بررسی معماری سیستم عامل ویندوز
(بررسی سطح کاربر و هسته)

مریم نژادکمالی

nezhadkamali@cert.um.ac.ir

آزمایشگاه تخصصی آپا در زمینه امنیت فن‌آوری اطلاعات و ارتباطات

http://cert.um.ac.ir

cert@um.ac.ir

ویرایش اول – خردادماه ۱۳۹۳

شماره سند: APA_FUM_W_MAL_0117

چکیده

سیستم‌های عامل جدید از نظر معماری اهمیت زیادی برای امنیت قائل می‌شوند. یکی از موارد مهم در بحث امنیت، حفاظت از خود سیستم عامل است و این مورد به افزایش پایداری سیستم عامل هم کمک می‌کند. قسمت اصلی سیستم عامل هسته می‌باشد که مدیریت منابع سیستم عامل و اشتراک آنها را برعهده دارد. بنابراین در معماری IA32 شرکت اینتل چهار سطح پیاده‌سازی شده تا از هسته‌ی سیستم عامل حفاظت شود. در ویندوز تنها از دو سطح استفاده می‌شود که به آنها سطح کاربر و هسته گفته می‌شود. در این مقاله قصد داریم که این دو سطح را در ویندوز بیشتر مورد بررسی قرار دهیم.

واژه‌های کلیدی

ویندوز، مود هسته، مود کاربر، PAE، AWE، 4GT.

۱- مقدمه

در ویندوز مرز بین سیستم عامل و برنامه‌های کاربر به مکانیزم‌های سخت‌افزاری نیز وابسته است. بنابراین بحث مودهای هسته و کاربر به بخش نرم‌افزاری محدود نمی‌شود و در پیاده‌سازی این دو مود حساس از سخت‌افزار هم به صورت جدی استفاده می‌شود.

در معماری IA32 شرکت اینتل چهار سطح پیاده‌سازی شده که ما به این سطوح اصطلاحاً حلقه[1] می‌گوئیم که هر چه به سمت داخل می‌رویم سطح دسترسی افزایش می‌یابد. این حلقه‌ها از ۰ تا ۳ نام‌گذاری می‌شوند و عبارتند از:

- حلقه ۰ یا مود هسته: در این حلقه به تمام منابع دسترسی داریم و در این مود هسته اجرا می‌شود.
- حلقه ۱ و۲: که می‌تواند دارای سطح دسترسی‌های مختلفی باشد.
- حلقه ۳ یا مود کاربر: دسترسی کاملا محدودی به منابع سیستم عامل دارد.

سیستم عامل ویندوز فقط از دو حلقه استفاده می‌کند که همان حلقه‌های ۰ و ۳ هستند.

۲- فضای کاربر و فضای هسته

در نسخه‌های ۳۲ بیتی ویندوز، برنامه‌ها از فضای مجازی 4GB برخوردار هستند. ویندوز به استفاده از صفحه‌بندی[2] علاقه‌مند است و این مکانیزم حافظه مجازی 4GB را به دو قسمت زیر تقسیم می‌کند:

- فضای کاربر (آدرس‌های خطی 0x00000000 تا 0x7FFFFFFF)
- فضای هسته (آدرس‌های خطی 0x80000000 تا 0xFFFFFFFF)

به صورت پیش‌فرض فضای کاربر نیمه‌ی پایینی محدوده حافظه است و فضای هسته نیمه‌ی بالایی را دربرمی‌گیرد. بنابراین فضای آدرس‌دهی 4GB به دو قسمت 2GB ای تقسیم می‌شود.

۲-۱- تخصیص فضای کاربر و هسته به پردازه‌ها

اگرچه محدوده‌ی آدرس‌های خطی برای تمام پردازه‌ها مشابه است (یعنی بین 0x00000000 و 0x7FFFFFFF)، با

[1] Ring
[2] Paging

این حال ویندوز ضمانت می‌کند که هر آدرس فیزیکی در این محدوده برای هر پردازه متفاوت نگاشت شود. به عبارت دیگر حتی اگر دو پردازه دسترسی به یک آدرس خطی داشته باشند، آدرس فیزیکی آنها متفاوت خواهد بود. یعنی هر پردازه فضای کاربر مخصوص به خودش را دارا است. برای مثال آدرس خطی 0x00020001 را به همراه دو راهنمای صفحه درنظر می‌گیریم که یکی در آدرس فیزیکی 0x06e83000 و دیگری در آدرس فیزیکی 0x014b6000 مقیم هستند. با استفاده از دستور vtop! که وظیفه‌ی تبدیل آدرس مجازی به آدرس فیزیکی را دارد، خروجی‌های زیر را خواهیم داشت:

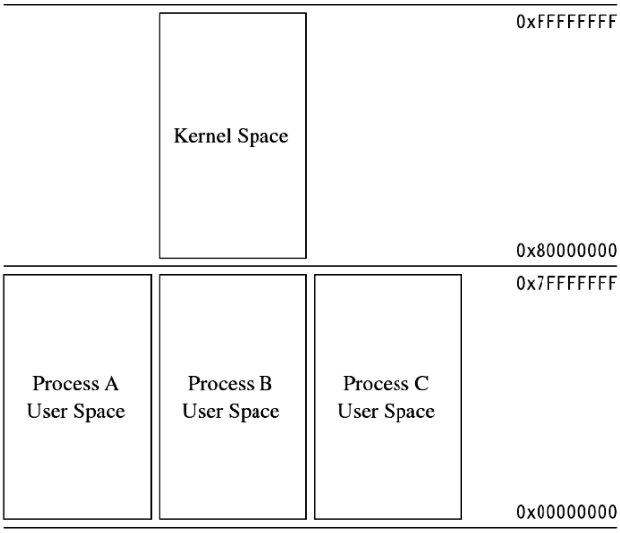
```
kd> !vtop 6e83 20001
Pdi 0 Pti 20
00020001 0db74000 pfn(0db74)

kd> !vtop 14b6 20001
Pdi 0 Pti 20
00020001 1894f000 pfn(1894f)
```

شکل ۱ – خروجی دستور vtop! **با استفاده از دو راهنمای صفحه متفاوت**

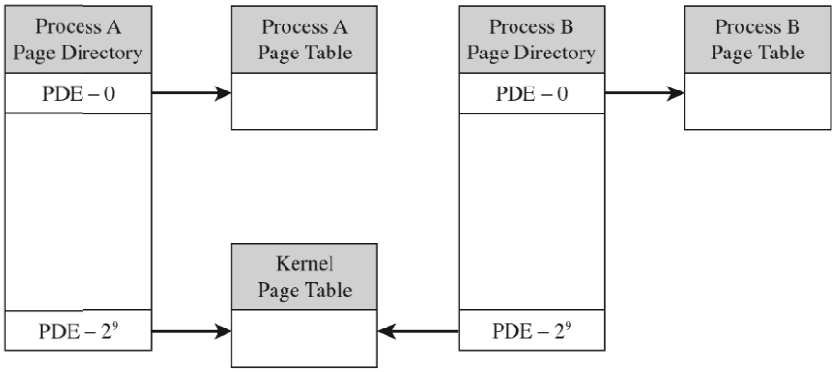
دستور اول نشان می‌دهدکه آدرس خطی 0x00020001 به یک بایت در حافظه فیزیکی و به یک صفحه با PFNای با شماره 0x0db74 اشاره دارد و دستور دوم نشان می‌دهد که آدرس خطی مشابه با قبلی به یک بایت در یک صفحه با PFNای که دارای شماره 0x1894f است اشاره می‌کند. بنابراین با این که یک آدرس خطی داشتیم ولی آدرس‌های فیزیکی متفاوتی برای آنها وجود داشت.

از طرفی اگر چه که هر پردازه‌ای فضای کاربر مخصوص به خودش را دارا است، با این حال تمامی آنها یک فضای هسته را به اشتراک دارند. و این یک ضرورت است، چرا که بدین وسیله تنها یک سیستم عامل می‌تواند وجود داشته باشد.



شکل ۲ – هر پردازه فضای کاربر مخصوص به خود و فضای هسته یکسان دارد.

بنابراین هر پردازه‌ای جدول صفحه مختص به خود را خواهد داشت، اما از طرفی مدخل جدول صفحه‌ی سطح ناظر هر پردازه به یک مجموعه جدول صفحه مشابه در هسته اشاره خواهد داشت (مطابق شکل۲).

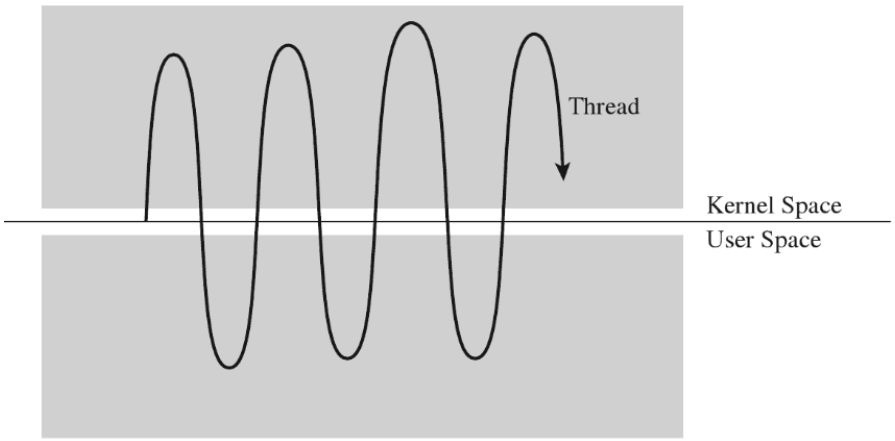


شکل ۳- مدخل جدول صفحه‌ی سطح ناظر هر پردازه به یک مجموعه جدول صفحه مشابه در هسته اشاره دارد.

۲-۲- دسترسی به فضای هسته

این تصور که کدهای کاربر و هسته در فضای محدود حافظه‌ی کاربر و هسته مربوط به خودشان به صورت مستقل از هم اجرا می‌شوند، تصوری اشتباه است. مطمئنا فضای آدرس هسته حفاظت شده است، به طوری‌که یک نخ برنامه کاربردی باید از طریق دروازه‌ی فراخوانی‌های سیستمی به هسته دسترسی پیدا کند. بنابراین یک نخ اجرایی می‌تواند از فضای کاربر شروع شود، از طریق دستور SYSENTER (یا INT 0x2E) به فضای هسته برود و سپس به فضای کاربر برگردد. در واقع دید درست نسبت به مسیرهای اجرایی در ویندوز این است که تصور کنید چندین نخ اجرایی مدام

در حافظه کاربر اجرا می‌شوند و به اوج می‌رسند و چندین بار بین فضای کاربر و هسته جابه جا می‌شوند و سپس دوباره به فضای کاربر بر می‌گردند.



شکل ٤- تغییر روند اجرای یک پردازه

در حالت عادی این دستورات ویژه (SYSENTER و INT 0x2E) به همراه رجیسترهای سطح سیستم و ساختمان داده‌هایی که توسط آنها استفاده می‌شود، در هنگام بوت سیستم توسط هسته تنظیم می‌شوند. این عمل باعث می‌شود تا کدهای برنامه‌ی کاربر نتواند به صورت دلخواهانه وارد فضای هسته شود و با امتیازات بالا اجرا شود. بدین صورت هسته یک فضای محافظت شده خواهد بود.

۲-۳- چیدمان فضای کاربر

برای این که بدانیم در فضای کاربر اجزا چگونه چیده شده‌اند سراغ ابزار kd.exe[1] میرویم. ابتدا با استفاده از دستور process! متوجه آدرس خطی برنامه Explorer.exe می‌شویم. سپس با فراخوانی دستور process. مشخص می‌کنیم که به دنبال محتوای این پردازه هستیم. و با استفاده از دستور peb! می‌توانیم ساختار PEB[2] پردازه را به دست آوریم.

[1] Kernel Debugger یک ابزار برای اشکال زدایی هسته است .

[2] Pocess Environment Block

```
kd> !process 0 0x1 Explorer.exe
PROCESS 834eed08  SessionId: 1  Cid: 0824    Peb: 7ffd6000  ParentCid: 0710
    DirBase: 02cfd000  ObjectTable: 93590f78  HandleCount: 479.
    Image: explorer.exe

kd> .process 834eed08
Implicit process is now 834eed08

kd> !peb
PEB at 7ffd6000
    InheritedAddressSpace:    No
    ReadImageFileExecOptions: No
    BeingDebugged:            No
    ImageBaseAddress:         00f50000
    Ldr                       77874cc0
    Ldr.Initialized:          Yes
    Ldr.InInitializationOrderModuleList: 002915f0 . 038d77b8
    Ldr.InLoadOrderModuleList:           00291570 . 038d77a8
    Ldr.InMemoryOrderModuleList:         00291578 . 038d77b0
    Base TimeStamp                     Module
    f50000 47918e5d Jan 18 21:45:01 2008 C:\Windows\Explorer.EXE
    777b0000 4791a7a6 Jan 18 23:32:54 2008 C:\Windows\system32\ntdll.dll
    76230000 4791a76d Jan 18 23:31:57 2008 C:\Windows\system32\kernel32.dll
    76610000 4791a64b Jan 18 23:27:07 2008 C:\Windows\system32\ADVAPI32.dll
    77550000 4791a751 Jan 18 23:31:29 2008 C:\Windows\system32\RPCRT4.dll
    769f0000 4791a6a5 Jan 18 23:28:37 2008 C:\Windows\system32\GDI32.dll
    76440000 4791a773 Jan 18 23:32:03 2008 C:\Windows\system32\USER32.dll
    76500000 4791a727 Jan 18 23:30:47 2008 C:\Windows\system32\msvcrt.dll
    765b0000 4791a75c Jan 18 23:31:40 2008 C:\Windows\system32\SHLWAPI.dll
    SubSystemData:     00000000
    ProcessHeap:       00290000
    ProcessParameters: 00290f00
    WindowTitle:  'C:\Windows\Explorer.EXE'
    ImageFile:    'C:\Windows\Explorer.EXE'
    CommandLine:  'C:\Windows\Explorer.EXE'
    DllPath:      'C:\Windows;C:\Windows\system32;C:\Windows\system;
...
```

شکل ۵ – آدرس برنامه Explorer **و** DLL **های آن در حافظه**

همان طور که در خروجی مشخص است می‌توانیم آدرس‌های خطی برنامه Explorer.exe و Dllهایی را که بار کرده است مشاهده کنیم و ترتیب قرار گرفتن آنها را در حافظه متوجه بشویم. همان طور که انتظار داریم آدرس‌ها همگی در محدوده‌ی فضای کاربر یعنی 0x00000000 تا 0x7FFFFFFF هستند.

یکی دیگر از راه‌هایی که می‌توان به وسیله آن به موقعیت برنامه‌ها در فضای کاربر پی برد، ابزار vmmap از

برنامه‌های Sysinternal است. این برنامه اطلاعات جامعی را در مورد اجزای مختلف پردازه می‌دهد.

۲-٤- تخصیص پویای فضای هسته

در نسخه‌های قدیمی‌تر ویندوز (مانند windows Server 2003, XP) اندازه و موقعیت منابع بحرانی سیستم ثابت بود. قسمت‌های مختلف سیستم عامل مانند مدخل‌های جدول صفحه، *کپی حافظه هسته*[1]، و کش سیستم به طور ایستا تخصیص داده می‌شدند و در آدرس‌های خطی مشخصی بودند.

با آمدن نسخه‌های ویندوز ویستا و ویندوز سرور ۲۰۰۸، هسته می‌تواند تخصیص حافظه را به صورت پویا انجام دهد و حافظه را منطبق با تغییرات تقاضا از دوباره بچیند. با این حال می‌توان با استفاده از یک ابزار، از حافظه‌ی هسته یک رونوشت گرفت و فهرست تمام Dllهایی که در حافظه بار شده اند و آدرس‌های آنان را به دست آورد.

```
kd> lm n
start    end        module name
8183c000 81be6000   nt       ntkrnlmp.exe
85ce6000 85d06000   mrxdav   mrxdav.sys
85d3e000 85e42000   VSTDPV3  VSTDPV3.SYS
85e42000 85e70000   msiscsi  msiscsi.sys
85e70000 85eb1000   storport storport.sys
85eb1000 85ebc000   TDI      TDI.SYS
85ebc000 85ed3000   rasl2tp  rasl2tp.sys
85ed3000 85ede000   ndistapi ndistapi.sys
85ede000 85f01000   ndiswan  ndiswan.sys
85f01000 85f10000   raspppoe raspppoe.sys
85f10000 85f24000   raspptp  raspptp.sys
85f24000 85f39000   rassstp  rassstp.sys
85f39000 85fc2000   rdpdr    rdpdr.sys
85fc2000 85fd2000   termdd   termdd.sys
85fd2000 85fdc000   mssmbios mssmbios.sys
...
```

شکل٦- فهرست DLL های بار شده در فضای هسته

فرمان lm n آدرس شروع و پایان تمام ماژول‌هایی[2] را که در فضای حافظه هسته، بار شده‌اند را فهرست می‌کند. همان‌طور که می‌بینید تمام ماژول‌ها در نیمه بالای حافظه یعنی حافظه هسته مقیم هستند.

[1] Memory image of kernel

[2] ماژول یک تصویر حافظه از یک فایل باینری است که شامل کد اجرایی است و می تواند به نمونه هایی از EXE. یا DLL. و یا SYS. اشاره داشته باشند .

۲-۵- تعیین میزان فضای کاربر و هسته

همانطور که گفته شد به صورت پیش‌فرض فضای 4GB یک پردازنده ۳۲ بیتی به دو قسمت 2GB تقسیم می‌شود، که به صورت پیش‌فرض نیمه پایینی حافظه مربوط به فضای کاربر و نیمه بالایی مربوط به فضای هسته است. در قسمت‌های بعدی روش‌هایی ذکر خواهد شد که می توان به وسیله آن این روش تخصیص را تغییر داد.

۲-۵-۱- میزان‌سازی 4GigaByte[1]

اختصاص دهی فضای 2GB به صورت پیش‌فرض است و حتما لازم نیست که تقسیم فضا به صورت ۵۰-۵۰ باشد. با استفاده از فرمان BCDedit.exe می‌توان مکان خط تقسیم را تغییر داد و به کاربر فضای 3 GB داد.

```
bcdedit /set increaseuserva 3072
```

شکل ۷ – دستور BCDedit **برای تغییر فضای کاربر به** 3GB

برای این که یک برنامه کاربردی بتواند از این فضای اضافی استفاده کند، یک پرچم خاص به نام IMAGE_FILE_LARGE_ADDRESS_AWARE در بخش سرآیند فایل باینری برنامه، باید به ۱ تنظیم شده باشد. این پرچم در حالت عادی توسط لینکر در زمانی که برنامه ساخته می‌شود، تنظیم می‌شود. برای مثال لینکر برنامه Visual Studio C++ یک گزینه به نام /LARGEADDRESSAWARE دارد. برای این که بتوانید متوجه شوید که در یک برنامه این پرچم تنظیم شده است یا نه، می‌توانید از dumpbin.exe که در SDK [2] وجود دارد استفاده کنید.

[1] 4GT) 4 Giga Tuning(
[2] Software development kit

```
dumpbin /headers C:\windows\system3 2\smss.exe
Microsoft (R) COFF/PE Dumper Version 9.00.30729.01
Copyright (C) Microsoft Corporation.  All rights reserved.
Dump of file C:\windows\system32\smss.exe

PE signature found

File Type: EXECUTABLE IMAGE

FILE HEADER VALUES
             14C machine (x86)
               4 number of sections
        4A5BBF10 time date stamp Mon Jul 13 16:11:12 2009
               0 file pointer to symbol table
               0 number of symbols
              E0 size of optional header
             122 characteristics
                   Executable
                   Application can handle large (>2GB) addresses
                   32 bit word machine
```

شکل ۸- سرآیند برنامه smss.exe

توجه کنید که آدرس‌های مجاور محدوده‌ی 2GB معمولا توسط dllهای سیستم استفاده می‌شود. بنابراین یک پردازه‌ی ۳۲ بیتی نمی‌تواند حافظه‌ی پیوسته‌ای بیش از 2GB را تخصیص دهد، حتی اگر تمامی فضای آدرس 4GB در دسترس باشد.

۲-۵-۲- تکنیک Address Windowing Extension

اگرچه 4GT به ما اجازه می‌دهد که بتوانیم فضای حافظه کاربر را به بیش از 3GB ارتقا دهیم، اما با این حال برخی ابزارهای مدیریت پایگاه داده یا نرم‌افزارهای مهندسی به بیش از این حافظه نیاز دارند. تکنیک AWE[1] می‌تواند پاسخگوی این مساله باشد. AWE این اجازه را به برنامه‌های ۳۲ بیتی کاربر می‌دهد که حافظه‌ی فیزیکی بیش از 64GB را در اختیار بگیرند. این امکان براساس توابع API است که در فایل سرآیند winbase.h تعریف شده‌اند. این توابع API این اجازه را به برنامه می‌دهند که به فضای حافظه بزرگتری از آنچه در اختیار دارند دسترسی پیدا کنند. در جدول شماره ۱ فهرستی از این توابع آورده شده است.

[1] Address Windowing Extension

جدول ۱- توابع API مربوط به AWE

API Function	Description
VirtualAlloc()	Reserves a region in the linear address space of the calling process
VirtualAllocEx()	Reserves a region in the linear address space of the calling process
AllocateUserPhysicalPages()	Allocate pages of physical memory to be mapped to linear memory
MapUserPhysicalPages()	Map allocated pages of physical memory to linear memory
MapUserPhysicalPagesScatter()	Like MapUserPhysicalPages() but with more bells and whistles
FreeUserPhysicalPages()	Release physical memory allocated for use by AWE

AWE از یک تکنیک به نام پنجره‌بندی[1] استفاده می‌کند. در این تکنیک مجموعه‌ای از ناحیه‌های[2] با اندازه ثابت (که به آنها پنجره گفته می‌شود) در فضای آدرس خطی یک برنامه تخصیص داده می‌شود و سپس به یک مجموعه از پنجره‌ها با اندازه ثابت بزرگتر در حافظه فیزیکی نگاشت می‌شود. حافظه‌ای که از طریق AWE اختصاص یافته باشد، به هیچ عنوان صفحه‌بندی نمی‌شود. AWE می‌تواند بدون PAE[3] نیز استفاده شود. البته اگر یک برنامه به منظور بالا بردن محدودیت تا بالاتر از 4GB از AWE استفاده کند، حتما نیاز دارد که PAE را نیز فعال کند. به علاوه، اگر کاربر برنامه‌ای را اجرا کند که از روال‌های AWE استفاده کرده باشد باید دارای امتیاز "Lock Pages in Memory" باشد.

۲-۵-۳- PAE دربرابر 4GT و AWE

PAE یکی از ویژگی‌های سخت‌افزار اینتل است که به پردازنده‌های ۳۲ بیتی اجازه می‌دهد که با آدرس‌های فیزیکی بیش از 4GB کار کنند. سیستم عامل‌هایی مانند ویندوز ۷ از PAE به جز برای آسان کردن DEP استفاده نمی‌کنند.

4GT یکی از ویژگی‌های مربوط به ویندوز است. با استفاده از 4GT فضای آدرس مجازی که برای برنامه‌ها موجود است، می‌تواند به بیش از 3GB ارتقا یابد. برای استفاده از 4GT نیازی نداریم که PAE را فعال کنیم. اگر یک برنامه واقعا بخواهد بیش از 3GB را در اختیار بگیرد باید از AWE استفاده کند. و اگر برنامه‌ای که از AWE استفاده می‌کند، بخواهد بیش از 4GB را در اختیار بگیرد باید PAE را نیز فعال کند.

[1] Windowing
[2] Regions
[3] Physical Address Extension

۳- مود کاربر و مود هسته

در بخش قبلی متوجه شدیم که حافظه به دو بخش فضای کاربر و هسته تقسیم شده است. فضای کاربر شامل کدهایی است که در یک حالت محصور اجرا می شوند، که اصطلاحا به آن مود کاربر می‌گویند. کدهایی که در مود کاربر اجرا می‌شوند به هیچ وجه نمی‌توانند با فضای هسته یا سخت‌افزار مستقیما ارتباط برقرار کنند، و یا دستوراتی را فراخوانی کنند که نیاز به امتیازات بالا دارد. درواقع تنها سیستم عامل و درایورهای مربوط به آن در فضای هسته اجرا می‌شوند. به این حالت که کدها در فضای هسته با امتیازات خاص اجرا می‌شوند، مود هسته گفته می‌شود. برنامه‌ای که در مود هسته است امتیازاتی را دارد و کارهایی را می‌تواند انجام دهد که یک برنامه در مود کاربر قادر به انجام نیست. در واقع کدهایی که در مود هسته اجرا می‌شوند بر دیگر قسمت‌های ماشین سلطنت می‌کنند.

۳-۱- رابطه مود و فضا

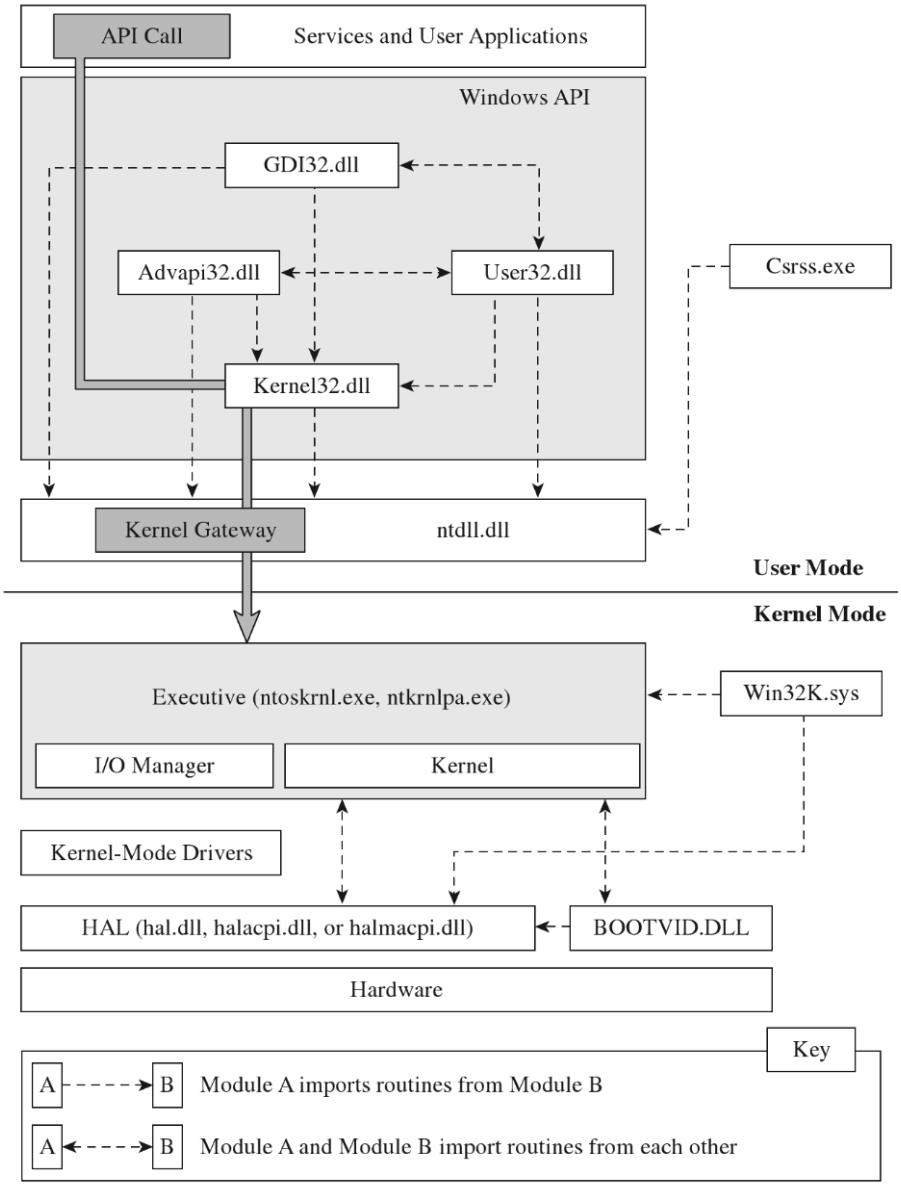
مود کاربر و مود هسته حالت‌هایی را مشخص می‌کنند که دستورات برنامه برای اجرا مجبور به تبعیت از آنها هستند. در واقع لغت مود مشخص می‌کند که یک کد چگونه اجرا شود و لغت فضا مکان آن را در حافظه مشخص می‌کند. بنابراین دو مفهوم وجود دارد که با نگاشت یک به یک به یکدیگر مرتبط می‌شوند. یعنی کدی که در فضای حافظه کاربر باشد با مود کاربر اجرا می‌شود و کدی که در فضای حافظه هسته باشد با مود هسته نیز اجرا می‌شود. البته این نگاشت مستقیم در شرایط عادی اتفاق می‌افتد و همیشه ضروری نخواهد بود. بلکه با دستکاری در GDT می‌توانیم کد موجود در فضای کاربر را با مود هسته اجرا کنیم و بالعکس.

۳-۲- اجزای سیستم عامل

سیستم‌عامل در واقع یک نرم‌افزار است که وظیفه‌ی مدیریت منابع سخت‌افزار و سرویس‌های مشترک بین برنامه‌ها را دارد. اجزای سیستم‌عامل به منظور تعامل و کار بخش‌های متفاوت سیستم‌عامل با یکدیگر طراحی شده است. هر کدام از بخش‌های سیستم عامل دارای اجزای مشخصی می باشد که در بخش‌های بعدی ذکر خواهد شد.

۳-۲-۱- اجزای مود هسته

در این قسمت می‌خواهیم در مورد اجزای اصلی هسته سیستم عامل صحبت کنیم و مشخص کنیم که هرکدام از آنها در کجای حافظه مقیم هستند و نقش‌های آنها را در حین فراخوانی‌های سیستمی حدس بزنیم. شکل زیر یک تصویر خلاصه از توضیحاتی را که خواهیم داد نمایش می‌دهد.



شکل ۹ – اجزای اصلی سیستم عامل

در بالای قسمت سخت‌افزار *لایه انتزاعی سخت افزار*[1] (HAL) قرار دارد. HAL به منظور جدا کردن سیستم‌عامل

[1] Hardware abstraction layer(HAL)

از سخت‌افزار ایجاد شده است. HAL به وسیله API های پیاده‌سازی شده در DLL مربوط به خودش جزییات مبتنی بر ماشین را پنهان می‌کند. درایورهای دستگاه که در مود هسته کار می‌کنند به جای ارتباط مستقیم با سخت‌افزار روال‌های HAL را فراخوانی می‌کنند. این کار کمک می‌کند که درایورها بیشتر قابل حمل باشند و به سخت‌افزار وابسته نباشند. فایل DLL اصلی HAL وابستگی شدیدی به سخت‌افزاری دارد که هم اکنون ویندوز در آن در حال اجرا است. برای مثال HAL بر روی ماشین‌هایی که ۶۴ بیت هستند با نام hall.dll شناخته می‌شود. برای کامپیوترهای ۳۲ بیتی که ACPI[1] را فراهم می‌کنند، HAL توسط یک فایل با نام halacpi.dll ایجاد می‌شود. در ماشین‌های ۳۲ بیتی که ACPI دارند و از چندین پردازنده استفاده می‌کنند، HAL با فایل hallmacpi.dll پیاده‌سازی می‌شود. فایل HAL در سیستم‌های desktop با نام hal*.dll شناخته می‌شود و در آدرس %windir%\system32 موجود می‌باشد. برای این که بدانیم چه نسخه‌ای از HAL بر روی سیستم ما موجود است می‌توانیم با استفاده از دستور lm n در kd.exe ماژول‌های بار شده در حافظه را ببینیم که فایل dll مربوط به HAL یکی از آنها است.

```
kd> lm n
start    end        module name
00510000 00572000   kd         kd.exe
64f00000 65286000   dbgeng     dbgeng.dll
6c700000 6c821000   dbghelp    dbghelp.dll
6eb10000 6eb58000   symsrv     symsrv.dll
74d20000 74d29000   VERSION    VERSION.dll
75ad0000 75b1a000   KERNELBASE KERNELBASE.dll
75bc0000 75bd9000   sechost    sechost.dll
75cd0000 75d70000   ADVAPI32   ADVAPI32.dll
76ec0000 76f61000   RPCRT4     RPCRT4.dll
76f70000 7701c000   msvcrt     msvcrt.dll
77280000 77354000   kernel32   kernel32.dll
776d0000 7780c000   ntdll      ntdll.dll
80ba5000 80bad000   kdcom      kdcom.dll
82805000 82c05000   nt         ntkrnlmp.exe
82c05000 82c2d000   hal        halacpi.dll
...
```

شکل ۱۰- ماژول های بار شده در حافظه

در پایین شکل ۸، در کنار HAL یک فایل به نام BOOTVID.DLL وجود دارد که وظیفه آن حمایت کردن از

[1] Advanced Configuration and power interface

گرافیک VGA اصلی در مرحله بوت شدن است.

هسته‌ی اصلی سیستم عامل ویندوز در فایل باینری nt*.exe قرار دارد. این فایل اجرایی کارهایش را در دو لایه پیاده‌سازی کرده است : مجری[1] و هسته. این نکته ممکن است به نظر عجیب برسد چرا که هرگاه از سیستم‌عامل صحبت می‌شود از لغت هسته به جای این دو لایه با هم استفاده می‌شود. در حقیقت در این مقاله نیز این اتفاق افتاده است. مجری، واسط فراخوانی‌های سیستمی و قسمت اعظم اجزای سیستم‌عامل را پیاده‌سازی می‌کند، که این اجزا شامل مدیر I/O، مدیر حافظه، مدیر پردازه‌ها و نخ ها و دیگر قسمت‌ها می‌باشد. درایورهای دستگاه که در مود هسته هستند، عموما بین HAL و قسمت مدیر I/O در مجری هستند. هسته، روتین‌های سطح پایین مثل زمان‌بندی نخ‌ها، همزمان‌سازی و سرویس‌های وقفه را پیاده‌سازی می‌کند. این روتین‌ها به منظور استفاده‌ی سرویس‌های سطح بالا توسط مجری ساخته می‌شود. مشابه HAL براساس ویژگی‌های سیستم نسخه‌های متفاوتی برای هسته/مجری وجود دارد که در جدول ۲ مشاهده می‌کنید.

جدول ۲- نسخه های هسته

PAE	Kernel File on Live System	Kernel File on Install DVD
Supported	Ntkrnlpa.exe	Ntkrpamp.exe
Not supported	Ntoskrnl.exe	Ntkrnlmp.exe

در خروجی شکل ۹ می‌توانید مشاهده کنید که نام فایل باینری هسته/مجری را با نام ntkrnlmp.exe نمایش داده است. اگر به جدول ۲ خوب نگاه کنید متوجه می‌شوید که هر دو فایل باینری هسته/مجری با پیشوند nt آغاز می‌شود. این پیشوند به ویندوز Nt یعنی پدر بزرگ ویندوز ۷ اشاره دارد. این پیشوند در اکثر نام‌ها، فراخوانی‌های سیستمی، پیغام‌های خطا و فایل‌های سرآیند وجود دارد.

فایل win32k.sys یکی دیگر از بازیگران مهم در قسمت هسته است. این درایور مود هسته دو بخش USER و سرویس‌های GDI[2] را پیاده‌سازی می‌کند. یک برنامه کاربر روتین‌های USER را فراخوانی می‌کند تا کنترل‌های GUI ساخته شود. GDI به منظور منتقل کردن گرافیک به وسیله خروجی استفاده می‌شود. برخلاف سایر سیستم‌های عامل،

[1] Executive
[2] Graphics device interface

ویندوز برای سرعت بخشیدن به اجرا اکثر کدهای گرافیک را در هسته انجام می‌دهد. برای این که متوجه شویم این اجزا چگونه به هم مرتبط می‌شوند می‌توانیم از ابزار dumpbin.exe استفاده کنیم. بدین وسیله می‌توانیم متوجه شویم که هر کدام از اجزا چه روتین‌هایی را از یکدیگر وارد کرده‌اند.

```
C:\windows\system32\> dumpbin.exe /imports hal.dll
```

شکل ۱۱ –استفاده از ابزار dumpbin.exe **برای نمایش** DLL **های وارد شده در** hall.dll

در جدول ۳ می‌توانید تمام این اجزا و فایل‌هایی را که استفاده کرده‌اند ببینید.

جدول ۳ – اجزای مود هسته

Component	Imports
hal*.dll	Nt*.exe, kdcom.dll, pshed.dll
BOOTVID.DLL	Nt*.exe, hal*.dll
Nt*.exe	hal*.dll, pshed.dll, bootvid.dll, kdcom.dll, clfs.sys, ci.dll
Win32k.sys	Nt*.exe, msrpc.sys, watchdog.sys, hal.dll, dxapi.sys

۳-۲-۲- اجزای مود کاربر

یک زیرسیستم محیطی مجموعه‌ای از فایل‌های باینری است که در مود کاربر اجرا می‌شود و به برنامه‌هایی که نوشته شده‌اند تا از قسمتی از زیرسیستم یا توابع API استفاده کنند اجازه می‌دهد که اجرا شوند. با استفاده از زیرسیستم‌ها می‌توان برنامه‌ای را که روی یک سیستم‌عامل دیگر (مثلا OS/2) نوشته شده است بدون تغییر امضا روی یک زیرسیستم دیگر اجرا کرد. برای فهمیدن انگیزه پشت این ایده نیاز داریم که در مورد سیر حافظه بیشتر بدانیم. زمانی که در سال ۱۹۹۶ ویندوز Nt 4.0 وارد بازار شد، پنج زیرسیستم مختلف را حمایت می‌کرد:

- زیرسیستم Win32
- زیرسیستم ویندوز روی ویندوز WOW[1]
- زیرسیستم NTVDM[2]
- زیرسیستم OS/2

[1] Window on window
[2] NT Virtual DOS Machine

- زیرسیستم POSIX

زیرسیستم Win32 از برنامه‌هایی حمایت می‌کند که از توابع Win32 API استفاده می‌کنند. این توابع در واقع یک واسط هستند که توسط کاربران بر روی ویندوزهای ۹۵ و NT استفاده می‌شوند. زیرسیستم WOW یک محیطی را فراهم می‌کند که برنامه‌های ویندوزهای ۱۶ بیتی که برای اجرا در ویندوز ۳.۱ نوشته شده بودند، بتوانند اجرا شوند. زیرسیستم NTVDM یک محیط خط فرمان برای برنامه‌های DOS قانونی را ایجاد می‌کند. زیرسیستم OS/2 برنامه‌هایی را که در سیستم عامل OS/2 برای ماشین‌های IBM نوشته شده بودند، را اجرا می‌کند و در نهایت زیرسیستم POSIX تلاشی برای جلب رضایت توسعه‌دهندگان UNIX بود.

هدف مایکروسافت از ایجاد این زیرسیستم‌های مختلف به هوس انداختن کاربران دیگر سیستم عامل‌ها به سمت ویندوز NT بود. با گذشت سال‌ها زیرسیستم‌های POSIX وOS/2 از رده خارج شدند. به عنوان یک جایگزین برای POSIX، ویندوزهای XP و سرور ۲۰۰۳ یک زیرسیستم به نام SFU[1] (سرویس‌های ویندوز برای UNIX) پیشنهاد دادند. با آمدن ویندوز ویستا این زیر سیستم با نام برنامه‌های مبتنی بر UNIX (SUA[2]) شناخته می‌شد. ویندوز ۷ و سرور ۲۰۰۸ نیز از SUA حمایت می‌کنند. البته امروزه با توجه به رشد ویندوز دیگر چندان اهمیتی به برآورده شدن تقاضاهای دیگر کاربران سیستم‌های عامل مختلف نمی‌شود.

زیرسیستم محیطی اصلی در ویندوز ۷ وسرور ۲۰۰۸ زیرسیستم Windows است. که نسل مستقیم زیرسیستم Win32 محسوب می‌شود که به خاطر وجود سیستم های ۶۴ بیتی مایکروسافت تصمیم گرفت که پسوند ۳۲ را از روی آن بردارد. زیرسیستم Windows شامل سه قسمت اصلی می‌شود.

- Client-Server Runtime SubSystem (Csrss.exe) موجود در مود کاربر
- درایور مود هسته‌ی Win32k.sys
- DLLهای مود کاربر که APIهای زیرسیستم را پیاده‌سازی می‌کنند.

زیرسیستم Client-Server Runtime نقش مدیریت پردازه‌ها و نخ‌ها را برعهده دارد و واسط خط فرمان را نیز به طور عملی حمایت می‌کند و یکی از فایل‌های اجرایی است که به طور دائمی در فضای کاربر است. در صورتی که برنامه Task Manager را باز کنید ، قطعا یک نمونه از Csrss.exe را خواهید دید.

[1] Windows Service for Unix
[2] Subsystem for UNIX-based Application

زیرسیستم Windows یک راه ارتباطی از طریق توابع API برای برنامه‌ها باز گذاشته است. این توابع بسیار مشابه توابع Win32 هستند و به صورت DLL پیاده‌سازی شده‌اند (مانند kernel32.dll , Advapi32.dll , User32.dll Gdi.dll و ...). در صورتی که یک تابع API نتواند به طور کامل در فضای کاربر پیاده‌سازی شود و به سرویس‌های موجود در مجری نیاز داشته باشد، باید کدهای موجود در کتابخانه ntdll.dll را فراخوانی کند تا مسیر برنامه به سمت کد مجری هدایت شود. در واقع توابع موجود در ntdll.dll مانند یک دروازه برای ورود به مجری هستند.

همان طور که در بخش مود هسته توانستیم با ابزار dumpbin.exe اجزا و ارتباط آنها را با هم ببینیم، در قسمت مود کاربر نیز این کار را انجام می‌دهیم، که خروجی آن به طور خلاصه در جدول زیر آمده است.

جدول ٤-اجزای مود کاربر

Component	Imports
Advapi32.dll	msvcrt.dll, ntdll.dll, kernelbase.dll, api*.dll, kernel32.dll, rpcrt4.dll, cryptsp. dll, wintrust.dll, sspicli.dll, user32.dll, bcrypt.dll, pcwum.dll
User32.dll	ntdll.dll, gdi32.dll, kernel32.dll, advapi32.dll, cfgmgr32.dll, msimg32.dll, powrprof.dll, winsta.dll
GDI32.dll	ntdll.dll, api*.dll, kernel32.dll, user32.dll, lpk.dll
Csrss.exe	Nt*.exe, csrsrv.dll
Kernel32.dll	ntdll.dll, kernelbase.dll, api*.dll
Ntdll.dll	None

همان طور که می‌دانید تنها جایی که سرویس‌ها می‌توانند اجرا شوند در مود هسته است، بنابراین وجود سرویس‌ها در قسمت بالای تصویر ۸ یعنی در مود کاربر ممکن است گیج کننده به نظر برسد. برای توضیح این ابهام باید یک مفهوم را توضیح بدهیم و آن هم سرویس‌های مود کاربر هستند. در واقع سرویس‌های مود کاربر، تنها یک برنامه مود کاربر هستند که در پشت صحنه اجرا می‌شوند و با کاربر ارتباطی ندارند و یا به صورت خیلی محدود ارتباط دارند و از طریق یک برنامه مود کاربر دیگر به نام SCM[1] مدیریت و اجرا می‌شوند. برنامه SCM توسط services.exe که در مسیر %systemroot%\system32 یافت می‌شود، اجرا می‌شود. به منظور ساده‌سازی مدیریت در SCM یک سرویس مود کاربر باید از توابع API ای استفاده کند که در Winsvc.h اعلان شده‌اند.

[1] Service Control Manager

٤- نتیجه‌گیری

در این مقاله با فضای هسته و کاربر آشنا شدیم و دانستیم که فضای 4GB ای به دو فضای 2GB تقسیم می‌شود. که این تقسیم‌بندی را می‌توان با تکنیک‌های AWE و یا 4GT تغییر داد. هم چنین با مودهای کاربر و هسته آشنا شدیم و به توصیف ساختار سیستم عامل بر اساس دو مود کاربر و هسته پرداختیم. ساختار هسته شامل قسمت‌های اصلی سخت‌افزار، HAL و هسته/مجری است. همپنین مود کاربر شامل قسمت‌های اصلی زیرسیستم Windows، توابع API و سرویس‌های مود کاربر است.

مراجع

[1] Bill Blunden :The.Rootkit.Arsenal.Escape.and.Evasion.in.the.Dark.Corners.of.the.System, Second Edition, 1969
[2] Address Windowing Extension [Online]. Available : http://msdn.microsoft.com/en-us/library/windows/desktop/aa366527(v=vs.85).aspx
[3] 4-Gigabyte Tuning[Online]. Available : http://msdn.microsoft.com/en-us/library/windows/desktop/bb613473(v=vs.85).aspx
[4] User Mode and Kernel Mode [Online]. Available : http://msdn.microsoft.com/en-us/library/windows/hardware/ff554836(v=vs.85).aspx
[5] Windows Programming: Kernel Mode vs User Mode [Online]. Available : http://en.wikibooks.org/wiki/Windows_Programming/User_Mode_vs_Kernel_Mode