Sistemas Operativos

Unidad 2: Procesos

"Si no puedes describir lo que está haciendo como un proceso, no sabes lo que está haciendo".

~ W. Edwards Deming

Procesos

Introducción

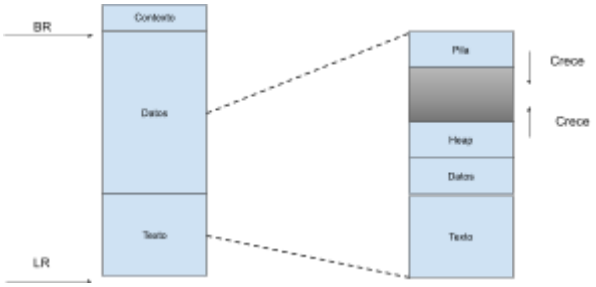
Ahora estamos a punto de embarcarnos en un estudio detallado de cómo se diseñan y construyen los sistemas operativos. El concepto más importante y central en cualquier sistema operativo es el de **proceso**: una abstracción de un programa en ejecución. Todo lo demás depende de este concepto.

En la actualidad, las computadoras pueden hacer varias cosas al mismo tiempo. Mientras se ejecuta un programa de usuario, una computadora también puede leer desde un disco y enviar texto a una pantalla o impresora. En un sistema multiprogramado, el CPU también cambia de un programa a otro, ejecutando cada uno durante decenas o cientos de milisegundos. Si bien, estrictamente hablando, en un instante de tiempo, el CPU está ejecutando sólo un programa, en el transcurso de 1 segundo, puede ejecutar varios programas, dando así a los usuarios la ilusión de paralelismo. A veces, se habla de pseudoparalelismo en este contexto, para contrastarlo con el verdadero paralelismo de hardware de los sistemas multiprocesador (que tienen dos o más CPU que comparten la misma memoria física).

Definición

De manera informal, podemos decir que un proceso es un programa en ejecución. Sin embargo, un proceso es mucho más que eso. Un proceso no sólo está compuesto por el código del programa sino que además posee datos globales y locales, información de contexto, áreas de memoria dinámica, en conclusión no es sólo código.

Una representación de la imagen de un proceso en memoria podría ser la siguiente:



La sección llamada “Contexto”, contiene todos los registros utilizados por el sistema operativo para poder gestionar el proceso, tales como el PC (program counter), IR (instruction record), etc.

La sección “Texto” contiene el código del programa, o mejor dicho las instrucciones que representan dicho programa.

La sección “Datos” es utilizada para almacenar datos y variables globales de proceso.

Por último tenemos las secciones “Pila” (stack) y “Heap” (o montículo, pero vamos a llamarlo heap) son utilizadas para almacenar datos temporales. La “Pila” es utilizada para almacenar variables globales, llamadas a funciones, parámetros, puntos de retorno, etc. y el “Heap” es utilizado para la asignación dinámica de memoria (estructuras dinámicas como pilas, colas y listas).

Corolario	**Programa vs Proceso** Un programa en sí mismo no es un proceso. Un programa es una entidad pasiva, un archivo que contiene instrucciones que se encuentran almacenadas en disco. Un proceso por su parte, es una entidad activa, que posee un contexto y recursos asociados.

Descripción y control de procesos

En la Unidad I vimos algunas definiciones para el concepto de proceso. También podríamos decir que un proceso es una entidad compuesta por un conjunto de elementos, principalmente el código del programa, sus datos y el ***bloque de control de proceso (BCP)***. Este último elemento es una estructura de datos que el sistema operativo crea y utiliza para poder gestionar el proceso. Los datos más importantes que incluye el BCP son:

- **Identificador (PID)**: identificador único del proceso.
- **Estado**: el estado en el que se encuentra el proceso
- **Prioridad**: nivel de prioridad relativo al resto de los procesos
- **Contador de programa (PC)**: dirección de memoria de la siguiente instrucción del programa a ejecutar
- **Datos de contexto**: datos que están presentes en los registros del procesador cuando el proceso está corriendo.

- **Información de estado de E/S**: incluye peticiones de E/S pendientes, archivos en uso, etc.

Otros datos que contiene:

- **Punteros a memoria**: incluye punteros al código del programa y datos, y bloques de memoria compartida.
- **Datos de auditoría**: tiempo de procesador, tiempo que está en ejecución, etc.

El punto más importante es que a través del **BCP** el sistema operativo puede gestionar cada uno de los procesos y proporcionar multiprogramación. Por ejemplo, cuando un proceso es suspendido para dar paso a otro proceso, los datos actuales de los registros del procesador se guardan en *los datos de contexto* del proceso que se va a suspender y del proceso elegido para ejecutar, se recuperan los datos del contexto que son cargados en los registros del procesador para continuar con su ejecución.

Estados de un proceso

Una de las responsabilidades del sistema operativo, si no es la más importante, es controlar la ejecución de los procesos y asignar recursos a los mismos. Para esto, el primer paso es definir qué comportamiento se desea que tengan.

Modelo de 2 estados

El modelo más simple que podemos construir es aquel en el que un proceso está siendo ejecutado por el procesador o no, es decir, en este modelo un proceso sólo puede estar en 2 estados *Ejecutando* o *No Ejecutando*.

Cuando el sistema operativo crea un proceso, crea su BCP y lo inserta al sistema en estado No Ejecutando. A partir de ese momento, el proceso existe y está esperando su oportunidad para ejecutar. En algún momento, el proceso que se está ejecutando será interrumpido y el *dispatcher* seleccionará otro proceso para ejecutar. El proceso interrumpido pasará a estado No Ejecutando.

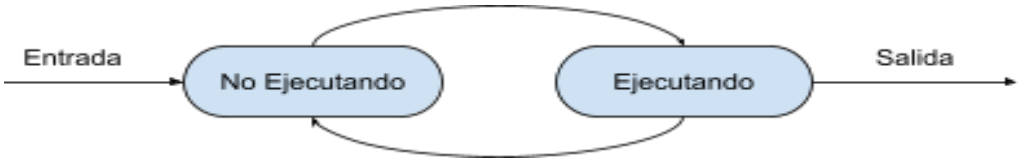
Definición	**¿Qué es el dispatcher?** Es un pequeño programa que se encarga de intercambiar los procesos en ejecución

A pesar de la simpleza de este modelo, vemos que para poder realizar el intercambio de procesos, el sistema operativo necesita tener información de los mismos que le permita

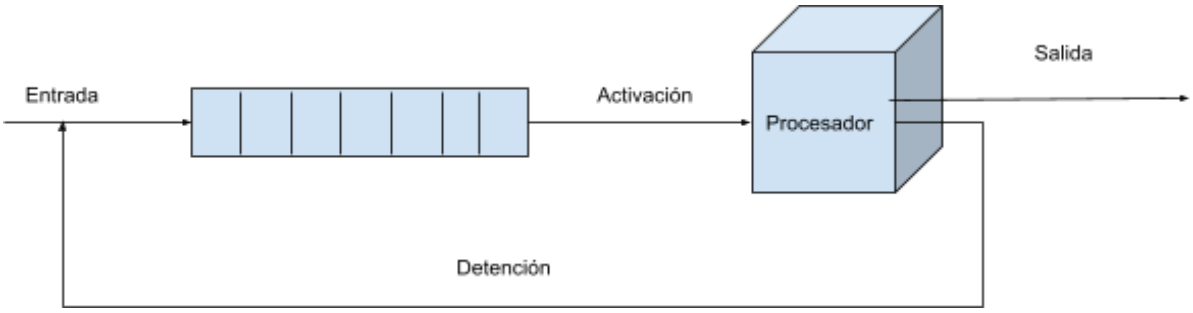
“seguirlos”, restaurarlos, etc. Es por esto, entre otras cosas, que surge el BCP que incluye el estado actual y su localización en memoria.

Los procesos que no se están ejecutando, esperan en una cola su turno para ejecutarse. Los elementos de estas colas, no son los procesos propiamente dicho sino que son punteros a los BCP de los procesos.

El siguiente gráfico es un esquema representativo del proceso descrito.



Una representación de la cola utilizada por el sistema operativo para los procesos detenidos:



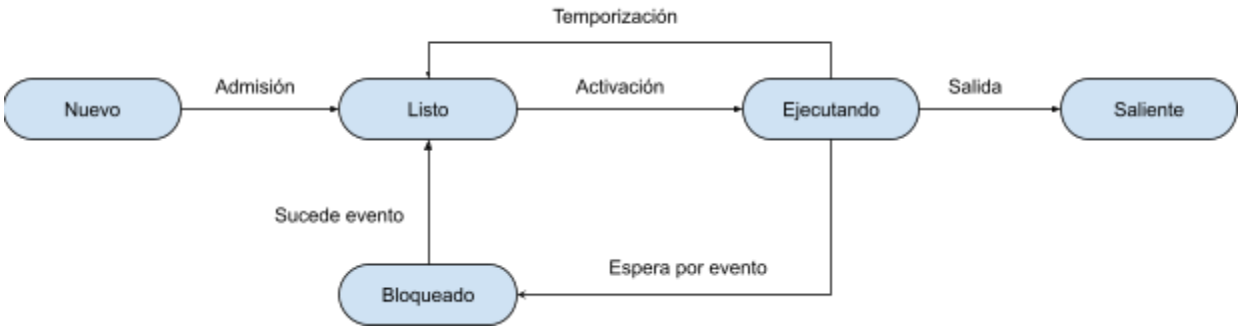
La cola en la que esperan los procesos, es una lista de tipo FIFO (primero que entra, primero que sale) y el procesador opera siguiendo una estrategia cíclica (round-robin) sobre todos los procesos.

Modelo de 5 estados

El modelo presentado anteriormente no resulta muy efectivo para gestionar los procesos ya que algunos de los procesos que se encuentran en la cola de espera, en estado *No Ejecutando*, podrían estar listos para ejecutar mientras que otros pueden estar en dicho estado debido a que se encuentran bloqueados esperando por una operación de E/S.

Para sortear este inconveniente, la solución es desdoblar el estado de *No Ejecutando* en 2 estados nuevos: *Listo* y *Bloqueado*. Para poder gestionar los procesos correctamente se agregan 2 estados más que son *Nuevo* y *Saliente*.

- **Nuevo**: un proceso que se acaba de crear y que aún no ha sido admitido en el grupo de procesos ejecutables por el sistema operativo. Se trata de un nuevo proceso que no ha sido cargado en memoria principal, aunque su BCP si ha sido cargado.
- **Listo**: un proceso que se encuentra listo para ejecutar.
- **Ejecutando**: un proceso que está actualmente en ejecución.
- **Bloqueado**: un proceso que no puede ejecutar hasta que se cumpla un evento determinado o se complete una operación de E/S.
- **Saliente**: un proceso que fue detenido o ha sido abortado.



Los estados *Nuevo* y *Saliente* son útiles para gestionar los procesos y poder administrar mejor los recursos del sistema.

El estado Nuevo

El estado Nuevo se corresponde con un proceso que acaba de ser definido, por ejemplo en un sistema de tiempo compartido, si un nuevo usuario intenta ingresar en el sistema, o en un sistema operativo por lotes si se intenta crear un nuevo trabajo.

¿Qué ocurre internamente?

Cuando el sistema operativo recibe la solicitud de un nuevo proceso, le asocia un nuevo identificador (PID) al proceso, se reservan y construyen todas las tablas utilizadas para gestionar dicho proceso.

En este punto, el proceso se encuentra en estado nuevo. Esto significa que se han construído todas las estructuras necesarias para la gestión del proceso, pero el proceso en sí no ha sido cargado en memoria principal, es decir, su código no ha sido cargado ni se ha reservado espacio en memoria principal para los datos del proceso. Cuando un proceso se encuentra en estado “Nuevo”, el programa permanece en almacenamiento secundario.

El sistema operativo puede mantener un proceso en estado “Nuevo” por razones de rendimiento o falta de memoria principal u otros recursos necesarios para la ejecución del proceso.

Analicemos algunas transiciones no triviales:

Ejecutando → Listo: la razón más común es que el proceso haya alcanzado su tiempo máximo ininterrumpido de ejecución (utilización del procesador). Otras razones pueden ser cambios en los niveles de prioridad de los procesos

Listo (o Bloqueado) → Saliente: imagine un escenario donde existe un proceso B que es hijo de un proceso A, y el proceso B se encuentra en estado Listo. Si el proceso A termina o ejecuta alguna instrucción que haga finalizar al proceso B, el mismo pasará de estado Listo a Saliente.

Tanto el estado Listo como el estado Bloqueado, manejan colas (FIFO) de espera para los procesos. De hecho, el sistema operativo crea colas por cada evento por el cual esperan los procesos. Cuando ocurre algún evento X todos los procesos en estado Bloqueado que esperaban por dicho evento son movidos al estado *Listo*.

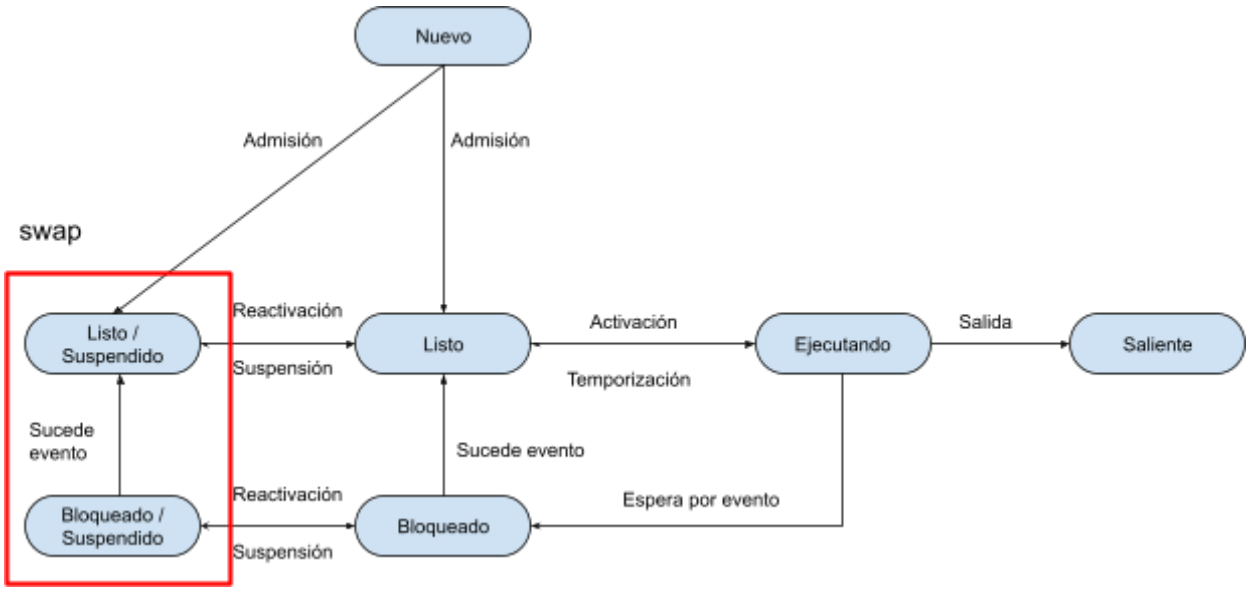
Modelo de 7 estados

Para entender el siguiente modelo, imagine por un momento que no existe la memoria virtual y por ende los procesos deben estar cargados completamente en memoria principal. Si en un momento determinado la memoria principal estuviese cargada por completo y todos los procesos se encontraran en estado *Bloqueado*, el procesador estaría ocioso y no se podrían admitir nuevos procesos.

Para resolver este problema surge el *swapping* (o memoria de intercambio) que implica mover una parte o todo el proceso de memoria principal a disco. Cuando ninguno de los procesos que se encuentra en memoria principal, se encuentra en estado *Listo*, el sistema operativo mueve uno de los procesos en estado *Bloqueado* a un nuevo estado que será el de *Suspendido* (disco). De esta manera se obtiene espacio para admitir un nuevo proceso en el sistema.

Tener en cuenta que el swapping es una operación de E/S.

Con este diseño se agregan nuevos estados y nuevas transiciones que reflejamos en el gráfico siguiente.



Algunas transiciones a destacar son:

Listo/Suspendido → Listo: cuando no hay procesos en estado listo el sistema tomará un proceso del estado Listo/Suspendido. Otro motivo puede ser que un proceso Listo/Suspendido tenga más prioridad que un proceso Listo

Nuevo → Listo/Suspendido: puede ocurrir que no haya espacio suficiente en memoria para el nuevo proceso y por eso el sistema operativo decide suspenderlo.

Bloqueado/Suspendido → Bloqueado: en ocasiones el sistema operativo puede traer a memoria principal un proceso suspendido basándose en estadísticas. Imagine que un proceso termina, liberando espacio en memoria y un proceso suspendido posee mayor prioridad que cualquiera de los procesos en estado Listo. Basándose en estadísticas el sistema operativo puede estimar que el evento que espera el proceso suspendido va a ocurrir y por esto es que un proceso suspendido en disco es traído a memoria principal.

Estructuras de control de procesos

Como mencionamos anteriormente, mucha de la información de un proceso, es utilizada por el sistema operativo para su gestión. Esta información, agrupada en el BCP, es cargada en los registros del procesador cuando el proceso está ejecutándose y vuelta a almacenar en el BCP cuando el proceso deja de ejecutarse (vuelve a estado Listo o es Bloqueado o Suspendido)

Ahora bien, además de estos atributos que se almacenan en el BCP ¿Qué información incluye la representación de un proceso?

Como mínimo, un proceso incluye un programa y posiciones de memoria para las variables locales, constantes y variables globales. Adicionalmente, un proceso incluye una pila que se utiliza para registrar las llamadas a funciones. En esta pila se almacenan las direcciones de memoria de retorno y los parámetros utilizados en cada llamada.

Podemos agrupar la información del bloque de control del proceso en 3 categorías:

- Identificación del proceso
- Información de estado del proceso: indica los contenidos de los registros del procesador cuando el proceso se está ejecutando. Algunos de estos registros son registros de control y estado, punteros de pila.
- Información de control del proceso: esta información es utilizada por el sistema operativo para controlar y coordinar los procesos activos.

Control de procesos

Modos de ejecución

El procesador posee 2 modos de ejecución, **el modo núcleo** asociado con el sistema operativo y **el modo usuario** asociado con los procesos de usuario. En modo núcleo, el software tiene el control absoluto del procesador (instrucciones, registros).

¿Cómo conoce el procesador en qué modo está ejecutando?

Dentro de la información de estado del proceso, se encuentra el PSW (program status word o palabra de estado del programa). El PSW contiene un bit que indica el modo de ejecución.

¿Cómo puede modificarse el modo de ejecución?

Este bit cambia en respuesta a determinados eventos como por ejemplo una llamada al sistema por parte del proceso que se está ejecutando.

Cambio de contexto (context switch)

A modo de introducción podemos decir que un cambio de contexto ocurre cada vez que el sistema operativo obtiene el control sobre el proceso que está actualmente en ejecución.

Un cambio de contexto puede ocurrir como respuesta a

- Una Interrupción

- Una Excepción
- Una llamada al sistema (system call)

En cualquiera de estos tres casos, el proceso en ejecución será interrumpido y el sistema operativo tomará el control para ejecutar la rutina de manejo de la interrupción, excepción o system call. Veamos algunas definiciones:

Interrupción

Una interrupción es un mecanismo mediante el cual el hardware del sistema se comunica con el sistema operativo. La característica más importante es que las interrupciones ocurren en momentos aleatorios durante la ejecución de un proceso, como mencionamos, en respuesta a señales del hardware. Cabe destacar que desde un proceso también se pueden generar interrupciones.

Algunas de las interrupciones más importantes son:

- **de reloj**: cuando ocurre, el sistema operativo pasa el proceso en ejecución (estado “Ejecutando”) a estado “Listo” cuando agotó su tiempo máximo de uso del procesador.

- **de E/S**: cuando ocurre, el sistema operativo determina si ocurrió algún evento por el cual están esperando 1 o más procesos y en tal caso, mueve todos los procesos en estado “Bloqueado” a “Listo” y los procesos en estado “Bloqueado/Suspendido” a “Listo/Suspendido”. El sistema operativo debe decidir si reanuda la ejecución del proceso que previamente se encontraba en estado “Ejecutando” o si lo expulsa y ejecuta otro proceso en estado “Listo” con mayor prioridad.

Excepción

Las excepciones ocurren cuando el procesador detecta una condición de error mientras ejecuta una instrucción como, por ejemplo, una división por cero. El procesador detecta una variedad de condiciones de error que incluyen protección de memoria, fallos de página y fallas internas de la máquina. En particular, una excepción es muy similar a una interrupción en cuanto a que se interrumpe la ejecución del proceso en ejecución, pero difiere en que una excepción se produce por una condición interna del proceso, mientras que una interrupción se produce por un evento externo al proceso.

El sistema operativo debe decidir si la excepción es irreversible, en cuyo caso el proceso se pone en estado “Saliente” y otro proceso será seleccionado para ser ejecutado.

Las excepciones podemos clasificarlas en:

- **Abort**: es una excepción que no siempre informa la ubicación precisa de la instrucción que causa la excepción y no permite reiniciar el programa o la tarea que causó la excepción. Se utilizan para informar errores graves, como errores de hardware y valores incoherentes o ilegales en las tablas del sistema.

- **Faults**: es una excepción que generalmente se puede corregir y que, una vez corregida, permite reiniciar el programa sin pérdida de continuidad. Cuando se informa de una falla, el procesador restaura el estado de la máquina al estado anterior al comienzo de la ejecución de la instrucción que falla. La dirección de retorno para el manejador de fallas apunta a la instrucción que falla, en lugar de a la instrucción que sigue a la instrucción que falla. Un ejemplo típico de esto es un page fault o fallo de página:

    - fallo de memoria: el procesador se encuentra con una referencia a una dirección de memoria que no se encuentra en memoria principal. El sistema operativo debe cargar el bloque (página o segmento) desde el disco en la memoria principal. Mientras todo esto ocurre el proceso que estaba en ejecución, es pasado a estado “Bloqueado”. Cuando el bloque de memoria es cargado, el proceso pasa a estado “Listo”.

- **Traps**: es una excepción que se informa inmediatamente después de la ejecución de la instrucción que generó el trap. Los traps permiten que la ejecución de un programa o tarea continúe sin perder la continuidad del programa. Es responsabilidad del sistema operativo analizar la naturaleza del trap y decidir si se reanuda la ejecución del proceso que la generó o no. La dirección de retorno del controlador de traps apunta a la instrucción que se ejecutará después de la instrucción que generó el trap. Ejemplos de trap pueden ser: breakpoint, división por cero, intento de acceso a memoria protegida.

Llamada al sistema (system call)

Cuando el proceso en ejecución realiza una llamada al sistema como, por ejemplo, apertura/lectura/escritura de archivos, el sistema operativo toma el control, salva el contexto del proceso en ejecución y ejecuta la rutina en cuestión. Al finalizar dicha rutina, restaura el contexto del proceso que realizó la llamada al sistema para que pueda continuar su ejecución.

Cambio de modo

Como mencionamos anteriormente, cuando ocurre una interrupción, se ejecuta una rutina del sistema operativo que trata dicha interrupción. Ahora bien ¿Qué ocurre si hay un proceso de usuario en ejecución?

Si había un proceso de usuario en ejecución, el procesador funcionaba en ***modo usuario*** y como la rutina de procesamiento de interrupciones es una rutina del sistema operativo, el procesador debe cambiar a ***modo núcleo*** (también llamado **modo kernel**) para poder ejecutar dicha rutina.

Ejecución del sistema operativo

Recuerde	• El sistema operativo es un conjunto de programas ejecutados por el procesador • El sistema operativo con frecuencia cede el control y depende del procesador para recuperar el control

Partiendo de la base de que el sistema operativo es un conjunto de programas (software como cualquier otro) y que depende del procesador para ser ejecutado en conjunto con el resto de los programas, existen 3 formas diferentes de abordar una solución al problema cómo controlar y organizar todos los programas que requieren ejecutarse:

Núcleo sin procesos

En esta solución, el sistema operativo se ejecuta por fuera de todo proceso de usuario, es más, el concepto de proceso tal y como fue detallado sólo aplica a procesos de usuarios. Con este diseño, cada vez que un proceso de usuario es interrumpido se realiza un cambio de contexto completo.

En esta alternativa, el sistema operativo es una única pieza de software separada del resto de los procesos de usuario.

Si volvemos a la unidad anterior, y analizamos los distintos diseños de sistemas operativos, encontraremos que existe una analogía con el diseño monolítico y/o en capas.

Ejecución dentro de los procesos de usuario

Esta alternativa, propone que el sistema operativo se ejecute virtualmente en el contexto del proceso de usuario. Con esta solución, una llamada al sistema se ejecuta dentro del entorno del proceso de usuario sin tener que realizar un cambio de contexto completo.

Cuando ocurre una interrupción, excepción o llamada a funciones del sistema, se salva parte del contexto del proceso y se realiza un cambio de modo del procesador para ejecutar las rutinas del sistema operativo, pero no se realiza un cambio de contexto completo, sino que se realiza un cambio de modo dentro del mismo proceso de usuario.

La ventaja primordial de esta solución, es que una vez que el sistema operativo finaliza su ejecución, si el proceso a restaurar es el mismo, no se incurre en un doble cambio de contexto.

Un concepto que se desprende de esto es que podemos ver a un proceso como un contenedor y que no necesariamente la relación entre proceso y programa es 1 a 1.

Para finalizar este modo de ejecución, podemos encontrar cierta analogía con la descripción que dimos en la Unidad I respecto de los microkernels.

Sistemas operativos basados en procesos

Esta última alternativa propone al sistema operativo como una colección de procesos de sistema. Al igual que en las otras alternativas, el software que forma parte del sistema operativo se ejecuta en modo núcleo. En esta alternativa, las principales funciones del sistema operativo se organizan como procesos independientes. Al igual que en la opción anterior, debe existir una porción del sistema operativo que se encuentre separada de todo proceso que brinde funcionalidad mínima para permitir el intercambio de procesos.

La ventaja de este tipo de solución es que impone una disciplina de diseño de programas que refuerza el uso de sistemas operativos modulares con mínimas y claras interfaces entre los módulos.

En este último caso, veremos que existe una semejanza entre este modo de ejecución y el diseño cliente-servidor de los sistemas operativos, descrito en la Unidad I.

Bibliografía utilizada

● William Stallings. Sistemas operativos. Pearson Education. S.A., Madrid, 2005. ISBN-84-205-4462-0.

● AndrewS.Tanenbaum. Modern Operating System. Pearson Education Inc., 2009. ISBN-Q-lB-filBMST-L

● Multilevel Feedback queue. Wikipedia, La enciclopedia libre, 2019 [consulta: 21 de marzo del 2019]. Disponible en https://en.wikipedia.org/wiki/Multilevel_feedback_queue