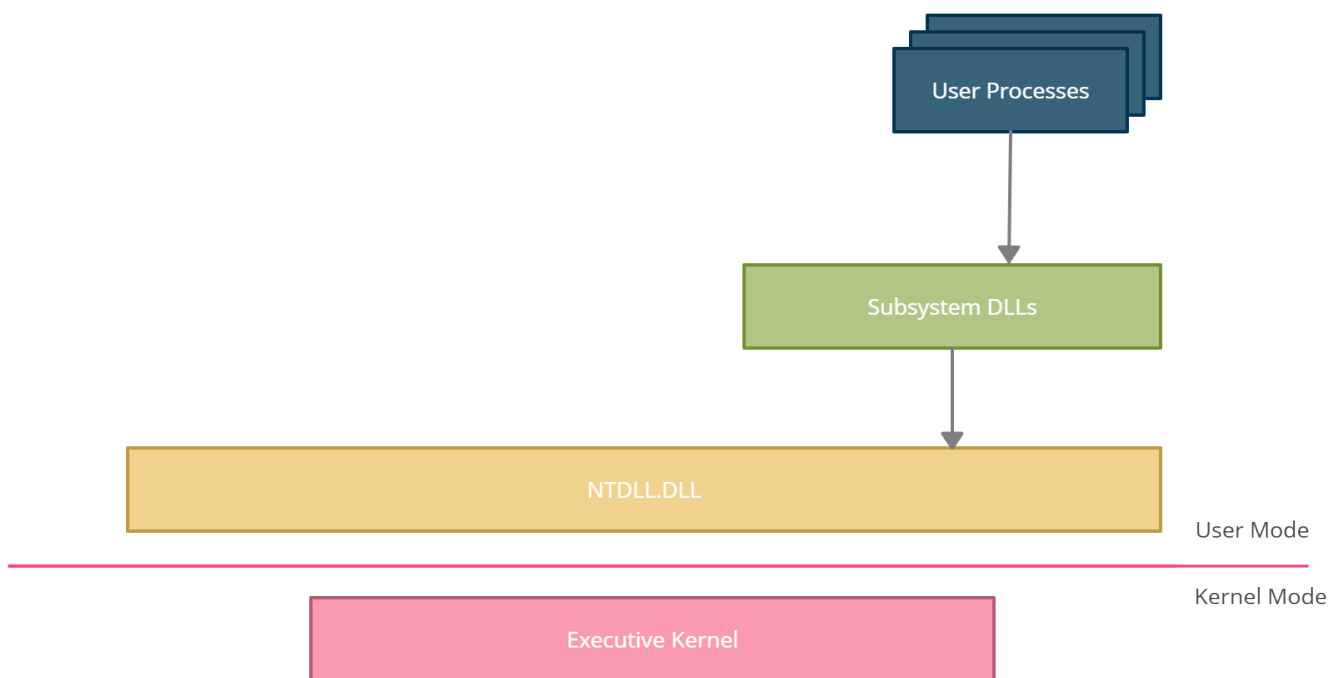


Windows Architecture

Introduction

This module explains the Windows architecture and what happens under the hood of Windows processes and applications.

A processor inside a machine running the Windows operating system can operate under two different modes: User Mode and Kernel Mode. Applications run in user mode, and operating system components run in kernel mode. When an application wants to accomplish a task, such as creating a file, it cannot do so on its own. The only entity that can complete the task is the kernel, so instead applications must follow a specific function call flow. The diagram below shows a high level of this flow.



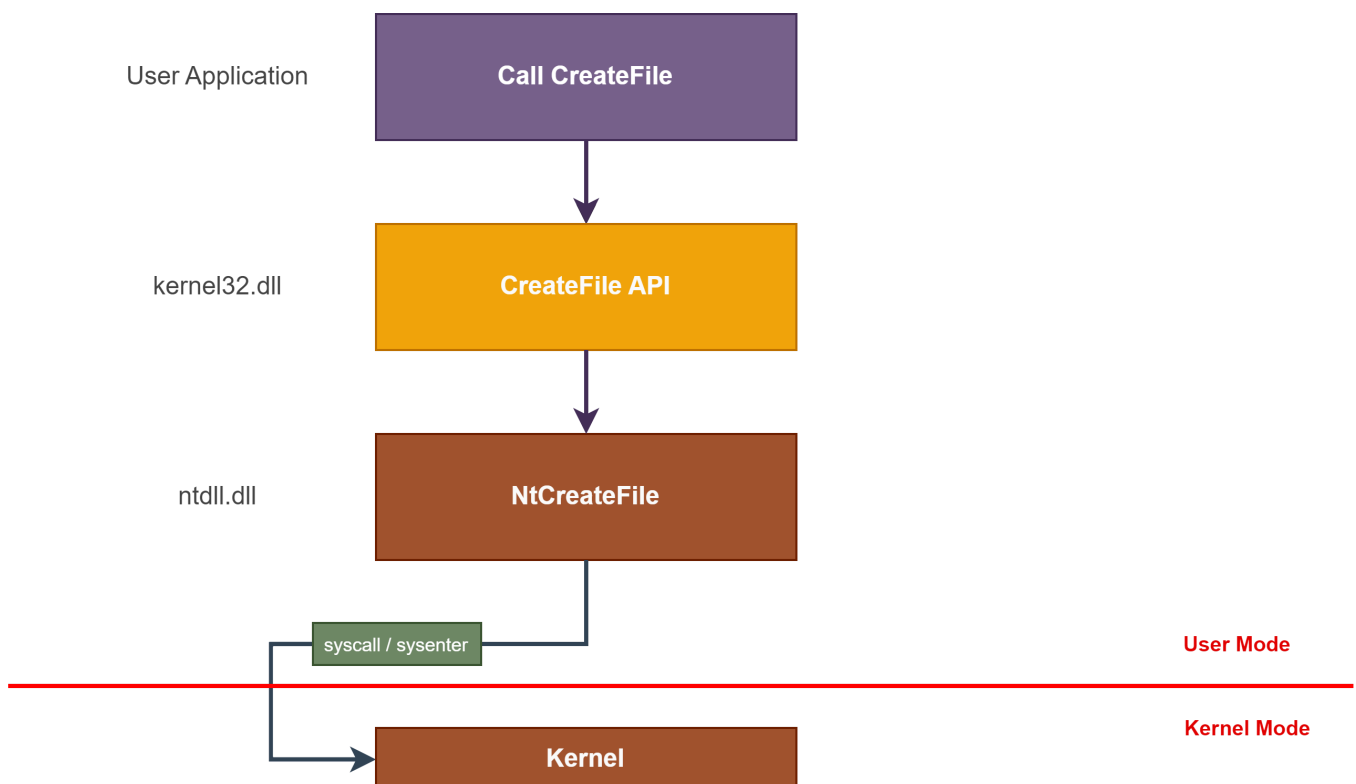
1. **User Processes** - A program/application executed by the user such as Notepad, Google Chrome or Microsoft Word.
2. **Subsystem DLLs** - DLLs that contain API functions that are called by user processes. An example of this would be `kernel32.dll` exporting the [CreateFile](#) Windows API (WinAPI) function, other common subsystem DLLs are `ntdll.dll`, `advapi32.dll`, and `user32.dll`.
3. **Ntdll.dll** - A system-wide DLL which is the lowest layer available in user mode. This is a special DLL that creates the transition from user mode to kernel mode. This is often referred to as the Native API or NTAPI.
4. **Executive Kernel** - This is what is known as the Windows Kernel and it calls other drivers and modules available within kernel mode to complete tasks. The Windows kernel is partially stored in a

file called `ntoskrnl.exe` under "C:\Windows\System32".

Function Call Flow

The image below shows an example of an application that creates a file. It begins with the user application calling the `CreateFile` WinAPI function which is available in `kernel32.dll`.

`kernel32.dll` is a critical DLL that exposes applications to the WinAPI and is therefore can be seen loaded by most applications. Next, `CreateFile` calls its equivalent NTAPI function, `NtCreateFile`, which is provided through `ntdll.dll`. `Ntdll.dll` then executes an assembly `sysenter` (x86) or `syscall` (x64) instruction, which transfers execution to kernel mode. The kernel `NtCreateFile` function is then used which calls kernel drivers and modules to perform the requested task.



Function Call Flow Example

This example shows the function call flow happening through a debugger. This is done by attaching a debugger to a binary that creates a file via the `CreateFileW` Windows API.

The user application calls the `CreateFileW` WinAPI.

File View Debug Tracing Plugins Favourites Options Help Mar 26 2022 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source

00007FF6568F101F	45:33C0	xor r8d,r8d
00007FF6568F1022	C74424 20 02000000	mov dword ptr ss:[rsp+20],2
00007FF6568F102A	BA 00000010	mov edx,10000000
00007FF6568F102E	FF15 CB0F0000	call qword ptr ds:[<&CreateFileW>]
00007FF6568F1037	48:83C4 48	add rsp,48
00007FF6568F1038	C3	ret
00007FF6568F103C	CC	int3
00007FF6568F103D	CC	int3
00007FF6568F103E	CC	int3
00007FF6568F103F	CC	int3
00007FF6568F1040	CC	int3
00007FF6568F1041	CC	int3
00007FF6568F1042	CC	int3
00007FF6568F1043	CC	int3
00007FF6568F1044	CC	int3
00007FF6568F1045	CC	int3
00007FF6568F1046	6666:0F1F8400 00000000	nop word ptr ds:[rax+rax],ax
00007FF6568F1050	48:3B0D B1F0000	cmp rcx,qword ptr ds:[<__security_cookie>]
00007FF6568F1057	75 10	jne <consoleapplication2.ReportFailure>

Next, CreateFileW calls its equivalent NTAPI function, NtCreateFile.

File View Debug Tracing Plugins Favourites Options Help Mar 26 2022 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source

00007FFACD475BE8	C745 98 40000000	mov dword ptr ss:[rbp-68],40
00007FFACD475BEF	48:8945 90	mov qword ptr ss:[rbp-70],rax
00007FFACD475BF3	53 0F7545 40	movdqu xmmword ptr ss:[rbp-60],xmm0
00007FFACD475BF8	48:FF15 C9051B00	call qword ptr ds:[<&NtCreateFile>]
00007FFACD475BF7	87 1 4400 00	mov dword ptr ds:[rax+rax],eax
00007FFACD475C04	6548:8B0C25 60000000	mov rcx,qword ptr ds:[60]
00007FFACD475C0D	4C:8BC6	mov r8,rsi
00007FFACD475C10	33D2	xor edx,edx
00007FFACD475C12	8BD8	mov ebx,eax
00007FFACD475C14	48:8B49 30	mov rcx,qword ptr ds:[rcx+30]
00007FFACD475C18	48:FF15 D1191B00	call qword ptr ds:[<&RtlFreeHeap>]
00007FFACD475C1F	0F1F4400 00	nop dword ptr ds:[rax+rax],eax

Finally, the NtCreateFile function uses a syscall assembly instruction to transition from user mode to kernel mode. The kernel will then be the one that creates the file.

File View Debug Tracing Plugins Favourites Options Help Mar 26 2022 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Thre

00007FFACFA6DB50	4C:8BD1	mov r10,rcx	NtCreateFile
00007FFACFA6DB53	B8 55000000	mov eax,55	SS:'U'
00007FFACFA6DB58	F60425 0803FE7F 01	test byte ptr ds:[7FFE0308],1	
00007FFACFA6DB60	75 03	jne ntdll.7FFACFA6DB65	
00007FFACFA6DB62	0F05	syscall	NtCreateFile
00007FFACFA6DB64	C3	ret	
00007FFACFA6DB65	CD 2E	int 2E	
00007FFACFA6DB67	C3	ret	

Directly Invoking The Native API (NTAPI)

It's important to note that applications can invoke syscalls (i.e. NTDLL functions) directly without having to go through the Windows API. The Windows API simply acts as a wrapper for the Native API. With that being said, the Native API is more difficult to use because it is not officially documented by Microsoft. Furthermore, Microsoft advises against the use of Native API functions because they can be changed at any time without warning.

Future modules will explore the benefits of directly invoking the Native API.

