El Kernel

Ejecución Directa

Correr el programa directamente en la CPU.

Limitar la Ejecución Directa

Se realiza por distintos mecanismos de hardware:

- Dual Mode Operation
- Privileged Instructions
- Memory Protection
- Timer Interrupts

Dual Mode Operation

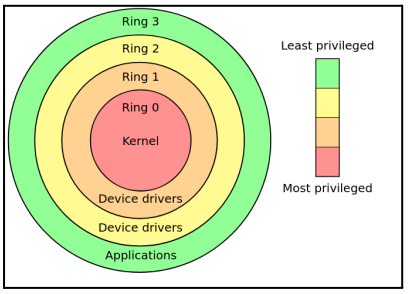
Kernel Land & User Land

Los Modos

Arquitectura x86 -> 4 modos de operaciones vía hardware -> **rings** (de 0 a 3):

- ring 0: **Modo Supervisor / Kernel / Monitor**
- ring 3: **Modo Usuario**

Cada instrucción a ser ejecutada es chequeada por el hardware para identificar en qué modo de operación se encuentra.



IOPL

Indicador que se encuentra en la CPU en modos protegido y largo.
Cuando el nivel de privilegio actual (CPL: CPL0, CPL1, CPL2, CPL3) del programa o tarea actual sea menor o igual que el IOPL, el/la mismo/a tendrá acceso a los puertos de I/O.

Privileged Instructions

Por la existencia del Modo Dual, los distintos modos poseen cada uno su propio set de instrucciones que pueden ser ejecutadas o no según el bit de modo de operación.

Memory Protection

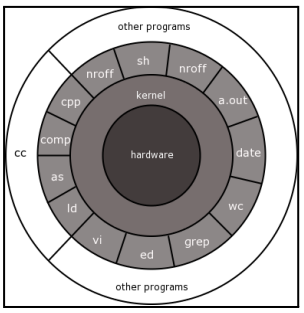
El SO y los programas que están siendo ejecutados deben residir ambos en memoria al mismo tiempo (el SO debe cargar el programa y hacer que comience a ejecutarse y el programa tiene que residir en memoria para poder hacerlo), de modo que -para

que la memoria sea compartida de forma segura- el SO debe poder configurar el hardware de forma tal que cada proceso pueda leer y escribir su propia porción de memoria.

Timer Interrupts

Es un mecanismo que le permite al Kernel -después de un determinado lapso de tiempo- desalojar al proceso de usuario en ejecución y volver a tomar el control del procesador -y así de toda la máquina-. Cada procesador tiene su propio contador. El reseteo del contador es una instrucción privilegiada.

Definición



Capa de software de más bajo nivel.
Controla los recursos de hardware y provee un ambiente en el cual los programas pueden ejecutarse.
Cuando el código fuente es ejecutado, la computadora pasa al estado **Kernel Mode**.

Tipos

- **Kernel Monolítico**: único proceso ejecutándose en memoria intercambiándose con la ejecución de los procesos de usuario.
- **Micro Kernel**: sólo implementa funcionalidad básica en el Ring 0; los servicios no imprescindibles en modo Kernel se implementan en Ring 1 o 2.

Tareas Específicas

- Planificar la ejecución de las aplicaciones
- Gestionar la memoria
- Proveer un sistema de archivos
- Creación y finalización de procesos
- Acceder a los dispositivos
- Comunicaciones
- Proveer una API

Modos de Transferencia

Modo Usuario -> Modo Kernel

- **Interrupciones** (evento externo)
    - Señal asincrónica al procesador por un evento externo
    - El procesador chequea si una interrupción es disparada
    - Orden de prioridad (por BCH):
        1. errores de la máquina
        2. timers

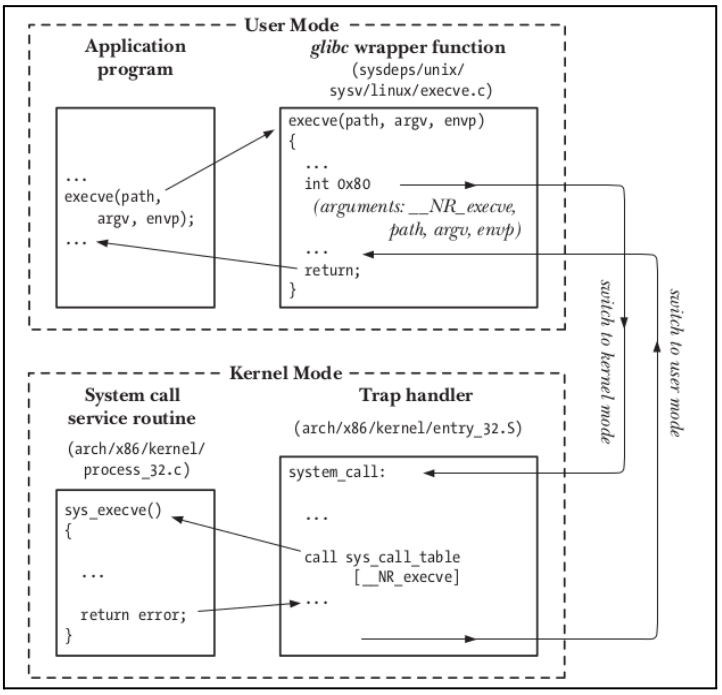
3. discos
4. network devices
5. terminales
6. interrupciones de software

- **Excepciones del procesador** (evento interno por un programa de usuario)
  - Parecido a interrupción
- **Ejecución de system calls** (evento intencional por una syscall)

System Call (syscall)

Punto de entrada controlado al Kernel que permite a un proceso solicitarle realizar alguna operación en su nombre:

- Cambia el modo de User Mode a Kernel Mode
- Conjunto fijo, cada una identificada por un único número
- Con parámetros que especifican información que debe ser transferida desde el User Space al Kernel Space



Llamada a una syscall

1. programa -> C wrapper -> syscall
2. wrapper.arguments -> trap_handling
3. wrapper.syscall -> %eax
4. wrapper -> trap machine instruction (int 0x80) -> trap handler 0x0
5. kernel.system_call() ? err : wrapper.err = errno

Kernel Mode -> User Mode

- **Nuevo Proceso**
- Después de
  - **interrupción**
  - **excepción del procesador**
  - **syscall**

  continúa con la ejecución del proceso interrumpido -> restaura de todos los registros
- Cambio entre **diferentes procesos** -> decide ejecutar otro proceso y carga el estado del nuevo a través de la PCB

El Proceso

Programa

Un Programa es un conjunto de instrucciones y datos que esperan en algún lugar del disco para saltar a la acción.

Fases de la Compilación

1. Procesamiento
2. Compilación
3. Ensamblaje
4. Link-Edición

Definición

Un Proceso es un programa en ejecución que incluye:

- archivos abiertos
- señales pendientes
- datos internos del Kernel
- estado completo del procesador
- espacio de direcciones de memoria
- uno o más hilos de ejecución, cada uno con:
  - un único program counter
  - un stack
  - un conjunto de registros
- una sección de datos globales

La Virtualización

★ De Memoria
★ De Procesador

De Memoria

Hace creer al proceso que tiene toda la memoria disponible para sí mismo.
El Address Space de un Proceso es:

- **Text**: instrucciones del programa
- **Data**: variables globales
- **Heap**: memoria dinámica
- **Stack**: variables locales

Memory Protection: Memoria Virtual

Abstracción por la cual la memoria física puede ser compartida por diversos procesos mediante la utilización de Virtual Address.

Traducción de Direcciones - Dirección Virtual a Dirección Física

Mapeo que se realiza por hardware en la **MMU**: Memory Management Unit.

De Procesador

Forma de virtualización más primitiva que consiste en dar la ilusión de la existencia de un único procesador para cualquier programa que requiera de su uso.

PCB: Process Control Block

Permite al SO:
- contabilizar los procesos en ejecución
- almacenar la información de un proceso

Por dentro

Un Proceso necesita permisos del Kernel del SO para:
- acceder a memoria perteneciente a otro proceso
- escribir/leer en el disco
- cambiar algún seteo del hardware del equipo
- enviar información a otro proceso

Arquitectura de Von Newman

El ciclo de una instrucción en VN es: Fetch - Decode - Execute - CP.

Contexto

Según [VAH]
- ➔ **User Address Space**: dividido en text, data, stack y heap
- ➔ **Control Information**: *u area y proc*
- ➔ **Credentials**: grupos IDs y UserId
- ➔ **Variables de Entorno**: valores heredados padre
- ➔ **Hardware Context**: contenido de los registros de propósito general
  - ◆ PC -> program counter
  - ◆ SP -> stack pointer
  - ◆ PWD -> processor status word
  - ◆ memory management registers
  - ◆ registros de la unidad de punto flotante

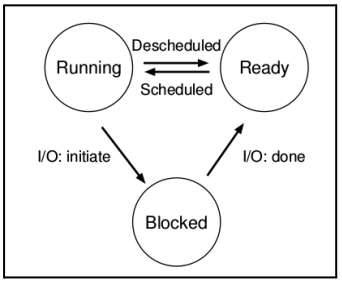
U Area y Estructura Proc

- U Area (user area):
  - Visible por el proceso cuando está ejecutándose
  - Información necesaria cuando el proceso está ejecutándose
- Proc Structure (aka: proc structure in linux):
  - Entrada en la *Process Table*
  - Información necesaria incluso cuando el proceso no se está ejecutando

Metamorfosis: De Programa a Proceso

1. El SO carga el programa en la memoria.
2. Se crea e inicializa el stack (memoria estática).
3. Se crea el heap (memoria dinámica).
4. Se setea el entry point de ejecución de las instrucciones.

Estados de un Proceso



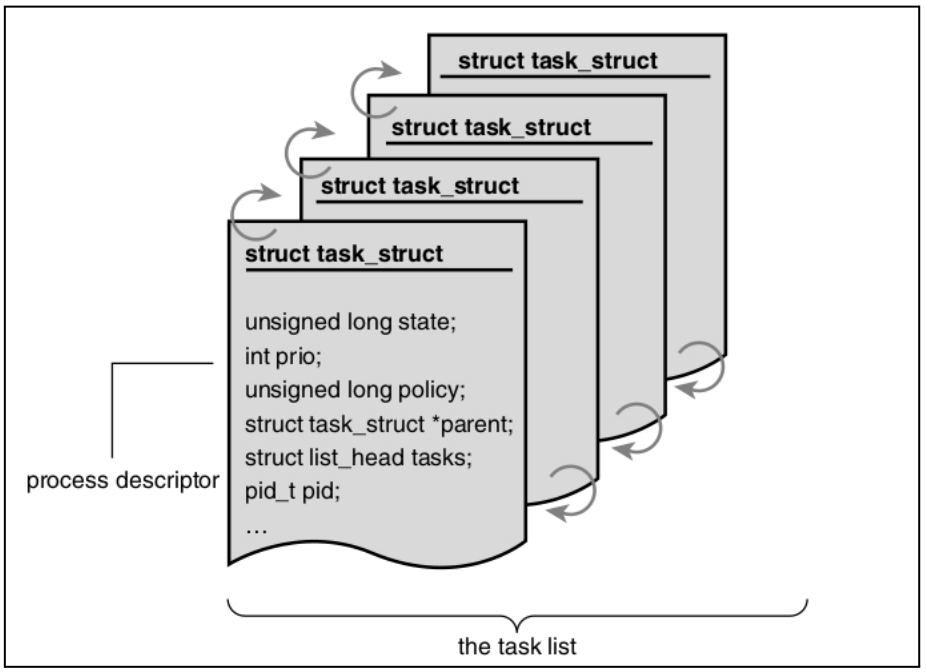
★ Running -> ejecutando instrucciones en un procesador
★ Ready -> listo para correr
★ Blocked -> ejecución en pausa hasta que algún evento suceda

Estados en Unix System V

➔ **Running User Mode**
➔ **Running Kernel Mode**
➔ **Ready to Run on Memory**
➔ **Asleep in Memory** -> bloqueado en memoria
➔ **Ready to Run but Swapped** -> listo para correr en memoria secundaria
➔ **Asleep Swapped** -> bloqueado en memoria secundaria
➔ **Preempt** -> running user mode que pasó antes por Kernel mode
➔ **Created** -> recién creado en un estado de transición
➔ **Zombie** -> el proceso padre ejecutó la syscall exit() y no existe más

Desde el Kernel de Linux

Task List: lista circular doblemente enlazada donde el Kernel almacena la lista de procesos.
Los procesos se identifican en Linux (al igual que en Unix) vía el PID.



Scheduling

Determina cuánto tiempo de CPU le toca a cada proceso y sucede en los SO de tipo *time sharing*.

Multiprogramación

Time Sharing ó Time Quantum

Período de tiempo que el Kernel le otorga a un proceso para poder compartir de forma concurrente un recurso computacional entre muchos usuarios.

Workload

Carga de trabajo de un proceso corriendo en el sistema. Cuanto mejor es el cálculo, mejor es la política de planificación.
Los supuestos sobre los procesos o *jobs* que se encuentran en ejecución son:

1. Cada uno se ejecuta la misma cantidad de tiempo
2. Todos llegan al mismo tiempo para ser ejecutados
3. Una vez que empieza uno, sigue hasta completarse
4. Todos usan únicamente CPU
5. El tiempo de ejecución (run-time) de cada uno es conocido

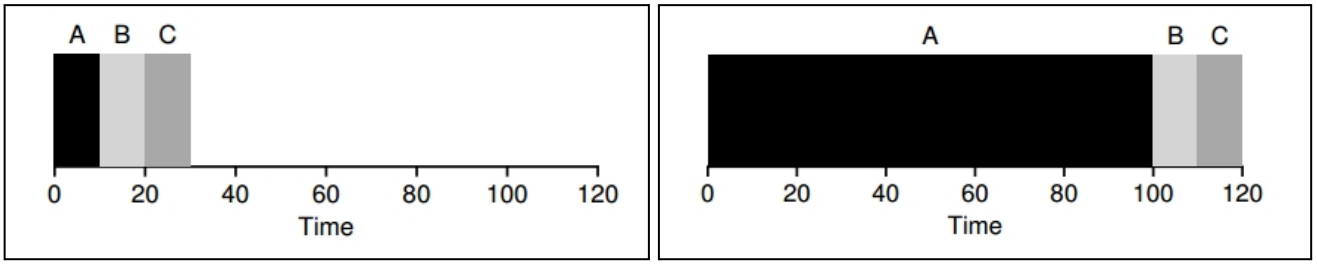
Métricas de Planificación

$$T_{turnaround} = T_{completion} - T_{arrival}$$

Políticas para Sistemas Mono-Procesador

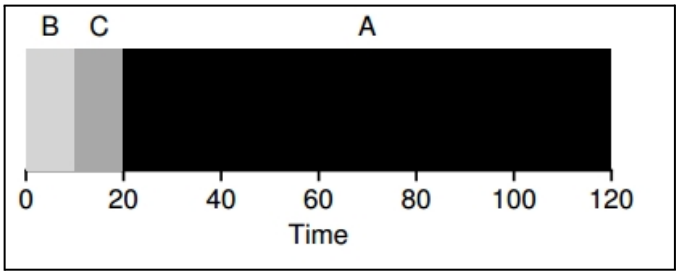
FIFO - First In First Out

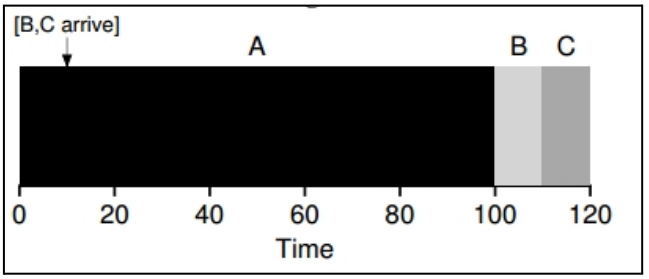
Es simple, fácil de implementar y funciona para las suposiciones iniciales.
Falla al relajar la suposición 1: se produce un efecto convoy.



SJF - Shortest Job First

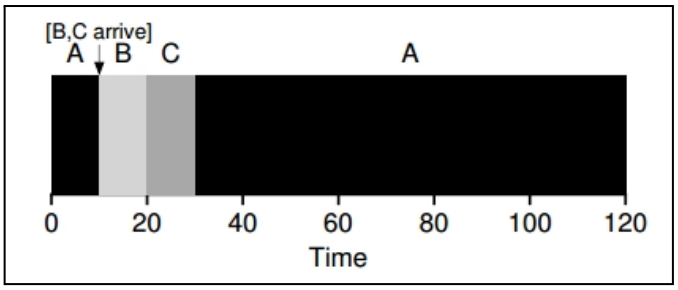
Se ordenan los procesos por tiempo de duración de menor a mayor.
Falla al relajar la suposición 2: el proceso que llega primero es el más largo.





STCF - Shortest Time to Completion

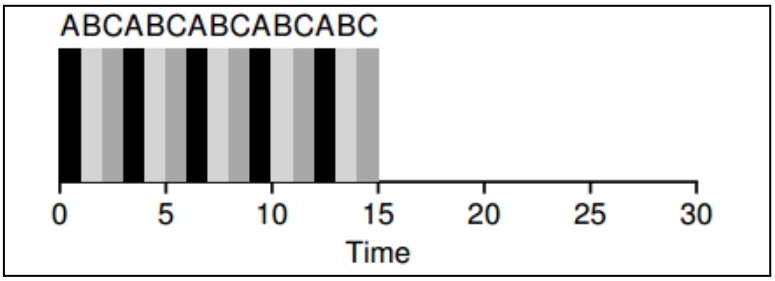
Al relajar la suposición 3: el planificador o sheduler puede adelantarse y determinar qué proceso debe ser ejecutado. Es una política preemptive.



Métrica de Tiempo de Respuesta

$$T_{response} = T_{firstrun} - T_{arrival}$$

Round Robin (RR)



1. se ejecuta un proceso por un período determinado de tiempo (slice)
2. transcurrido el período se pasa a otro proceso
3. repito cambiando de proceso en la cola de ejecución

Es importante la elección de un buen time slice: tiene que amortizar el cambio de contexto sin hacer que esto resulte en que el sistema no responda más.
Reduce el response time pero tiene un terrible turnaround time.

Multi-Level Feedback Queue (MLFQ)

Técnica de planificación que intenta atacar 2 problemas:

1. Optimizar el turnaround time
2. Minimizar el response time

Reglas Básicas

- **Regla 1**: if (PA > PB) then exec(A)
- **Regla 2**: if (PA == PB) then RR(A,B)

El planificador establece las prioridades basándose en el comportamiento observado:

➔ Si no utiliza la CPU, mantiene su prioridad alta

➔ Si utiliza intensivamente por largos períodos de tiempo el CPU, reducirá su prioridad

El ajuste se realiza según:

- **Regla 3**: cuando entra en el sistema se pone con la más alta prioridad
- **Regla 4a**: si usa un time slice mientras se está ejecutando su prioridad se reduce en una unidad
- **Regla 4b**: si renuncia al uso de la CPU antes de un time slice completo se queda en el mismo nivel de prioridad

Problema

1. Starvation: si hay demasiadas tareas interactivas se van a combinar para consumir todo el tiempo del CPU y las de larga duración nunca se van a ejecutar.
2. Se podrían reescribir los programas para obtener más tiempo de CPU.

Segundo Approach

- **Regla 5**: después de cierto período de tiempo *S* se mueven a la cola con más prioridad.

Así,

- se garantiza que los procesos no van a starve -> al ubicarse en la cola tope con las otras de alta prioridad se van a ejecutar utilizando RR y en algún momento recibirá atención,
- si un proceso que consume CPU se transforma en interactivo el planificador lo tratará como tal una vez que haya recibido el boost de prioridad.

El valor del tiempo *S* deberá ser determinado correctamente: si es demasiado alto los procesos que requieren mucha ejecución van a caer en starvation; si es muy pequeño las tareas interactivas no van a poder compartir adecuadamente la CPU.

Gaming

Para prevenir que ventajeen al planificador: se lleva una mejor contabilidad del tiempo de uso de la CPU en todos los niveles de la MLFQ. Reescribiendo:

- **Regla 4**: una vez que una tarea usa su asignación de tiempo en un nivel dado (sin importar cuántas veces renunció al uso de la CPU) su prioridad se reduce.

Proportional Share

Intenta garantizar que cada tarea tenga cierto porcentaje de tiempo de CPU. Cada tanto se realiza un sorteo para determinar qué proceso tiene que ejecutarse a continuación tal que los que deban ejecutarse con más frecuencia tienen que tener más probabilidades de *ganar la lotería*.

La Memoria

El Espacio de Direcciones o Address Space

Proceso que contiene todo el estado de la memoria de un programa en ejecución:

- El **código fuente** del programa vive en lo alto del espacio de direcciones. Se puede poner allí pues es estático y no necesitará más espacio mientras que el programa se ejecute.
- Mientras se esté ejecutando, el **heap** y el **stack** pueden crecer o achicarse y se colocan en los extremos del espacio de direcciones enfrentados entre sí permitiendo su crecimiento en direcciones opuestas (el heap empieza después del código fuente y crece hacia abajo y el stack empieza al final del espacio de direcciones y crece hacia arriba).

Abstracción fundamental sobre la memoria de una computadora que el SO le provee al programa en ejecución (aka: virtualización de memoria).

Metas

➔ Transparencia: el programa debe comportarse como si él estuviera alojado en su propia área de memoria física privada.
➔ Eficiencia: en términos de tiempo y espacio para no hacer que los programas corran más lentos ni usar demasiada memoria para las estructuras de virtualización.
➔ Protección: mediante el aislamiento entre procesos (cada uno tiene que ejecutarse en su propio caparazón aislado y seguro de los avatares de otros con fallas o incluso maliciosos).

El API de Memoria

Tipos de Memoria

★ Memoria de Stack: su reserva y liberación es manejada por el compilador en nombre del programador.
★ Memoria de Heap: es obtenida y liberada explícitamente por el programador.

Address Translation

Mecanismo proporcionado por el hardware que permite -a partir de una dirección virtual- obtener la dirección física correspondiente.

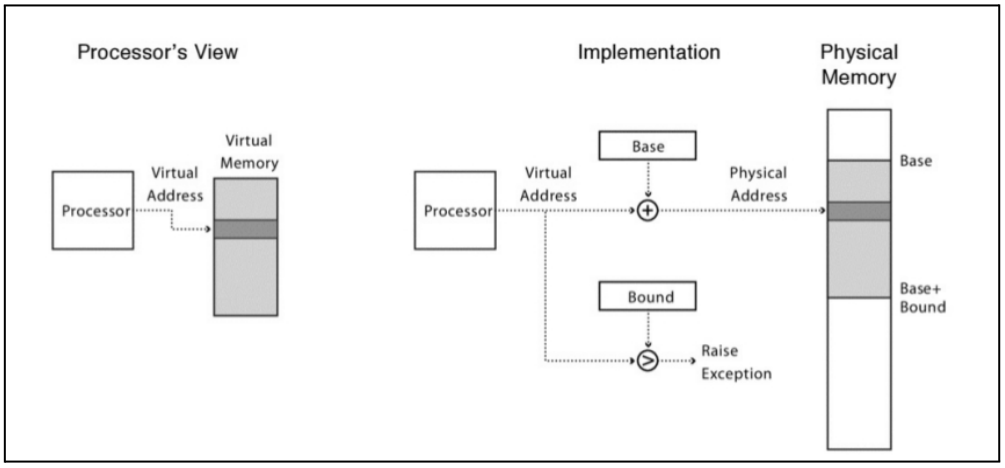
Memoria Segmentada o Dynamic Realocation[DAH]

Si un proceso genera una dirección virtual que es mayor que los límites o una dirección que es negativa, la CPU genera una excepción y el proceso finaliza.

Base & Bound

Dos registros en el CPU: registro base y registro límite o segmento (MMU). El proceso genera una referencia, el procesador la traducirá como: $physical\,address = virtual\,address + base$.
El bound register puede ser definido:

1. mantiene el tamaño del address space -> el hardware le suma primero el registro base.
2. almacena la dirección física del fin del espacio de direcciones.

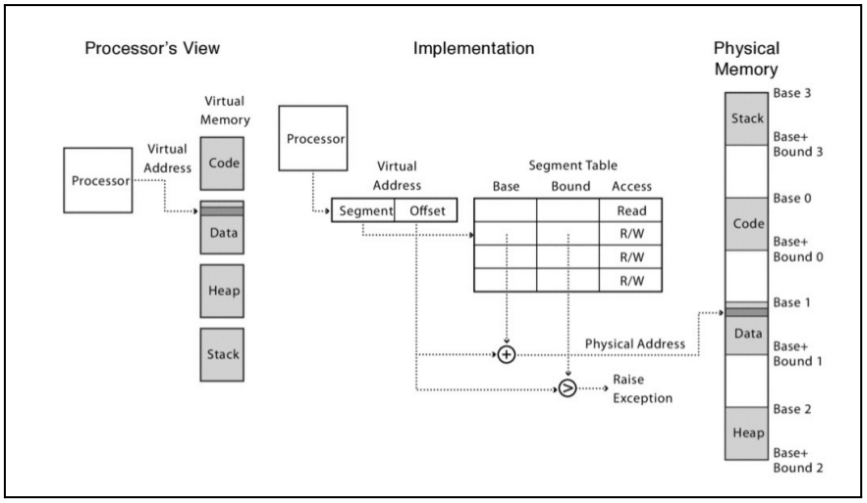


Se tiene un sólo registro base y un sólo segmento.

Segment Table

Pares de *(registro base, segmento)* por cada proceso -> cada entrada controla una porción virtual del address space.
La memoria virtual tiene dos componentes:

- Número de segmento: índice de la tabla para ubicar el inicio del segmento en la memoria física.
- Offset de segmento: sumado al registro base y comparado contra el registro bound para que el proceso no acceda a regiones indebidas de memoria.



Problema: Fragmentación externa.

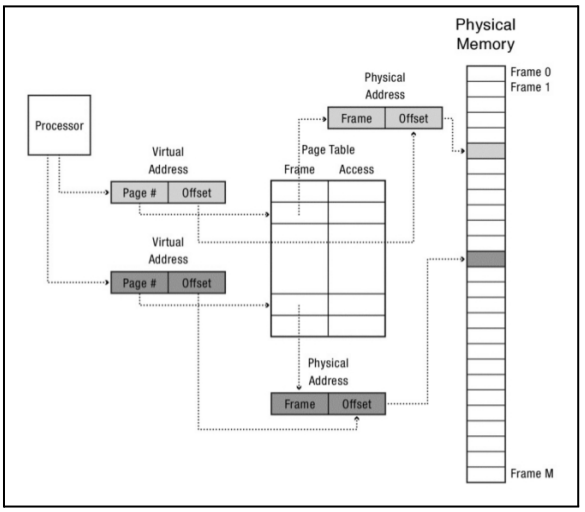
Memoria Paginada

La memoria es reservada en pedazos de tamaño fijo (potencia de 2): **page frames**. Se tiene una tabla de páginas por cada proceso cuyas entradas contienen punteros a las page frames.

Page Table

La memoria virtual tiene dos componentes:

- Número de página virtual: índice en la page table para obtener el page frame en la memoria física.
- Offset dentro de esa página: sumado a la page table compone la physical frame page.



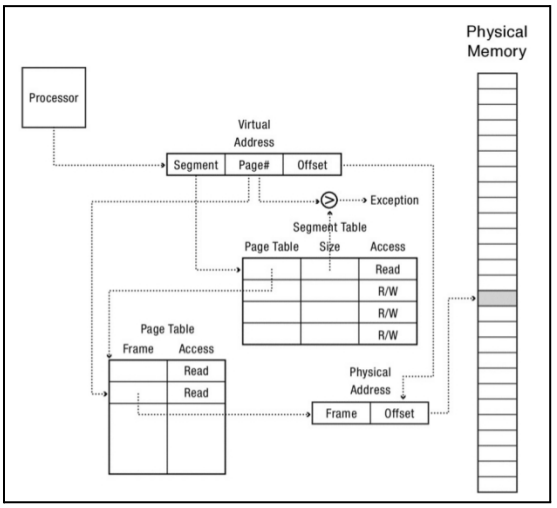
Problema: Saber qué espacio de la memoria está vacío es complicado.

Translation multilevel

Paged Segmentation

La memoria virtual tiene 3 componentes:

1. Número de segmento: índice para la segment table que aloja la page table para ese segmento.
2. Número de página virtual dentro de ese segmento: índice de la page table que contiene una page frame en la memoria física.
3. Offset dentro de la página: sumado al physical page frame que pertenece a la page table compone la physical address.



Tres Niveles de Page Tables

Implementa múltiples niveles de page tables.

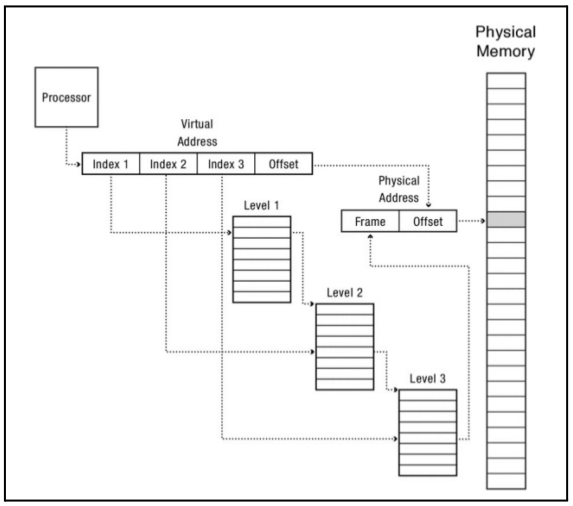
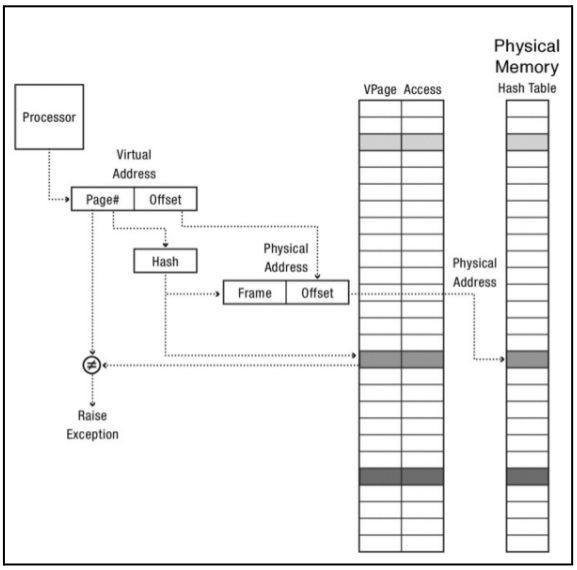


Tabla de Hash por Software



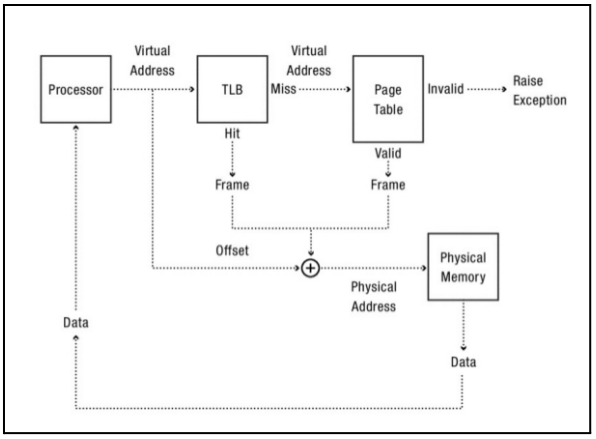
Hacia una eficiente Address Translation

Lookaside buffer

Pequeña tabla a nivel hardware que contiene los resultados de las recientes traducciones de memorias realizadas. Cada entrada de la tabla mapea una virtual page a una physical page.

Se chequean todas las entradas de la TLB contra la virtual page:

- ➔ **TLB hit**
- ➔ **TLB miss**



La TLB

Se extrae la virtual page number (VPN) de la virtual address y se chequea si la TLB tiene esa traducción para esa VPN

1. TLB HIT: existe la traducción
    a. se extrae el Page Frame Number (PFN) de la entrada en la TLB
    b. se concatena con el offset de la virtual address original
    c. se conforma la physical address para acceder a la memoria
2. TLB MISS: no existe la traducción
    a. el hardware accede la page table para encontrar la traducción
    b. si la referencia a la memoria virtual generada por el proceso es válida y accesible se actualiza la TLB con la traducción hecha por el hardware
    c. el hardware vuelve a buscar la instrucción en la TLB y la referencia a la memoria es procesada rápidamente

Consistencia de la TLB

El SO tiene que asegurarse que cada programa ve sólo su memoria:

- **Context Switch**: descarta el contenido de la TLB (flush) pues las direcciones virtuales del viejo proceso no son válidas para el nuevo.
- **Reducción de Permiso**: mantener la TLB consistente con la page table.
- **TLB shutdown**: cada vez que una entrada en la page table es modificada, la correspondiente entrada en todas las TLB tiene que ser descartada.

Concurrencia

La Abstracción

Un thread es una secuencia de ejecución atómica que representa una tarea planificable de ejecución que se caracteriza por:

- ➔ id
- ➔ conjunto de valores de registros
- ➔ stack propio
- ➔ política y prioridad de ejecución
- ➔ propio errno
- ➔ datos específicos

One thread per process

Proceso con una única secuencia de instrucciones ejecutándose de inicio a fin.

Many thread per process

Un programa es visto como threads ejecutándose dentro de un proceso con derechos restringidos.

Many single-threaded processes

Limitación de algunos SO que permitían varios procesos, cada uno con un único thread, resultando varios threads ejecutándose en kernel mode.

Many kernel threads

El kernel puede ejecutar varios threads en Kernel Mode.

Thread Scheduler

Cada thread corre en un procesador virtual exclusivo con una velocidad variable e impredecible: desde el punto de vista del thread, cada instrucción se ejecuta inmediatamente una detrás de otra.

El **Multi-threading** puede ser:

1. Cooperativo: no hay interrupción a menos que se solicite
2. Preemptivo: un thread en estado de running puede ser desalojado

Estructura y Ciclo de Vida de un Thread

El SO guarda y carga el estado de cada thread. Existen dos estados:

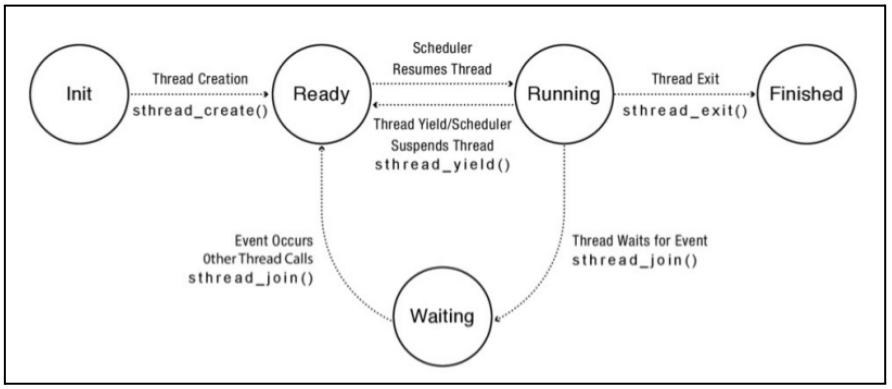
1. Per thread
2. Compartido entre varios threads

TCB: El Estado Per-thread y Threads Control Block

Estructura que representa el estado de cómputo de un thread y almacena:

- el puntero al stack del thread
- una copia de sus registros en el procesador

Estados de un Thread



Threads y Linux

Diferencias Proceso/Thread

	Threads	Procesos
Heap	✅	❌
Stack	❌	❌
File Descriptors	✅	❌
Contexto del FS	✅	❌
Signals	✅	❌
Code Segment	✅	✅
Data Segment	✅	✅
Metadata	❌	❌
Registros de CPU	❌	❌

Sincronización

Race Conditions

El resultado de un programa depende de cómo se intercalan las operaciones de los threads que se ejecutan dentro de un proceso.

Operaciones Atómicas

No pueden dividirse en otras y se garantiza la ejecución de la misma sin tener que intercalar ejecución.

Locks

Variable que permite la sincronización mediante la **exclusión mutua**: cuando un thread tiene lock ningún otro puede tenerlo. Tiene dos estados: busy & free.

Propiedades

Asegura:

- ➔ Exclusión Mutua: como mucho un solo thread posee el lock a la vez.
- ➔ Progress: si nadie posee el lock alguno debe poder obtenerlo.
- ➔ Bounded Waiting: si T quiere acceder al lock y existen varios en la misma situación, los demás tienen una cantidad finita de posibles accesos antes que T lo haga.

Sección Crítica

Sección del código fuente encerrada dentro de un lock que se necesita ejecutar en forma atómica.

Condition Variables: Esperando por un cambio

Permiten a un thread esperar a otro para tomar una acción y compartir algún estado que está protegido por un lock.

Errores

- ➔ **Non-deadlock Bugs**
  - ◆ Atomicity violation
  - ◆ Order Violation
- ➔ **Deadlock Bugs**
  - ◆ Un thread obtiene un lock y no lo libera y los demás threads se bloquean
  - ◆ Condiciones necesarias:
    - **Exclusión mutua**
    - **Hold-and-Wait**
    - **No preemption**
    - **Circular wait**

File System

Es una abstracción del sistema operativo que provee datos persistentes con un nombre.

Asocia a un archivo un identificador que permite compartir la información entre los distintos programas.

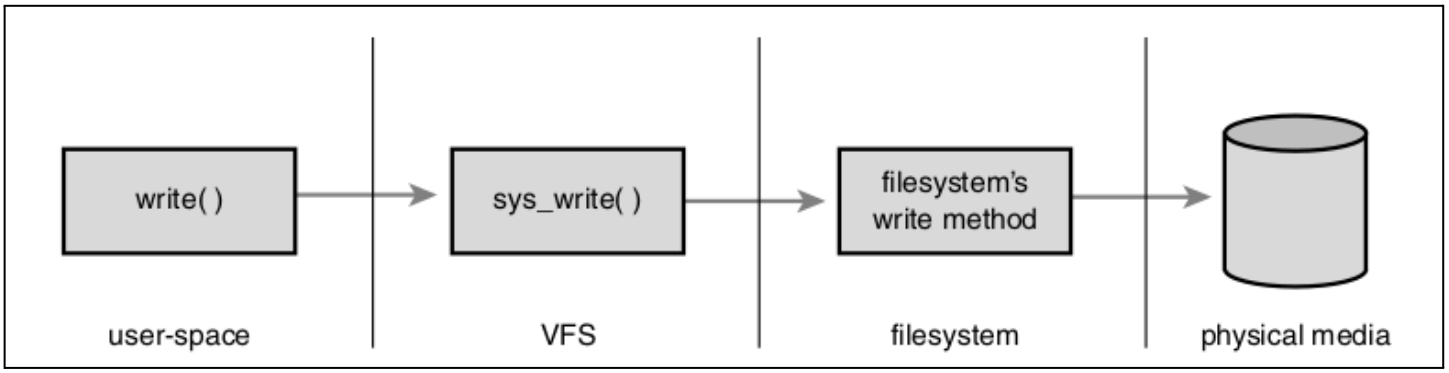
- archivos: compuestos por un conjunto de datos
- directorios: definen nombres para los archivos

El Virtual File System (VFS)

Subsistema del Kernel que implementa las interfaces de archivos y el sistema de archivos provistos a los programas corriendo en modo usuario. Todos los sistemas de archivos deben basarse en VFS para coexistir e inter-operar. Habilita a los programas a utilizar las syscalls de Unix para leer y escribir en diferentes sistemas de archivos y diferentes medios.

File System Abstraction Layer

Capa de abstracción general que le permite al kernel manejar muchos tipos de sistemas de archivos de forma fácil y limpia:



Objetos

- Super bloque: representa a un sistema de archivos
- Inodo: representa a un determinado archivo
- Dentry: representa una entrada de directorio
- File: representa a un archivo asociado a un determinado proceso

Operaciones

Aplicadas por el Kernel:

- super_operations -> sobre el FS (ejemplo: write_inode, sync_fs)
- inode_operations -> sobre un archivo (ejemplo: create, link)
- dentry_operations -> sobre un directorio (ejemplo: d_compare, d_delete)
- file_operations -> sobre un archivo abierto por un proceso (ejemplo: read, write)

Archivos

Colección de datos con un nombre específico para referirse a una cantidad arbitraria de datos siendo éste la abstracción de más alto nivel que la que subyace en el dispositivo de almacenamiento.

Información Almacenada

Metadata

Información acerca del archivo que es comprendida por el SO:

- tamaño
- fecha de modificación
- propietario
- información de seguridad

Datos

Información almacenada (arreglo de bytes sin tipo).

Directorios

Lista de nombres human-friendly y su mapeo a un archivo o a otro directorio.

Definiciones

- root directory -> directorio del que cuelgan todos los demás
- absolute path -> ruta desde el directorio raíz
- relative path -> path relativo equivalente a partir del directorio actual
- current directory -> directorio actual
- hard link -> mapeo entre el nombre y el archivo subyacente
- soft links -> cuando un archivo puede ser llamado por diferentes nombres
- volumen -> colección de recursos físicos de almacenamiento
- mount point -> punto en el cual el root de un volumen se engancha dentro de la estructura existente de otro file system