**Apuntes Sistemas Operativos Tema 0.1:**

1. Definición de sistema operativo:

. - ***Transparencias:***

“El SO es un programa que controla la ejecución de los programas de aplicación y actúa como interfaz entre las aplicaciones y el hardware.”

***. – Internet:***

Un sistema operativo es un conjunto de programas que permite manejar la memoria, disco, medios de almacenamiento de información y los diferentes periféricos o recursos de nuestra computadora, como son el teclado, el mouse, la impresora, la placa de red, entre otros.

Los periféricos utilizan un driver o controlador y son desarrollados por los fabricantes de cada equipo. Encontramos diferentes sistemas operativos como Windows, Linux, MAS OS, en sus diferentes versiones. También los teléfonos y tablets poseen un sistema operativo.

1. Que hace un sistema operativo:

***. – Transparencias:***

***Maquina abstracta:***

* Es responsabilidad del SO construir una máquina abstracta, que es:
  + Más fácil (esconde detalles del hardware).
  + Uniforme (independencia del dispositivo) de programar, que el hardware subyacente.
* Construye abstracciones:
  + procesos e hilos.
  + archivos y directorios
  + memoria lineal

***Arbitraje:***

* Permite que los usuarios/procesos utilicen y compartan (espacial/temporal) de forma transparente los recursos como si estuviesen solos en la máquina.
* La multiplexación puede ser:
  + Espacial: memoria.
  + Temporal: CPU, memoria.

***. – Internet:***

Dentro de las tareas que realiza el sistema operativo, en particular, se ocupa de gestionar la memoria de nuestro sistema y la carga de los diferentes programas, para ello cada programa tiene una prioridad o jerarquía y en función de la misma contará con los recursos de nuestro sistema por más tiempo que un programa de menor prioridad.

El sistema operativo se ocupa también de correr procesos. Llamamos proceso a la carga en memoria de nuestro programa, si no está cargado en memoria nuestro programa simplemente “no corre”.

1. SO multiprogramado

***. – Concepto Transparencias:***

El SO mantiene en memoria varios programas y alterna su ejecución solapando la ejecución de un programa con las realizaciones de entradas/salidas de otro, es decir, cuando un proceso libera la CPU para esperar por una E/S, el SO le asigna la CPU a otro programa.

Objetivo: aumentar la productividad (cantidad de trabajo realizada por unidad de tiempo) de la CPU.

***. – Concepto Internet:***

Se denomina multiprogramación a una técnica por la que dos o más procesos pueden alojarse en la memoria principal y ser ejecutados concurrentemente por el procesador o CPU.

Con la multiprogramación, la ejecución de los procesos (o hilos) se va solapando en el tiempo a tal velocidad, que causa la impresión de realizarse en paralelo (simultáneamente). Se trata de un paralelismo simulado, dado que la CPU solo puede trabajar con un proceso cada vez (el proceso activo).

De ahí que, en rigor, se diga que la CPU ejecuta «concurrentemente» (no simultáneamente) varios procesos; en un lapso de tiempo determinado, se ejecutarán alternativamente partes de múltiples procesos cargados en la memoria principal.

1. SO de tiempo compartido:

. – ***Concepto Transparencias:***

¿Qué ocurriría si un proceso no realiza E/S?

--\_>Monopolizaría la CPU.

El SO mantiene en memoria varios programas y alterna su ejecución retirándole el control de la CPU a un proceso para dárselo a otro programa.

Objetivo: construir sistemas interactivos.

Se denominan así pues el SO fuerza a los procesos a compartir el tiempo de CPU.

***. – Concepto Internet:***

Los sistemas de tiempo compartido se crearon para brindar el uso interactivo de un sistema de computador a un costo razonable.

Un sistema operativo de tiempo compartido utiliza técnicas de planificación y programación concurrente para dar la apariencia que se están ejecutando simultáneamente múltiples procesos y ofrecer a cada usuario una pequeña porción del tiempo de una computadora. Cada usuario tiene por lo menos un programa individual en la memoria.

Este tipo de SO Permiten la simulación de que el sistema y sus recursos son todos para cada usuario. El usuario hace una petición a la computadora, está la procesa tan pronto como le es posible, y la respuesta aparecerá en la terminal del usuario.

Los principales recursos del sistema, el procesador, la memoria, dispositivos de E/S, son continuamente utilizados entre los diversos usuarios, dando a cada usuario la ilusión de que tiene el sistema dedicado para sí mismo. Esto trae como consecuencia una gran carga de trabajo al SO, principalmente en la administración de memoria principal y secundaria

1. Características de un SO de Tiempo Compartido

* Dan la apariencia de que cada usuario tiene una máquina para sí mismo.
* La mayoría utilizan algoritmo de planificación circular. Es decir, a cada proceso se le da una fracción de tiempo para ejecutarse y al terminar esta, se ejecuta el siguiente proceso. El primer proceso se forma hasta que le toque su turno nuevamente.
* Evitan monopolizar el procesador; esto se logra asignando tiempos de procesador durante el cual se permite ejecutarse de forma ininterrumpida una parte del código de un proceso.
* La gestión de archivos y memoria, debe proporcionar protección y control de acceso debido a que pueden existir múltiples usuarios accediendo un mismo archivo.
* Pueden ser multiusuario, esto es, el uso de un sistema por más de una persona al mismo tiempo.

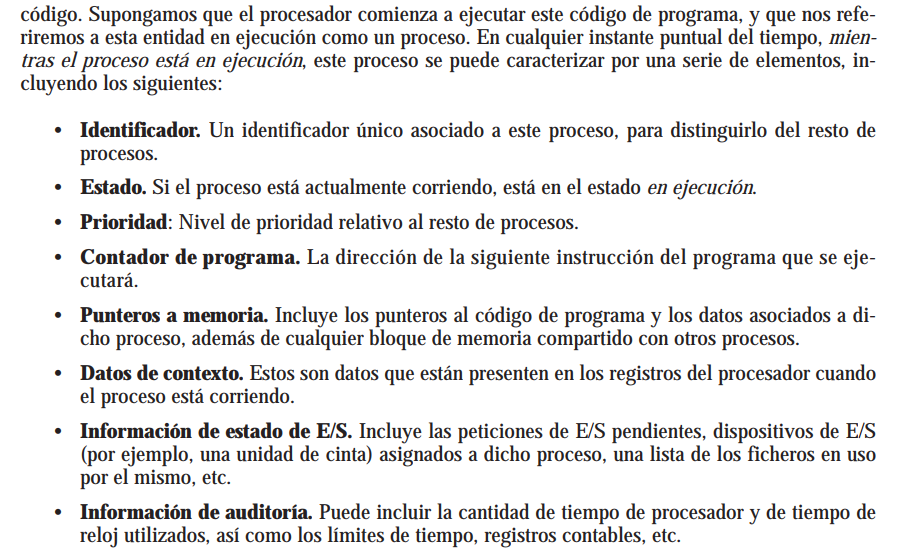
1. Que es un Proceso y como se materializa

***. – Concepto Transparencias:***

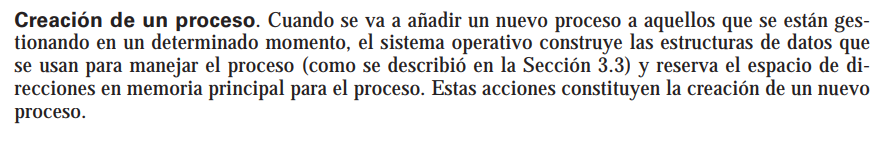
* Razones para introducir procesos:
  + Simplicidad – aislar las actividades
  + Velocidad – solapar actividades
  + Seguridad – limitar efectos de un error
* Un proceso es un recurso (CPU) virtual que nos hace creer que tenemos más de una CPU.
* Se materializa mediante la estructura de datos PCB (Bloque de Control de Proceso).

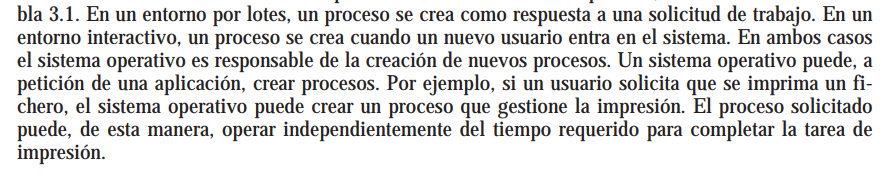
El principal concepto en cualquier sistema operativo es el de proceso. Un proceso es un programa en ejecución, incluyendo el valor del program counter, los registros y las variables. Conceptualmente, cada proceso tiene un hilo (thread) de ejecución que es visto como un CPU virtual.

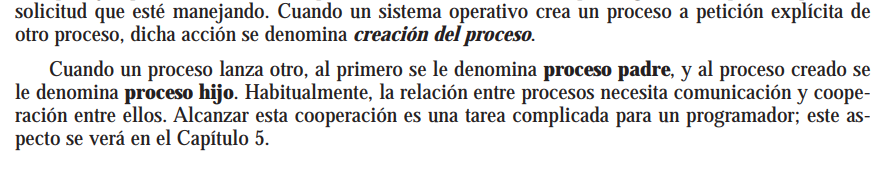
***. – Stallings:***

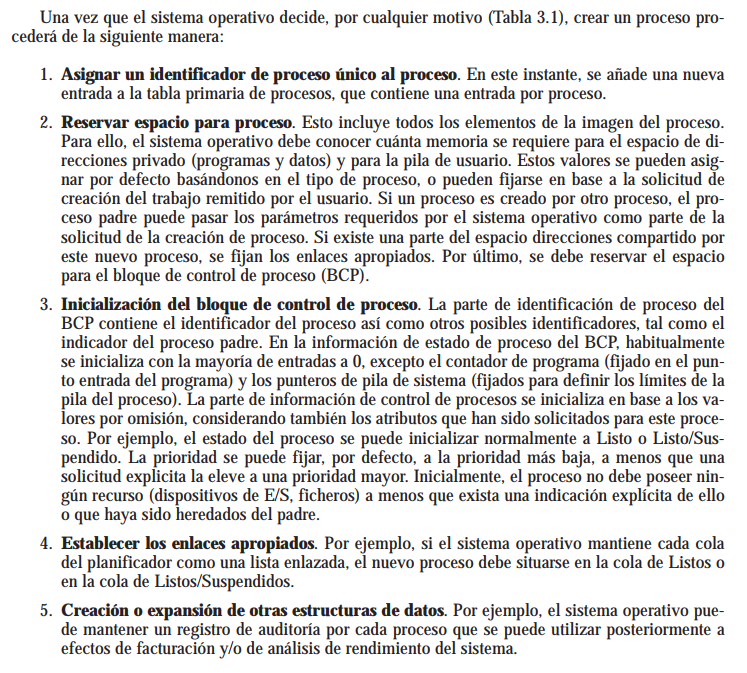


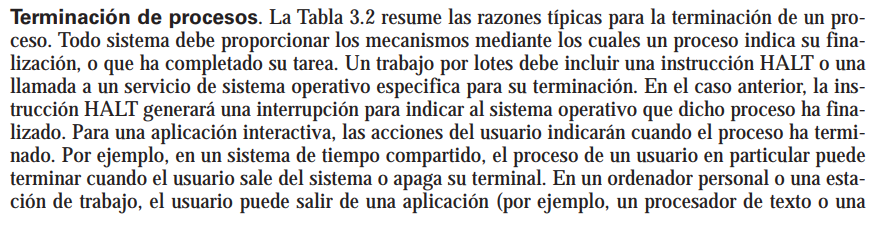
***. – Creación y terminación de procesos:***

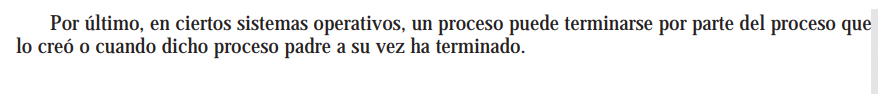




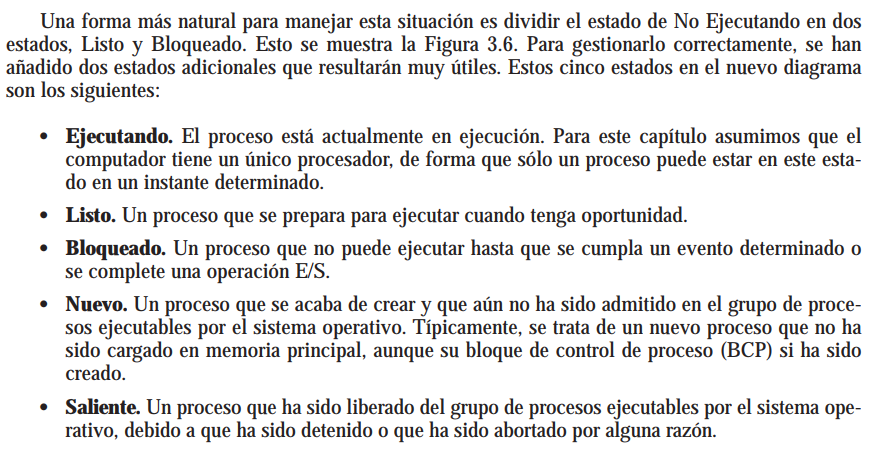


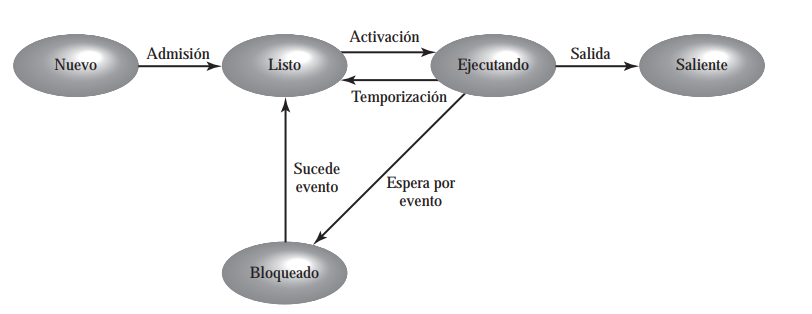


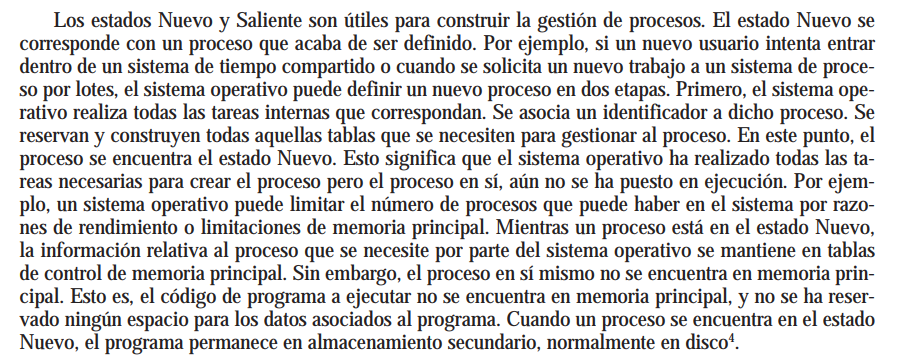


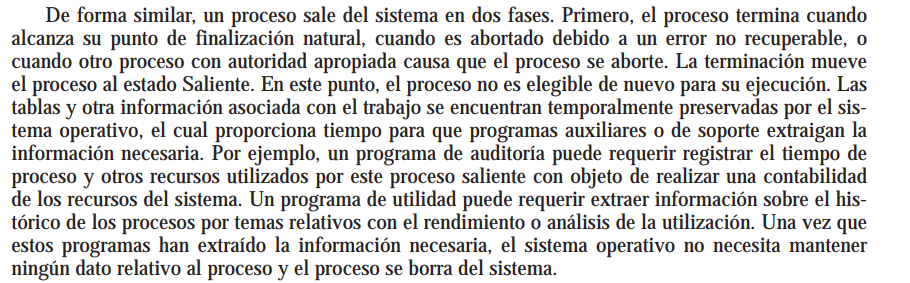
 

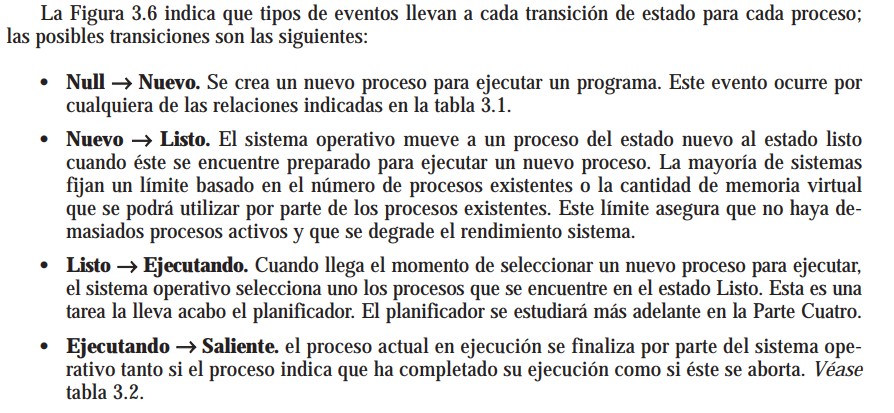
***. – Modelo de 5 Estados:***

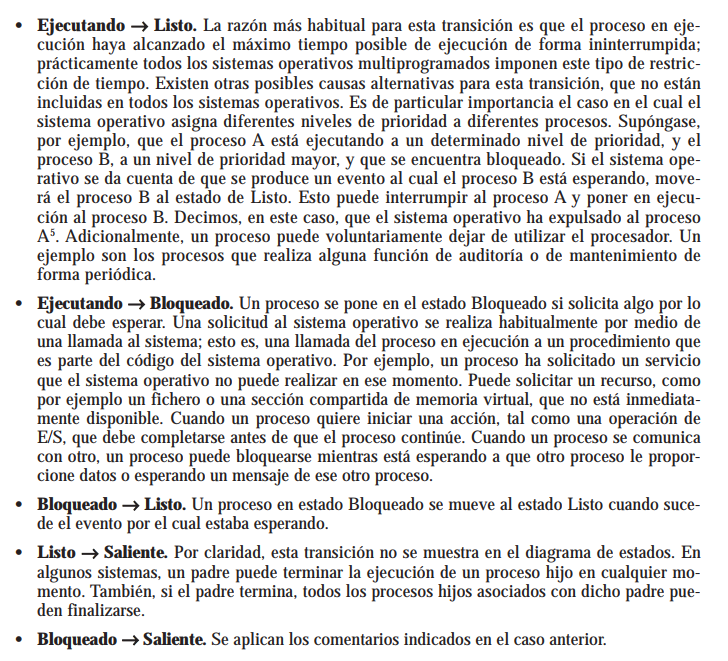




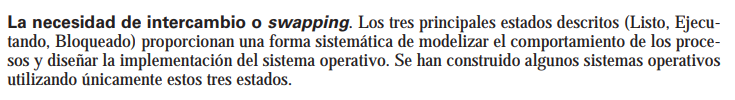




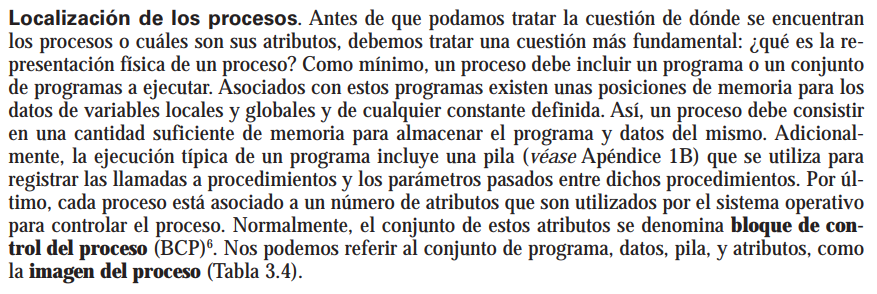


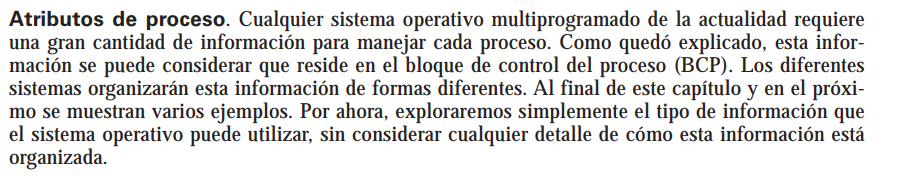


***. – Procesos Suspendidos:***

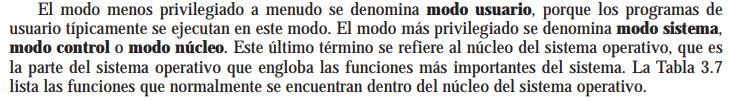


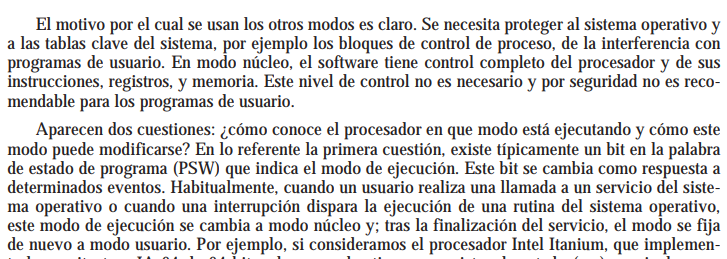
***. – Localización y atributos de los procesos:***

******

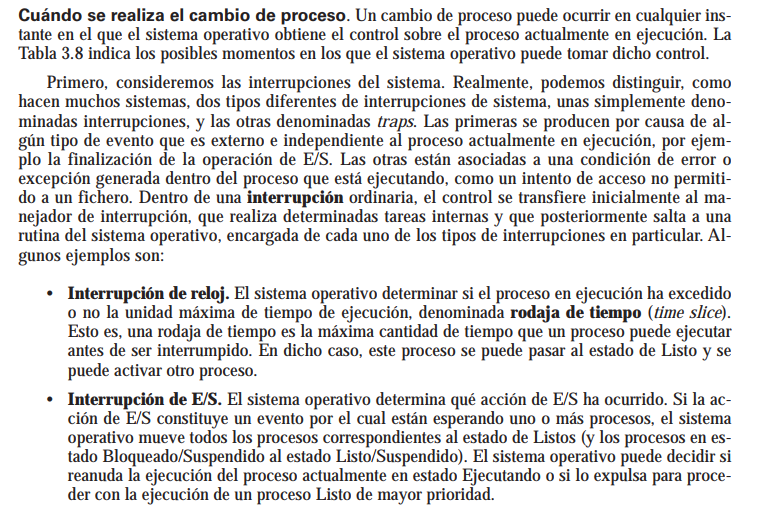


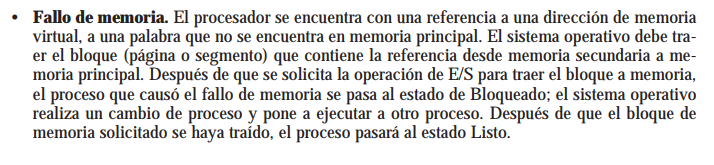
. – Modos de ejecución de un proceso:





. – Cambios de un proceso:





***. – Que es la imagen de un proceso:***

La imagen de un proceso es eso que pasa cuando un proceso tiene una imagen de los procesos dentro del mismo proceso(hilos):

* Espacio en memoria para almacenar los distintos procesos
* Espacio en memoria para almacenar la ‘pila de ejecución’ (Estructura de tamaño intermedio, donde se almacenan datos temporales necesarios en un proceso.)
* Bloque de control de proceso (bloqueo de control de proceso)’, (Registro con el cual el Sistema Operativo, toma información sobre cada proceso.)

1. Que es el PCB

El bloque de control del proceso (BCP) 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 BCP 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. Un proceso resulta conocido para el sistema operativo y por tanto elegible para competir por los recursos del sistema sólo cuando existe un BCP activo asociado a él. El bloque de control de proceso es una estructura de datos con campos para registrar los diferentes aspectos de la ejecución del proceso y de la utilización de recursos.

1. Que contiene el PCB

Toda la información que necesitamos para retomar el estado de ejecución de un programa:

* Dónde está el código y los datos
* Una pila de ejecución
* Valores de los registros del procesador
* Recursos del sistema en uso: memoria, archivos en uso, etc.
* Estado de la ejecución

1. Estados, PCB, etc.



▷ Cuando un proceso está ejecutándose, los valores de los registros (su contexto) son los del procesador.

▷ Cuando el SO detiene la ejecución de un proceso para bloquearlo, salva los registros del procesador en el PCB.

▷ La acción de conmutar entre procesos se denomina cambio de contexto.

▷ En procesadores con dos modos de funcionamiento, separamos el contexto en dos partes (una por modo):

* Contexto de usuario
* Contexto kernel – usado por el SO

▷ Un proceso se puede ejecutar en dos modos

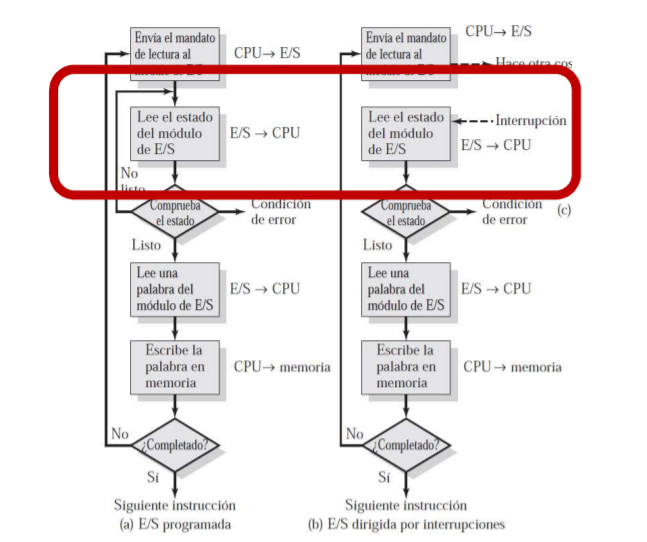
. – ***Como obtenemos un servicio del SO***

La biblioteca del lenguaje en uso (por ejemplo, la biblioteca estándar de C) funciona como envoltura a la instrucción INT que genera una trampa (excepción) que conmuta la CPU de modo usuario a modo Kernel. Para ello, hay que salvar en la pila de usuario del proceso el contexto de ejecución del proceso.

Cuando pasamos a modo kernel, la rutina (hay una única puerta de entrada) que atiende las llamadas al sistema determina la validez de los parámetros de la llamada y transfiere el control a la rutina kernel que sabe cómo atender la llamada.

La ejecución del proceso llamador en modo kernel utiliza la pila kernel para apilar llamadas a función posteriores. Funcionalmente es como una llamada a función, pero a nivel de ejecución tiene una penalización mucho más alta.

. ***- Entradas/salidas programadas y e/s gestionadas por interrupción.***



1. Definición de interrupción y de excepción

***. - Interrupción:***

Evento externo al procesador que necesita atención (se notifica activando el voltaje del pin INT del procesador). Son ajenas a la actividad en ejecución cuando se producen por lo que no debe modificarse el contexto actual de ejecución.

***. - Excepción:***

Evento interno al procesador que necesita atención (normalmente debido a un error en la ejecución de la instrucción actualmente en ejecución). Como no son ajenas a lo que estaba haciendo la CPU, si será necesario tocar el contexto actual de ejecución. La gestión de interrupciones y excepciones es muy similar.

Tema 1: Arquitecturas de los SOs

Tipos de arquitecturas de SOs:

-Introducción

-Monolítica

-Capas

-Máquinas virtuales

+Soporte de Linux la virtualización

Sistemas operativos de propósito especifico

-Sistemas de tiempo compartido

-Sistemas de tiempo-real

-Sistemas distribuidos y paralelos

# Tipos de arquitecturas de Sos

## Introducción

Características de una buena arquitectura de un sistema operativo:

-Eficiente: El sistema operativo ha de ser suficientemente eficiente como para no degradar el funcionamiento global del sistema y sus componentes.

-Fiable: Dos aspectos a considerar:

+Robusto: El sistema operativo debe responder de forma predecible comprendiendo los distintos partes y elementos del sistema en condiciones de error.

+El sistema operativo debe protegerse activamente a sí mismo y a los usuarios frente a acciones accidentales o malintencionadas.

-Adaptable: Deberíamos poder añadir/modificar funcionalidad del SO de forma flexible.

Arquitectura de un SOs:

Hoy en día para alcanzar a estas características partimos de lo que denominamos arquitectura software, debemos distinguir entre:

-Arquitectura de diseño: como se diseña “sobre el papel”

-Arquitectura de ejecución: como se construyen los programas del sistema operativo y como se ejecutan.

Ambas arquitecturas pueden ser distintas para un mismo sistema, aunque hoy en día la mayoría de los sistemas operativos son esencialmente monolíticos (ejecución de un solo programa a la vez), aunque estos pueden trabajar por capas (Windows o Linux).

## Estructura monolítica

Todos los componentes se ejecutan en modo kernel. El SO es el único “programa ejecutable”.

El modelo de obtención de servicio es la llamada procedimiento “protegido” que tiene el menor coste.

-------------------------------------------------------------------------------------------------------------------

Definición Stalling (En español):

-Núcleo monolítico:

Hasta hace relativamente poco tiempo, la mayoría de los sistemas operativos estaban formados por **un gran núcleo monolítico**. Estos grandes núcleos proporcionan la mayoría de las funcionalidades consideradas propias del sistema operativo, incluyendo la planificación, los sistemas de ficheros, las redes, los controladores de dispositivos, la gestión de memoria y otras funciones. Normalmente, un núcleo monolítico se implementa como un **único proceso**, con **todos los elementos compartiendo el mismo espacio de direcciones**. Una arquitectura micronúcleo asigna sólo unas pocas funciones esenciales al núcleo, incluyendo los espacios de almacenamiento, comunicación entre procesos (IPC), y la planificación básica.

Estructura modular:

La mayoría de los núcleos Linux son monolíticos. Como se mencionó anteriormente en el capítulo, un núcleo monolítico es aquél que incluye prácticamente toda la funcionalidad del **sistema operativo en un gran bloque de código** que ejecuta como un único proceso con un único espacio de direccionamiento. Todos los componentes funcionales del núcleo tienen acceso a todas las estructuras internas de datos y rutinas. **Si los cambios se hacen sobre cualquier porción de un sistema operativo monolítico, todos los módulos y rutinas deben volverse a enlazar y reinstalar, y el sistema debe ser reiniciado para que los cambios tengan efecto**. Como resultado, cualquier modificación, como por ejemplo añadir un nuevo controlador de dispositivo o función del sistema de fichero, es difícil. **Este problema es especialmente agudo para Linux**, cuyo desarrollo es global y ha sido realizado por un grupo de programadores independientes asociados de forma difusa.

Aportación Mauerer:

Elements of the kernel:

This section provides a brief overview of the various elements of the kernel and outlines the areas we will examine in more detail in the following chapters. Despite its monolithic approach, **Linux is surprisingly well structured**. Nevertheless, it is inevitable that its individual elements interact with each other; they share data structures, and (for performance reasons) cooperate with each other via more functions than would be necessary in a strictly segregated system.

Ampliación de hebras kernel:

Kernel threads are processes started directly by the kernel itself. They delegate a kernel function to a **separate process** and execute it there in ‘‘parallel‘‘ to the other processes in the system (and, in fact, in parallel to execution of the kernel itself). Kernel threads are often referred to as (kernel) daemons. They are used to perform, for example, the following tasks:

❑ To **periodically synchronize modified memory pages** with the block device from which the pages originate (e.g., files mapped using mmap).

❑ **To write memory pages** into the swap area if they are seldom used.

❑ To manage **deferred actions**.

❑ To implement transaction journals for filesystems.

Basically, there are two types of kernel thread:

❑ Type 1 — The thread is started and waits until requested by the kernel to perform a specific action.

❑ Type 2 — Once started, the thread runs at periodic intervals, checks the utilization of a specific resource, and takes action when utilization exceeds or falls below a set limit value. The kernel uses this type of thread for continuous monitoring tasks.

-------------------------------------------------------------------------------------------------------------------------------

Características del Sistema monolítico:

-Son **difíciles de comprende**r ya que se trata de un único programa, lo que es difícil de modificar y mantener.

-**No confinamiento de errores**, lo que puede dar lugar a un colapso total del sistema (pantallazo azul en Windows o panic() en Linux).

-Construcción de un programa final a base de **módulos compilados separadamente que se unen mediante un ligador**.

- Buena definición de parámetros de enlace entre las distintas rutinas existentes, que puede provocar **mucho acoplamiento**.

- Carecen de **protecciones y privilegios** al entrar a rutinas que manejan diferentes aspectos de los recursos de la computadora, como memoria, disco, etc.

- Generalmente están hechos a medida, por lo que **son eficientes y rápidos en su ejecución y gestión**, pero por lo mismo carecen de flexibilidad para soportar diferentes ambientes de trabajo o tipos de aplicaciones.

Arquitectura monolítica kernel el Linux:

Los kernels actuales de Linux son de este tipo (al igual que la mayoría de los sistemas operativos de producción) por razones de eficiencia.

Esto no quiere decir que el código fuente del sistema no está estructurado (como se ha mencionado en la aportación Mauerer en la pg.2, ver directorio /usr/src/linux/1 ).

Indica que el programa ejecutable correspondiente en complejo, en componentes e interacciones entre los mismos (recordar mapa del kernel en http://www.makelinux.net/kernel\_map/).

## Arquitectura de capas

Implementado como una serie de capas, cada una de ellas es una maquina abstracta respecto a la de abajo, jerárquicamente hablando.

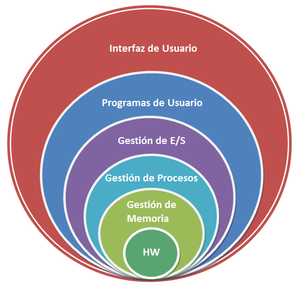
Por modularidad, las capas se seleccionan para que cada una utilice sólo funciones de las capas inferiores.

------------------------------------------------------------------------------------------------------------------------------

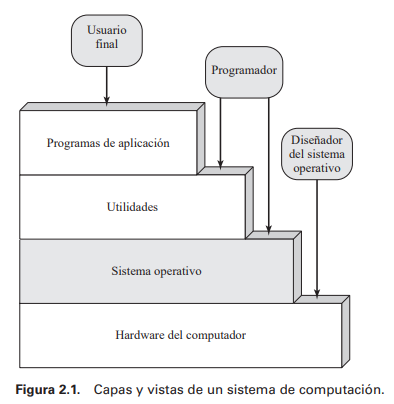
Internet(<http://sistemasoperativospj.blogspot.com/p/estructura-de-sistemas-operativos.html>)

Este modelo puede ser ejecutado en la mayoría de las computadoras, ya sean grandes o pequeñas.

Este sistema sirve para toda clase de aplicaciones, por tanto, es de propósito general y cumple con las mismas actividades que los sistemas operativos convencionales. Los procesos pueden ser tanto servidores como clientes. **Es una generalización del modelo de estructura simple para un sistema monolítico**. En esta arquitectura, el sistema operativo se divide en una **jerarquía de capas**, donde cada capa solamente **utiliza los servicios de la capa inferior y se enfoca en ofrecerle una interfaz clara y bien definida a la capa superior**. En cada capa **se encapsulan funciones específicas**, así cada capa se encarga de una **función individual** del sistema operativo.

**Su** **principal ventaja radica en poseer un diseño mucho más modular, seguro y escalable que el monolítico**. En este modelo, **lo único que hace el núcleo es controlar la comunicación entre los clientes y los servidores**. Al separar el sistema operativo en partes, cada una de ellas controla una parte del sistema, como el servicio a archivos, servicios a procesos, servicio a terminales o servicio a la memoria, y **cada parte es pequeña y controlable**. Además, como **todos los servidores se ejecutan como procesos en modo usuario y no en modo núcleo**, no tienen acceso directo al hardware. **En consecuencia, si hay un error en el servidor de archivos, éste puede fallar, pero esto no afectará en general a toda la máquina.**

1En informática, el acoplamiento es la forma y nivel de interdependencia entre módulos de software; una medida de qué tan cercanamente conectados están dos rutinas o módulos de software; ​ así como el grado de fuerza de la relación entre módulos.



Ventajas y desventajas de arquitectura de capas:

Ventajas:

-Reutilización de capas

-Facilita la estandarización

-Dependencias se limitan a intra-capa

-Contención de cambios a una o pocas capas

Desventajas:

-A veces no se logra la contención del cambio y se requiere una cascada de cambios en varias capas.

-Pérdida de eficiencia.

-Trabajo innecesario por parte de capas más internas o redundante entre varias capas.

-Dificultad de diseñar correctamente la granularidad de las capas.

-Los sistemas de capas son jerárquicos pero los reales son más complejos, p. ej.

+ El sistema de archivos podría ser un proceso en la capa de memoria virtual.

+La capa de memoria virtual podría usar archivos como almacén de apoyo de E/S

-Existe sobrecarga de comunicaciones entre las distintas capas.

-A menudo, los sistemas se modelan con esta estructura (Arq. diseño) pero no se construyen así (Arq. ejecución).

## Máquinas virtuales

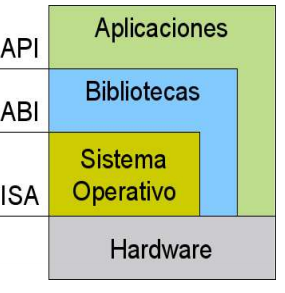
Virtualización es una tecnología que combina o divide recursos de computación para presentar uno o varios entornos de operación utilizando metodologías como particionamiento o agregación ya sea hardware o software, simulación de máquinas completa o parcial, emulación, tiempo compartido, y otras.

Denominamos Máquina Virtual (MV) al software que ‘simula’ una computadora y que puede ejecutar programas como si fuese unos computadores ‘real’.

Podemos verlo como un duplicado eficiente y aislado de una máquina real.

Los procesos que ejecuta una MV están limitados por los recursos y abstracciones que proporciona. Además, estos procesos están aislados del resto de procesos en otras MVs.

Virtualización: tipos

Podemos clasificar las MVs:

-Máquina virtual de procesos:

I. Nivel biblioteca: Wine

II. Nivel Lenguaje programación: JVM, Microsoft .NET CLI

III. Nivel SO: Jail, Containers

- Máquina virtual de sistemas

I. Emulación: Traducir ISA1 →ISA2

II. Virtualización:

1. Hipervisor nativo (Tipo 1)

2. Hipervisor anfitrión (Tipo 2)

Hipervisores

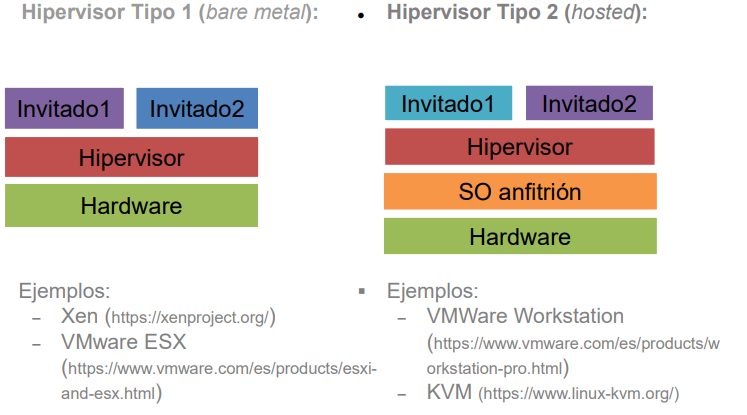
¿Qué es un hipervisor?

Cuando se crea una máquina virtual (VM, del inglés virtual machine), esta se ejecuta sobre la base de una máquina real no virtual, por ejemplo, un ordenador. La VM, por lo tanto, **depende del hardware físico**, por lo que debe existir una capa adicional entre los dos niveles que se haga responsable de la administración: se trata del hipervisor, un software que se hace cargo de **gestionar los recursos** necesarios para su funcionamiento. Este programa, también conocido como monitor de máquina virtual o **virtual machine monitor** (VMM), se encarga de **asignar memoria RAM, espacio en el disco duro, componentes de red o rendimiento del procesador en el marco del sistema**.

De esta manera, **varias y diferentes máquinas virtuales pueden ejecutarse en el sistema host**, ya que el hipervisor se asegura de que **no interfieran entre s**í y de que todas tengan a su disposición los recursos que necesiten.

En principio, el sistema huésped (es decir, la virtualización) no percibe en absoluto las medidas de gestión del VMM: el hipervisor abstrae el hardware de tal manera **que la VM asume que se encuentra un entorno de hardware establecido**. Como los requisitos en relación con los programas que se ejecutan cambian constantemente, incluyendo los de las máquinas virtuales, una gran ventaja del hipervisor es que **puede ir proporcionando los recursos según sea necesario**. El sistema huésped tampoco se da cuenta de ello: la máquina virtual no tiene forma de reconocer la existencia de otras máquinas que se ejecuten en el mismo hardware físico.

Como el hipervisor es solo una capa abstracta y la máquina virtual reproduce un entorno de hardware, el **sistema huésped no está vinculado a un hipervisor concreto**, del mismo modo que no está relacionado con un sistema host específico. Por este motivo, la virtualización a través del **hipervisor resulta muy interesante para los servicios en la nube**: los proveedores de servidores simplemente pueden trasladar el entorno de prueba virtualizado a otra máquina física sin causar problemas a los programas que se ejecutan en la VM. El virtual machine monitor también ofrece varias opciones de administración al usuario. De esta manera, cada sistema huésped se puede organizar y configurar. En casi todos los casos, es posible crear, configurar, ejecutar y eliminar máquinas virtuales a través de una interfaz gráfica de usuario.

Tipo de hipervisores:

Hipervisores, implementación:

Los SOs están construidos para ejecutarse en modo kernel. Ahora es el hipervisor el que se ejecuta en modo kernel ¿cómo se solventa? = ¿cómo manejo las instrucciones privilegiadas?

Según la implementación podemos separarlos en:

-Traducción binaria de código binario “crítico” del SO invitado (no de las aplicaciones).

-Paravirtualización: El código del SO invitado es recompilado para amoldarlo a la API1 anfitriona, eliminando la necesidad de que la MV atrape las instrucciones privilegiadas.

-Virtualización asistida por hardware: trata de resolver problemas del hardware relativos a la gestión de trampas. Ej. VT-x de Intel o SVM de AMD.

## Soporte de Linux a la virtualización

Linux soporta una virtualización ligera basada en dos mecanismos:

-Espacios de nombres (namespaces): permite que se vean propiedades globales del sistema desde diferentes aspectos o “vistas”.

-------------------------------------------------------------------------------------------------------------------

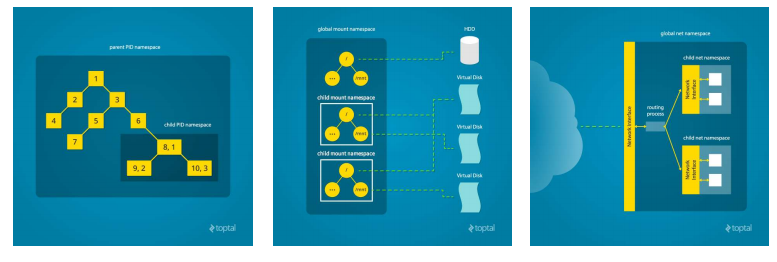
Ampliación de Mauerer:

During the development of kernel 2.6, support for namespaces was integrated into numerous subsystems. **This allows different processes to have different views of the system**. Traditionally, Linux (and Unix in general) use numerous global quantities, for instance, process identifiers: **Every process in the system is equipped with a unique identifier (ID),** and this ID can be employed by users (or other processes) to refer to the process — by sending it a signal, for instance. With namespaces, formerly global resources are **grouped differently**: Every **namespace can contain a specific set of PIDs**, or can provide different views of the filesystem, where mounts in one namespace do not propagate into different namespaces.

Namespaces are useful; for example, **they are beneficial for hosting providers**: Instead of setting up one physical machine per customer, they can instead use containers2 implemented with namespaces to create **multiple views of the system** where each seems to be a complete Linux installation from within the container and does not interact with other containers: They are **separated and segregated from each other. Every instance looks like a single machine** running Linux, but in fact, many such instances can operate simultaneously on a physical machine. This helps use resources more effectively. In contrast to full virtualization solutions like KVM, only a single kernel needs to run on the machine and is responsible to manage all containers.

Not all parts of the kernel are yet fully aware of namespaces, and I will discuss to what extent support is available when we analyze the various subsystems.

-------------------------------------------------------------------------------------------------------------------------------



Los espacios de nombre actualmente incluidos son:

- System V IPC – mecanismos de comunicación entre procesos.

- mounts - sistemas de archivos montados.

- UTS (Unix Timesharing System) – información del sistema (nombre, versión, tipo arquitectura, etc.)

- pid – espacio de nombres de identificadores de procesos

- network – conjunto de dispositivos de red.

- userid – permite limitar los recursos por usuario.

Otros Sos que soportan una virtualización similar son: Zones de Solaris o Jails de FreeBSD.

Posibles usos de los namespaces:

- Servidores Privados Virtuales (VPS) como por ejemplo Linux Containers (lxc.sourceforge.net)

1El término API es una abreviatura de Application Programming Interfaces, que en español significa interfaz de programación de aplicaciones.

2Los contenedores de aplicaciones son entornos ligeros de tiempo de ejecución que proporcionan a las aplicaciones los archivos, las variables y las bibliotecas que necesitan para ejecutarse, maximizando de esta forma su portabilidad.

- Application checkpoint and restart (ACS)

- En clúster: o Sustitución de NFS o Re-construcción de /proc

-------------------------------------------------------------------------------------------------------------------

Ampliación de internet(<https://cutt.ly/akhtoA6>):

Los espacios de nombres o namespaces permiten crear aplicaciones PHP más organizadas, flexibles y legibles. Los namespaces o espacios de nombres permiten crear aplicaciones complejas con mayor flexibilidad evitando problemas de conflictos entre clases y mejorando la legibilidad del código.

-------------------------------------------------------------------------------------------------------------------

-Grupos de control (cgroups)- permite la gestión particionada de recursos asignables a tareas o grupos de tareas.

Controlan los recursos asignados a una tarea/grupo de tareas.

Diferentes subsistemas (controladores de recursos):

-cpu: utilizado por el planificador para suministrar el acceso de las tareas de un cgroup a la CPU.

-cpuset: asigna CPUs individuales y memoria en multicores.

-devices: permite/deniega el acceso a un dispositivo.

-memory: limita el uso de memoria a tareas de un cgroup, y genera informes automáticos.

-blkio: establece los límites de accesos de E/S desde/hacia dispositivos de bloques (discos, USB, …)

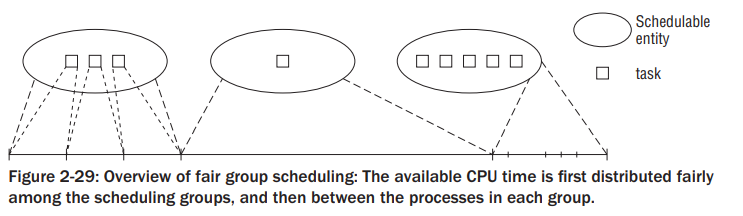
-ns: subsistemas de espacios de nombres.

-------------------------------------------------------------------------------------------------------------------

Ampliación de internet(<https://cutt.ly/tkhyyVf>):

Los cgroups o grupos de control, son una característica del kernel Linux que permite que los procesos se organicen en grupos jerárquicos con el fin de limitar y monitorear el uso de varios tipos de recursos. Con cgroups cada proceso corre en su propio espacio del kernel y de la memoria. Cuando se tienen la necesidad, un administrador puede configurar fácilmente un cgroup para limitar los recursos que puede utilizar un proceso.

Ampliación del Mauerer:

The kernel also offers control groups, which allow — via the special filesystem cgroups — creating arbitrary collections of tasks, which may even be sorted into multiple hierarchies. The situation is illustrated in Figure 2-29.

-------------------------------------------------------------------------------------------------------------------

Diferentes usos de los grupos de control:

Varias formas de uso:

- Seudo-sistema de archivos

- cgroupsfs

- herramientas libcgroup

- demonio engine rules

-Ejemplos, podemos:

+ Ver los grupos con cat /proc/cgroups

+ Ajustarlos a través de /sys/fs/cgroup/

+Algunas ordenes se construyen sobre cgroups: cpuset

Además, el kernel da soporte a MV como UML, XEN, KVM(KERNEL VIRTUAL MACHINE), VirtualBox, VMWare, Wine, etc.

## Sos de propósito especifico

## Sistemas de tiempo compartido

Soportan el uso “interactivo” del sistema:

- Cada usuario tiene la ilusión de disponer de la máquina completa.

- Tratan de optimizar el tiempo de respuesta.

-Basados en asignar fracciones de tiempo - se reparte el tiempo de CPU de forma equitativa entre los procesos. Permiten la participación activa de los usuarios en la edición, depuración de programas, y ejecución de los procesos.

-------------------------------------------------------------------------------------------------------------------

Ampliación de stallings:

La técnica se denomina tiempo compartido, porque se **comparte el tiempo de procesador** entre múltiples usuarios. En un sistema de tiempo compartido, múltiples usuarios acceden simultáneamente al sistema a través de terminales, **siendo el sistema operativo el encargado de entrelazar la ejecución de cada programa de usuario en pequeños intervalos de tiempo o cuantos de computación**. Por tanto, si hay n usuarios activos solicitando un servicio a la vez, cada usuario sólo verá en media 1/n de la capacidad de computación efectiva, sin contar la sobrecarga introducida por el sistema operativo. Sin embargo, dado el tiempo de reacción relativamente lento de los humanos, **el tiempo de respuesta de un sistema diseñado adecuadamente debería ser similar al de un computador dedicado**.

Ampliación de internet(<https://sistemasoperativos.angelfire.com/html/1.4.4.html>):

Permiten la simulación de que el sistema y sus recursos son todos para cada usuario. El usuario hace una petición a la computadora, **está la procesa tan pronto como le es posible**, y la respuesta aparecerá en la terminal del usuario.

**Los principales recursos del sistema, el procesador, la memoria, dispositivos de E/S, son continuamente utilizados entre los diversos usuarios**, dando a cada usuario la **ilusión de que tiene el sistema dedicado para su mismo**. Esto trae como consecuencia una **gran carga de trabajo al Sistema Operativo**, principalmente en la administración de memoria principal y secundaria.

Ejemplos de Sistemas Operativos de tiempo compartido son Multics, OS/360 y DEC-10.

Características de los Sistemas Operativos de tiempo compartido:

-**Populares representantes de sistemas multiprogramados multiusuario**, Ej.: sistemas de diseño asistido por computador, procesamiento de texto, etc.

   -Dan la **ilusión de que cada usuario tiene una máquina para sí**.

-Mayoría utilizan **algoritmo de reparto circular**.

-Programas se ejecutan con **prioridad rotatoria** que se incrementa con la espera y disminuye después de concedido el servicio.

-**Evitan monopolización** del sistema asignando tiempos de procesador (time slot).

     -Gestión de memoria proporciona **protección a programas residentes**.

-Gestión de archivo debe **proporcionar protección y control de acceso** debido a que pueden existir múltiples usuarios accediendo a un mismo archivo.

-------------------------------------------------------------------------------------------------------------------

Implementación del sistema de tiempo compartido:

Implementar tiempo compartido requiere modificar la RSI de Reloj para:

- Contabilizar el tick actual de reloj al proceso en ejecución

- Cuando el proceso haya consumido su tiempo (n.º ticks consumidos = tiempo asignado), lo marcamos como preparado y planificamos.

Para verlo en Linux es mejor usar una versión antigua (2.4.1) en el archivo kernel/timer.c, función do\_timer.c().

## Sistemas de tiempo real

Un sistema de tiempo-real (RTS) es un sistema informático que reacciona con el entorno (responde a eventos físicos) a los que debe responder en un plazo determinado.

**Las tareas ejecutadas tienen asociado tiempo límite (deadline)** en el que deben ejecutarse. Si no se satisface el deadline los **resultados pueden ser inútiles e incluso catastróficos**.

Un RTS necesita el soporte de un SO de tiempo-real (RTOS), que debe garantizar la planificación de todas las tareas de forman que **cumplan sus plazos**.

Prácticamente todos los SOs actuales incorporan alguno de los requisitos de tiempo-real.

-------------------------------------------------------------------------------------------------------------------

Ampliación internet(<http://icaro.eii.us.es/descargas/Tema6-parte1.pdf>).

-------------------------------------------------------------------------------------------------------------------

## Sistemas distribuidos y paralelos

Sistema distribuido:

Un sistema distribuido es un sistema multicomputador donde las diferentes **CPUs no comparten memoria, ni reloj**.

**El objetivo básico es compartir recursos** (hard o soft).

También permite un **aumento de la fiabilidad** del sistema: si una parte falla, el resto puede seguir con la ejecución.

-------------------------------------------------------------------------------------------------------------------

Ampliación del stallings:

Ventajas de los DDP (distributed data procesing) *(ya lo subrayáis vosotros con vuestros huevos que son las 11 de la noche y estoy hasta la polla)*:

• Receptividad. Las instalaciones de computación local se pueden gestionar de tal manera que pueden satisfacer más directamente las necesidades de gestión de las organizaciones locales que una instalación central, que intenta satisfacer las necesidades de toda la organización.

• Disponibilidad. Con múltiples sistemas interconectados, la pérdida de un sistema podría tener un impacto mínimo. Los sistemas y componentes prioritarios (por ejemplo, computadores con aplicaciones críticas, impresoras y dispositivos de almacenamiento masivo) podrían estar replicados de forma que un sistema de recuperación puede asumir la carga rápidamente después de un fallo.

• Compartición de recursos. El hardware más caro se puede compartir entre los usuarios. Los ficheros de datos se pueden gestionar y mantener de forma centralizada, pero con acceso a toda la organización. Los servicios para el personal, los programas y las bases de datos se pueden desarrollar en toda la organización y se pueden distribuir a las distintas instalaciones.

• Crecimiento incremental. En una instalación centralizada, un incremento de la carga de trabajo o la necesidad de un nuevo conjunto de aplicaciones normalmente supone la compra de gran equipamiento o de significativas actualizaciones de software. Esto implica un gasto importante. Además, estos cambios pueden requerir la conversión o reprogramación de aplicaciones existentes, con el correspondiente riesgo de errores y degradación del rendimiento. En un sistema distribuido, se pueden reemplazar gradualmente las aplicaciones o los sistemas, huyendo del enfoque de «todo o nada». Por último, los equipos antiguos se pueden dejar en las instalaciones para ejecutar aplicaciones sencillas, si el coste de pasar la aplicación a una nueva máquina no está justificado.

• Mayor participación y control del usuario. Con equipos más pequeños, más manejables, físicamente localizados cerca del cliente, el usuario tiene más posibilidades de influir en el diseño y funcionamiento del sistema, a través de la interacción con el personal técnico o a través de su supervisor inmediato.

• Productividad del usuario final. Los sistemas distribuidos tienden a tener mayor rapidez de respuesta, ya que cada pieza del equipo está realizando un trabajo más pequeño. Además, las aplicaciones y las interfaces pueden estar optimizadas a las necesidades del departamento. Los gestores del departamento pueden evaluar la efectividad de la parte local de la instalación y hacer los cambios apropiados. 13-Capitulo 13 12/5/05 16:27 Página 592 Para lograr estos beneficios, el sistema operativo debe proporcionar diversas funciones de soporte para DDP. Estas incluyen el software para el intercambio de datos entre máquinas, la capacidad de los clusters de computadores de lograr alta disponibilidad y altas prestaciones y la capacidad de manejar procesos en entornos distribuidos.

-------------------------------------------------------------------------------------------------------------------

Sistemas paralelos:

SOs para sistemas multiprocesadores: **sistemas de computador con varios procesadores que comparten memoria y un reloj**.

Sustentan **aplicaciones paralelas** cuyo objetivo es obtener aumento de velocidad computacional.

Multiprocesamiento Simétrico (SMP) - todos los procesadores pueden ejecutar tanto código del SO como de las aplicaciones.

-------------------------------------------------------------------------------------------------------------------

Ampliacion de mis colegones de la universidad de la plata (<http://sedici.unlp.edu.ar/bitstream/handle/10915/21327/Documento_completo.pdf?sequence=1&isAllowed=y>)

-------------------------------------------------------------------------------------------------------------------

Tema 2: Procesos e hilos

Implementación de las abstracciones proceso/hilo:

-Descriptor de proceso/hilo

Diagrama de estados y transiciones:

-Estados y sus propiedades

-Operaciones sobre los procesos

Planificación de la CPU:

-Tipos de planificadores y algoritmos de planificación básicos

-El planificador de Linux

-Ahorro de energía

# Implementación de las abstracciones proceso/hilo

En Linux, el PCB (Bloque de control de proceso) se denomina descriptor de proceso.

Viene definido por la estructura task\_struct, que podemos encontrar en la cabecera de las fuentes include/linux/sched.h:

-------------------------------------------------------------------------------------------------------------------

Ampliacion de stallings:

Supongamos que el procesador comienza a ejecutar este código de programa, y que nos referiremos a esta entidad en ejecución como un proceso. En cualquier instante puntual del tiempo, mientras el proceso está en ejecución, este proceso se puede caracterizar por una serie de elementos, incluyendo los siguientes:

• Identificador. Un identificador único asociado a este proceso, para distinguirlo del resto de procesos.

• Estado. Si el proceso está actualmente corriendo, está en el estado en ejecución.

• Prioridad: Nivel de prioridad relativo al resto de procesos.

• Contador de programa. La dirección de la siguiente instrucción del programa que se ejecutará. • Punteros a memoria. Incluye los punteros al código de programa y los datos asociados a dicho proceso, además de cualquier bloque de memoria compartido con otros procesos.

• Datos de contexto. Estos son datos que están presenten en los registros del procesador cuando el proceso está corriendo.

• Información de estado de E/S. Incluye las peticiones de E/S pendientes, dispositivos de E/S (por ejemplo, una unidad de cinta) asignados a dicho proceso, una lista de los ficheros en uso por el mismo, etc.

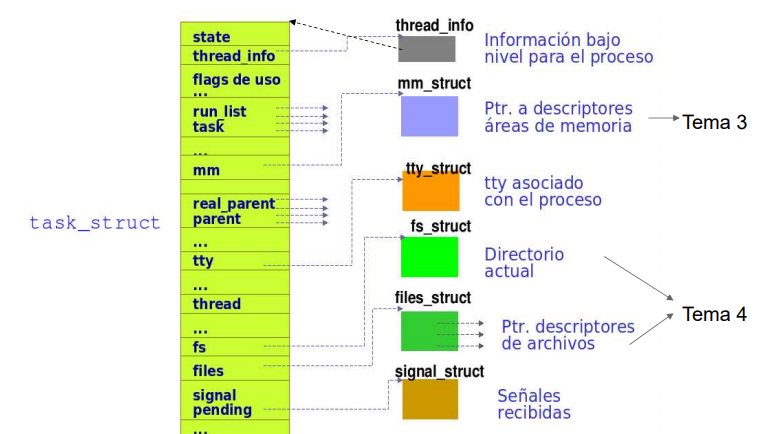
• Información de auditoría. Puede incluir la cantidad de tiempo de procesador y de tiempo de reloj utilizados, así como los límites de tiempo, registros contables, etc.

La información de la lista anterior se almacena en una estructura de datos, que se suele llamar **bloque de control de proceso** (process control block) (Figura 3.1), que el **sistema operativo crea y gestiona**. El punto más significativo en relación al bloque de control de proceso, o BCP, es que **contiene suficiente información de forma que es posible interrumpir el proceso cuando está corriendo y posteriormente restaurar su estado de ejecución como si no hubiera habido interrupción alguna**. El BCP es la herramienta clave que **permite al sistema operativo dar soporte a múltiples procesos y proporcionar multiprogramación**. Cuando un proceso se interrumpe, los valores actuales del contador de programa y los registros del procesador **(datos de contexto) se guardan en los campos correspondientes del BCP** y el estado del proceso se cambia a cualquier otro valor, como bloqueado o listo (descritos a continuación). El sistema operativo es libre ahora para poner otro proceso en estado de ejecución. El contador de programa y los datos de contexto se recuperan y cargan en los registros del procesador y este proceso comienza a correr.

**De esta forma, se puede decir que un proceso está compuesto del código de programa y los datos asociados, además del bloque de control de proceso o BCP**. Para un computador monoprocesador, en un instante determinado, como máximo un único proceso puede estar corriendo y dicho proceso estará en el estado en ejecución.

-------------------------------------------------------------------------------------------------------------------

## Descriptor de procesos Linux



-------------------------------------------------------------------------------------------------------------------

Ampliacion de internet:

En Linux, un proceso o tarea, está representado por una estructura de datos task\_struct. Linux mantiene una task, que es un vector lineal de punteros a cada estructura task\_struct definida en un instante dado. La estructura task\_struct contiene información de varios tipos:

· Estado. Representa el estado de ejecución de un proceso (en ejecución, interrumpible, no interrumpible, parado y zombie).

· Información de planificación. Representa a la información necesaria para la planificación de procesos en Linux. Un proceso puede ser normal o en tiempo real y tiene una prioridad. Los procesos en tiempo real se planifican antes que los procesos normales y se utilizan prioridades relativas dentro de cada categoría. El tiempo en el que un proceso puede ejecutarse se controla mediante un contador.

· Identificadores. Cada proceso tiene un único identificador de proceso, y tiene también identificadores de usuario y de grupo. Un identificador de grupo se utiliza para asignar privilegios de acceso a recursos de un grupo de usuarios.

· Comunicación entre procesos. Linux soporta los mecanismo de IPC de UNIX

· Vínculos. Cada proceso incluye un vínculo con su proceso padre, vínculos con sus hermanos (procesos con el mismo padre) y vínculos con todos sus hijos.

· Tiempos y temporizadores. Incluye el instante de creación del proceso y la cantidad de tiempo del procesador consumido hasta el instante. Un proceso puede tener también asociados uno o más temporizadores de intervalo. Un proceso define un temporizador de intervalo mediante una llamada al sistema; cuando el tiempo termina se envía una señal al proceso. Un temporizador puede ser de un solo uso o periódico.

· Sistema de archivos. Incluye punteros a cualquier archivo abierto por el proceso.

· Memoria virtual. Define la memoria virtual asignada al proceso.

· Contexto específico del procesador. La información de registros y pila que forma el contexto del proceso (espacio de direccionamiento del proceso).

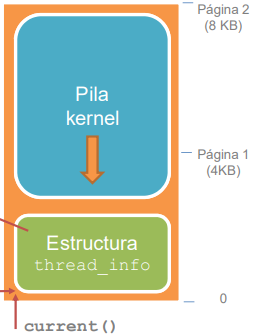
-------------------------------------------------------------------------------------------------------------------

Las subestructuras mm\_struct, files\_struct, fs\_struct, tty\_struct, signal\_struct se desgajan de la principal por varios motivos:

• No se asignan cuando no es necesario. Por ejemplo, una hebra kernel o ‘demonio’ no tiene asignado terminal por lo que tty\_struct→NULL, tampoco tiene definido espacio de usuario (solo código y datos kernel) por lo que mm\_struct→NULL.

• Permiten su compartición cuando sea necesario (volveremos sobre ellos al hablar de hilos y clone()).

Estructura thead\_info:

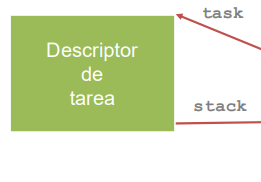
Contiene información de ejecución de bajo nivel sobre el proceso/hilo y permite acceder a la pila kernel del mismo.

Algunos campos:

♣ TIF\_SIGPENDING - señales pendientes

♣ TIF\_NEED\_RESCHED – es necesario re-planificar

♣ cpu – identificador CPU donde se ejecuta(ó)

♣ preempt\_count – contador de apropiaciones

Macros:

♣ current()