

קישור, טעינה ומבנה קובץ הרצה



תרגול 2 - הנדסה לאחור - חורף תש"פא
©אלי ביהם, אביעד כרמל, נערך ע"י טל שנקר ועידן רז



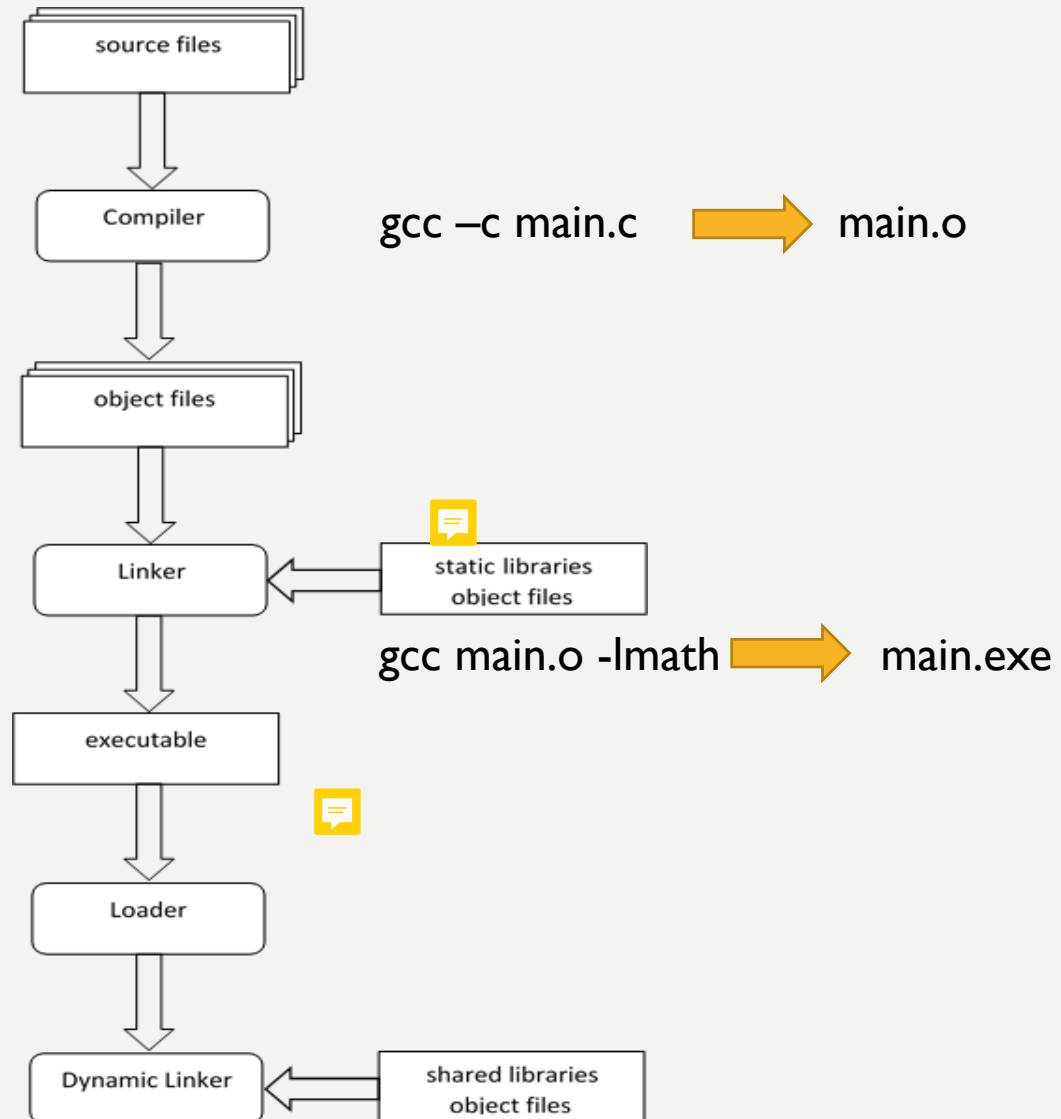
תכנית היום

- סקירה של התהליך: מקבצי קוד C, לקובץ הרצה, ולבסוף לתהליך אשר מריץ את הקוד



- למה זה חשוב לנו? כדי לעשות RE נרצה:
 - לנצל מידע מכל אחד מהשלבים
 - לשנות תוכן של קובץ ההרצה/התהליך הרץ


מקוד לקובץ הרצה



מקובץ עם קוד ב C אל OBJECT FILE

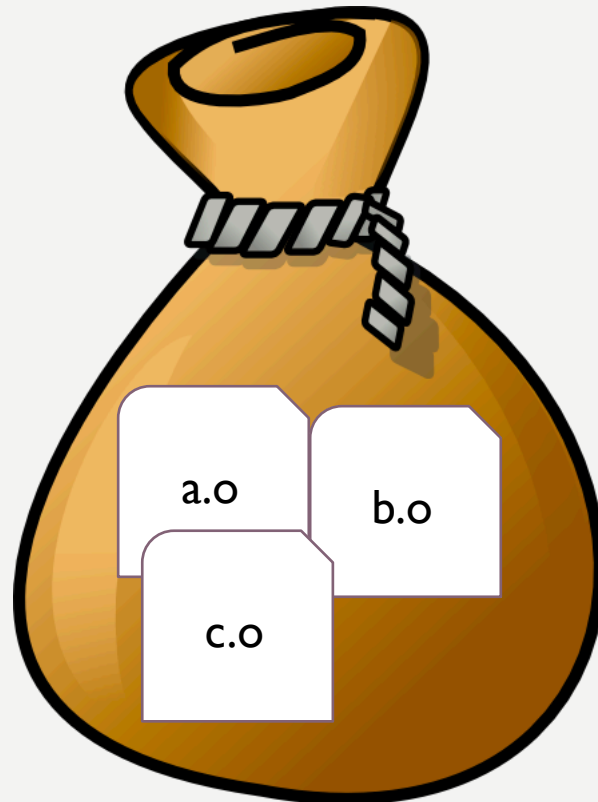
- נרצה לקמפל מספר קבצי C
- כל קובץ מקומפל בנפרד
- לכל קובץ נוצר object file אשר מכיל:
 - object code – קוד אסמבלי "עם חורים"
 - מידע בעבור קישור הקובץ לקבצים אחרים
 - מידע נוסף לא רלוונטי עבורנו

קישור מספר OBJECT FILES

- נרצה לקחת את ה object files שנוצרו בשקף הקודם ולאחדם לקובץ הרצה יחיד.
- בשלב זה, ניתן לפתור תלויות בין symbols בקבצים שונים (לדוגמה, קובץ עם קוד שקורא לפונקציה אשר ממומשת בקובץ אחר).
- מה עם פונקציות ספרייה? לדוגמה, printf? איפה המימוש שלהם? 




מה היא ספריה?

- בגדול: פשוט אוסף של כמה object files
- בדיוק רב יותר: לפעמים מכילה מידע שולי נוסף – לדוגמה, בהינתן שם של פונקציה, באיזה object file בספריה מופיע מימוש הפונקציה (אם בכלל)



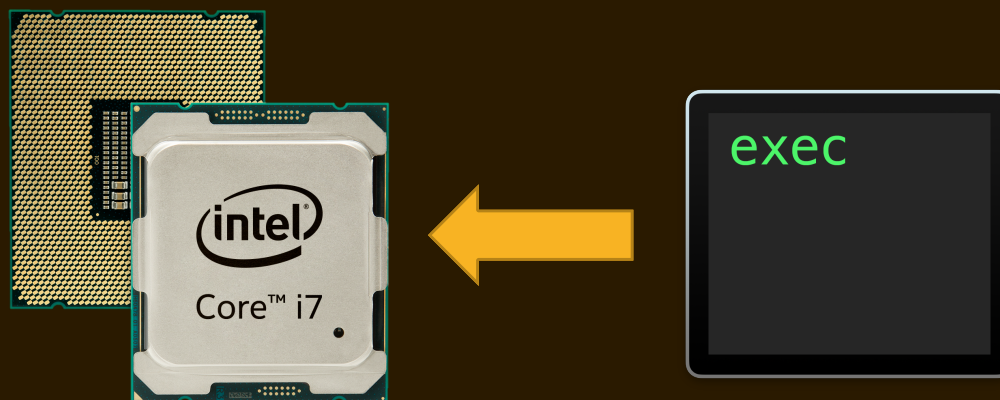
קישור עם ספריות

• ישנם מספר דרכים לקשר object files עם ספרייה:

1. קישור סטטי – להעתיק את ה-object files הרלוונטיים מתוך הספרייה ישירות לקובץ ההרצה. 
2. קישור דינאמי בזמן טעינה – "לסמן" בקובץ ההרצה "לפני הרצת התוכנה, יש לטעון את הספרייה...." (בהמשך נראה איך זה קורא בפועל). 
3. קישור דינאמי בזמן ריצה – קריאות מערכת הפעלה אשר טוענות ספרייה בזמן ריצה (LoadLibraryEx לדוגמה – יהיה שימושי בעוד מספר שיעורים). 
4. קישור דינאמי Delayed – "מסמנים" בקובץ ההרצה כי יש צורך בספרייה מסוימת. היא תטען בזמן ריצה אך ורק אם יהיה בא שימוש.



מקובץ הרצה לתהליך שרץ



מבנה קובץ ההרצה - מוטיבציה

- קובץ הרצה (סיומת EXE/DLL ב-Windows) מכיל מידע נוסף מעבר לשפת מכונה, למשל:

- מידע לזמן טעינת התכנית
- מידע על פונקציות חיצוניות שמיובאות מספריות מערכת (DLL)
- מידע ל Debugger - שמות משתנים, וכו'
- מידע שהשאיר הקומפיילר לצרכים אחרים

- מידע זה שימושי ביותר ל-RE
 - החל מהבנת מבנה הקריאות (מי קורא למי ולאיזה משתנים)
 - דרך קבלת רמזים משמות הפונקציות והמשתנים
 - אילו ספריות מערכת בשימוש



מוטיבציה - המשך

- ביצוע שינויים בקובץ הרצה
 - דורשים הכרת מבנה קובץ הרצה
 - למשל, וירוסים משתילים קוד עוין בקבצי הרצה קיימים
- בנוסף, פעולת ה-loader הטוען קובץ הרצה לזיכרון, משפיעה על מבנה הנתונים בזיכרון אותו רואים בזמן ריצה




מדוע צריך מבנה לקובץ הרצה?

- מדוע לא ניתן להסתפק בקוד אסמבלי בלבד בתוך EXE?
 - למשל קובץ .com מכיל אסמבלי טהור
- סוגי הנתונים הבאים (ועוד) נמצאים בקבצי הרצה בנוסף לקוד התוכנית:
 - איך להפנות לקוד שנמצא בספריות מערכת - עלינו לדווח ל-loader כיצד למצוא אותו (כלומר שם הפונקציה ובאיזה DLL היא נמצאת)
 - מה קורה אם הקוד נטען במקום לא ידוע מראש בזיכרון? עלינו לספק טבלת relocation עם פירוט איפה וכיצד לתקן את הקוד (בדומה ל-reloc ב-PDP)



מדוע צריך מבנה לקובץ הרצה?

- איך נגדיר הרשאות לדפי זיכרון?
 - נרצה שמשתנים קבועים וקוד התוכנית יהיו במצב read-only בעוד שגלובליים רגילים יהיו ניתנים לכתיבה.
 - מה יקרה בקוד הבא: `char * a = "abc"; a[0]=0;` 
- אנו נתמקד בקבצי הרצה של חלונות – מבנה PE
 - בלינוקס הפורמט דומה ונקרא ELF

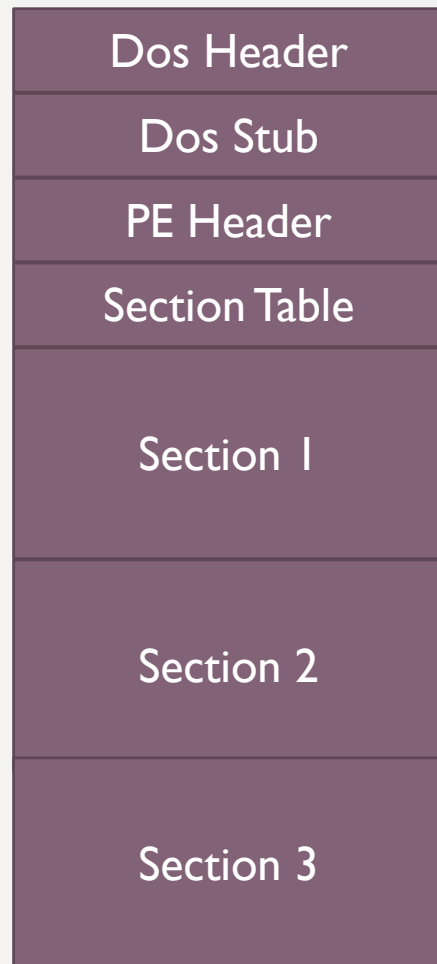
EXE/DLL file:

Dos Header
Dos Stub
PE Header
Section Table
Section 1
Section 2
Section ..

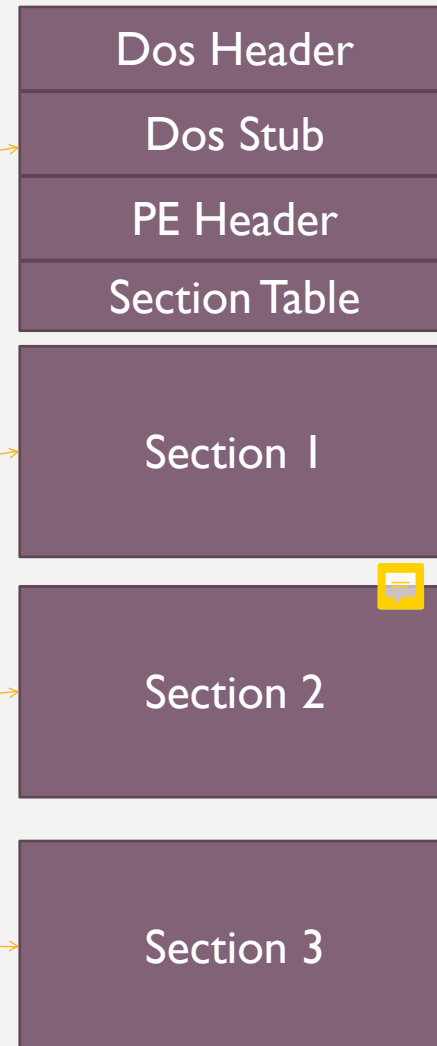
סקירת מבנה PE

מיקום החלקים בזיכרון

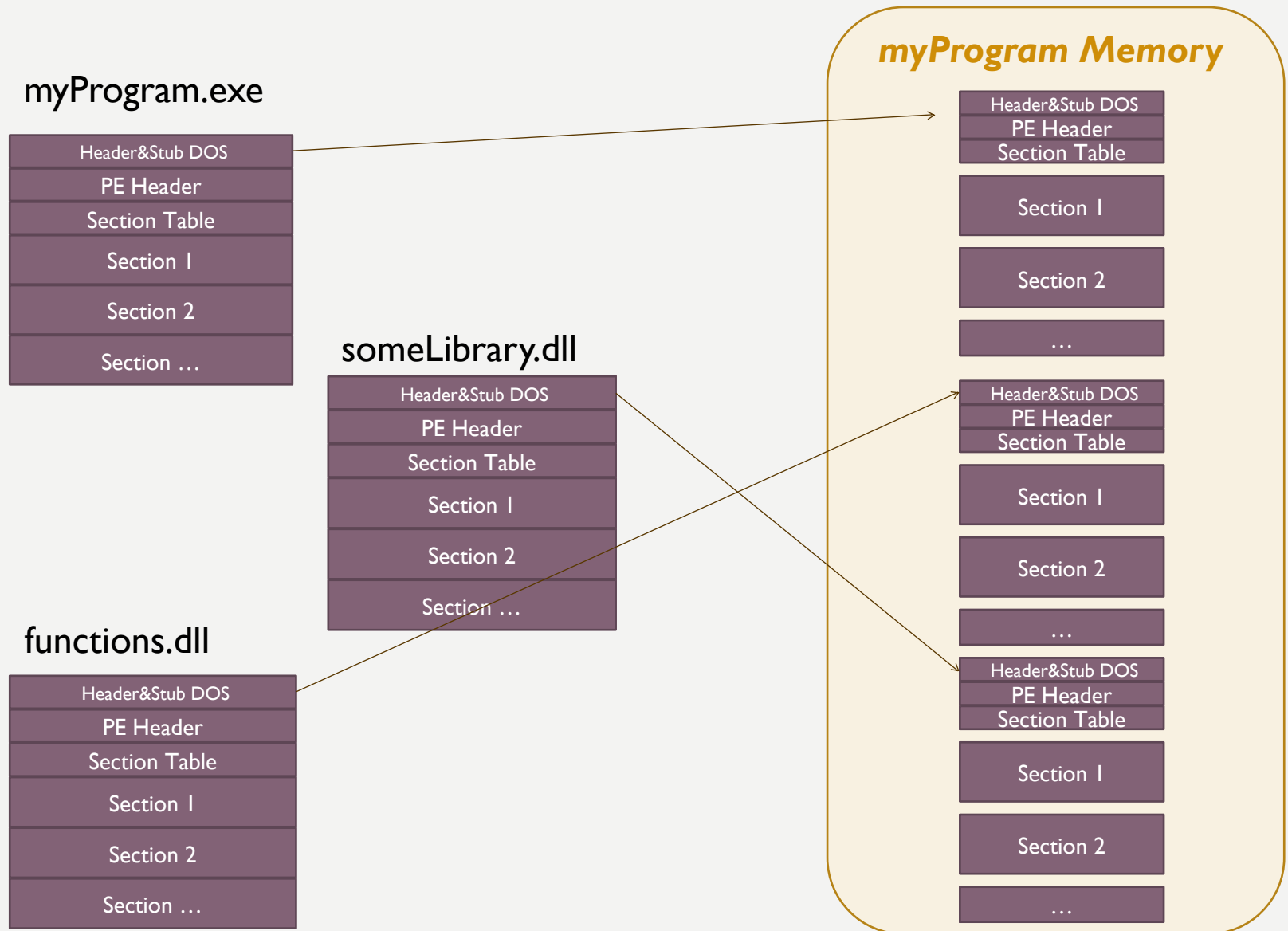
מבנה הקובץ



מיקום בזיכרון



מיקום החלקים בזיכרון



DOS HEADER & STUB

Dos Header
Dos Stub
PE Header
Section Table

- מתחיל ב MZ (כל קובץ EXE מתחיל ב MZ , תבדקו...)
 - מ"ה (או מעכשיו ה Loader) מדלגת על החלק הזה וישר מתחילה מה- PE Header.
 - החלק מכיל קוד אסמבלי 16 ביט. במידה ולא מריצים את הקובץ דרך מ"ה (למשל ב DOS) , אז הקוד ירוץ.
- מדובר בקובץ הרצה DOS תקני.

4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ	E.
B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	7	@	...
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	00
0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68
69	73	20	70	72	6F	67	72	61	60	20	63	61	6E	6E	6F	is	program	canno		
74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t	be	run	in	DOS
6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode
50	45	00	00	4C	01	05	00	32	68	95	50	00	00	00	00	PE	..	L0	\$.2k	0P....

כותרת PE

```
typedef struct _IMAGE_OPTIONAL_HEADER {
    WORD        Magic;
    BYTE        MajorLinkerVersion;
    BYTE        MinorLinkerVersion;
    DWORD       SizeOfCode;
    DWORD       SizeOfInitializedData;
    DWORD       SizeOfUninitializedData;
    DWORD       AddressOfEntryPoint;
    DWORD       BaseOfCode;
    DWORD       BaseOfData;
    DWORD       ImageBase;
    DWORD       SectionAlignment;
    DWORD       FileAlignment;
    ...
    DWORD       SizeOfImage;
    DWORD       SizeOfHeaders;
    DWORD       CheckSum;
    WORD        Subsystem;
    WORD        DllCharacteristics;
    ...
    DWORD       SizeOfHeapCommit;
    DWORD       LoaderFlags;
    DWORD       NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY
    DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER;
```

Dos Header
Dos Stub
PE Header
Section Table

IMAGE_NT_HEADERS STRUCT :
Signature // "PE"
FileHeader **IMAGE_FILE_HEADER**
OptionalHeader **IMAGE_OPTIONAL_HEADER32**

IMAGE_FILE_HEADER STRUCT :
 Machine WORD
 NumberOfSections WORD
 TimeDateStamp DWORD
 PointerToSymbolTable DWORD
 NumberOfSymbols DWORD
 SizeOfOptionalHeader WORD
 Characteristics WORD

OPTIONAL HEADER

פירוט

```
typedef struct _IMAGE_OPTIONAL_HEADER {
```

```
    WORD        Magic;  
    BYTE        MajorLinkerVersion;  
    BYTE        MinorLinkerVersion;  
    DWORD       SizeOfCode;  
    DWORD       SizeOfInitializedData;  
    DWORD       SizeOfUninitializedData;  
    DWORD       AddressOfEntryPoint;  
    DWORD       BaseOfCode;  
    DWORD       BaseOfData;  
    DWORD       ImageBase;  
    DWORD       SectionAlignment;  
    DWORD       FileAlignment;  
    ...  
    DWORD       SizeOfImage;  
    DWORD       SizeOfHeaders;  
    DWORD       CheckSum;  
    WORD        Subsystem;  
    WORD        DllCharacteristics;
```

```
    ...
```

```
    DWORD       SizeOfHeapCommit;  
    DWORD       LoaderFlags;  
    DWORD       NumberOfRvaAndSizes;  
    IMAGE_DATA_DIRECTORY  
    DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];  
} IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER;
```

- AddressOfEntryPoint - מהי הכתובת הראשונה להרצה.

- ImageBase - לאיפה לטעון את התוכנית (לא מובטח שהתוכנית תטען לכתובת הזאת).



- DataDirectory - בשקף הבא.

DATA DIRECTORY

0	Export symbols
1	Import symbols
2	Resources
3	Exception Section
4	Security
5	Base relocation
6	Debug
7	Copyright string
8	?
9	Thread local storage (TLS)
10	Load configuration
11	Bound Import
12	Import Address Table
13	Delay Import
14	COM descriptor



Array of :

```
struct _IMAGE_DATA_DIRECTORY {  
    DWORD VirtualAddress; // rva  
    DWORD Size;  
} IMAGE_DATA_DIRECTORY
```

Dos Header
Dos Stub
PE Header
Section Table

- **RVA** – *relative virtual address*
תזכורת - זו כתובת יחסית של החל מה
.image base
- כל כתובת כזאת יכולה להצביע ל section
אחר. בדרך כלל רוב הרשומות לא קיימות
(כלומר המצביע הוא null).



טבלת ה SECTIONS

Array of:

IMAGE_SECTION_HEADER STRUCT

Name db 8

VirtualSize DWORD

VirtualAddress DWORD

SizeOfRawData DWORD

PointerToRawData DWORD

PointerToRelocations DWORD

PointerToLinenumbers DWORD

NumberOfRelocations WORD

NumberOfLinenumbers WORD

Characteristics DWORD

- Name - שם ה section
 - .text בדרך כלל מכיל קוד.
 - .data משתנים.

- VirtualSize - הגודל הנדרש ל section בזיכרון.

- Characteristics - הרשאות (Execute,read,write)

טבלת ה SECTIONS

Array of:

IMAGE_SECTION_HEADER STRUCT

Name db 8

VirtualSize DWORD

VirtualAddress DWORD

SizeOfRawData DWORD

PointerToRawData DWORD

PointerToRelocations DWORD


PointerToLinenumbers DWORD

NumberOfRelocations WORD

NumberOfLinenumbers WORD

Characteristics DWORD

- VirtualAddress - RVA לכתובת
- שבו ה section צריך להימצא בזיכרון.

▪  **RVA** – *relative virtual address*

▪ זו כתובת יחסית החל מהמקום בו התוכנית נטענה בזיכרון.

- PointerToRawData - offset מתחילת הקובץ ל-section.

- SizeOfRawData - גודל ה section בקובץ. 

טבלת ה SECTIONS

Array of:

IMAGE_SECTION_HEADER STRUCT

Name db 8

VirtualSize DWORD

VirtualAddress DWORD

SizeOfRawData DWORD

PointerToRawData DWORD

PointerToRelocations DWORD

PointerToLinenumbers DWORD

NumberOfRelocations WORD

NumberOfLinenumbers WORD

Characteristics DWORD

- $\text{VirtualSize} - \text{SizeOfRawData}$ לא בהכרח זהים.

- לעיתים SizeOfRawData יהיה גדול מ- VirtualSize , זה קורה מכיוון שהגודל של ה-Section בקובץ עשוי להיות מיושר לגודל מסוים, ולכן בסוף כל section יהיו בתים נוספים לטובת היישור (בדרך כלל אפסים).
- שימו לב ש- VirtualSize יכול גם להיות גדול מ- SizeOfRawData . למשל אם רוצים להקצות משתנים לא מאותחלים, שלא דורשים מקום בקובץ.

0	Export symbols
1	Import symbols
2	Resources
3	Exception Section
4	Security
5	Base relocation
6	Debug
7	Copyright string
8	?
9	Thread local storage (TLS)
10	Load configuration
11	Bound Import
12	Import Address Table
13	Delay Import
14	COM descriptor

טבלת EXPORT

מיקום פונקציות בתכנית

מבנה

```
struct _IMAGE_EXPORT_DIRECTORY {  
    DWORD Characteristics;  
    DWORD TimeDateStamp;  
    WORD MajorVersion;  
    WORD MinorVersion;  
    DWORD Name;  
    DWORD Base;  
    DWORD NumberOfFunctions;  
    DWORD NumberOfNames;  
    DWORD AddressOfFunctions;  
    DWORD AddressOfNames;  
    DWORD AddressOfNameOrdinals;  
};
```

→ "kernel32.dll"



- ה Export Section מכיל את כל המידע לגבי הפונקציות שהתוכנית מייצאת.
- בדרך כלל המידע יהיה ריק עבור קבצי EXE.
- ניתן לעשות export (או import) לכל פונקציה לפי מספר (ordinal) או לפי שם.




• לכל פונקציה יש מספר, אך לא תמיד יש שם.

רשימת הכתובות

AddressOfFunctions (rva)

0	Address of Function 1000
1	Address of Function 1001
.	Address of Function 1002
.	Address
.	Address
	Address
N	Address

- המערך AddressOfFunctions מכיל כתובות לפונקציות. 
- אינדקס במערך זה, בתוספת Base, הוא המספר של הפונקציה (ordinal).
- תפקיד ה-Base הוא לומר מה המספר של הפונקציה הראשונה במערך.
- בדוגמא משמאל ה-Base יחזיק את הערך 1000, וכך נניח גם בשאר השקפים.

```
DWORD Name;  
DWORD Base;  
DWORD AddressOfFunctions;  
DWORD AddressOfNames;  
DWORD AddressOfNameOrdinals;
```

גישה לפי מספר

AddressOfFunctions (rva)

0	Address of Function 1000
1	Address of Function 1001
.	Address of Function 1002
.	Address
.	Address
	Address
N	Address

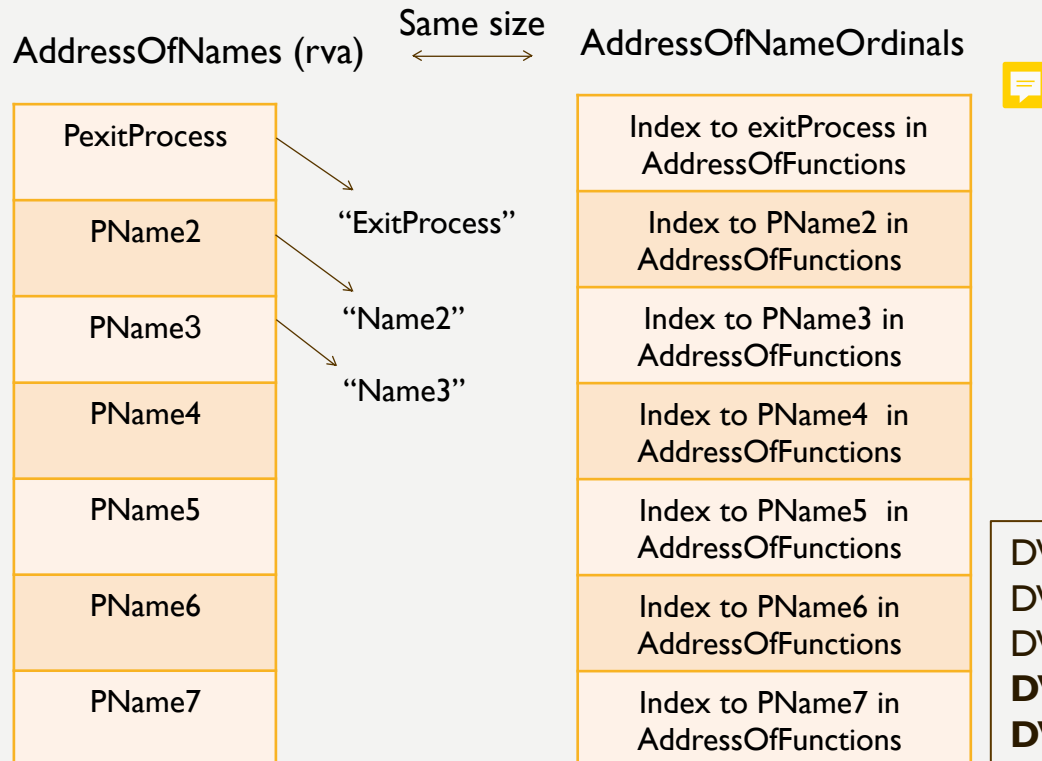
עבור טעינה לפי מספר, ה-Loader מבצע את השלבים הבאים:

1. הפחתת Base מהמספר.
2. התוצאה היא אינדקס ל AddressOfFunctions.

```
DWORD Name;  
DWORD Base;  
DWORD AddressOfFunctions;  
DWORD AddressOfNames;  
DWORD AddressOfNameOrdinals;
```

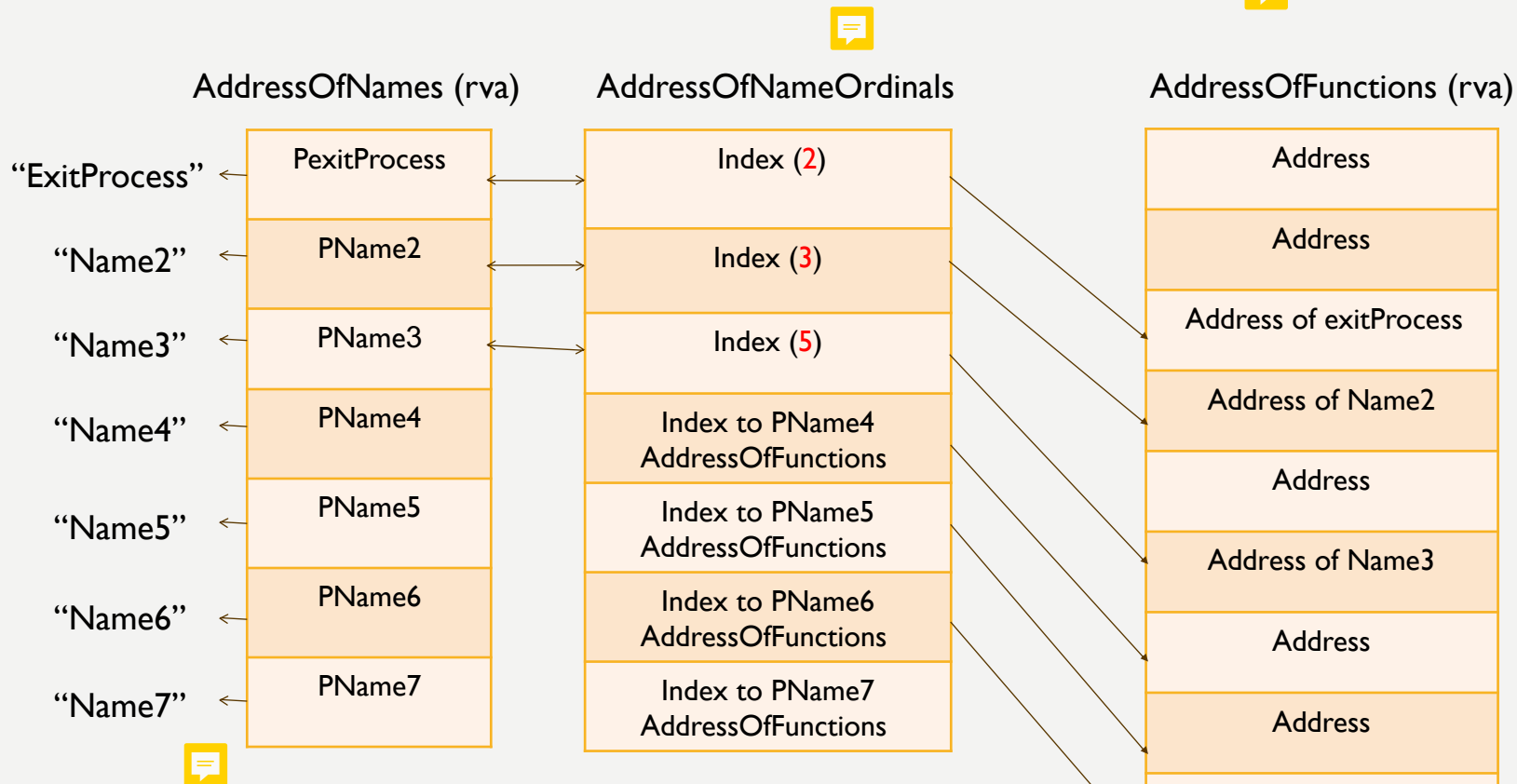
גישה לפי שם

- במידה ונרצה לקבל כתובת של פונקציה לפי שם, ה-Loader מבצע את השלבים הבאים:
1. יש לחפש במערך AddressOfNames את השם של הפונקציה המבוקשת. נשמור את האינדקס.
 2. עם האינדקס ניגש למערך המקביל AddressOfNameOrdinals ונקבל מספר.
 3. מספר זה הוא האינדקס ל AddressOfFunctions כפי שראינו בשקף הקודם. (כלומר האינדקס הוא מספר הפונקציה פחות base)



DWORD Name;
DWORD Base;
DWORD AddressOfFunctions;
DWORD AddressOfNames;
DWORD AddressOfNameOrdinals;

התמונה המלאה



DWORD Name;
DWORD Base;
DWORD AddressOfFunctions;
DWORD AddressOfNames;
DWORD AddressOfNameOrdinals;

0	Export symbols
1	Import symbols
2	Resources
3	Exception Section
4	Security
5	Base relocation
6	Debug
7	Copyright string
8	?
9	Thread local storage (TLS)
10	Load configuration
11	Bound Import
12	Import Address Table
13	Delay Import
14	COM descriptor



טבלת IMPORT

מיקום פונקציות
בספריות מערכת

מבנה

Array of structs (null terminated):

IMAGE_IMPORT_DESCRIPTOR STRUCT

OriginalFirstThunk DWORD

TimeStamp DWORD

ForwarderChain DWORD

Name DWORD

FirstThunk DWORD

→ Pointer to a DLL name. e.g., "kernel32.dll".

→ *OriginalFirstThunk* And *FirstThunk* are pointers to:

Array of:

IMAGE_THUNK_DATA UNION

Ordinal DD

Name DD

ניתן לטעון פונקציה לפי מספר (ordinal) או לפי שם.
ה-loader בודק את ה-MSB של ה-union ואם הוא דולק סימן
שזה מספר, אחרת כתובת למחרוזת (שם הפונקציה).

תוכן בקובץ PE

Array of structs:

IMAGE_IMPORT_DESCRIPTOR STRUCT

OriginalFirstThunk DWORD

TimeDateStamp DWORD

ForwarderChain DWORD

Name DWORD

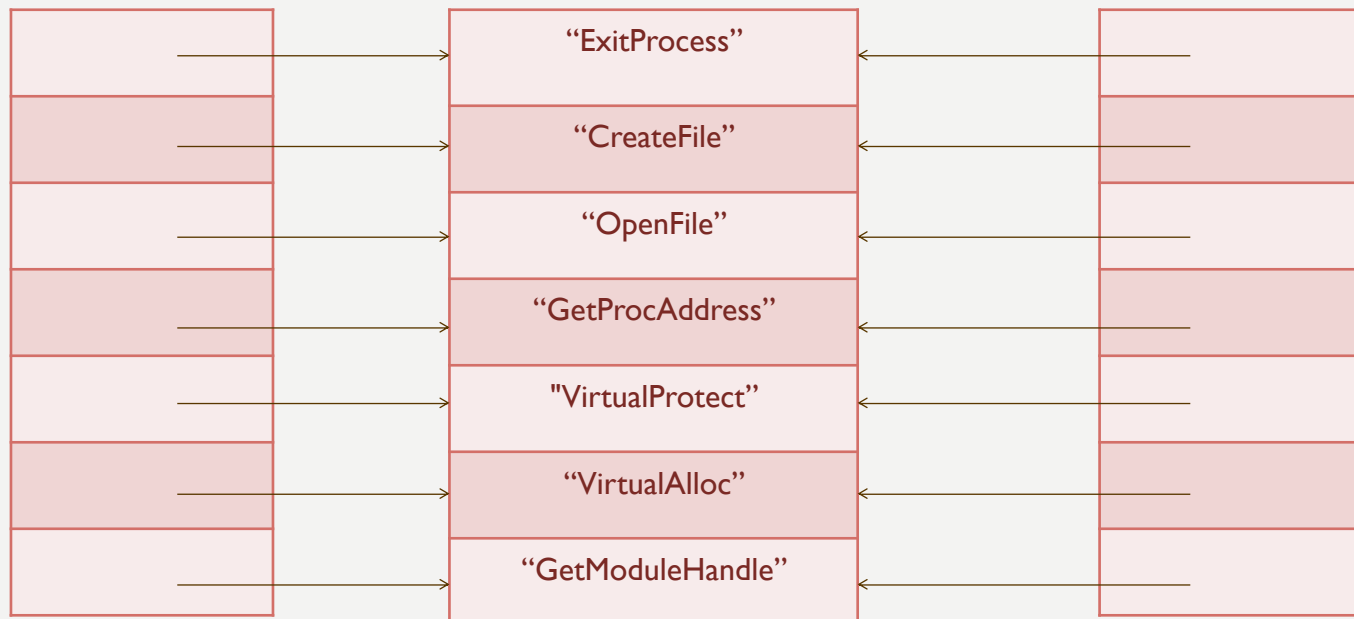
FirstThunk DWORD

→ "Kernel32.dll"

OriginalFirstThunk

קובץ/מחרוזות בזכרון

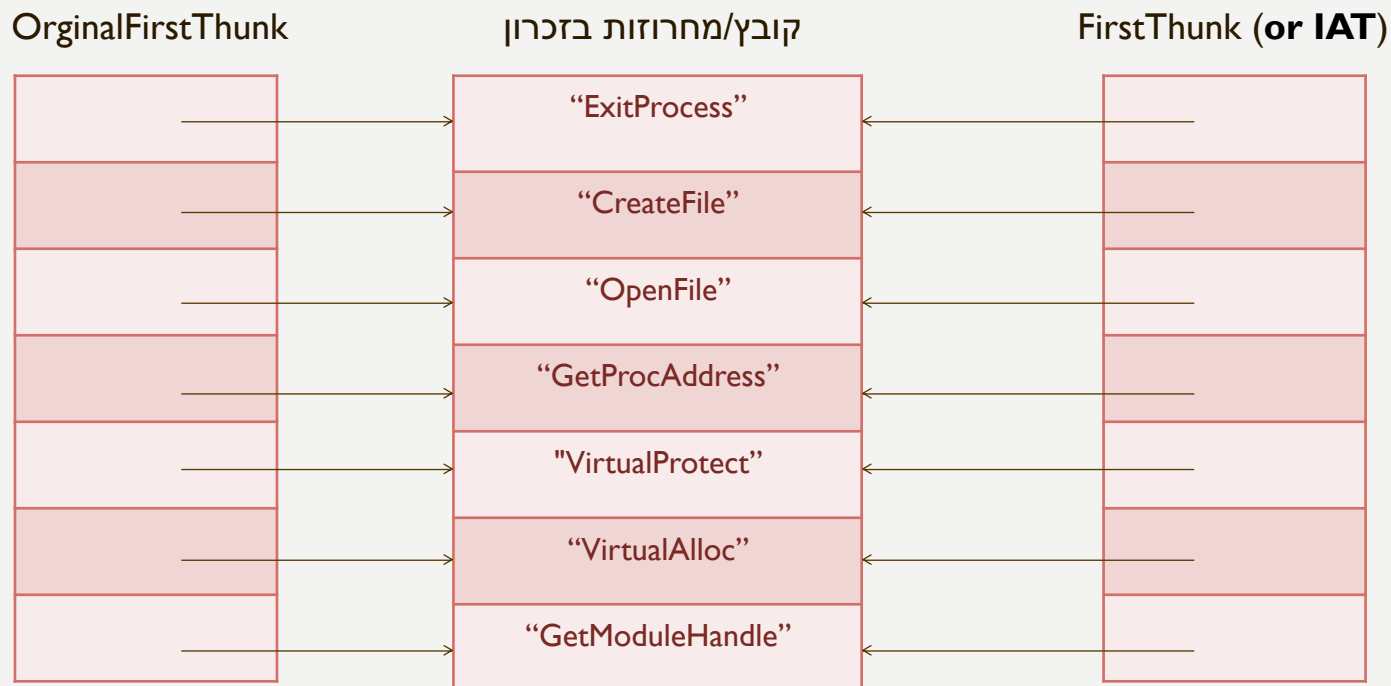
FirstThunk (or IAT)



עדכון בזמן טעינה

ה Loader מבצע את השלבים הבאים:

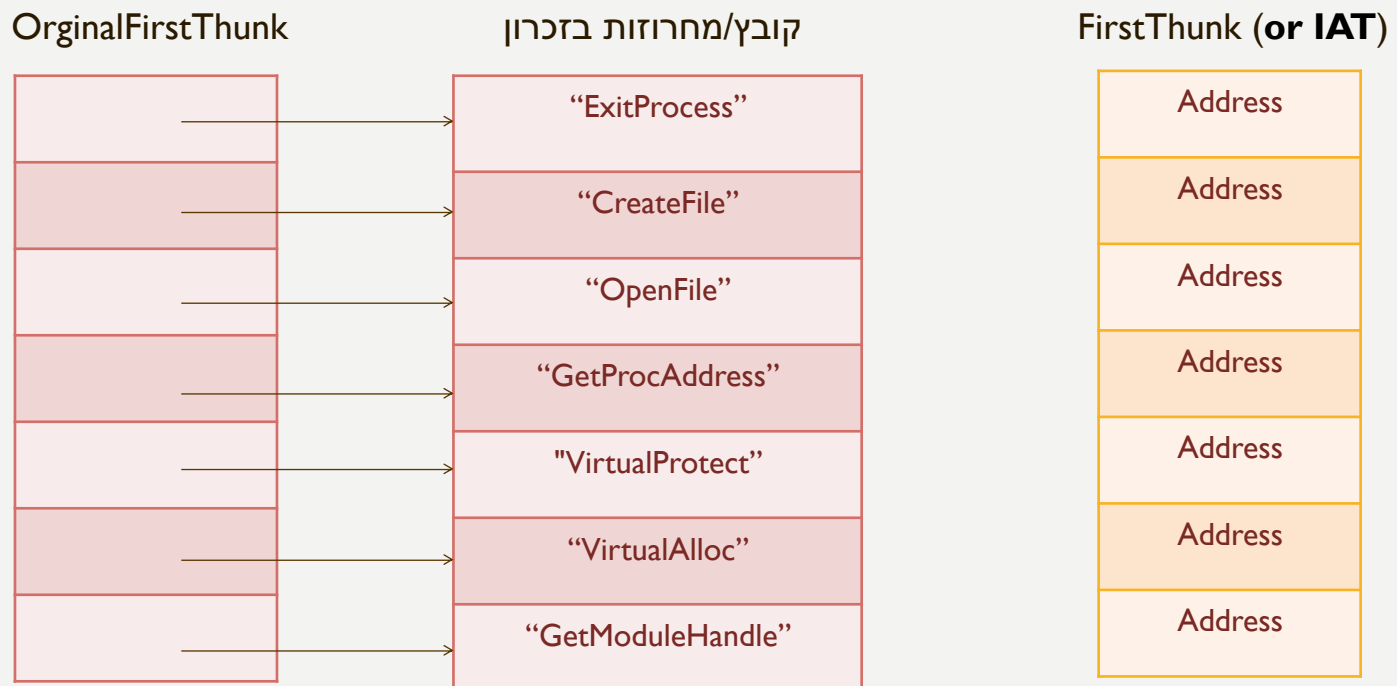
1. חיפוש ה-DLL "kernel32.dll" בתיקייה של התוכנית. אם לא נמצא - חיפוש בתיקיות מערכת, לדוגמא system32.
2. טעינת ה-DLL לזיכרון (העתקה לזיכרון, פרסור ה-PE, מעבר על ה import table וכו').



עדכון בזמן טעינה

3. עדכון טבלת ה-IAT עם הכתובות לפונקציות.

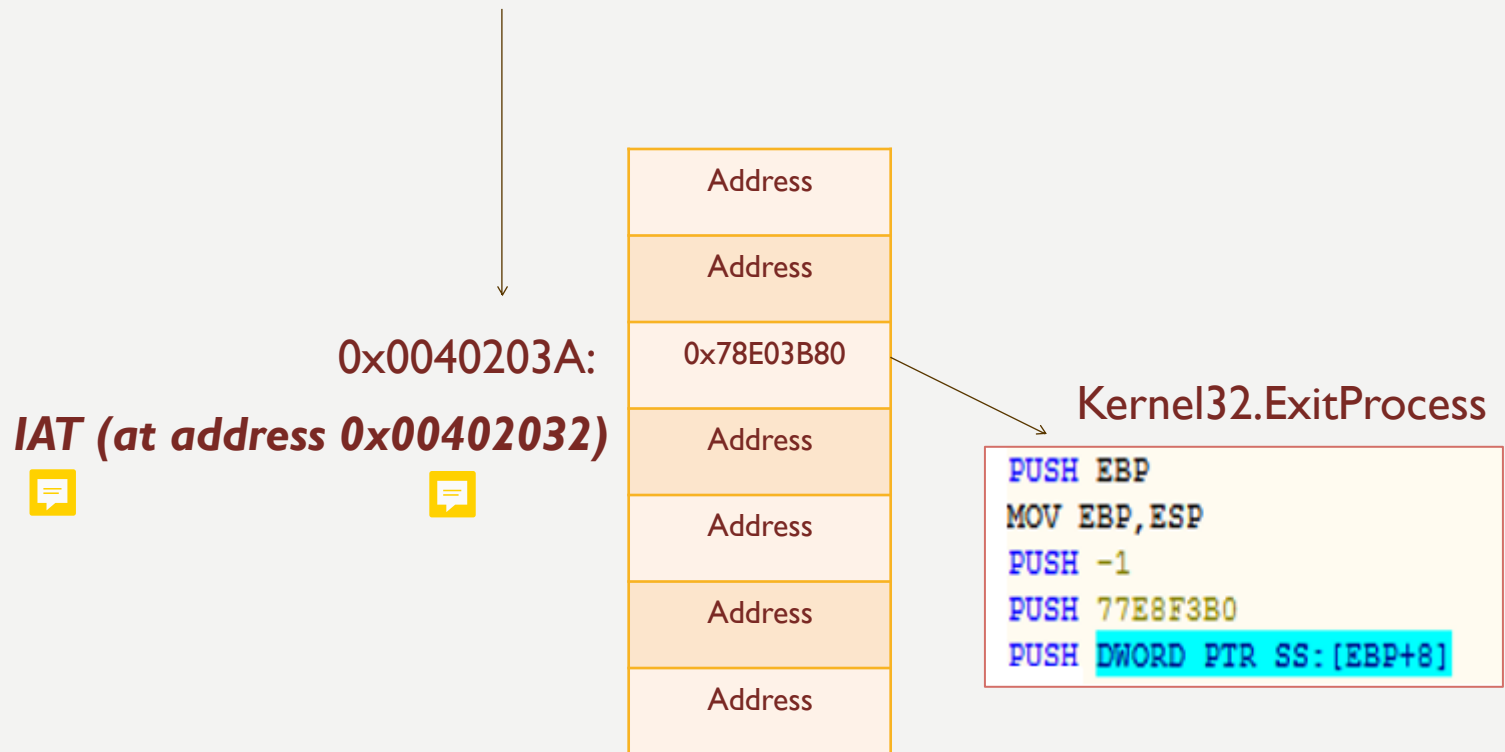
- עם כתובות של הפונקציות ב-kernel32.
- לצורך זה יש לעבור על טבלת ה-Export של kernel32.dll.



שימוש בזמן ריצה

נניח שהתוכנית שלנו נטענה ל-0x00400000 , ובכתובת 0x0040657A יש את הפקודה הבאה:

```
CALL DWORD PTR DS:[0040203A] ; // call to ExitProcess
```



הרשאות והקשר DLL



DLL וניהול זיכרון

- כל תהליך שטוען DLL ממפה אותו למרחב הזיכרון הווירטואלי שלו.
- פונקציות של DLL רצות עם ההרשאות ובהקשר של החוט (thread) של התוכנית אשר קרא להן:
 - יש שימוש באותה טבלת קבצים פתוחים
 - ה-DLL משתמש במחסנית ובמרחב הזיכרון של החוט אשר קרא לו
 - ה-DLL מקצה בheap של התהליך אשר טען אותו

חומר נוסף לקריאה

מאד מומלץ



- [https://msdn.microsoft.com/en-us/library/windows/desktop/ms680547\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms680547(v=vs.85).aspx)
- https://en.wikibooks.org/wiki/X86_Disassembly/Windows_Executable_Files
- https://en.wikipedia.org/wiki/Portable_Executable
- https://en.wikipedia.org/wiki/Object_file
- https://en.wikipedia.org/wiki/Dynamic-link_library
- [https://en.wikipedia.org/wiki/Library_\(computing\)#Shared_libraries](https://en.wikipedia.org/wiki/Library_(computing)#Shared_libraries)
- [https://en.wikipedia.org/wiki/Loader_\(computing\)](https://en.wikipedia.org/wiki/Loader_(computing))
- https://en.wikipedia.org/wiki/Dynamic-link_library#Symbol_resolution_and_binding