Apunte de Sistemas Operativos FIUBA

lcondoriz

Agosto 2023

Índice

1. Introducción

Resumen hecho por lcondoriz.

2. ¿Qué es un sistema operativo?

Un sistema operativo es el software encargado de hacer que la ejecución de los programas parezca elgo fácil. La principal forma para lograr esto es mediante el concepto de **virtualización**. Esto significa que el sistema operativo toma un recurso físico (*Ej. Memorias, procesadores, persistencia*) y lo transforma en algo mas general y fácil de usar.

El sistema operativo unix esta conformado por:

- **Kernel:** Es el programa central del sistema operativo que tiene control total sobre todo en el sistema. Facilita las interacciones entre el hardware y el software y gestiona los recursos como la memoria, el procesador, los dispositivos y las interrupciones. Es el pedazo de código que esta interactuando con privilegios absolutos sobre el harware y provee una interfaz a los usuarios
- **Drivers:** Son programas que permiten al kernel comunicarse con los dispositivos de hardware y controlar su funcionamiento
- **Syscall:** Es la interfaz que permite a los programas de usuario solicitar servicios al kernel mediante llamadas al sistema
- **System utilities:** Son programas que realizan funciones básicas del sistema operativo como la administración de archivos, procesos, usuarios, redes, etc. Ejemplos ls, cat, mv, pwd...etc.
- **Disk pkgs:** Son paquetes de software que se instalan en el disco duro y que proporcionan funcionalidades adicionales al sistema operativo3.
- **Entorno gráfico:** Es la parte del sistema operativo que permite al usuario interactuar con el sistema mediante una interfaz visual basada en ventanas, iconos, menús, etc.
- **Usuario:** Es la persona que utiliza el sistema operativo y sus aplicaciones.

Un sistema operativo deberá virtualizar varios recursos físicos:

- El procesador.
- La memoria.
- La persistencia (dispositivos de almacenamiento).

Un sistema operativo cumple simultáneamente tres roles:

- Referee: gestiona recursos compartidos
- Ilusionista: abstracción del hardware
- Pegamento: servicios comunes

Modos de ejecución de un SO:

- **Kernel mode:** Es el modo privilegiado en el que se ejecuta el sistema operativo y tiene acceso completo y sin restricciones al hardware y a toda la memoria. Gracias a la existencia del kernel, los programas son independientes del hardware subyacente.
- **User mode:** Es el modo restringido en el que se ejecutan las aplicaciones y tiene un espacio de direcciones virtuales privado y limitado. El modo usuario no puede acceder directamente al hardware ni a las direcciones de memoria reservadas para el sistema operativo. El modo usuario debe hacer llamadas al sistema para solicitar servicios al sistema operativo. El modo usuario es también llamado modo esclavo, estado problemático o modo restringido.

3. El kernel

En un sistema operativo, el kernel (núcleo en español) es la parte central que actúa como intermediario entre el hardware y el software. Es el componente esencial del sistema operativo y se encarga de administrar los recursos del sistema, proporcionando servicios básicos a las aplicaciones y controlando el acceso y la comunicación con el hardware.

El kernel tiene varias responsabilidades clave, entre las que se incluyen:

1. **Gestión de memoria:** El kernel se encarga de asignar y liberar memoria para los procesos y programas en ejecución, asegurándose de que cada proceso tenga acceso a la cantidad de memoria necesaria y evitando conflictos.
2. **Gestión de procesos:** El kernel administra los procesos en ejecución, asignándoles los recursos necesarios, programando su ejecución y asegurándose de que se ejecuten de manera segura y eficiente. También maneja la planificación de la CPU, decidiendo qué procesos se ejecutan y durante cuánto tiempo.
3. **Gestión de dispositivos:** El kernel proporciona una capa de abstracción para los dispositivos de hardware, permitiendo que las aplicaciones accedan a ellos de manera uniforme. Controla la comunicación y el acceso a los dispositivos, gestionando los controladores correspondientes.

4. **Gestión de archivos:** El kernel proporciona servicios para crear, leer, escribir y eliminar archivos en el sistema de almacenamiento. También maneja la organización y el acceso a los directorios y archivos del sistema.
5. **Seguridad y control de acceso:** El kernel controla el acceso a los recursos del sistema, aplicando políticas de seguridad y garantizando que las aplicaciones y los usuarios solo puedan acceder a los recursos permitidos.

El kernel es entonce la capa de software de más bajo nivel en la computadora.[apunte]

3.1. Ejecución Directa

Significa, correr el programa directamente en la CPU. Esto otorga la ventaja de tener rapidez. Aunque también esto viene con algunos problemas: se da control total al programa sin supervisión, abministrar la ejecucuión de programas, comienzo, fin, nuevo.

Debido a esto se necesita limitar la ejecución directa. Hoy en día esto ya no existe en un sistema operativo serio.

3.2. Limitar la Ejecución Directa

Para poder limitar la ejecución directa se necesitan ciertos mecanismos de hardware:

- **Dual Mode Operation** - Modo de operación dual.
- **Privileged Instructions** - Instrucciones Privilegiadas.
- **Memory Protection** - Proteccion de Memoria.
- **Timer Interrupts** - Interrupciones por temporizador.

Modo Dual de Operaciones

El modo de operación dual, es un mecanismo que proveen todas los procesadores a y algunos microprocesadores modernos. El hardware debe tener la capacidad de funcionar en dos modos diferentes: modo kernel y modo usuario. El modo kernel tiene privilegios más altos y acceso completo a todos los recursos del sistema, mientras que el modo usuario tiene acceso limitado y restringido a ciertos recursos. El sistema operativo se ejecuta en modo kernel, mientras que las aplicaciones de usuario se ejecutan en modo usuario.

Existen dos modos operacionales utilizados de la CPU:

- **Modo Usuario o User Mode:** que ejecuta instrucciones en nombre del usuario.
- **Modo Supervisor o Kernel o Monitor:** que ejecuta instrucciones en nombre del kernel del sistema operativo y estas son instrucciones privilegiadas.

3.3. Protección del Sistema:

¿Cual es el hardware necesario para que el kernel del sistema operativo pueda proteger a Usuarios y aplicaciones de otros usuarios?

Instrucciones Privilegiadas

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.

Protección de Memoria

El SO y los programas que están siendo ejecutados deben residir ambos en memoria al mismo tiempo (el SO debe cargar el programa, 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 (Interrupciones por temporizador)

Todos los procesadores poseen un dispositivo de hardware llamado Interrupciones de Tiempo que periódicamente le permita al kernel desalojar al proceso de usuario en ejecución y volver a tomar el control del procesador, y así de toda la máquina.

3.4. Visión General (Definiciones concretas)

Sistema Operativo: Es el software que controla los recursos de hardware de una computadora y provee un ambiente bajo el cual los programas pueden ejecutarse. Habitualmente a este software se lo llama el Kernel.

Gracias a la existencia del kernel los programas son independientes del hardware subyacente, es decir, se comunican con el kernel y no con el hardware directamente.

El kernel es entonces la capa de software de más bajo nivel en la computadora. El kernel es un programa.

3.5. Modos de Transferencia

Formas de alternar entre modo usuario y modo Kernel.

De Modo Usuario a Modo Kernel

- **interrupciones:** son eventos asíncronos que ocurren en el hardware o en el software que requieren la atención del kernel.

- **excepciones del procesador:** son eventos síncronos que ocurren en el procesador como resultado de la ejecución de una instrucción.
- **ejecución de system calls (llamadas al sistema):** son eventos síncronos que ocurren cuando una aplicación de usuario ejecuta una instrucción especial que le pide al kernel que realice una acción en su nombre.

Interrupciones

Una interrupción es una señal asincrónica enviada hacia el procesador de que algún evento externo ha sucedido y pueda requerir de la atención del mismo.

El procesador está continuamente chequeando si una interrupción es disparada. Cuando se dispara una interrupción, el procesador completa o detiene la instrucción que se esté ejecutando, guarda todo el contexto y comienza a ejecutar el manejador de esa interrupción en el Kernel.

El orden de prioridad de las interrupciones (según BCH) es:

- Errores de la Máquina.
- Timers.
- Discos.
- Network devices.
- Terminales.
- Interrupciones de Software.

Excepciones del Procesador

La otra forma por la cual se necesitaría pasar de modo usuario a modo kernel es por un evento de hardware causado por un programa de usuario. Por ejemplo, si un programa de usuario intenta dividir por cero, el procesador genera una excepción de división por cero. El procesador entonces pasa al modo kernel y ejecuta el manejador de la excepción de división por cero. Acceder fuera de la memoria del proceso, intentar escribir en memoria de solo lectura, ntentar ejecutar una instrucción privilegiada en modo usuario, etc.

System Calls

Las System Calls son funciones que permiten a los procesos de usuario pedirle al kernel que realice operaciones en su nombre. Las system calls son la interfaz entre el kernel y las aplicaciones de usuario.

3.6. System Calls

Una system call (llamada al sistema) es un punto de entrada controlado al kernel permitiendo a un proceso solicitar que el kernel realice alguna operación en su nombre [KER](cap. 3).

Algunas características generales de las system calls son:

- Una system call cambia el modo del procesador de user mode a kernel mode, por ende la CPU podrá acceder al área protegida del kernel.
- El conjunto de system calls es fijo. Cada system call esta identificada por un único número, que por supuesto no es visible al programa, este sólo conoce su nombre.
- Cada system call debe tener un conjunto de parámetros que especifican información que debe ser transferida desde el user space al kernel space.

Llamada a una System Call

Desde el punto de vista de un programa llamar a una system call es mas o menos como invocar a una función de C. Por supuesto, detrás de bambalinas muchas cosas suceden:

1. El programa realiza un llamado a una system call mediante la invocación de una función wrapper (envoltorio) en la biblioteca de C.
2. Dicha función wrapper tiene que proporcionar todos los argumentos al system call trap_handling. Estos argumentos son pasados al wraper por el stack, pero el kernel los espera en determinados registros. La función wraper copia estos valores a los registros.
3. Dado que todas las system calls son accedidas de la misma forma, el kernel tiene que saber identificarlas de alguna forma. Para poder hacer esto, la función wrapper copia el número de la system call a un determinado registro de la CPU.
4. La función wrapper ejecuta una instrucción de código maquina llamada trap machine instruction (int 0x80), esta causa que el procesador pase de user mode a kernel mode y ejecute el código apuntado por la dirección 0x80 (128) del vector de traps del sistema.
5. En respuesta al trap de la posición 128, el kernel invoca su propia función llamada syste_call() (arch/i386/entry.s) para manejar esa trap. Este manejador:
   a) graba el valor de los registros en el stack del kernel.
   b) verifica la validez del numero de system call.

   c) invoca el servicio correspondiente a la system call llamada a través del vector de system calls, el servico realiza su tarea y finalmete le devuelve un resultado de estado a la rutina system_call().
   d) se restauran los registros almacenado en el stack del kernel y se agrega el valor de retorno en el stack.
   e) se devuelve el control al wraper y simultáneamente se pasa a user mode.

6. Si el valor de retorno del la rutina de servicio de la system call da error, la función wrapper setea el valor en errno.

3.7. Tipos de Kernel

Existen básicamente dos tipos de estructuras de kernel:

- **Monolítico:** El kernel es un único programa (en realidad proceso) que se ejecuta continuamente en la memoria de la computadora intercambiándose con la ejecución de los procesos de usuario. Contiene todos los servicios del sistema operativo en el mismo espacio de direcciones.
- **Microkernel:** Es un kernel que se ejecuta en modo kernel y contiene solo los servicios esenciales del sistema operativo, como la gestión de memoria, la planificación de procesos y la comunicación entre procesos. Todos los demás servicios del sistema operativo se ejecutan como procesos de usuario fuera del kernel. El kernel proporciona una interfaz de programación de aplicaciones (API) para que los procesos de usuario puedan comunicarse con los servicios del kernel. El kernel microkernel es el tipo de kernel utilizado por los sistemas operativos macOS e iOS de Apple.
- **Híbrido:** Es un kernel que se ejecuta en modo kernel y contiene algunos servicios del sistema operativo en el mismo espacio de direcciones, mientras que otros servicios se ejecutan como procesos de usuario fuera del kernel. El kernel híbrido es el tipo de kernel utilizado por el sistema operativo Windows de Microsoft.

Existen básicamente dos tipos de estructuras de kernel:

4. Introducción: x86

4.1. Ley de Moore

En 1965 Gordon Moore, cofundador de Intel formuló una ley empírica que se ha podido constatar hasta nuestros días que dice:

"Aproximadamente cada dos años se duplica el número de transistores en un microprocesador por unidad de área"

4.2. Arquitectura x86: Hardware

Registros de Segmento

- **CS**: Segmento de código.
- **DS**: Segmento de datos.
- **SS**: Segmento de pila.
- **ES**: Segmento extra.

4.2.1. El Stack

El stack o pila es una estructura de datos que almacena información de forma temporal y ordenada, siguiendo el principio LIFO. El stack se usa para guardar los datos locales de una función, las direcciones de retorno de las llamadas a funciones y los parámetros que se pasan a las funciones. Más precisamente la ejecución de un programa se basa practicamente en pusher.

4.2.2. Estructura del Stack Frame

En la arquitectura x86 los programas utilizan el stack del programa para soportar la llamada a funciones (o procedimientos). La máquina utiliza el stack para:

- Almacenar los parámetros de la función.
- Almacenar las variables locales de la función.
- Almacenar el valor de retorno de la función.
- Almacenar los registros que se deben preservar.

5. El Proceso

5.1. De Programa a Proceso

```
1        #include <stdio.h>
2
3        int main() {
4            printf("hello, world\n");
5        }
6
```

Script 1: hola mundo.

La Compilación:

1. **La fase de procesamiento**. El preprocesador (cpp) sustituye las macros (#) y se eliminan los comentarios del código fuente. El resultado es un archivo de código C preprocesado con extensión .i .

2. **La fase de compilacion**. El compilador (cc) traduce el programa .i a un archivo de texto .s que contiene un programa en lenguaje assembly.

3. **La fase de ensablaje**. A continuación el ensamblador (as) traduce el archivo .s en instrucciones de lenguaje de máquina (binario) empaquetándolas en un formato conocido como programa objeto realocable. Este es almacenado en un archivo con extensión .o

4. **La fase de link edicion**. Generalmente los programas escritos en lenguaje C hacen uso de funciones que forman parte de la biblioteca estandar de C que es provista por cualquier compilador de ese lenguaje. Por ejemplo la función printf(), la misma se encuentra en un archivo objeto pre compilado que tiene que ser mezclado con el programa que se esta compilando, para ello el linker realiza esta tarea teniendo como resultado un archivo objeto ejecutable.

   Es decir se combinan los archivos objeto con las bibliotecas y dependencias necesarias para formar el archivo ejecutable final.

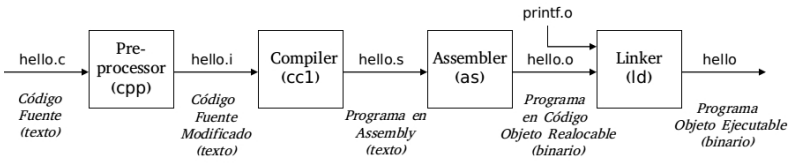

Figura 1: Proceso de compilacion.

5.2. Un Programa en Unix

Un programa es un archivo que posee toda la información de como construir un proceso en memoria [KER](cap. 6).

Un programa contiene:

1. **Formato de Identificación Binaria:** Cada archivo ejecutable posee META información describiendo el formato ejecutable. Esto permite al kernel interpretar la información contenida en el mismo archivo.

   Formatos en Unix:

   - **OUT** Assembler Output → Salida del compilador de C
   - **COFF** Common Object File Format → Utilizado en las versiones de System V compartidas, y símbolos de depuración en sistemas Unix.
   - **ELF** Executable and Linking Format → Utilizado en la actualidad

2. **Instrucciones de Lenguaje de Máquina:** Almacena el código del algoritmo del programa.

3. **Dirección del Punto de Entrada del Programa:** Identifica la dirección de la instrucción con la cual la ejecución del programa debe iniciar.
4. **Datos:** El programa contiene valores de los datos con los cuales se deben inicializar variables, valores de contantes y de literales utilizadas en el programa.
5. **Simbolos y Tablas de Realocación:** Describe la ubicación y los nombres de las funciones y variables de todo el programa, así como otra información que es utilizada por ejemplo para debugg.
6. **Bibliotecas Compartidas:** describe los nombres de las bibliotecas compartidas que son utilizadas por el programa en tiempo de ejecución así como también la ruta del linker dinámico que debe ser usado para cargar dicha biblioteca.
7. **Otra información:** El programa contiene además otra información necesaria para terminar de construir el proceso en memoria.

El Sistema Operativo más precisamente el Kernel se encarga de:

1. Cargar instrucciones y Datos de un programa ejecutable en memoria.
2. Crear el Stack y el Heap.
3. Transferir el Control al programa.
4. Proteger al SO y al Programa.

5.3. El Proceso

"Un proceso es la ejecución de un programa de aplicación con derechos restringidos; el proceso es la abstracción que provee el Kernel del sistema operativo para la ejecución protegida"- [DAH].

Un proceso incluye:

- Los Archivos abiertos.
- Las señales(signals) pendientes.
- Datos internos del kernel.
- El estado completo del procesador.
- Un espacio de direcciones de memoria.
- Uno o más hilos de Ejecución. Cada thread contiene
  - Un único contador de programa.
  - Un Stack.

- Un Conjunto de Registros.
- Una sección de datos globales

El proceso:

- Una "instancia" de un programa
- Tiene su propia memoria: código, datos, stack, heap
- Tiene un identificador único: PID.
- Tiene un conjunto de "archivos abiertos": Descriptores de Archivos.

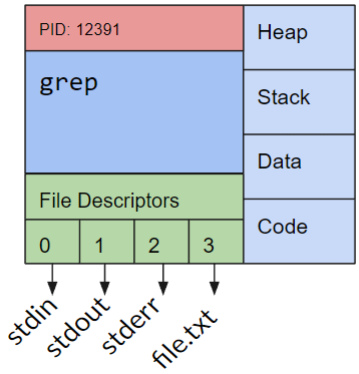

Figura 2: Proceso.

5.4. La Virtualización

Crear una abstracción que haga que un dispositivo de hardware sea mucho más fácil de utilizar.

- Virtualización de memoria.
- Virtualizacion de procesador.

Virtualización de Memoria

La virtualización de memoria le hace creer al proceso que este tiene toda la memoria disponible para ser reservada y usada como si este estuviera siendo ejecutado sólo en la computadora (ilusión).

Todos los procesos en Linux, está dividido en 4 segmentos:

- **Text Segment:** Contiene el código del programa.

- **Data:** Almacena las Variables Globales (extern o static en C).
- **Heap:** Memoria Dinámica Alocable.
- **Stack:** Almacena las Variables Locales y trace de llamadas.

Protección de Memoria Para que un proceso se ejecute tiene que estar residente en memoria, pero a su vez el sistema operativo tiene que estar residente en memoria.

- *El proceso tiene que estar en memoria para poder ejecutarse.*
- *El sistema operativo tiene que estar ahí para:*
  - *iniciar la ejecución del programa*
  - *manejar las interrupciones.*
  - *y/o atender las systems call..*

Es más, otros procesos podrían estar simultáneamente en memoria para poder compartir la memoria de forma segura, para ello el sistema operativo tiene que poder configurar el hardware de forma tal que cada proceso pueda leer y escribir solo su propia memoria (No la memoria del sistema operativo tampoco la de otros procesos. Ya que sino el proceso en cuestión podría incluso modificar al Kernel del sistema operativo. Para ello el Hardware debe proveer un mecanismo de protección de memoria, (que se verán detalladamente mas adelante).

Uno de estos mecanismos es denominado Memoria Virtual, la memoria virtual es una asbtracción por al cual la memoria física puede ser compartida por diversos procesos.

Un componente clave de la memoria virtual son las direcciones virtuales, con las direcciones virtuales, para cada proceso su memoria inicia en el mismo lugar, la dirección 0.

Cada proceso piensa que tiene toda la memoria de la computadora para si mismo, si bien obviamente esto en la realidad no sucede. El hardware traduce la dirección virtual a una dirección física de memoria.

Traducción de Direcciones Se traduce una Dirección Virtual (emitida por la CPU) en una Dirección Física (la memoria). Este mapeo se realiza por hardware, más específicamente por Memory Management Unit (MMU).

Virtualización de Procesador

La virtualización de procesamiento es la forma de virtualización más primitiva, consiste en dar la ilusión de la existencia de un único procesador para

Figura 3: mmu.

cualquier programa que requiera de su uso. De esta forma, se provee:

Simplicidad en la programación:

- Cada proceso cree que tiene toda la CPU.
- Cada proceso cree que todos los dispositivos le pertenecen.
- Distintos dispositivos parecen tener el mismo nivel de interfaces.
- Las interfaces con los dispositivos son más potentes que el bare metal.

Aislamiento frente a Fallas:

- Los procesos no pueden directamente afectar a otros procesos.
- Los errores no colapsan toda la máquina.

¿Cómo se provee la ilusión de tener varios CPUs?: El SO crea esta ilusión mediante la virtualización de la CPU a través del kernel.

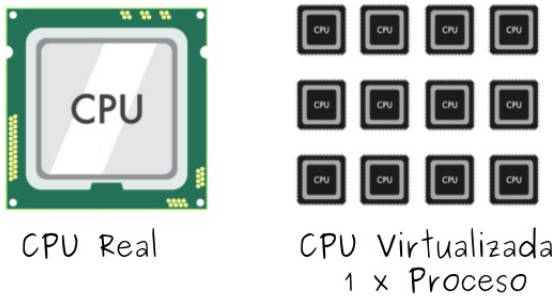

Figura 4: Virtualizacion Cpu.

Viéndolo desde el punto de vista de la abstracción y virtualización:

Entonces

"un proceso es básicamente una abstracción de un programa en ejecución."

Se ha de tener en cuenta que el Kernel en sí mismo también es un proceso y que la abstracción del proceso provee **ejecución, aislamiento y protección**. Estos tres conceptos pueden merecer varios capítulos de un libro. El sistema

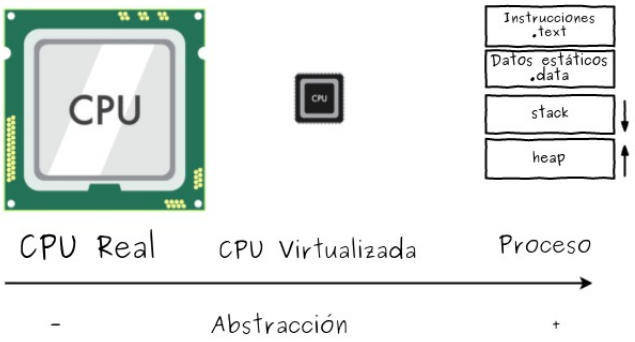

Figura 5: Virtualizacion Cpu.

operativo lleva la contabilidad de todos los procesos que se están ejecutando en la computadora mediante la utilización de una estructura llamada Process Control Block o PCB. La PCB almacena toda la información que un sistema operativo debe conocer sobre un proceso en particular:

- Donde se encuentra almacenado en memoria.
- Donde la imagen ejecutable esta en el disco.
- Que usuario solicito su ejecución.
- Que privilegios tiene ese proceso.

5.5. El Proceso: por dentro

La idea detras de la abstracción es como virtualizar una CPU, es decir, como hacer para que una única procesador pueda ejecutar múltiples procesos.

Un Proceso necesita permisos del Kernel del SO para:

- Acceder a memoria perteneciente a otro proceso.
- Antes de escribir o leer en el disco.
- Antes de cambiar algún seteo del hardware del equipo.
- Antes de enviar información a otro proceso.

5.6. El API de Procesos

Que debe incluir cualquier interfaz de un SO:

- Creación (Create): todo sistema operativo debe incluir una forma de crear un nuevo proceso.

- Destrucción(Destroy): así como existe una interface para crear un proceso debe existir una interface para destruirlo por la fuerza.
- Espera (wait): A veces es útil esperar a que un proceso termine su ejecución por ende algún tipo de interface de espera debe ser provista.
- Control Vario (Miscellaneous Control): Además de esperar o matar a un proceso otros tipos de operaciones deben poder realizarse. Por ejemplo, suspender su ejecución por un tiempo y luego reanudarla.
- Estado (Status) : Tiene que existir una forma de saber sobre la situación del proceso y su estado. Cuánto hace que se está ejecutando, en que estado se encuentra, etc.

Estas son las acciones básicas que todo SO debe proveer sobre la **abstracción de la CPU.**

6. Scheduling o Planificación de Procesos

6.1. Scheduling o Planificación de Procesos

time slice o time quantum: período de tiempo que el kernel le otorga a un proceso. Para S.O de tipo Time Sharing.

6.2. Time Sharing

Los Sistemas Operativos llamados **time sharing** surgen de la idea de los programadores de tener *"toda una computadora para uno mismo"*.

Planificador o Scheduler: Es un componente del sistema operativo que se encarga de decidir qué proceso se ejecuta en cada momento.

6.3. Números y el Workload

El **Workload** es carga de trabajo de un proceso corriendo en el sistema.

6.4. Métricas de Planificación

políticas de planificación o scheduling

1. **Rurnaround time:** tiempo en el cual el proceso se completa menos el tiempo de arribo al sistema:

$$T_{Turnaround} = T_{CompletionTime} - T_{ArrivalTime} \tag{1}$$

2. **Response time:** tiempo que tarda el sistema en responder a una solicitud:

$$T_{Response} = T_{FirstResponse} - T_{ArrivalTime} \tag{2}$$

6.5. Políticas Para Sistemas Mono-procesador

Se estudiarán las políticas de planificación para un sistema que posea un solo procesador o CPU con un solo núcleo de procesamiento.

1. **First In, First Out (FIFO)**
2. **Shortest Job First (SJF)**
3. **Shortest Time-to-Completion (STCF)**
4. **Round Robin (RR)**
5. **Multi-Level Feedback Queue (MLFQ)**

First In, First Out (FIFO)

Este algoritmo asigna la CPU al proceso que llegó primero y lo ejecuta hasta que termina o se bloquea. Luego, el siguiente proceso en la cola es seleccionado y se le asigna la CPU.

Shortest Job First (SJF)

Este algoritmo asigna la CPU al proceso que tiene el menor tiempo de ejecución. Se ejecuta el proceso de duración mínima, una vez finalizado esto se ejecuta el proceso de duración mínima y así sucesivamente.

Shortest Time-to-Completion (STCF)

En este caso el planificador se adelanta y si hay un proceso que puede terminar antes, ejecuta ese primero, posponiendo la ejecución del que se estaba ejecutando. Esto lo realiza con un context. switch. Es pre-empitve.

Round Robin (RR)

La idea del algoritmo es bastante simple, se ejecuta un proceso por un período determinado de tiempo **(slice)** y transcurrido el período se pasa a otro proceso, y así sucesivamente cambiando de proceso en la cola de ejecución [Round Robin Paper].

Con este método de planificación, la métrica de *response* mejora, pero la de *turnaround* empeora, ya que se atrasa el retorno de todos los programas.

6.6. Multi-Level Feedback Queue (MLFQ)

Multi-Level Feedback Queue (MLFQ) es un algoritmo de planificación de procesos en sistemas operativos. Este algoritmo utiliza múltiples colas de prioridad para asignar la CPU a los procesos. Los procesos se colocan en la cola de prioridad más baja al llegar y se ejecutan hasta que se bloquean o terminan. Si un proceso no se completa en la cola de prioridad más baja, se mueve a la

siguiente cola de prioridad más alta.

MLQF intenta atacar principalmente 2 problemas:

- Intenta optimizar el turnaround time mediante la ejecución de la tarea mas corta primero.(No sabemos cual es)
- Intenta hacer que el sistema tenga un tiempo de respuesta interactivo para los usuarios.(Minimizar el response time)

6.7. MLQF: Las reglas básicas

Las 2 reglas básicas de MLFQ:

- **REGLA 1:** si la prioridad (A) es mayor que la prioridad de (B), (A) se ejecuta y (B) no.
- **REGLA 2:** si la prioridad de (A) es igual a la prioridad de (B), (A) y (B)se ejecutan en Round-Robin.
- **REGLA 3:** Cuando un proceso entra en el sistema, se le asigna la prioridad más alta.
- **REGLA 4:** Una vez que una tarea usa su asignación de tiempo en un nivel dado (independientemente de cuantas veces haya renunciado al uso de la CPU) su prioridad se reduce: (Por ejemplo baja un nivel en la cola de prioridad)
- **REGLA 4a:** Si una tarea usa un time slice mientras se esta ejecutando su prioridad se reduce de una unidad (baja la cola una unidad menor)
- **REGLA 4b:** Si una tarea renuncia al uso de la CPU antes de un *time slice* completo se queda en el mismo nivel de prioridad.
- **REGLA 5:** Después de cierto periodo de tiempo S, se mueven las tareas a la cola con mas prioridad.

La clave para la planificación MLFQ subyace entonces en cómo el planificador setea las prioridades. En vez de dar una prioridad fija a cada tarea, MLFQ varia la prioridad de la tarea basándose en su comportamiento observado REGLA 3, 4a, 4b.

PROBLEMA Con este Approach de MLFQ **Starvation o indefiniciónes**: Si una tarea es muy larga, puede que nunca llegue a ejecutarse. Para solucionar esto se utiliza la REGLA 5.

6.8. Planificación: Proportional Share

La planificación por proporcional share es un tipo de planificación de procesos que trata de garantizar que cada proceso obtenga un cierto porcentaje de tiempo de CPU según su prioridad. Una forma de implementar este tipo de planificación es mediante el lottery scheduling, que consiste en realizar sorteos periódicos para determinar qué proceso debe ejecutarse a continuación; los procesos que deben ejecutarse más a menudo deben tener más posibilidades de ganar el sorteo.

La ventaja de este método es que utiliza la aleatoriedad para evitar casos extremos, reducir el estado necesario y acelerar la decisión. La desventaja es que no garantiza un reparto exacto del tiempo de CPU, sino solo probabilístico.

7. La Memoria

7.1. La Abstracción del Espacio de Direcciones: Introducción

La memoria física puede ser imaginada como un arreglo de direcciones de memoria una detrás de otra.
Esta "abstracción" es manejable mientras la cantidad de memoria necesitada esta en un rango "manejable" y cual es ese rango. En DOS (Disk Operating System) ese rango lo daba la elección de una arquitectura determinada, la del 8086, que permite tener direcciones de memoria de hasta 220 bits, es decir 1 MegaByte de memoria.

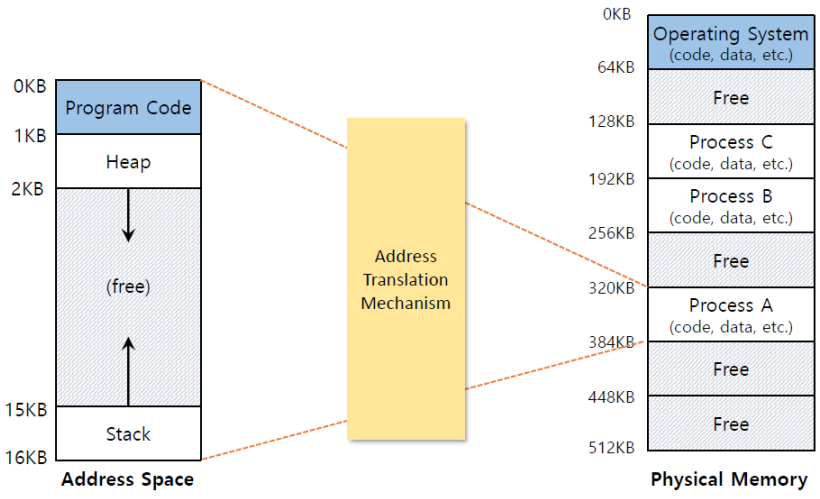

Figura 6: Address traslation.

7.2. El Espacio de Direcciones o Address Space

El Address Space de un proceso contiene todo el estado de la memoria de un programa en ejecución. Es la abstracción para la memoria.

El Espacio de Direcciones o Address Space es la abstracción fundamental sobre la memoria de una computadora. Consiste en dar un mecanismo fácil de usar a lo usuarios de la computadora.

Cuando se describe el espacio de direcciones se está describiendo la abstracción que el Sistema Operativo le proporciona al programa en ejecución sobre la memoria de la computadora.

Cuando el Sistema Operativo implementa esta abstracción, se dice que el O.S. está Virtualizando la Memoria ya que el programa en ejecución cree que está cargado en un lugar particular de la memoria (la posición 0 dirección virtual o virtual address) y tiene potencialmente toda la memoria para él.

Metas principales de la virtualización es :

- transparencia.
- eficiencia: tiempo y espacio.
- protección: proteger a los procesos unos de otros como también proteger al sistema operativo de los procesos.
    - aislamiento: cada proceso tiene su propio espacio de direcciones aislado.

7.3. El API de Memoria

7.3.1. Tipos de Memoria

- **Memoria de stack:** su reserva y liberación es manejada implícitamente por el compilador en nombre del programador por esta razón a veces también se denomina memoria automática.
- **Memoria de heap:** es la memoria que se reserva y libera explícitamente por el programador.

7.4. Address Translation

Existen dos puntos importantes a la hora de virtualizar memoria:

- flexibilidad
- eficiencia

Para lograr esto se usa la técnica de **address translation** o traducción de direcciones (Hardware-Based Address Translation).

7.5. Hacia una eficiente Address Translation

Mecanismos para mejorar el rendimiento de la traducción de las direcciones.

Se usara un **caché (o escondrijo)**, que consiste en una copia de ciertos datos que pueden ser accedidos mas de una vez más rápidamente.

Uno de los problemas del **address translation** reside en la **velocidad de la traducción** para ello se utilizan técnicas que mejoran la velocidad de esta traducción. Se utiliza un mecanismo de hardware llamado **Translation-Lookaside Buffer**; o tambien conocido como **TLB**. La TLB es parte de la MMU y es simplemente un mecanismo de cache de las traducciones mas utilizadas entre los pares virtual to physical address. Por ende un mejor nombre para este mecanismo podría ser address translation cache.

Por cada referencia a la memoria virtual, el hardware primero chequea la TLB para ver si esa traducción esta guardada ahí; si es así la traducción se hace rápidamente sin tener que consultar a la page table (la cual tiene todas las traducciones).

Normalmente se chequean todas las entradas de la TLB contra la virtual page, si existe matcheo el procesador utiliza ese matcheo para formar la physical address, ahorrándose todos los pasos de la traducción.
Cuand existe matcheo TLB hit, cuando del proceso anterior no existe matcheo en la TLB , se dice que se tiene un TLB miss

8. Concurrencia

Concurrencia cuando hay una existencia simultánea de múltiples procesos/hilos en ejecución.

8.1. La Abstracción

Un thread es una secuencia de ejecución atómica que representa una tarea planificable de ejecución.

- **Secuencia de ejecución atómica:** Cada thread ejecuta una secuencia de instrucciones como lo hace un bloque de código en el modelo de programación secuencial.
- **Tarea planificable de ejecución:** El sistema operativo tiene injerencia sobre el mismo en cualquier momento y puede ejecutarlo, suspenderlo y continuarlo cuando él desee.

Threads vs procesos

Proceso: un programa en ejecución con derechos restringidos.

thread: una secuencia independiente de instrucciones ejecutándose dentro de un programa.

Thread Scheduler

El Thread Scheduler (planificador de hilos) es una parte fundamental del sistema operativo encargada de administrar y controlar la ejecución de los hilos de ejecución en un entorno multitarea. Su función principal es asignar el tiempo de CPU disponible a los diferentes hilos de manera equitativa y eficiente, maximizando el rendimiento del sistema.

El Thread Scheduler toma decisiones sobre qué hilo debe ejecutarse en un momento dado, determinando el orden y la duración de la ejecución de los hilos en función de ciertas políticas de planificación.
En la actualidad hay dos formas de que los threads se relacionen entre sí:

- **Multi-threading Cooperativo:** Los threads se comunican entre sí y cooperan para realizar una tarea. No hay interrupción a menos que se solicite.
- **Multi-threading Preemptivo:** Consiste en que un threads en estado de running puede ser movido en cualquier momento.

8.2. Estructura y Ciclo de Vida de un Thread

El S.O. provee la ilusión de que cada uno de estos threads se ejecutan en su propio procesador, haciendo de forma transparente que se ejecuten o paren su ejecución.
Para que la ilusión sea creíble, el sistema operativo debe guardar y cargar el **estado** de cada thread. Como cualquier thread puede correr en el procesador o en el kernel, también debe haber estados compartidos, que no deberían cambiar entre los modos.

Para poder entender la abstracción hay que comprender que existen dos estados:

- El estado per thread.
- El estado compartido entre varios threads.

8.2.1. El Estado Per-thread y Threads Control Block (TCB)

Thread Control Block (TCB) estructura que representa el estado de un thread. La TCB almacena el estado per-thread de un thread:
Para poder crear múltiples threads y pararlos y rearrancarlos, el S.O. debe poder almacenar en la TCB el estado actual del bloque de ejecución:

- El puntero al stack del thread.
- Una copia de sus registros en el procesador.

8.2.2. Metadata referente al thread que es utilizada para su administración

8.3. Sincronización

La sincronización se refiere a la coordinación y control de la ejecución de múltiples hilos (Thread) para garantizar que se realicen correctamente las operaciones compartidas y evitar problemas como las condiciones de carrera y la inconsistencia de los datos.

8.3.1. Race Conditions

Una race condition se da cuando el resultado de un programa depende en como se intercalaron las operaciones de los threads que se ejecutan dentro de ese proceso.

Una Race Condition (condición de carrera) es un fenómeno indeseable que ocurre cuando dos o más hilos o procesos acceden y manipulan simultáneamente un recurso compartido sin una sincronización adecuada. El resultado de una Race Condition es impredecible y puede llevar a resultados incorrectos o inesperados en el programa.

Las Race Conditions ocurren cuando la ejecución de múltiples hilos o procesos no está coordinada correctamente y depende del orden o tiempo exacto en el que se realicen las operaciones. Esto puede suceder cuando los hilos comparten datos y realizan operaciones de lectura y escritura en esos datos sin una protección adecuada.

Por ejemplo, considera dos hilos que comparten una variable numérica y realizan la siguiente secuencia de operaciones:

- Hilo 1: Lee el valor actual de la variable.
- Hilo 2: Lee el valor actual de la variable.
- Hilo 1: Incrementa el valor leído en 1.
- Hilo 2: Incrementa el valor leído en 1.
- Hilo 1: Escribe el nuevo valor en la variable.
- Hilo 2: Escribe el nuevo valor en la variable.

Si ambos hilos ejecutan estas operaciones de manera simultánea y sin sincronización, puede ocurrir una Race Condition. Dependiendo del orden de ejecución de los hilos y de las operaciones individuales, el resultado final puede ser inconsistente y no determinista. Por ejemplo, el resultado podría ser que ambos hilos incrementen el valor en 1 y sobrescriban los cambios del otro, o podrían basarse en valores desactualizados, entre otras posibilidades.

Las Race Conditions pueden ser difíciles de detectar y depurar, ya que su comportamiento puede ser intermitente y depender de factores como la velocidad relativa de los hilos, la planificación del sistema operativo y otros eventos concurrentes. Para evitar las Race Conditions, se utilizan mecanismos de sincronización, como bloqueos, semáforos, mutex, entre otros, que aseguran el acceso exclusivo a los recursos compartidos y garantizan la consistencia y la integridad de los datos.

8.4. Operaciones Atómicas

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

El "Problema de la Heladera Llena" es un escenario abstracto que ilustra un desafío común en la programación concurrente, que consiste en coordinar el acceso a un recurso compartido entre múltiples hilos o procesos.

Heisenbug es un error no determinístico, el nombre se puso en honor al físico Heisenberg. Este tipo de error desaparece cuando uno quiere debuggearlo ya que depende de condiciones como el del intercalado del scheduler.

El término **interleaves** se utiliza en el contexto de la programación concurrente para referirse a la forma en que las instrucciones o acciones de múltiples hilos o procesos se entrelazan o se ejecutan en un orden no determinista.

8.5. Una Mejor Solución Locks

Un forma menos compleja de alcanzar una solución para el problema de la heladera es mediante la utilización de locks. Un **lock** es una variable que permite la sincronización mediante la exclusión mutua, cuando un thread tiene el candado o lock ningún otro puede tenerlo.

La idea principal es que un proceso asocia un lock a determinados estados o partes de código y requiere que el thread posea el lock para entrar en ese estado. Con esto se logra que sólo un thread acceda a un recurso compartido a la vez.

Esto permite la exclusión mutua, todo lo que se ejecuta en la región de código en la cual un thread tiene un lock, garantiza la atomicidad de las operaciones.

8.6. Errores comunes de concurrencia

En concurrencia el concepto de **dead lock** aparece cuando entre dos o más threads uno obtiene el lock y por algún motivo nunca libera el mismo haciendo que sus compañeros se bloqueen. Según Dahlin un deadlock es un ciclo de espera a través de un conjunto de threads, en el cual cada thread espera que algún otro thread en el ciclo tome alguna acción.

9. Preguntas y Respuestas

9.1. El kernel

Ejercicio 9.1. *¿Qué es el Kelnel?*

El kernel es la parte central del sistema operativo que gestiona los recursos del hardware y las interacciones con el software.

El kernel es responsable de ejecutar los programas, administrar la memoria, controlar los dispositivos de entrada y salida, y proporcionar seguridad y estabilidad al sistema.

Ejercicio 9.2. *¿Que es el ejecución directa?*

Significa ejectar un programa directamente en la CPU. Beneficios: rapidez, Problemas: seguridad, confiabilidad, portabilidad.

Limitar la ejecución directa:

- **Modo de operación dual:** es un mecanismo que proveen todas los procesadores, intercambia entre user mode y kernel mode.
- **Instrucciones Privilegiadas:** set de instrucciones que poseen cada modo de operación.
- **Proteccion de Memoria:** como la memoria es compartida, el SO debe poder configurar el hardware de forma tal que cada proceso pueda leer y escribir su propia porción de memoria.
- **Interrupciones por temporizador:** mecanismo le permita al kernel desalojar al proceso y volver a tomar el control del procesador.

9.2. El proceso

Ejercicio 9.3. *Describa que es un proceso: qué abstrae, cómo lo hace, cuál es su estructura. Además explique el mecanismo por el cual el proceso cree tener la memoria completa de la máquina cuando en realidad solo tiene lo necesario para su funcionamiento.*

Un proceso es **la ejecución de un programa** de aplicación con derechos restringidos; el proceso es la abstracción que provee el Kernel del sistema operativo para la ejecución protegida.

Cuando se habla de derechos restringidos se está diciendo que está corriendo en una máquina donde hay dual mode, donde hay un kernel que tiene modo kernel y un set de instrucciones privilegiadas y este pobre tipo

corre en el ring 3 con nada de privilegios.

Cuando hace referencia a la abstracción se refiere a la virtualización del procesador (CPU), memoria y dispositivos de entrada/salida. Esta abstracción permite que varios procesos se ejecuten simultáneamente en la misma máquina, compartiendo los recursos físicos subyacentes de manera segura y asilada.

La abstracción del proceso provee ejecución, aislamiento y protección.

El mecanismo es la virtualización de memoria, que es una abstracción por la cual la memoria física puede ser compartida por diversos procesos. El sistema operativo asigna una cantidad limitada de memoria física a cada proceso, pero el proceso cree que tiene acceso a toda la memoria de la máquina. Cada proceso tiene su propio espacio de direcciones virtuales que es mapeado a la memoria física subyacente por el sistema operativo. La virtualización de memoria se logra mediante el uso de paginación y segmentación.

Ejercicio 9.4. *Cuál/cuáles mecanismos utiliza el kernel para garantizar el aislamiento entre procesos. Estos mecanismos están relacionados con el hardware, porque deben existir y donde se ve su funcionamiento.*

El kernel utiliza la virtualización de memoria para garantizar el aislamiento entre procesos.

Ejercicio 9.5. *¿Que es la virtuaizacion?*

Es crear una abstracción que haga que un dispositivo de hardware sea mucho más fácil de utilizar. Existen dos tipos de virtualización:

- **Virtualización de memoria:** Le hace creer al proceso que este tiene toda la memoria disponible.
  - Protección de Memoria: Memoria Virtual.
  - Traducción de Direcciones.
- **Virtualizacion de procesador:** Consiste en dar la ilusión de la existencia de un único procesador para cualquier programa que requiera de su uso.

  De esta forma, se prove:
  - Simplicidad en la programación.
  - Aislamiento frente a Fallas.

Ejercicio 9.6. *¿Cuales son los mecanismos de protección de memoria?*

La memoria virtual es una asbtracción por al cual la memoria física puede ser compartida por diversos procesos.

Un componente clave de la memoria virtual son las direcciones virtuales, con las direcciones virtuales, para cada proceso su memoria inicia en el mismo lugar, la dirección 0.

El hardware traduce la dirección virtual a una dirección física de memoria, se realiza por hardware (MMU).

Ejercicio 9.7. *¿Que es el address space?¿Que partes tiene?¿Para qué sirve?. Describa el/los mecanismos para crear un proceso en unix, sus sycalls, ejemplifique.*

El address space es el espacio de direcciones virtuales que un proceso puede utilizar. Está dividido en varias áreas: text, data, stack y heap. El propósito del address space es mantener separados los procesos y evitar que un proceso escriba en los datos de otro proceso.

Para la creacion de un proceso:

- única forma es llamando a la system call *fork*.

Ejercicio 9.8. *¿Que es el stack ? Explique el mecanismo de funcionamiento del stack para x86 de la siguiente funcion int read(void *buff, size_t num, int fd);.Como se pasan los parametros, direccion de retorno?.*

El stack o pila es una estructura de datos que almacena información de forma temporal y ordenada, siguiendo el principio LIFO (Last In, First Out), es decir, el último en entrar es el primero en salir. El stack se usa para guardar los datos locales de una función, las direcciones de retorno de las llamadas a funciones y los parámetros que se pasan a las funciones.

Para la función read(void *buff, size_t num, int fd), que lee num bytes del archivo identificado por fd y los almacena en el buffer buff, se puede usar el stack para pasar los parámetros de la siguiente manera:

- Se empujan los parámetros al stack en orden inverso, es decir, primero fd, luego num y finalmente buff.
- Se llama a la función read con la instrucción call, que empuja la dirección de retorno al stack y salta a la etiqueta de la función.

- Dentro de la función read, se accede a los parámetros usando el registro ebp (base pointer) como referencia. El registro ebp se usa para guardar el valor del registro esp (stack pointer) al entrar en la función, y así poder acceder a los parámetros y variables locales sin importar cómo cambie el esp durante la ejecución de la función
- Se usa la convención cdecl para limpiar el stack después de la llamada a la función. Esta convención establece que el código que llama a la función es responsable de restaurar el esp al valor que tenía antes de empujar los parámetros. Esto se hace sumando al esp el tamaño total de los parámetros.

Un posible código en ensamblador x86 para este ejemplo sería:
; Código que llama a la función read ; Supongamos que fd = 3 (stdin), num = 100 y buff apunta a una zona de memoria reservada mov eax, 3 ; fd push eax ; empujar fd al stack mov eax, 100 ; num push eax ; empujar num al stack mov eax, buff ; buff push eax ; empujar buff al stack call read ; llamar a la función read add esp, 12 ; limpiar el stack (3 parámetros de 4 bytes cada uno)
; Código de la función read read: push ebp ; guardar el valor anterior de ebp mov ebp, esp ; copiar el valor de esp a ebp ; Ahora los parámetros se pueden acceder como [ebp+8], [ebp+12] y [ebp+16] ; Aquí iría el código para leer del archivo y escribir en el buffer ; usando las instrucciones syscall o int 80h mov esp, ebp ; restaurar el valor de esp pop ebp ; restaurar el valor de ebp ret ; retornar a la dirección guardada en el stack

Ejercicio 9.9.

9.3. La Memoria

Ejercicio 9.10. *¿Que es la memoria virtual? ¿Qué mecanismos conoce, describa los tres que a usted le parezcan más relevantes?*

La Memoria Virtual es un mecanismo de protección de memoria, provisto por el Hardware. La memoria virtual es una asbtracción por al cual la memoria física puede ser compartida por diversos procesos.

1. *¿Que es la memoria virtual? ¿Qué mecanismos conoce, describa los tres que a usted le parezcan más relevantes?*

   - **La memoria segmentada** es una técnica de gestión de memoria que divide el espacio de memoria de un proceso en segmentos lógicos más pequeños y coherentes, en lugar de tratarlo como un espacio de memoria continuo y uniforme. Cada segmento representa una porción

lógica de la memoria y puede contener diferentes tipos de datos, como código, datos, pila, tabla de símbolos, etc.

Cada segmento tiene un tamaño (Bound o registro límite o Segmento) y una dirección base (registro base) asociada. La dirección base indica la ubicación física donde comienza el segmento en la memoria física, mientras que el tamaño representa la longitud del segmento. En lugar de utilizar direcciones absolutas, se utilizan direcciones relativas dentro de cada segmento.

La memoria segmentada ofrece varias ventajas. Permite una mayor flexibilidad en la asignación y el uso de memoria, ya que los segmentos pueden crecer o contraerse dinámicamente según las necesidades del proceso. También facilita el compartimiento de memoria entre diferentes procesos, ya que es posible compartir segmentos comunes entre ellos, lo que puede ahorrar espacio y mejorar la eficiencia.

Sin embargo, la segmentación también puede presentar desafíos, como la fragmentación externa, que ocurre cuando hay espacios vacíos entre segmentos que no se pueden utilizar para almacenar otros segmentos. Esto puede llevar a un desperdicio de memoria. Además, la gestión de los segmentos y la traducción de direcciones pueden requerir una mayor complejidad en el hardware y el sistema operativo.

En resumen, la memoria segmentada es una técnica de gestión de memoria que divide el espacio de memoria de un proceso en segmentos lógicos, lo que proporciona flexibilidad y compartición de memoria, pero puede implicar desafíos como la fragmentación externa.

- **La memoria paginada** es una técnica de gestión de memoria en la que la memoria se divide en fragmentos de tamaño fijo llamados "page frames". En lugar de dividir la memoria en segmentos lógicos, como en la memoria segmentada, la memoria paginada la divide en páginas de tamaño uniforme. Cada página tiene un número de página virtual y una dirección física correspondiente.

  El mecanismo de traducción de direcciones en la memoria paginada es similar al de la memoria segmentada. Cada proceso tiene una tabla de páginas (page table) que contiene entradas que mapean las páginas virtuales a las direcciones físicas de los page frames en la memoria física. Cuando un proceso accede a una dirección virtual, se utiliza la tabla de páginas para obtener la dirección física correspondiente.

  La dirección virtual consta de dos componentes: el número de página virtual y el desplazamiento (*offset*) dentro de esa página. El número de página virtual se utiliza como índice en la tabla de páginas para obtener la dirección física del page frame correspondiente. Luego, se concatena el desplazamiento para obtener la dirección física completa.

  La memoria paginada ofrece varias ventajas, como una mayor eficiencia en la gestión de la memoria y la capacidad de compartir páginas entre procesos, lo que permite la memoria compartida. También faci-

lita la protección de la memoria, ya que cada página se puede asignar permisos individuales de lectura, escritura y ejecución.
Un aspecto importante de la memoria paginada es que proporciona una vista lógica de la memoria lineal para cada proceso, aunque las páginas pueden estar dispersas por toda la memoria física. Esto significa que las direcciones virtuales son continuas y lineales para el proceso, aunque las páginas físicas pueden estar ubicadas en diferentes ubicaciones físicas.
En sistemas de paginación multinivel, como el utilizado en la arquitectura x86, se pueden utilizar múltiples niveles de tablas de páginas para gestionar direcciones virtuales más grandes de manera eficiente. Esto permite una mayor flexibilidad y eficiencia en la gestión de la memoria.
En resumen, la memoria paginada es una técnica de gestión de memoria en la que la memoria se divide en páginas de tamaño fijo y se utiliza una tabla de páginas para traducir direcciones virtuales a direcciones físicas. Proporciona una vista lógica de la memoria lineal para cada proceso y ofrece ventajas como una gestión eficiente de la memoria y la capacidad de compartir páginas entre procesos.

- **Paged Segmentation (Segmentación paginada)** es una combinación de la segmentación y la paginación. Consiste en dividir el espacio de direcciones lógicas en segmentos de tamaño variable, y luego dividir cada segmento en páginas de tamaño fijo. Cada segmento tiene una tabla de páginas asociada, que se almacena en una tabla de segmentos. El proceso de traducción de las direcciones lógicas a físicas es, primero se busca el segmento en la tabla de segmentos, luego se busca la página en la tabla de páginas del segmento, y finalmente concatena el frame de la oage table con el offset para obtener la dirección física completa.
  - Reduce la fragmentación externa.
  - Mejora el rendimiento.
  - Proporciona un buen equilibrio entre flexibilidad y rendimiento

2.

Ejercicio 9.11. *Dado un sistema de paginación de 2 niveles de indirección, en el cual una v.a tiene un longitud de 32 bits, 10 bits para el PDI, 10 bits para el page table y finalmente 12 bits ara el offset. Indicar la cantidad de memoria en bytes máximas que pueden ocupar un proceso. justifique*

En un sistema de paginación de dos niveles de indirección con una dirección virtual de 32 bits

- 10 bits para el índice del directorio de páginas (PDI)

- 10 bits para la tabla de páginas
- 12 bits para el desplazamiento

La cantidad máxima de memoria que puede ocupar un proceso se calcula de la siguiente manera:

Primero, determinamos el tamaño de la página, que está dado por el desplazamiento. Como se utilizan 12 bits para el desplazamiento, el tamaño de la página es de 2^12 bytes, o 4KB.

Luego, determinamos el número total de entradas en la tabla de páginas, que está dado por el índice de la tabla de páginas. Como se utilizan 10 bits para el índice de la tabla de páginas (segment), hay 2^10, o 1024, entradas en la tabla de páginas.

Finalmente, determinamos el número total de tablas de páginas, que está dado por el índice del directorio de páginas. Como se utilizan 10 bits para el índice del directorio de páginas, hay 2^10, o 1024, tablas de páginas.

Por lo tanto, la cantidad máxima de memoria que puede ocupar un proceso es el tamaño de la página multiplicado por el número total de entradas en la tabla de páginas multiplicado por el número total de tablas de páginas. Esto es, 4 KB * 1024 * 1024, o 4 GB. Por lo tanto, un proceso puede ocupar un máximo de 4 GB de memoria en este sistema de paginación.

Convertir 2^{32} a gigabytes (GB):

Cuando decimos 2^{32}, estamos utilizando notación exponencial. En este caso, 2^{32} significa "2 elevado a la potencia de 32". Para convertir esto a bytes, recordamos que 1 byte es igual a 2^0, 1 kilobyte (KB) es igual a 2^{10} bytes, 1 megabyte (MB) es igual a 2^{20} bytes, y 1 gigabyte (GB) es igual a 2^{30} bytes.

Entonces, para convertir 2^{32} a gigabytes, dividimos 2^{32} por 2^{30}:

$$\frac{2^{32}}{2^{30}} = 2^{32-30} = 2^2 = 4$$

Por lo tanto, 2^{32} bytes es igual a 4 gigabytes. En el contexto de la memoria de un sistema informático, esta es la razón por la cual a menudo escuchamos que un sistema de 32 bits puede direccionar hasta 4 GB de memoria.

9.4. Concurrencia

1. **¿Que es un thread?. Use su API para crear un programa que use 5 thread para incrementar una variable compartida por todos en 7 unidades/thread hasta llegar a 100**

   Un thread es una secuencia de ejecución atómica que representa una tarea planificable de ejecución. También una secuencia independiente de instrucciones ejecutándose dentro de un programa.

```
1 #include <stdio.h>
```

```
2
3                   int main() {
4                   printf("hello, world\n");
5                   }
6
```

Script 2: hola mundo.

2.

9.5. File System

Ejercicio 9.12. El superbloque de un sistema de archivos indica que el inodo correspondiente al directorio raiz es el #43. En la siguiente secuencia de comandos y siempre partiendo de ese directorio raiz, se pide indicar la cantidad de inodos y bloquies de datos a los que se precisa acceder (leer) para resolver la ruta dada a cat(1) o stat(1)

```
# mkdir /dir /dir/s /dir/s/w
# touch /dir/x /dir/s/y
# stat /dir/s/w/x    // Inodos: ......    Bloque Datos: ......
# stat /dir/s/y      // Inodos: ......    Bloque Datos: ......

# ln /dir/s/x  /dir/h
# ln -s /dir/s/y /dir/y
# cat /dir/h         // Inodos: ......    Bloque Datos: ......
# cat /dir/y         // Inodos: ......    Bloque Datos: ......

```

Script 3: hola mundo.

Ayuda: todos los directorios ocupan un bloque. La idea es que describan como stat llega a los archivos.

Respuesta 9.1. Ver la Figura 9.5, también se puede ver la carpeta "markodown-chatGPT".

Como dice desde el directorio raíz.

- **stat /dir/s/y:** Entonces se accede al inode del directorio "dir"(1 inode) se accede a su bloque (1 bloque) que contiene el directorio "s", se accede al inode del directorio "s"(1 inode) se accede a su bloque (1 bloque), que contiene el directorio "w", se accede al inode del directorio "w"(1 inode) se accede a su bloque (1 bloque), está vacío porque no se creó ningún archivo. Entonces: 3 inodes y 3 bloques
- **stat /dir/s/y:** Para este caso se da lo mismo que el anterior, pero en vez de acceder al inode del directorio "w"se accede al inode del archivo z"(1 inode) se accede a su bloque (1 bloque). Entonces: 3 inodes y 3 bloques.
- **cat /dir/h:** Suponiendo que el comando "ln /dir/s/x /dir/h"no falle, sería 1 inode y 1 bloque. Pero como falla, no se crea el hardlink, entonces no se accede a ningún inode ni bloque.
- **cat /dir/y:** Sería 2 inode y 2 bloque. Por que es un symlink. Se accede al inode del directorio "dir"(1 inode) se accede a su bloque (1 bloque) que contiene el symlink z", se accede al inode de z"que esta en el directorio "sz luego a su bloque (1 bloque). Entonces: 2 inodes y 2 bloques.

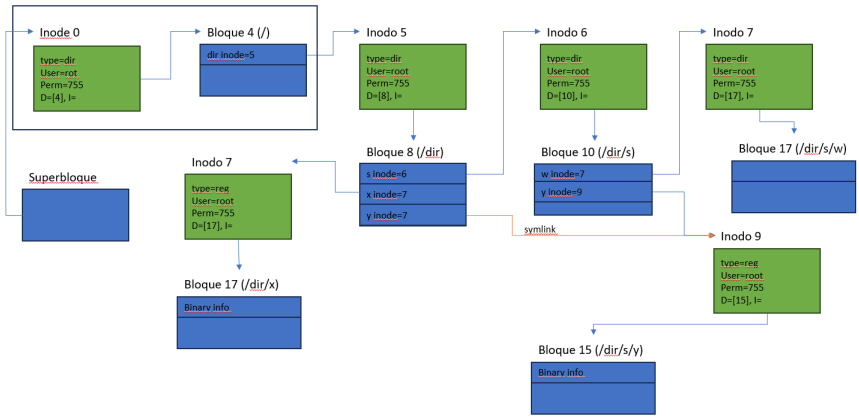

Figura 7: File System de la respuesta del Ejercicio 9.12 PowerPoint

Ejercicio 9.13. Describir que es un File System.

Respuesta 9.2. Un File System es un conjunto de estructuras de datos y algoritmos que permiten la administración de archivos en un sistema operativo.

- **Superbloque:** contiene información sobre el sistema de archivos, como el tamaño del sistema de archivos, el tamaño de los bloques, el número de bloques, el número de inodos, etc. Apunta al inodo del directorio raíz.

Ejercicio 9.14. Sea un disco que posee 2049 bloques de 4KiB y un sistema operativo cuyos i-nodos son de 512 bytes. Definir un sistema de archivos FFS. Explique las decisiones tomadas.

Respuesta 9.3. La estructura de un File System es la siguiente:

- **Superbloque:** apunta al inodo del directorio raíz.
- **bitmap inodos:** indica qué inodos están libres y cuáles están ocupados.
- **bitmap bloques:** indica qué bloques están libres y cuáles están ocupados.
- **inodos:** contiene información sobre los archivos, como el tamaño, los permisos, etc.
- **bloques de datos:** contienen los datos de los archivos.

Tenemos que el disco posee 2049 bloques de 4KB, entonces el tamaño del disco es de 2049 * 4KB = 8196KB. El tamaño del sistema de archivos es de 8196KB - 512KB = 7684KB. El tamaño de los bloques es de 4KB. El número de bloques es de 7684KB / 4KB = 1921 bloques. El número de inodos es de 7684KB / 512B = 15 inodos. Entonces, el superbloque contiene información sobre el sistema de archivos, como el tamaño del sistema de archivos, el tamaño de los bloques, el número de bloques, el número de inodos, etc. Apunta al inodo del directorio raíz. El bitmap de inodos indica qué inodos están libres y cuáles están ocupados. El bitmap de bloques indica qué bloques están libres y cuáles están ocupados. Los inodos contienen información sobre los archivos, como el tamaño, los permisos, etc. Los bloques de datos contienen los datos de los archivos.

Ejercicio 9.15. Los nombres de archivo no se almacenan en los i-nodos, sino en bloques de datos. ¿Por qué?.

Este diseño posibilita la implementación de enlaces de tipo, marque la opción correcta:

- symlink
- hardlink
- ambos ✓
- ninguno

Respuesta 9.4. Esta elección permite tanto enlaces simbólicos (symlink) como enlaces duros (hardlink). Los enlaces simbólicos son referencias a otros archivos por su nombre, mientras que los enlaces duros son múltiples entradas de directorio que apuntan al mismo inodo (y por lo tanto, al mismo conjunto de bloques de datos). Ambos tipos de enlaces comparten el mismo conjunto de bloques de datos, pero tienen diferentes inodos y nombres de archivo. Este diseño proporciona flexibilidad y eficiencia en la gestión de enlaces en el sistema de archivos. Ver Figura 9.5.

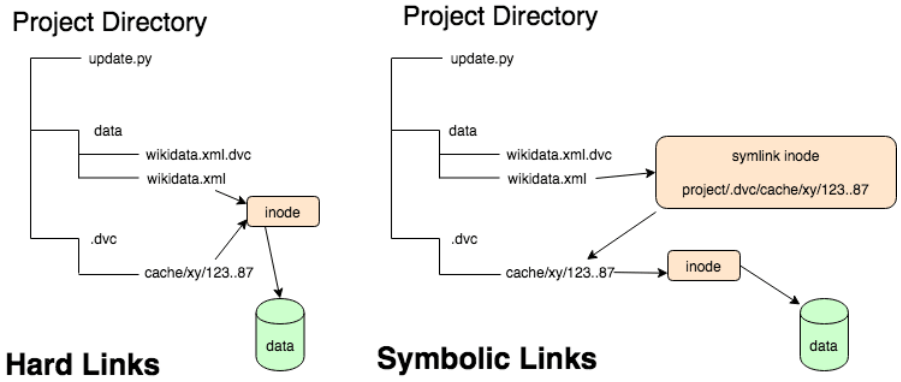

Figura 8: Hardlink y Symlink. Ejercicio 9.15

Ejercicio 9.16. ¿Qué es un inodo? ¿Qué información contiene?

9.6. Definiciones sueltas

El sistema operativo tiene que poder configurar el hardware de forma tal que cada proceso pueda leer y escribir solo su propia memoria.

10. Programación

1. **fork():**
2. **getpid()**

10.1. Fork

La única forma de que un usuario cree un proceso en el sistema operativo UNIX es llamando a la system call fork.
Crea un proceso y devuelve su id.

¿Que hace fork?:

- Crea y asigna una nueva entrada en la **Process Table** para el nuevo proceso.
- Asigna un número de ID único al proceso hijo.
- Crea una copia lógica del **contexto** del proceso padre, algunas de esas partes pueden ser compartidas como la sección text.
- Realiza ciertas operaciones de I/O.
- Devuelve el número de ID del hijo al proceso padre, y un 0 al proceso hijo.

Código de error:

- **EAGAIN:** No hay suficientes recursos disponibles para crear un nuevo proceso.
- **ENOMEM:** No hay suficiente memoria disponible para crear un nuevo proceso.
- **EPERM:** El proceso no tiene permiso para crear un nuevo proceso.

Referencias

[1] Resuemen de la catedra. En: (). URL: https://fisop.github.io/apunte/memory.html.

[2] Github donde se halla el projecto. En: 1 (). URL: https://github.com/fernk01/Resumenes_Latex/tree/sistemas_operativos.