ارائه یک چهارچوب کاربردی جهت نوشتن Device Driver در سیستم عامل ویندوز

ناصر جهانگیری[1]
دانشکده فنی تهران
jahangiri@ece.ut.ac.ir

سید کاظم جهانبخش[2]
دانشگاه صنعتی شریف
jahanbakhsh@mehr.sharif.edu

مسعود یعقوبی[3]
دانشگاه صنعتی شریف
yaghubi@mehr.sharif.edu

بیژن وثوقی وحدت[4]
دانشگاه صنعتی شریف
vahdat@sharif.edu

چکیده: درایورهای ویندوز از *Multitasking* و برنامه های ۳۲ بیتی در *protected mode* و ۱۶ بیتی در *real mode* پشتیبانی می کنند. نوشتن اینگونه درایورها نیازمند شناسایی محیط سیستم عامل ویندوز و نقشی که این درایورها در ویندوز ایفا می کنند، می باشد. درایورهای ویندوز با بهره مندی از تمام قابلیت های سیستم عامل، امکان دسترسی مستقیم به سخت افزار و بهره مندی از بالاترین اولویت اجرایی را فراهم می کنند. از این رو شناسایی ساختار درایورها و نحوه ارتباط آنها با اجزا سیستم عامل ویندوز، و در پایان ارائه یک چارچوب برای آن می تواند یک دید کلی ایجاد کند.[1]

واژه های کلیدی: ***Device Driver***، ***VxD***، ***Virtual Machine***، ***Virtual Machine Manager***.

۱- مقدمه

بسیاری از وظایفی که برنامه نویسان سیستم با آن در تقابل هستند نوشتن یک *device driver* برای یک سخت افزار مشخص می باشد.

پروسسورهای قدیمی *80x86* همگی ۱۶ بیتی بوده و تنها در *real mode* به اجرای دستورات (*instruction*ها) می پرداختند. مبتنی بر این پروسسورها سیستم عامل قدیمی *DOS* شکل گرفت. *DOS* یک محیط ۱۶ بیتی با سرعت و حافظه پایین می باشد. این معایب به همراه عدم توانایی آن در *Multitasking* موجب کاهش شدید کارآیی این محیط شده است. با ظهور پروسسورهای ۳۲ بیتی جدید که قابلیت کارکردن در *protected mode* را دارا هستند، سیستم عامل ویندوز شکل گرفت. محیط ۳۲ بیتی ویندوز، از سرعت بالایی برخوردار است. علاوه بر این، قابلیت *Multitasking*، اختصاص حافظه بالا و پشتیبانی همزمان از برنامه های ۳۲ بیتی *protected mode* و برنامه های ۱۶ بیتی *real mode* از ویژگی های بارز این محیط محسوب می شوند.

درایور نویسی در محیط *DOS* ساده می باشد ولی چرا درایور نویسی در محیط ویندوز علی رغم پیچیدگی محیط آن، مورد توجه است؟

نوشتن *device driver* نیاز به دانستن چگونگی دسترسی به *I/O*، حافظه و بکار بردن اینتراپت هاست.

---

۱ دانشجوی کارشناسی ارشد مخابرات دانشکده فنی دانشگاه تهران

۲ دانشجوی کارشناسی ارشد الکترونیک دانشکده برق دانشگاه صنعتی شریف

۳ دانشجوی کارشناسی ارشد قدرت دانشکده برق دانشگاه صنعتی شریف

۴ استادیار دانشکده برق دانشگاه صنعتی شریف

در محیط *DOS* در هر لحظه فقط یک برنامه اجرا می شود که به تنهایی تمامی منابع سیستم را در اختیار دارد. به این ترتیب دسترسی به *I/O*، حافظه و ... به سادگی قابل انجام است.

برخلاف محیط *single task* داس، محیط ویندوز از اجرای همزمان چند *task* با هم پشتیبانی می کند. دسترسی همزمان چند برنامه به یک منبع مشخص (مانند یک پورت مشخص یا حافظه) قاعدتا مشکل ساز خواهد بود، مگر آنکه برنامه ها مجبور به رعایت یک سری قواعد تحمیل شده از سوی بخشی از سیستم عامل در بالاترین سطح باشند. این قواعد را اجزاء واقع در هسته سیستم عامل وضع می کنند و بر اجرای آنها به طور پیوسته نظارت دارند.

به همین منظور آشنایی با اجزائی از سیستم عامل که وظیفه پشتیبانی و نظارت بر منابع سیستم را عهده دار هستند، الزامی است.[1]

۲- تعاریف اولیه

۲-۱- Real Mode:

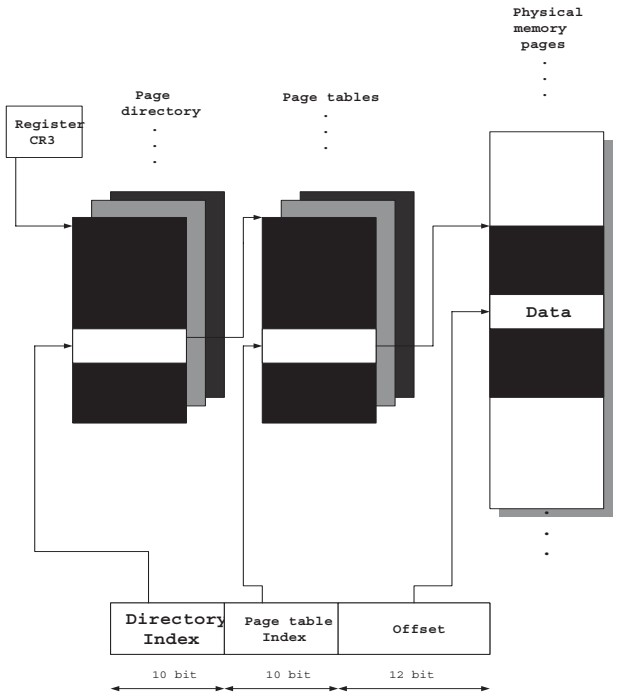
پروسسور *80x86* در این مد، محیطی را برای برنامه فراهم می کند که در هر لحظه تنها یک *task* (برنامه) به حافظه و *I/O* سیستم دسترسی آزاد د اشته باشد. روش آدرس دهی در این مد، استفاده از سگمنت و آفست برای تولید آدرس فیزیکی ۲۰ بیتی می باشد. حداکثر حافظه ممکن تنها یک مگا بایت می باشد.[1]

۲-۲- Protected Mode:

پروسسورهای *80386* و بالاتر اینتل در این مد از *Multitasking*، حفاظت از اطلاعات (*Data Security*) و حافظه مجازی (*Virtual Memory*) بهره می گیرند.

در *protected mode* روش آدرس دهی کاملا متفاوت است. رجیستر سگمنت (*Register Segment*) به صورت یک انتخاب کننده ۱۶ بیتی یک *Segment Descriptor* را از جدول *Descriptor* انتخاب می کند. *Segment Descriptor* شامل یک آدرس پایه (*Base Address*) همراه با یک سری اطلاعات کنترلی است؛ با جمع کردن آدرس پایه استخراج شده از *Descriptor* با آفست استخراج شده از *instruction operand* آدرس مجازی ۳۲ بیتی حاصل می شود. به این ترتیب یک برنامه *protected mode* می تواند تا ۴ گیگا بایت (2^{32} بایت) حافظه را بکار ببرد. از آنجا که کامپیوترهای امروزی عملا به ۴ گیگا بایت حافظه فیزیکی دسترسی ندارند، سیستم عامل برای حل این مشکل از یک فایل *disk swap* به صورت موقت استفاده می کند یعنی از نگاشت حافظه به داخل دیسک سخت بهره می گیرد.

برای رسیدن به چنین محدوده ای از حافظه مجازی، سیستم عامل باید قادر باشد تا هر *page* مربوط به حافظه مجازی را در حافظه فیزیکی بازیابی کند. آدرس مجازی توسط *page table*ها به یک آدرس فیزیکی تبدیل می شود. پروسس تبدیل آدرس مجازی به فیزیکی در شکل زیر آورده شده است.[1]



شکل ۱: تبدیل آدرس مجازی به فیزیکی

۲-۳- Virtual 8086 Mode (V86 Mode):

یک بخش از *protected mode* است که به برنامه های *DOS* و *real mode* اختصاص دارد. روش آدرس دهی در این مد مشابه *real mode* می باشد ولی آدرس ۲۰ بیتی حاصل، یک آدرس مجازی می باشد. این واقعیت امکان نگاشته شدن فضای

آدرسی یک مگا بایتی برنامه های *real mode* را به حافظه قابل دسترسی توسط برنامه های *protected mode* فراهم می کند.

۲-۴- سطوح دسترسی:

در پروسسورهای جدید هر بار که آدرسی تولید می شود یک سری بررسی های حفاظتی (*protection checks*) صورت می گیرد. بررسی های حفاظتی بر پایه چهار سطح دسترسی می باشند؛ که در آن *level0* (*ring 0*) بالاترین سطح دسترسی و *level3* (*ring 3*) کمترین سطح دسترسی را شامل است.

به صورت عمومی یک برنامه با سطح دسترسی پایین اجازه دسترسی به سگمنت ها با سطوح دسترسی بالاتر از خود را ندارد. این قابلیت روی پروسسورهای *Intel*، به سیستم عامل اجازه می دهد تا بر روی ساختارهای *code* و *data*های قابل استفاده توسط هر برنامه نظارت داشته باشد. در صورتیکه یک *protection violation* به هر دلیل رخ دهد، منجر به یک *exception* شده که باید توسط سیستم عامل سرویس داده شود.[1]

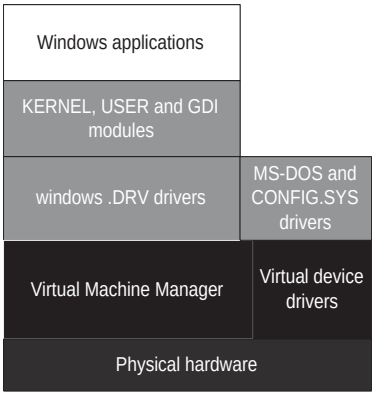
۲-۵- Ring 0:

بالاترین سطح اولویت در پروسسور می باشد، هسته سیستم عامل (درایورها) در این سطح اجرا می شود.

۳- آشنایی کلی با ویندوز ۹۸:

ویندوز ۹۸ یک سیستم عامل *protected mode*، ۳۲ بیتی می باشد که با فراهم کردن حافظه مجازی، قابلیت آدرس دهی تا ۴ گیگا بایت را دارا بوده و از *Multitasking* نیز پشتیبانی می کند. درایور نویسی در ویندوز شامل برنامه نویسی ماژول های ۳۲ بیتی در *ring 0* است که تحت عنوان *VxD* (*Virtual Device Driver*) شناخته می شود. هسته (*Kernel*) سیستم عامل ویندوز ۹۸ مجموعه ای از *VxD* هاست که تحت نظارت *VMM* (*Virtual Machine Manager*)، که خود نیز یک *VxD* می باشد، کار می کند.

مدل عمومی برای بکار بردن و راه اندازی *device*های سخت افزاری در سیستم عامل ویندوز استفاده از *VxD*ها به عنوان *device driver* می باشد. *VxD*ها از طریق اینتراپت ها و دسترسی به *I/O*، *device* را نصب و راه اندازی می کند. برنامه ها نیز از طریق فراخوانی سرویس ها و توابع تولید شده توسط *VxD*ها به *device*های سخت افزاری دسترسی می یابند.



شکل۲: جایگاه *VxD*ها در ویندوز ۹۸

در ویندوز ۹۸، برنامه ها از طریق ماژول های *User*، *Kernel* و *GDI* به درایورهای ویندوز در *ring 3* دسترسی دارند. از سوی دیگر درایورهای ویندوز به همراه درایورهای *DOS* بر روی *VxD*ها و *VMM* بنا می شوند که اینها حد واسط سخت افزار و *ring 3* می باشد.[2],[1]

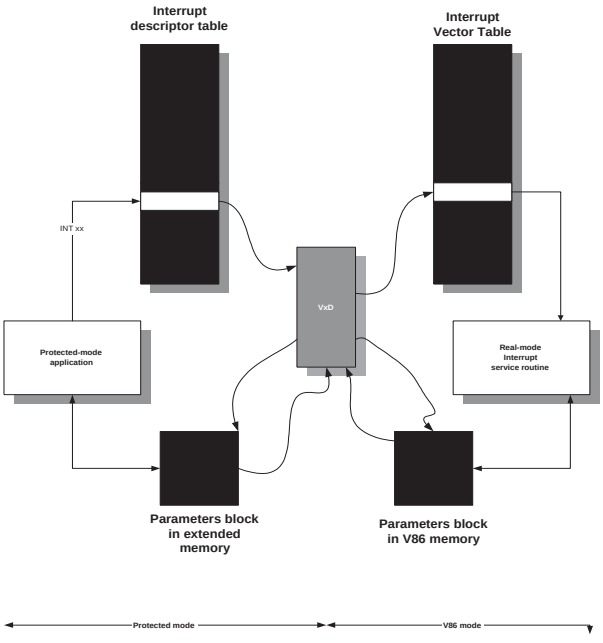
۳-۱- ماشین مجازی (Virtual Machine)

برنامه های ویندوز در یک محیط *Multitasking* اشتراکی کار می کنند، جایی که آنها بطور متناوب کنترل را به سیستم عامل برمی گردانند تا اجازه اجرا به سایر برنامه ها را بدهند. لذا برای اینکه به برنامه های مختلف اجازه به اشتراک گذاشتن پروسسور و سایر منابع که تنها یک وجود خارجی دارند، داده شود، ویندوز ۹۸ ماشین های مجازی (*VM*) را بکار می برد. هر *VM* فضای حافظه، فضای ورودی-خروجی و *IVT* (*Interrupt Vector Table*) مخصوص به خود را دارد. همچنین *BIOS-ROM*، *MS-DOS device driver*های لازم و برنامه های *TSR* (*Terminate & Stay Resident*) به فضای آدرسی *VM* نگاشته (*map*) می شوند. *VM*، یک کلک نرم افزاری است، که برنامه ها در قالب آن اجرا می شوند و آنچنان با برنامه رفتار می کند گویی که برنامه تا یک ماشین فیزیکی تقابل دارد. مجازی

سازی سخت افزار و نرم افزار در راستای تحقق *VM* ابزار کلیدی محسوب می شود.[1]

۳-۲- سخت افزار مجازی (Virtual Hardware)

یک برنامه *ring 3* در *protected mode* بطور مستقیم نمی تواند به سخت افزار دسترسی داشته باشد. در ویندوز یک راه ساده برای دسترسی به سخت افزار، استفاده از امکانات *real mode* در *protected mode* می باشد. یکی از این روش ها، مجازی سازی نرم افزار (*Software Virtualization*) می باشد. برای مجازی سازی نرم افزار نیاز به سیستم عاملی است که درخواست هایی را که از حد فاصل بین *protected mode* و *real mode* می گذرد، قطع کرده و پس از تثبیت وضعیت رجیسترها، مد کاری *CPU* را از *protected mode* به *real mode* تغییر دهد. این کار از طریق *VxD* صورت می گیرد. *VxD* تقاضای اینتراپت *protected mode* را به یک تقاضا در *IVT* مربوط به *real mode* تبدیل کرده و در ادامه، تبدیلات پارامترهای لازم را مطابق با *real mode* انجام می دهد (سیستم عامل *real mode* در اینجا *V86 mode* است)، سپس *VxD* نتایج حاصله از *real mode* را به *protected mode* برمی گرداند. کل عملیات در شکل زیر نشان داده شده است.[1],[2]



شکل۳: مجازی سازی *BIOS* و *MS-DOS* توسط *VxD*

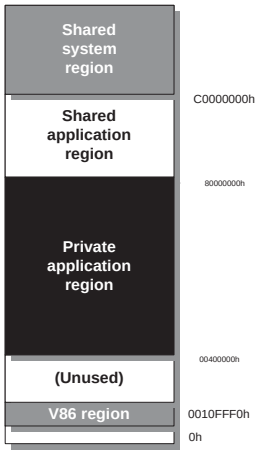
۳-۳- VMM (Virtual Machine Manager)

VMM هسته مرکزی ویندوز ۹۸ است. *VMM* یک چارچوب برای مدیریت *VM*ها بنا کرده و از آن محافظت می کند. *VxD*ها به همراه *VMM* برای مجازی سازی سخت افزار و فراهم سازی سرویس های سیستم برای سایر *VxD*ها و برنامه های *ring 3* کار می کنند. *VMM* خود یک *VxD* است که در ابتدای بالا آمدن سیستم عامل، *load* می شود. *VMM* به عنوان یک درایور مجازی به همراه *VM*ها این امکان را فراهم می سازد تا برنامه ها همزمان با هم اجرا شوند. بطوریکه هر یک از آنها تصور می کند که به تنهایی با ماشین واقعی در تقابل می باشد. *VMM* یک سری توابع کاربردی تولید می کند که تقریبا توسط همه *VxD*ها فراخوانی می شوند؛ و از این طریق بر همه *VxD*ها نظارت دارد.

سرویس های *VMM* در برگیرنده مدیریت حافظه، *handle* کردن اینتراپت ها و مدیریت خطاهای *protection* می باشد.[1]

۳-۴- مدیریت حافظه (Memory Management)

نکته مهم در این جا دانستن نحوه بخش بندی فضای آدرس توسط *VMM* می باشد. *VMM* فرآیند *paging* را برای پیاده سازی فضای مجازی ۳۲ بیتی بکار می برد. همچنین *VMM* برای پیاده سازی مدیریت حافظه، فضای آدرس قابل دسترسی سیستم را به ۵ ناحیه مطابق شکل زیر بخش بندی می کند.



شکل۴: فضای آدرس مجازی ویندوز ۹۸

- ناحیه *V86*: متعلق به *VM* در حال اجراست.

- ناحیه خصوصی *Application*: این قسمت از حافظه مجازی متعلق به پروسس *win32* است. *VM* یا برنامه *win32* که به یک پروسس مشخص تعلق دارد، هر یک محدوده خاص خود را در این بخش در اختیار می گیرد.
- ناحیه مشترک *Application*: ویندوز تعدادی از *DLL*های *ring 3* و همه برنامه های *win16* را در این ناحیه از فضای آدرسی قرار می دهد.
- ناحیه مشترک سیستم: این ناحیه متعلق به اطلاعات و برنامه هایی از سیستم است که بین همه پروسس ها و *VM*ها مشترک می باشد. این بخش حافظه مکانی است که *VMM* و سایر *VxD*ها در آن جای می گیرند. *Page*های هر ناحیه *V86* مربوط به یک *VM* نیز به این ناحیه از حافظه نگاشته می شود. این به یک *VxD* اجازه می دهد که به حافظه *V86* مربوط به یک *VM* خاص دسترسی داشته باشد؛ حتی اگر آن *VM* در حافظه حضور نداشته باشد.[2]

۳-۵- handle کردن اینتراپت ها:

اینتراپت مکانیسم اساسی برای معلق کردن فرآیند اجرایی حالت نرمال کامپیوتر می باشد تا سیستم عامل بتواند یک حالت استثنایی را *handle* کرده یا یک سرویس سیستم را در اختیار برنامه قرار دهد. خلاصه عملیاتی که یک *VMM* برای این منظور انجام می دهد بدین صورت است:

VMM هنگامی که یک اینتراپت رخ می دهد بیدار می شود، سپس آن اینتراپت را *handle* کرده و در ادامه دستور *IRETD* را اجرا می کند.

VMM شامل مجموعه بزرگی از *handler*های اینتراپتی مرتبه اول (*first level interrupt handler*) می باشد. بطوریکه برای هر اینتراپت موجود در سیستم یکی را اختصاص می دهد و وظیفه آن اینست که حالت برنامه اینتراپت خورده را ذخیره کرده و کنترل را به *handler* اینتراپتی مرتبه دوم (*second level interrupt handler*) که سرویس دهنده اصلی اینتراپت است، منتقل کند.

جدول توصیف کننده اینتراپت IDT (Interrupt Descriptor Table) :

هر *VM* شامل ۲ تا *IDT* می باشد: یکی برای برنامه های *V86 mode* و دیگری برای برنامه های *protected mode*. *VMM*، *IDT*ی یک *VM* را در زمان راه اندازی آن ایجاد می کند. مقدار اولیه *IDT* از یک جدول معین که *VMM* و سایر *VxD*ها در طول راه اندازی سیستم ایجاد می کنند بدست می آید.

انواع اینتراپت ها:

ویندوز ۹۸ بین اینتراپت های سخت افزاری، نرم افزاری و *exception*ها تفاوت قائل می شود. پروسسور، *exception*ها را زمانی تولید می کند که تشخیص دهد سیستم عامل نیاز به تحلیل برخی شرایط استثنایی دارد.

کلاس بندی اینتراپت ها مشخص می کند که چگونه *VxD*ها برنامه های *protected mode* را شکار کند و چگونه *VMM* به آنها سرویس دهد. *VxD*ها سرویس های مختلفی از *VMM* را برای بدام انداختن انواع اینتراپت ها بکار می برند.

یکی از اهداف *VMM* این بود که به نحوی مجازی سازی کامپیوتر را انجام دهد که ویندوز و بخش های مختلف مربوط به *MS-DOS* با یکدیگر همزمان اجرا شوند. برای اینکه مجازی سازی درست انجام شود پیش فرض *VMM* برای تحلیل هر اینتراپت این است که آن را به *VM* متناظر منعکس کرده تا *ring 3* با آن درگیر شود.

انعکاس اینتراپت بدین صورت است که تصاویر رجیسترهای ذخیره شده متعلق به *VM* مورد جستجو را تغییر می دهد تا *ISR* (*Interrupt Service Routine*) مناسب کنترل را بدست گیرد. انعکاس فوری یک اینتراپت سخت افزاری همیشه کار درستی نیست زیرا ممکن است *handler* مربوط به آن اینتراپت در یک *VM* دیگر به غیر از *VM* فعال در لحظه وقوع اینتراپت قرار داشته باشد. لذا *VM* و سایر *VxD*ها نیاز به اینتراپت های نرم افزاری دارند تا منابع مربوط به این اینتراپت ها را مجازی سازی کنند.[1],[2]

۴- نتیجه گیری:

همانگونه که در مقدمه مطرح شد، استفاده همزمان چندین *application* از یک سخت افزار، تابع رعایت یک سری قواعد می باشد. هر *VxD* یی که مجازی سازی یک سخت افزار را انجام می دهد، یک سری سرویس تحت نظارت *VMM* برای استفاده سایر *VxD*ها و *application*ها صادر می کند. هر *application* از طریق سرویس های منتشر شده درخواست استفاده از یک سخت افزار را می دهد. *VMM* برای هر کدام از این *application*ها یک فضای حافظه اختصاص می دهد که در آن به فضای آدرسی *VxD*های در ارتباط با آن سخت افزار اشاره شده است. سپس *VxD*ها به این درخواستها پاسخ داده نتایج را به فضای آدرس *application* مورد نظر *map* می کند. این فضای آدرسی *VM* نام دارد.[1],[2]

چارچوب یک *VxD* ساده به زبان اسمبلی:

```
Myvxd.asm
.386p
include VMM.inc; required
include debug.inc; optional but usual
Declare_Virtual_Device …;
Begin_Control_Dispatch myvxd
        [Control_Dispatch macros]
End_Control_Dispatch myvxd
VXD_IDATA_SEG
        [Initialization – only data]
VXD_IDATA_ENDS
VXD_ICODE_SEG
        [Initialization – only code]
VXD_ICODE_ENDS
VXD_LOCKED_DATA_SEG
        [page – locked - data]
VXD_LOCKED_DATA_ENDS
VXD_LOCKED_CODE_SEG
        [page – locked -code]
VXD_LOCKED_CODE_ENDS
End.
```

ماکروی *Declare_Virtual_Device* دارای ساختار زیر است:

Declare_Virtual_Device Name, MajorVer, MinorVer, CtrlProc, DeviceNum, InitOrder, V86Proc, PMProc, RefData

Name، یک اسم *n* بایتی است که برای هر *VxD* قابل اجرا روی سیستم، واحد می باشد.

MajorVer و *MinorVer* ،پارامترهای اطلاعاتی در مورد نسخه درایور می باشند.

CtrlProc، اسم *Device Control Procedure* می باشد، که پیغامهای کنترلی سیستم را در طول زندگی درایور پروسس میکند.

DeviceNum ، بعضی از درایورها نیاز به یک معرف (*Identifier*) واحد دارند، تا سایر نرم افزارها بتوانند آنها را آدرس دهی کنند.

InitOrder، هر درایوری که یک *Initialization Order* دارد، که آن را در یک دنباله ترتیبی با همه درایورهای دیگر موجود در سیستم قرار می دهد.

هر *VxD* نیاز به یک *Device Control Procedure* دارد تا به به پیغام های کنترلی سیستم جواب دهد.

```
Begin_Control_Dispatch vxdname
        Control_Dispatch message, function
.
.
End_Control_Dispatch vxdname
```

ماکروی *Begin_Control_Dispatch*، این کد اسمبلی را در یک سگمنت *lock* شده قرار می دهد. ماکروی *Control_Dispatch* مشخص کننده تابعی است که یک پیغام کنترلی را *handle* میکند.

ماکروهای *VXD_IDATA_SEG* و *VXD_ICODE_SEG* به ترتیب آغاز سگمنت داده ها و کدهای مربوط به بخش راه اندازی *VxD* را مشخص می کنند.

ماکروهای *VXD_LOCKED_DATA_SEG* و *VXD_LOCKED_CODE_SEG* به ترتیب آغاز سگمنت داده ها و کدهایی می باشند که باید همواره در حافظه حضور داشته باشند. برای مثال کدی که باید اینتراپت را *handle* کند باید در *locked_page*ها (سگمنت *LCODE*) آورده شود، زیرا ویندوز نمی تواند به یک *page fault* در حالی که یک اینتراپت سخت افزاری را *handle* می کند، پاسخ دهد.[3],[2]

مراجع

[۱] سید کاظم جهانبخش، ناصر جهانگیری، مسعود یعقوبی. "درایور یک کارت در محیط ویندوز ۹۸"، پایان نامه کارشناسی دانشگاه صنعتی شریف، بهار ۱۳۸۱.

[۲] Walter Oney. *Systems Programming for Windows 98*, Microsoft Press, 1996.

[۳] Device Driver Kit Documentation (DDK 98).