CFGS DAM

Primer Curso

Programación de Servicios y Procesos

UT1: Sistemas Multitarea

UT1: Sistemas Multitarea

- **Sistemas multitarea**
  - **Conceptos básicos. Tipos**
  - **SSOO Multitarea**
  - **Tipos de programación sobre sistemas multitarea**
    - Secuencial, concurrente, paralela y distribuida
- **Procesos**
  - **Concepto de proceso**
  - **Estados de un proceso**
  - **Conmutación de procesos**
  - **Gestión de procesos con Java**
- **Planificación de procesos**
  - **Planificador**
  - **Despachador**
- **Hilos o hebras**
  - **Definición**
  - **Características**
  - **Implementación**

Introducción

- En este tema aprenderemos
  - Conceptos básicos importantes
  - Que son y para que sirven los Procesos e hilos, base de esta asignatura.
  - Como realizan su gestión/planificación los SSOO, en especial los SSOO multitarea.
  - Comandos para ello en Windows y Linux

Sistemas Multitarea (SM)

- La multitarea es la característica de los SOPs que permite que *varios procesos o aplicaciones se ejecuten aparentemente al mismo tiempo, compartiendo uno o más procesadores.*
  - *Nota: Esto se puede extrapolar a una aplicación en ciertos lenguajes como Java o C#, que, como veremos a lo largo del curso, pueden ejecutar varias tareas internas "a la vez".*

- Por lo tanto, un Sistema multitarea es aquel sistema que permite la ejecución de varias tareas "de forma simultánea" y transparente al usuario.
  - Ejemplos: Windows, Linux o MAC

Sistemas Multitarea (SM)

- Objetivo PSP:

  Crear programas que aprovechen las capacidades de un SM

  =

  aquellos capaces de dar servicio a más de un proceso a la vez .

SM. Historia

- A principios de la década de 1960, primero Conway y luego Dennis y Van Horn, presentaron la idea de múltiples hilos de control y algunos de los problemas de acceso a recursos compartidos, como la memoria.

- Posteriormente **Dijkstra** instigó el estudio de la programación concurrente en su artículo "*Procesos secuenciales de cooperación*", publicado en 1967.
  - En ese artículo vemos la introducción de algunos problemas ahora bien conocidos como "The Dining Philosophers "," The Sleeping Barber "y" The Dutch Flag Problem ", y quizás lo más importante, el problema de la sección crítica y su solución utilizando semáforos.
  - El mismo artículo introduce la noción de interbloqueo y presenta un algoritmo que puede detectar la posible presencia de bloqueos.

SM. Conceptos Básicos

Para poder empezar a entender cómo se 'ejecutan' varios programas a la vez, es imprescindible adquirir ciertos conceptos básicos.

- **Programa/Aplicación:**
  - Se puede considerar un programa a toda la información almacenada en una o varias carpetas del disco que resuelve una necesidad concreta para los usuarios.

- **Proceso:**
  - De manera muy simplificada puede definirse "proceso" como un **programa en ejecución**.
  - Para que un proceso se pueda ejecutar tiene que estar en memoria toda la información que necesita (código, datos, CP, …).
  - Es importante destacar que los procesos son entidades independientes, aunque ejecuten el mismo programa/aplicación.
    - De tal forma, pueden coexistir dos procesos que ejecuten el mismo programa, pero con diferentes datos.
    - Así, por ejemplo, se pueden tener dos instancias del programa Microsoft Word ejecutándose a la vez, modificando cada una un fichero diferente.

SM. Conceptos Básicos

- **Ejecutable**:
  - Contiene la información necesaria para crear un proceso a partir de un programa.
    - Es decir, llamaremos "ejecutable" al fichero que permite poner el programa en ejecución como proceso.

- **Demonio**:
  - Proceso no interactivo que está "ejecutándose" continuamente en segundo plano.
    - Es decir, es un proceso controlado por el sistema sin ninguna intermediación del usuario.
  - Suelen proporcionar un servicio básico para el resto de procesos.
  - Ejemplo: El recolector de basura de Java

SM. Conceptos Básicos

- **Sistema operativo:**
  - Programa que hace de intermediario entre las aplicaciones del usuario y el hardware del ordenador.

  - Funciones principales:
    - Ejecutar los programas del usuario, creando los procesos y gestionando su ejecución para evitar errores y mejorar el uso del equipo.
    - Hacer de interfaz entre el usuario y los recursos del ordenador,
      - Esto permite al programador acceder a los recursos hardware de manera 'sencilla'.
    - Utilizar los recursos del computador de forma eficiente.
      - El sistema operativo es el encargado de repartir los recursos entre procesos en función de las políticas que tenga.

SM. Conceptos Básicos

Sistemas operativos. Clasificación

Se pueden clasificar en base a 2 criterios:

a) **Según las tareas que pueden ejecutar**
   - MonoTarea
     - Solo pueden ejecutar una tarea de forma simultánea
     - Ej. Los regidos por MS-DOS
   - MultiTarea
     - Pueden ejecutar múltiples tareas simultáneas de forma real o simulada.

b) **Según el número de procesadores que pueden usar**
   - Monoprocesador
   - Multiprocesador / Multinúcleo
     - Sistemas de memoria compartida
     - Sistemas de memoria distribuida

SSOO Multitarea.

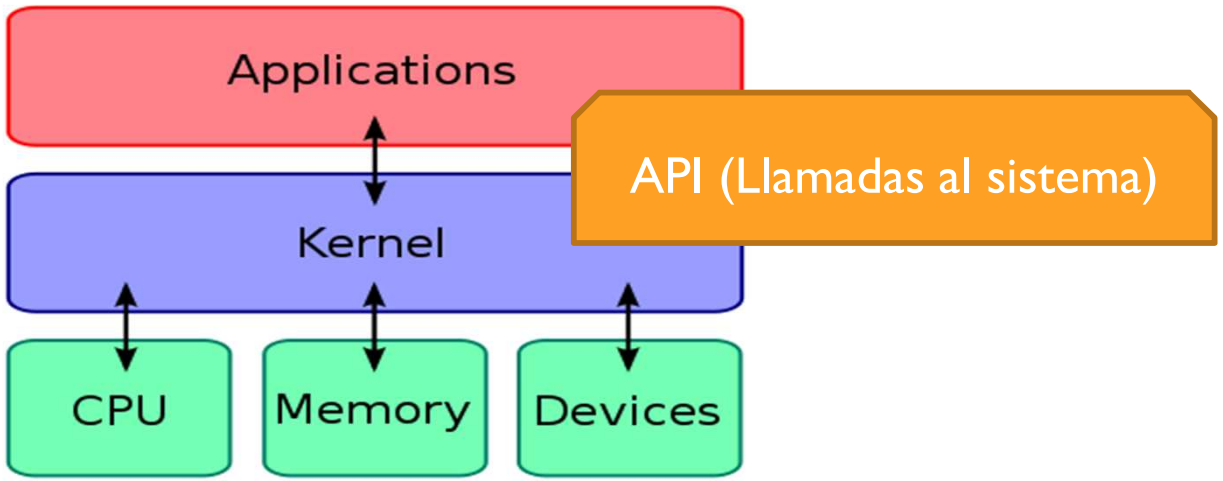
Tipos

- Un sistema Multitarea es aquel que permite la ejecución "de varias tareas a la vez".
- Si esto lo extrapolamos a un sistema operativo nos saldrán los SSOO Multitarea.
- 3 Tipos:
  - **Cooperativo**.
    - Los propios procesos de usuario son quienes, "gentilmente", ceden la CPU al sistema operativo a intervalos regulares.
    - Son bastante problemáticos: si el proceso de usuario se interrumpe y no cede la CPU al SO, todo el sistema quedará en espera, sin poder hacer nada.
    - Ejemplo: Windows 3.1
  - **Preferente o Preemptivo.**
    - El sistema operativo es el encargado de administrar el/los procesador(es), repartiendo el tiempo de uso de éste entre los procesos que estén esperando para utilizarlo. Es decir, la CPU da preferencia a los procesos del sistema operativo.
    - Da la sensación de estar ejecutando varios procesos a la vez.
    - Ejemplos: Unix y sus derivados (FreeBSD, Linux), VMS y Windows NT y posteriores.
  - **Real**
    - Es aquel en la que varios procesos se ejecutan realmente al mismo tiempo, en distintos micros.
    - Si hay más procesos que micros, normalmente se usa la política preferente o preemptiva.
    - Ejemplos: Mac OS X, Unix, Linux, Windows XP,/7/8 y superiores etc.

SSOO Multitarea.

Funcionamiento

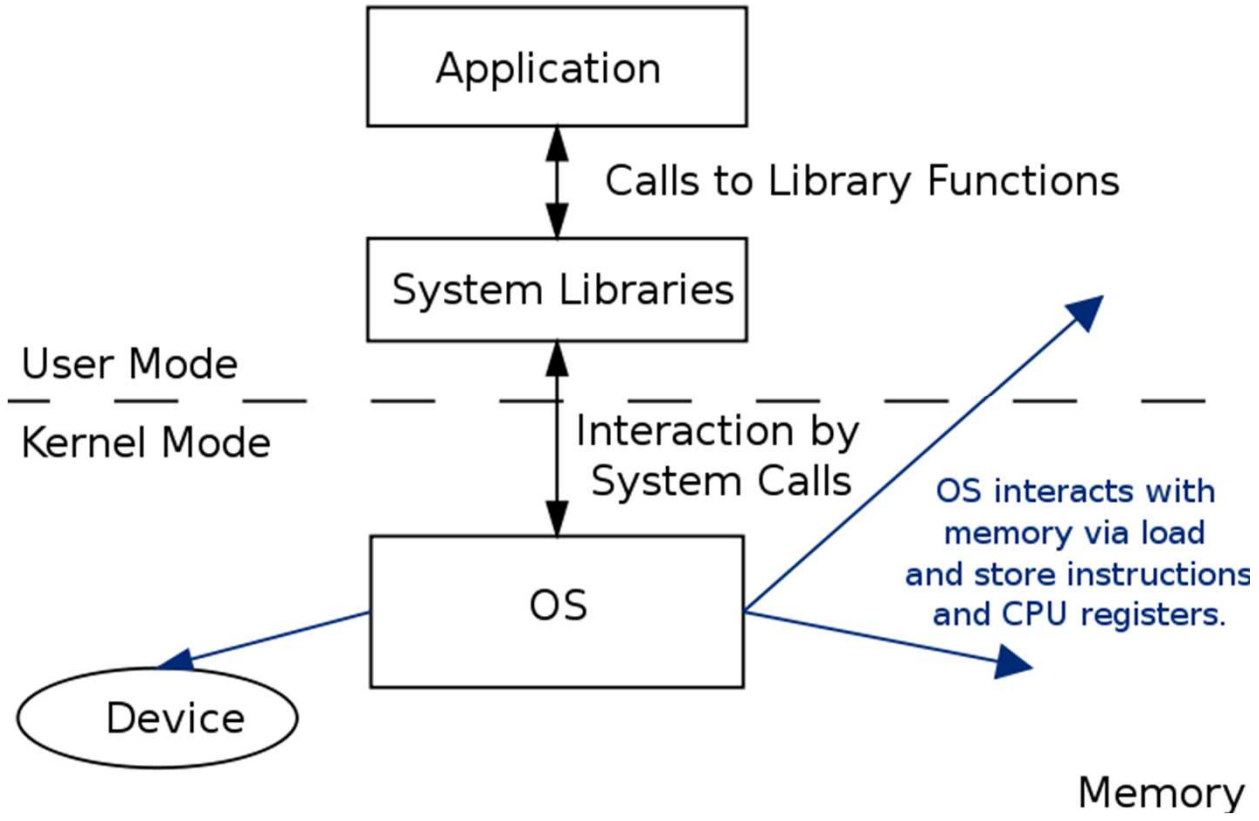
- Una aplicación puede usar los servicios del SSOO a través de las API(s) que este proporcione.
  - Estas API's están escritas en el lenguaje con el que se creó el SSOO (C/C++ normalmente)
  - Se dice entonces que las aplicaciones acceden al SSOO a través de las denominadas "llamadas al sistema", consistentes en llamar a una o más funciones del API.



SSOO Multitarea.

Funcionamiento II

- El esquema anterior, donde las aplicaciones llaman directamente al kernel, ya no se usan.
- Hoy en día, es más complejo:
  - Modelo básico:



SSOO Multitarea.

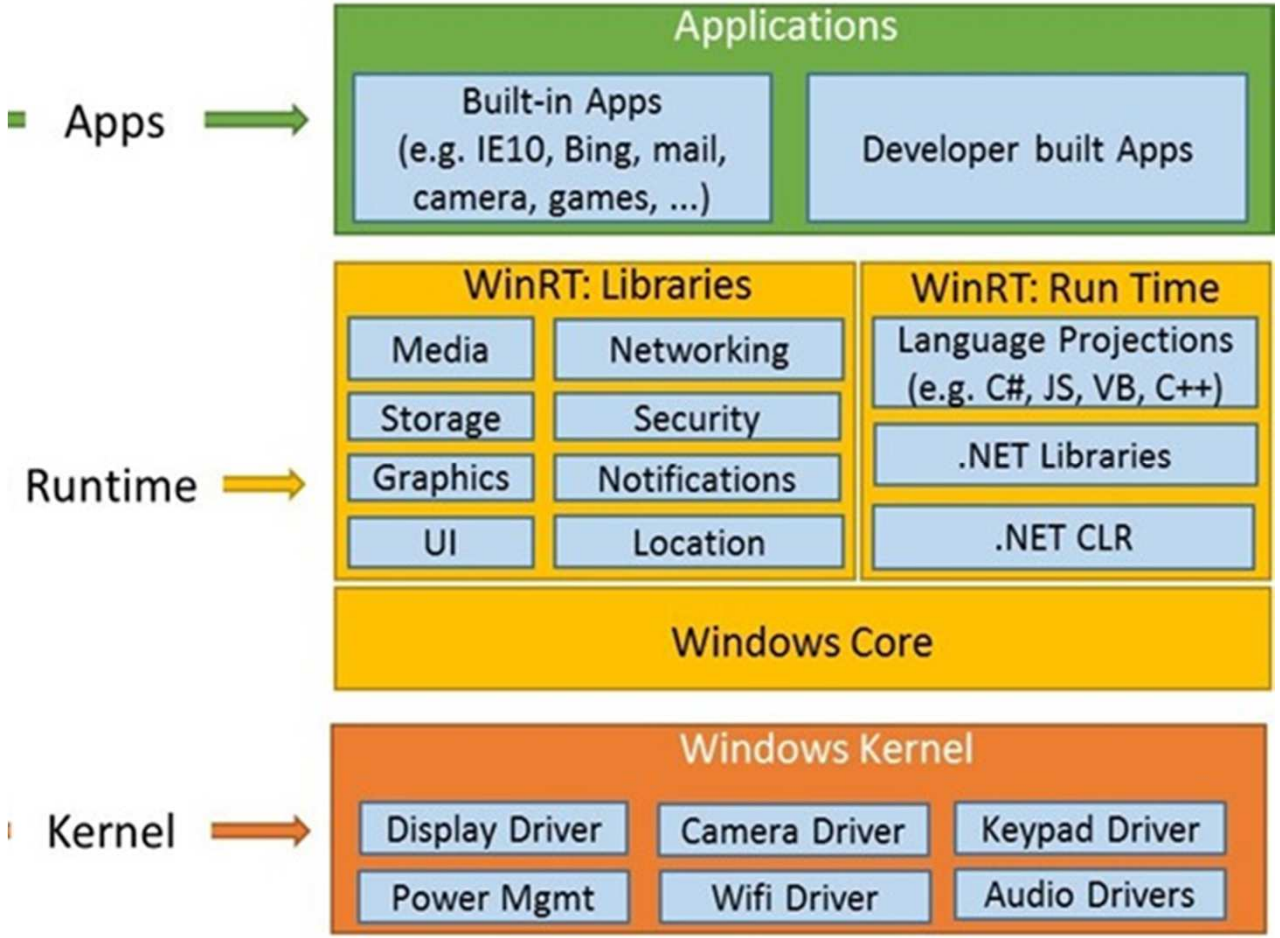
Funcionamiento III

- La mayoría de los SSOO actuales disponen, al menos, de un modo dual de operación (kernel/user mode) para proteger las partes "sensibles" del sistema.
- Así:
  - El código de los usuarios se ejecuta en modo usuario.
    - Menor prioridad
    - Menores permisos de escritura en memoria
    - Interrumpible (su ejecución) en cualquier momento por el SSOO
  - El código del SSOO se ejecuta en modo kernel.
    - Mayor prioridad
    - Mayores permisos de escritura en memoria
    - No siempre interrumpible
- *Nota: Linux siempre ha tenido modo dual, pero Windows lo ha adaptado porque en un principio no lo tenía y los usuarios podían ejecutar programas "maliciosos" fácilmente.*

SSOO Multitarea.

Funcionamiento IV

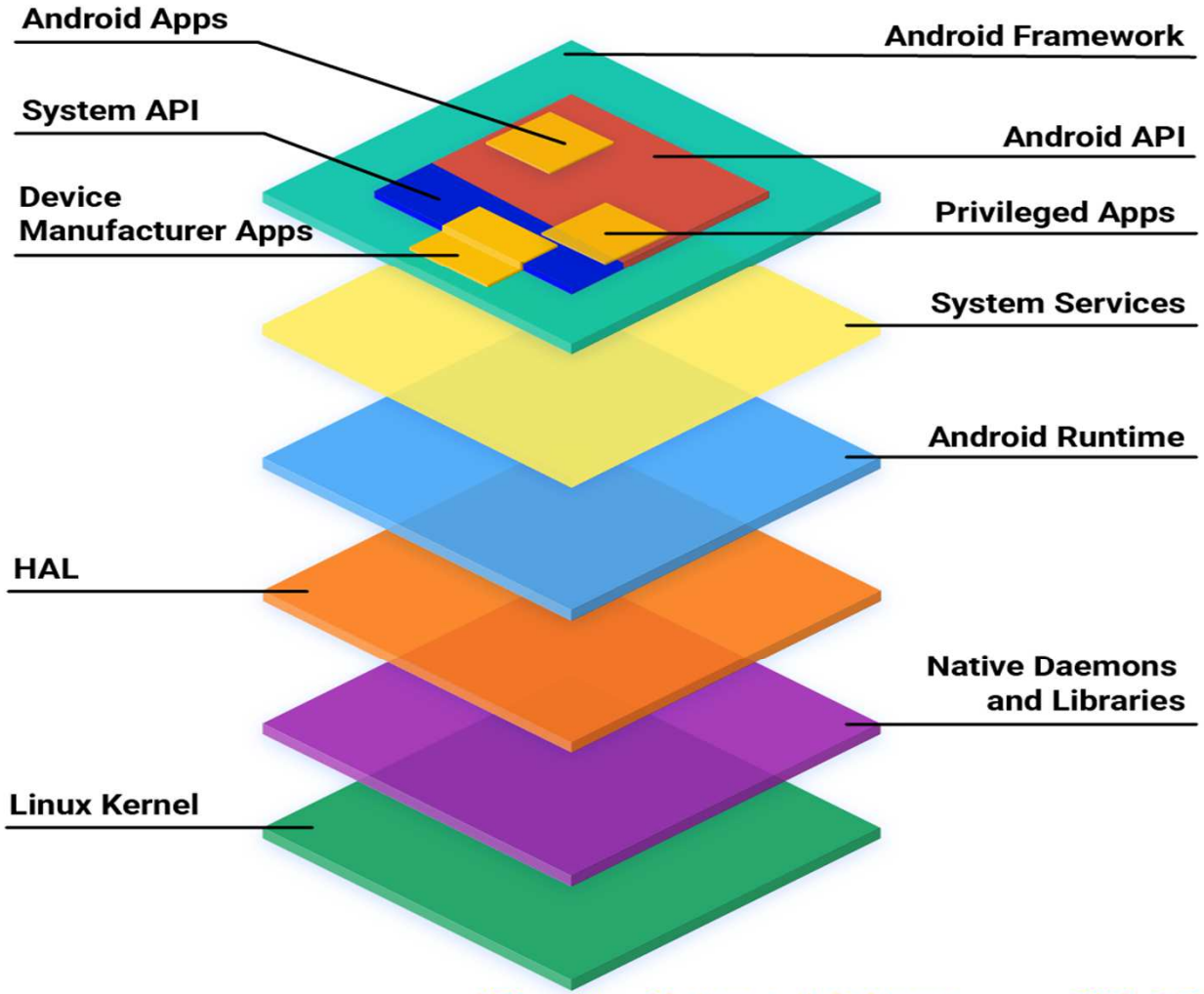
- Este esquema puede ser mucho más complejo, como x ej en Windows 8 y superiores:



SSOO Multitarea

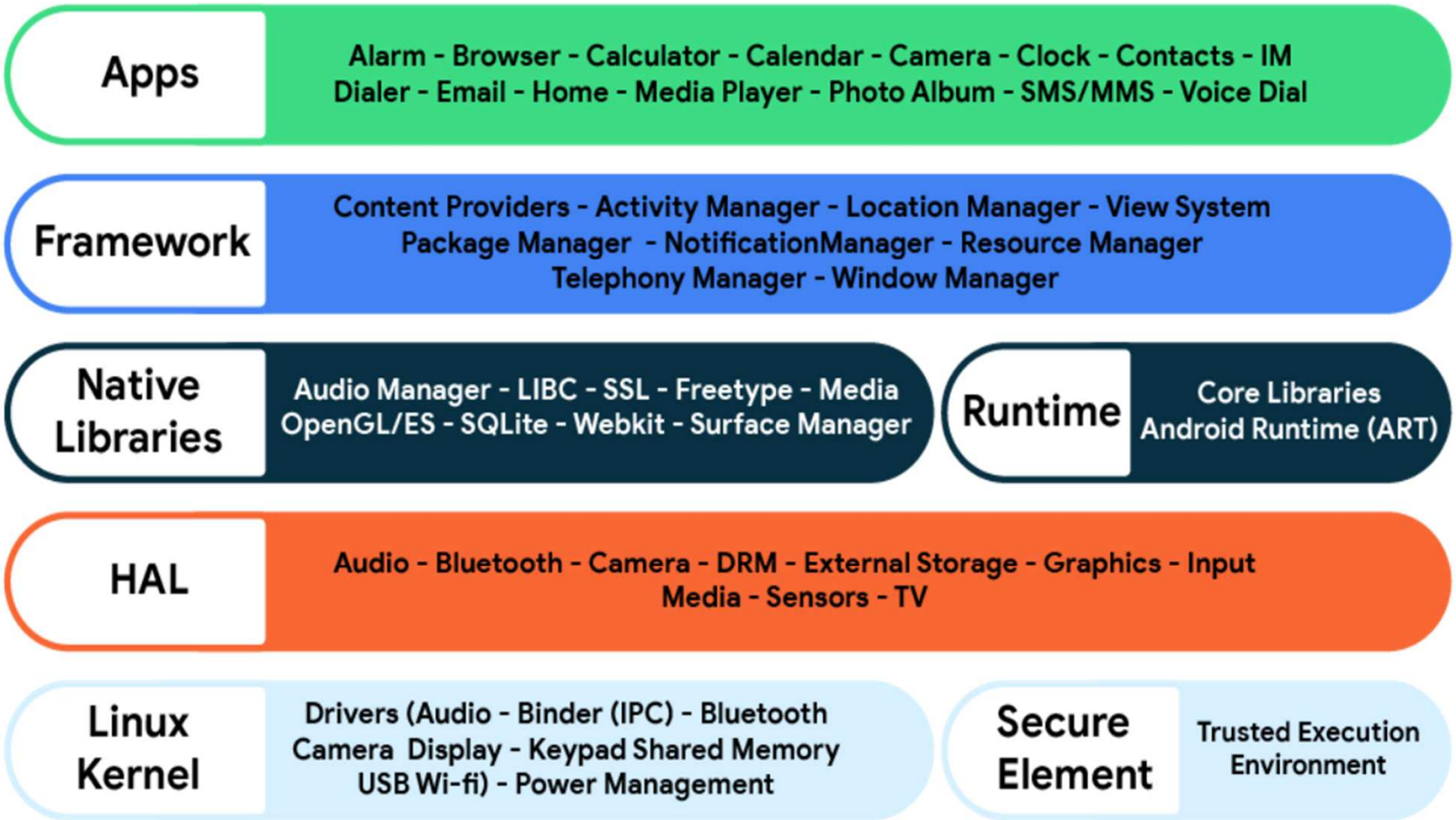
Funcionamiento V

- Android:



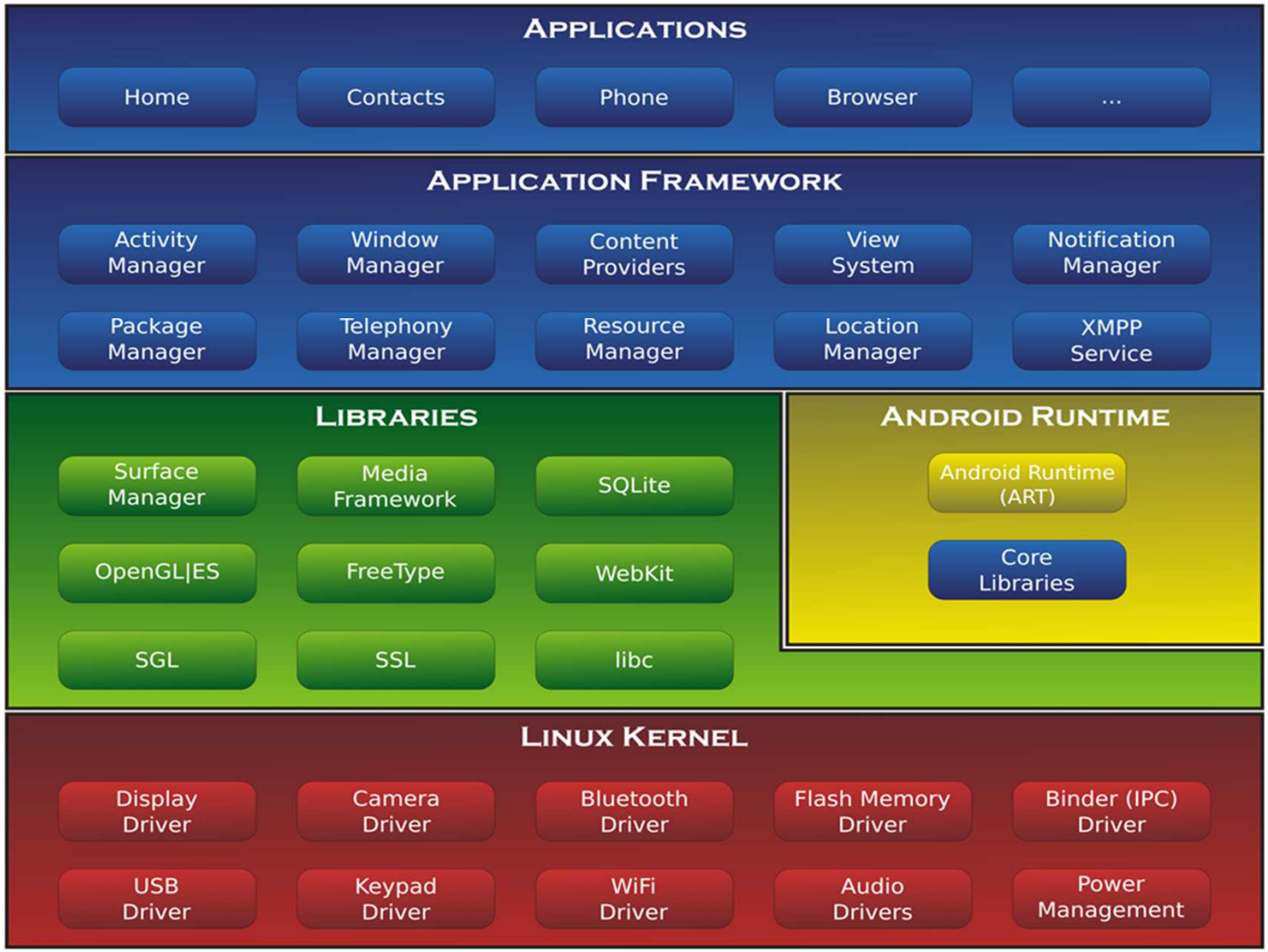
SSOO Multitarea

Funcionamiento VI



SSOO Multitarea

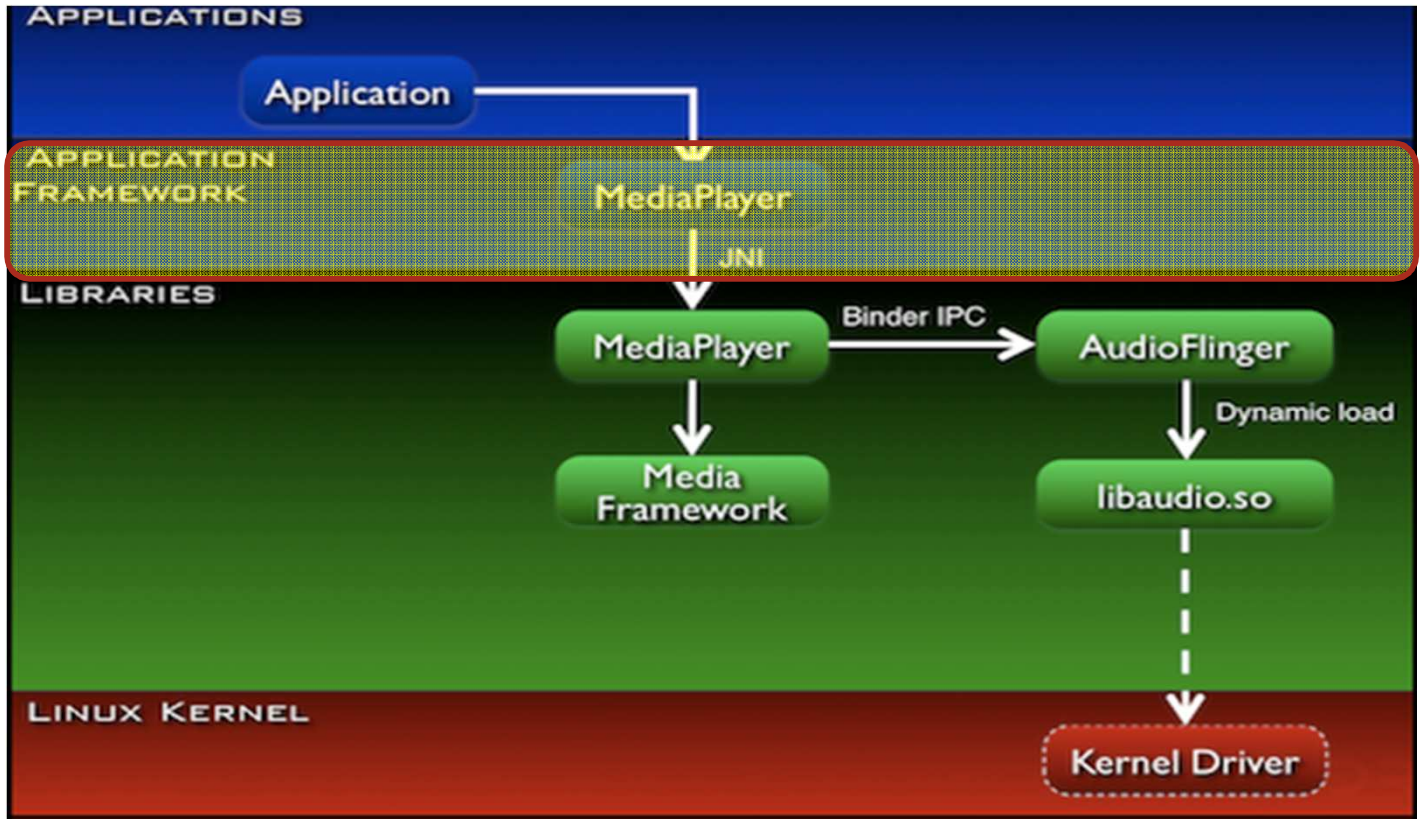
Funcionamiento VII



SSOO Multitarea.

Acceso por parte de las aplicaciones

- Hoy en día, por tanto, el programador no suele utilizar las llamadas al sistema directamente, sino que utiliza una API de más alto nivel (como la MediaPlayer de Java) que le facilita el trabajo.
- Ejemplo: *Como reproducir un sonido/canción en Android:*



*Java Native Interface (**JNI**) es un framework de programación que permite que un programa escrito en Java ejecutado en la máquina virtual java (JVM) pueda interactuar con programas escritos en otros lenguajes como C, C++ y ensamblador.

Sistemas Multitarea.

Tipos de programación sobre estos sistemas

Sobre los Sistemas Multitarea podemos ejecutar 4 tipos o técnicas de programación:

- *Secuencial*
- *Concurrente*
  - Multiprogramación
  - Programación Paralela
  - Programación Distribuida

A continuación detallaré cada una de ellas.

Programación Secuencial

- También conocida como *estructura secuencial*, es aquella en la que una instrucción o acción sigue a otra en secuencia.
- Es más simple y fácil de usar.
  - Como las instrucciones están relacionadas, será más sencillo entender lo que hace cada función en una instrucción.
  - Las tareas se llevan a cabo de tal manera que la salida de una es la entrada de la siguiente y así sucesivamente hasta finalizar un proceso
- Es decir, es aquella en la que
  - un programa que hace que se ejecute una instrucción, y
  - *no puede ejecutarse ninguna otra instrucción hasta que ésta haya finalizado.*

Programación Concurrente

- Los SSOO multitarea permiten, gracias al uso de una técnica denominada ***programación concurrente***, la posibilidad de tener "en ejecución" múltiples tareas.
- Dichas tareas se pueden ejecutar en:
  - una sola unidad central de proceso (Sistemas monoprocesador),
  - en varios procesadores o
  - en una red de computadores.
- pasando la técnica a llamarse, respectivamente:
  - Multiprogramación,
  - Programación Paralela y
  - Prog. Distribuida.

Programación Concurrente II

- Concurrencia. Definición general:
  - Acaecimiento o concursos de varios sucesos(procesos) en un mismo tiempo. (WordReference)
  - Es la simultaneidad en la ejecución de múltiples tareas interactivas. Estas tareas pueden ser un conjunto de procesos, o hilos de ejecución creados por un único programa. (Wikipedia)

Programación Concurrente. Técnicas

- Vamos a hacer nuestra primera aproximación a estas tres técnicas.
- MonoProcesador
  - Multiprogramación
    - 1 CPU → Muchos procesos.
- MultiProcesador
  - Varias CPU's →Muchos procesos
  - **Programación Paralela**
    - Un único ordenador con muchas CPU's / núcleos
    - Sincronización por reloj
    - Comparten memoria
  - **Programación Distribuida**
    - Mas de un ordenador conectado en red
      - Cada uno puede tener uno o mas de un procesador
    - No comparten memoria ni reloj → Sincronización por Paso de mensajes

Programación Concurrente. Ventajas

- La programación concurrente que se emplea sobre los SSOO multitarea lleva asociado una serie de ventajas e inconvenientes, que detallaremos más adelante.
  - Evidentemente a mayor número de procesadores y/o máquinas normalmente estas ventajas se verán acrecentadas.
- Aumento de la velocidad de ejecución(= Mayor V de respuesta)
  - Si hago varios trabajos a la vez acabaré antes que si voy uno a uno.
- Utilización de la CPU más eficiente
  - Por ejemplo, la CPU que no quedará libre cuando un proceso requiera una operación de E/S.
- Procesar grandes volúmenes de información de manera efectiva
  - Ej: Divide y vence

Programación Concurrente.
Inconvenientes

- La desventaja o problema principal de esta técnica proviene de la *gestión de recursos compartidos*.
  - Hay compartición de recursos que impiden actuar de forma simultánea o paralela.
  - De hecho, en algunos casos ni intercalando, como en el caso de la impresión.

Programación Concurrente.
Inconvenientes

- Veamos un ejemplo explicativo:

  Preparación de una receta

  - Verter ingrediente 1 y 2 en un bol y batir. Dejar reposar en 2 minutos
  - Verter ingrediente 3 y 4 en un cazo. Calentar 2 minutos
  - Verter la mezcla del bol 1 con el 2
  - Dejar enfriar y servir

¿Cuántos cocineros necesito y cómo se puede preparar?

Programación Concurrente.
Inconvenientes

- Posibles formas de llevarla a cabo:

  a) Un cocinero realiza un paso tras otro, de tal forma que el siguiente no se lleva a cabo hasta completar el anterior.

  b). Uno o mas cocineros realizan un paso tras otro, pero, si es posible, empieza el siguiente paso aunque no se haya acabado al anterior.
  - Por ejemplo verter ingredientes 3 y 4 mientras reposan el 1 y el 2
  - Así, habría 3 bloques de instrucciones (1/2, 3/4, 5/6), cada una en un proceso que se podría 'cocinar' concurrentemente..

  c). Cada conjunto de pasos es realizado simultáneamente por cocineros diferentes.
  - Uno por ejemplo puede dedicarse a los ingredientes 1 y 2, y otro a los 3 y 4
  - Nota: no se puede hacer la mezcla final (paso 5) hasta que no estén completos los primeros 4 pasos, incluso aunque tuviera 100 cocineros.

Programación paralela

- **No confundir programación concurrente/paralela con computación paralela**
  - Una **computadora paralela** es una colección de procesadores, típicamente del mismo tipo, interconectados de alguna manera para permitir la coordinación de sus actividades y el intercambio de datos.
    - Hay varias formas diferentes de computación paralela: paralelismo a nivel de bit, paralelismo a nivel de instrucción, paralelismo de datos y paralelismo de tareas.
  - Un **algoritmo paralelo** es un algoritmo que puede ser ejecutado por partes en el mismo instante de tiempo por varias unidades de procesamiento, para finalmente unir todas las partes y obtener el resultado correcto.
    - La máxima aceleración posible de un programa como resultado de la paralelización se conoce como la ley de Amdahl.
  - Técnicamente es el procesador el que se encarga de ejecutar de forma paralela los procesos, pudiendo ayudar el programador dividiendo sus programas en "trocitos de ejecución" en forma de subprocesos o hilos.

Programación Distribuida

- Se define un **sistema distribuido** como aquel en el que los componentes de HW o SW se encuentran en dispositivos "alejados" y unidos a través de una red.
- Existen varios modelos de programación para la comunicación entre procesos de diferentes equipos en un sistema distribuido:
  - **Sockets**
    - El mas extendido
    - Se trabaja a nivel de puerto
  - **RPC** (Remote Procedure Call)
    - Permite a un proceso cliente llamar a un procedimiento de otro proceso (servidor).
  - **RMI** (Remote Method Invocation)
    - Es como el RPC pero para objetos
    - Permite que objetos de diferentes procesos se comuniquen: Un objeto de un proceso puede invocar métodos de otro objeto de otro proceso en ejecución
  - **CORBA** (Common Object Request Broker Architecture)
    - Es un estándar definido por la Object Management Group (OMG)
    - Permite que diversos componentes de software, escritos en múltiples lenguajes de programación, corran en diferentes computadoras, puedan trabajar juntos; es decir, facilita el desarrollo de aplicaciones distribuidas en entornos heterogéneos.
    - Genera esqueletos a partir del lenguaje IDL

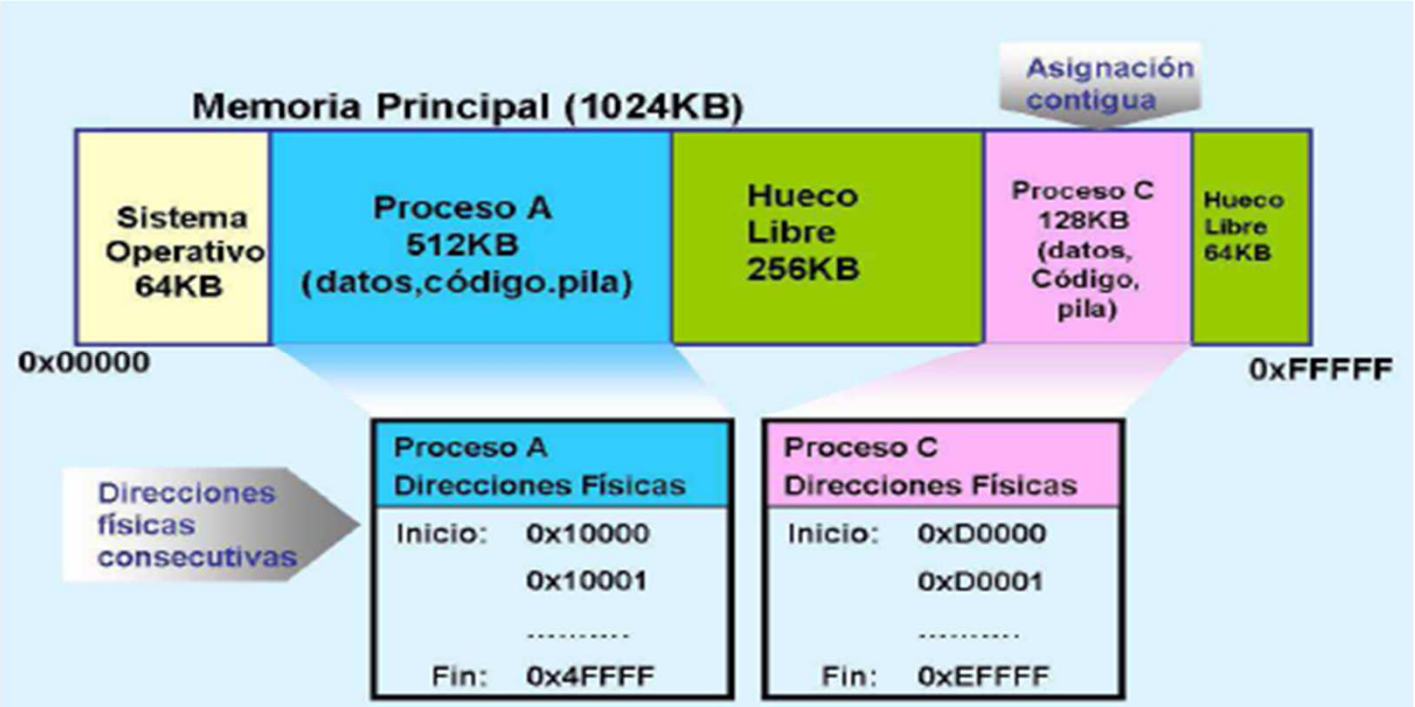
Procesos. Conceptos

- Los libros de texto usan los términos proceso y tarea para referirse normalmente a lo mismo.
- Para el SSOO un proceso no se refiere únicamente al código y a los datos que usa nuestro programa.
  También incluye todo lo necesario para poder gestionar su ejecución.

- Básicamente consiste en:
  - **Código ejecutable del programa**
  - **Memoria de trabajo**
    - Los datos (globales o compartidos por las funciones)
    - El montón o heap, de donde se coge la memoria dinámica.
  - **Pila**
    - Para variables locales y gestión de funciones
  - **Estado de ejecución**
    - Valor de los registros del procesador en un momento dado para dicho programa.
    - Contador de programa → Dirección de la siguiente instrucción a ejecutar
    - Permisos, límites en memoria, ficheros abiertos, etc (depende del SSOO).

Procesos. Características

- Para que un proceso pueda empezar su ejecución, debe residir completamente en memoria, y tener asignados todos los recursos que necesite en ese momento.
- Cada proceso está protegido del resto de procesos.
  - Ningún otro proceso podrá escribir en las zonas de memoria que pertenezcan a ese proceso (esto es así en todos los SSOO modernos).



Procesos. Características II

- El proceso es el elemento central de la ejecución y el que realiza los trabajos del usuario a través del uso de los recursos del sistema.
- Para que un proceso de usuario acceda a un recurso deberá solicitarlo al sistema, el cual, le cederá o no el recurso al proceso.
- **Dos o más procesos pueden asociarse al mismo programa.**
  - con diferentes datos (es decir, con distintas imágenes de memoria)
  - y en distintos momentos de su ejecución (con diferentes contadores de programa).
  - En ese caso siempre son entidades independientes, aunque ejecuten el mismo programa.
  - Ejemplo:
    - Se pueden tener, por ejemplo, dos instancias del programa NotePad o chrome ejecutándose a la vez, visualizando cada una un fichero/página diferente.
  - Para que los datos de uno no interfieran con los del otro, cada proceso se ejecuta en su propio espacio de direcciones en memoria.

Imagen de un Proceso

- *Para el sistema operativo un proceso es un conjunto de estructuras de datos* a gestionar, llamado "**imagen del proceso**":
  - Es decir, que la imagen del proceso es **toda la información que almacena el SSOO sobre un proceso en memoria.**

- Está formada por **4 trozos** distribuidos normalmente en 2 bloques:
  - Código / datos / pila
    - Parte principal del proceso (generada a partir del ejecutable)
  - PCB *(Bloque de control de Proceso)*
    - Contiene toda la información necesaria para que el SSOO pueda manejar al proceso, principalmente los valores de los registros de la CPU.

- Nota: **A cada trozo se le llama segmento**, y en algunos SSOO **pueden estar separados en memoria.**

Imagen de un Proceso

- Los segmentos se pueden almacenar de forma junta o disjunta en la memoria.
- Lo que si se almacena seguro por separado son:
  - Las instrucciones, datos y pila se almacenan en la memoria de procesos de usuario.
  - El PCB se almacena en la memoria destinada al S.O.

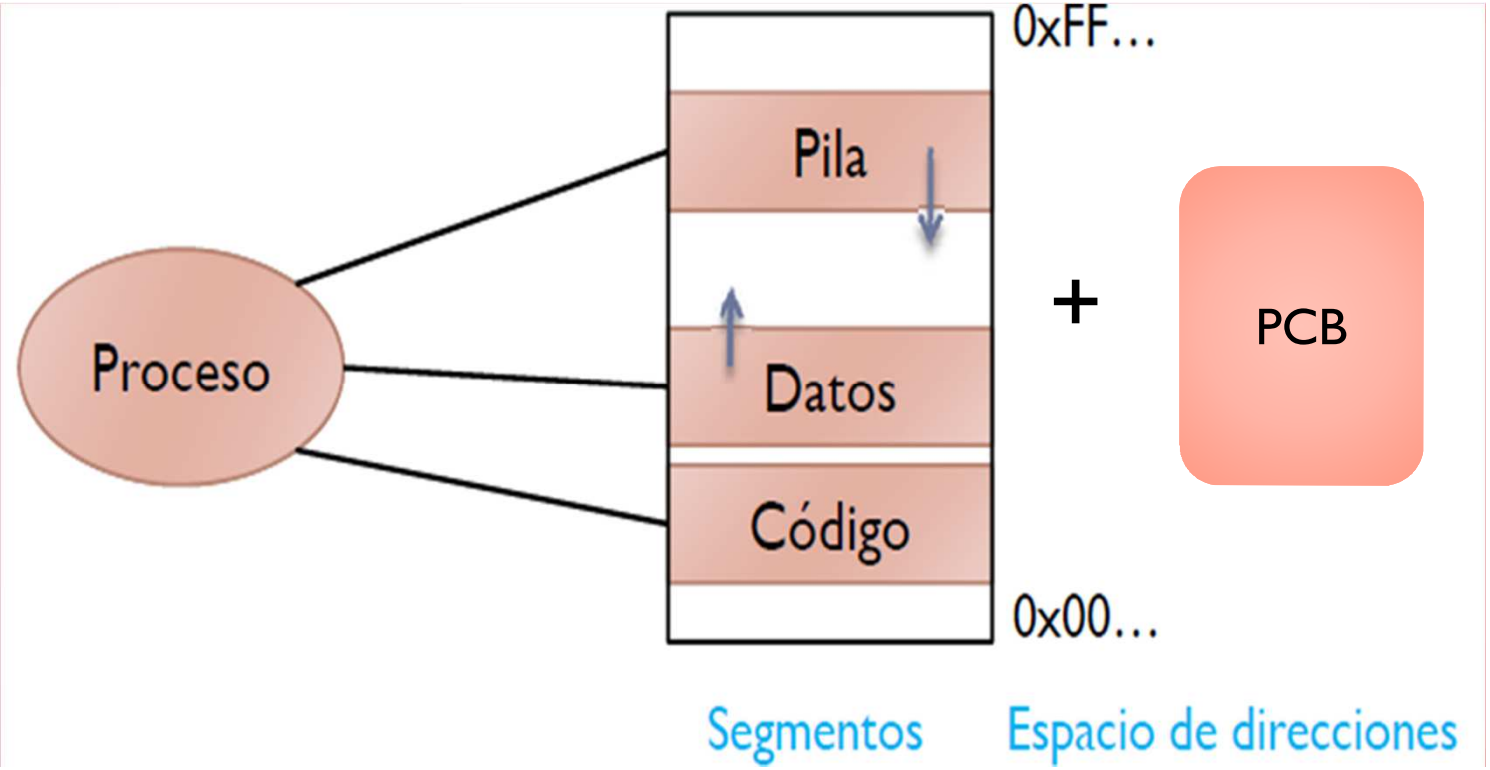
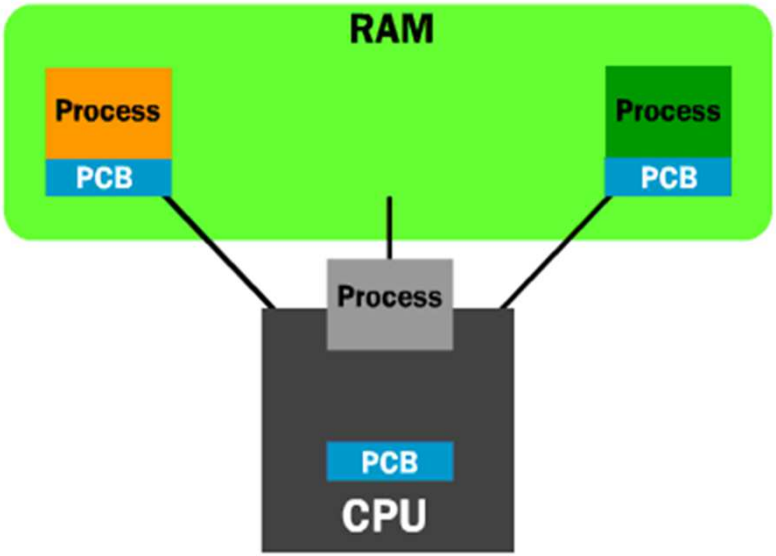


Imagen de un Proceso

- Esta forma de trabajar ofrece ventajas para la multiprogramación:
  - La parte que se intercambia y se lleva a la CPU es el PCB.
  - El resto permanece "siempre" en memoria, por lo que es fácil para el programa en ejecución relacionarse con sus diferentes partes:



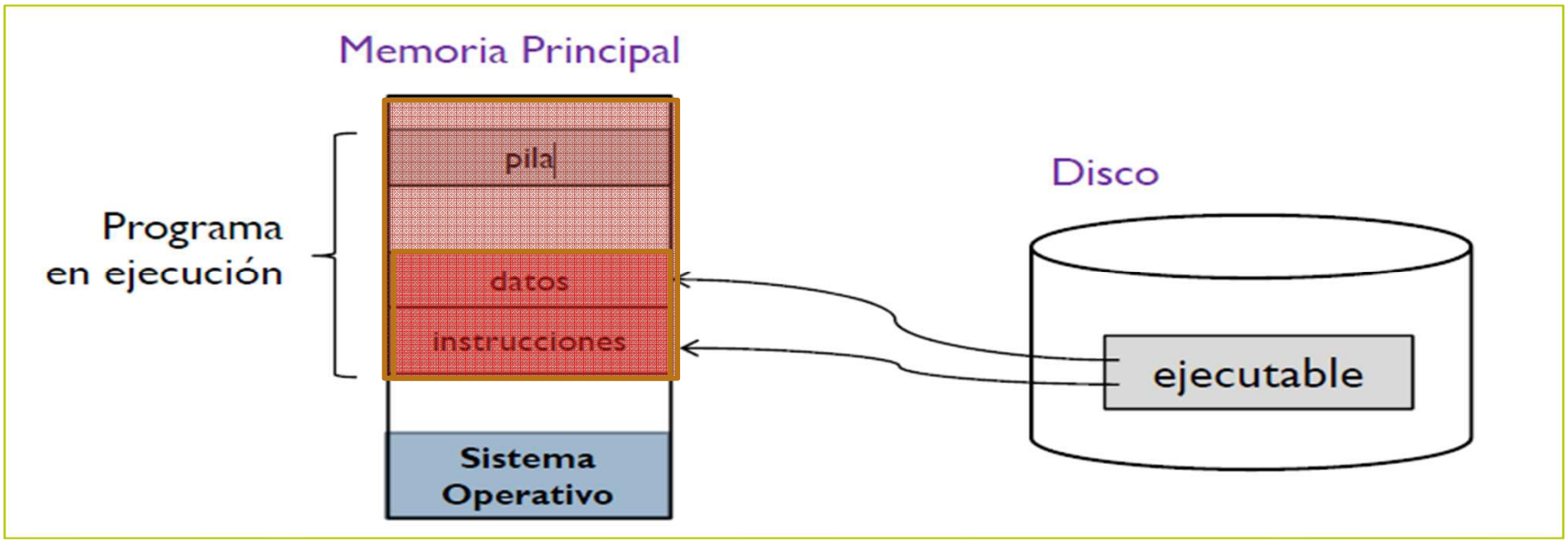


Imagen de un Proceso en Unix

	Contexto del Usuario
Texto del Proceso	Instrucciones de máquina ejecutables del programa.
Datos del proceso	Datos del proceso accesibles por el programa.
Pila del usuario	Contiene los argumentos, las variables locales, y los punteros de las funciones que ejecutan en modo usuario.
Memoria compartida	Memoria compartida con otros procesos, utilizada para la comunicación interproceso.
	Contexto de los Registros
Contador de programa	Dirección de la próxima instrucción a ejecutar; puede estar en el espacio de memoria del núcleo o del usuario, de este proceso.
Registro de estado del procesador	Contiene el estado del hardware en el momento de la expulsión; el formado y el contenido dependen del hardware.
Puntero de pila	Señala a la cima de la pila del núcleo o de la pila del usuario, dependiendo del modo de operación en el momento de la expulsión.
Registros de propósito general	Dependientes del hardware.
	Contexto del Sistema
Entrada de la tabla de procesos	Define el estado de un proceso; esta información está siempre accesible para el sistema operativo.
Area U (de usuario)	La información de control del proceso a la que se necesita acceder sólo en el contexto del proceso.
Tabla de regiones de preproceso	Define una traducción de direcciones virtuales a físicas; también contiene un campo de permiso que indica el tipo de acceso permitido para el proceso: sólo lectura, lectura/escritura o lectura/ejecución.
Pila del núcleo	Contiene los marcos de pila de los procedimientos del núcleo cuando el proceso ejecuta en modo del núcleo.

Bloque de Control de Proceso (PCB)

- El Bloque de control del proceso o BCP o en inglés PCB (Process Control Block) es un registro especial donde el sistema operativo agrupa toda la información que necesita conocer respecto a un proceso particular.
- Cada vez que se crea un proceso, el sistema operativo crea el PCB correspondiente para que sirva como descripción en tiempo de ejecución durante toda la vida del proceso.
- Cuando el proceso termina, su BCP es borrado y el registro puede ser utilizado para otros procesos.
- Características
  - Es la estructura de datos central del SSOO
  - Contiene toda la información que el SSOO necesita saber sobre los procesos
  - Los leen y modifican todos los módulos o trozos de código del S.O.
  - Se debe proteger para evitar errores de utilización
    - Para ello se crea un módulo del S.O. que contiene a todos los PCB's
    - Sólo se permite acceder a ellos a través de funciones de este módulo
  - Es una estructura de datos con campos para registrar los diferentes aspectos de la ejecución del proceso y de la utilización de recursos.

Bloque de Control de Proceso (PCB)

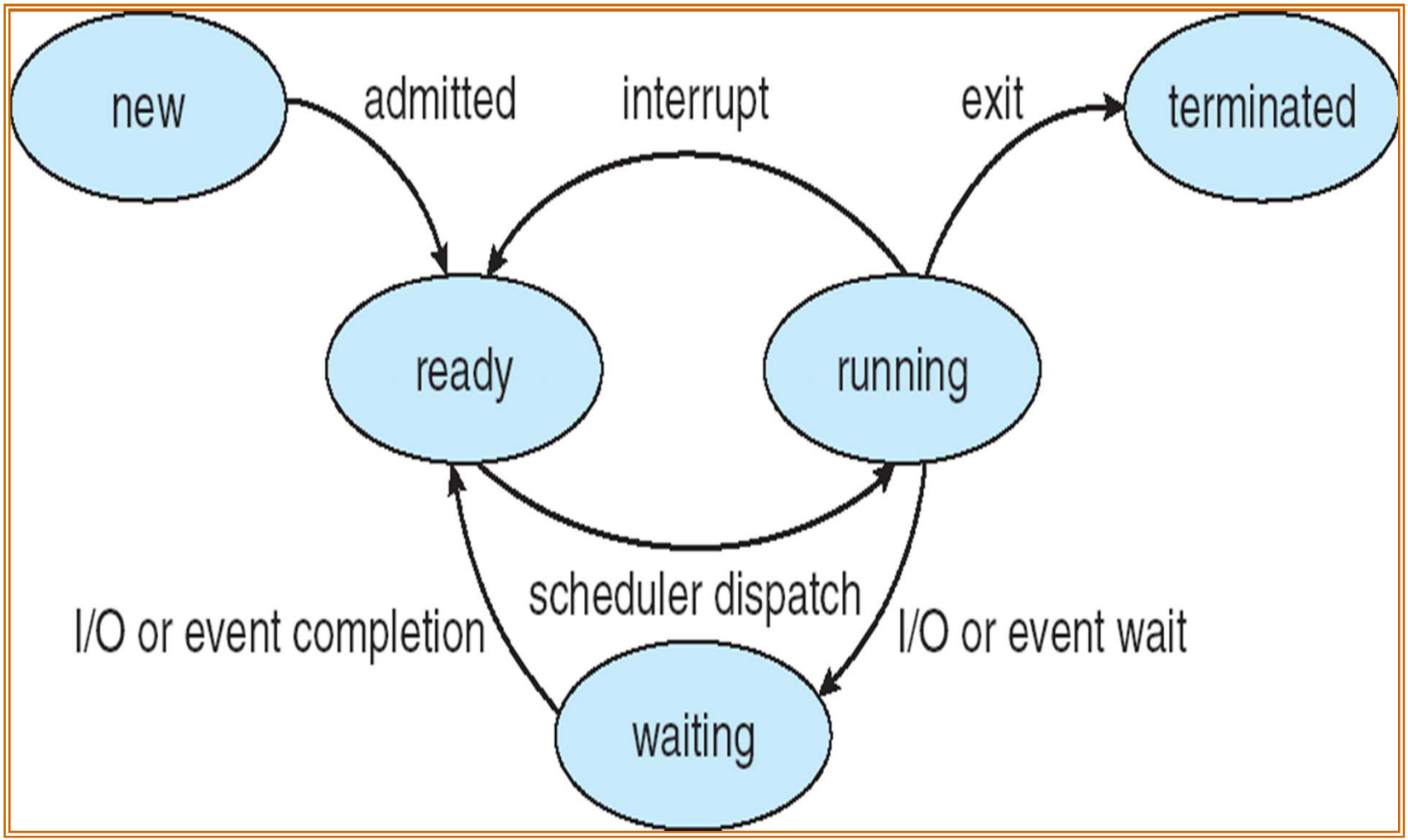
- La información almacenada en un BCP incluye típicamente algunos o todos los campos siguientes:
  - **Estado del proceso**.
    - Por ej. listo, en espera, bloqueado.
  - Número o **Identificador del proceso** (Process Identificator -**PID**-).
  - **Contador de Programa:**
    - Dirección de la próxima instrucción a ejecutar.
  - **Registros.**
    - Valores de registro de CPU. Se utilizan también en el cambio de contexto.
  - **Espacio de direcciones de memoria** (límites de memoria).
  - **Lista de recursos asignados** (incluyendo descriptores de archivos y sockets abiertos).
  - Etc:
    - Prioridad en caso de utilizarse dicho algoritmo para planificación de CPU.
    - Estadísticas del proceso.
    - Datos del propietario (owner).
    - Permisos asignados.
    - Signals pendientes de ser servidos. (Almacenados en un mapa de bits)

- Esta lista es simplemente indicativa, ya que cada sistema operativo tiene su propio diseño de PCB

Estados de un Proceso

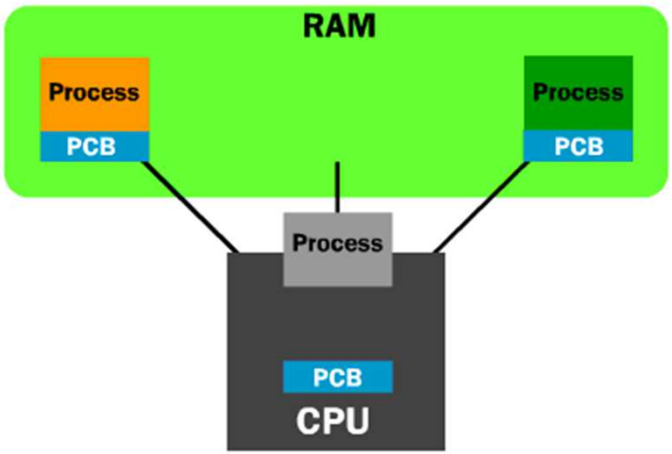
- Un estado describe la situación actual de un proceso
- Conforme se ejecuta un proceso cambia su estado
  - nuevo: El proceso se está creando
  - en ejecución: Se están ejecutando sus instrucciones
  - en espera: Está esperando que ocurra algún evento (ej. E/S)
  - listo: Está esperando que le asignen la CPU
  - terminado: Ha terminado su ejecución

Estados de un Proceso

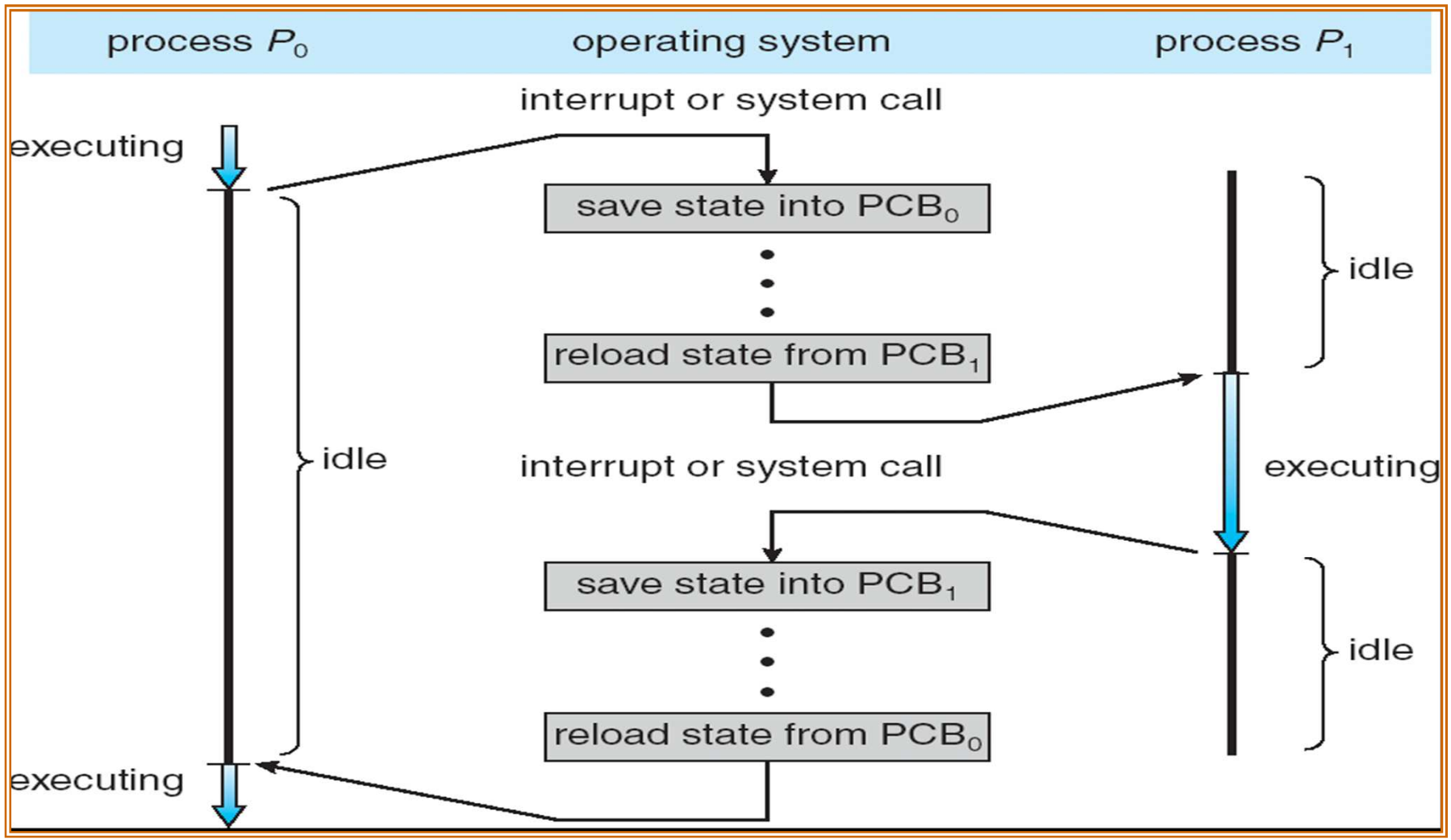


Cambio de Contexto

- Cuando se cambia el proceso que posee la CPU, el sistema debe salvar el estado del viejo proceso y cargar el estado salvado del nuevo proceso
- El contexto se guarda para luego poder recuperarlo
- Se guarda en el PCB
- El tiempo que dura el cambio de contexto es un gasto extra para el sistema, ya que no hace nada útil durante la conmutación
- El tiempo requerido para la conmutación depende del soporte del procesador



Conmutación de Procesos



Idle= tiempo ocioso o de inactividad en el que el proceso no hace nada

Planificación de procesos

- **Definición y Conceptos Básicos**
- El término planificación de procesos hace referencia a un conjunto de políticas y mecanismos del SO que gobiernan el orden en que se ejecutan los procesos (Milenković)

- *El planificador de procesos es el módulo del SO* que se encarga de mover los procesos entre las distintas colas de planificación, en base a un algoritmo, determinando por tanto los procesos que se ejecutarán y los que esperarán.

- *La ejecución de un proceso consiste en una alternancia entre ráfagas de CPU y ráfagas de E/S*
  - Un *proceso limitado por E/S* (I/O bound) es aquél que pasa más tiempo haciendo E/S que usando la CPU (tiene ráfagas de CPU cortas)
  - Un *proceso limitado por CPU* (CPU bound) es aquél que pasa más tiempo computando que haciendo E/S (tiene ráfagas de CPU largas)

Planificación de procesos.

El Planificador

- *Se encarga de escoger un proceso de entre los que están en memoria listos para ejecutarse* , asignando la CPU al proceso elegido
- La decisión de planificación puede ocurrir:
  - 1. Cuando un proceso pasa de ejecución a espera
  - 2. Cuando un proceso pasa de ejecución a listo
  - 3. Cuando un proceso pasa de espera a listo
  - 4. Cuando un proceso termina
- Un planificador es no expropiativo (nonpreemptive) cuando sólo planifica en los casos 1 y 4
- En otro caso decimos que el planificador es expropiativo (preemptive)

Planificación de procesos
El Dispatcher (despachador)

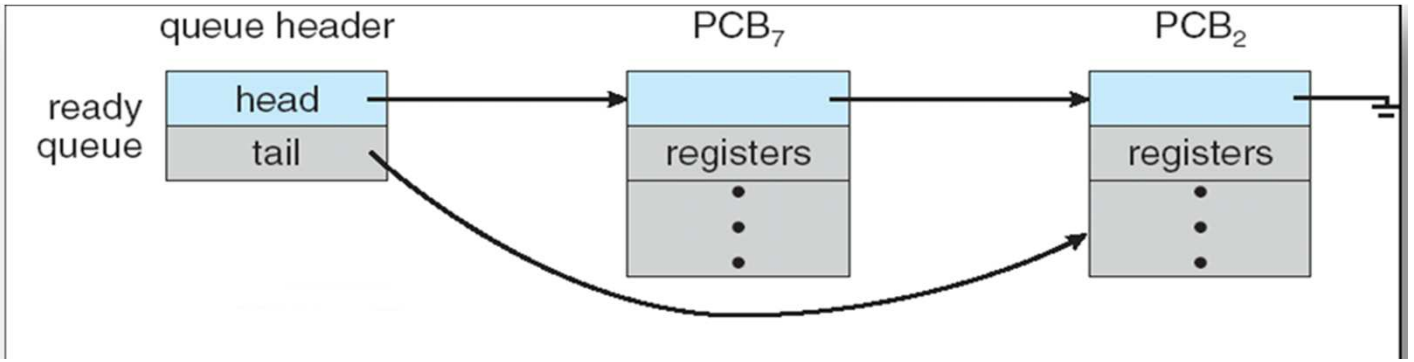
- El despachador es *el módulo del planificador que cede la CPU al proceso elegido* para ejecutarse. Para ello el despachador tiene que:
  - Realizar una conmutación de contexto
  - Cambiar la máquina a modo usuario (no privilegiado)
  - Saltar al punto apropiado del programa para continuar con su ejecución
- El tiempo que tarda el despachador en detener un proceso y poner otro en ejecución se denomina latencia del despachador.
  Debe ser lo más pequeña posible

Resumen práctico

- El planificador coge los procesos y los pone en la cola adecuada en base a un algoritmo de planificación de procesos.
- El despachador los coge de ellas y los lleva a ejecución, haciendo la operación contraria cuando sea necesario.
- Es importante la latencia del despachador para conseguir un buen tiempo de respuesta.

Colas de Planificación de Procesos

- Los procesos se encuentran en colas y se mueven entre ellas
- Cola de trabajos: conjunto de todos los procesos en el sistema
- Cola de procesos listos: conjunto de procesos que se encuentran en memoria principal, listos y esperando ejecutarse
- Colas de dispositivo: conjunto de procesos esperando un dispositivo de E/S
- Cada cola (lista enlazada) tienen 2 apuntadores, uno al nodo que contiene el PCB del primer proceso, y otro al último:



Gestión de Procesos con Java

- Java permite crear procesos / invocar órdenes sobre el sistema operativo e interactuar con ellos a través de la información que estos generan tras y durante su ejecución.
  - Ejs: lanzar el navegador, la calculadora, el acrobat reader para ver un PDF, etc.
- No es recomendable su uso salvo para casos concretos y necesarios, puesto que se pierde el objetivo básico de java, que es poder ejecutar el programa en cualquier plataforma.
- Evolución:
  - Con la primera versión (JDK1.0)
    - Se incluyen las clases base **Runtime** y **Process**, dentro del paquete java.lang
  - En el JDK1.5
    - Se añade la clase **ProcessBuilder**
  - En el JDK1.7
    - Aparecen **métodos** para redirigir las salidas generadas por los procesos: redirectOutput() y redirectError()

Gestión de Procesos con Java. La clase Runtime

- Encapsula al proceso del intérprete Java que esté en ejecución.
- No se puede crear una instancia Runtime:
  - Hay que obtener una referencia al objeto Runtime que se está ejecutando actualmente: **Runtime.getRuntime()**
- Solo es válido su uso en programas seguros:
  - Los applets y otros fragmentos de código de los que se desconfíe habitualmente no pueden llamar a ninguno de los métodos de la clase Runtime sin activar una SecurityException.
- Su función principal (al menos para nosotros) es la capacidad de ejecutar programas mediante el método exec, el cual está sobrecargado en varias formas, siendo esta la más completa:
  - **Process Runtime.exec(String[] comando, String[] envp, File dir)**
    - Ejecuta el comando especificado y sus argumentos (comando) en un proceso hijo independiente, con el entorno envp y el directorio de trabajo por defecto dir.
    - String[] comando es para un comando y sus argumentos→ new String[]{"cmd", "/c", "dir", "/s"}
    - OJO!!! El método exec() no actúa como intérprete de comandos; Simplemente ejecuta un programa.
      - Es decir no pongas cosas como …exec (java ejemplo1 'Hola!' >> salida.txt) porque no funcionará.
      - Cmd /c → Ejecuta la orden y cierra el intérprete de comandos
  - Nota: Se puede usar cmd, powershell o cualquier intérprete de comandos que reconozca el SSOO destino.

Gestión de Procesos con Java.
La clase Runtime II

- Además de para gestionar procesos (objetivo del tema), Runtime se puede usar para:
  - Conocer la cantidad de objetos y el espacio libre que hay memoria.
    - ***totalMemory***(): memoria total disponible para la Máquina Virtual Java
    - ***freeMemory***(): cantidad de memoria libre disponible en este momento.
  - Obtener la cantidad de procesadores disponibles: ***availableProcessors***()
  - Invocar al recolector de basura: ***gc()***
  - Detener la MVJ de forma normal (**exit**()) o forzosa (**halt**())
  - Cargar DLLs (**loadLibrary(String libname)**) o
  - ficheros como librerías dinámicas (**load(String filename)**)
  - Etc.

Gestión de Procesos con Java

- La Clase Process
- public abstract class Process extends Object
  - Clase para manejar un proceso del sistema dentro de nuestro programa Java.

- Métodos:
  - destroy()
    - Destruye el proceso
  - exitValue()
    - Valor que devuelve el proceso una vez finalizado
  - getInputStream()
    - Para leer de nuestro extremo de la tubería.
    - Todo lo que genere el proceso lanzado le llegará a nuestro proceso Java como entrada a través de este flujo.
    - x ej coger la salida de la orden dir
  - getOutputStream()
    - Para escribir en nuestro extremo de la tubería.
    - Todo lo que enviemos por este flujo le llegará al proceso (como entrada de datos).
    - x ej enviar datos a la orden More
  - getErrorStream()
    - Para leer en nuestro programa los posibles errores que genere el proceso.
  - waitFor()
    - Hace que el thread actual (nuestro proceso Java, el que ejecuta la linea p.waitFor()) espere, si es necesario, hasta que el proceso p haya terminado su ejecución.

- Nota: Existe una tubería conectada entre el proceso creado con Runtime y nuestro programa Java (se crea en el constructor).

Gestión de Procesos con Java

- Ejemplo básico de funcionamiento

```
try
{
   Process p = Runtime.getRuntime().exec ("directorio/ejecutable");
    /*donde directorio/ejecutable es el path del ejecutable y un nombre */
}
catch (Exception e)
{
  /*Se lanza una excepción si no se encuentra en ejecutable o el fichero no es ejecutable*/
}
```

- Nota:
  - En windows, heredados de MS-DOS, existen ciertos comandos que no poseen un ejecutable propio, como por ejemplo los comandos dir, cls o date.
  - Estos comandos que no tienen ejecutable, llamados internos, no se puede arrancar directamente desde java.
  - Hay que hacerlo de forma "indirecta", a través del intérprete de comandos de Windows (CMD) y decirle cual es el comando que se tiene que ejecutar.
  - Ejemplo:

Process p = Runtime.getRuntime().exec ("cmd /c dir");

  - **Cmd /c** → Ejecuta el comando y finaliza (cierra el intérprete de comandos)
  - **Cmd /k** →Ejecuta el comando y sigue activo

  - Nota: esto no pasa en linux ya que no hay comandos internos, (todos los comandos tienen ejecutable asociado).

Gestión de Procesos con Java

Leer la salida del proceso

- Al coger un proceso/ejecutar un programa desde java, la salida del proceso se redirige a nuestro programa java, en vez de a la pantalla.
- Para leer lo que nos devuelve el proceso usaremos el flujo que nos devuelve el método getInputStream() o el getErrorStream() del "Process" devuelto por Runtime.exec()..,
- Puesto que un InputStream es algo incómodo para leer cadenas de texto, lo metemos dentro de un BufferedReader para hacer el proceso algo más cómodo.

- Ejemplo:

```
/* Se obtiene el stream de salida del cmd y entrada en mi programa */
InputStream is = p.getInputStream();

/* Se prepara un BufferedReader para poder leer la salida más comodamente. */
BufferedReader br = new BufferedReader (new InputStreamReader (is));

//Se gestionan los datos recibidos
String aux = br.readLine();
while (aux!=null) // Mientras se haya leido alguna linea
{
     // Se hace lo que sea, por ejemplo, escribir la linea leída sobre la pantalla
     System.out.println (aux);
    // y se lee la siguiente.
     aux = br.readLine();
}
```

Gestión de Procesos con Java. ProcessBuilder

- La versión 1.5 del JDK añade una nueva forma para crear y ejecutar procesos del SSOO: **java.lang.ProcessBuilder**, que en la 1.8 se ve potenciada.

- Con esta nueva clase:
  - Cada instancia de ProcessBuilder gestiona una colección de atributos de proceso.
  - Aparece el método start(), que creará una nueva instancia del proceso con estos atributos.
    - Podemos llamar a start() repetidamente desde la misma instancia para crear nuevos subprocesos con atributos idénticos o parecidos.
    - Esto permite demorar la ejecución, así como ejecutar el comando varias veces.
    - Nota: Eso si, siempre se lanza un proceso independiente por cada start()

Gestión de Procesos con Java. ProcessBuilder

- Cada processBuilder maneja los siguientes atributos de proceso:
- Un comando: (command)
  - una lista de cadenas que indican el programa externo que se invocará y sus argumentos.
- Un entorno (environment,)
  - (que es un mapeo dependiente del sistema de variables, a valores).
  - El valor inicial es una copia del entorno del proceso actual. (System.getenv()).
  - *Nota: En la versión 1.8 aparecen nuevos comandos para gestionarlo más fácilmente como el compute, que permite evaluar variables o Replace, que permite cambiar el valor de una variable de entorno.*
- Un directorio de trabajo.
  - Nota: Si no se indica, el valor predeterminado es el directorio de trabajo actual del proceso actual, generalmente el directorio indicado en la propiedad del sistema user.dir.
- Mas todos los flujos, incluidos los nuevos de redirección a ficheros.

Gestión de Procesos con Java. ProcessBuilder

- Ejemplo: Ejecuta un archivo de música desde Java

```
try{
    String comando = new String("c:\\musica\\salsa.mp3");
    ProcessBuilder p=
                new ProcessBuilder("cmd.exe","/c","start",comando);
    p.start();
}catch(Exception e) {}
```

- Nota: Observa la diferencia del paso de parámetros entre Runtime. .getRuntime().exec y processBuilder:
  - *Runtime.getRuntime().exec(...) takes a single string and passes it directly to a shell or cmd.exe process.*
  - *The ProcessBuilder constructors, on the other hand, take a varargs array of strings or a List of strings, where each string in the array or list is assumed to be an individual argument.*

Gestión de Procesos con Java. ProcessBuilder

- Así, x ej, en Windows:

  Runtime.getRuntime().exec("C:\DoStuff.exe -arg1 -arg2");

  - Que pasará C:\DoStuff.exe -arg1 -arg2 al cmd.exe, el cual ejecutará el prog con los 2 argumentos.

- OJO!

  ProcessBuilder b = new ProcessBuilder("C:\DoStuff.exe -arg1 -arg2");

  Fallaría.

  - En su lugar usaríamos (En 3 parámetros):

    ProcessBuilder b = new ProcessBuilder("C:\DoStuff.exe", "-arg1", "-arg2");

Gestión de Procesos con Java

- Ejemplo: Acceso a la lista de Procesos
- Muchas veces es necesario acceder al listado de los procesos en ejecución del SSOO
  - para controlar si una aplicación está corriendo o no.
  - para evitar que haya más de una instancia del mismo proceso corriendo en el servidor.

Gestión de Procesos con Java

- Para entenderlo vamos a crear un programa que devuelva toda la información disponible sobre un proceso dado SI se encuentra dentro de la lista de procesos del sistema (Windows)

```
try{
   Process proc = Runtime.getRuntime().exec("wmic.exe");
   BufferedReader input = new BufferedReader(
                              new    InputStreamReader(proc.getInputStream()));
   OutputStreamWriter oStream = new OutputStreamWriter(proc.getOutputStream());

   oStream .write("process where name= 'nuestroProceso.exe' ");
   while ((line = input.readLine()) != null)
   {
   if (line.contains("nuestroProceso.exe"))
        return true;
   }
   input.close();
} catch (Exception ex) {
   // handle error
}
```

Gestión de Procesos con Java

- Ejemplo 2: Acceso a la lista de Procesos en Windows con WMIC:

  (Inicio/Ejecutar/wmic; Process)

- Ejemplo 3: Buscar un proceso (chrome.exe) dentro de la lista de procesos

```
wmic:root\cli>process where name=chrome.exe
Nodo: PC-JAVIER1
Error:
Descripción = Consulta no válida


wmic:root\cli>process where name='chrome.exe'
Caption     CommandLine
nPagedPoolUsage  QuotaPeakPagedPoolUsage  ReadOperationCount  ReadTransferCount
chrome.exe  "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" http:/
                483                       12345               20878777
chrome.exe  "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --type
                406                       18381               776192
chrome.exe  "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --type
                280                       22456               991079
chrome.exe  "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --type
                288                       6932                482553

wmic:root\cli>_
```

OJO!!! **las comillas son obligatorias al dar el nombre del proceso**

- Notas:
  - **wmic process** saca todos los procesos que actualmente están en la memoria del sistema.
  - Mira el anexo llamado WMIC.pdf si quieres aprender más sobre el tema.

Gestión de Procesos con Java

- Ejemplo: Acceso a la lista de Procesos
- Linux:

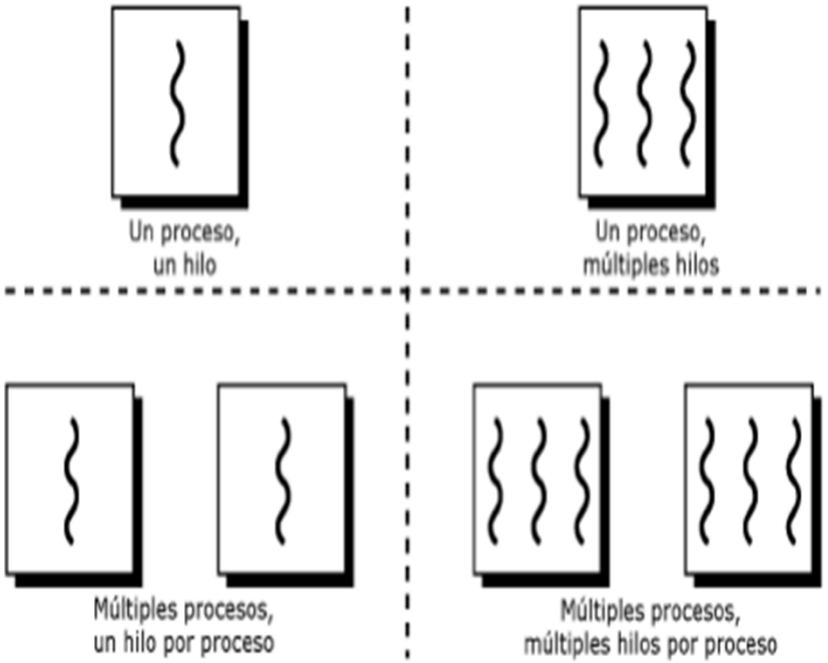
```
try{
Process p = Runtime.getRuntime().exec(new String[] { "bash", "-c", "ps aux | grep
nuestroProceso" });
BufferedReader input = new BufferedReader(new InputStreamReader(p.getInputStream()));

while ((line = input.readLine()) != null)
{
      if (line.contains("nuestroProceso "))
      {
      // el processo que estas buscando esta corriendo
      }
}
catch (Exception e)
{
   // handle error
}
```

- En este caso basta con lanzar el comando "ps" y gestionar su salida.
- Nota: ps aux me saca una lista detallada con todos los procesos del sistema, siendo grep la orden que extrae la linea con nuestro proceso (si la hay).

Hilos o hebras. Definición

- Un hilo es una unidad básica de utilización de la CPU consistente en un juego de registros y un espacio de pila.
- Comparte el código, los datos y los recursos con sus peers(pares) o hilos hermanos (hilos lanzados dentro de un mismo proceso pesado).
- A partir de ahora distinguiremos entre:
  - Hilos o procesos ligeros y
  - Tareas o procesos pesados
- El término multihilo aparece ahora
  - para hacer referencia a la capacidad
  - de un SO/lenguaje de programación
  - para mantener varios hilos de
  - ejecución dentro del mismo proceso.

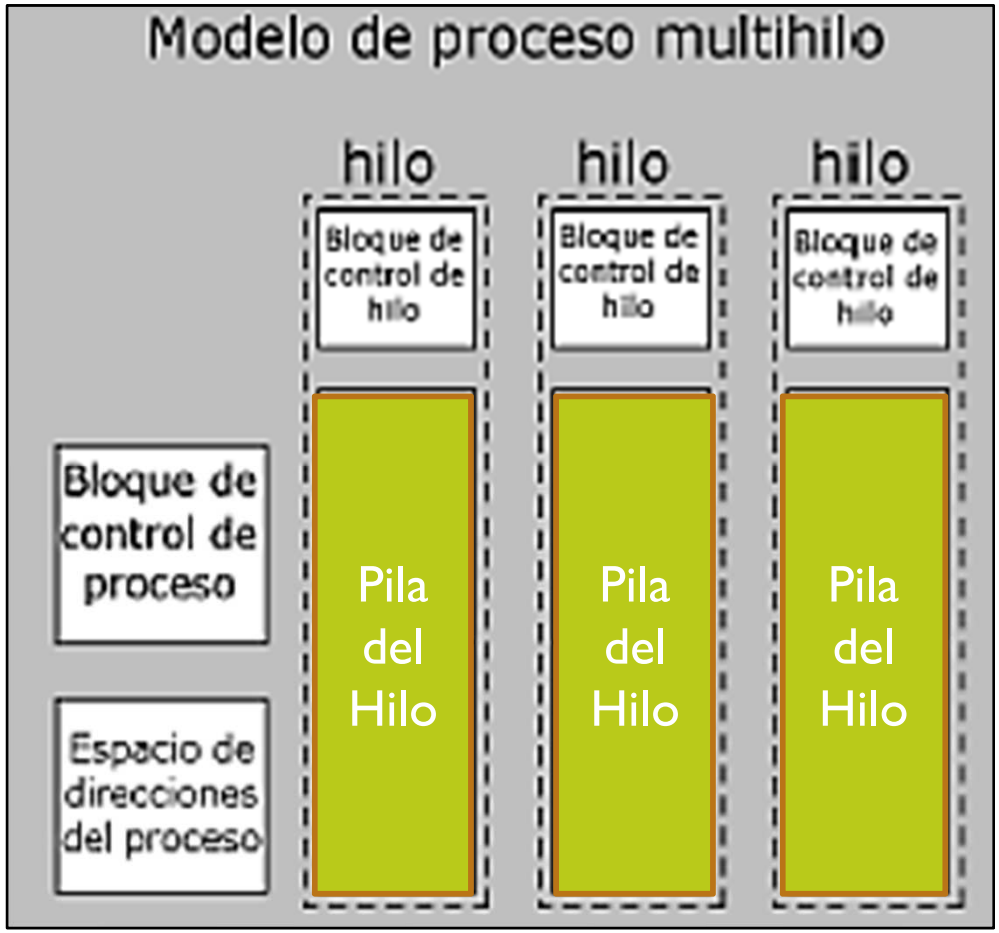


- Una tarea (o proceso pesado) está formada ahora por una o más hebras
- Una hebra sólo puede pertenecer a una tarea

Hilos o hebras. Definición

- Por lo tanto un hilo (proceso ligero) es una unidad básica de utilización de la CPU consiste en:
  - un contador de programa
  - un juego de registros
  - un espacio de pila.
- Y los procesos ahora contendrán uno o varios hilos que comparten la misma memoria y recursos.

  Así se pueden contener miles de hilos sin apenas coste adicional para el Sistema. Habría que intercambiar los PCB's de los hilos, que son mucho más pequeños(ligeros) que los PCB's de las tareas.

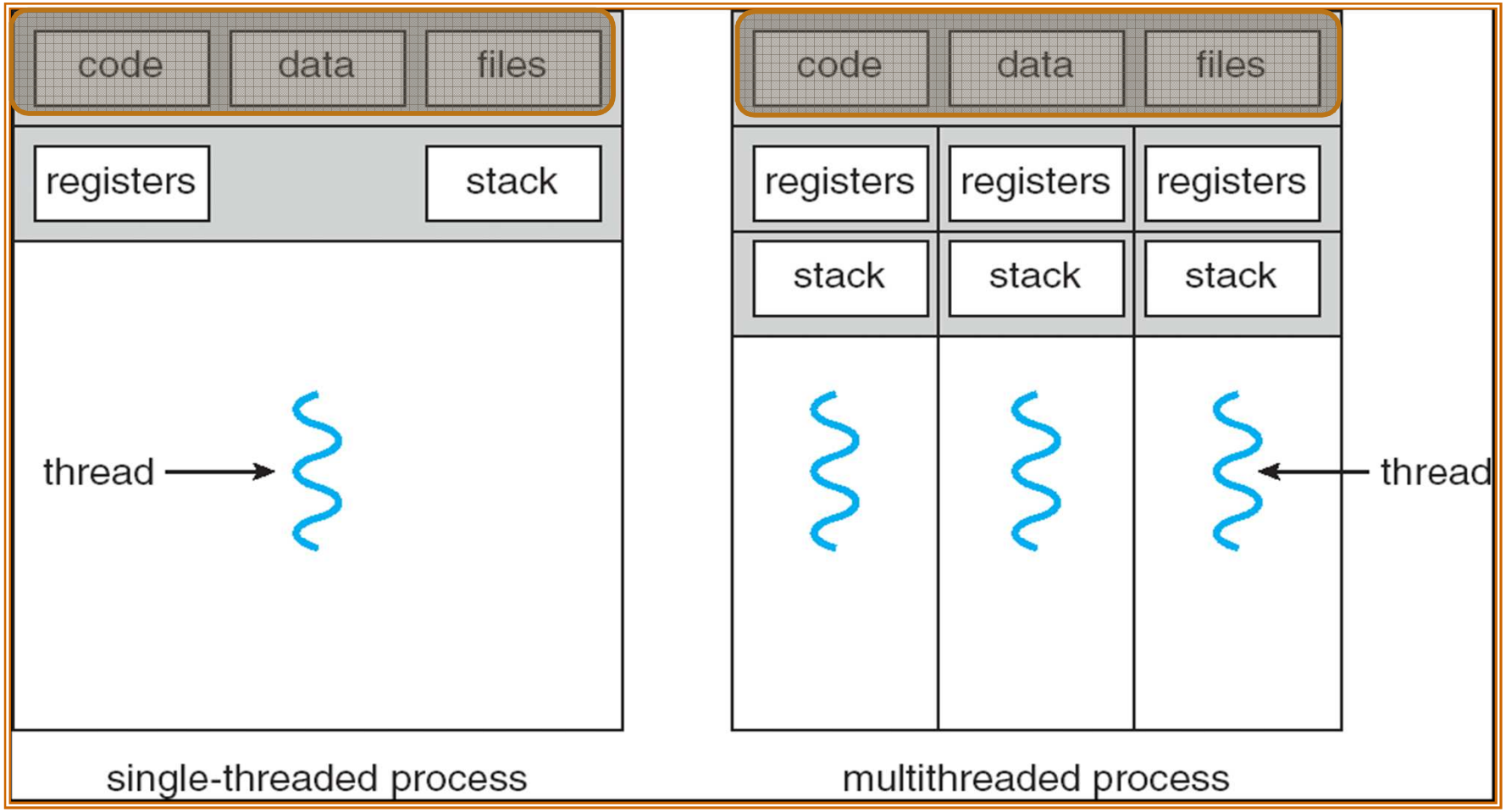


Hilos o hebras. Definición

- Recursos compartidos entre los hilos:
  - Código (instrucciones).
  - Variables globales.
  - Recursos (Ficheros y otros dispositivos abiertos).

- Recursos no compartidos entre los hilos (= cada hilo tiene su copia):
  - Contador del programa
    - cada hilo puede ejecutar una sección distinta de código.
  - Estado
    - distintos hilos pueden estar en ejecución, listos o bloqueados esperando un evento.
    - Registros de CPU.
  - Pila
    - para las variables locales de los procedimientos a las que se invoca después de crear un hilo.

Hilos o hebras. Definición



Hilos. Características

- Se comparten recursos. La compartición de la memoria permite a las hebras pares comunicarse sin usar ningún mecanismo de comunicación entre procesos del SO
- El cambio de contexto es más rápido gracias a que hay que almacenar menos información en el PCB
- No hay protección entre las hebras. Una hebra puede escribir en la pila de otra hebra del mismo proceso
- Solo hay que usarlas en el caso necesario, ya que entre usar un proceso con una hebra y varias siempre gana el "uno".

Procesos vs Hilos

- Programar con hilos, al ser más ligeros, presenta las siguientes ventajas:
  - Se tarda mucho menos tiempo en crear un nuevo hilo en un proceso existente que en crear un nuevo proceso.
  - Se tarda mucho menos tiempo en terminar un hilo que un proceso.
  - Se tarda mucho menos tiempo en conmutar entre hilos de un mismo proceso que entre procesos.
  - Los hilos hacen más rápida la comunicación que entre procesos:
    - al compartir memoria y recursos, se pueden comunicar entre sí sin invocar el núcleo del SO.
    - Es decir, no hay que usar flujos de comunicación que dependan del SO

¿Cuándo usarlos?

- Programas/trozos de código que ejecuten varias tareas a la vez
  - Los programas que realizan una variedad de actividades se pueden diseñar e implementar mediante hilos.
- Trabajo interactivo y en segundo plano:
  - En un programa de hoja de cálculo, un hilo podría estar leyendo la entrada del usuario y otro podría estar ejecutando las órdenes y actualizando la información.
- Procesamiento asíncrono:
  - Se podría implementar, con el fin de protegerse de cortes de energía, un hilo que se encargará de salvaguardar el buffer de un procesador de textos una vez por minuto.

Implementación de los hilos

- Hilos gestionados por el núcleo del SO
  - El SO es consciente de la existencia de hebras y controla su ejecución.
  - Ejemplos
    - Windows XP/7/8/10/…, Solaris /Linux, Mac OS X, Android, …
  - Características:
    - La conmutación de contexto entre hebras es más lenta
    - Si una hebra se bloquea las hebras pares pueden continuar
    - Todas las hebras reciben el mismo tiempo de CPU

- Hilos en nivel de Proceso → programador
  - Las gestión de los hilos es realizada por las bibliotecas del lenguaje/programa.
  - El SO no sabe nada de la existencia de las hebras
  - Ejemplo: Java → La gestión de hilos la hace la MVJ
  - Características:
    - Las hebras a nivel de usuario realizan el cambio de contexto más rápidamente
    - Si el proceso que las contiene es sacado de la CPU (por ejemplo por operación de E/S) ninguna hebra puede ejecutarse en su lugar.
    - Tiempo de CPU diferente para hilos de distintas tareas(procesos).