

INTRODUCCIÓN AL REVERSING CON IDA PRO DESDE CERO PARTE 63.

Contents

INTRODUCCIÓN AL REVERSING CON IDA PRO DESDE CERO PARTE 63.	1
SEGMENTACION Y PAGINACION DE MEMORIA PARA ARQUITECTURA x86.....	1
Memoria Real también llamada Física o Memoria RAM.	1
Memoria Virtual.....	2
MODO REAL	3
MODO PROTEGIDO	4
CR3	4

SEGMENTACION Y PAGINACION DE MEMORIA PARA ARQUITECTURA x86

En esta parte del tutorial trataremos de ver el manejo de memoria en x86 y mas adelante trataremos de actualizarlo para 64 bits.

Este es un tema denso, incluye tablas que hay que entender y aprender a ubicar así que iremos despacio.

Veamos algunos conceptos choreados de Internet

Memoria Real también llamada Física o Memoria RAM.

Esta memoria es empleada para el almacenamiento de instrucciones y ejecución de procesos en nuestra computadora, la capacidad de esta memoria es la capacidad real con la que cuenta nuestra máquina para ejecutar nuestros programas.

Memoria Virtual

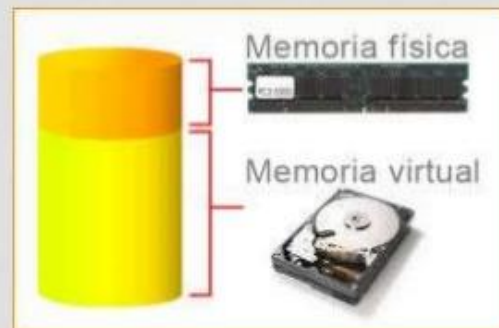
La Memoria Virtual más que una memoria como tal es una técnica utilizada en la computación para simular que se tiene una mayor capacidad de Memoria Real, la técnica consiste en pasar cierta cantidad de archivos al disco duro para almacenarlos allí momentáneamente, en cuanto son requeridos los datos, estos son enviados de regreso a la Memoria Real para su ejecución.

Al aplicar esta técnica se puede obtener una mayor capacidad para la ejecución de instrucciones, sin embargo, esta también conlleva sus desventajas, como podría ser el hecho de que las aplicaciones ejecutadas por Memoria Virtual podrían correr de manera más lenta, esto debido a que el tiempo requerido por el sistema para acceder a los datos es mucho mayor que el tiempo requerido para acceder a los datos almacenados por Memoria Real.

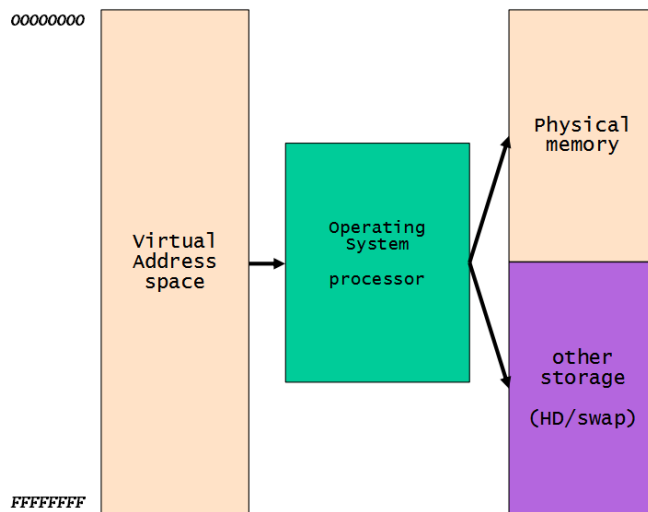
Cuando se utiliza toda la RAM, se usa la memoria virtual, que es usar espacio en el disco para simular más memoria real de la que tenemos.

MEMORIA VIRTUAL

- Los sistemas operativos obviamente no la pueden utilizar, lo que provoca un estancamiento en los procesos del computador.
- Ahí entra a jugar su papel la memoria virtual.
- La memoria virtual es un espacio (SWAP) en el disco duro (HDD) que se usa como si fuera RAM.



Allí vemos en la imagen que el sistema operativo toma la memoria física que tiene, mas la memoria virtual (swap) y en x86 crea para cada proceso un espacio de direcciones virtuales que son las direcciones que vemos en nuestro debugger cuando trabajamos cotidianamente.



Estas direcciones virtuales para cada proceso comienzan en la dirección 0 y terminan en 0xffffffff y en Windows la mitad inferior o sea desde 0x0 hasta 0x7fffffff corresponde a la parte de nivel usuario donde residen los ejecutables y módulos del programa, la parte desde 0x7fffffff hasta 0xffffffff corresponde al kernel.

Por lo tanto, vemos que como cada proceso tiene un espacio de memoria con las mismas direcciones, el tema es que el mismo trabaja con direcciones virtuales, o sea en mi proceso por ejemplo 0x401000 tiene determinado código y en otro proceso la misma dirección virtual 0x401000 tiene un código diferente o otra cosa, y es que ambos corresponden a direcciones físicas o reales diferentes en la memoria.

DIRECCIONES FISICAS Y DIRECCIONES VIRTUALES

Una dirección virtual es lo que ve una aplicación o lo que vemos cuando debuggeamos, una dirección física o real es lo que el hardware real y el sistema operativo ven.

MODULO REAL

Intel, para mantener compatibilidad con versiones anteriores en sus procesadores (excepto IA-64) lo que hace es que en su estado inicial (después del reinicio) el procesador se inicia en un modo llamado modo real, durante el proceso de arranque del sistema operativo, el procesador se cambia al modo protegido.

Se utiliza en el modo real la segmentación, que es un método antiguo de manejo de memoria, allí cada dirección lógica apunta directamente a la memoria física.

Veamos algunos ejemplos (era 286-386):

La arquitectura 286 introdujo 4 segmentos: CS (segmento de código) DS (segmento de datos) SS (segmento de pila) ES (segmento adicional).

```
physical_address : = segment_part × 16 + offset
```

Si tengo la dirección

06EFh:1234h

Para calcular la dirección física multiplico la parte del segmento por 16 (0x10 hexa) y le sumo el offset.

0x6ef*0x10+ 0x1234

```
hex(0x06EF *0x10 + 0x1234)
'0x8124'
```

Y obtengo 0x8124 que es la dirección física, no vamos a ahondar mas en el modo real pues ya no se usa salvo por cuestiones de compatibilidad.

MODO PROTEGIDO

En el modo protegido la parte del segmento es reemplazada por un selector de 16 bits, aunque la translación a dirección física es un poco más compleja.

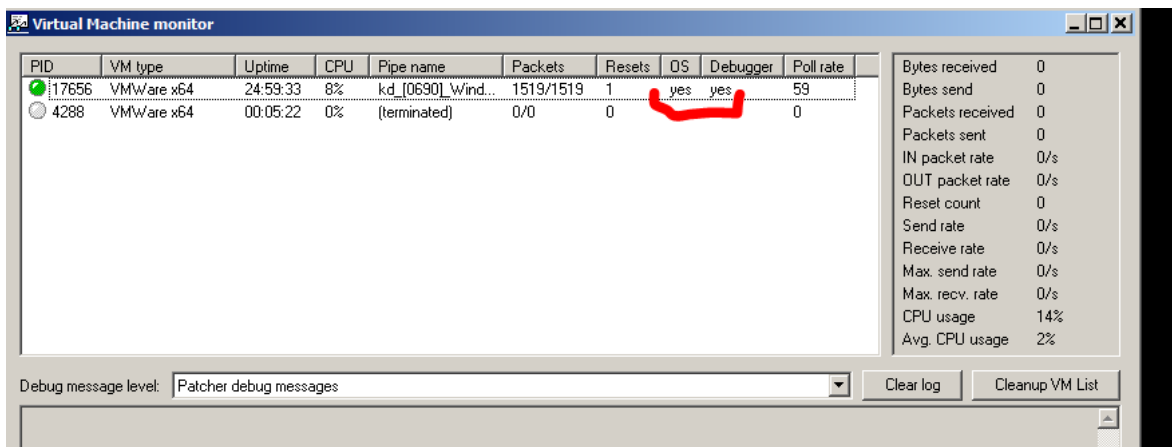
Existen los llamados registros de control que vimos por encima en las partes anteriores como deshabilitar SMEP tocando el cr4, pero en este caso el que nos importa es el CR3.

CR3

Used when [virtual addressing](#) is enabled, hence when the PG bit is set in CR0. CR3 enables the processor to translate linear addresses into physical addresses by locating the page directory and [page tables](#) for the current task.

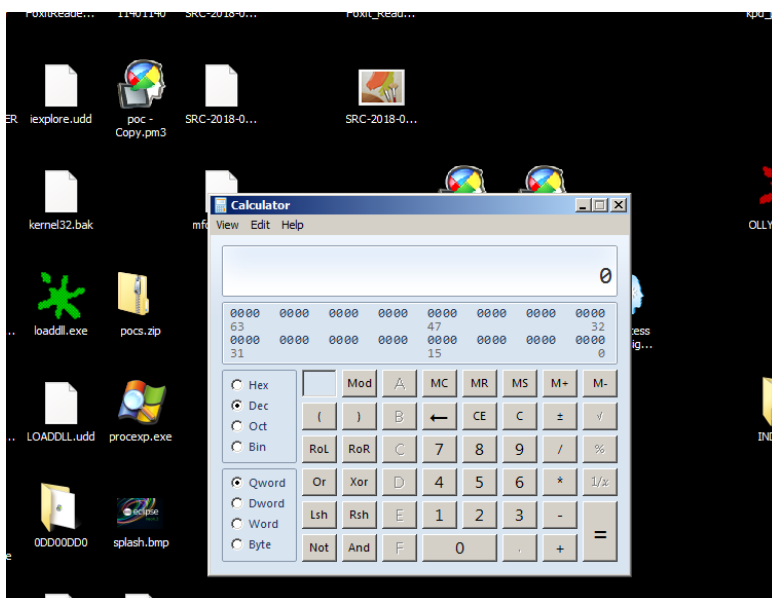
Bueno vemos que se utiliza para convertir direcciones virtuales en físicas, así que tratemos de ver cómo usarlo, obviamente desde un proceso en modo user no podremos tener los privilegios para acceder a leerlo, así que deberemos debuggear el kernel como hemos visto en las partes anteriores.

Vamos a la parte práctica que es la que mas me gusta.



Ahí arranque el sistema operativo y estoy debuggeando por ahora con Windbg.

Voy a arrancar una calculadora que será el proceso donde voy a trabajar.



Ahora en el Windbg hago BREAK para que se detenga.

```
Kernel'compipe,reset=0,reconnect,port=\\.\pipe[kd_0690]_Windows_Seven_Ultimate_-_i386_-_1'- WinDbg:1...
File Edit View Debug Window Help

Command

Microsoft (R) Windows Debugger Version 10.0.15063.468 X86
Copyright (c) Microsoft Corporation. All rights reserved.

Opened \\.\pipe[kd_0690]_Windows_Seven_Ultimate_-_i386_-_1
Waiting to reconnect...
Connected to Windows 7 7601 x86 compatible target at (Thu Apr 26 10:22:39.358 2018 (UTC -
Kernel Debugger connection established.

***** Symbol Path validation summary *****
Response Time (ms) Location
Deferred SRV*c:\symbols*http://msdl.microsoft.com/download/symbols
Symbol search path is: SRV*c:\symbols*http://msdl.microsoft.com/download/symbols
Executable search path is:
Windows 7 Kernel Version 7601 MP (1 procs) Free x86 compatible
Built by: 7601.23796.x86fre.win7sp1_ldr.170427-1518
Machine Name:
Kernel base = 0x82638000 PsLoadedModuleList = 0x82784e30
System Uptime: not available
nt!DbgLoadImageSymbols+0x47:
826506af cc int 3
kd> g
KDBG: Refreshing KD connection
Break instruction exception - code 80000003 (first chance)
*****
* You are seeing this message because you pressed either
* CTRL+C (if you run console kernel debugger) or
* CTRL+BREAK (if you run GUI kernel debugger),
* on your debugger machine's keyboard.
*
* THIS IS NOT A BUG OR A SYSTEM CRASH
*
* If you did not intend to break into the debugger, press the "g" key, then
* press the "Enter" key now. This message might immediately reappear. If it
* does, press "g" and "Enter" again.
*
*****
nt!RtlpBreakWithStatusInstruction:
826b39d8 cc int 3

Ln 0, Col 0 Sys 0:KdSrv:S Proc 000:0 Thrd 001:0 ASM OVR CAPS NUM
```

Cambiamos el contexto al del proceso calc.

!process 0 0

```
Image: \SystemRoot\system32\cmd.exe
PROCESS 85024030 SessionId: 1 Cid: 047c Peb: 7ffd8000 ParentCid: 0ca8
DirBase: bed525a0 ObjectTable: 9e853440 HandleCount: 91.
Image: calc.exe
```

.process /i 85024030

Y luego G

Para saber el proceso actual

!process -1 0

```
nt!RtlpBreakWithStatusInstruction:
326b39d8 cc int 3
l: kd> !process -1 0
PROCESS 85024030 SessionId: 1 Cid: 047c Peb: 7ffd8000 ParentCid: 0ca8
DirBase: bed525a0 ObjectTable: 9e853440 HandleCount: 91.
Image: calc.exe
```

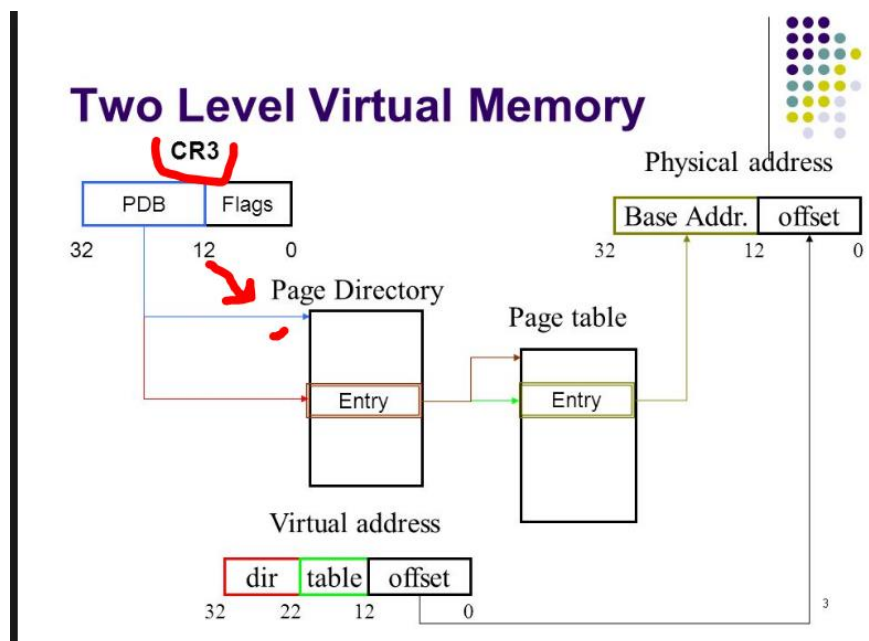
Ya estamos en el proceso calc.

Veamos el valor del registro CR3 con

```
image: calc.exe  
1: kd> r cr3  
cr3=bed525a0
```

Recuerden que este comando funciona solo debuggeando kernel, debuggeando en user mode no funciona.

Bueno CR3 contiene la dirección física de la PAGE DIRECTORY TABLE (ya veremos que esto tiene un agregado si esta habilitado PAE)



Estamos en la parte inicial del dibujo la idea es que CR3 nos apunta a la dirección física de la Page Directory Table (PDT) que es el primer paso, esta es una tablita con entradas cada una de las mismas llamada PAGE DIRECTORY ENTRY (PDE).

Vemos que el comando **!process** en DirBase nos muestra la misma info o sea el mismo valor de CR3.

```

implicit process is now 85024030
1: kd> !process
PROCESS 85024030 SessionId: 1 Cid: 047c Peb: 7ffd8000 ParentCid: 0ca8
  DirBase: bed525a0 ObjectTable: 9e853440 HandleCount: 91.
  VadRoot: 874a44a8 Vads 99 Clone 0 Private 1148. Modified 2. Locked 0.
  DeviceMap 9d8cc860
  Token a48ef178
  ElapsedTime 00:01:40.556
  UserTime 00:00:00.000
  KernelTime 00:00:00.000
  QuotaPoolUsage[PagedPool] 138236
  QuotaPoolUsage[NonPagedPool] 5980
  Working Set Sizes (now,min,max) (2425, 50, 345) (9700KB, 200KB, 1380KB)
  PeakWorkingSetSize 2433
  VirtualSize 72 Mb
  PeakVirtualSize 73 Mb
  PageFaultCount 2688
  MemoryPriority FOREGROUND
  BasePriority 8
  CommitCharge 1210

```

También se puede hallar a partir del EPROCESS

```

1: kd> dtnt!_EPROCESS 85024030
+0x000 Pcb : _KPROCESS
+0x098 ProcessLock : _EX_PUSH_LOCK
+0x0a0 CreateTime : _LARGE_INTEGER 0x01d3dd61`fc2cff90
+0x0a8 ExitTime : _LARGE_INTEGER 0x0
+0x0b0 RundownProtect : _EX_RUNDOWN_REF

```

Como esta allí dentro de KPROCESS y tiene la misma dirección dumpeamos KPROCESS

```

+0x090 VdmIrapcHandler : Ptr32 void
1: kd> dt _KPROCESS 85024030
ntdll!_KPROCESS
+0x000 Header : _DISPATCHER_HEADER
+0x010 ProfileListHead : _LIST_ENTRY [ 0x85024040 - 0x85024040 ]
+0x018 DirectoryTableBase : 0xbed525a0
+0x01c LdtDescriptor : _KGDTENTRY
+0x024 Int21Descriptor : _KIDTENTRY
+0x02c ThreadListHead : _LIST_ENTRY [ 0x85020f28 - 0x84dfdc40 ]
+0x034 ProcessLock : 0
+0x038 Affinity : _KAFFINITY_EX
+0x044 ReadyListHead : _LIST_ENTRY [ 0x85024074 - 0x85024074 ]
+0x04c SwapListEntry : _SINGLE_LIST_ENTRY
+0x050 ActiveProcessors : _KAFFINITY_EX

```

Y allí nos dice que la dirección física es 0xbed525a0 coincide.

Si uso el comando dd para mirar esa dirección de memoria.

```

+0x090 VdmIrapcHandler : (null)
1: kd> dd bed525a0
bed525a0 ?????????? ?????????? ??????????
bed525b0 ?????????? ?????????? ??????????
bed525c0 ?????????? ?????????? ??????????
bed525d0 ?????????? ?????????? ??????????
bed525e0 ?????????? ?????????? ??????????
bed525f0 ?????????? ?????????? ??????????
bed52600 ?????????? ?????????? ??????????
bed52610 ?????????? ?????????? ??????????

```


No hay nada allí debería estar la tabla, el problema es que el comando dd sirve solo para mostrar direcciones virtuales, no las físicas, para ello se usa el comando **!dd** con el signo de admiración delante.

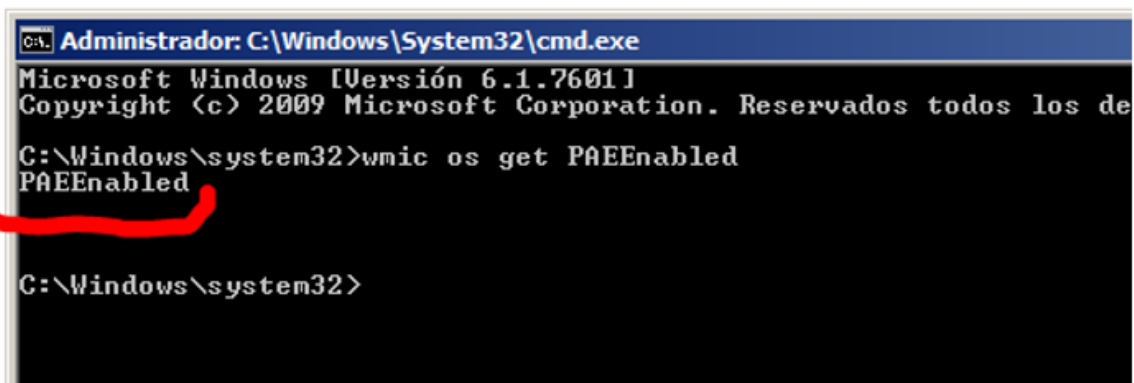
```
bed52610  ?????????? ?????????? ?????????? ??????????  
1: kd> !dd bed525a0  
#bed525a0 1b508801 00000000 1b809801 00000000  
#bed525b0 1b80a801 00000000 1c40b801 00000000  
#bed525c0 293e6801 00000000 298e7801 00000000  
#bed525d0 295e8801 00000000 28ce9801 00000000  
#bed525e0 22f43801 00000000 23144801 00000000  
#bed525f0 23245801 00000000 22946801 00000000  
#bed52600 2905d801 00000000 28d5e801 00000000  
#bed52610 2835f801 00000000 28560801 00000000
```

Ahora si estamos bien.

Vamos a asumir que PAE esta habilitado se puede chequear así y hoy día es casi norma que lo esté, el que tiene dudas de lo que es vea acá

https://es.wikipedia.org/wiki/Extensi%C3%B3n_de_direcci%C3%B3n_f%C3%ADsica

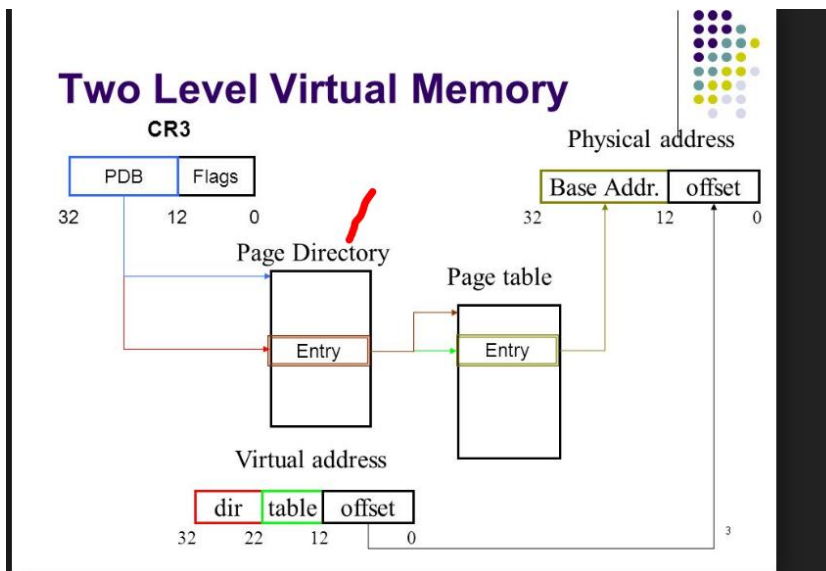
Se chequea así



```
C:\Windows\system32>wmic os get PAEEnabled  
PAEEnabled  
  
C:\Windows\system32>
```

Hay una tablita más (cuando PAE está habilitado) que ahí no se muestra que es PDPT table que es una tabla que cada entrada apunta a una PAGE DIRECTORY es como una tabla intermedia entre CR3 y PAGE DIRECTORY, igual no cambia nada eso.

Mirando la tablita Page Directory y viendo las entradas, ahí mismo nos muestra que cada entrada apunta a otra tabla llamada PAGE TABLE.



Así que cada PAGE DIRECTORY tiene 1024 entradas, donde cada una (llamada PDE), apunta a una PAGE TABLE.(PT)

Si utilizo el comando !dq en el valor de cr3, veo la tabla intermedia PDPT en hay 4 entradas y cada entrada es de 8 bytes y el contenido apunta a la PDT. (aunque eso depende de un detalle que faltaba que explico un poco mas adelante, en este caso toma la primera de las 4 entradas de la PDPT y asi seguimos el tute buscando el contenido de ese primera entrada y dicho contenido es la base de la PDT, pero la PDPT tiene cuatro entradas y puede usar las otras entradas, mas adelante explico en que caso)

```
#bed52610 2835f801 00000000 28560801 00000000
l: kd> !dq bed525a0
#bed525a0 00000000`1b508801 00000000`1b809801
#bed525b0 00000000`1b80a801 00000000`1c40b801
#bed525c0 00000000`293e6801 00000000`298e7801
#bed525d0 00000000`295e8801 00000000`28ce9801
#bed525e0 00000000`22f43801 00000000`23144801
#bed525f0 00000000`23245801 00000000`22946801
#bed52600 00000000`2905d801 00000000`28d5e801
#bed52610 00000000`2835f801 00000000`28560801
```



Así que ya la primera entrada de la PDPT tiene el valor en mi maquina 0x1b508801 con la cual apuntamos a la base de la PT (hay que poner a cero los 12 bits inferiores de esta dirección), luego de eso tenemos la dirección física base de la PT.)

Me queda pasado a binario.

```

pfn 1ad5e      ---DA---UWEV  pfn 1b50e      ----A---UR-V
1: kd> .formats 0x1b508801
Evaluando expresión:
Hex:          1b508801
Decimal:      458262529
Octal:        03324104001
Binary:       00011011 01010000 10001000 00000001
Chars:        .P..
Time:         Mon Jul  9 20:08:49 1984
Float:        low 1.72493e-022 high 0
Double:       2.26412e-315

```

00011011 01010000 1000-100000000001

Ya que los 12 de la parte baja son offset los reemplazo por ceros para que me de la base

00011011010100001000000000000000

Esto pasado a Hexa me da **0x1B508000** que sería la base de la PDT

Ahora que ya obtuvimos la dirección física de la base de la PDT, vamos a hacer **.reload /f** para que cargue todos los módulos.

```

baa2auuu  badiuuu  spta.sys
1: kd> lm
start      end             module_name
00740000  00800000  calc       (pdb symbols)      c:\symbols\calc.pdb\971D2945E99843
02800000  72abc000  oleacc     (pdb symbols)      c:\symbols\oleacc.pdb\67620D076A2E
72f20000  301b0000  WindowsCodecs (pdb symbols)      c:\symbols\WindowsCodecs.pdb\
73110000  73142000  WINMM      (pdb symbols)      c:\symbols\winmm.pdb\7AFD98FCAAD34
73700000  73891000  gdiplus    (pdb symbols)      c:\symbols\MicrosoftWindowsGdiPlus
73bd0000  73c10000  UxTheme    (private pdb symbols) c:\symbols\UxTheme.pdb\5BECAB35E77
73d70000  73f0e000  COMCTL32   (pdb symbols)      c:\symbols\comctl32.pdb\B4CE90AAB9
74690000  74699000  VERSION    (pdb symbols)      c:\symbols\version.pdb\52234E5C7EC
75260000  7526c000  CRYPTBASE  (pdb symbols)      c:\symbols\cryptbase.pdb\59713402
75570000  755bb000  KERNELBASE (pdb symbols)      c:\symbols\kernelbase.pdb\EA8878

```

Luego con **lm** veo que la base del ejecutable calc es **0x740000**, así que ese será el header del ejecutable, veamos.

```

baa2auuu  badiuuu  spta.sys
1: kd> dd 00740000
00740000  00905a4d 00000003 00000004 0000ffff
00740010  000000b8 00000000 00000040 00000000
00740020  00000000 00000000 00000000 00000000
00740030  00000000 00000000 00000000 000000d8
00740040  0eba1f0e cd09b400 4c01b821 685421cd
00740050  70207369 72676f72 63206d61 6f6e6e61
00740060  65622074 6e757220 206e6920 20534f44
00740070  65646f6d 0a0d0d2e 00000024 00000000

```

Es el típico header con el MZ etc

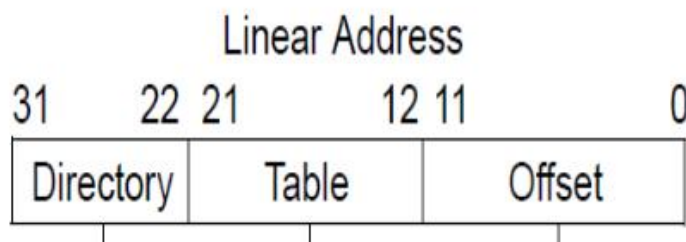
```

1: kd> db 00740000
00740000  4d 5a 90 00 03 00 00 00-04 00 00 00 ff ff 00 00  MZ.....
00740010  b8 00 00 00 00 00 00 00-40 00 00 00 00 00 00 00  .....@.....
00740020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
00740030  00 00 00 00 00 00 00 00-00 00 00 00 d8 00 00 00  .....
00740040  0e 1f ba 0e 00 b4 09 cd-21 b8 01 4c cd 21 54 68  ....!..L.!Th
00740050  69 73 20 70 72 6f 67 72-61 6d 20 63 61 6e 6e 6f  is program canno
00740060  74 20 62 65 20 72 75 6e-20 69 6e 20 44 4f 53 20  t be run in DOS
00740070  6d 6f 64 65 2e 0d 0d 0a-24 00 00 00 00 00 00 00  mode....$.

```

Tomemos esa dirección 0x740000 trataremos de hallar su dirección física

Una dirección virtual como 0x740000 se puede dividir en



Así que la pasamos a binario.

```

00740070  6d 6f 64 65 2e 0d 0d 0a-24 00 00 00 00 00 00 00
1: kd> .formats 00740000
Evaluate expression:
Hex:      00740000
Decimal:  7602176
Octal:    00035000000
Binary:   000000000 01110100 000000000 000000000
Chars:    .t...
Time:     Sun Mar 29 20:42:56 1970
Float:    low 1.06529e-038 high 0
Double:   3.75597e-317

```

00000000 01110100 00000000 00000000

Separando los últimos 12 bytes, luego dos partes de 9 y una de 2 eso nos da 12+9+9+2=32 bits

00-000000011- 101000000 - 000000000000

Bueno los 12 bits mas bajos , son los 12 bits mas bajos de la dirección física o **Byte Index** (000000000000)

Los siguientes 9 bits son un índice a la PAGE TABLE o **Page table index** (101000000)

Los siguientes 9 son un índice a la PAGE DIRECTORY o **Page directory index** (000000011)

Aqui faltaba agregar que esos dos bytes mas altos, marcan que numero de entrada de las cuatro que hay en la PDPT se va a usar, en este caso esos dos bytes son cero, asi que se utiliza la primera entrada de la PDPT por eso no habia problema, sino se usan las que entradas que estan a continuacion segun el numero de ese indice de esos dos bytes)

- PDPT index=0x0
- Page directory index = 000000011 = 0x3
- Page table index = 101000000 = 0x140
- Byte index = 000000000000 = 0x0

Recordemos que 0x1B508000 era la base de la PDT, como cada entrada tiene 8 bytes si quiero encontrar la dirección de la tercera entrada ya que el índice es 0x3.

- Page directory index = 000000011 = 0x3
- Page table index = 101000000 = 0x140
- Byte index = 000000000000 = 0x0

hex(0x1B508000+ 0x3*8)

Obtengo la dirección física de la tercera entrada.

'0x1b508018'

Esta es la PTE sabemos que su contenido apunta a la PT, veamos el contenido

```
Double: 2.0000000000000000
1: kd> !dd 0x1b508018
#1b508018 1ad5e867 00000000 1b049867 00000000
#1b508028 00000000 00000000 00000000 00000000
#1b508038 00000000 00000000 00000000 00000000
#1b508048 00000000 00000000 1b2f1867 00000000
#1b508058 00000000 00000000 1ab23867 00000000
#1b508068 1b94a867 00000000 1bf53847 00000000
#1b508078 1b654867 00000000 1a4e3867 00000000
#1b508088 1b5e8867 00000000 00000000 00000000
```

A esa dirección le debemos quitar los 12 bits inferiores y completarla con ceros para dejar la base

1ad5e867

00011010 11010101 1110**1000 01100111**

La base seria

00011010 11010101 1110**0000 00000000**

Si lo unimos


00011010110101011110**000000000000**

Por lo tanto la base de la PT de mi dirección es

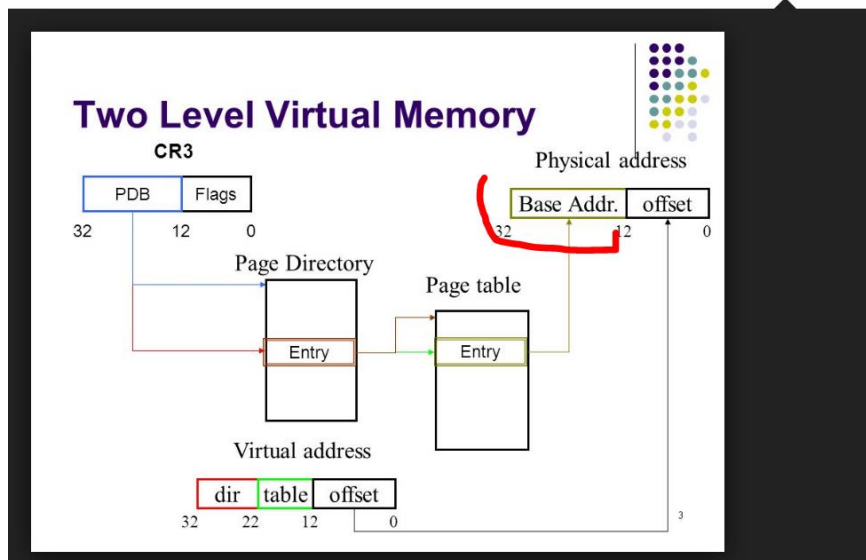
0x1AD5E000

Hago lo mismo que con la otra tabla busco el índice de la PT que en mi caso era 0x140, como cada entrada es de 8 bytes de largo multiplico por 8 y le sumo la base.

- Page directory index = 00000011 = 0x3
- Page table index = 10100000 = 0x140
- Byte index = 000000000000 = 0x0


: hex(0x1AD5E000+ 0x140*8)
]: '0x1ad5ea00'

Eso me da '0x1ad5ea00' que es la PTE, el contenido de la misma es la Base Address



```
Double: 2.22441e-315
: kd> !dd 0xad5ea00
1ad5ea00 1b50e025 00000000 1bdb9025 00000000
1ad5ea10 1b7b3005 00000000 1b7b3005 00000000
1ad5ea20 1b1b4005 00000000 1b5b5005 00000000
1ad5ea30 1b5b6005 00000000 1acb7005 00000000
1ad5ea40 1bab8005 00000000 1b31e005 00000000
1ad5ea50 1a114005 00000000 1ab15005 00000000
1ad5ea60 1b016005 00000000 1b417005 00000000
1ad5ea70 00000000 00000000 00000000 00000000
```

Allí vemos en el cuadrito que la dirección física esta compuesta por base address + offset

```
#1ad5ea70 00000000 00000000 00000000 00000000
1: kd> .formats 1b50e025
Evaluate expression:
Hex: 1b50e025
Decimal: 458285093
Octal: 03324160045
Binary: 00011011 01010000 11100000 00100101
Chars: .P.%
Time: Tue Jul 10 02:24:53 1984
Float: low 1.72778e-022 high 0
Double: 2.26423e-315
```

Ahora debemos reemplazar los 12 bytes menores por el Byte Index de la dirección buscada.

- Page directory index = 000000011 = 0x3
- Page table index = 10100000 = 0x140
- Byte index = 000000000000 = 0x0

00011011 01010000 11100000 00100101

Queda

000110110101000011100000 00000000

Que es

0x1B50E000 que es la dirección física o real correspondiente a la dirección virtual 0x740000

```
t1b50e070 65646f6d 0a0d0d2e 00000024 00000000
.: kd> !db 1B50E000
t1b50e000 4d 5a 90 00 03 00 00 00-04 00 00 00 ff ff 00 00 MZ.....
t1b50e010 b8 00 00 00 00 00 00 00-40 00 00 00 00 00 00 .....@.....
t1b50e020 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
t1b50e030 00 00 00 00 00 00 00 00 00-00 00 00 00 d8 00 00 00 .....
t1b50e040 0e 1f ba 0e 00 b4 09 cd-21 b8 01 4c cd 21 54 68 .....!..L.!Th
t1b50e050 69 73 20 70 72 6f 67 72-61 6d 20 63 61 6e 6e 6f is program canno
t1b50e060 74 20 62 65 20 72 75 6e-20 69 6e 20 44 4f 53 20 t be run in DOS
t1b50e070 6d 6f 64 65 2e 0d 0d 0a-24 00 00 00 00 00 00 00 mode....$......
```

Si comparo con el contenido de la dirección virtual

