|  |
| --- |
|  |
| UD 2: Programación multiproceso (I) |
|  |
| Programación de servicios y procesos |



**sEIM**

4 de septiembre de 2012

Autor: Dani

UD 2: Programación multiproceso (I)

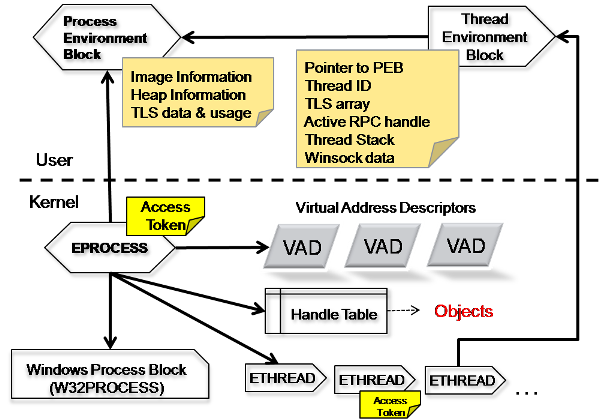
Programación de servicios y procesos

Estructuras de datos para gestionar procesos:

Cada proceso de Windows está representado con 2 componentes.

**EPROCESS** (Executive Process Block) reside en modo kernel y tiene un punter a PEB.

**PEB** (Process Environment Block) reside en modo usuario.

[](http://sites.google.com/site/sarvasite/windows-os-internals/processes-threads/process-internals/Process+threads.png)

**Figure: Estructuras de datos para un proceso ejecutándose en el Sistema.**

# Contenido de EPROCESS

|  |  |
| --- | --- |
| **Element** | **Purpose** |
| KPROCESS or PCB | Kernel process block (KPROCESS) or Process Control Block (PCB) contiene:   * Cabecera de dispatcher (para planificar hilos) * Tiempo de Kernel & user * Spinlock de proceso y afinidad de procesador. * Estado del proceso * Prioridad * Thread quantum (Cuota de tiempo permitida para un hilo) * Puntero a la estructura KTHREAD del proceso. |
| Identificación del proceso | Id de proceso, ID del proceso padre, nombre de la imagen. |
| Quota block | Límites de memoria paginada y no paginada, estadísticas de uso. |
| Virtual Address  Descriptors (VADs) | Lista enlazada de descriptors de memoria virtual. Se usan por el gestor de memoria para el seguimiento de las direcciones virtuales que está usando el proceso. |
| Puerto de excepción LCP | Usado por el gestor para comunicarse con el proceso cuando uno de los hilos causa excepción. |
| Puerto de depuración LCP | Usado por el gestor para comunicarse con el proceso, cuando uno de sus hilos causa un evento de depuración. |
| Token de acceso | Perfil de seguridad del proceso |
| Tabla de manejadores | Dirección a la table de manejadores por proceso. |
| Process Environment Block (PEB) | * Estructura residente en modo usuario, contiene: * Información de la imagen (dirección base, números de versión,lista de módulos) * Información de heap de proceso (#heaps, tamaño del heap, lista de heaps) * Thread Local Storage (TLS) datos y uso. |
| Windows Subsystem Process Block (W32PROCESS) | Estructura paralela mantenida por CSRSS.EXE para cada proceso Windows que usa funciones USER o GDI). Se crea la primera vez que se invoca a una función de dichas librerías. |

# Kernel Variables:

La lista de variables globales del Kernel. Se refieren a todos los procesos en ejecución dentro del sistema.

|  |  |  |
| --- | --- | --- |
| **Variable** | **Tipo** | **Descripcion** |
| *PsActiveProcessHead* | Queue Header | Cabeza de lista de los bloques de proceso |
| *PsIdleProcess* | EPROCESS | Bloque de procesos en 2º plano. |
| *PsInitialSystemProcess* | Pointer to EPROCESS | Puntero al bloque de proceso de proceso del sistema inicial que contiene los hilos del sistema |
| *PspCreateProcessNotifyRoutine* | Array of pointers | Array de punteros a rutinas que se llamarán en la creación y destrucción del proceso (max 8 rutinas) |
| *PspCreateProcessNotifyRoutineCount* | DWORD | Contador de rutinas de notificación del proceso |
| *PspLoadImageNotifyRoutine* | Array of pointers | Array de punteros a las rutinas que se invocarán en la carga de la imagen. |
| *PspLoadImageNotifyRoutineCount* | DWORD | Contador de rutinas de carga de proceso registradas. |
| *PspCidTable* | Pointer to HANDLE\_TABLE | Tabla de manejadores e IDs del hilo cliente. |

# Contadores de rendimiento:

Windows mantiene el ciertos de contadores para el seguimiento de los procesos del sistema.

**Privileged Time:** Seguimiento del tiempo que consumen los hilos en modo kernel, en un interval concreto.

**User Time**: Seguimiento del tiempo que consumen los hilos del procesos en modo usuario.

**Processor Time**: *suma de los anteriores.*

**Thread count:** nº de hilos actuales del proceso.

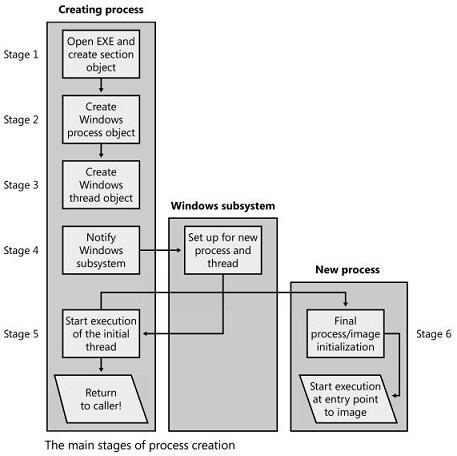
**Handle count:**número de manejadores abiertos por el proceso.

**ID process:** ID del proceso

**Elapsed time:** segundos transcurridos desde la creación del proceso.

# Main Stages of Process Creation:

Secuencia de pasos que desencadena la función CreateProcess():

[](https://sites.google.com/site/sarvasite/windows-os-internals/processes-threads/process-internals/ProcessCreationStages.JPG?attredirects=0)

## Paso 1: Abriendo el ejecutable que dará lugar al proceso.

Comprobar que existe la imagen especificada. Si no se especifica imagen, será la primera palabra de la línea de comandos.

Comprobar restricciones de uso y permisos.

Si es un ejecutable de Windows, se ejecuta directamente

Si no es un ejecutable de windows, abrirá otros procesos de apoyo y volverá al inicio paso 1.

Por ejemplo:

|  |  |
| --- | --- |
| **Tipo de imagen.** | **Proceso utilizado?** |
| POSIX | Run Posix.exe |
| MSDOS .exe, .com, or .pif | Run Ntvdm.exe |
| Win16 application | Run Ntvdm.exe |
| MSDOS .bat or .cmd | Run Cmd.exe |

Despues, si se encuentra un fichero de imagen Windows correcto, crea la sección de objetos. Todavía no se mapea en memoria, pero ya está abierta.

Si la imagen es un DLL, la función CreateProcess falla.

Comprueba la clave de registro "Image File Execution Options",para ver si tiene un debugger asociado.

## Paso 2: Creando el Windows Executive Process Object

Ya tenemos una imagen ejecutable de windwos válida abierta y una sección para maperlo en el espacio de direcciones del proceso. **(Step 1)**

Se crea una Windows Executive Process object para ejecutar la imagen llamando a la función interna: *NtCreateProcess.* Esto supone:

* Configurar EPROCESS Block
* Crear e inicializar el espacio de direcciones del proceso.
* Inicializar la lista de espacios de trabajo, VAD y mapear la imagen en el espacio de direcciones.
* Inicializar KPROCESS (KPROCESS)
* Configurar PEB
* Ntdll.dll es mapeado dentro del proceso.

**Stage 3: Creando el hilo inicial, su pila y contexto de ejecución.**

* El proceso no hace nada hasta que no se crea su primer hilo.
* Lo primero que se hace es inicializar la pila y el contexto de ejecución.
* Después se crea el hilo mediante la función *NtCreateThread.*
* *NtCreateThread* llama a *PspCreateThread*, y ejecuta los siguientes pasos:
  + *Se incrementa el contador de hilos del proceso.*
  + *Se crea* Executive Thread Block (ETHREAD)
  + *Se genera el* Thread Id.
  + Se configure TEB (Thread Environment Block) en el modo usuario.
  + *KeInitThread* configure el bloque KTHREAD.

El hilo ya está listo para su ejecución.

**Paso 4: Notificar al subsistema de Windows acerca del Nuevo proceso.**

* Kernel32.dll envía mensaje a Windows Subsystem para hacer que se prepare para los Nuevo proceso e hilo.
* El mensaje contiene lo siguiente:
  + Manejadores de hilo y proceso
  + Entradas en las flag de creación de proceso.
  + ID de proceso padre.
  + Flag especificando que es una aplicación de Windows para que CSRSS no muestre el cursor de inicio.
* CSRSS.exe Hace lo siguiente:
  + Establece la prioridad del proceso.
  + Aloja el process block de CSRSS
  + Configura Puerto de excepciones y de debug y actúa como “dispatcher” de mensajes para el proceso si algun evento de debug o excepción es lanzado en el sistema.
  + Aloja el Thread Block de CSRSS
  + Increments its count of processes
  + Crea e inicializa W32PROCESS: Estructura de datos para cada proceso, que el subsistema de Windows usa en modo Kernel.
  + Se muestra el cursor de inicio de la aplicación.

## Paso 5: Iniciar y ejecutar el Hilo inicial

Hasta ahora el hilo y el proceso han sido creados e inicializados. El subsistema de Windows sabe que hay un nuevo proceso. Salvo que se especifique la opción CREATE\_SUSPEND, el hilo arrancará.

## Paso 6: Ejecutando la inicialización del proceso en el contexto del nuevo proceso.

* KiThreadStartup: rutina de inicia en modo kernel para iniciar un Nuevo hilo.
  + *KiThreadStartup* llama a la rutina *PspUserThreadStartup*, pasando como parámetro la dirección de inicio del hilo en modo usuario.
  + *PspUserThreadStartup pone en la cola un APC en modo usuario para ejecutar la rutina de inicialización de la carga de la imagen* (*LdrInitializeThunk está en* NTDLL.DLL)
* *El hilo ejecuta el APC cuando vuelve a modo usuario, y* *LdrInitializeThunk* para inicializar:
* loader
* heap manager
* NLS tables
* Thread-local Storage (TLS) array
* Critical Section structures
* Carga las DLLs necesarias, y llama a las funciones de dicha DLL con DLL\_PROCESS\_ATTACH
* Tras esto, el hilo prosigue su ejecución.