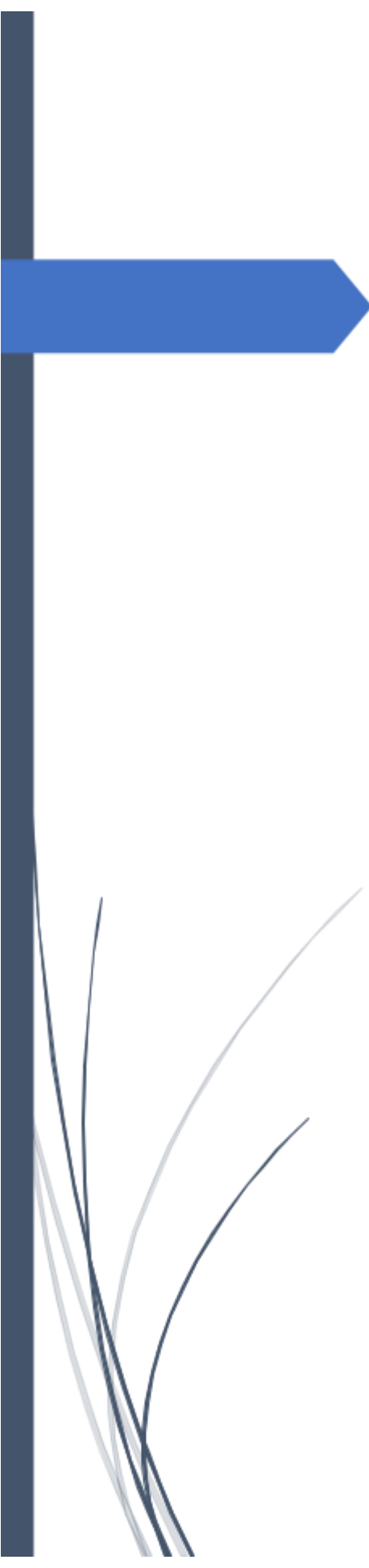




Day 1 What is a process?

Understanding and Troubleshooting
Windows Applications

Lowell Vanderpool & Nathan Vanderpool
TECH SAVVY PRODUCTIONS

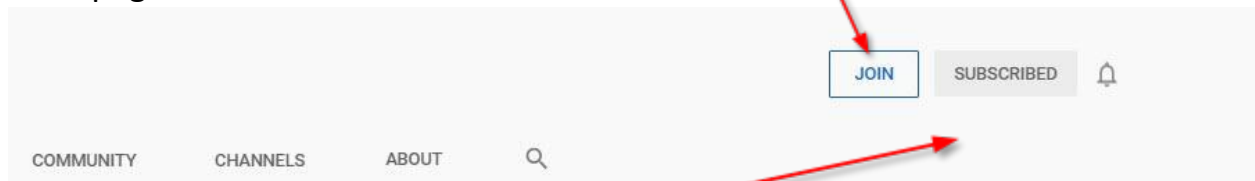
CONTACT US:

mrvanderpool@techsavvyproductions.com

nathan@techsavvyproductions.com

SUPPORT US:

If you would like to support the channel, Join our channel membership, it's \$2.99/month (less than a Starbucks coffee); see the "Join" button on our channel homepage.



OR

Subscribe to the channel as it helps our channel perform better on YouTube's algorithm.

SOCIAL MEDIA AND WEBSITE:

Check out our **Website**: <https://www.techsavvyproductions.com>

Follow us on **Twitter**: @_TechSavvyTeam

Like us on **Facebook**:

<https://www.facebook.com/Tech-Savvy-Productions-105287381500897>

Mr.V **Linkedin**: <https://www.linkedin.com/in/lowell-vanderpool-57970623/>

Nathan **Linkedin**: <https://www.linkedin.com/in/nathan-vanderpool-50a27822/>

Follow on **Instagram**: techsavvyproductions

<https://www.instagram.com/techsavvyproductions/>



We translate subtitles on our videos into the following languages: عربى, български, 简体中文), 中國傳統的), Nederlands, Suomalainen, Pilipino, français, Deutsche, हिंदी, Magyar, bahasa Indonesia, 日本語, 한국어, norsk, Polskie, português, Română, русский, Española, Kiswahili, Svenska, and Tiếng Việt

Social media logos and “Tech Savvy Productions” teaser created by The 11th Hour:
<https://www.youtube.com/user/The11thHOUR/featured>

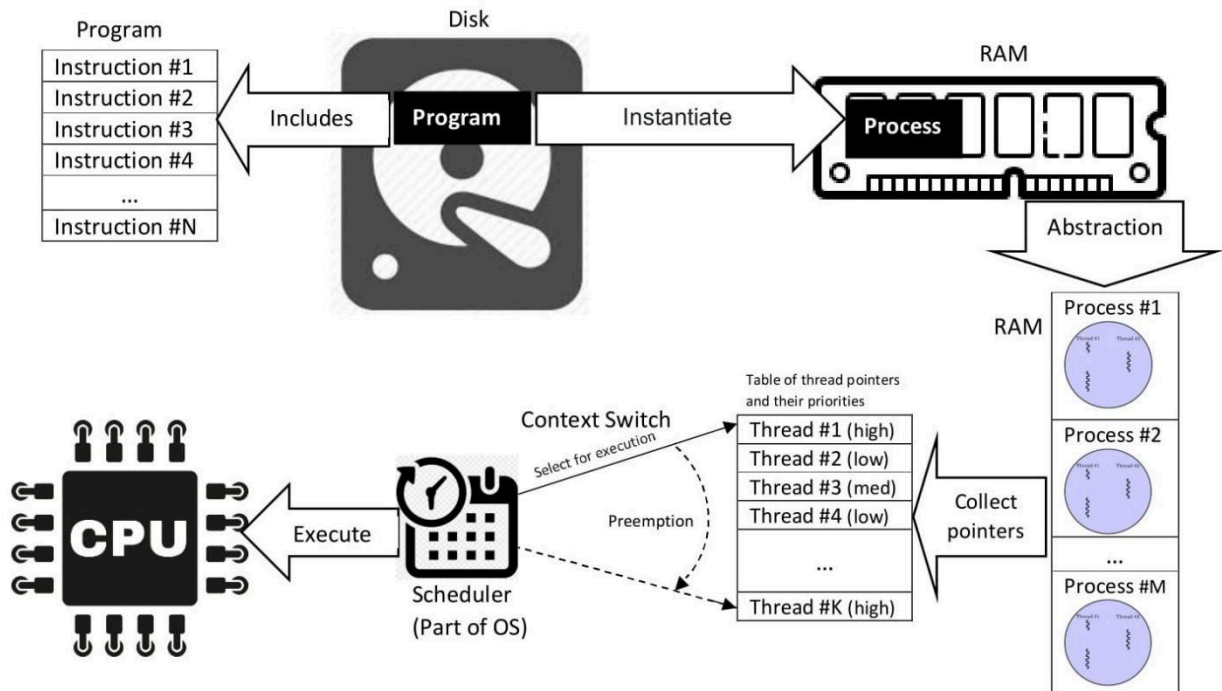


Figure 1 A great view of a process in action

Processes and Threads

<https://docs.microsoft.com/en-us/windows/win32/procthread/processes-and-threads>

An application consists of one or more processes.

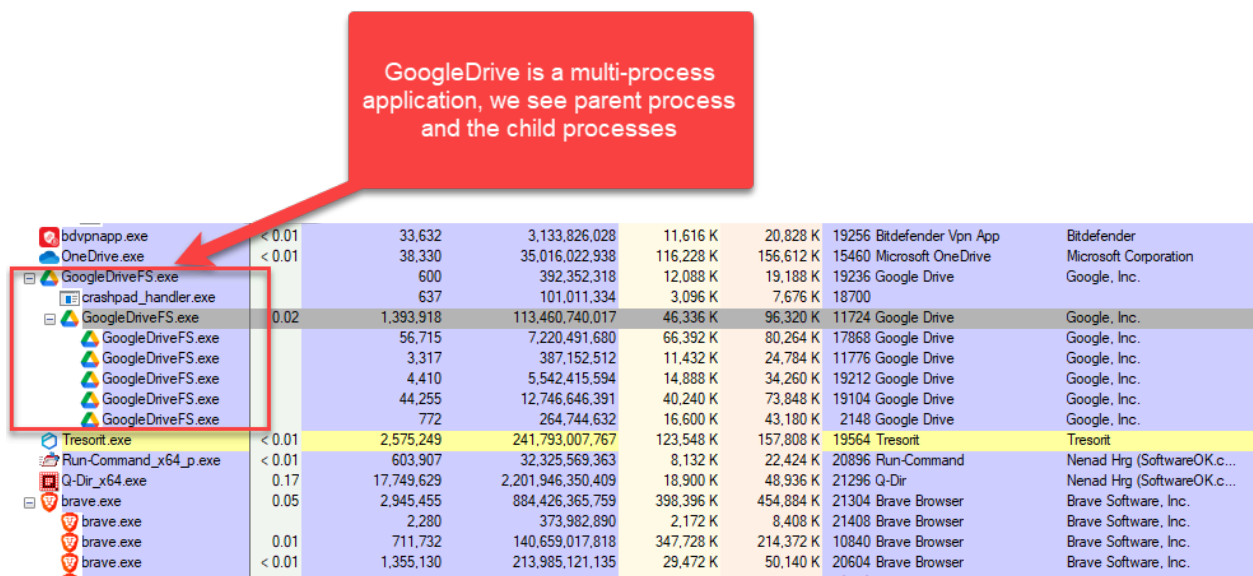


Figure 2 Google Drive is made up of many processes

Tresorit.exe is a application with a single process

bdvnpapp.exe	< 0.01	2,575,249	3,133,826,028	11,616 K	20,828 K	19256 Bitdefender Vpn App	Bitdefender
OneDrive.exe	< 0.01	38,330	35,016,022,938	116,228 K	156,612 K	15460 Microsoft OneDrive	Microsoft Corporation
GoogleDriveFS.exe		600	392,352,318	12,088 K	19,188 K	19236 Google Drive	Google, Inc.
crashpad_handler.exe		637	101,011,334	3,096 K	7,676 K	18700	
GoogleDriveFS.exe	0.02	1,393,918	113,460,740,017	46,336 K	96,320 K	11724 Google Drive	Google, Inc.
GoogleDriveFS.exe		56,715	7,220,491,680	66,392 K	80,264 K	17868 Google Drive	Google, Inc.
GoogleDriveFS.exe		3,317	387,152,512	11,432 K	24,784 K	11776 Google Drive	Google, Inc.
GoogleDriveFS.exe		4,410	5,542,415,594	14,888 K	34,260 K	19212 Google Drive	Google, Inc.
GoogleDriveFS.exe		44,255	12,746,646,391	40,240 K	73,848 K	19104 Google Drive	Google, Inc.
GoogleDriveFS.exe		772	264,744,632	16,600 K	43,180 K	2148 Google Drive	Google, Inc.
Tresorit.exe	< 0.01	2,575,249	241,793,007,767	123,548 K	157,808 K	19564 Tresorit	Tresorit
Run-Command_x64_p.exe	< 0.01	603,907	32,325,569,363	8,132 K	22,424 K	20896 Run-Command	Nenad Hrg (SoftwareOK.c...
Q-Dir_x64.exe	0.17	17,749,629	2,201,946,350,409	18,900 K	48,936 K	21296 Q-Dir	Nenad Hrg (SoftwareOK.c...
brave.exe	0.05	2,945,455	884,426,365,759	398,396 K	454,884 K	21304 Brave Browser	Brave Software, Inc.
brave.exe		2,280	373,982,890	2,172 K	8,408 K	21408 Brave Browser	Brave Software, Inc.
brave.exe	0.01	711,732	140,659,017,818	347,728 K	214,372 K	10840 Brave Browser	Brave Software, Inc.
brave.exe	< 0.01	1,355,130	213,985,121,135	29,472 K	50,140 K	20604 Brave Browser	Brave Software, Inc.

Figure 3 Tresorit.exe is a single application

A process, in the simplest terms, is an executing program.

Name	Description	Company Name	Path
~FontCache-FontFa...			C:\Windows\ServiceProfiles\LocalService\AppData\Local\...
activeds.dll	ADs Router Layer DLL	Microsoft Corporation	C:\Windows\System32\activeds.dll
activeds.tlb	Microsoft (R) ActiveDS Typelib	Microsoft Corporation	
adslidpc.dll	ADs LDAP Provider C DLL	Microsoft Corporation	
adsnt.dll	ADs Windows NT Provider DLL	Microsoft Corporation	
advapi32.dll	Advanced Windows 32 Base API	Microsoft Corporation	C:\Windows\System32\advapi32.dll
amdihk64.dll	Radeon Settings: Host Service	Advanced Micro Devices, Inc.	
amsi.dll	Anti-Malware Scan Interface	Microsoft Corporation	C:\Windows\System32\amsi.dll
atcuf64.dll	Bitdefender Active Threat Control ...	Bitdefender	C:\Program Files\Bitdefender\atcuf\26...
aticfx64.dll	aticfx64.dll	Advanced Micro Devices, Inc.	C:\Program Files\AMD\aticfx64.dll
atiu9p64.dll	atiu9pag.dll	Advanced Micro Devices, Inc.	C:\Program Files\AMD\atiu9p64.dll
atiumd64.dll	atiumd64.dll	Advanced Micro Devices, Inc.	C:\Program Files\AMD\atiumd64.dll
atiumd6a.dll	Radeon Video Acceleration Univer...	Advanced Micro Devices, Inc.	C:\Program Files\AMD\atiumd6a.dll
authz.dll	Authorization Framework	Microsoft Corporation	C:\Windows\System32\authz.dll
bcrypt.dll	Windows Cryptographic Primitives ...	Microsoft Corporation	C:\Windows\System32\bcrypt.dll
bcryptprimitives.dll	Windows Cryptographic Primitives ...	Microsoft Corporation	C:\Windows\System32\bcryptprimitives.dll
cbfconnect20.dll	CBFS Connect 2020	Callback Technologies, Inc.	C:\Users\YouTubeCreation\AppData\Local\Tresorit\v0.8\...
cfgmgr32.dll	Configuration Manager DLL	Microsoft Corporation	C:\Windows\System32\cfgmgr32.dll
clbcatq.dll	COM+ Configuration Catalog	Microsoft Corporation	C:\Windows\System32\clbcatq.dll
clr.dll	Microsoft .NET Runtime Common	Microsoft Corporation	C:\Windows\Microsoft.NET\Framework64\v4.0.30319\clr.dll

Tresorit.exe executes the many *.dll files loaded into the process. Only a process can execute a *.dll file.

Figure 4 Processes are executing programs

One or more threads run in the context of the process. A *thread* is the basic unit to which the operating system allocates processor time. A thread can execute any part of the process code, including parts currently being executed by another thread.

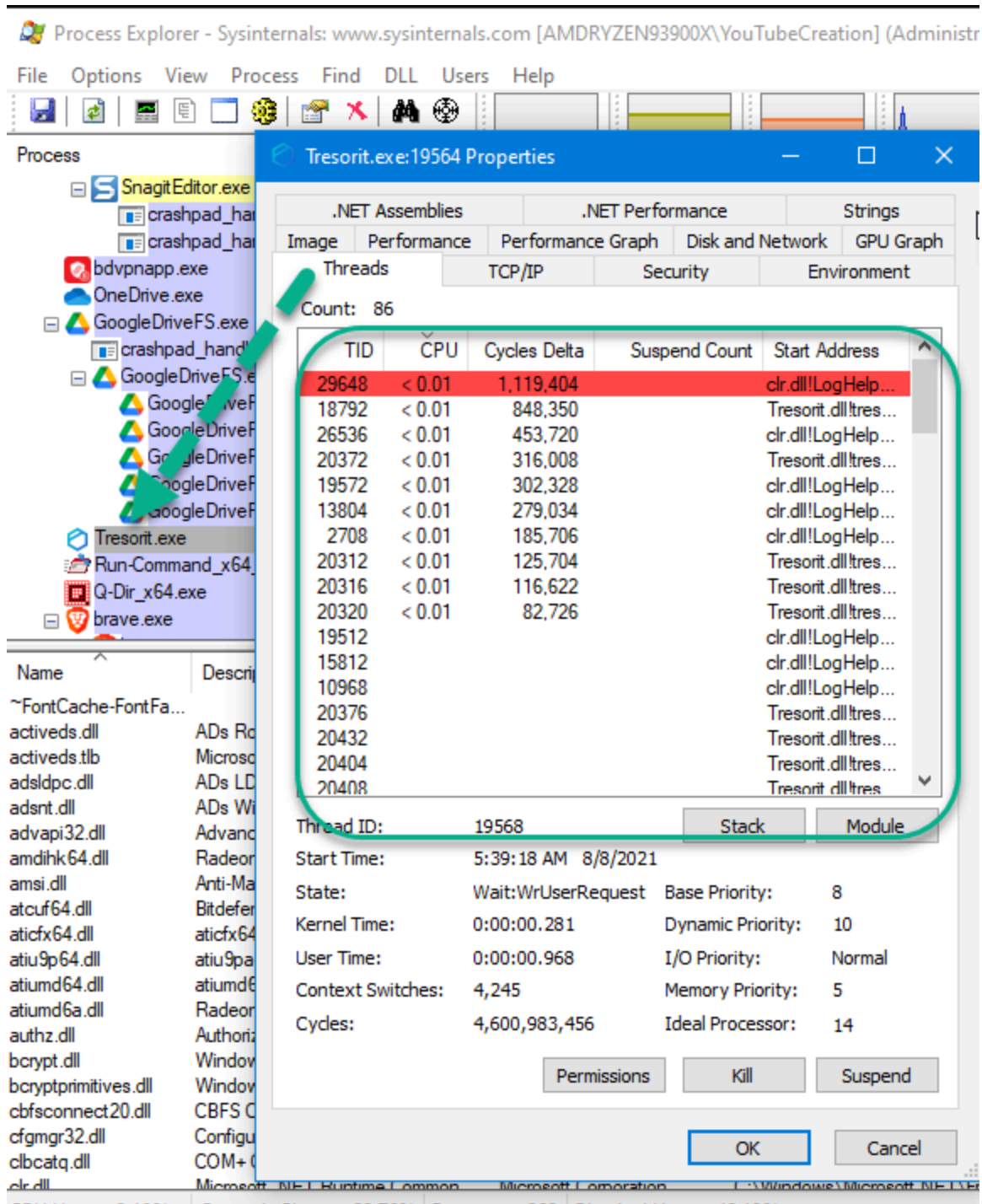


Figure 5 Threads run inside a process and are executed by the CPU

A *job object* allows groups of processes to be managed as a unit. Job objects are namable, securable, sharable objects that control attributes of the processes associated with them. Operations performed on the job object affect all processes associated with the job object.

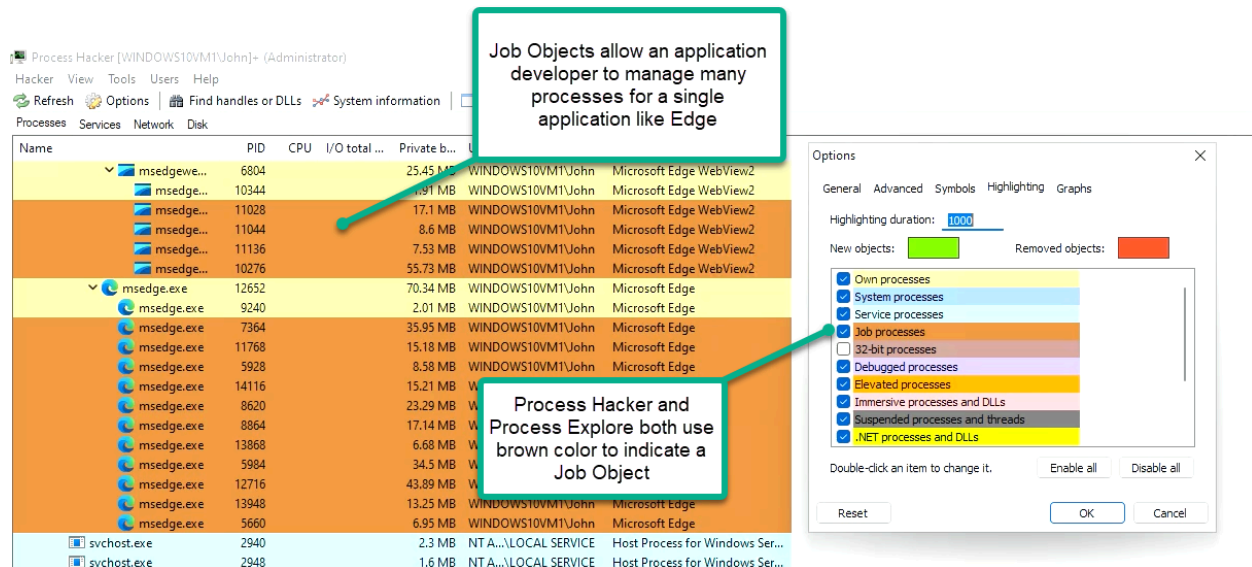


Figure 6 Job Object

A *thread pool* is a collection of worker threads that efficiently execute asynchronous callbacks on behalf of the application. The thread pool is primarily used to reduce the number of application threads and provide management of the worker threads.

A *fiber* is a unit of execution that must be manually scheduled by the application. Fibers run in the context of the threads that schedule them.

User-mode scheduling (UMS) is a lightweight mechanism that applications can use to schedule their own threads. UMS threads differ from [fibers](#) in that each UMS thread has its own thread context instead of sharing the thread context of a single thread.

- [What's New in Processes and Threads](#)
- [About Processes and Threads](#)
- [Using Processes and Threads](#)
- [Process and Thread Reference](#)

About Processes and Threads

<https://docs.microsoft.com/en-us/windows/win32/procthread/about-processes-and-threads>

Each *process* provides the resources needed to execute a program.

A process has:

- a virtual address space
- executable code
- open handles to system objects
- a security context
- a unique process identifier
- environment variables
- a priority class
- minimum and maximum working set sizes
- and at least one thread of execution.

Each process is started with a single thread, often called the *primary thread*, but can create additional threads from any of its threads.

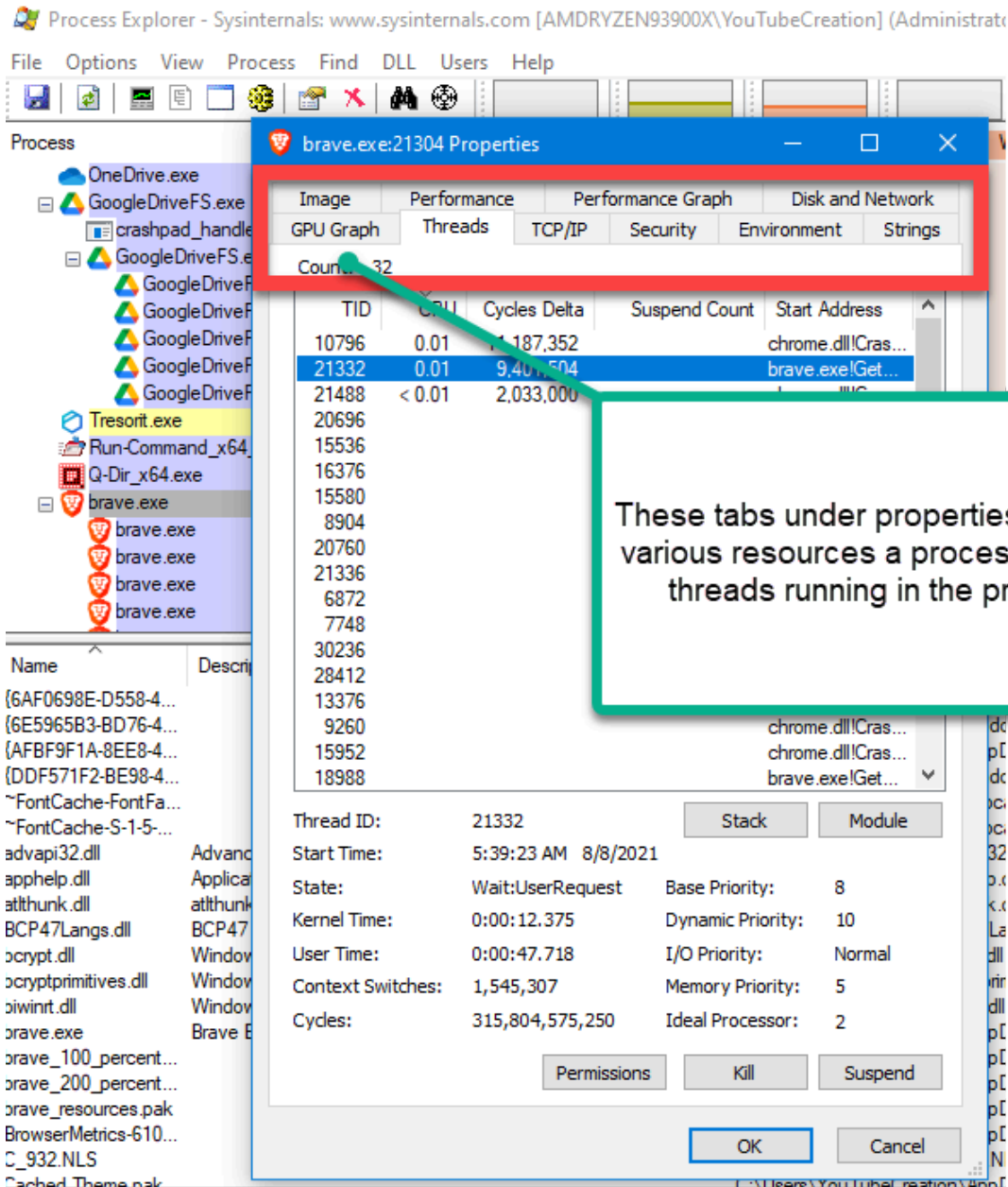
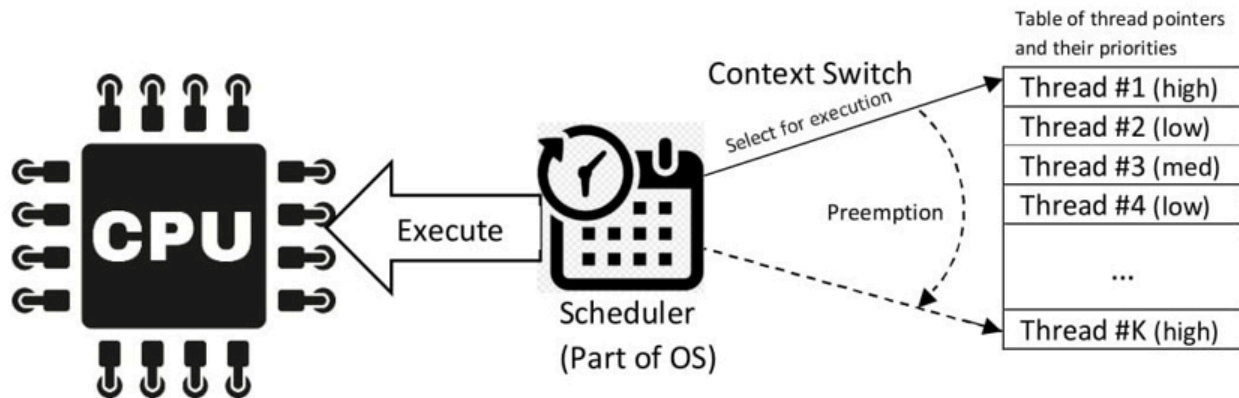


Figure 7 Resources a process gives

A *thread* is the entity within a process that can be scheduled for execution. All threads of a process share its virtual address space and system resources. In addition, each thread maintains exception handlers, a scheduling priority, thread local storage, a unique thread identifier, and a set of structures the system will use to save the thread context until it is scheduled.



The *thread context* includes the thread's set of machine registers, the kernel stack, a thread environment block, and a user stack in the address space of the thread's process. Threads can also have their own security context, which can be used for impersonating clients.

Threads have TID, CPU utilization, a stack and a set of registers

brave.exe:21304 Properties

Working Set PID Description

Working Set	PID	Description
19,188 K	19236	Google Drive
7,676 K	18700	
96,564 K	11724	Google Drive
80,264 K	17868	Google Drive

Count: 25

TID	CPU	Cycles Delta
10796	0.02	17,084,800
21332	0.01	2,227,484
21488	0.01	1,239,218
30236		
16376		
6872		
15536		
21336		
20696		
20760		
20228		
13376		
28412		
15580		
7748		
9260		
1340		
1336		

Stack for thread 10796

Index	Address
0	ntoskml.exe!KeSynchronizeExecution+0x5b66
1	ntoskml.exe!KeWaitForSingleObject+0x1460
2	ntoskml.exe!KeWaitForSingleObject+0x98f
3	ntoskml.exe!KeWaitForSingleObject+0x233
4	ntoskml.exe!ExWaitForRunDownProtectionRelease+0x7dd
5	ntoskml.exe!KeWaitForSingleObject+0x3a29
6	ntoskml.exe!KeWaitForSingleObject+0x1787
7	ntoskml.exe!KeWaitForSingleObject+0x98f
8	ntoskml.exe!KeWaitForMultipleObjects+0x2be
9	ntoskml.exe!ObWaitForMultipleObjects+0x2f0
10	win32kfull.sys!memcmp+0x872
11	win32kfull.sys!+0x1bee
12	win32k.sys!EngSaveFloatingPointState+0x28df
13	ntoskml.exe!KeWaitForSingleObject+0x7a49

Thread ID: 10796

Start Time: 5:39:38 AM 8/8/2021

State: Wait:UserRequest

Kernel Time: 0:00:12.000

User Time: 0:00:02.062

Context Switches: 385,695

Cycles: 58,318,988,553

Base Priority: 8

Dynamic Priority: 10

I/O Priority: Normal

Memory Priority: 5

Ideal Processor: 3

Permissions Kill Suspend

OK Cancel

- A **thread** is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers, (and a thread ID.)
- Traditional (heavyweight) processes have a single thread of control - There is one program counter, and one sequence of instructions that can be carried out at any given time.
- As shown in Figure below, multi-threaded applications have multiple threads within a single process, each having their own program counter, stack and set of registers, but sharing common code, data, and certain structures such as open files.

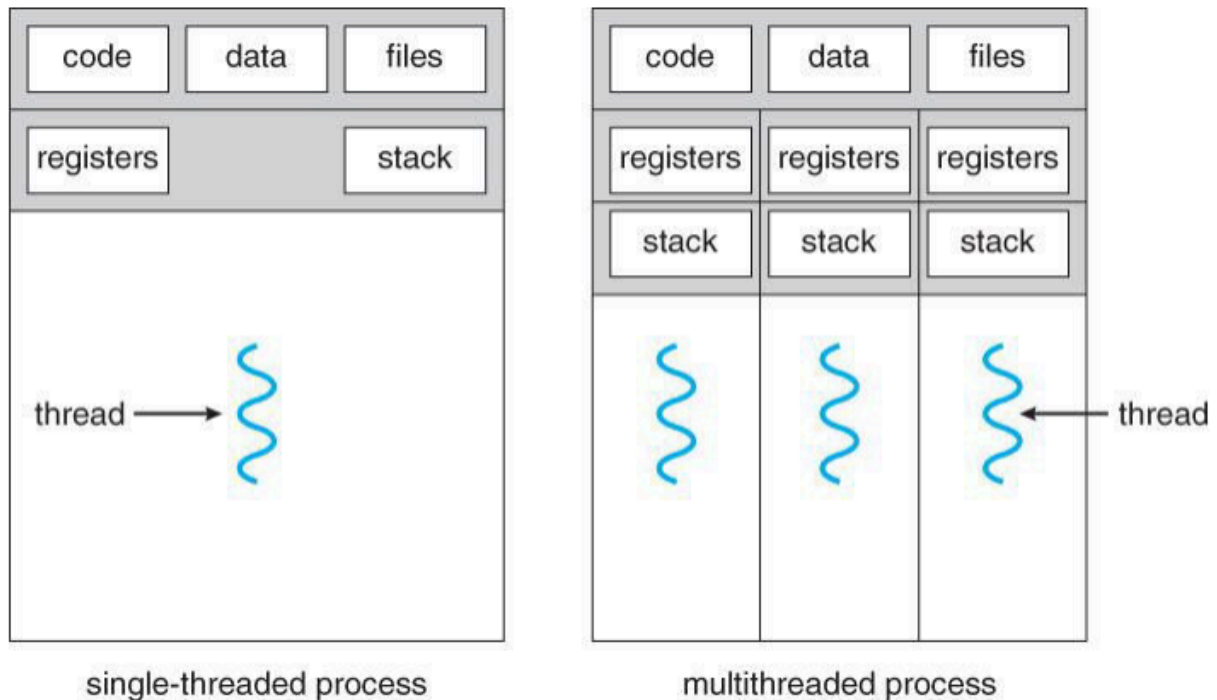


Fig. - Single-threaded and multithreaded processes

Tools for working with Windows Processes

Process Explorer written by Mark Russinovich CTO of Azure

Process Hacker Open Source by Wen Jia Liu & Steven G



<https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer>

<https://processhacker.sourceforge.io/index.php>

Process Explorer v16.42

- <https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer>
- Check often for updates on this tool
- Portable
- Microsoft's lawyers must approve

CPU	Private Bytes	Working Set	PID	Description	Company Name	Network Receiv...
Susp...	184 K	74,908 K	136			
	10,364 K	90,164 K	228			
97.38	60 K	8 K	0			2.7 KB
0.20	204 K	1,932 K	4			
0.16	0 K	0 K	n/a	Hardware Interrupts and DPCs		
	1,080 K	1,272 K	888	Windows Session Manager	Microsoft Corporation	
	708 K	28,940 K	4624			
< 0.01	2,712 K	5,508 K	1280	Client Server Runtime Process	Microsoft Corporation	
	1,368 K	6,476 K	1504	Windows Start-Up Application	Microsoft Corporation	
< 0.01	7,036 K	12,072 K	1576	Services and Controller app	Microsoft Corporation	
	13,012 K	29,688 K	1712	Host Process for Windows S...	Microsoft Corporation	
	14,616 K	23,324 K	10312	WMI Provider Host	Microsoft Corporation	
	3,356 K	8,272 K	11880	Sink to receive asynchron...	Microsoft Corporation	
	3,536 K	11,556 K	14964	COM Surrogate	Microsoft Corporation	
	24,156 K	73,520 K	12004			
	8,384 K	28,572 K	2572	Runtime Broker	Microsoft Corporation	
	121,564 K	193,964 K	14156	Search application	Microsoft Corporation	
	18,408 K	48,328 K	8236	Runtime Broker	Microsoft Corporation	
	45,836 K	58,216 K	13164	YourPhone	Microsoft Corporation	
	6,248 K	6,344 K	5708	Host Process for Setting Syn...	Microsoft Corporation	
	8,224 K	31,996 K	3760	Runtime Broker	Microsoft Corporation	
	353,924 K	392,656 K	6588			3.7 KB
	8,680 K	32,156 K	4800	Runtime Broker	Microsoft Corporation	
	16,496 K	45,536 K	13324		Microsoft Corporation	
	30,520 K	71,768 K	17440	Cortana	Microsoft Corporation	
	5,572 K	25,404 K	20940	Runtime Broker	Microsoft Corporation	
	3,224 K	11,484 K	22096	Runtime Broker	Microsoft Corporation	
	31,588 K	79,172 K	23616	Windows Shell Experience H...	Microsoft Corporation	
	7,748 K	30,752 K	23728	Runtime Broker	Microsoft Corporation	
	149,772 K	96,708 K	23596	Application Frame Host	Microsoft Corporation	
	52,360 K	2,668 K	24504	Store	Microsoft Corporation	
	7,852 K	28,400 K	24524	Runtime Broker	Microsoft Corporation	
	9,556 K	29,220 K	24184	System Settings Broker	Microsoft Corporation	
	21,984 K	2,184 K	25044	Settings	Microsoft Corporation	
	22,872 K	2,100 K	20148			
	3,272 K	8,076 K	11532	Runtime Broker	Microsoft Corporation	
	341,524 K	255,936 K	19976			
	2,908 K	7,760 K	26628	Runtime Broker	Microsoft Corporation	

Introduction to Process Explorer

Ever wondered which program has a particular file or directory open? Now you can find out. *Process Explorer* shows you information about which handles and DLLs processes have opened or loaded.

The *Process Explorer* display consists of two sub-windows. The top window always shows a list of the currently active processes, including the names of their owning accounts, whereas the information displayed in the bottom window depends on the mode that *Process Explorer* is in: if it is in handle mode you'll see the handles that the process selected in the top window has opened; if *Process Explorer* is in DLL mode you'll see the DLLs and memory-mapped files that the process has loaded. *Process Explorer* also has a powerful search capability that will quickly show you which processes have particular handles opened or DLLs loaded.

The unique capabilities of *Process Explorer* make it useful for tracking down DLL-version problems or handle leaks, and provide insight into the way Windows and applications work.

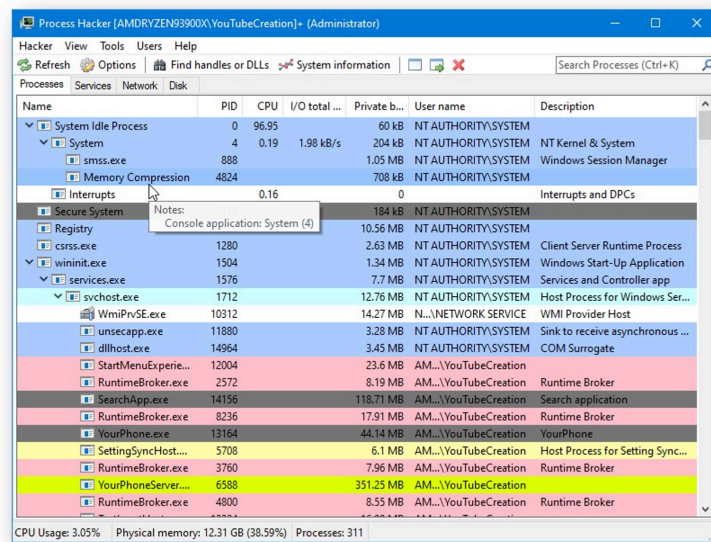
Learn More

Here are some other handle and DLL viewing tools and information available at Sysinternals:

- [The case of the Unexplained...](#) In this video, Mark describes how he has solved seemingly unsolvable system and application problems on Windows.
- [Handle](#) - a command-line handle viewer
- [ListDLLs](#) - a command-line DLL viewer
- [PsList](#) - local/remote command-line process lister
- [PsKill](#) - local/remote command-line process killer
- [Defrag Tools: #2 - Process Explorer](#) In this episode of Defrag Tools, Andrew Richards and Larry Larsen show how to use Process Explorer to view the details of processes, both at a point in time and historically.
- [Windows Sysinternals Primer: Process Explorer, Process Monitor and More](#) Process Explorer gets a lot of attention in the first Sysinternals Primer delivered by Aaron Margosis and Tim Reckmeyer at TechEd 2010.

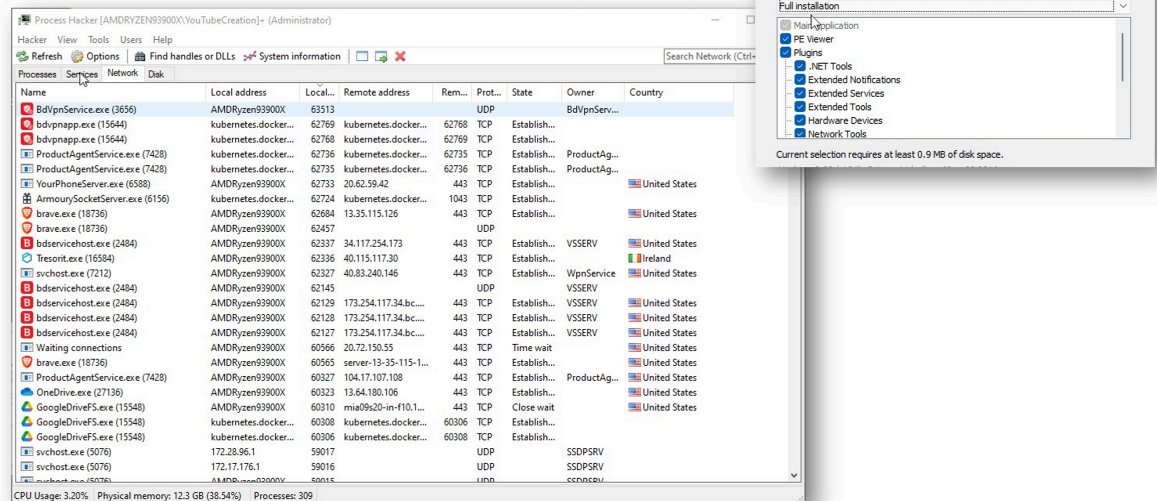
Process Hacker

- <https://processhacker.sourceforge.io/>
- Download and use the portable version
- There is an installable version **with more features**

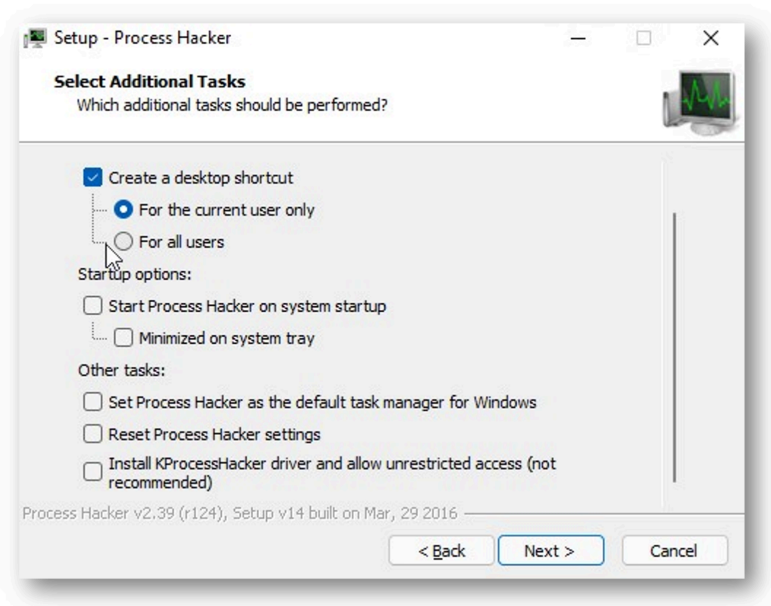


5

Why the open-source project?



6



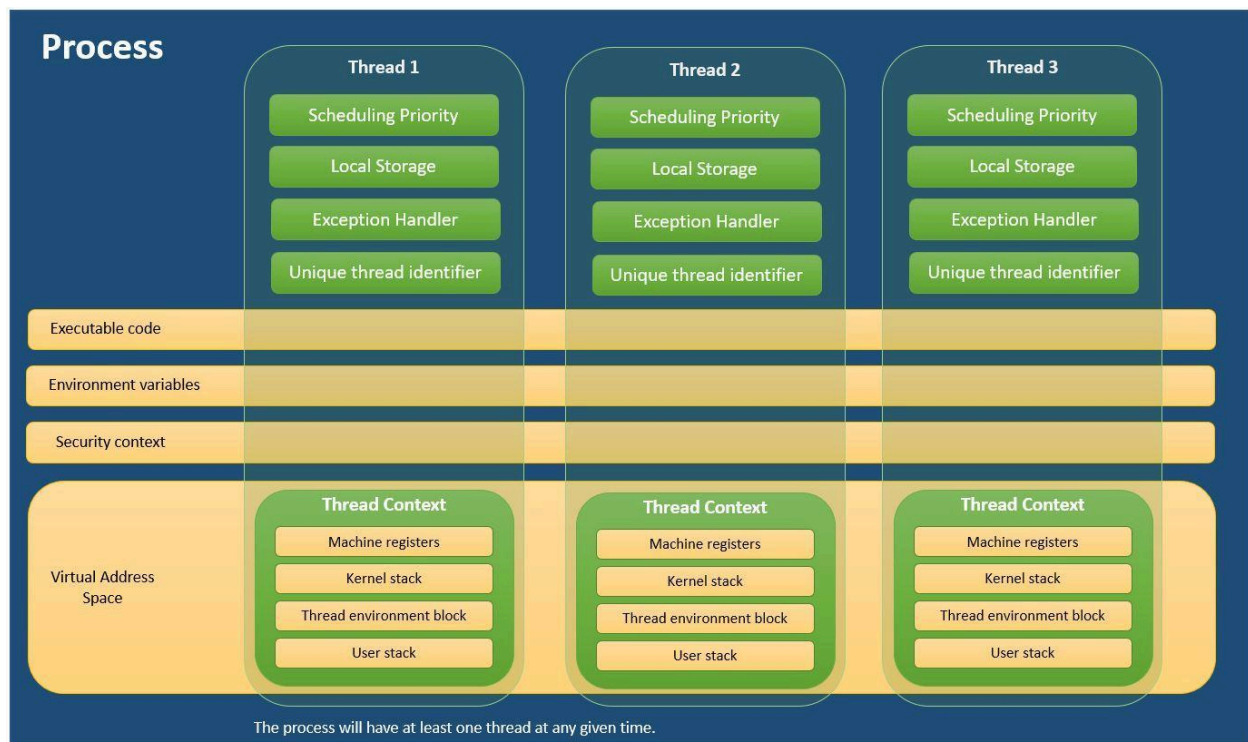
- Process Hacker normally runs under Admin rights
- KProcessHacker driver allows for removing malware

7

Differences between a process and a thread

	Process	Thread
Definition	An executing instance of a program is called a process.	A thread is a subset of the process.
Process	It has its own copy of the data segment of the parent process.	It has direct access to the data segment of its process.
Communication	Processes must use inter-process communication to communicate with sibling processes.	Threads can directly communicate with other threads of its process.
Overheads	Processes have considerable overhead.	Threads have almost no overhead.
Creation	New processes require duplication of the parent process.	New threads are easily created.
Control	Processes can only exercise control over child processes.	Threads can exercise considerable control over threads of the same process.
Changes	Any change in the parent process does not affect child processes.	Any change in the main thread may affect the behavior of the other threads of the process.
Memory	Run in separate memory spaces.	Run in shared memory spaces.
File descriptors	Most file descriptors are not shared.	It shares file descriptors.

Read the chart above and look carefully at this graphic and you can see many of the chart's concepts clearly show in the graphic below.



Process and Thread real world example

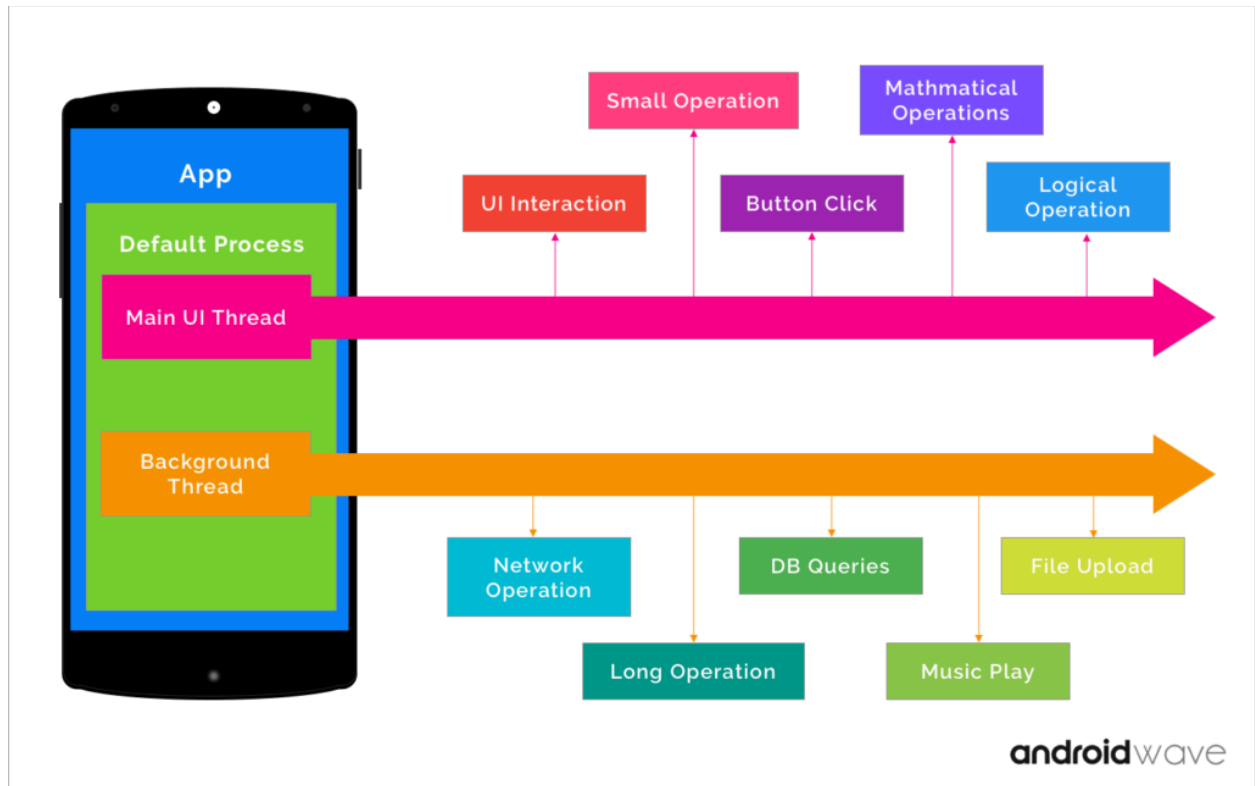


Here, The Process (or Task) name is known as **"Construct road"**.

Ok, how can I construct a road, so I need some resource for construction, right?

Resources:

- 1) Boundary of road(Area)
 - a. In technically we can call it as virtual address space, it has unique process identifier to identify the work
 - b. Access to limited boundary of area | security context
 - c. Also, you can match the other properties of **process** with the above example (environment, priority, etc.)
- 2) Hardware or labor
 - a. Number of "hardware or labor" count is based on road contractor mind set; suppose he/she want to finish the work quickly then he/she has to assign more people on this work
 - b. I.e. each worker can access this limited area (Shared boundary space)
 - c. At least one person needed to start work on this
 - d. Each person has their own identifier
 - e. So, can we call it as **Thread**? if yes match the other properties of thread with this example (exception handling, thread context etc.)

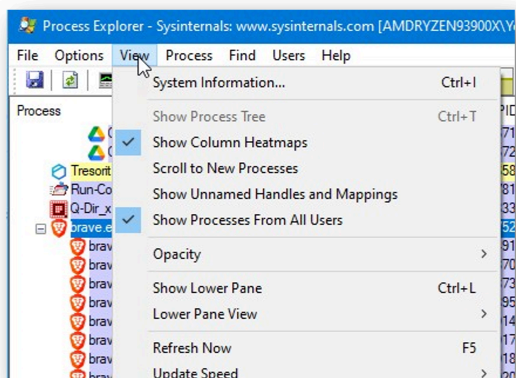


Child-Parent processes

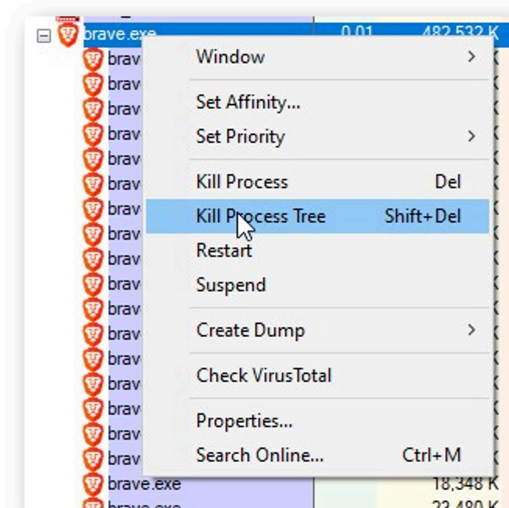
A child process is a process that is created by another process, called the parent process.

brave.exe	0.05	482,328 K	345,552 K	17524 Brave Browser	Brave Software, Inc.
brave.exe		2,232 K	7,244 K	4912 Brave Browser	Brave Software, Inc.
brave.exe	< 0.01	520,544 K	227,324 K	18704 Brave Browser	Brave Software, Inc.
brave.exe	0.01	29,956 K	48,612 K	18736 Brave Browser	Brave Software, Inc.
brave.exe		13,844 K	18,736 K	18956 Brave Browser	Brave Software, Inc.
brave.exe	< 0.01	43,112 K	72,776 K	19148 Brave Browser	Brave Software, Inc.
brave.exe		61,692 K	78,848 K	19172 Brave Browser	Brave Software, Inc.
brave.exe		16,360 K	32,616 K	19184 Brave Browser	Brave Software, Inc.
brave.exe		16,492 K	32,264 K	19200 Brave Browser	Brave Software, Inc.
brave.exe		102,776 K	83,800 K	19260 Brave Browser	Brave Software, Inc.
brave.exe		16,684 K	37,048 K	19344 Brave Browser	Brave Software, Inc.
brave.exe		38,188 K	71,764 K	19352 Brave Browser	Brave Software, Inc.
brave.exe		17,128 K	33,848 K	19396 Brave Browser	Brave Software, Inc.
brave.exe	< 0.01	34,548 K	68,272 K	18320 Brave Browser	Brave Software, Inc.
brave.exe		51,640 K	44,548 K	18616 Brave Browser	Brave Software, Inc.
brave.exe		63,784 K	57,664 K	19304 Brave Browser	Brave Software, Inc.
brave.exe		16,508 K	32,280 K	19500 Brave Browser	Brave Software, Inc.
brave.exe		101,720 K	82,316 K	19676 Brave Browser	Brave Software, Inc.
brave.exe		18,348 K	35,140 K	19840 Brave Browser	Brave Software, Inc.
brave.exe		23,480 K	39,096 K	20088 Brave Browser	Brave Software, Inc.
brave.exe		16,872 K	32,584 K	20328 Brave Browser	Brave Software, Inc.

35

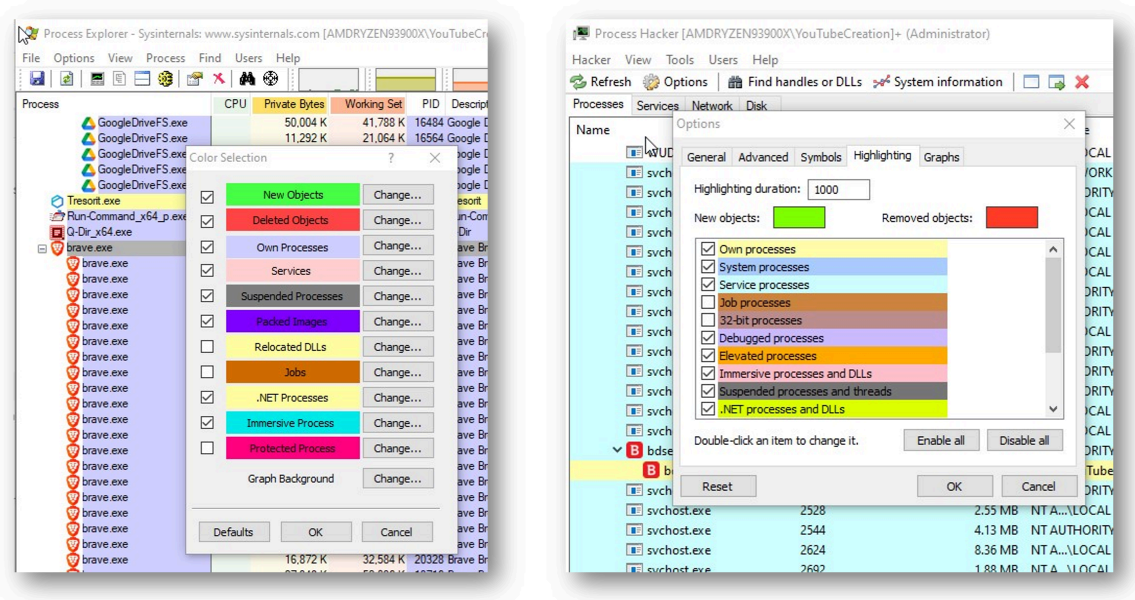


- View the child -parent relationship
- Kill (terminate) the parent and all the child processes



36

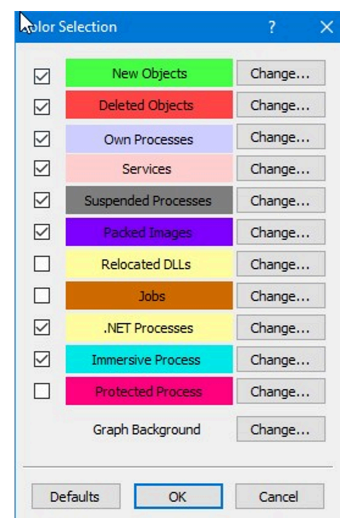
Process Explorer and Process Hacker identity different types of process with colors



37

Types of Processes

- New Objects: Process or threads created
- Deleted Objects: termination
- Own Processes: processes launched when you logged on
- Services: all your Service processes
- Suspended Processes: power saving mode, in memory but no CPU execution
- Packed images: Processes with small portions of code visible the rest either compressed or encrypted (Malware loves this feature)



38

Newer applications like “modern or UWP” are controlled by the Process Lifetime manager PLM

PowerShell Processes

Get-Process

Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

PS C:\Users\John> Get-Process

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
88	6	1136	2640		4896	0	AggregatorHost
314	20	8868	25016	0.59	10792	2	ApplicationFrameHost
270	29	21064	30344	1.08	2268	2	backgroundTaskHost
279	19	15236	23148	0.05	8636	2	backgroundTaskHost
277	14	3972	19960	0.03	13552	2	backgroundTaskHost
41	6	832	4644		13104	0	conhost
802	51	28800	17632	0.55	9404	2	Cortana
615	25	1912	2740		704	0	csrss
142	10	1660	1192		796	1	csrss
530	22	2000	3308		5752	2	csrss
434	17	3928	15968	0.83	1324	2	ctfmon
495	21	6560	13088		4104	0	dasHost
259	17	6480	14616		6996	0	DbgSvc
223	14	3096	12996	0.05	7836	2	dllhost
266	28	6816	14016	0.22	8348	2	dllhost
268	15	4492	13984		9640	0	dllhost
685	25	18416	12688		1828	1	dwm
1018	81	81448	91492		5988	2	dwm
3515	814	84224	156288	57.59	7456	2	explorer
33	6	1688	1132		768	0	fontdrvhost
33	6	1608	1080		772	1	fontdrvhost
33	7	1956	2896		5876	2	fontdrvhost
684	34	15404	29504	0.89	11320	2	GameBar
261	13	2872	13600	0.06	9960	2	GameBarFTServer
0	0	60	8		0	0	Idle
211	17	11752	4596		3356	0	inetinfo
573	33	13332	34268	38.08	14060	2	last-changed-files
527	29	11620	23920		1876	1	LugonUI
54	7	1136	1700		940	0	LsaIso
1562	23	7676	14544		948	0	lsass
0	0	760	233112		2320	0	Memory Compression
942	53	70976	10148	6.81	6188	2	Microsoft.Photos

Get-Process explorer | Format-List *


```
PS C:\Users\John> Get-Process explorer | Format-List *
```

```
Name           : explorer
Id              : 7456
PriorityClass    : Normal
FileVersion     : 10.0.22000.120 (WinBuild.160101.0800)
HandleCount     : 3465
WorkingSet      : 159686656
PagedMemorySize : 85868544
PrivateMemorySize : 85868544
VirtualMemorySize : 752824320
TotalProcessorTime : 00:00:58.0937500
SI              : 2
Handles         : 3465
VM              : 2204071047168
WS              : 159686656
PM              : 85868544
NPM             : 842584
Path            : C:\WINDOWS\Explorer.EXE
Company         : Microsoft Corporation
CPU             : 58.09375
ProductVersion  : 10.0.22000.120
Description     : Windows Explorer
Product         : Microsoft® Windows® Operating System
__NounName      : Process
BasePriority     : 8
ExitCode        :
HasExited       : False
ExitTime        :
Handle          : 3676
SafeHandle       : Microsoft.Win32.SafeHandles.SafeProcessHandle
MachineName     : .
MainWindowHandle : 197344
MainWindowTitle :
```

Get-Process | Out-GridView

Get-Process Out-GridView								
Filter								
+ Add criteria ▼								
Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName	
90	6	1140	2644	0.03	4,896	0	AggregatorHost	
363	21	6512	29308	0.75	10,792	2	ApplicationFrameHost	
288	14	7220	14444	0.14	8,288	0	audiodg	
277	14	3972	19928	0.03	13,552	2	backgroundTaskHost	
72	6	3548	4924	0.03	8,084	2	cmd	
257	14	6772	22292	0.55	7,156	2	conhost	
41	6	832	4632	0.03	13,104	0	conhost	
802	51	28800	17792	0.55	9,404	2	Cortana	
630	25	1896	2820	0.86	704	0	csrss	
145	10	1712	1308	0.39	796	1	csrss	
527	22	2100	3448	3.16	5,752	2	csrss	
438	17	4088	16512	1.30	1,324	2	ctfmon	
491	21	6340	12452	0.59	4,104	0	dasHost	
259	17	6524	14744	2.02	6,996	0	DbgSvc	
257	16	3544	14984	0.11	7,836	2	dllhost	
276	27	6756	14288	0.25	8,348	2	dllhost	
268	15	4500	14216	0.17	9,640	0	dllhost	
693	27	18908	21200	0.16	1,828	1	dwm	
969	117	103492	129268	18.77	5,988	2	dwm	
2,904	100	61368	174556	3.13	8,732	2	explorer	
33	6	1716	1164	0.30	768	0	fontdrvhost	
33	6	1636	1092	0.22	772	1	fontdrvhost	
33	7	1984	3008	0.11	5,876	2	fontdrvhost	
684	34	15404	20612	0.80	11,320	2	GameBar	

Out-GridView gives **Add criteria**

Get-Process | Out-GridView

Filter

+ Add criteria ▼

☐ Handles

☐ NPM(K)

☐ PM(K)

☐ WS(K)

☐ CPU(s)

☐ Id

☐ SI

☐ ProcessName

Add

Cancel

				CPU(s)	Id	SI	ProcessName
				0.03	4,896	0	AggregatorHost
				0.75	10,792	2	ApplicationFrameHost
				0.14	8,288	0	audiodg
				0.03	13,552	2	backgroundTaskHost
				0.03	8,084	2	cmd
				0.55	7,156	2	conhost
				0.03	13,104	0	conhost
				0.55	9,404	2	Cortana
				0.86	704	0	csrss
145	10	1712	1308	0.39	796	1	csrss
527	22	2100	3448	3.16	5,752	2	csrss
438	17	4088	16512	1.30	1,324	2	ctfmon
491	21	6340	12452	0.59	4,104	0	dasHost
259	17	6524	14744	2.02	6,996	0	DbgSvc
257	16	3544	14984	0.11	7,836	2	dllhost
276	27	6756	14288	0.25	8,348	2	dllhost
268	15	4500	14216	0.17	9,640	0	dllhost
693	27	18908	21200	0.16	1,828	1	dwm

Tasklist Command-line

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.22000.120]
(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>tasklist

Image Name                      PID Session Name        Session#    Mem Usage
=====
System Idle Process             0 Services             0           8 K
System                          4 Services            0          18,492 K
Secure System                   88 Services           0          17,152 K
Registry                       144 Services           0          37,984 K
smss.exe                       592 Services           0           504 K
csrss.exe                      704 Services           0           2,836 K
wininit.exe                    776 Services           0           1,728 K
csrss.exe                      796 Console            1           1,308 K
winlogon.exe                   876 Console            1           3,592 K
services.exe                   920 Services           0           7,344 K
LsaIso.exe                     940 Services           0           1,728 K
lsass.exe                      948 Services           0          15,376 K
svchost.exe                    624 Services           0          28,164 K
fontdrvhost.exe               772 Console            1           1,092 K
fontdrvhost.exe               768 Services           0           1,164 K
svchost.exe                    1076 Services           0          14,948 K
svchost.exe                    1124 Services           0           5,304 K
svchost.exe                    1192 Services           0          81,712 K
svchost.exe                    1268 Services           0           3,396 K
svchost.exe                    1300 Services           0           7,224 K
svchost.exe                    1356 Services           0           4,536 K
svchost.exe                    1392 Services           0           1,220 K
svchost.exe                    1408 Services           0           3,908 K
```

Computer Process

In [computing](#), a **process** is the [instance](#) of a [computer program](#) that is being executed by one or many [threads](#). It contains the program code and its activity. Depending on the [operating system](#) (OS), a process may be made up of multiple threads of execution that execute instructions [concurrently](#).^{[1][2]}

While a computer program is a passive collection of [instructions](#) typically stored in a file on disk, a process is the execution of those instructions after being loaded from the disk into memory. Several processes may be associated with the same program; for example, opening up several instances of the same program often results in more than one process being executed.

[Multitasking](#) is a method to allow multiple processes to share [processors](#) (CPUs) and other system resources. Each CPU (core) executes a single [task](#) at a time. However, multitasking allows each processor to [switch](#) between tasks that are being executed without having to wait for each task to finish ([preemption](#)). Depending on the operating system implementation, switches could be performed when tasks initiate and wait for completion of [input/output](#) operations, when a task voluntarily yields the CPU, on hardware [interrupts](#), and when the operating system scheduler decides that a process has expired its fair share of CPU time (e.g, by the [Completely Fair Scheduler](#) of the [Linux kernel](#)).

A common form of multitasking is provided by CPU's [time-sharing](#) that is a method for interleaving the execution of users' processes and threads, and even of independent kernel tasks - although the latter feature is feasible only in preemptive [kernels](#) such as [Linux](#). Preemption has an important side effect for interactive processes that are given higher priority with respect to CPU bound processes, therefore users are immediately assigned computing resources at the simple pressing of a key or when moving a mouse. Furthermore, applications like video and music reproduction are given some kind of real-time priority, preempting any other lower priority process. In time-sharing systems, [context switches](#) are performed rapidly, which makes it seem like multiple processes are being executed simultaneously on the same processor. This simultaneous execution of multiple processes is called [concurrency](#).

For security and reliability, most modern [operating systems](#) prevent direct [communication](#) between independent processes, providing strictly mediated and controlled inter-process communication functionality.



Contents

- [Representation](#)
- [Multitasking and process management](#)
 - [Process states](#)
- [Inter-process communication](#)

- [History](#)
- [See also](#)
- [Notes](#)
- [References](#)
- [Further reading](#)
- [External links](#)

Representation

CPU[] 2.0%		Tasks: 16 total, 1 running									
Mem[] 13/123MB		Load average: 0.37 0.12 0.04									
Sup[] 0/109MB		Uptime: 00:00:50									
PID	USER	PRI	NI	VRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
3692	per	15	0	2424	1204	980	R	2.0	1.0	0:00.24	htop
1	root	16	0	2952	1852	532	S	0.0	1.5	0:00.77	/sbin/init
2236	root	20	-1	2316	728	472	S	0.0	0.6	0:01.06	/sbin/udevd --daemon
3224	dhcp	18	-2	2412	552	244	S	0.0	0.4	0:00.00	dhclient3 -e IF_NE
3488	root	18	0	1692	516	448	S	0.0	0.4	0:00.00	/sbin/getty 38400
3491	root	18	0	1696	520	448	S	0.0	0.4	0:00.01	/sbin/getty 38400
3497	root	18	0	1696	516	448	S	0.0	0.4	0:00.00	/sbin/getty 38400
3500	root	18	0	1692	516	448	S	0.0	0.4	0:00.00	/sbin/getty 38400
3501	root	16	0	2772	1196	936	S	0.0	0.9	0:00.04	/bin/login --
3504	root	18	0	1696	516	448	S	0.0	0.4	0:00.00	/sbin/getty 38400
3539	syslog	15	0	1916	704	564	S	0.0	0.6	0:00.12	/sbin/syslogd -u s
3561	root	18	0	1840	536	444	S	0.0	0.4	0:00.79	/bin/dd bs 1 if /p
3563	klog	18	0	2472	1376	408	S	0.0	1.1	0:00.37	/sbin/klogd -P /va
3590	daemon	25	0	1960	428	308	S	0.0	0.3	0:00.00	/usr/sbin/atd
3604	root	18	0	2336	792	632	S	0.0	0.6	0:00.00	/usr/sbin/cron
3645	per	15	0	5524	2924	1428	S	0.0	2.3	0:00.45	-bash

F1Help F2Setup F3Search F4Invert F5Free F6SortBuf F7Nice F8Nice F9Kill F10Quit

A list of processes as displayed by [htop](#)

Name	Description	PID	PPID	Username	CPU %/s	CPU Time	KIO Bandwidth	KIO Write	Network Size	Memory	Shared Memory
System	systemd	1	0	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
init	systemd	2	1	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	3	1	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	4	3	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	5	4	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	6	5	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	7	6	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	8	7	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	9	8	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	10	9	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	11	10	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	12	11	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	13	12	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	14	13	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	15	14	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	16	15	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	17	16	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	18	17	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	19	18	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	20	19	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	21	20	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	22	21	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	23	22	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	24	23	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	25	24	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	26	25	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	27	26	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	28	27	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	29	28	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	30	29	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	31	30	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	32	31	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	33	32	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	34	33	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	35	34	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	36	35	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	37	36	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	38	37	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	39	38	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	40	39	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	41	40	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	42	41	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	43	42	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	44	43	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	45	44	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	46	45	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	47	46	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	48	47	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	49	48	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	50	49	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	51	50	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	52	51	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	53	52	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	54	53	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	55	54	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	56	55	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	57	56	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	58	57	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	59	58	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	60	59	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	61	60	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	62	61	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	63	62	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	64	63	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	65	64	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	66	65	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	67	66	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	68	67	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	69	68	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	70	69	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	71	70	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	72	71	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	73	72	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	74	73	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	75	74	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	76	75	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	77	76	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	78	77	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	79	78	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	80	79	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	81	80	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	82	81	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	83	82	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	84	83	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	85	84	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	86	85	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	87	86	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	88	87	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	89	88	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	90	89	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	91	90	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	92	91	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	93	92	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	94	93	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	95	94	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	96	95	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	97	96	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	98	97	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-daemon	dbus-daemon	99	98	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B
dbus-launch	dbus-launch	100	99	root	0%	0.00s	0.00B/s	0.00B/s	0.00B	0.00B	0.00B

A process table as displayed by [KDE System Guard](#)

In general, a computer system process consists of (or is said to *own*) the following resources:

- An *image* of the executable [machine code](#) associated with a program.
- Memory (typically some region of [virtual memory](#)); which includes the executable code, process-specific data (input and output), a [call stack](#) (to keep track of active [subroutines](#) and/or other events), and a [heap](#) to hold intermediate computation data generated during run time.
- Operating system descriptors of resources that are allocated to the process, such as [file descriptors](#) ([Unix](#) terminology) or [handles](#) ([Windows](#)), and data sources and sinks.
- [Security](#) attributes, such as the process owner and the process' set of permissions (allowable operations).
- [Processor](#) state ([context](#)), such as the content of [registers](#) and physical memory addressing. The *state* is typically stored in computer registers when the process is executing, and in memory otherwise.^[1]

The operating system holds most of this information about active processes in data structures called [process control blocks](#). Any subset of the resources, typically at least the processor state, may be associated with each of the process' [threads](#) in operating systems that support threads or *child* processes.

The operating system keeps its processes separate and allocates the resources they need, so that they are less likely to interfere with each other and cause system failures (e.g., [deadlock](#) or [thrashing](#)). The operating system may also provide mechanisms for [inter-process communication](#) to enable processes to interact in safe and predictable ways.

Multitasking and process management

Main article: [Process management \(computing\)](#)

A [multitasking operating system](#) may just switch between processes to give the appearance of many processes [executing](#) simultaneously (that is, in [parallel](#)), though in fact only one process can be executing at any one time on a single [CPU](#) (unless the CPU has multiple cores, then [multithreading](#) or other similar technologies can be used).^[a]

It is usual to associate a single process with a main program, and child processes with any spin-off, parallel processes, which behave like [asynchronous](#) subroutines. A process is said to *own* resources, of which an *image* of its program (in memory) is one such resource. However, in multiprocessing systems *many* processes may run off of, or share, the same [reentrant](#) program at the same location in memory, but each process is said to own its own *image* of the program.

Processes are often called "tasks" in [embedded](#) operating systems. The sense of "process" (or task) is "something that takes up time", as opposed to "memory", which is "something that takes up space".^[b]

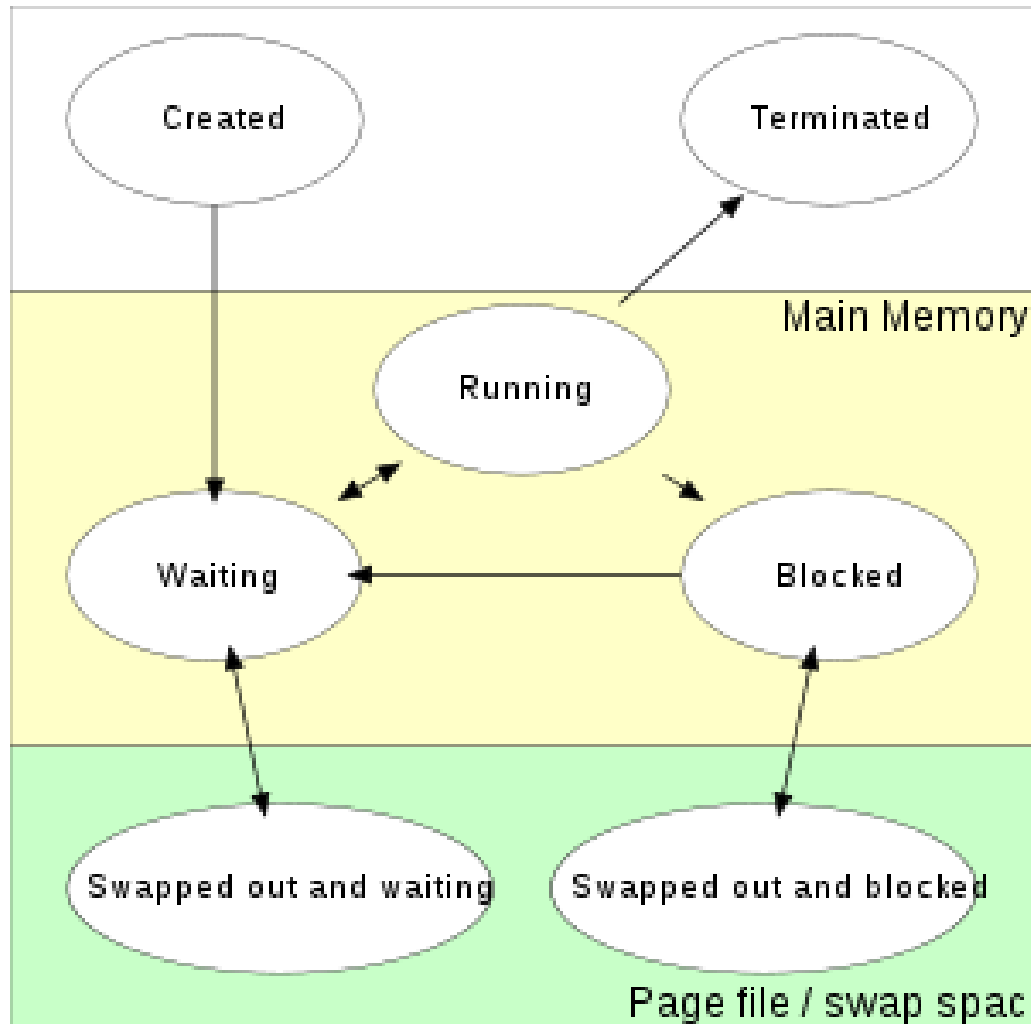
The above description applies to both processes managed by an operating system, and processes as defined by [process calculi](#).

If a process request something for which it must wait, it will be blocked. When the process is in the [blocked state](#), it is eligible for swapping to disk, but this is transparent in a [virtual memory](#) system, where regions of a process's memory may be really on disk and not in [main memory](#) at any time. Note that even

portions of active processes/tasks (executing programs) are eligible for swapping to disk, if the portions have not been used recently. Not all parts of an executing program and its data have to be in physical memory for the associated process to be active.

Process states

Main article: [Process state](#)



The various process states, displayed in a [state diagram](#), with arrows indicating possible transitions between states.

An operating system [kernel](#) that allows multitasking needs processes to have [certain states](#). Names for these states are not standardized, but they have similar functionality.^[1]

- First, the process is "created" by being loaded from a [secondary storage](#) device ([hard disk drive](#), [CD-ROM](#), etc.) into [main memory](#). After that the [process scheduler](#) assigns it the "waiting" state.
- While the process is "waiting", it waits for the [scheduler](#) to do a so-called [context switch](#). The context switch loads the process into the processor and changes the state to "running" while the previously "running" process is stored in a "waiting" state.
- If a process in the "running" state needs to wait for a resource (wait for user input or file to open, for example), it is assigned the "blocked" state. The process state is changed back to "waiting" when the process no longer needs to wait (in a blocked state).
- Once the process finishes execution, or is terminated by the operating system, it is no longer needed. The process is removed instantly or is moved to the "terminated" state. When removed, it just waits to be removed from main memory.^{[1][3]}

Inter-process communication

Main article: [Inter-process communication](#)

When processes need to communicate with each other they must share parts of their [address spaces](#) or use other forms of inter-process communication (IPC). For instance in a shell pipeline, the output of the first process need to pass to the second one, and so on; another example is a task that can be decomposed into cooperating but partially independent processes which can run at once (i.e., using concurrency, or true parallelism - the latter model is a particular case of concurrent execution and is feasible whenever enough CPU cores are available for all the processes that are ready to run).

It is even possible for two or more processes to be running on different machines that may run different operating system (OS), therefore some mechanisms for communication and synchronization (called [communications protocols](#) for distributed computing) are needed (e.g., the Message Passing Interface, often simply called [MPI](#)).

History

See also: [History of operating systems](#)

By the early 1960s, computer control software had evolved from [monitor control software](#), for example [IBSYS](#), to [executive control software](#). Over time, computers got faster while [computer time](#) was still neither cheap nor fully utilized; such an environment made [multiprogramming](#) possible and necessary. Multiprogramming means that several programs run [concurrently](#). At first, more than one program ran on a single processor, as a result of underlying [uniprocessor](#) computer architecture, and they shared scarce and limited hardware resources; consequently, the concurrency was of a *serial* nature. On later systems with [multiple processors](#), multiple programs may run concurrently in [parallel](#).

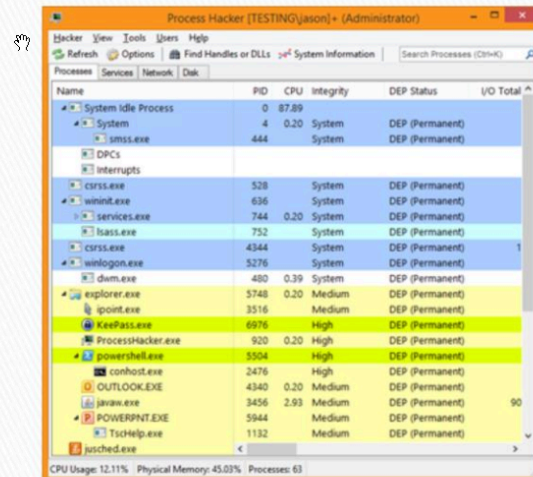
Programs consist of sequences of instructions for processors. A single processor can run only one instruction at a time: it is impossible to run more programs at the same time. A program might need some [resource](#), such as an input device, which has a large delay, or a program might start some slow operation, such as sending output to a printer. This would lead to processor being "idle" (unused). To keep the processor busy at all times, the execution of such a program is halted and the operating system switches the processor to run another program. To the user, it will appear that the programs run at the same time (hence the term "parallel").

Shortly thereafter, the notion of a "program" was expanded to the notion of an "executing program and its context". The concept of a process was born, which also became necessary with the invention of [re-entrant code](#). [Threads](#) came somewhat later. However, with the advent of concepts such as [time-sharing](#), [computer networks](#), and multiple-CPU [shared memory](#) computers, the old "multiprogramming" gave way to true [multitasking](#), multiprocessing and, later, [multithreading](#).

What Is Process Hacker?

What Is Process Hacker?

- ▶ A free graphical tool for managing processes, threads, memory, handles, modules and tokens on Windows.
- ▶ Similar to Microsoft's *Process Explorer*, but open source and a bit more fun.
- ▶ Great for forensics, combating malware, troubleshooting, and getting Windows-dirty.



Process Hacker is a free, open source, graphical tool for managing 32-bit and 64-bit Microsoft Windows processes, services, threads, memory, handles, modules, Security Access Tokens (SATs) and network connections.

It is a wonderful tool for analyzing and combating malware, understanding low-level details of the Windows operating system, troubleshooting, and experimentation.

Process Hacker is similar to the famous Sysinternals Process Explorer tool from Microsoft (<http://www.microsoft.com/sysinternals>), but open source and a bit more fun. Now that Process Explorer is the property of Microsoft Corporation, Process Explorer cannot be enhanced with features which might embarrass Microsoft.

There are also no legal hassles when redistributing Process Hacker or its source code (no Microsoft lawyers = good thing). Examining the source code of Process Hacker is an interesting way to learn more about Windows internals, and Process Hacker itself is an actively maintained project.

Fortunately, if you prefer Process Explorer, almost all of this presentation applies to it as well. So please feel free to use Process Hacker or Process Explorer as you wish.

Both tools are great. And if you have questions, don't forget about the discussion forums for Process Hacker (<http://processhacker.sourceforge.net/forums/>) and Sysinternals Process Explorer (<http://forum.sysinternals.com>).

Installation and Configuration

- ▶ <http://processhacker.sourceforge.net>
- ▶ Install Debugging Tools for Windows.
- ▶ Configure the symbols path.
- ▶ Enable plugins.
- ▶ Add all columns.
- ▶ Run from USB drive if you wish.
- ▶ Install the KProcessHacker driver too.

Installation and Configuration

Process Hacker installs on 32-bit or 64-bit Windows XP-SP2 and later. Download binaries and source from <http://processhacker.sourceforge.net>.

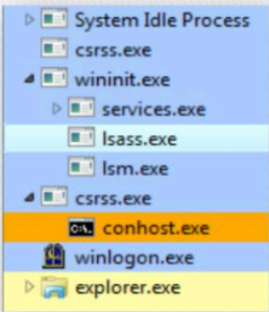
If you want to run Process Hacker from a USB drive, just copy the download zip file's contents to the USB drive and launch processhacker.exe with the -settings option with the path to your preferences file.

Make sure to install the latest version of Microsoft's "Debugging Tools for Windows" (<http://processhacker.sourceforge.net/faq.php>), and then in Process Hacker, pull down the Hacker menu > Options > Symbols tab > confirm that you have a valid path to DBGHELP.DLL.

Next, in order to see the names of functions in stack traces, create a folder named "C:\Symbols" on your hard drive. Then, in Process Hacker, pull down the Hacker menu > Options > Symbols tab > and enter the following string into the Search Path: SRV*C:\Symbols*<http://msdl.microsoft.com/download/symbols> Finally, in almost every window with a table, you can right-click on the columns header and Choose Columns. When in doubt, add them all.

Process Tree View

- ▶ Color Highlighting for Processes
- ▶ Child Processes Are Indented
- ▶ Run As and Create Service
- ▶ Hide Signed Processes
- ▶ Create Crash Dump File
- ▶ Terminate Process/Thread
- ▶ Inject DLL



Process Tree View

A "process" is an environment in which a thread can run, providing a virtual memory address space, handles to objects, a security context, and other resources. A running process can spawn new processes, and thus there can exist parent-child relationships among processes. These parent-child relationships are shown as indentations in the "tree view" of Process Hacker. Every process contains at least one thread, and may contain many.

A "thread" is that which has at least one stack and actually performs computing work within a process. A "job" is a collection of one or more processes, though a given process might not be a member of a job (look for a Job tab). A job allows a group of processes to be managed as a unit, such as placing restrictions on what those processes can do.

In Process Hacker, you can perform many tasks related to processes, including:

- Show different types of processes in different colors (Hacker menu > Options > Highlighting tab).
- Run commands as Local System in the desired session and desktop (Hacker menu > Run As).
- Create a new driver or interactive service (Tools menu > Create Service).
- Hide processes with verified digital signatures (View menu > Hide Signed Processes).
- Create a crash dump file of a process for analysis, but doesn't crash the process (right-click process > Create Dump File).
- Terminate almost any process, even if protected by some rootkits (right-click process > Terminate).
- Inject a DLL or EXE into the memory of a running process (right-click process > Misc > Inject DLL).

Tip: You can run Process Hacker itself with Local System privileges (Hacker menu > Run As). You can also create a shortcut to do the same using the Sysinternals psexec.exe tool, like so "psexec.exe -s -d -i 1 processhacker.exe".

When Combating Malware

- ▶ **Look for packed processes:**
 - Very common with malware, rarely used otherwise.
- ▶ **Focus on unsigned processes:**
 - Malware usually is not digitally signed.
- ▶ **And when it's hammer time:**
 - Terminate threads and processes.
 - Suspend malware team members first.

When Combating Malware

It is very common for malware executables to be packed. A "packed" process is compressed and/or encrypted, perhaps multiple times, along with a tiny bit of decompression/decryption code which is executed first in order to reconstruct (in memory, not on the hard drive) the working form of the malware.

Some legitimate programs use packing to thwart reverse engineering, but this is rare. Hence, enable highlighting of packed processes and focus on them first when combating malware.

To scan the file system for packed binaries, use free tools like Mandiant's Red Curtain(www.mandiant.com).

Mandiant Red Curtain

Overview

MANDIANT Red Curtain is free software for Incident Responders that assists with the analysis of malware. MRC examines executable files (e.g., .exe, .dll, and so on) to determine how suspicious they are based on a set of criteria. It examines

multiple aspects of an executable, looking at things such as the Entropy (in other words, randomness), indications of packing, compiler and packing signatures, the presence of digital signatures, and other characteristics to generate a threat "score." This score can be used to identify whether a set of files is worthy of further investigation.

Malware binaries are rarely digitally signed, so add the "Verification Status" and "Verified Signer" columns, then hide signed processes (View menu) to quickly identify those processes which lack a verified signature from a trusted code-signing certificate. Keep in mind, however, that it is still very common for legitimate programs to not have signatures. Also, a driver for the Stuxnet worm was digitally signed by a trusted issuer, so the presence of a signature does not automatically rule out that binary as being malicious.

Malware will sometimes try to hide its processes from tools like Task Manager and tasklist.exe. There are a variety of rootkit techniques for hiding processes, but some techniques affect only a few Windows API functions and user-mode data structures, while others hook kernel-mode native functions and modify deep kernel-mode objects too, which makes these cloaking techniques much harder to defeat. To try to find hidden processes on 32-bit systems (Tools menu > Hidden Processes), Process Hacker avoids simply asking the Windows API for a list of running processes and instead tries to open every possible process ID number individually using a Windows API function which hopefully is not being hooked by the rootkit (this is the "brute force" method). Process Hacker can also examine every open handle to see what process it is associated with (the "CSR handle" method).

If a process is visible by brute-forcing PIDs or following handles, but it's not visible when simply asking the Windows API for all running processes, then that process is considered hidden.

Keep in mind, though, that these detection techniques can be beaten, so you'll need to use additional tools like GMER (www.gmer.net) and offline file system analysis to improve your odds of detecting any rootkits. Also, the hidden processes feature is not available on 64-bit Windows.

GMER is an application that detects and removes [rootkits](#) .

It scans for:

- hidden processes

- hidden threads

- hidden modules

- hidden services

- hidden files

- hidden disk sectors (MBR)

- hidden Alternate Data Streams

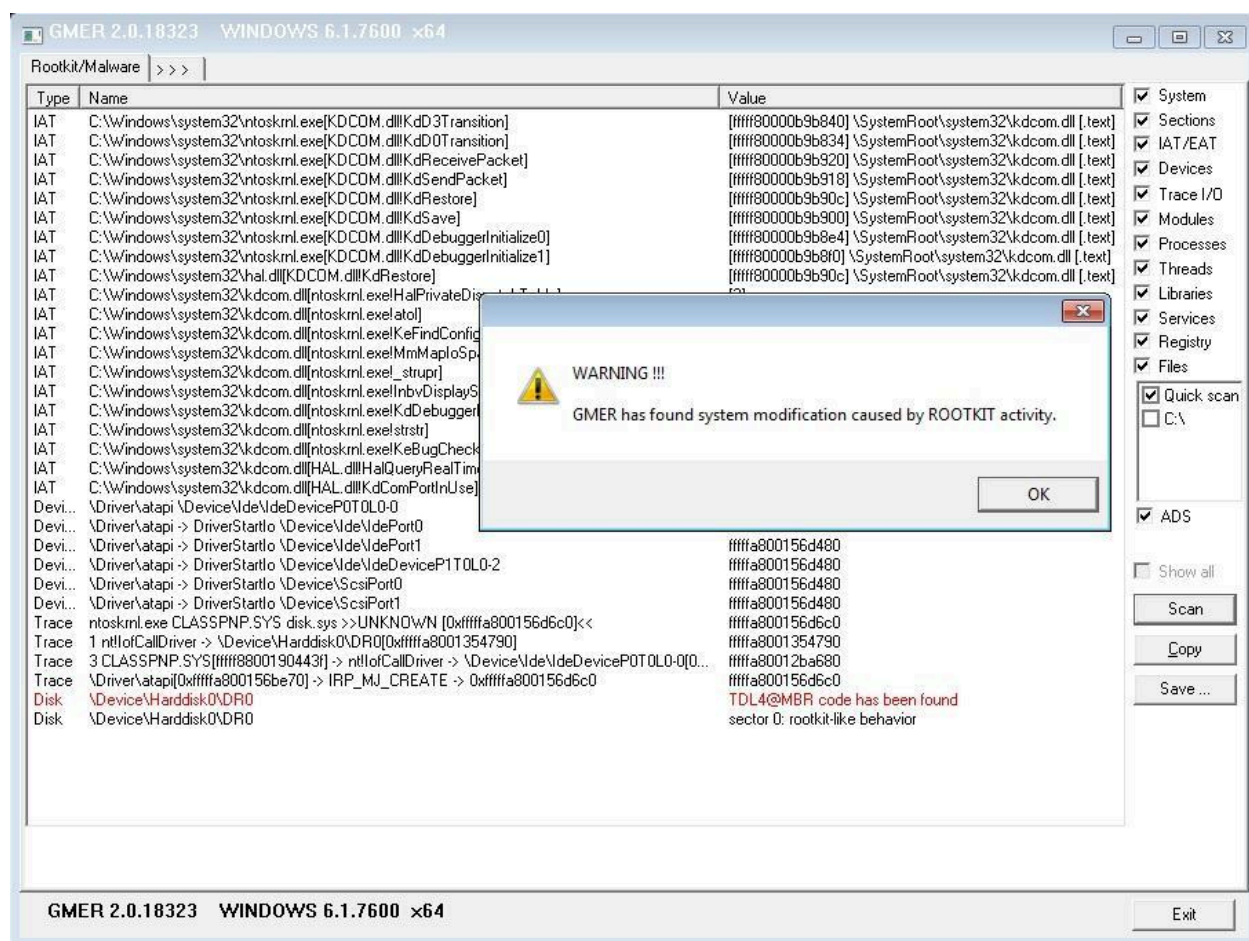
- hidden registry keys

- drivers hooking SSDT

- drivers hooking IDT

- drivers hooking IRP calls

- inline hooks



If you try to kill a malware process and it stubbornly refuses to die, right-click that process and select Terminate (or use the Terminator plugin). The Terminator will try a variety of creative and aggressive methods of disrupting that process until it goes away. If you terminate a process and it simply reappears again, look for other malware processes or device drivers working as a team.

You may need to right-click and Suspend each team member process before terminating all of them. If the malware process reappears after rebooting, you'll need to investigate how it is persisting in the registry or file system using tools like Sysinternals AutoRuns.

Idle, System, DPCs and Interrupts

System Idle Process, System, Deferred Procedure Calls (DPCs) and Interrupts are not actually full processes in that they do not have virtual address spaces of their

own, but they do consume CPU cycles. (If you don't see DPCs on a separate line in Process Hacker > Hacker menu > Options > Advanced tab > uncheck 'Enable cycle-based CPU usage'.)

Idle, System, DPCs, Interrupts

- ▶ **These are not real processes:**
 - They do not have virtual address spaces.
 - But they do consume CPU cycles.
 - Idle and System created during kernel initialization by process manager functions from `ntoskrnl.exe`.
- ▶ **Deferred Procedure Calls (DPCs):**
 - Queued up kernel function calls at IRQL 2.
 - Used by device drivers for I/O requests, as timers, thread context switching, and other kernel tasks.

There is one idle thread per CPU core. Idle threads aren't actually idle, they perform useful functions such as enabling/disabling interrupts, dispatching software interrupts for DPCs, dispatching some of the other threads waiting for a CPU, and granting kernel debuggers access to the OS. Idle threads run in a continuous loop as they check for work to perform. Any other thread can preempt an idle thread.

During kernel initialization, the first quasi-process created is the Idle "process" (it's not a real full-blown process) with PID = 0. Kernel bootstrapping proceeds later to create the System "process" (also not a full real process) and an Idle thread dispatches the first System thread. Idle is shown as the parent of the System process, but this is a fictional artifact of the kernel initialization sequence (winload.exe copies `ntoskrnl.exe` into memory and hands control to it, `ntoskrnl.exe` contains the functions collectively called the "Process Manager", and it's the Process Manager which creates both Idle and System).

Deferred Procedure Calls (DPCs) are requests for kernel functions to be executed on behalf of device drivers and other kernel components. Each CPU core has a queue of DPCs which are not executed immediately (they are "deferred") but only after the CPU has finished its current higher-priority task. Device drivers use their Interrupt Service Routines (ISRs) to create DPCs to manage I/O requests. Kernel components use DPCs to handle things like timer expirations and performing context switches from one thread to another. DPCs execute at IRQ Level 2, so a DPC can interrupt any normal thread (which run at IRQ Level 0), but another hardware interrupt (IRQ Level 3-15) can interrupt a DPC.

Hal.dll imposes an IRQ Level scheme: IRQ Level 0 for user-mode threads ("passive", "low" or "normal" priority), IRQ Level 1 for Asynchronous Procedure Calls (APCs), IRQ Level 2 for DPCs ("dispatch" priority), IRQ Level 3+ for hardware interrupts. Only non-paged memory can be accessed at IRQ Level 2 or higher because page faults themselves are handled at IRQ Level 2. Threads have their own priorities (0-31), but a higher IRQ Level thread can always preempt a lower-IRQ Level thread, no matter what their relative thread priorities are.

In Process Hacker, the Interrupts includes the CPU time spent processing hardware and software Interrupt Service Routines (ISRs). Often, an ISR will simply create a DPC to do the real work, but at a lower IRQ Level. CPU utilization for Interrupts should be low on average

Idle, System, DPCs, Interrupts

- ▶ **These are not real processes:**
 - They do not have virtual address spaces.
 - But they do consume CPU cycles.
 - Idle and System created during kernel initialization by process manager functions from `ntoskrnl.exe`.
- ▶ **Deferred Procedure Calls (DPCs):**
 - Queued up kernel function calls at IRQL 2.
 - Used by device drivers for I/O requests, as timers, thread context switching, and other kernel tasks.

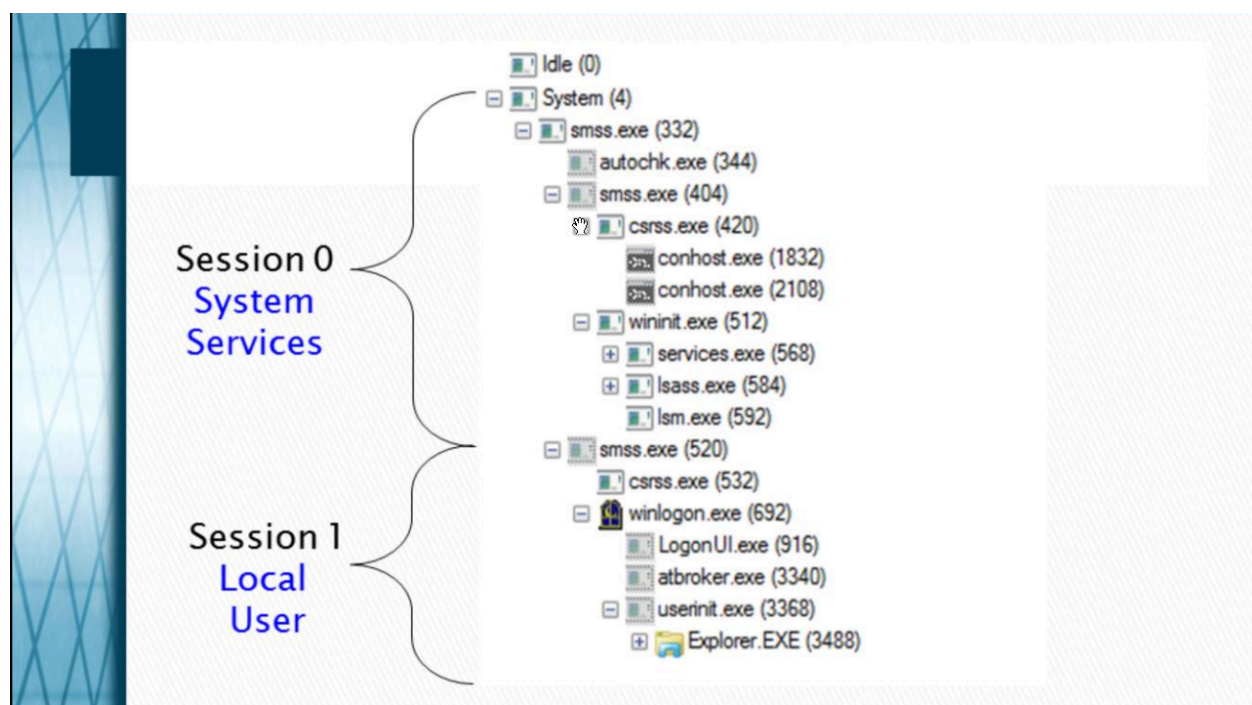
SMSS.EXE - The Session Manager

The Session Manager (`smss.exe`) is the first user-mode process created after kernel initialization. It is responsible for loading the rest of the registry, building the list of environment variables to be inherited by other processes, performing post-boot delayed file delete/rename changes, opening additional swap files, creating symbolic links for DOS device names like LPT1 and COM1, loads known DLLs into shared memory sections, and other tasks. More importantly, it also launches `csrss.exe`, `wininit.exe` and `winlogon.exe`.

The first session created by `smss.exe` is for operating system services (session 0), not for human users. When session 0 is created, `smss.exe` copies itself into that new session, starts `csrss.exe` and `wininit.exe` (plus other tasks), then `smss.exe` exits (the original instance under System still runs). This is why in tree view you see `csrss.exe` and `wininit.exe` in session 0 without any parent processes, that is to say, why the session 0 `csrss.exe` and `wininit.exe` processes are left-aligned without any indentation.

New sessions are created by smss.exe by its calling certain functions in the kernel's memory manager.

When a new user session is created (session 1 and greater), smss.exe copies itself into that new session, starts csrss.exe and winlogon.exe in that session (plus other tasks), then the smss.exe in the new session exits. This is why csrss.exe and winlogon.exe in session 1 are shown in the process tree without any parents. Remember, wininit.exe for session 0, winlogon.exe for user sessions greater than 0.



What Is A "Session" Anyway?

Loosely speaking, your session is all of the processes you own, plus your desktop. Strictly speaking, a session is a small area of kernel address space which is not mapped into the virtual address spaces of all processes, but only into some, namely, into the processes originally created for the OS (session 0) or a user (session 1 or greater) by smss.exe and the child processes of these parents.

This area of kernel memory is called "session space" and is shared by all the processes in the same session. Each session contains its own copy of win32k.sys and associated video drivers, a per-session directory in the object manager's namespace (\Sessions*), and a per-session paged pool of memory. The

win32k.sys driver is that part of the Windows subsystem which manages the output of GUI windows on the desktop (USER, GDI and DirectX), gathers input from the keyboard and mouse, and sends input messages from the mouse or keyboard to the correct process.

The processes in a session share the same station and desktop(s)

Sysinternals Process Monitor Tree

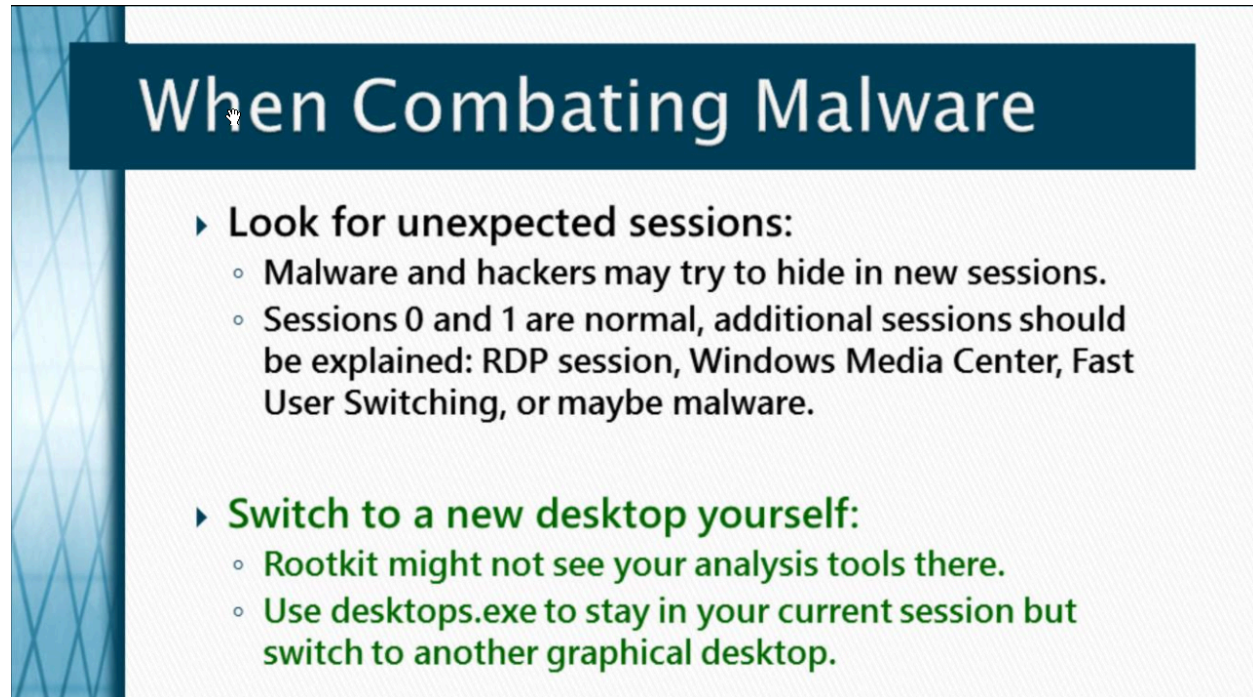
This screenshot was made with Sysinternals Process Monitor after configuring that tool for boot logging and then rebooting the computer (<http://www.microsoft.com/sysinternals>). The screenshot shows parent processes even if those processes are no longer running at the time the screenshot was made.

It is an historical snapshot of parent-child process relationships. The processes no longer running when the screenshot was made are shown with greyed-out icons. The numbers in parentheses are their Process ID numbers (PIDs). The screenshot is from a computer running 64-bit Windows 7 Ultimate.

To see this yourself, in Process Monitor pull down the Options menu > Enable Boot Logging. Reboot the computer. Open Process Monitor again, and when prompted, save the boot log to a file. Then pull down the File menu > Open > load the boot log > Tools menu > Process Tree.

The screenshot shows how smss.exe creates session 0 for csrss.exe and wininit.exe, then smss.exe exits. In session 0, wininit.exe then runs services.exe,

lsass.exe and lsm.exe. For session 1, smss.exe runs csrss.exe and winlogon.exe, then winlogon.exe spawns logonui.exe and userinit.exe



When Combating Malware

- ▶ **Look for unexpected sessions:**
 - Malware and hackers may try to hide in new sessions.
 - Sessions 0 and 1 are normal, additional sessions should be explained: RDP session, Windows Media Center, Fast User Switching, or maybe malware.
- ▶ **Switch to a new desktop yourself:**
 - Rootkit might not see your analysis tools there.
 - Use desktops.exe to stay in your current session but switch to another graphical desktop.

When Combating Malware

Sessions 0 and 1 are normal, but additional sessions should be explained. Additional sessions can come from RDP connections, remote Windows Media Center sessions with an extender (like an Xbox), and Fast User Switching on shared computers, but beyond these examples any unexpected sessions should be investigated. Because window stations and desktop objects have Access Control Lists (ACLs), it's possible that a rootkit might try to harden these ACLs and then hide the entire station.

A window station (or "winstation") contains a clipboard and one or more desktops.

Each window station is a securable object, hence, each has an access control list. When a window station is created, it is associated with the calling process and assigned to the session of that process. The interactive window station, named "Winsta0", is the only window station which can display a graphical user interface

and receive user input through one of its desktops. Each RDP remote desktop session has its own Winsta0.

A window station can have one or more desktop objects associated with it. The desktops associated with the interactive window station, Winsta0, can be made to display a user interface and receive user input, but only one of these desktops can be seen by the user at a time (this is called the "active desktop"). Windows messages can be sent only between processes that are on the same desktop. Each process which depends on the Windows subsystem is associated with a desktop, station and session (smss.exe and csrss.exe help implement the Windows subsystem, so they do not depend on it of course, so wininit.exe is the first).

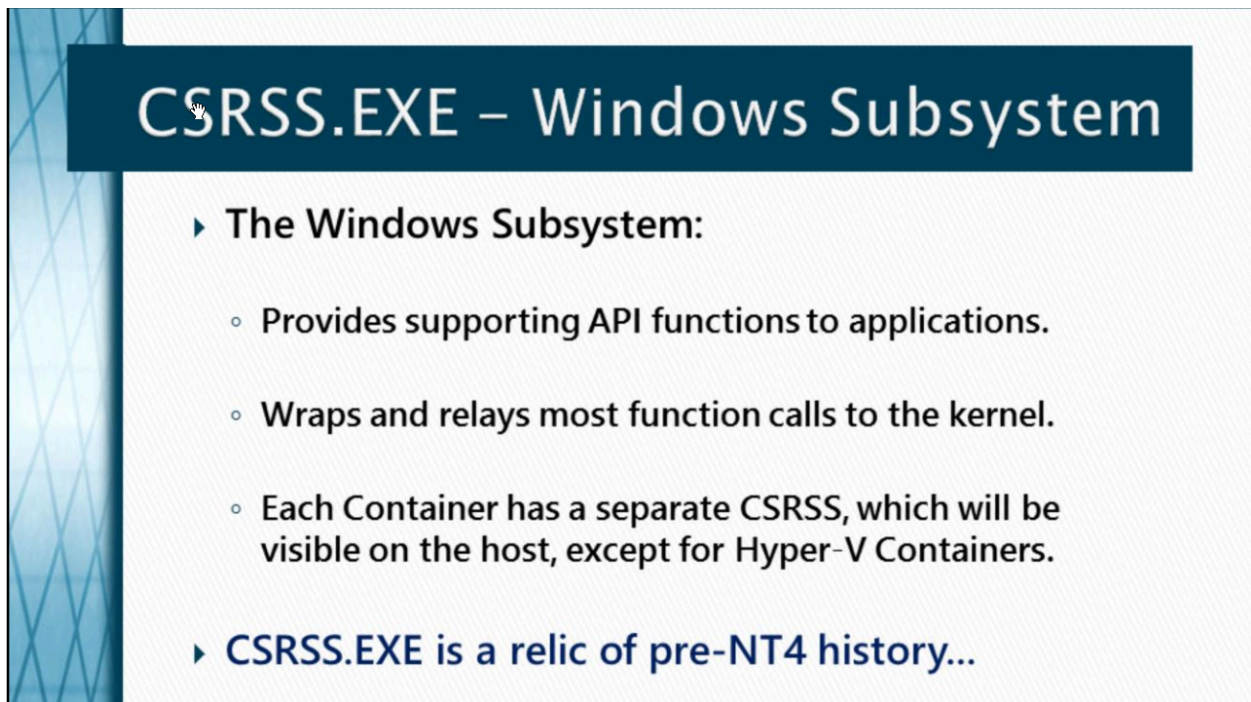
Get the Sysinternals desktops.exe tool to play around with multiple desktops in your interactive window station. When you run this tool, you'll get a new icon next to your clock in the status area of the taskbar. Clicking this icon shows four different desktops, including the current one, to which you can switch. Each desktop has its own taskbar, program windows and "desktop" as we normally use that word in everyday conversation. Only one desktop can be active at any given time, but because they all share the same window station, there is one shared clipboard.

If you initiate logoff in one of the desktops, you'll log off from all of them because you are really exiting your entire session.

If you are combating malware which is blocking your forensics tools, you might try switching to another session or desktop in order to try to hide *from it*. For example, if you run the Sysinternals desktops.exe utility, you'll stay in your current session and window station, but you'll be able to switch among four desktops.

Each desktop has its own graphical programs and Windows GUI messages can only be sent (or intercepted) between processes on the same desktop. Or with RDP or Fast User Switching, you could log on with a new session entirely, though still with the default desktop. So if you can't run your favorite AV or forensics tools in the

default desktop because of rootkit monitoring, try running those tools in a new desktop instead



CSRSS.EXE – Windows Subsystem

- ▶ **The Windows Subsystem:**
 - Provides supporting API functions to applications.
 - Wraps and relays most function calls to the kernel.
 - Each Container has a separate CSRSS, which will be visible on the host, except for Hyper-V Containers.
- ▶ **CSRSS.EXE is a relic of pre-NT4 history...**

CSRSS.EXE - The Windows Subsystem Process

A "subsystem" provides part of the environment in which processes can run. A subsystem makes various operating system functions available for processes to call as needed, such as for accessing the file system, using the network, or drawing graphical windows.

The Windows subsystem loads DLLs into the address space of a process which can provide these functions themselves or relay these function calls down to the kernel or to other supporting processes. There are many DLLs that implement the Windows subsystem (kernel32.dll, advapi32.dll, user32.dll, gdi32.dll) and an additional one for the POSIX subsystem (psxdll.dll).

The Windows subsystem also includes csrss.exe as a helper process. Prior to Windows NT 4.0, csrss.exe was very important because it handled drawing graphical windows, but since then this functionality has been moved into the win32k.sys kernel-mode driver for performance reasons. Drawing text-based consoles used to be handled by csrss.exe too, but with Windows 7 and Server

2008-R2, even this functionality has been moved into a different process (conhost.exe) for security reasons. Spawning new processes and threads, mapping drive letters, handling much of the shutdown procedure, loading win32k.sys into the kernel at boot-up, and creating temp files is still managed by csrss.exe, but not much else of importance anymore, so csrss.exe is mainly an historical relic maintained for backwards compatibility. A future Windows version may eliminate it entirely.

On Server 2016 and later, each Docker-style Container will have its own separate csrss.exe process, and these csrss.exe processes will be visible in the list of processes on the Container host server. The exception is when using a Hyper-V Container, in which case the csrss.exe process for the Container will not be visible in the list of processes on the host because the Container is being "wrapped" or protected by Hyper-V



WININIT.EXE - The Windows Initialization Process

After the kernel is initialized and smss.exe is running, smss.exe creates session 0, copies itself into session 0, launches csrss.exe, launches wininit.exe, then that session-0 copy of smss.exe exits. At this point, csrss.exe and wininit.exe are still

running in session 0.

The main job of wininit.exe is to

- 1) launch services.exe to start services and more device drivers,
- 2) launch lsass.exe to handle security, and
- 3) launch lsm.exe to do some of the session management. (Note: In Windows XP and Server 2003, lsass.exe was launched by winlogon.exe instead.)

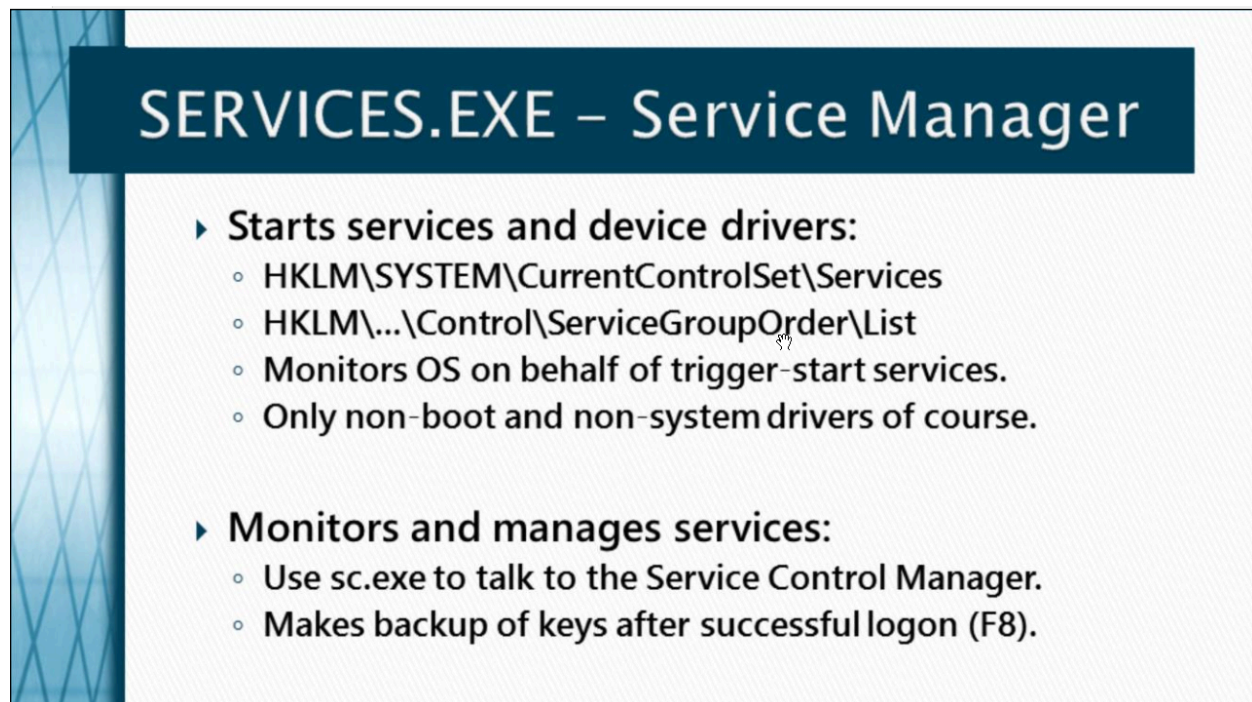
LSM.EXE or LSM.DLL - The Local Session Manager

On Windows Vista and 7, the Local Session Manager (lsm.exe) creates, destroys and manipulates sessions, mainly by directing smss.exe to do so. On Windows 10 and later, this process was converted into a regular shared-svchost.exe DLL service (lsm.dll). There is session 0 for services and session 1 is typically for the interactive user logon, but additional sessions may exist for the sake of Fast User Switching, Windows Media Center Extenders, and Remote Desktop Protocol (RDP) connections, such as RDP connections to a Server 2008-R2 machine running Remote Desktop Services.

When an RDP user connects, the LSM directs smss.exe to create a new session (including csrss.exe, winlogon.exe and logonui.exe). If there is already a desktop for that user, then the RDP connection is linked to that desktop and its currently-running processes in its current session. If a new desktop is required, then userinit.exe and explorer.exe run like normal, but in another new session. An exception to this is when an RDP connection is made for Remote Assistance, in which case the RDP connection goes to the Remote Assistance Server (raserver.exe) and a new session is not created by smss.exe; instead, the RDP

stream is simply linked to the desktop of the user being helped, either for passive viewing or for shared control of the mouse and keyboard.

The LSM never has any child processes of its own. Because multiple smss.exe processes can be spawned concurrently, it improves the performance of servers running Remote Desktop Services (formally known as "Terminal Services") when responding to many simultaneous logon requests.



SERVICES.EXE – Service Manager

- ▶ **Starts services and device drivers:**
 - HKLM\SYSTEM\CurrentControlSet\Services
 - HKLM\...\Control\ServiceGroupOrder\List
 - Monitors OS on behalf of trigger-start services.
 - Only non-boot and non-system drivers of course.
- ▶ **Monitors and manages services:**
 - Use sc.exe to talk to the Service Control Manager.
 - Makes backup of keys after successful logon (F8).

SERVICES.EXE - The Service Control Manager

The job of services.exe is to read keys in the registry representing services and device drivers (under HKLM\SYSTEM\CurrentControlSet\Services), then run these processes and load these drivers according to their dependency groups (as defined in HKLM\SYSTEM\CurrentControlSet\Control\ServiceGroupOrder\List). Boot and system drivers are not loaded at this time, they are already loaded.

When a service does not have a "Group" value to identify its dependency group, it is run afterwards.

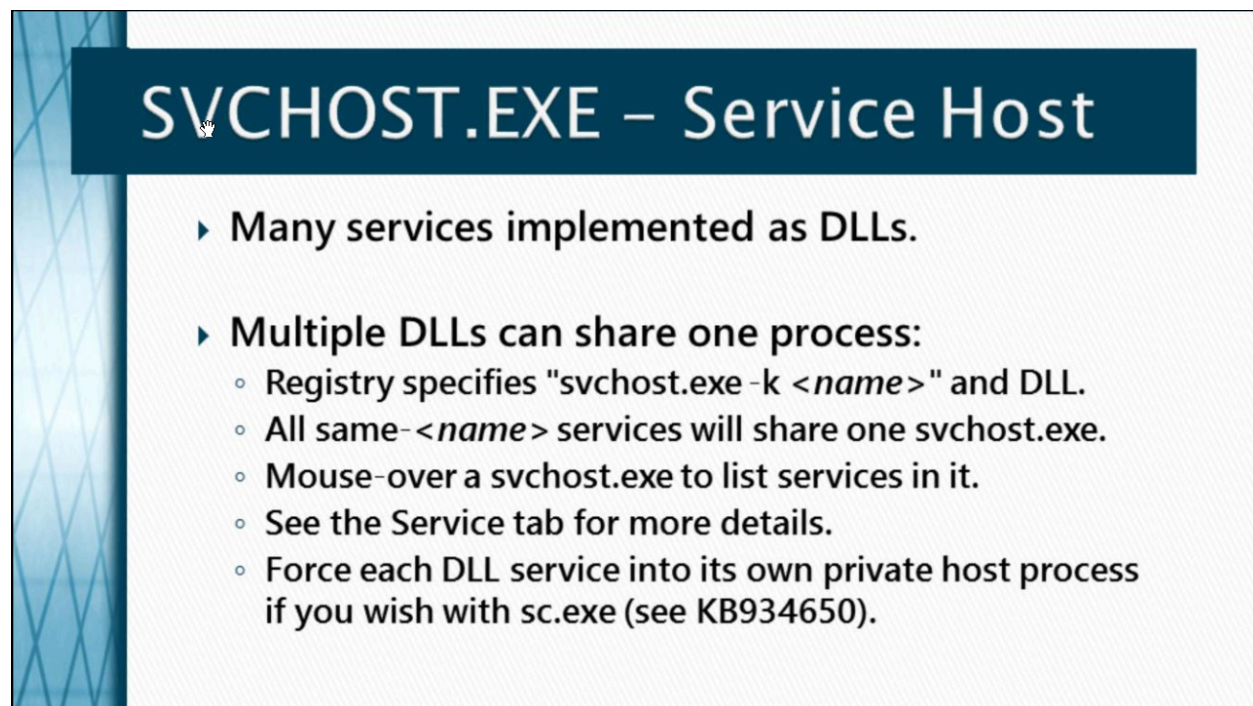
By default, non-interactive services running as Local System are attached to an invisible window station named "Service-0x0-3e7\$" in session 0; when these

services are run interactively, they still run in session 0, but are attached to the Winsta0 station instead.

Windows services are not just fire-and-forget processes: services.exe maintains an in memory database of service information, such as error state and recovery actions.

Use the built-in sc.exe tool to interact with services.exe.

After a successful interactive user logon, services.exe will also make a backup copy of the keys under HKLM\SYSTEM\CurrentControlSet\Services known as the "last known good configuration", which can be restored at the next reboot with the F8 key



SVCHOST.EXE – Service Host

- ▶ Many services implemented as DLLs.
- ▶ Multiple DLLs can share one process:
 - Registry specifies "svchost.exe -k <name>" and DLL.
 - All same-<name> services will share one svchost.exe.
 - Mouse-over a svchost.exe to list services in it.
 - See the Service tab for more details.
 - Force each DLL service into its own private host process if you wish with sc.exe (see KB934650).

SVCHOST.EXE - The Generic Service Host Process

Many services are implemented as DLLs which can be loaded together into a generic shared svchost.exe process launched by services.exe. The registry entry for the service will include a ServiceDll value to identify the DLL to be used and a path to "svchost.exe -k <name>" to load it. All DLL-based services with the same

"<name>" will share the same svchost.exe process; for example, both the DHCP Client and Windows Event Log services are DLL-based, and they share a single svchost.exe because both are launched with the "svchost.exe -k LocalServiceNetworkRestricted" command.

Hold your mouse pointer over a svchost.exe process to see a pop-up list of the services contained within it.

Sharing a single host process is nice for conserving OS resources, but not great for troubleshooting or analyzing infected services. You can force a DLL-based service into its own svchost.exe process with the sc.exe command (see KB251192 and KB934650).

Modules

- ▶ **Module = DLL, EXE, or Resource File.**
- ▶ **Inspect a DLL or EXE module to show:**
 - ASLR and DEP compatibility.
 - Sections in the PE format of the file.
 - Imports/exports (functions it calls/provides).
- ▶ **System modules include SYS drivers.**
- ▶ **Inject or unload modules yourself.**
- ▶ **Right-click > Send To > VirusTotal.com**

Modules And Digital Signatures

The modules of a process are the Dynamic Linked Libraries (DLLs), resource files and the original executable image (EXE) loaded into the memory address space of that process. The modules loaded into the System "process" at the top includes

drivers (*.sys) and ntoskrnl.exe. Remember that System is not really a process. In the Modules tab of a process, right-click the column headers, select Choose Columns, and add every possible column. Make sure to add the "Verification Status" and "Verified Signer" columns in particular, perhaps moving them near the top of the list.

When combating malware, modules which fail their digital signature tests, or which have no signature, deserve additional analysis. You can add the same columns in the process tree view and then hide processes with verified signatures (View menu > Hide Signed Processes).

If you right-click a DLL or EXE module, you can select Inspect to show its Portable Executable (PE) sections, ASLR compatibility (Dynamic Base), DEP compatibility (NX Support), the functions it calls from other modules (Imports tab), and the functions it makes available to other modules (Exports tab). You can also add ASLR and DEP as columns.

You can also unload a module when you right-click it (which usually kills the process) or load a DLL or EXE into the process (go back to tree view, right-click the process > Miscellaneous > Inject DLL). When you inject a DLL, the operating system runs the DllMain() function within it automatically.

Another great feature for fighting malware is the ability to right-click a binary on the Modules tab (or in the main process list) and Send To web sites which perform free AV scanning with many different scanners, e.g., VirusTotal.com. This is a really fast and convenient way to at least do basic checks on suspicious binaries, especially when they are not digitally signed and not found on other similar computers in your organization.

Memory and Strings

The 4GB virtual address space of an x86 32-bit process is typically divided into two regions: user space (0x00000000-0x7FFFFFFF) and system space (0x80000000-

0xFFFFFFFF). The 16EB virtual address space of an x64 64-bit process is also divided into user space (0x0000000000000000-0x7FFFFFFFFFFFFFFF) and system space (0x8000000000000000-0xFFFFFFFFFFFFFFFF), though not all of this is addressable today due to software and hardware limitations.

Memory and Strings

- ▶ **View or edit virtual memory pages!**
- ▶ Dump areas of memory to a file.
- ▶ Change protection bits.
- ▶ Free or decommit memory.
- ▶ **Dump in-memory strings:**
 - Filter with regular expressions.
 - Save strings to clipboard or to text file.

Virtual-to-Physical Address Mapping

The user space is private to that process, except for shared memory sections; and the system space is common to all processes, except for a small area of system space called "session space", which is only common across processes in the same smss.exe session.

An address in virtual memory can be common across processes because each committed page of virtual memory is backed by or "mapped to" a page of real physical memory, and two virtual memory addresses (like in two processes) can be mapped to the same physical address. Hence, a DLL can be loaded into physical memory once, then mapped into the system space of all processes using virtual addresses, saving a great deal of physical memory.

Kernel Mode vs. User Mode

The CPU can be in "kernel mode" or "user mode". The CPU must be in kernel mode in order to read/write memory in system space. The CPU can be in either kernel mode or user mode to read/write memory in user space. When in kernel mode, the CPU can execute any instruction it supports and can read/write any memory anywhere.

Malware running in kernel mode can do anything. The CPU and OS tightly control the switching of the CPU's mode in order to try to maintain security.

Reserved vs. Committed Memory

A process can "reserve" an area of virtual memory, which means those addresses have been set aside for possible future use, or the process can "commit" an area of memory, which means those virtual addresses have actually been mapped to physical memory. Reserved virtual memory does not consume any physical memory, but committed virtual memory does consume physical memory.

Free, Mapped, Image or Private Memory

"Free" virtual memory is neither reserved nor committed, but can be so as needed. "Mapped" memory is either part of a heap, a section of memory shareable with other processes, or all/part of the contents of a file. "Image" memory contains the contents of file, usually a binary, cached in memory. "Private" memory is for the private use of the process, such as for a thread stack. (See the Sysinternals vmmap.exe and rammap.exe tools for a better view of memory use.)

Protection Bits

The page tables in system space, which do all the virtual-to-physical address mappings, also contain per-page protection bits, similar to permissions. An area of memory can be marked as read-only (R), write (W), execute (X), some combination of these, plus a few other markings used internally for housekeeping. Finding mapped, committed memory marked as RWX could be interesting,

especially if a dump of the region began with "MZ" and "This program cannot be run in DOS mode" (Stuxnet did this, for example).

Memory Tab in Process Hacker

The Memory tab in Process Hacker permits several useful actions:

- View or edit areas of virtual memory (right-click > Read/Write Memory).
 - Save an area of memory to a binary dump file (right-click > Save).
 - Change the protection bits, with limitations (right-click > Change Protection).
 - Free or decommit memory (right-click > Free or Decommit). Decommitted memory is still reserved.
-
- Dump the ASCII and/or Unicode strings found in the process virtual address space, then filter these strings using a regular expression pattern (Strings button > OK > Filter button). Strings can be copied to the clipboard or saved as a TXT file. For combating malware, the ability to dump and search strings from in-memory data is very important. The strings of a process are helpful in identifying malware types and analyzing their purposes and capabilities. If you wish to script the analysis of strings from the command line, get the Sysinternals strings.exe tool.

When Combating Malware

- ▶ **Malware often persists through reboots as a service or device driver:**
 - Check digital signature status of modules.
 - Examine registry keys used by services.exe.
 - Research hashes of modules (Bit9 FileAdvisor).
 - Look for unusual imported/exported functions.
 - Dump and search process strings.
 - With packed processes, examine memory regions.
 - SANS' *Windows Memory Forensics* (FOR526).

When Combating Malware

If you are a piece of malware, you've got a persistence problem: how to survive reboots? You'll have to somehow modify the hard drive, BIOS or device firmware, because main memory is reset each time the power button is pressed. A common way to survive reboots (and get loaded automatically with elevated privileges too) is for the malware to get loaded by or as a service or device driver. By modifying an existing binary which is already loaded by services.exe, the malware can survive reboots without changing the registry.

Or by adding a new binary and editing the registry, the malware can again be run by services.exe automatically.

So, when combating malware, pay special attention to the child processes of services.exe and the associated registry keys. In particular, it's common for malware

to load a DLL into a svchost.exe because there are so many of these processes. An infected machine might also have a malicious device driver visible in the Modules tab of the System "process", like Stuxnet, or rootkit tricks might be in use to hide them.

Make sure to check the digital signature status of modules and device drivers, confirm that the hashes of these binaries are known to be good (like with <http://fileadvisor.bit9.com>), examine the imported and exported functions of suspicious modules (look up the function names in Google/Bing), examine the strings of a process (look for strange URLs, file system paths, embedded passwords, or other "hinky" data), and with packed processes in particular you may need to examine raw memory regions (these regions can be dumped to .bin files also).

If these tasks seem difficult (because they are) you might consider taking *Reverse Engineering Malware* (FOR610), *Windows Memory Forensics* (FOR526) or *Advanced Forensics* (FOR508).

LSASS.EXE – Local Security

- ▶ **Shared process for security services:**
 - Authenticates users and creates security tokens:
 - Security Accounts Manager (local user accounts)
 - Active Directory (on a domain controller)
 - Netlogon (when joined to a domain)
 - Enforces password policies and other restrictions.
 - Writes to the Security event log.
 - Maintains the Netlogon secure channel to a controller.
 - Involved in many cryptographic operations.
 - **A favorite target of hackers and malware!**

LSASS.EXE - The Local Security Authority Subsystem

Another process launched by wininit.exe is lsass.exe, which is a special shared-host process for security-related services like Active Directory on a domain controller (ntdsa.dll, kdcsvc.dll), Netlogon (netlogon.dll), Security Accounts Manager for local accounts (samsrv.dll), and Crypto Next Generation Key Isolation (keyiso.dll).

Many critical operations are handled by lsass.exe, such as user authentication, enforcing password policies, creating handles to new Security Access Tokens (SATs), writing to the Security event log, and maintaining the Netlogon RPC secure channel with a domain controller. Not surprisingly, this process is the target for many attacks, such as DLL-injections to dump password hashes or to perform pass-the-hash impersonation attacks.

Incidentally, the Stuxnet worm created multiple lsass.exe processes at various times during its lifecycle, precisely because this process name is well-known and trusted.

WINLOGON.EXE – User Logon

- ▶ Controls the logon desktop, locked desktops, and secure screen savers.
- ▶ Intercepts the **Ctrl-Alt-Del** keystroke and runs logonui.exe for the user:
 - Communicates with lsass.exe to authenticate the user, loads the user's profile, runs userinit.exe, which runs explorer.exe (among other things) and exits.

WINLOGON.EXE - The Windows User Logon Process

Right after boot-up, the original smss.exe creates session 1, copies itself into session 1 as a new process, launches csrss.exe and winlogon.exe, then that smss.exe exits, leaving csrss.exe and winlogon.exe still running. Next, winlogon.exe creates a window station (Winsta0) and displays the interactive desktop, which means winlogon.exe is now in control of the local keyboard, mouse and monitor (and running as Local System).

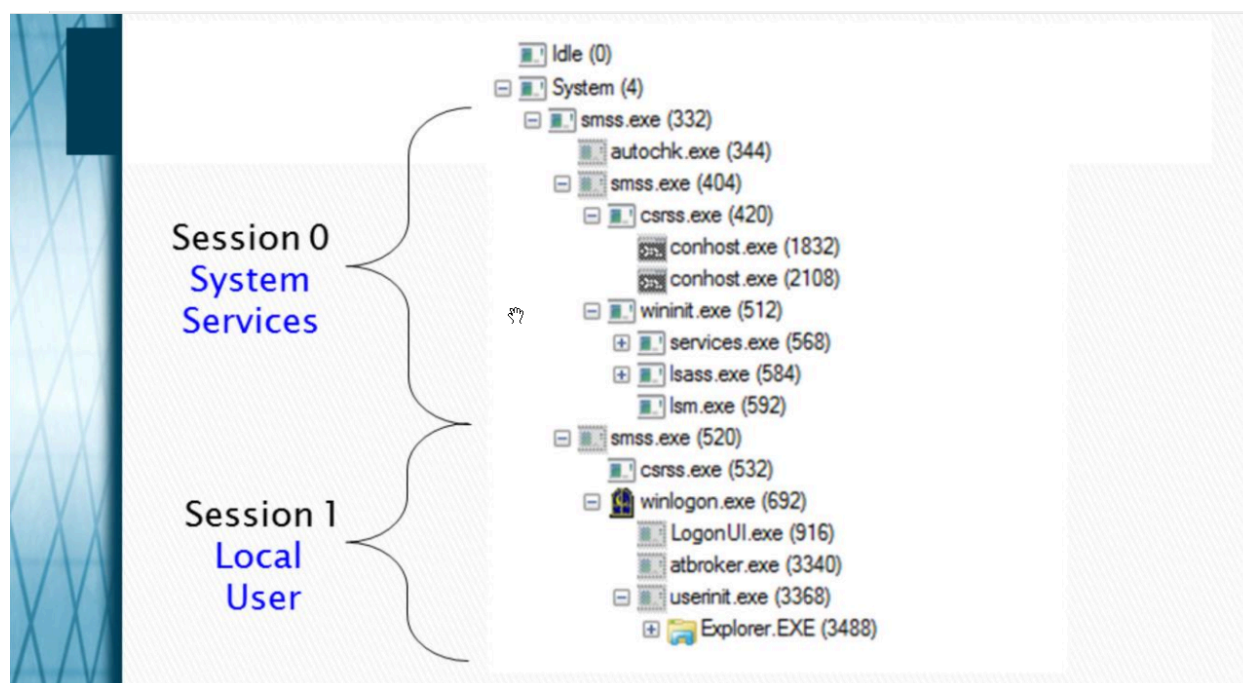
Whenever a user hits Ctrl-Alt-Del, winlogon.exe switches to another desktop and launches a special program, logonui.exe, to interact with the user. The user may be logging on initially, (un)locking the desktop, changing her password or some other task, but the user is interacting with logonui.exe on a special desktop, not winlogon.exe on the default desktop. This special desktop in which logonui.exe runs is called the "secure desktop" because its permissions only allow winlogon.exe to manage it, thus hopefully preventing trojans from capturing

keystrokes. When a user locks her desktop, it's actually winlogon.exe doing the locking.

When authenticating, logonui.exe loads DLLs called "credential providers" which can handle the password, smart card or, with a third-party provider, biometric information, to authenticate against the local SAM database, Active Directory, or some other third-party authentication service. The resulting credentials are given to winlogon.exe, which calls functions within lsass.exe to do the actual authentication checking. If the authentication succeeds, lsass.exe gives winlogon.exe a handle to a Security Access Token (SAT) for the user, then winlogon.exe uses that token to set permissions on the interactive desktop (not the secure desktop) for the user's benefit, loads the user's profile, launches userinit.exe with that user's token, and then userinit.exe typically runs explorer.exe.

Once userinit.exe finishes its other tasks, such as running logon scripts, it exits, leaving explorer.exe still running (which is why explorer.exe doesn't have a running parent in tree view).

Incidentally, when user environment debug logging is enabled (KB221833), it is mainly winlogon.exe writing to the userenv.log file. And while userinit.exe and explorer.exe are the typical initial processes, registry edits permit different or additional processes to be launched for the user instead (for example, when an Xbox opens a Windows Media Center session on a remote computer as an extender, userinit.exe runs mcrmgr.exe instead). Finally, atbroker.exe helps users with disabilities navigate winlogon.exe's desktops and logonui.exe's prompts



Sysinternals Process Monitor Tree

This screenshot was made with Sysinternals Process Monitor after configuring that tool for boot logging and then rebooting the computer (<http://www.microsoft.com/sysinternals>). To see this yourself, in Process Monitor pull down the Options menu > Enable Boot Logging. Reboot the computer. Open Process Monitor again, and when prompted, save the boot log to a file. Then pull down the File menu > Open > load the boot log > Tools menu > Process Tree. The screenshot shows how smss.exe creates session 1 for csrss.exe and winlogon.exe, smss.exe exits, then winlogon.exe spawns logonui.exe and userinit.exe.

Credentials are collected by logonui.exe, given to winlogon.exe, which checks them with lsass.exe, then winlogon.exe runs userinit.exe to create the user's desktop environment for explorer.exe to create the Start menu, taskbar and desktop icons.

Explorer.exe is run by default as the user's initial process, but actually any process could be launched first instead.

Security Access Token (SAT)

- ▶ Every process has a SAT to identify it.
- ▶ **The SAT includes:**
 - SID of the user.
 - SIDs of the groups to which the user belongs.
 - The user's privileges.
 - The user's logon session number.
- ▶ Enable, disable or remove privileges.
- ▶ Change the SAT's own permissions.

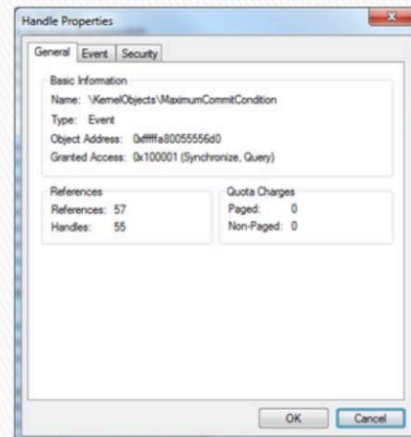
Security Access Token (SAT)

Every process and thread has an associated Security Access Token (SAT) to represent the identity under which that process or thread is running. The SAT includes the user's Security ID (SID) number, the SIDs of the groups to which the user belongs, the user's privileges, the session number, and other information. The SAT is created by lsass.exe (and the kernel's security reference monitor) when a user logs on, then it is attached to every process the user launches. Not only can you view SAT data on the Token tab of a process, you can also enable/disable/remove privileges (but not add them) and modify the permission on the token object itself.

When User Account Control (UAC) modifies the SAT in an un-elevated process, you can also see the original elevated SAT (the Linked Token).

Handles & Network Connections

- ▶ Each process has a table of handles to represent its access to particular objects.
 - System process shows the kernel handle table, only accessible from kernel mode.
- ▶ A handle includes a pointer to the target object and the process' access permissions.
- ▶ **You can forcibly close handles, change their permissions, and trigger handled events.**



Handles

Every process has an associated table of handles. A "handle" represents the access of a process to another object. Some of these objects exist only in the kernel, such as threads, tokens, sessions, ports and processes; while other objects are created by the kernel to represent "real" resources to which the OS controls access, such as files and registry keys. A handle includes a pointer to the target object, as well as a list of access permissions. Processes can acquire handles by creating new objects, requesting the object by name, inheriting the handle from parent processes, and other methods. Before a thread in a process can interact with an object, the process must obtain a handle. In the kernel, only the object manager can create handles, and it does so only after checking permissions with the security reference monitor.

On the Handles tab of a process, not only can you view handles, you can forcibly close them, change their permissions, and change their "Inheritable" and "Protect from Close" flags. If the handle is to an event, you can also trigger that event by hand and see what the process does -- it's fun! If you right-click a process in tree view and run the Terminator, you can try to close all of its handles in one shot (double-click CH1).

When you examine the handles of the System process, you are seeing a different handle table: the kernel handle table. Kernel handles are only accessible in kernel mode, but from within any process in kernel mode

Thread Stacks

- ▶ **Top of Stack**
- ▶ **Module Name**
- ▶ **Function**
 - Need symbols!
- ▶ **Return Offset**
- ▶ **Stack Bottom**

	Name
0	ntoskrnl.exe!KIDeliverApc+0x1c7
1	ntoskrnl.exe!KCommitThreadWait+0x3dd
2	ntoskrnl.exe!KeWaitForSingleObject+0x19f
3	ntoskrnl.exe!KSuspendThread+0x54
4	ntoskrnl.exe!KIDeliverApc+0x201
5	ntoskrnl.exe!KCommitThreadWait+0x3dd
6	ntoskrnl.exe!KeWaitForSingleObject+0x19f
7	win32k.sys!xxxRealSleepThread+0x257
8	win32k.sys!xxxSleepThread+0x59
9	win32k.sys!xxxRealInternalGetMessage+0x7dc
10	win32k.sys!xxxInternalGetMessage+0x35
11	win32k.sys!NtUserGetMessage+0x75
12	ntoskrnl.exe!KSystemServiceCopyEnd+0x13
13	USER32.dll!NtUserGetMessage+0xa
14	USER32.dll!GetMessageW+0x34
15	calc.exe!WinMain+0x1dca
16	calc.exe!_LDUnscale+0x1ea
17	kernel32.dll!BaseThreadInitThunk+0xd
18	ntdll.dll!RtlUserThreadStart+0x1d

Copy Refresh Close

Thread Stacks

Each process must have one or more threads. A thread has a stack (the process does not) and the thread is what actually performs work (the process is just a supporting environment for threads). The Threads tab of a process shows the current threads in that process. Each thread has a unique ID number (TID), a multitasking priority for scheduling by the dispatcher, and a starting address for a function in a module.

If you have symbols configured, the start address of a thread will be formatted like "*module !function*", where *module* is the DLL or EXE which asked the OS to create the thread and *function* is what was given to the OS as the first thing the thread

should do. If you don't have symbols configured, the start address will simply show "*module +number*" where *number* is a hex offset into the module of a function.

Manipulate Threads

If you right-click a thread, you can terminate, suspend or resume it. You can also cancel any synchronous input/output procedures it is carrying out, change its CPU affinity, multitasking priority, input/output priority, and permissions. If the thread manipulates a graphical desktop window, it will usually be shown with yellow highlighting, and you can right-click and select Windows to see which ones.

Inspect Stack

If you right-click a thread and select Inspect (or just double-click the thread), you will see the thread's stack. The bottom of the stack is historically the first function call, then later function calls appear as you move up the stack in the dialog box, each function calling the one above it, from bottom to top. When you get to the top of the stack, you'll see the most recent function call.

Strictly speaking, most threads begin life as a call to "ntdll.dll ! RtlUserThreadStart",

so that is what's shown at the bottom of the stack, but that wouldn't be very informative if shown on the Threads tab; the thread's interesting starting function (as shown on the Threads tab) is typically the second or third function call up from the bottom of the stack.

Line 14 in the screenshot looks like "USER32.DLL ! GetMessageW+0x34". The module is "USER32.DLL", the called function is "GetMessageW", and the return address offset is "0x34". What is the return address offset used for? Notice that line 14 shows that GetMessageW called another function above it named "NtUserGetMessage" in line 13. Later, when NtUserGetMessage finishes (line 13), control should return to the address of GetMessageW (line 14) and add 0x34

bytes to the stack pointer to clean up the stack so that the stack is the same as before the other function was called.

Misc Notes

Sometimes in a stack trace you'll see strange "_", "@" and "?" characters. These come from the "decorations" of the symbols in the symbol files. The decorations indicate the function's calling convention, the function's number of argument bytes, whether it is a symbol from a precompiled header (pch), an imported symbol from another module, linker-generated strings, or other low-level details too numerous to describe here.

You will often see function calls in the stack with a name like "*Wait*" since most threads spend most of their time just sitting and waiting for other events to occur. Each user-mode process has two stacks per thread: one user-mode, one kernelmode. Additionally, each CPU core has its own Deferred Procedure Call (DPC) stack too, but this is a different kind of stack.

When you see a thread referred to as "2c5.4ba", the "2c5" is the process ID number in hex and the "4ba" is the thread ID number in hex, hence, "2c5.4ba" refers to PID 709 with TID 1210.

When Combating Malware

- ▶ It's not always obvious that a given process is malicious or infected.
- ▶ Even when we know a process is bad, that doesn't tell us its purpose.
- ▶ We need to know what the malware is doing through its handles, network behaviors, call stack, etc.

When Combating Malware

When you are not certain whether a process is malicious or not, examining the SAT, handles, listening ports, network connections, and stacks of that process can help. Often, malware requires elevated privileges in order for the initial infection to succeed (such as having the Debug Programs privilege) or will seek to raise the privileges of a victim process; in either case, viewing the Security Access Token (SAT) of the process helps to understand what it can and cannot do.

The handles and networking characteristics of a process reveals what that process is interested in and what it may be doing. If a process appears to be one thing (a game, for example), but has handles, listening ports and live network connections which are very unusual for that type of program, it's a sign that the process may be malicious.

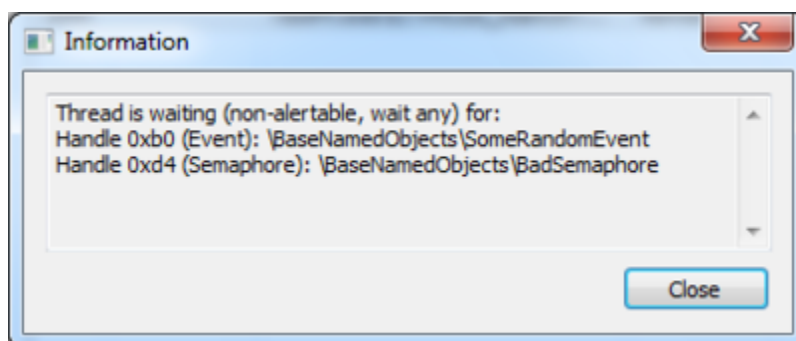
The imported and exported functions of a process can be very revealing, and so can the stack traces of the threads in a process for the same reasons. When viewing stack traces, we are getting a glimpse into what a process is doing at a moment in time. When a process appears to be one thing, again, but we see strange function calls for that supposed type of process, it's another sign that the process may be infected or malicious.

Plugin docs:

Wait analysis is a simple extension of stack tracing and is only useful on 32-bit systems. You should read the [thread stacks article](#) first.

To access wait analysis, right-click on a thread and choose **Analyze > Wait**. If the thread is waiting for something, Process Hacker will tell you what it is. Otherwise, it will display a message informing you that the thread is not waiting. You should familiarize yourself with some of the most common scenarios, explained below.

Waiting for objects

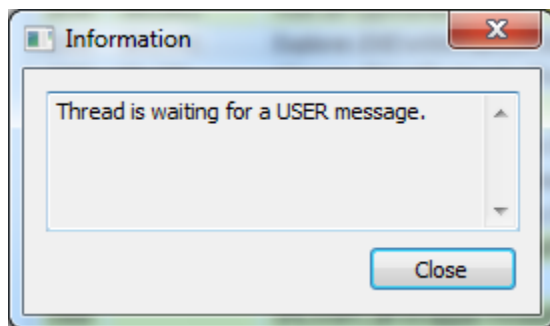


This occurs due to the NtWaitForSingleObject or NtWaitForMultipleObjects system call. Process Hacker gives you a list of objects which the thread is waiting on. If a program is hanging and you don't know what to do, try searching for one of these objects in

the **Find Handles and DLLs** (Ctrl+F) window. You can proceed from there; try removing/stopping the listed programs or modifying their behaviour.

If an object is unnamed, not to worry; note down the handle value (e.g. 0xb0, 0xd4), go to the Handles tab, make sure **Hide unnamed handles** is not ticked, and double-click the corresponding handle. You can copy the object address into the **Find Handles and DLLs** window to see what processes have a handle to that object.

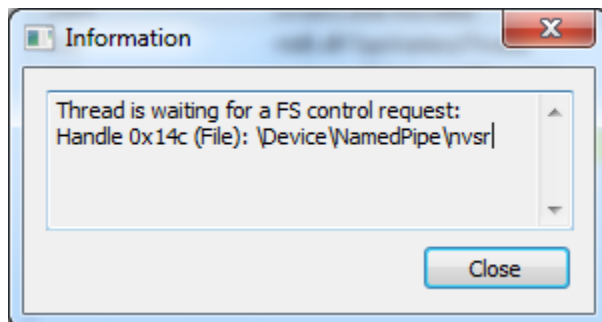
Waiting for a USER message



WAUserMessage.png (25.95 KiB) Viewed 9834 times

This usually occurs in GUI-based programs, in GUI threads. It can also be found in COM-based programs. For GUI programs it simply means the program is waiting for some user-related event such as mouse movement or keyboard activity. If it's a problem, try using Spy++ to monitor the messages that the process is receiving.

I/O



WAIo.png (21.82 KiB) Viewed 9834 times

The message could read waiting for a file or I/O control request. This type of wait means the program is waiting for I/O of some kind, although that doesn't tell you much. However:

- If the file name starts with **\Device\NamedPipe**, you can search for that using **Find Handles and DLLs** to see what process is on the other end of the pipe.
- If the file name represents an actual file (**C:\some\path\file.txt**), the program is trying to do synchronous I/O on it and is waiting for a particular operation to finish. There's no easy way to determine what that operation is. This can also happen with named pipes.
- If the file name starts with **\Device** and you see *I/O control* request, it's probably due to the program trying to communicate with a driver, and the driver is simply taking a long time to do whatever it's doing. The kernel-mode stack trace may help you immensely for this case.

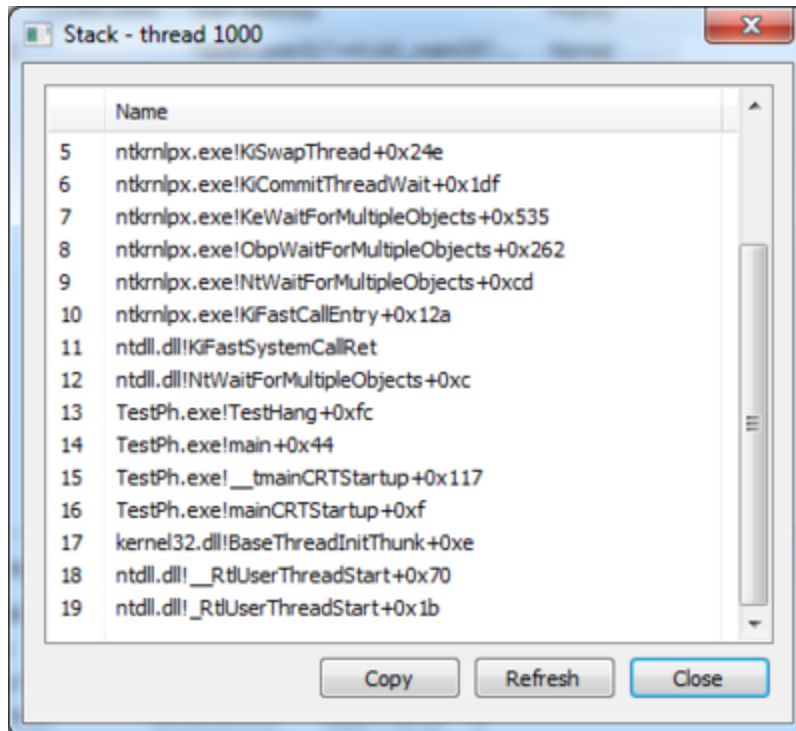
Threads can be examined in Process Hacker by double-clicking them or right-clicking and selecting "Inspect".

This opens a new window showing you the thread's stack trace, call stack, or whatever else you want to call it. The stack trace is a powerful tool you can use to tell what a thread is doing at a moment in time.

Functions

Functions are essential to programmers, and one of the reasons is because they provide a way of re-using and structuring code. For example, I might write some mathematical code which uses the *sin* function. In some other code I might use *sin* again. But I only have to write the *sin* calculation code once.

Look at the following stack trace:



Each line has three parts:

module!function+displacement (e.g. ntdll.dll!NtWaitForMultipleObjects+0xc)

The stack trace works bottom-up; so BaseThreadInitThunk decided to call mainCRTStartup, which called another function which eventually called the TestHang function. Note that the displacement values (0xe, 0xf, 0x117, etc.) indicate the **return address**, not the point at which the function was called. so TestHang will jump back to main plus 44 bytes when it finishes running. That's also why you see KiFastSystemCallRet at the top of the user-mode stack trace.

Note that you'll often see these other lines:

module+offset (e.g. somedll.dll+0x1234) **address** (e.g. 0x12345678)

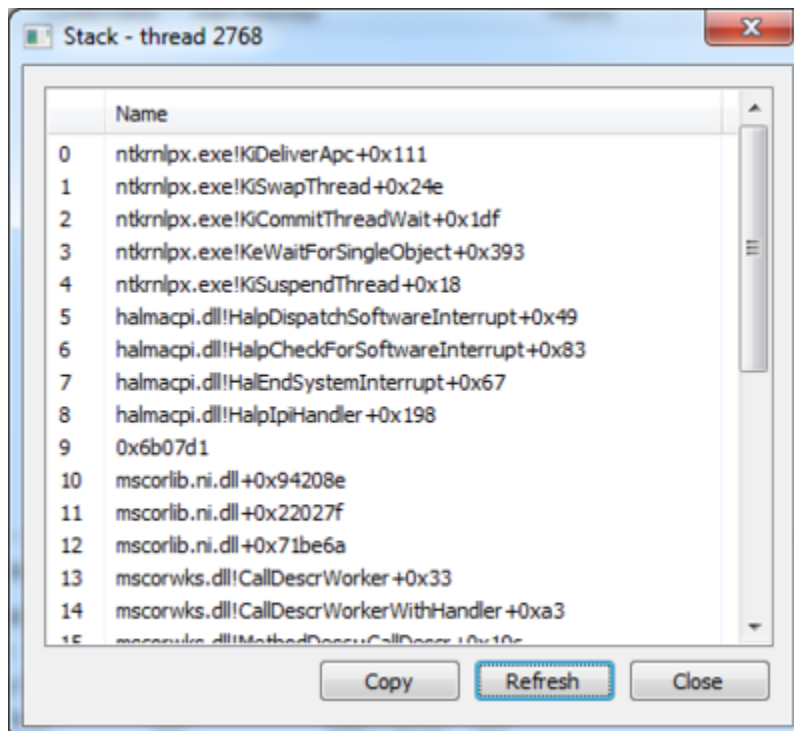
This usually indicates symbols are missing or something unusual (like .NET's JIT).

Kernel-mode vs. user-mode

In most versions of Windows Process Hacker will show you the kernel-mode stack trace as well. This is not usually very important, but can be useful sometimes. The kernel-mode part of the stack trace usually begins with the module being the kernel; in this case it's ntkrnlpx.exe. On most machines look for ntkrnlpa.exe and ntoskrnl.exe.

Most of the time you will see names such as KeWaitForSingleObject or KeWaitForMultipleObjects in the kernel-mode stack trace. This indicates that the thread is waiting for something to happen and is not running anything. The reason it is common to see this is simply because most threads running in most processes simply don't do very much; they wait for some external event, process it, and get back to waiting. It's most likely you'll catch the thread while it's sleeping.

Here's what it looks like when a thread is doing some heavy processing in user-mode:



Compare this with the first stack trace. You'll see that there's no ntdll.dll stack frame because the thread isn't making a system call (or calling any of ntdll.dll's functions). The first kernel-mode frame has HalpIpiHandler. The reason this

appears is due to the way in which Process Hacker gets kernel-mode stack traces: it queues a special kernel-mode APC to the thread which performs a stack trace inside the thread itself. So in this case you can see Process Hacker had to interrupt this particular program (which the kernel did with an **interrupt**) to get its kernel-mode stack trace. This type of stack trace indicates that the program (which is eating away at your available CPU time) is some kind of infinite loop or is simply taking a very long time to do whatever it's doing (e.g. audio/video encoding).

This **plugin adds some extra Network tab** features including:

- * Local connection service names (e.g. http for connections on port 80).
- * Remote connection service names (e.g. https for connections on port 443).
- * GeoIP lookup for network connections (e.g. Australia for remote connection addresses).

For local and remote service names;

#1: Right-click the column headers on the Network tab and select Customize.

#2: Add the Local Service and Remote Service columns.

For GeoIP lookup of remote network connections:

#1: Right-click the column headers on the Network tab and select Customize.

#2: Add the Country column.

Name	Country
 chrome.exe (2396)	 United States
 chrome.exe (2396)	 United States
 chrome.exe (2396)	 United States
 chrome.exe (2396)	 United States
 chrome.exe (2396)	 United States
 Microsoft.VsHub.Server.HttpHost.exe (6324)	 United States
 chrome.exe (2396)	 United States
 chrome.exe (2396)	 United States
 chrome.exe (2396)	 United States
 VsHub.exe (4248)	 Hong Kong
 svchost.exe (1056)	 Hong Kong
 explorer.exe (3488)	 Hong Kong
 chrome.exe (2396)	 Australia
 Waiting connections	 Australia
 chrome.exe (2396)	 Australia
 Waiting connections	

Process Hacker Command Line Reference

Process Hacker supports a limited number of command line options, listed below.

`-settings filename`

This option allows you to specify the location of Process Hacker's settings file. *filename* can be a relative path, in which case the current working directory at startup is used as the base.

Examples:

ProcessHacker.exe -settings settings.xml

-nosettings

Disables settings. Settings are set to their defaults at startup, and no settings are saved.

-noplugins

Disables plugins, even if the "Enable plugins" option is set.

-newinstance

Starts a new instance of Process Hacker, even if the "Allow only one instance" option is set.

-v

Forces Process Hacker's main window to be displayed at startup, even if the "Start hidden" option is enabled.

-hide

Hides Process Hacker's main window at startup, even if the "Start hidden" option is disabled.

-elevate

Prompts for elevation if Process Hacker is not started with elevated privileges.

-c -ctype *objecttype* -cobject *object* -caction *action* -cvalue *value*

Enables command mode. The status of the operation is returned in the exit status of the process.

Possible values of *objecttype*:

- "process". *object* is the process ID, and *action* can be "terminate", "suspend", "resume", "priority", "iopriority" or "pagepriority".
- "service". *object* is the service name, and *action* can be "start", "continue", "pause", "stop" or "delete".
- "thread". *object* is the thread ID, and *action* can be "terminate", "suspend" or "resume".

Examples:

ProcessHacker.exe -c -ctype process -cobject 1424 -caction terminate

ProcessHacker.exe -c -ctype process -cobject 5896 -caction priority -cvalue high

ProcessHacker.exe -c -ctype service -cobject Winmgmt -caction pause

-s

Enables silent mode. No error messages are displayed for command mode, -installkph and -uninstallkph.

-ras

Enters run-as-service mode. This is used internally by the Run As command.

-nokph

Disables KProcessHacker. Process Hacker will not attempt to load the driver or connect to it.

-installkph

Installs KProcessHacker as a System Start service.

-uninstallkph

Deletes the KProcessHacker service.

-debug

Shows the debug console early in the startup process.

-showoptions -hwnd *parentwindow* -point *x,y*

Displays the Advanced tab of the options window only. *parentwindow* specifies the parent window handle in hexadecimal and *x,y* specifies the location of the options window.

-phsvc

Enters phsvc mode. This exposes a LPC-based API currently used by Process Hacker for tasks that require elevation.

-priority *r|h|n|l*

Sets the priority of Process Hacker to realtime (r), high (h), normal (n) or idle (l).

-selectpid *pid*

Selects *pid* in a new or existing instance of Process Hacker.

-sysinfo *section*

Opens the System Information window at startup, and optionally navigates to the specified section.

