1 Funciones del sistema operativo informatico

Para que un ordenador pueda hacer funcionar un **programa informático** (también conocido como *aplicación* o *software*), debe contar con la capacidad necesaria para realizar cierta cantidad de operaciones preparatorias que puedan garantizar el intercambio entre el *procesador*, la *memoria* y los *recursos físicos* (periféricos).

Un **sistema operativo** (a veces también citado mediante su forma abreviada *OS* en inglés) es un programa que controla la ejecución de los programas de aplicación y que actúa como interfaz entre el usuario de un computador y el hardware real del mismo.

Otra definición más formal: Un *sistema operativo* puede ser contemplado como una colección organizada de extensiones *software* del *hardware*, consistentes en rutinas de control que hacen funcionar al computador y proporcionan un entorno para la ejecución de *programas*.

El *sistema operativo* se encarga de crear el vínculo entre los recursos materiales, el usuario y las aplicaciones (procesador de texto, videojuegos, etcétera). Cuando un *programa* desea acceder a un recurso material, no necesita enviar información específica a los dispositivos periféricos; simplemente envía la información al sistema operativo, el cual la transmite a los periféricos correspondientes a través de su *driver* (controlador).

De esta forma, el sistema operativo permite la "disociación" de programas y hardware, principalmente para simplificar la gestión de recursos y proporcionar una interfaz de usuario sencilla con el fin de reducir la complejidad del equipo.

1.1. Conceptos básicos. Definiciones

Antes de profundizar en los sistemas operativos vamos a detallar conceptos previos y definiciones que están relacionadas según el tipo de usuario que utiliza el sistema operativo.

1.1.1. Conceptos relacionados con los usuarios del sistema informático

Existen varios tipos de usuarios de un sistema informático desde el punto de vista del mundo de los sistemas operativos:

- usuario de órdenes
- usuarios programador
- usuario diseñador del sistema
- usuario administrador del sistema

Según los diferentes tipos de usuario existentes que manejan un sistema informatico tenemos varias definiciones importantes que debemos conocer previamente al estudio en profundidad de las funciones de un sistema operativo.

Conceptos relacionados con el usuario de órdenes:

- **Usuario**: elemento (persona, máquina) identificable por el sistema.
- **Sesión**: conjunto de acciones desarrolladas por el usuario desde que entra (login) hasta que sale (logout).
- **Programa**: conjunto de instrucciones destinadas a resolver un problema.
- **Fichero: (o archivo)**: conjunto de datos relacionados almacenados en almacenamiento no pedecedero.
- **Programa del sistema**: acciones relacionadas con el SO.
- **Intérprete de órdenes**: programa del sistema que recoge y manda ejecutar las órdenes del usuario.

Conceptos relacionados con el usuario programador:

- **Llamadas al sistema**: Mecanismo que utilizan los programas de aplicación para solicitar que el sistema operativo haga algo.
- **Niveles de ejecución**: Distintos modos de ejecución del procesador, que determinan que instrucciones se pueden ejecutar en cada momento. Los programas de usuario se ejecutan en modo normal, mientras que el código del sistema operativo lo hace en modo privilegiado.

Conceptos relacionados con el usuario diseñador:

- **Sistema de gestión de procesos**: encargado de crear, eliminar, suspender, reanudar, comunicar y sincronizar procesos.

- **Sistema de gestión de memoria**: encargado de la memoria principal
  - Controla particiones libres/ocupadas
  - Asigna/libera espacios.
  - Llama a la memoria principal.
- **Sistema de gestión de E/S**: encargado de los dispositivos de E/S. Permite su compartición ordenada, minimiza efectos de diferencia de velocidad, uniformiza distintos dispositivos.
- **Sistema de gestión de ficheros:** encargado de los ficheros. Define:
  - Concepto y tipos de ficheros.
  - Gestiona almacenamiento y operaciones.
- **Núcleo (kernel) del sistema operativo:** programa individual que siempre está cargado en memoria principal y que se está ejecutando permanentemente en el computador.

Conceptos relacionados con el administrador de recursos / sistema

- **Procesos**: programas en ejecución que compiten por el uso de recursos. **Cualquier programa en ejecución**.
- **Recursos**: Cualquier componente fisico o virtual de disponibilidad limitada del sistema informático.

1.1.2. Concepto de recurso

Un *proceso* necesita ciertos recursos para realizar satisfactoriamente su tarea. Un **recurso**, (o *recursos del sistema*), es cualquier componente fisico o virtual de disponibilidad limitada en un sistema informático. Cada dispositivo conectado a un sistema informático es un *recurso*. Cada componente interno del sistema es un *recurso*.

Los *recursos virtuales* del sistema incluyen ficheros, panel de control y áreas de memoria.

Estos recursos computacionales habitualmente son:

- **Tiempo de CPU**. Necesario para ejecutarse, ya que el encargado de ejecutar las instrucciones es esta unidad.
- **Memoria**. El proceso necesita memoria tanto para albergar su propio código (el que va a ejecutar) como para acceder y modificar la estructura de datos asociados al proceso.

- **Archivos**. Un proceso puede requerir acceso a un fichero alojado en la memoria secundaria de almacenamiento para recuperar información. También puede ser al reves, que el proceso lo que necesite sea escribir datos en un medio no volátil para almacenarlos de forma permanente.
- **Dispositivos de E/S**. Un proceso puede solicitar acceder a un dispositivo de entrada o de salida: leer un dato desde el teclado, mostrar información por pantalla, controlar el funcionamiento de la impresora, etc.

Los **recursos** se asignan a un proceso en dos momentos:

- Cuando se crea.
- Durante su ejecución.

1.2. Los procesos

No hay que confundir *proceso* con *programa*. Un **proceso** es **cualquier programa en ejecución**. Es ejecutado por el usuario o el sistema.

Generalmente en el caso de que lo ejecute el sistema se suele denominar **servicio** en sistemas *Windows*, aunque también puede recibir el nombre de **demonio** en entornos *UNIX* y *Linux*.

Cada proceso se compone de un código que se ejecuta (programa), y unas estructuras de datos, estando ambos cargados en memoria. Se puede definir un proceso como una instancia de un programa en ejecución, de manera que el programa lleva los datos asociados a cada ejecución del mismo. Por ejemplificar esto de manera simple ponemos el siguiente caso: podemos abrir dos veces un programa como la calculadora de Windows (dos instancias del programa *calc.exe* en Windows). Son dos procesos diferentes y tienen asociados datos diferentes en sus diferentes ejecuciones.

Programa	**Proceso**
Estático.	Dinámico.
No tiene contador de programa.	Tiene un contados de programa.
Existe desde que se instala hasta que se borra	Su ciclo de vida comprende desde que se activa hasta que termina.

Cuadro 1.1: Diferencias entre proceso y programa

La estructura de datos asociada al proceso sirve para identificar unívovamente cada proceso y controlar todos los aspectos de su ejecución. Más adelante definiremos y profundizaremos mas en esta estructura de datos.

1.2.1. Estados de un proceso

En cualquier sistema operativo es básico conocer el comportamiento que exhibirán los distintos procesos y el conjunto de estados que pueden atravesar.

1.2.1.1. Modelo de dos estados

Es el modelo mas sencillo que puede construirse. Un proceso puede estar ejecutándose en el procesador o no. Así pues, un proceso puede estar en dos estados: *Ejecución* o *No ejecución*.

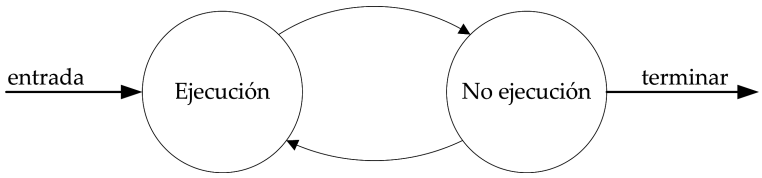

Figura 1.1: Esquema de un modelo de dos estados de un proceso

Cuando el SO crea un nuevo proceso, éste entra en el sistema en el estado de *No ejecución*. De este modo, el proceso existe, es conocido por el SO y está esperando la oportunidad de ejecutarse. En un momento dado, el sistema operativo decide otorgar el procesador a un proceso determinado con lo que dicho proceso pasará de estado *No ejecución* a *Ejecución*.

Cada cierto tiempo, el proceso en ejecución es interrumpido y el sistema operativo seleccionará un nuevo proceso para que tome el control del procesador. El proceso interrumpido pasa del estado de *Ejecución* al de *No ejecución* mientras que el proceso elegido realiza la transición inversa.

Incluso en este modelo tan simple, se aprecian ya algunos de los elementos importantes en el diseño de SSOO. Cada proceso debe representarse de forma que el sistema operativo tenga conocimiento de su estado actual y de su posición en memoria.

Aquellos procesos que no estén en estado de ejecución deberán almacenarse en algún tipo de estructura de datos mientras esperan que el sistema operativo les otorgue el control sobre el procesador. La estructura más básica para realizar esta función sería una cola de procesos.

Dicha cola consiste en una lista enlazada de bloques en la que cada uno de estos bloques representa a un *proceso*. Cada bloque consistirá en un puntero a la estructura de datos donde el SO guarda toda la información relativa al proceso.

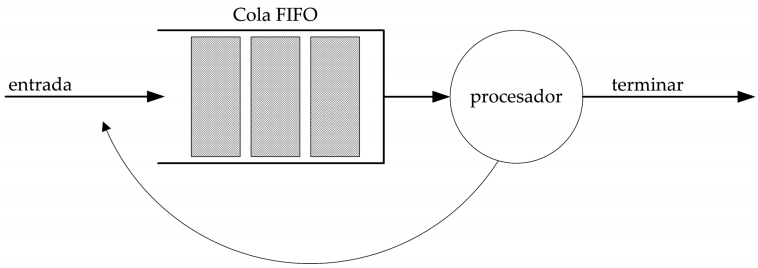

Figura 1.2: Esquema de un sistema de cola FIFO

Cada vez que el SO cree un nuevo proceso se introducirá el correspondiente bloque al final de la cola, acción que también se llevará a cabo cuando un proceso sea expropiado del procesador en favor de otro o cuando se bloquee en espera de que se complete una operación de E/S.

Cuando un proceso termine su ejecución, será descartado del sistema.

Una cola es una lista "primero en entrar, primero en salir" (*FIFO, First-in, First-out*) y el procesador opera según un turno rotatorio (*round-robin*) con todos los procesos disponibles. otorgando a cada proceso una cierta cantidad de tiempo para ejecutar y luego volviendo a la cola

Si todos los procesos estuvieran siempre listos para ejecutar, la disciplina de comportamiento de cola presentada sería eficaz, ya que el procesador irá ejecutando siguiendo un turno rotatorio con los procesos disponibles tal y como hemos descrito. De esta forma, a cada proceso de la cola se le otorga una cierta cantidad de tiempo para ejecutar. Si no se presentan bloqueos y transcurrido éste volverá a la cola para optar de nuevo a tener control sobre el procesador.

Sin embargo, **esta implementación no es adecuada** debido a que algunos procesos en estado de no ejecución estarán listos para ejecutar mientras que otros se encontrarán a la espera de obtener algún recurso solicitado o a que se complete una *operación de E/S*. Así pues, utilizando una cola sencilla el SO podria no seleccionar exactamente el proceso que está en el extremo mas antiguo de la cola. Más bien, el SO tendría que recorrer la lista buscando el proceso que no esté bloqueado y que lleve más tiempo en espera.

1.2.1.2. Otros modelos de estados

Una forma de afrontar esta situación (nada óptima) es dividir el estado de no ejecución en dos; los estados **listo** y **bloqueado**.

Así se consigue que mientras un proceso espera por un recurso el sistema operativo pueda ejecutar otra tarea, optimizando así el procesador y utilizándolo todo lo posible.

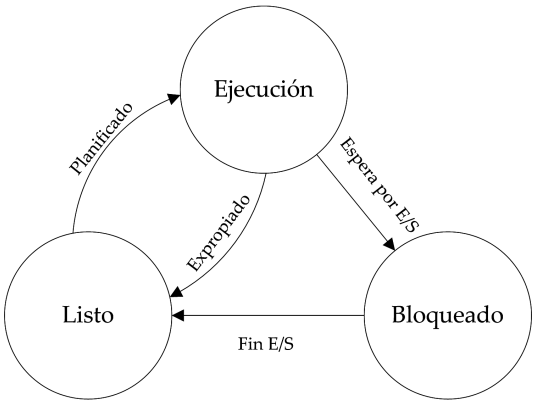

Figura 1.3: Diagrama básico de estados de un proceso.

Existen muchos otros modelos de estados más reales y con más características que el modelo de tres estados. Sin entrar mucho en detalle, es lógico pensar que este diagrama no contempla muchas de los estados habituales de los procesos con los que los usuarios están acostumbrados a trabajar.

Por ejemplo, en el modelo de tres estados no es posible *suspender* un proceso (desactivar su ejecución hasta nueva orden, pero sin matarlo). Tampoco permite estados en los que el proceso ha terminado su ejecución pero que por algunos motivos no queremos que desaparezcan de inmediato del sistema las estructura de datos asociadas. Pongamos que un programa de utilidades necesita extraer información sobre la historia del proceso con fines relativos a un análisis de uso o de rendimiento. Esto no sería posible si el SO eliminase del sistema toda la información relativa al proceso de manera inmediata tras finalizar su ejecución.

Además, podemos necesitar lanzar un proceso en un momento determinado y que el sistema no tenga memoria disponible en ese preciso momento. De cara al usuario sería muy contraproducente no ejecutar su tarea y pedirle que la vuelva a ejecutar en otro momento de forma manual. Peor todavía si ese proceso que se necesita ejecutar se corresponde a un proceso lanzado por otro proceso ya en ejecución o si lo ejecuta el propio sistema operativo sin intervención del usuario. En cualquier caso, denegar la ejecución por falta de recursos es inviable.

La mejor forma de afrontar este problema por el sistema operativo es *comprometerse a ejecutarlo cuando pueda.* Así, el SO puede crear el proceso y asignarle los recursos correspondientes a falta de memoria, y albergarlo en algún sitio en modo *espera* hasta que tenga recursos suficientes para su ejecuón disponibles (fundamentalmente y como indicamos suele ser memoria disponible).

La siguiente figura muestra un diagrama de estados más cercano a la realidad, donde se contemplan los *estados suspendidos* de los procesos (no cargados en memoria principal), y el estado de *creado / nuevo*, donde se asignan los recursos correspondientes y se crean las estructuras de datos asociadas al proceso aunque el SO todavía no se comprometa a ejecutarlo.

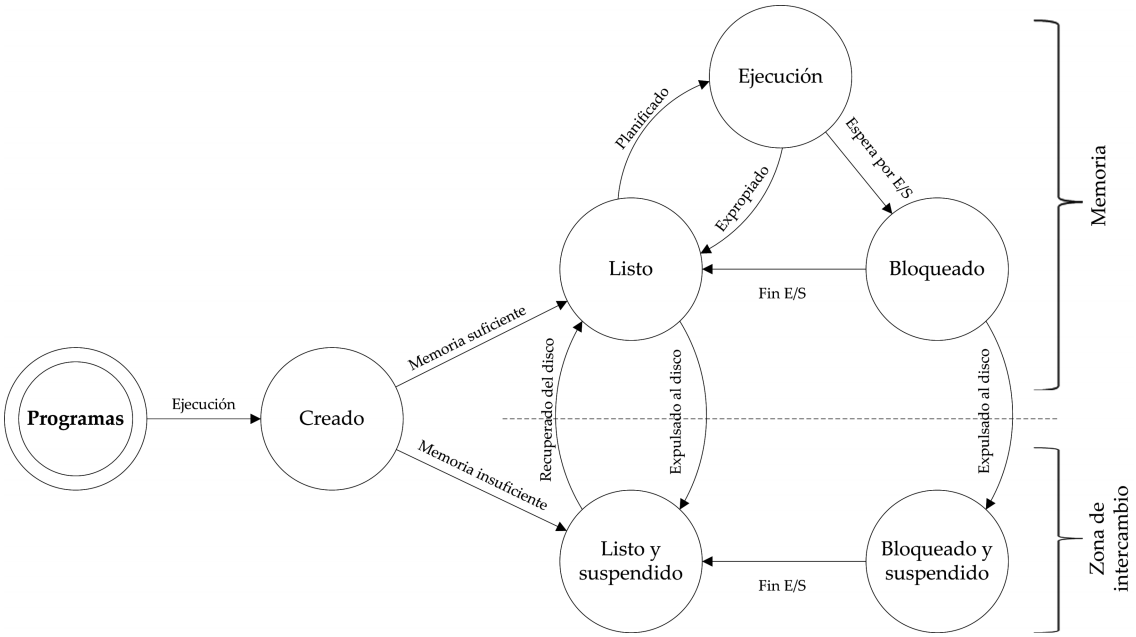

Figura 1.4: Diagrama más completo de estados de un proceso.

En este diagrama se separan la *zona de memoria* (memoria principal) de la *zona de intercambio*. La zona de intercambio es un recurso de los sistemas operativos actuales para garantizar la ejecución de los procesos aunque no tengamos suficiente memoria disponible. Esta zona de intercambio también se conoce como **memoria virtual** del sistema, y se realiza a través de *swapping*[1].

Su justificación es la siguiente: si todos los procesos están en bloqueados esperando un suceso y no hay memoria disponible para nuevos procesos, el procesador estará desocupado. La solución pasa por suspender el proceso, para garantizar la ejecución de más procesos. Ademas, así "ampliamos" la memoria principal, habilitando un lugar mas grande donde albergar la información de los procesos creados que de otra forma sería imposible de alojar.

Sin embargo este diagrama presentado no contempla el estado **zombie** (donde un proceso ha terminado su ejecución pero todavía permanecen la estructura de datos asociada al mismo en el sistema), habitual en prácticamente todos los sistemas actuales.

[1]El ***swapping*** Es un mecanismo o modo de interrelacionar la memoria principal (la que contiene el proceso en ejecución, los datos de proceso inmediato y los resultados intermedios) con la secundaria, de tal modo que se produce un intercambio de procesos entre ambas cuyo resultado es la simulación de un sistema multitarea o la potenciación de memoria central a base de recursos de la memoria secundaria.

Actividad práctica

1. Busca información sobre los modelos de 5 estados, 6 estados y 7 estados.
   a) Realiza los gráficos correspondientes
   b) Comenta las diferencias entre ellos.
2. Busca información sobre el modelo de estados de procesos en entornos UNIX.
   a) Realiza el gráfico correspondiente.

1.2.2. Escructura de datos de un proceso

El sistema mantiene toda la información sobre un proceso en una estructura llamada **bloque de control de procesos** (***PCB*** - *Process Control Block*).

Cuando se crea un proceso el sistema operativo crea una estructura PCB que sirve para describirle hasta que se elimina. En esta estructura se almacena la información que necesita el sistema para controlar al proceso, identificarle unívocamente y dar cuenta de sus recursos y toda la información relativa a la ejecución del mismo.

Habitualmente el *PCB* consta de los siguientes campos:

- **PID**: El identificador del proceso. Cada tabla posee uin identificador único llamado *PID* que sirve de localizador. Por otro lado, los procesos que se encuentrarn en estado ejecución y crean otros procesos pasan a ser denominados *procesos padres.* Los *procesos hijo* utilizan los *PID* para identificar al padre.
- El **estado** del proceso (activo, preparado, bloqueado...).
- El **estado hardware** (contador de programa, punteros de pila, etc).
- La información para **gestionar la memoria** (punteros, tablas, registros...).
- La información del **estado del subsistema de E/S** (dispositivos asociados al proceso, lista de archivos abiertos, etc).
- La información de **planificación**. Esta información incluye los punteros a las colas de planificación y cualesquiera otros parámetros de planificación que se requieran.
- La **prioridad**. Habitualmente los sistemas operativos cuando realizan la planificación sobre qué proceso debe ser el siguiente en obtener los recursos del procesador tienen en cuenta además del turno en la cola correspondiente en la que se encuentren la *prioridad* que tiene cada uno de los procesos. Así el sistema podrá lanzar unos procesos antes que otros, independientemente del organ natural en el que se encuentren.

Figura 1.5: Bloque de Control de Procesos (***PCB*** - *Process Control Block*)

Cuando el sistema operativo cambia la tarea asignada al procesador utiliza una zona del *PCB* en la que deja toda la información necesaria para que pueda proseguir la ejecución cuando el procesador retome la tarea. Por ello no sólo se guarda el contador de programa, sino todos los registros que contengan resultados intermedios y el resto de registros que influyan en la ejecución, tales como los registros de condición y los punteros de pila.

El sistema operativo mantiene una lista de *PCB* para cada uno de los estados del sistema. Cada proceso pertenece a una única lista. Así posee una lista de los procesos que están en el estado *suspendido*, una lista de procesos que están *bloqueados*, una lista de procesos que están *suspendidos,* etc.

Si el sistema es uniprocesador la lista de procesos activos se reduce a una sola entrada. Normalmente las listas de los estados *preparado* y *bloqueado* se organizan como listas enlazadas.

1.3. Los archivos.

EL sistema operativo proporciona una visión uniforme para todos los sisteams de almacenamiento, definiendo unidades lógicas de almacenamiento denominadas **archivos** (también llamados *ficheros*), que no es más que un conjunto de información relacionada definida por su creador.

Los archivos contienen información binaria, pero segun el tipo y su formato, la interpretación será diferente: instrucciones (programas), caracteres (archivos de texto),