7. Gestión de procesos en PowerShell

- Procesos
  - Información sobre procesos
  - Tabla de procesos
  - Estados de un proceso
  - Crear procesos
  - Parar procesos
  - El procesador
  - Hilos
  - Comunicación entre procesos
  - Segundo plano
  - Servicios

Uno de los conceptos más importantes en los sistemas operativos es el proceso que se define como un programa en ejecución. Los programas son un conjunto de archivos que están almacenados en algún dispositivo de almacenamiento (disco duro, USB, etc.) y que por sí solos no tienen ningún funcionamiento, pero cuando ese conjunto de archivos se ejecutan entonces pasan a ser un proceso.

Procesos

El sistema operativo se encarga de: crear y destruir procesos, suspender y reanudar procesos, y sincronizar y comunicar procesos.

Información sobre procesos

Cmdlet

```
Get-Process
```

Alias

```
ps
gps
```

Parámetros y alias de los parámetros

```
Name                {ProcessName}
Id                  {PID}
InputObject         {}
IncludeUserName     {}
ComputerName        {Cn}
Module              {}
FileVersionInfo     {FV, FVI}
Verbose             {vb}
Debug               {db}
ErrorAction         {ea}
WarningAction       {wa}
InformationAction   {infa}
ErrorVariable       {ev}
WarningVariable     {wv}
InformationVariable {iv}
OutVariable         {ov}
OutBuffer           {ob}
PipelineVariable    {pv}
```

WMI

```
Get-WmiObject -Class win32_process
```

Ejemplos

Obtener información básica sobre procesos

```
Get-Process
```

Obtener información ampliada sobre procesos

```
Get-Process | Select-Object *
gps | Select-Object *
ps | Select-Object *
```

Obtener información sobre las propiedades de los procesos

```
Get-Process | Select-Object Name, Id, Company
gps | Select-Object Name, Id, Company
ps | Select-Object Name, Id, Company
```

Obtener información ordenada sobre las propiedades de los procesos

```
Get-Process | Select-Object Name, Id, Company | Sort-Object Name
gps | Select-Object Name, Id, Company | Sort-Object Name
ps | Select-Object Name, Id, Company | Sort-Object Name
```

Obtener los 5 primeros procesos ordenados

```
Get-Process | Select-Object Name, Id, Company | Sort-Object Name | Select-Object -First 5
gps | Select-Object Name, Id, Company | Sort-Object Name | Select-Object -First 5
ps | Select-Object Name, Id, Company | Sort-Object Name | Select-Object -First 5
```

Listar los procesos que tengan consumo alto de tiempo de CPU

```
Get-Process | select cpu,id,name | sort cpu -Descending
```

Listar los nombres de procesos que se están ejecutando

```
ps | select Name
```

Listar los procesos junto el fabricante

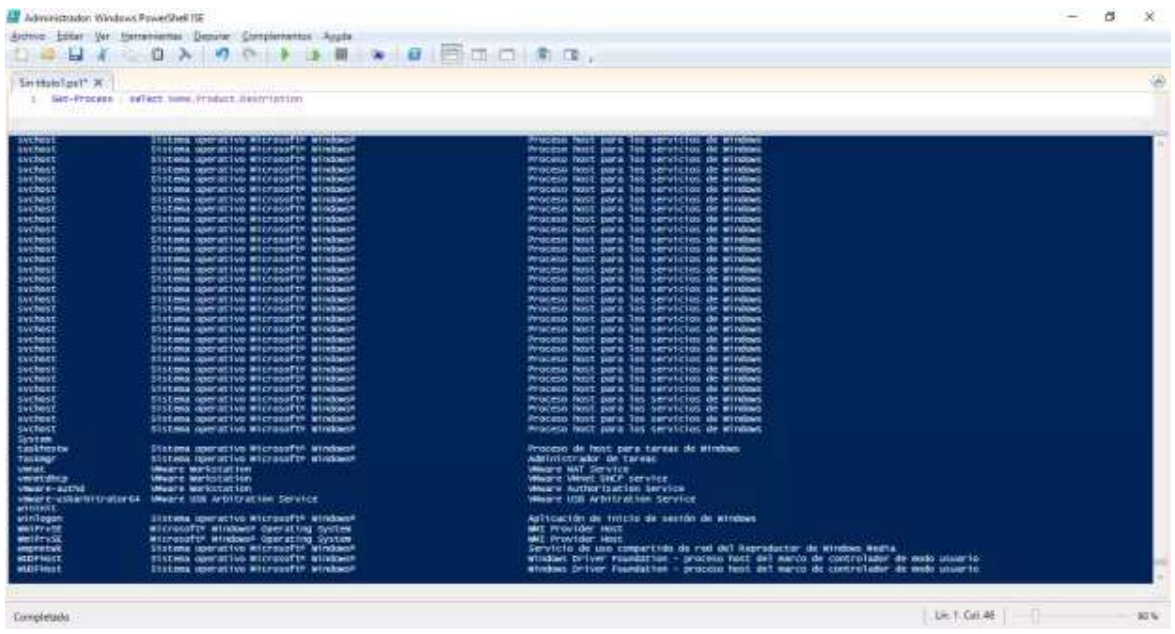
```
ps | select Name,Company
```

Información sobre la ruta de los procesos

```
(ps) | %{if($_.path){(Get-ChildItem $_.path | select VersionInfo)}}
```

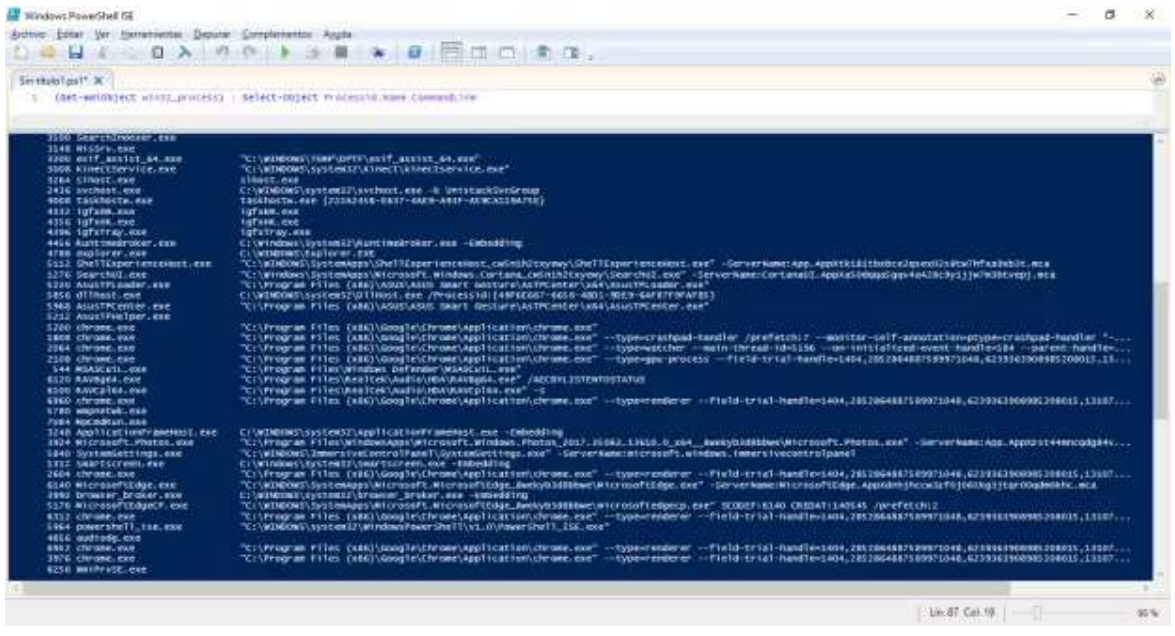
Mostrar la descripción de los procesos que se están ejecutando

```
Get-Process | select Name,Product,Description
```



Obtener más información sobre los procesos (línea de comandos)

```
(Get-WmiObject win32_process) | Select-Object Processid,Name,CommandLine
```

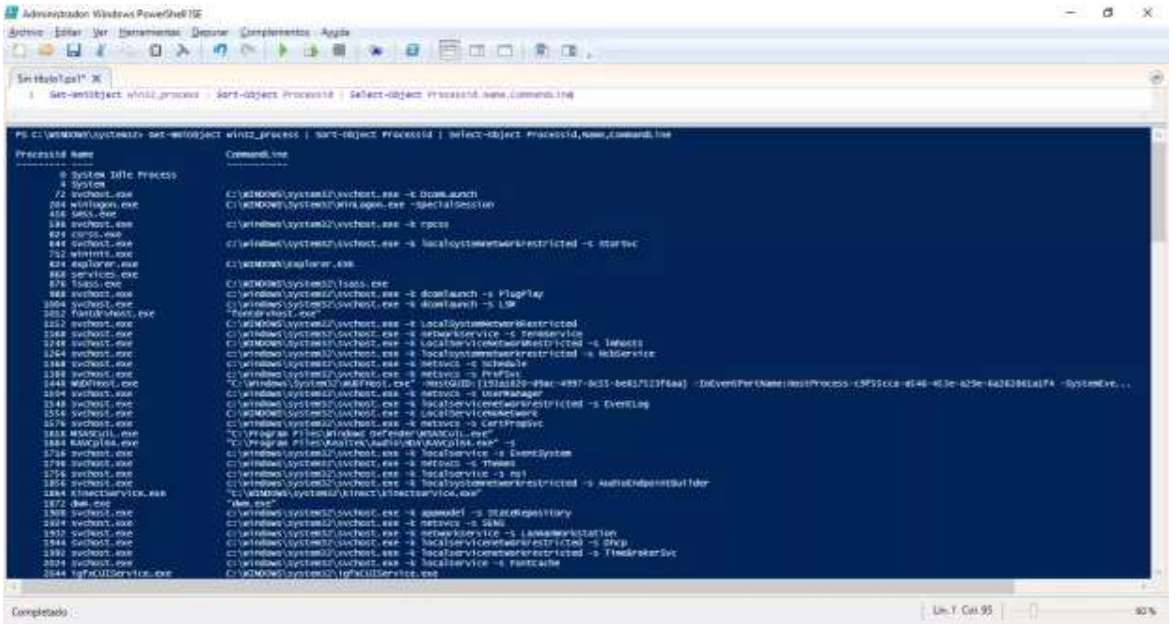


Obtener información sobre el comando ejecutado analizando procesos con una llamada WMI

```
Get-WmiObject -Class win32_process | select name, Path, ExecutablePath, CommandLine | Format-Custom
```

Ver los programas que se están ejecutando desde que se arranca el sistema operativo

```
Get-WmiObject win32_process | Sort-Object Processid | Select-Object Processid,Name,CommandLine
```



Mostrar información sobre un proceso indicando el nombre del proceso

```
Get-Process -Name notepad
```

Buscar procesos que no responden

```
(Get-Process).Where{-not $_.Responding}
```

Buscar un proceso que se está ejecutando

```
(Get-Process).Where{$_.Name -like "chrome"}
```

Módulos que se ejecutan en los procesos

```
Get-Process -Module
ps -Module
```

Abrir Notepad y listar las dll que están cargadas cuando se ejecuta, ver la ruta dónde se ejecutan

```
Get-Process -Name notepad -Module
```

Información sobre la ruta de los módulos de los procesos

```
ls (ps -Module | Select-Object Filename).Filename | Select VersionInfo | Format-Custom
```

Listar todas las propiedades de un proceso en concreto proceso utilizando WMI

```
(Get-WmiObject win32_process) | Where-Object processName -eq notepad.exe
```

Mostrar los procesos que ejecuta un usuario

```
(Get-Process).Threads | Select-Object Id,ThreadState | Sort-Object Id
```

Ejercicios

Contar el número de procesos que se están ejecutando en el sistema

```
(Get-Process).count
(Get-WmiObject Win32_OperatingSystem).NumberOfProcesses
```

Sacar las dll y realizar función hash

```
Get-Process -Module | Select-Object FileName | %{Write-Host $_.FileName, (Get-FileHash $_.FileName)}
```

Sacar las dll y realizar función hash para el proceso Notepad

```
Get-Process -Name notepad -Module | Select-Object FileName | %{Write-Host $_.FileName, (Get-FileHash $_.FileName)}
```

Ver los programas que se arrancan cuando un usuario inicia sesión.

```
Get-WmiObject -Class win32_startupCommand
```

Analizar procesos y guardar cada uno de los procesos en un fichero txt

```
New-Item -ItemType Directory -Path d:\power\ -Name procesos
foreach ($proceso in (Get-Process | Select-Object Name).name){
New-Item -Path D:\power\procesos -Name $proceso -Value $proceso -ItemType file
}
```

- # Tabla de procesos

Cuando un proceso se ejecuta, el sistema operativo le asigna un espacio de direcciones y lo añade a una tabla de procesos. El espacio de direcciones de cada proceso contiene las instrucciones y datos del programa, además tiene una pila que lleva la cuenta de las llamadas a procedimientos.
Los procesos se dividen en partes. Estas partes son las que se llevan a la memoria dependiendo de la técnica de gestión que se utilice: paginación, segmentación, intercambio, memoria virtual, etc.

La tabla de procesos contiene información de cada uno de los procesos que se están ejecutando. El contenido de una entrada (hay una entrada por cada proceso) en la tabla de procesos puede ser el siguiente:

- Identificador del proceso. Es un número que utiliza el sistema operativo para identificar los procesos de forma única, se conoce por la abreviatura PID (Process ID).
- Información del estado del proceso. Preparado, en ejecución o bloqueado.
- Prioridad. El sistema operativo puede asignar distinta importancia a cada proceso.
- Dirección de la memoria. Zona de la memoria donde se ha cargado el proceso.
- Directorio de trabajo. Directorio del usuario que ejecuta el programa.
- Tiempo utilizado por el procesador. Tiempo que el proceso se ha estado ejecutando en el procesador.

Ejemplos

Listar la siguiente información sobre los procesos:

- La cantidad de memoria no paginada que el proceso está utilizando, en kilobytes (NPM).
- La cantidad de memoria paginable que el proceso está utilizando, en kilobytes (PM).
- El tamaño del conjunto de trabajo del proceso, en kilobytes. El conjunto de trabajo se compone de las páginas de memoria que hace poco se hace referencia mediante el proceso (WS).
- La cantidad de memoria virtual que el proceso está utilizando, en megabytes. La memoria virtual incluye el almacenamiento de los archivos de paginación en el disco (VM).
- La cantidad de tiempo de procesador que el proceso se ha utilizado en todos los procesadores, en cuestión de segundos (CPU).

```
Get-Process | Select-Object NPM,PM,WS,VM,CPU
```

Listar los procesos que tengan consumo alto de tiempo de CPU

```
Get-Process | select cpu,id,name | sort cpu -Descending
```

Mostrar el tiempo transcurrido en la ejecución de un proceso

```
Get-Process | Select-Object Id,TotalProcessorTime
```

Mostrar información sobre un proceso indicando el ID del proceso

```
Get-Process -Id 2732
```

Mostrar los identificadores de los procesos junto con sus identificadores padres

```
Get-WmiObject win32_process | Select-Object ParentProcessId,ProcessId
```

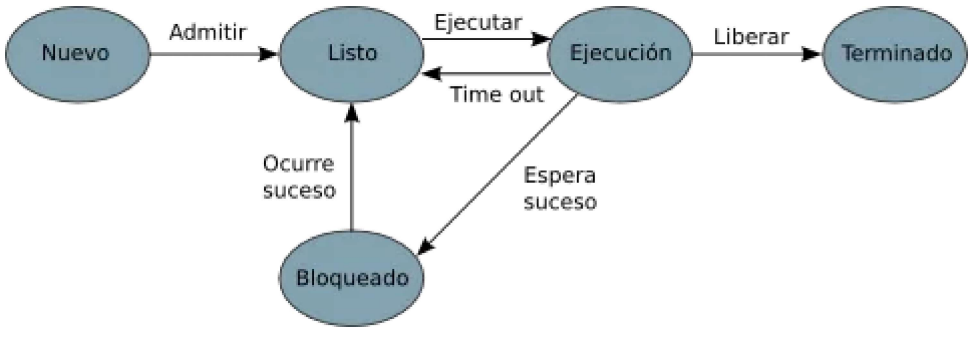
Ejercicio

Mostrar los procesos cuya zona de memoria para trabajar es mayor que 100 MB

```
#WS(K): The size of the working set of the process, in kilobytes. The working set consists of the pages of memory that were
@(Get-Process).where{$_.WorkingSet -gt 100MB}
@(Get-Process).where{$_.WS -gt 100MB}
```

Estados de un proceso

Los procesos cuando se ejecutan pueden atravesar distintos estados desde que se inician hasta que finalizan. Cuando un proceso se inicia, primero se mete en una cola de trabajos; cuando es admitido por el sistema, pasa a una cola de procesos que están preparados y esperando para ejecutarse; cuando el procesador asigna tiempo de ejecución, el proceso pasa de estar preparado a ejecución, y cuando el proceso necesita alguna señal o dato, pasa al estado bloqueado (se lo introduce en la cola de bloqueados).



Crear procesos

Los procesos se pueden crear de varias formas:

- Cuando se arranca el sistema operativo, éste ejecuta procesos necesarios para llevar a cabo distintas funciones.
- Por petición de un usuario del sistema, escribiendo un comando en una línea de comandos o pulsando encima de algún icono.
- Cuando un proceso hace una llamada al sistema para crear un nuevo proceso (para crear un proceso en Windows se utiliza función «CreateProcess()», en Linux se utiliza la llamada «fork()»).

En PowerShell se pueden arrancar procesos de las siguientes formas:

- Iniciar un proceso con un cmdlet

Cmdlet

```
Start-Process
```

Alias

```
saps
start
```

Parámetros y alias de los parámetros

```
FilePath              {PSPath}
ArgumentList          {Args}
Credential            {RunAs}
WorkingDirectory      {}
LoadUserProfile       {Lup}
NoNewWindow           {nnw}
PassThru              {}
RedirectStandardError {RSE}
RedirectStandardInput {RSI}
RedirectStandardOutput {RSO}
Verb                  {}
WindowStyle           {}
Wait                  {}
UseNewEnvironment     {}
Verbose               {vb}
Debug                 {db}
ErrorAction           {ea}
WarningAction         {wa}
InformationAction     {infa}
ErrorVariable         {ev}
WarningVariable       {wv}
InformationVariable   {iv}
OutVariable           {ov}
OutBuffer             {ob}
PipelineVariable      {pv}
```

Ejemplo

Iniciar el programa Notepad

```
Start-Process notepad
```

- Invocar un cmdlet

Cmdlet

```
Invoke-Expression
```

Ejemplo

Invocar el cmdlet Get-Process

```
$str = "Get-Process"
Invoke-Expression $str
```

- Crear un bloque e invocarlo

Cmdlet

```
Invoke-Command
```

Ejemplo

Invocar la ejecución del programa Ping

```
$scriptblock = {ping host3}
Invoke-Command -scriptblock $scriptblock
```

- Iniciar una instancia de una aplicación mediante WMI

WMI

```
([Wmiclass]'Win32_Process').GetMethodParameters('Create')
Invoke-WmiMethod -Path win32_process -Name create -ArgumentList notepad.exe
```

Ejemplo

Arrancar notepad iniciando una instancia mediante WMI

```
([Wmiclass]'Win32_Process').GetMethodParameters('Create')
Invoke-WmiMethod -Path win32_process -Name create -ArgumentList notepad.exe
```

- # Parar procesos

Igual que existen formas de crear procesos, también existen varias formas de acabar con los procesos:

- El proceso acaba de ejecutarse de forma normal.
- El proceso ha sufrido un error.
- Por petición de un usuario del sistema, escribiendo un comando en una línea de comandos o pulsando encima de algún icono.
- Cuando un proceso hace una llamada al sistema para acabar con un proceso (para terminar con un proceso en Windows se utiliza la función «ExitProcess()» y en Linux la llamada «kill()»).

Hay varias formas de parar un proceso en PowerShell:

- Parar un proceso con un cmdlet

Cmdlet

```
Stop-Process
```

Alias

```
spps
kill
```

Parámetros y alias de los parámetros

```
Name                {ProcessName}
Id                  {}
InputObject         {}
PassThru            {}
Force               {}
Verbose             {vb}
Debug               {db}
ErrorAction         {ea}
WarningAction       {wa}
InformationAction   {infa}
ErrorVariable       {ev}
WarningVariable     {wv}
InformationVariable {iv}
OutVariable         {ov}
OutBuffer           {ob}
PipelineVariable    {pv}
WhatIf              {wi}
Confirm             {cf}
```

Ejemplo

Parar un proceso

```
Get-Process -Name notepad | Stop-Process
```

- Parar un proceso mediante una invocación a un método WMI

Cmdlet

```
Invoke-WmiMethod
```

Ejemplo

Parar el proceso Notepad mediante una invocación a un método WMI

```
Get-WmiObject -Class Win32_Process -Filter “name='notepad.exe'” | Invoke-WmiMethod -Name Terminate
```

- Parar un proceso mediante el método Kill

Método para parar procesos

```
.Kill()
```

Ejemplo

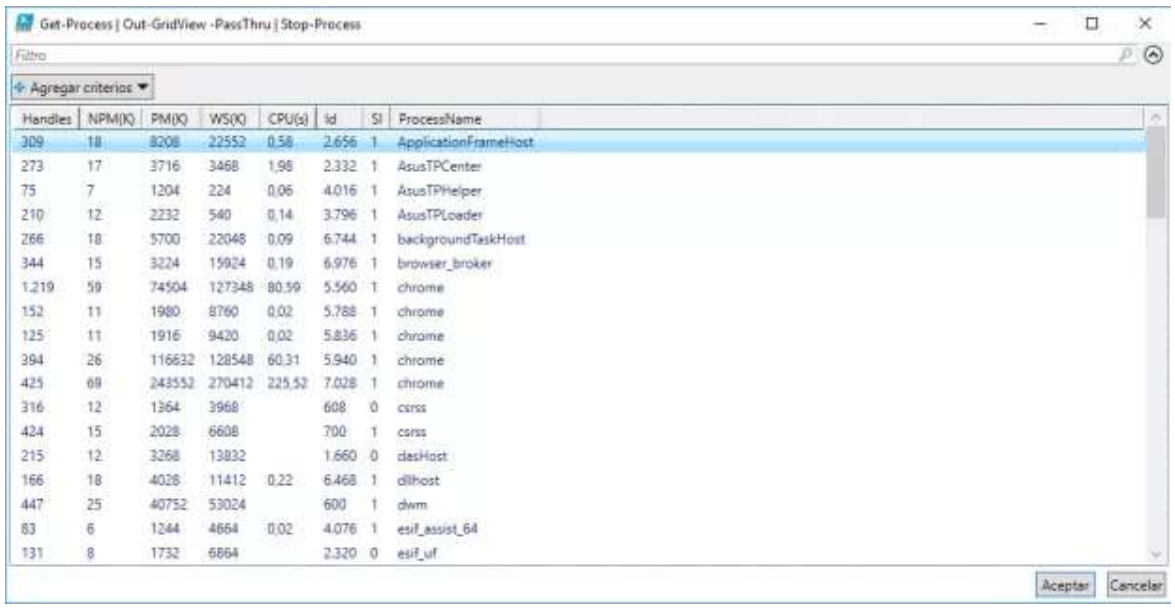
Parar el proceso Notepad mediante el método Kill

```
Get-Process -Name notepad | ForEach-Object -Process {$_.Kill()}
```

Ejercicio

Parar procesos de forma gráfica

```
Get-Process | Out-GridView -PassThru | Stop-Process
```



- El procesador

Los procesos se pueden ejecutar casi concurrentemente (ejecución simultánea de varios procesos) por el procesador, incluso cuando sólo se dispone de un procesador, dando la sensación de casi paralelismo, aunque el verdadero paralelismo sólo se da con múltiples procesadores o con procesadores con varios núcleos (cada núcleo puede ejecutar un proceso), con un solo procesador se cambia tan rápido de un proceso a otro que nos da esa sensación de paralelismo, esto se denomina multiprogramación.
Cualquier proceso que se está ejecutando puede suspender por la ocurrencia de un evento determinado, por ejemplo, que otro proceso se tenga que ejecutar, una interrupción, etc.
Cuando un proceso tiene que parar y dejar pasar a otro se produce un cambio de contexto, se guarda el contexto del proceso que se estaba ejecutando y se ejecuta el contexto del nuevo proceso. El contexto de un proceso contiene información para que el sistema operativo pueda gestionar el proceso.
Cmdlets con llamadas WMI
Utilización de CPU

```
Get-WmiObject Win32_PerfFormattedData_PerfOS_Processor
```

Información detallada de procesos en el procesador

```
Get-WmiObject Win32_PerfRawData_PerfProc_Process
```

Carga del procesador

```
Get-WmiObject Win32_Processor
```

Ejemplo

Ver la carga del procesador

```
Get-WmiObject Win32_Processor | Select-Object LoadPercentage
```

- Hilos

Un proceso tiene miniprocesos que se llaman hilos, la razón de dividir un proceso en unidades más pequeñas es que la mayoría de los procesos están realizando varias acciones a la vez (leer del disco, esperar un clic del ratón, etc.) y puede ser que alguna de esas acciones se bloquee y deje bloqueado al proceso completo.
Utilizando los hilos, el bloqueo sólo se produce en una parte y no todo el proceso, esto quiere decir que otros hilos se pueden seguir ejecutando sin problema.
Algunas características de los hilos:

- Los procesos crean hilos.
- Los hilos tienen prioridad.
- Se pueden ver los hilos que se ejecutan.
- Se pueden ver las prioridades de los hilos.
- Se pueden relacionar hilos con procesos y servicios.

Listado de hilos que se están ejecutando
WMI

```
Get-WmiObject -Class Win32_Thread
```

Ejemplos

Proceso que crea un hilo

```
(Get-WmiObject -Class Win32_Thread).ProcessHandle | %{
Get-Process -Id $_
}
```

Mostrar la prioridad de los hilos de los procesos

```
(Get-WmiObject -Class Win32_Thread) | Select-Object ProcessHandle,Priority,PriorityBase | Sort-Object ProcessHandle
(Get-Process).Threads | Select-Object Id,CurrentPriority,BasePriority | Sort-Object Id
```

Conocer el estado de los hilos de los procesos

```
(Get-WmiObject -Class Win32_Thread) | Select-Object ProcessHandle,ThreadState | Sort-Object ProcessHandle
(Get-Process).Threads | Select-Object Id,ThreadState | Sort-Object Id
```

Ejercicios

Mostrar los hilos que se están ejecutando en relación con los servicios y los procesos

```
$i=0
(Get-WmiObject -Class Win32_Thread) | %{
$i++
Write-Host $i,$_.Handle,$_.ProcessHandle,(Get-WmiObject -Class Win32_Service | Where-Object State -EQ 'Running' | Where-Obje
}
```

Comunicación entre procesos

En ocasiones los procesos tienen que comunicarse unos con otros pasándose información, la forma de comunicarse entre procesos se puede realizar mediante, por ejemplo, zonas de memoria compartida.
Existen algunas situaciones que el sistema operativo debe controlar para que la comunicación entre procesos sea correcta. Una de ellas es asegurar que si un proceso utiliza una zona o archivo compartido, no deje a otros procesos utilizar esos recursos a la vez, es decir, excluir del acceso a otros, esto se conoce como exclusión mutua. La solución es controlar el acceso a la región crítica, que se puede definir como la parte de un programa en la cual se accede a un recurso compartido y donde no debe haber más de un proceso accediendo simultáneamente. Habrá, por tanto, que ordenar y controlar el acceso que hacen los programas a esa región critica, no puede haber dos procesos simultáneamente en sus regiones críticas», entonces si un programa entra en la sección crítica, no permitirá el acceso a otro programa.

Segundo plano

Los procesos que se ejecutan pueden estar en primer plano o segundo plano: los de primer plano interactúan con los usuarios, y los de segundo plano o demonios realizan una función específica sin tener que actuar con el usuario (aunque podrían hacerlo).
En PowerShell se pueden crear trabajos y actúan en segundo plano.
Cmdlet

```
Start-Job
```

Alias

```
sajb
```

Parámetros y alias de los parámetros

```
DefinitionName       {}
DefinitionPath       {}
Type                 {}
Name                 {}
ScriptBlock          {Command}
Credential           {}
FilePath             {}
LiteralPath          {PSPath}
Authentication       {}
InitializationScript {}
RunAs32              {}
PSVersion            {}
InputObject          {}
ArgumentList         {Args}
Verbose              {vb}
Debug                {db}
ErrorAction          {ea}
WarningAction        {wa}
InformationAction    {infa}
ErrorVariable        {ev}
WarningVariable      {wv}
InformationVariable  {iv}
OutVariable          {ov}
OutBuffer            {ob}
PipelineVariable     {pv}
```

Ejemplos

Iniciar un trabajo en segundo plano con PowerShell

```
Start-Job -ScriptBlock {Get-Process}
```

Iniciar un script en segundo plano con PowerShell

```
Start-Job -FilePath "c:\scripts\script.ps1"
```

Iniciar cinco trabajos en segundo plano con PowerShell

```
1..5 | %{Start-Job -ScriptBlock {Get-Process}}
```

- # Servicios

Los procesos en segundo plano que realizan distintas funciones, algunas relacionadas con el sistema operativo y otras no, se denominan servicios, y se están ejecutando permanentemente en el sistema.
Los servicios se pueden iniciar, detener, pausar, reanudar, etc. Estas acciones, normalmente, sólo las puede realizar el administrador de forma local o remota. Un ejemplo de servicio de sistema es el servicio de escritorio remoto, que permite conectarse remotamente al equipo.
Los servicios ejecutan procesos y los procesos tienen hilos, hay relación entre procesos e hilos y entre servicios e hilos.
Listar todos los servicios
Cmdlet

```
Get-Service
```

Alias

```
gsv
```

Parámetros y alias de los parámetros

```
Name                {ServiceName}
ComputerName        {Cn}
DependentServices   {DS}
RequiredServices    {SDO, ServicesDependedOn}
DisplayName         {}
Include             {}
Exclude             {}
InputObject         {}
Verbose             {vb}
Debug               {db}
ErrorAction         {ea}
WarningAction       {wa}
InformationAction   {infa}
ErrorVariable       {ev}
WarningVariable     {wv}
InformationVariable {iv}
OutVariable         {ov}
OutBuffer           {ob}
PipelineVariable    {pv}
```

WMI

```
Get-WmiObject -Class Win32_Service
Get-WmiObject -query "select * from win32_service"
```

Ejemplos

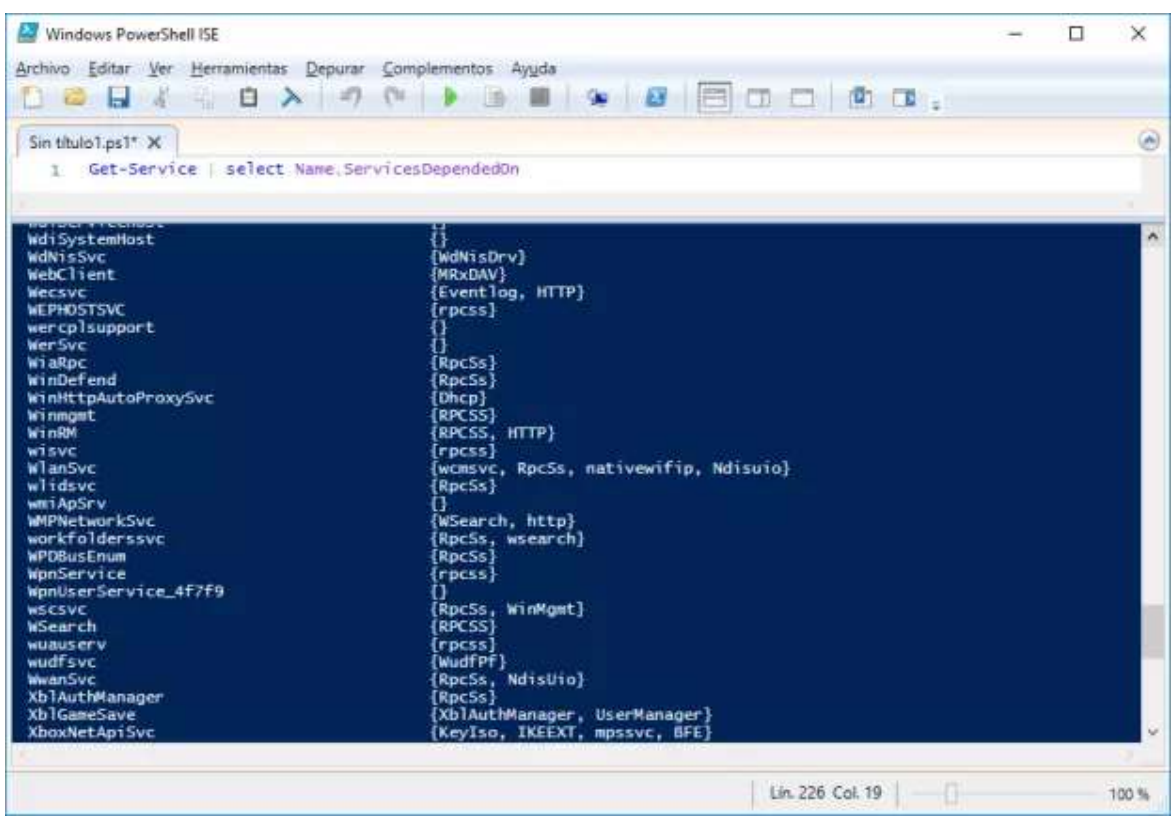
Ver servicios que están “Running”

```
Get-WmiObject -Class Win32_Service | Where-Object State -EQ 'Running'
```

Ejercicios

Ver los servicios que dependen de otros servicios

```
Get-Service | select Name,ServicesDependedOn
```



Mostrar los procesos que se están ejecutando en relación con los servicios

```
(Get-WmiObject -Class Win32_Service | Where-Object State -EQ 'Running') | %{
Write-Host $_.Name,$_.ProcessId,$_.State,(Get-Process -Id $_.ProcessId).Name
```

```
}
```

Mostrar los procesos que se están ejecutando en relación con los servicios (solo llamadas WMI)

```
(Get-WmiObject -Class Win32_Service | Where-Object State -EQ 'Running') | %{
Write-Host $_.Name,$_.ProcessId,$_.State,(Get-WmiObject -Class win32_process | Where-Object ProcessId -EQ $_.ProcessId).Proc
}
```

Mostrar los procesos que se están ejecutando en relación con los servicios (solo llamadas WMI y más información)

```
(Get-WmiObject -Class Win32_Service | Where-Object State -EQ 'Running') | %{
Write-Host $_.Name,$_.ProcessId,$_.State,(Get-WmiObject -Class win32_process | Where-Object ProcessId -EQ $_.ProcessId | sel
}
```

Mostrar los servicios que se están ejecutando en relación con los procesos y los hilos

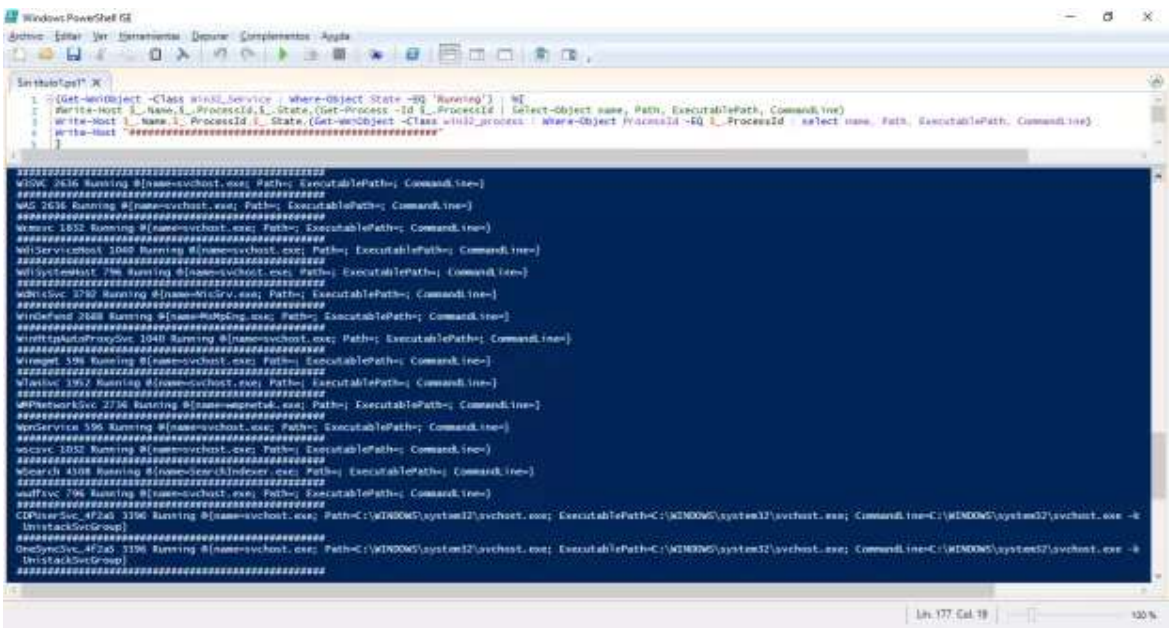
```
(Get-WmiObject -Class Win32_Service | Where-Object State -EQ 'Running') | %{
Write-Host $_.Name,$_.ProcessId,$_.State,(Get-Process -Id $_.ProcessId).Name,(Get-WmiObject -Class Win32_Thread | Where-Obje
}
```

Mostrar los hilos que se están ejecutando en relación con los servicios y los procesos

```
$i=0
(Get-WmiObject -Class Win32_Thread) | %{
$i++
Write-Host $i,$_.Handle,$_.ProcessHandle,(Get-WmiObject -Class Win32_Service | Where-Object State -EQ 'Running' | Where-Obje
}
```

Mostrar información avanzada de los procesos (Path, ExecutablePath, CommandLine) que se están ejecutando en relación con los servicios

```
(Get-WmiObject -Class Win32_Service | Where-Object State -EQ 'Running') | %{
#Write-Host $_.Name,$_.ProcessId,$_.State,(Get-Process -Id $_.ProcessId | Select-Object name, Path, ExecutablePath, CommandL
Write-Host $_.Name,$_.ProcessId,$_.State,(Get-WmiObject -Class win32_process | Where-Object ProcessId -EQ $_.ProcessId | sel
Write-Host "#####################################################"
}
```



Related posts

- Cmdlets relacionados con tareas básicas y de administración en el sistema operativo Windows
- Windows PowerShell aliases
- 4. Gestión del sistema de archivos en PowerShell
- Windows Post Exploitation Cmdlets Execution (PowerShell)
- Seguridad informática con PowerShell
- Sistemas operativos monopuesto
- 10. Gestión del rendimiento en PowerShell
- 1. Introducción a PowerShell
- 9. Gestión de la red en PowerShell
- 8. Gestión de usuarios en PowerShell

PowerShell

Chrome, Directory, DisplayName, FileVersionInfo, foreach, ForEach-Object, fork, Format-Custom, Get-ChildItem, Get-FileHash, Get-Process, Get-Service, Get-WmiObject, gsv, Invoke-Command, Invoke-Expression, Invoke-WMIMethod, itemType, Jesús Niño Camazón, kill, Linux, ls, Module, New-Item, Out-GridView, Path, ping, pm, ps, PSVersion, sajb, saps, Scripts, Select-Object, Sistema Operativo, size, Sort, Sort-Object, spps, Start-Job, Start-Process, Stop-Process, USB, value, verdadero, VersionInfo, Virtual, Where-Object, win32_operatingsystem, win32_process, win32_processor, win32_Service, win32_startupCommand, Windows, WMI, Write-Host

← Vaciar y restablecer el contenido de la caché de resolución del cliente DNS en Windows 10

Recopilación de Scripts día 08/07/2017 →

ᵕ