیف گر پیکربندی می باشد. StartPosition نیز آدرس توصیف گر پیکربندی یا آدرس توصیف گری است که شما می خواهید داخلش جستجو کنید. پارامترهای باقیمانده نیزمعیارهای جستجوی توصیف گر را تعیین می کنند. مقدار ۱ نشان دهنده آن است که شما نمی خواهید هیچ معیاری را در جستجو به کار برید. شما می توانید به دنبال توصیف گر واسط بعدی که شامل هیچ یا چند تا از خواص زیر می شود بگردید:

- The given InterfaceNumber
- The given AlternateSetting index
- The given InterfaceClass index
- The given InterfaceSubclass index
- The given InterfaceProtocol index

هنگامی که USBD_ParseConfigurationDescriptorEx یک توصیف گر واسط را بر می گرداند, شما آن را به عنوان عضو توصیف گر واسط یک عامل , داخل بردار ساختار های USBD_INTERFACE_LIST_ENTRY حفظ می کنیدو سپس به قبل بر می گردید تا توصیف گر بعدی را تجزیه و تحلیل کنید. بردار سلول های لیست واسط یکی از پارامترهای فراخوانی احتمالی USBD_CreateConfigurationRequestEx خواهد بود. هر سلول این بردار یک نمونه از ساختار زیر است:

```
typedef struct _USBD_INTERFACE_LIST_ENTRY {
```

```
    PUSB_INTERFACE_DESCRIPTOR InterfaceDescriptor;
    PUSBD_INTERFACE_INFORMATION Interface;
} USBD_INTERFACE_LIST_ENTRY, *PUSBD_INTERFACE_LIST_ENTRY
;
```

هنگامی که شما یک سلول را در بردار مقدار دهی اولیه می نمایید , شما قطعه توصیف گر واسط را مساوی آدرس یک توصیف گر واسط که قصد فعال کردنش را دارید قرار می دهید و قطعه واسط را NULL می کنید. شما یک سلول برای هر واسط تعریف می کنید و سپس سلول دیگری را با InterfaceDescriptor=NULL برای نشان دادن پایان اضافه می کنید.

در مثال زیر با فرض وجود یک واسط کد ها ی زیر برای ایجاد لیست واسط استفاده شده اند:

```
PUSB_INTERFACE_DESCRIPTOR pid =
  USBD_ParseConfigurationDescriptorEx(pcd, pcd, -1, -1,
  -1, -1, -1);
USBD_INTERFACE_LIST_ENTRY interfaces[2] = {
  {pid, NULL},
  {NULL, NULL},
  };
```

که ما توصیفگر پیکربندی را برای قراردادن اولین توصیف گر واسط تجزیه می کنیم .سپس یک آرایه دو بعدی برای تشریح, آن تک مدار واسط تعریف می کنیم.

اگر شما نیاز به فعال کردن تعداد بیشتر واسط به خاطر پشتیبانی دستگاه مرکب خود داشته باشید شما می توانید فرخوانی عمل تجزیه را تکرار کنید. برای مثال:

```
۱-
ULONG size = (pcd->bNumInterfaces + 1) *
    sizeof(USBD_INTERFACE_LIST_ENTRY);
  PUSBD_INTERFACE_LIST_ENTRY interfaces =
    (PUSBD_INTERFACE_LIST_ENTRY) ExAllocatePool(NonPaged
Pool,
    size);
  RtlZeroMemory(interfaces, size);
  ULONG i = 0;
  PUSB_INTERFACE_DESCRIPTOR pid =
    (PUSB_INTERFACE_DESCRIPTOR) pcd;
۲-
  while ((pid = USBD_ParseConfigurationDescriptorEx(pcd,
    pid, ...)))
۳-
    interfaces[i++].InterfaceDescriptor = pid++;
```

۱- ما ابتدا حافظه ای را برای سلول های لیست واسط به اندازه واسط هایی که در این پیکربندی وجود دارند باضافه یک تخصیص می دهیم. ما بردارهای سلول را صفر قرار می دهیم. هرجا که ما مقداردهی برداررا در حلقه بعدی متوقف کنیم , سلول بعدی برای نشان دادن پایان بردار NULL خواهد بود.

۲- فراخوانی تجزیه تمام معیار های مربوط به دستگاه شما را دربر می گیرد. در اولین تکرار حلقه ,pid به توصیف گر پیکربندی اشاره می کند. در تکرار های بعدی, آن به توصیف گرواسط ماقبل, برگشت داده شده از فراخوانی قبلی اشاره می کند.

۳- در اینجا ما اشاره گر را مقدار دهی اولیه می کنیم تا به یک توصیف گر واسط اشاره کند. درهر افزایش از تکرار بعدی ما المان بعدی آرایه را مقداردهی اولیه می کنیم. هر افزایش در pid باعث ارتقاء ما از پیکربندی واسط جاری به پیکربندی واسط بعدی برای تجزیه می شود.

مرحله بعدی در فرآیند پیکربندی خلق URB است که ما نیاز به ارسال آن برای پیکربندی دستگاه داریم:

```
PURB selurb = USBD_CreateConfigurationRequestEx(pcd, int
erfaces);
```

به علاوه خلق یک URB , USBD_CreateConfiguratioRequestEx عضوهای واسط سلول های USBD_INTERFACE_LIST شما را برای اشاره کردن به ساختارهای USBD_INTERFACE_INFORMATION مقداردهی اولیه می نماید. این ساختارهای اطلاعاتی به صورت فیزیکی در بلوک های حافظه مشابه با URB قرار گرفته اندو بنابراین هنگامی که در پایان شما تابع ExFreePool را برای بازگرداندن URB ها صدا می زنید , آزاد شده وبه heap بر می گردند. یک ساختار اطلاعاتی واسط به صورت زیر تعریف می شود:

```
typedef struct _USBD_INTERFACE_INFORMATION {
  USHORT Length;
  UCHAR InterfaceNumber;
  UCHAR AlternateSetting;
  UCHAR Class;
  UCHAR SubClass;
  UCHAR Protocol;
  UCHAR Reserved;
  USBD_INTERFACE_HANDLE InterfaceHandle;
```

```
  ULONG NumberOfPipes;
  USBD_PIPE_INFORMATION Pipes[1];
  } USBD_INTERFACE_INFORMATION, *PUSBD_INTERFACE_INFORMA
TION;
```

بردارساختار اطلاعات لوله چیزی است که برای ما مهم است زیرا بقیه فیلدهای ساختار وقتی که ما URB ها را می فرستیم , توسط درایور مولد مقداردهی می شوند. . هر کدام از آن ها مانند زیر است:

```
typedef struct _USBD_PIPE_INFORMATION {
  USHORT MaximumPacketSize;
  UCHAR EndpointAddress;
  UCHAR Interval;
  USBD_PIPE_TYPE PipeType;
  USBD_PIPE_HANDLE PipeHandle;
  ULONG MaximumTransferSize;
  ULONG PipeFlags;
  } USBD_PIPE_INFORMATION, *PUSBD_PIPE_INFORMATION;
```

بنابراین ما یک آرایه از سلول های USBD_INTERFACE_LIST داریم که هر کدامشان به ساختار USBD_INTERFACE_INFORMATION که در بر گیرنده یک آرایه ازساختار های USBD_PIPE_INFORMATION است , اشاره می کنند. وظیفه بلادرنگ ما , پر کردن عضو MaximumTransferSize از هر کدام از ساختارهای اطلاعاتی لوله است البته اگر میل به پر کردن آن از جانب درایور مولد نباشیم. مقدار پیش فرض

USBD_DEFAULT_MAXIMUM_TRANSFER_SIZE است که برابر مقدار PAGE_SIZE در DDK می باشد. مقداری که ما تعیین می کنیم رابطه مستقیمی با ماکزیمم سایز پاکت برای اندپوینت , یا به مقدار داده ای که اندپوینت در هر ترنزکشن می تواند جذب کند , ندارد. در عوض , آن نشان دهنده بزرگترین مقداری ازداده است که ما سعی داریم با یک URB منتقل کنیم. این مقدار می تواند از مقدار داده ای که برنامه کاربردی به درایور می فرستد یا از درایور دریافت می کند کمتر باشد. در این حالات درایور باید برای شکستن درخواست برنامه کاربردی به قطعاتی کوچکتر یا هم اندازه با این ماکزیمم سایز آماده شود. در قسمت مدیریت لول های تبادل توده ای در مورد این موضوع بحث می شود.

هدف از تهیه ماکزیمم سایز تبا دل , ریشه در الگوریتم زمان بندیی که درایور کنترلر میزبان از آن برای تقسیم درخواست های URB هر ترنزکشن به فریم های باس استفاده می کند , دارد. اگر ما یک مقدار بزرگ از داده را ارسال کنیم , احتمال این وجود دارد که داده ما یک فریم از وسیله دیگر را دربر گیرد , بنابراین ما می خواهیم

مطالباتمان را بوسیله معین کردن یک ماکزیمم سایز معقول , برای URB ها یی که در هر بار می فرستیم اداره کنیم.

کد مورد نیاز برای مقداردهی اولیه ساختار های اطلاعاتی لوله از جهاتی مانند این است:

```
for (ULONG ii = 0; ii < <number of interfaces>; ++ii)
  {
  PUSBD_INTERFACE_INFORMATION pii = interfaces[ii].Inter
face;
  for (ULONG ip = 0; ip < pii->NumberOfPipes; ++ip)
    pii-
>Pipes[ip].MaximumTransferSize = <some constant>;
  }
```

هرزمان که مقداردهی اولیه ساختارهای اطلاعاتی لوله تمام شد, شما آماده ارسال URB پیکربندی هستید:

```
SendAwaitUrb(fdo, selurb);
```

2-3-3- یافتن دستگیره ها (Finding The Handles)

اتمام موفقیت آمیز URB پیکر بندی انتخابی , با واگذاری چند مقدار دستگیره توام است که شما باید آنها را برای استفاده های بعدی حفظ کنید :

- عضو UrbSelectConfiguration.ConfigurationHandle از URB یک دستگیره برای پیکربندی است.
- عضو InterfaceHandle از ساختار USBD_INTERFACE_INFORMATION در بر گیرنده یک دستگیره برای واسط است.
- هر کدام از ساختار های USBD_PIPE_INFORMATION حاوی یک دستگیره لوله برای اتمام لوله مورد مکاتبه است.

برای مثال , کد زیردو مقدار دستگیره را حفظ می کند:

```
typedef struct _DEVICE_EXTENSION {
.
.
  USBD_CONFIGURATION_HANDLE hconfig;
  USBD_PIPE_HANDLE hpipe;
  } DEVICE_EXTENSION, *PDEVICE_EXTENSION;
pdx->hconfig = selurb-
>UrbSelectConfiguration.ConfigurationHandle;
pdx->hpipe = interfaces[0].Interface-
>Pipes[0].PipeHandle;

ExFreePool(selurb);
```

2-3-4- خاتمه دادن وسیله (Shutting Down the Device)

هنگامی که درایور یک درخواست IRP_MN_STOP_DEVICE دریافت می کند, شما باید با خلق و ارسال یک درخواست پیکربندی انتخابی با اشاره گر پیکربندی تهی , دستگاه را به حالت پیکربندی نشده در آورید:

```
URB urb;
UsbBuildSelectConfigurationRequest(&urb,
  sizeof(_URB_SELECT_CONFIGURATION), NULL);
SendAwaitUrb(fdo, &urb);
```

۲-۳-۵- مدیریت لوله

در ادامه مدیریت لوله را در انواع مختلف تبادل داده USB ,به صورت مختصر توضیح می دهیم.

۲-۳-۵-۱- مدیریت لوله های تبادل توده ای (Managing Bulk Transfer Pipes)

کد زیر شامل یک اندپوینت توده ای ورودی است که باهر بار خواندن مقدار ثابتی را بر می گرداند:

```
URB urb;
UsbBuildInterruptOrBulkTransferRequest(&urb,
  sizeof(_URB_BULK_OR_INTERRUPT_TRANSFER),
  pdx->hpipe, Irp-
>AssociatedIrp.SystemBuffer, NULL, cbout,
  USBD_TRANSFER_DIRECTION_IN | USBD_SHORT_TRANSFER_OK, N
ULL);
```

```
status = SendAwaitUrb(fdo, &urb);
```

کد در زمینه یک دستگیره برای فراخوانی یک DeviceioControl که از روش بافر برای پردازش داده استفاده می کند, اجرا می شود, چون فیلد SystemBuffer از IRP اشاره به مکان داده ای که باید تبادل شود دارد. متغییر cbout سایز بافر داده ای که ما باید پر کنیم را در خود دارد.

شما با یک فلگ نشان می دهید که در حال خواندن (USBD_TRANSFER_DIRECTION_IN) یا در حال نوشتن (USBD_TRANSFER_DIRECTION_OUT) اندپوینت هستید. شما به صورت دلخواه می توانید با فلگ دیگری (USBD_SHORT_TRANSFER_OK) نشان دهید که آیا مایل به دریافت اطلاعاتی کمتر از ماکزیمم سایز اندپوینت هستید یا خیر . دستگیره لوله آن چیزی است که شما در زمان IRP_MN_START_DEVICE در موقعیتی که قبلا شرح داده شد تسخیر می کنید.

۲-۳-۵-۲- مدیریت لوله های وقفه ای (Managing Interrupt Pipes)

از دید دستگاه باس, یک لوله وقفه تقریبا مشابه یک لوله تبادل توده است. تنها تفاوت مهم این دو این است که میزبان به اندپوینت وقفه ای با یک فرکانس ضمانت شده سرکشی می کند. وسیله با یک NAK پاسخ خواهد داد مگر اینکه اندپوینت وقفه ای داشته باشد برای گزارش یک رویداد وقفه ای , دستگاه در صورتی که تمام داده مورد نظرضمیمه شده باشد یک ACK را به میزبان بر می گرداند.

از دید درایور, اداره کردن یک لوله وقفه ای کاملا پیچیده تر از نوع توده ای آن است . در لوله نوع توده ای درایور هنگام نیاز به خواندن یا نوشتن داده از یا به لوله توده , فقط احتیاج به خلق یک URB مناسب و ارسال آن به درایور باس دارد. اما در مورد لوله وقفه برای انجام کارهای در نظر گرفته شده , اساسا درایور احتیاج به نگهداری یک درخواست خواندن در همه زمان ها دارد.

۲-۳-۵-۳- درخواست های کنترلی

اگر به خاطر بیاورید , تعداد ۱۱ نوع خواسته کنترلی استاندارد وجود دارد. ما هیچوقت در مورد درخواست های SET_ADDRESS صریحا بحث نمی کنیم زیرا درایور باس آن را در هنگام اتصال دستگاه انجام می دهد. قبل از اینکه ما کنترل را در درایور خود بدست بگیریم , درایور باس یک آدرس به دستگاه تخصیص می دهد و توصیف گر دستگاه را برای شناسایی درایور دستگاه می خواند. ما قبلا در مورد چگونگی ساخت URB ها یی که منجر به ارسال درخواست های کنترلی از جانب درایور باس برای گرفتن توصیف گرها یا اعمال یک پیکربندی یا یک

واسط شود , در بخش های شروع درخواست ها و پیکربندی توضیح داده ایم . در ادامه ناگفته های باقیمانده در مورد انواع ترنزکشن های کنترلی بحث می شود.

۲-۳-۵-۴- خصوصیات کنترل

اگر ما بخواهیم یک ویژگی دستگاه , واسط یا یک اندپوینت را ست یا پاک کنیم , ما یک URB خصیصه ارایه می کنیم.

کدی که در ادامه آمده است , یک خصیصه واسط به نام مشخصه فروشنده را , ست می کند:

```
URB urb;
UsbBuildFeatureRequest(&urb,
  URB_FUNCTION_SET_FEATURE_TO_INTERFACE,
  FEATURE_LED_DISPLAY, 1, NULL);
status = SendAwaitUrb(fdo, &urb);
```

آرگومان دومی به UsbBuildFeatureRequest می فهماند که ما قصد ست کردن یا پاک کردن خصیصه متعلق به دستگاه , یک واسط و یا یک اندپوینت یا یک سلول مخصوص فروشنده را داریم. این پارامتر ۸ مقدار ممکن را می تواند بگیرد که به فرم زیر هستند:

```
URB_FUNCTION_ [SET | CLEAR] _FEATURE_TO_
  [DEVICE | INTERFACE | ENDPOINT | OTHER]
```

آرگومان سوم به UsbBuildFeatureRequest خصیصه را می شناساند. در چهارمین آرگومان یک سلول مشخص نوع آنچه را که آدرس داده شده است مشخص می کند.

USB دو خصیصه استاندارد را که شما می توانید با یک URB خصیصه از آن ها استفاده کنید, تعریف کرده است: خصیصه remote wake-up و خصیصه endpoint stall (ایست نقطه پایانی) . اگرچه شما احتیاجی به ست و پاک کردن آنها ندارید, زیرادرایورباس آن را بصورت خودکارانجام می دهد.هنگامی که شما یک درخواست IRP_MN_WAIT_WAKE صادر می کنید , درایور باس از وجود این قابلیت در دستگاه مطمئن می شود و به صورت خودکار این قابلیت را فعال می کند. درایورباس یک درخواست پاک کردن خصیصه را در هنگامی که شما یک URB , RESET_PIPE صادر می کنید , صادر می کند.

۲-۳-۵-۵- تعیین کردن وضعیت

هنگامی که شما مایل به بدست آوردن موقعیت دستگاه, واسط , یا یک اندپوینت هستید , یک URB را برای دریافت وضعیت تنظیم می کنید . برای مثال:

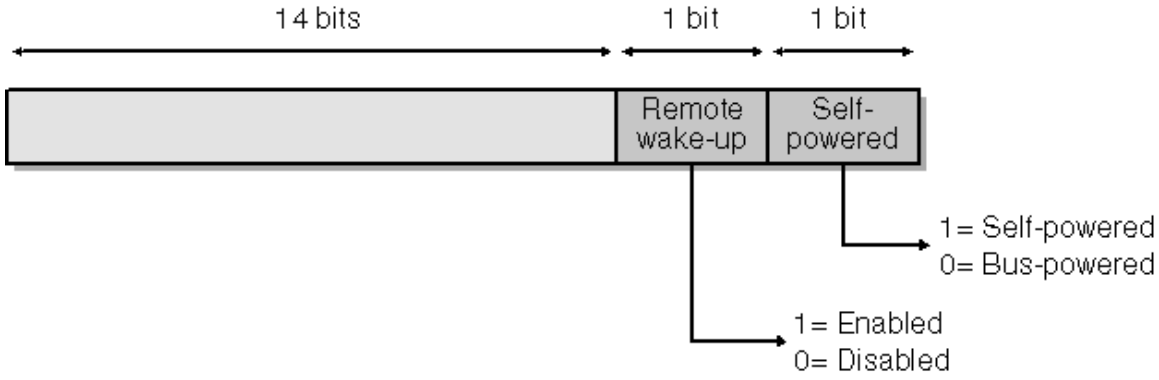
```
URB urb;
USHORT epstatus;
UsbBuildGetStatusRequest(&urb,
  URB_FUNCTION_GET_STATUS_FROM_ENDPOINT,
  <index>, &epstatus, NULL, NULL);
SendAwaitUrb(fdo, &urb);
```

شما می توانید چهار گونه مختلف URB را در یک درخواست دریافت وضعیت به کار برید, که به شما اجازه می دهد ماسک وضعیت دستگاه را به صورت کلی , برای یک واسط و یک اندپوینت مشخص بدست اورید.

Operation Code	Retrieve Status From...
URB_FUNCTION_GET_STATUS_FROM_DEVICE	Device as a whole
URB_FUNCTION_GET_STATUS_FROM_INTERFACE	Specified interface
URB_FUNCTION_GET_STATUS_FROM_ENDPOINT	Specified endpoint
URB_FUNCTION_GET_STATUS_FROM_OTHER	Vendor-specific object

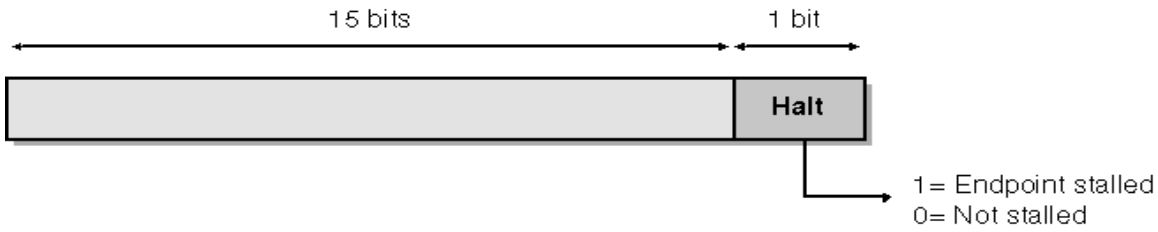
جدول ۲-۲- انواع Urb دردرخواست دریافت وضعیت

ماسک وضعیت یک دستگاه مشخص می کند که آیا دستگاه خود توان Self-Powered است و آیا خاصیت remote wake-up آن فعال است یا خیر.



شکل ۲-۵ . بیت های وضعیت دستگاه

ماسک برای یک اندپوینت , بیانگر این است که آیا اندپوینت درحال حاضردروضعیت ایست است.

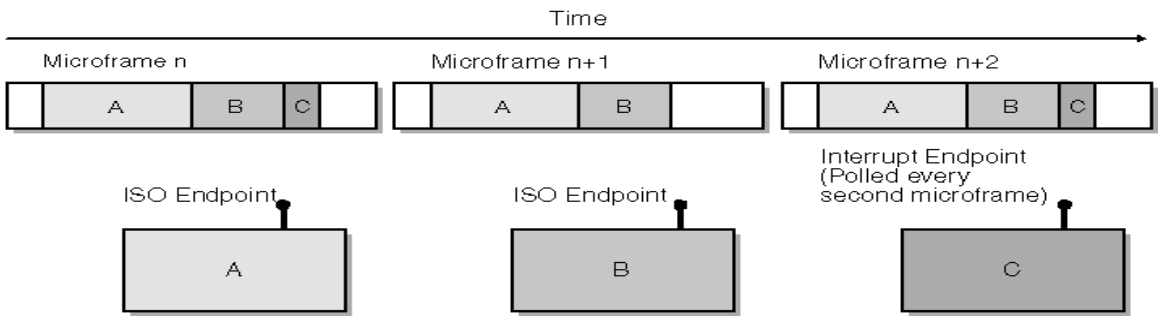


شکل ۲-۶. بیت ها ی وضیت اندپوینت

USB همچنین بیت های وضعیت سطح واسط را نیز که مربوط به مدیریت توان در خصوصیات مدیریت توان واسط هست را معین کرده است.

۲-۳-۵-۶- مدیریت لوله های همزمان (Managing Isochronous Pipes)

حدف یک لوله همزمان ایجاد یک مجرای ارتباطی برای تبادل داده های حساس زمانی با یک نظم و ترتیب ضمانت شده است. درایور باس بالغ بر ۸۰ درصد از پهنای باند باس را برای تبادلات همزمان و وقفه در نظر می گیرد. این بدین معنی است که هر میکروفریم ۱۲۵ میلی ثانیه ای شامل زمان رزرو شده کافی برای تبادل ماکزیمم سایز تبادل از یا به اندپوینت های همزمان یا وقفه فعال است. شکل زیر این موضوع را برای سه دستگاه مختلف نشان می دهد. دستگاه های A و B هر کدام یک اندپوینت همزمان دارند که مقدار زیادی از زمان هر میکروفریم را به خود اختصاص داده اند. دستگاه C دارای یک اندپوینت وقفه است که متناوبا در هر دو میکروفریم سر کشی می شود. آن دارای یک قسمت کوچک رزرو شده در هر میکروفریم یک ثانیه ای است. میکرو فریم هایی که شامل سرکشی به اندپوینت وقفه ای دستگاه نمی شوند این زمان را برای تبادل توده ای یا هدف های دیگر استفاده می کنند.



شکل ۲-۷- زمان بندی میکروفریم ها

ساختار فایل های INF

این فایل ها معمولا در شاخه inf در شاخه اصلی ویندوز یافت می شوند . سیستم عامل برای یافتن این فیل ها در ابتدا شاخه ذکر شده را جستجو می کند و در صورتی که آن را پیدا نکرد , آدرسش را از کاربر می پرسد. یک فایل inf ممکن است که بسیار بزرگ و پیچیده باشد , اما ساختار تمام آن ها شبیه هم هستند و از یک الگوی خاصی پیروی می کنند. به کمک این الگو با تغییرات بسیار کم می توان به نتایج مورد نظر رسید. سازندگان تراشه های USB معمولا یک نمونه از آن ها را به خریداران ارایه می کنند. در زیر یک فایل inf نشان داده شده است.

```
[Version]
Signature="$CHICAGO$"
DriverVer=01/29/2004,2.0.1600.0
Provider=%Provider%
Class=EllisysProtocolAnalyzers
ClassGUID={D8854594-A4EF-480e-B8D8-CBDDADB4F3B4}
[ClassInstall]
AddReg=ClassAddReg
[ClassInstall32]
AddReg=ClassAddReg
[ClassAddReg]
HKR,,,,"%ClassName%"
HKR,,Icon,,-20
[Manufacturer]
%Manufacturer%=Models
[DestinationDirs]
DefaultDestDir=10,System32\Drivers
[SourceDisksNames]
1=%SourceDisk%,,,.
[SourceDisksFiles]
ellex200.sys=1
[Models]
%DeviceDesc%=Install,USB\VID_0ABA&PID_8002
[Install]
CopyFiles=Install.CopyFiles
AddReg=Install.AddReg
[Install.CopyFiles]
ellex200.sys,,,2
[Install.AddReg]
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,ellex200.sys
[Install.NT]
CopyFiles=Install.CopyFiles
[Install.NT.Services]
AddService=ellex200,2,Install.NT.AddService
[Install.NT.AddService]
```

```
DisplayName=%SvcDesc%
ServiceType=1
StartType=3
ErrorControl=1
ServiceBinary=%10%\system32\drivers\ellex200.sys
[Strings]
ClassName="Ellisys protocol analyzers"
Provider="Ellisys "
Manufacturer="Ellisys"
SourceDisk="USB Explorer 200 Installation Disk"
DeviceDesc="USB Explorer 200"
SvcDesc="USB Explorer 200 Driver (ellex200.sys)"
```

همان طور که در این فایل مشاهده می شود , یک فایل inf. از چندین بخش تشکیل شده که عنوان هر بخش بین دو علامت],[نوشته شده است . به عنوان مثال بخش زیر فایل DLL دستگاه USB را معرفی می کند.

```
[PCBase.Files.Dll]
        dllfile.dll
```

در حقیقت تمام اطلاعاتی که در مورد درایور هر دستگاه و سخت افزار درون یک فایل inf گنجانده شده است . به عنوان مثال در پنجره خواص (Properties) هر سخت افزار یک قسمت Driver وجود دارد و این قسمت اطلاعات خود را از فایل inf مربوط به سخت افزار می گیرد. در یک فایل inf نکات زیر را باید در نظر گرفت.

- تمام اطلاعات فایل های inf در چندین قسمت مرتب شده اند و نام هر بخش در یک کروشه نوشته شده است . نام اکثر بخش ها استاندارد هستندو ویندوز آن ها را می شناسد.
- در این فایل اطلاعات بعد از سمی کلن ";" در نظر گرفته نمی شوند, زیرا از این علامت برای نوشتن توضیحات اضافی استفاده می شود.
- اگر در انتهای یک خط علامت "/" اسلش وجود داشت , نشان می دهد که این خط ادامه دارد.
- متنهایی که بین دو علامت ٪ قرار می گیرند , یک اشاره گر به رشته هستند . به عنوان مثال در صورتی که از عبارت زیر استفاده شود.

Provider="PC Base"

بعد از این عبارت در هر مکانی که عبارت %Provider% گنجانده شده باشد , ویندوز "PC Base" را جایگزین این عبارت قرار می دهد.

- بعضی از مقدار دهی ها در این فایل مقدار ورودی را مشخص می کنند. به عنوان مثال عبارت زیر کلاس دستگاه را USB تعریف می کند.

Class=USB

- بعضی از مقدار دهی ها نیز برای ذخیره سازی در رجیستری ویندوز استفاده می شوند.

به عنوان مثال :

HKR , , Installer , , dllfile.dll

همان طور که گفته شد , این فایل ها از بخش های مختلف و استانداردی تشکیل شده اند. در زیر تعدادی از قسمت های استاندارد فایل های inf توضیح داده شده اند.

[Version]

از این قسمت معمولا به عنوان سر آغاز فایل استفاده می شود و یک بخش الزامی برای همه فایل های inf است.

قسمت [Version] از خطوط زیر تشکیل شده است.

```
[Version]
Signature="$CHICAGO$"
DriverVer=01/29/2004,2.0.1600.0
Provider=%Provider%
Class=EllisysProtocolAnalyzers
ClassGUID={D8854594-A4EF-480e-B8D8-CBDDADB4F3B4}
```

هر کدام از عبارات فوق که قبل از علامت "=" قرار گرفته اند , به عنوان یک کلید شناخته می شوند.

کلید Signature برای تعیین نسخه های ویندوزی استفاده می شود که دستگاه می تواند با آن ها کار کند.

که مقدار $CHICAGO$ نشان دهنده تمام نسخه های ویندوز است.

کلید کلاس , نام دستگاهی که فایل برای آن نوشته شده است را مشخص می کند. کلید GUID, classGUID مربوط به دستگاه را که یک مشخصه ۱۲۸ بیتی است را مشخص می کند. کلید Provider نام سازندگان دستگاه را مشخص می کند. کلید Layout نام دیسک های منبع و فایل هایی که برای نصب لازم هستند را مشخص می کند.

[ClassInstall]

این بخش یک کلاس جدید را در بخش کلاس های رجیستری ویندوز باز می کند . فقط در صورتی که دستگاه تا کنون در سیستم نصب نشده باشد, ویندوز از این بخش استفاده خواهد کرد . به عنوان مثال:

```
[ClassInstall]
Addreg=Class.Addreg
```

کلید Addreg این کلاس جدید را در رجیستری نصب می کند.

[Manufacture]

در این بخش نام سازندگان دستگاه مورد نظر مشخص می شود.

```
[Manufacture]
%MfgName%=CompanyName
```

[DestinationDirs]

این قسمت نام شاخه هایی را نشان می دهد که از آیتم های CopyFiles , RenFiles و DelFiles استفاده خواهند کرد. به عنوان مثال:

```
[DestinationDirs]
DefaultDestDir=10,System32\Drivers
```

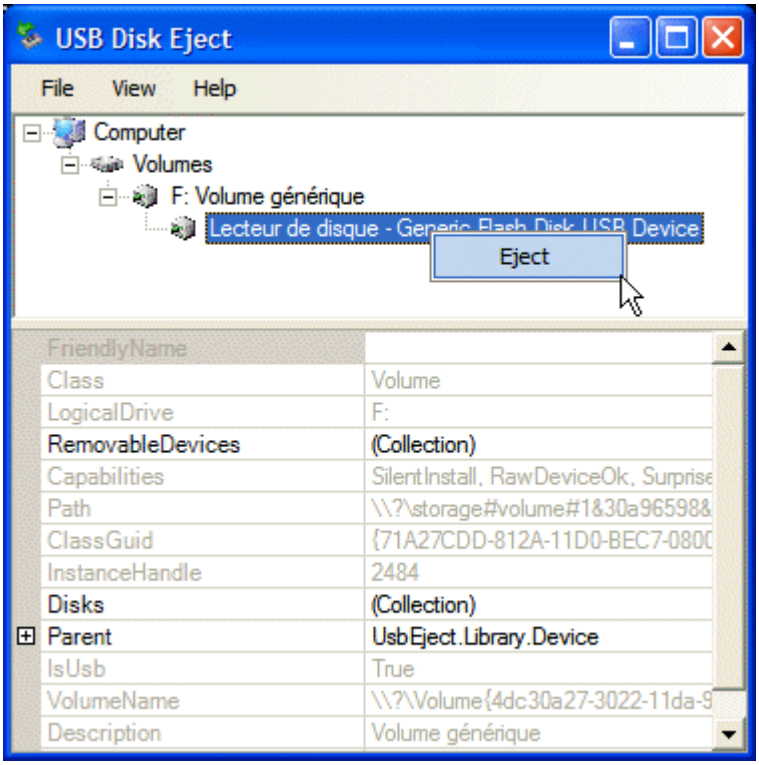
[String]

تمام رشته هایی که در بخش های دیگر آمده اند, به این قسمت ارجاع داده شده اند . به عنوان مثال:

```
[Strings]
ClassName="Ellisys protocol analyzers"
Provider="Ellisys "
Manufacturer="Ellisys"
SourceDisk="USB Explorer 200 Installation Disk"
DeviceDesc="USB Explorer 200"
SvcDesc="USB Explorer 200 Driver (ellex200.sys)"
```

ساخت یک فایل inf

برای ساخت فایل های inf. از نرم افزار Notepad که در تمام کامپیوتر ها موجود است, می توان استفاده کرد . زیرا این فایل ها از نوع متنی معمولی هستند. البته نرم افزار های پیشرفته تری برای این کار ساخته شده اند , که از جمله آن هامی توان از InfCatReady , ChkInf , Geninf , InfEdit نام برد.



مقدمه

اگر چه قطع کردن یک دستگاه USB از طریق ویندوز کاری بسیار آسان است ولی این کار در برنامه نویسی واقعا پیچیده است. توجهات پایه ای بسیاری در مرز برنامه نویسی درایور هسته وجود دارد. یکی از آن ها فهمیدن و پیاده سازی وظیفه های مورد نیاز است. در این پروژه مجبور به سوییچ کردن میان کد های کنترلی درایور هسته , Windows Setup و مدیریت پیکر بندی API ها و WMI وغیره هستیم.

به عنوان یک مزیت در این پروژه , نحوه خواندن شماره سریال سخت افزار دیسک های USB نیز توضیح داده شده است.

پیش زمینه

در ابتدا باید در مورد API های مختلف ویندوز , صرف نظراز Win32 بحث کنیم.

۱. The Windows DDK(Driver Development Kit) - یک کیت توسعه درایور که امکاناتی از قبیل : محیط ساخت , ابزارها و مستنداتی درباره تهیه یک درایور برای ویندوز دارد.

۲. The Setup Api – یک API که بخشی از DDK می باشد ولی در واقع از API های عمومی روتین های Setup هست که در برنامه های نصبی به طور شاخص مورد استفاده قرار می گیرد, شبیه تابع CM_Request_Device_Eject که قلب این پروژه می باشد.

۳. The DeviceIoControl function. - تابع مشکل گشا , مد کاربر در بطن مد هسته است.

۴. WMI (Windows Management Instrumentation). – دراین پروژه به طور کامل از WMI استفده نشده است زیرا WMI قطع وسیله را هندل نمی کند. در این پروژه از یک روند هیبریدی برای استفاده از WMI در مدیریت دستگاه استفاده شده است.

۵. Windows Messages. – پیغام ویندوز WM_DEVICECHANGE در این کاربرد واسط گرافیکی برای تازه سازی درخت در هنگام رویدادی در درخت مدیریت وسیله , استفاده شده است.

حال به تعریف چند ضابطه می پردازیم:

۱. Physical Disks - قسمت حقیقی سخت افزار کاربر نهایی

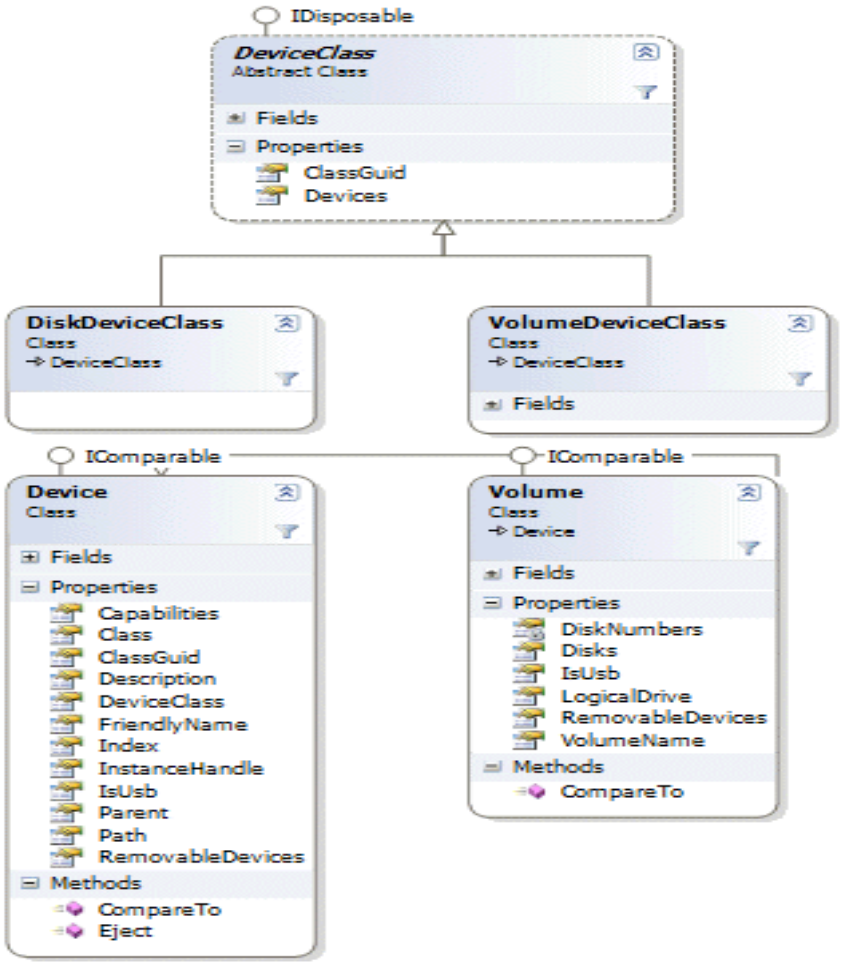
۲. Volumes - درعمل دو نوع ولوم وجود دارد: ولوم هایی که توسط ویندوز شناخته می شوند و ولوم هایی که ما آن ها را می شنا سیم که شامل درایو های A: تا Z: می باشند که با نام دیسک های منطقی نیز نامیده می شوند.

۳. DDK – Devices ویندوز دو نوع دیسک های فیزیکی و درایوها را به عنوان DEVICE می شناسد. اساس یک Device است که Volume از آن مشتق شده است .

Using the code

این پروژه یک فرم پروژه Visual Stdio 2005(.NET 2.0) است. یک فولدر به نام Library حاوی قلب پروژه و یک Object Oriented API برای هندل کردن دیسک های USB موجود است.

کلاس ها به صورت دیاگرام زیر توصیف شده اند:



همان طور که می بینید ۵ کلاس اصلی وجود دارد که هر کلاس در فایل cs. مربوط به خود تعریف شده است:

۱. DeviceClass – یک کلاس تجریدی که نشان دهنده یک کلاس دستگاه فیزیکی است. آن در بر گیرنده یک لیست از دستگاه ها در این کلاس است.

۲. DiskDeviceClass – نشان دهنده تمام دیسک ها ی سیستم است.

۳. VolumeDeviceClass – نشان دهنده تمام ولوم های سیستم است.

۴. Device – نشان دهنده یک دستگاه عام از هر نوع (disk,volume,etc...) می باشد. توجه شود که کلاس دیسک وجود ندارد زیرا در این پروژه یک دیسک در مقایسه با یک دستگاه دارای ویژگی های خاصی نمی باشد.

۵. Volume – نشان دهنده یک ولوم ویندوز است. یک ولوم دارای یک خصیصه LogicalDrive است که در صورت درایو حرفی فرض شدن , مقدار دهی می شود.

نویسنده : حسین

پایمرد

کد زیرچگونگی ایجکت کردن تمام ولوم های USB را نشان می دهد:

```
VolumeDeviceClass volumeDeviceClass = new
VolumeDeviceClass();
foreach (Volume device in volumeDeviceClass.Devices)
{
    // is this volume on USB disks?
    if (!device.IsUsb)
        continue;

    // is this volume a logical disk?
    if ((device.LogicalDrive == null) ||
(device.LogicalDrive.Length == 0))
        continue;

    device.Eject(true); // allow Windows to display any
relevant UI
}
```

Points of Interest

CM_Request_Device_Eject function

این یک تابع SetupApi برای ایجکت کردن یک دستگاه می باشد. آن یک هندل نمنه دستگاه (devInst) را به عنوان ورودی می گیرد. تابع در صورت مقداردهی دومین آرگومان (pVetoType) رفتار متفاوتی خواهد داشت. اگر آن مقداردهی شود, پوسته ویندوزچیزی به کاربران پایانی نمایش نخواهد داد و عملیات ایجکت در صورت موفقیت یا شکست چیزی نشان نمی دهد. ولی در غیر این صورت پوسته ویندوز توسط جعبه های دیالوگ یا پیغام یا بالون های ویندوز به کاربر نهایی گزارش می دهد.

مشکل اساسی در اینجا در مورد درایوهای معین حرفی در صورت ایجکت شدن است.

وسیله برای ایجکت شدن باید یک Disk Device باشد , نه یک Volume Device . چنانچه الگوریتم کلی به صورت زیر می باشد:

۱. تعیین تمام دیسک های منطقی موجود با استفاده از ویژگی `Environment.GetLogicalDrives` از .NET.

۲. تعیین نام حقیقی ولوم برای هر درایو منطقی با استفاده از ویژگی `GetVolumeNameForVolumeMountPoint` از Win32

۳. تعیین اینکه آیا برای هر Volume Device دیسک منطقی نظیری وجود دارد.

۴. تعیین Disk Device های فیزیکی بوجود آمده از Volume Device برای هر Volume Device با استفاده از کد کنترلی IO , IOCTL_VOLUME_GET_VOLUME_DISK_EXTENTS از DDK

۵. به علت اینکه مدیریت پیکر بندی Plug-And-Play (PnP) یک مرتبه بندی از دستگاه می سازد , یک Disk Device فیزیکی , بطور کلی یک وسیله قابل ایجکت نیست . بنابراین برای هر Disk Device فیزیکی , آنچه را که یک وسیله برای ایجت شدن در طی فرآیند ایجکت نیاز دارد تعیین می کنیم. دستگاه برای ایجکت شدن , اولین دستگاه در مرتبه بندی با قابلیت CM_DEVCAP_REMOVABLE است.

GetVolumeNameForVolumeMountPoint function

برای نظیر کردن ولوم های ویندوز با دیسک های منطقی (مرحله ۳ الگوریتم) , یک ترفند وجود دارد. تمام ولوم های ویندوز می توانند به یک نحو معینی به صورت یکتا ادرس دهی شوند. "\\?\Volume{GUID}\" که GUID مشخص کننده ولوم می باشد.

درهنگام سرشماری دستگاه های ولوم (با استفاده ازراهنمای کلاس دستگاه ولوم GUID_DEVINTERFACE_VOLUME) مسیر دستگاه ولوم می تواند به صورت مستقیم در فراخوانی به GetVolumeNameForVolumeMountPoint استفاده شود.

Enumerating all devices for a given class

کد زیر نشان می دهد که .NET چگونه سر شماری یک نوع معین از دستگاه های فیزیکی را انجام می دهد:

```
int index = 0;
while (true)
{
    // enumerate device interface for a given index
    SP_DEVICE_INTERFACE_DATA interfaceData = new
SP_DEVICE_INTERFACE_DATA();
    if (!SetupDiEnumDeviceInterfaces(
        _deviceInfoSet, null, ref _classGuid, index,
interfaceData))
    {
        int error = Marshal.GetLastWin32Error();
        // this is not really an error...
```

```
        if (error != Native.ERROR_NO_MORE_ITEMS)
            throw new Win32Exception(error);
        break;
    }

    SP_DEVINFO_DATA devData = new SP_DEVINFO_DATA();
    int size = 0;
    // get detail for all the device interface
    if (!SetupDiGetDeviceInterfaceDetail(
        _deviceInfoSet, interfaceData, IntPtr.Zero, 0, ref
size, devData))
    {
        int error = Marshal.GetLastWin32Error();
        if (error != Native.ERROR_INSUFFICIENT_BUFFER)
        throw new Win32Exception(error);
    }

    // allocate unmanaged Win32 buffer
    IntPtr buffer = Marshal.AllocHGlobal(size);
    SP_DEVICE_INTERFACE_DETAIL_DATA detailData = new
SP_DEVICE_INTERFACE_DETAIL_DATA();
    detailData.cbSize = Marshal.SizeOf(

typeof(Native.SP_DEVICE_INTERFACE_DETAIL_DATA));

    // copy managed struct buffer into unmanager win32
buffer
    Marshal.StructureToPtr(detailData, buffer, false);

    if (!SetupDiGetDeviceInterfaceDetail(
        _deviceInfoSet, interfaceData, buffer, size, ref
size, devData))
    {
        Marshal.FreeHGlobal(buffer); // don't forget to
free memory
        throw new
Win32Exception(Marshal.GetLastWin32Error());
    }

    // a bit of voodoo magic. This code is not 64 bits
portable :-)
    IntPtr pDevicePath = (IntPtr)((int)buffer +
Marshal.SizeOf(typeof(int)));
```

```
    string devicePath =
Marshal.PtrToStringAuto(pDevicePath);
    Marshal.FreeHGlobal(buffer);
    index++;
}
```

Refreshing the UI when a disk is inserted or removed

برای انجام این کار روش های بسیاری وجود دارد . ما در این پروژه این کار را با تسخیر WM_DEVICECHANGE از پیغام های ویندوز انجام داده ایم. شما فقط مجبور به override کردن پروسه پیش فرض ویندوز برای هر فرم ویندوز در برنامه خود هستید:

```
protected override void WndProc(ref Message m)
{
    if (m.Msg == Native.WM_DEVICECHANGE)
    {
        if (!_loading)
        {
            LoadItems(); // do the refresh work here
        }
    }
    base.WndProc(ref m);
}
```

A word on Serial Numbers

دراین قسمت کمی در مورد شماره سریال های دیسک سخت بحث می کنیم. هنگام صحبت در مورد دیسک ها و ولوم های ویندوز در واقع دو شماره سریال وجود دارد:

۱. A volume software serial number

- در طی فرآیند فرمت در نظر گرفته می شود. این یک مقدار ۳۲ بیتی است که به راحتی توسط GetVolumeInformation از تابع های با قاعده Win32 قابل خواندن است.

۲. A disk vendor's hardware serial number – این شماره سریال در کارخانه توسط فروشنده اعمال می شود و قابل تغییر نیست. متاسفانه این شماره سریال برای دستگاه های USB اختیاری می باشد و اکثر آن ها این شماره را ندارند.

حال سوال این است که چگونه می توان شماره سریال سخت افزار را خواند؟

WMI , آسان ترین راه , اگر چه شبیه یک جادو است.

```
1.        // browse all USB WMI physical disks
2.        foreach(ManagementObject drive in new
  ManagementObjectSearcher(
3.            "select * from Win32_DiskDrive where
  InterfaceType='USB'").Get())
4.        {
5.            // associate physical disks with partitions
6.            foreach(ManagementObject partition in new
  ManagementObjectSearcher(
7.                "ASSOCIATORS OF
  {Win32_DiskDrive.DeviceID='" + drive["DeviceID"]
8.                  + "'} WHERE AssocClass =
  Win32_DiskDriveToDiskPartition").Get())
9.            {
10.               Console.WriteLine("Partition=" +
  partition["Name"]);
11.
12.               // associate partitions with logical disks
  (drive letter volumes)
13.               foreach(ManagementObject disk in new
  ManagementObjectSearcher(
14.                   "ASSOCIATORS OF
  {Win32_DiskPartition.DeviceID='"
15.                     + partition["DeviceID"]
16.                     + "'} WHERE AssocClass =
  Win32_LogicalDiskToPartition").Get())
17.               {
18.                   Console.WriteLine("Disk=" +
  disk["Name"]);
19.               }
20.           }
21.
22.           // this may display nothing if the physical
  disk does not have a hardware serial number
23.           Console.WriteLine("Serial="
24.            + new
  ManagementObject("Win32_PhysicalMedia.Tag='"
25.            + drive["DeviceID"] + "'")["SerialNumber"]);
26.       }
```