

Resumen Unidad 1

Modo real

Direccionamiento en modo real

Como en modo real la paginación no está activada la **dirección lógica es igual a la física**.

Un registro de segmento me facilita un área en memoria para el direccionamiento de mi segmento y almacena la dirección de comienzo de los segmentos activos en memoria. Cada segmento tiene su propia dirección de comienzo y puede ser leído escrito o ejecutado.

Para obtener la dirección física se carga mi selector (que es el contenido de mi registro de segmentos) en la parte visible de mi registro de segmentos el cual se le corre 4 bytes a la izquierda y especifica la dirección base de mi segmento. Para acceder al segmento a este selector se le suma el offset y obtengo mi dirección física que es de 21-bits debido al acarreo que se produce.

Modo protegido

Segmentación

En modo protegido el selector tiene una dirección de base lineal asociada a el que se encuentra en un descriptor de segmentos. Entonces la función de mi selector es apuntar a dicho descriptor de segmento en la GDT. La dirección base lineal que se encuentra en el descriptor de segmentos es entonces sumada a un offset para general la dirección lineal de 32-bits la cual como paginación no está activada la dirección lógica es igual a la física.

Dicho selector es de 16-bits y contiene 2-bits para el nivel de privilegio 1-bit TI para seleccionar a que tabla vamos a acceder y un índice que se multiplica por ocho para escalar en los descriptors que contiene dicha tabla. Los bits de privilegio se usan cuando se quiere acceder a un nivel de privilegio mayor y la MMU (Unidad de mantenimiento de memoria compara estos bits con el nivel de privilegio del descriptor al cual queremos acceder.

Acceso a la LDT

Cuando el bit TI está en 1 indica que tenemos que acceder a un descriptor en la LDT. Entonces en LDTR se almacena un selector fijo el cual siempre apunta a un descriptor de LDT en GDT que contiene la información de la LDT, como la dirección base y el límite de la tabla. Una vez que ya obtuve esto tengo que volver a mi selector el cual apunta al descriptor de segmentos de la tabla para luego sumar la dirección base lineal que se encuentra en el descriptor mas el offset de la dirección lógica para obtener así la dirección base lineal.

Paginación

Es posible comenzar a ejecutar un programa, cargando solo una parte del mismo en memoria, y el resto se cargará bajo la solicitud.

Esta se habilita con el PG del registro CR0. Entonces la unidad de paginación organiza el espacio de direcciones físicas en 2^{20} páginas de 4K. El procesador usa dos niveles de tablas para trasladar la dirección lineal a dirección física.

Tenemos 3 componentes para el mecanismo de paginación:

Campo de Directorio (10-bits): Se usa como un índice dentro del directorio de páginas para especificar la entrada de directorio de páginas (PDE), la cual **especifica la dirección base de la tabla de páginas**.

Campo de páginas (10-bits): Se usa como un índice dentro de la tabla de páginas para especificar la entrada de traba de páginas (PTE), la cual **especifica la dirección base del marco de páginas**.

Offset (12-bits): Se usa para especificar uno de los 4096 bytes de la memoria del marco de pagina para poder acceder al dato.

Tipos	Accesos a memoria
Segmentación acceso a LDT	3
Segmentación acceso a GDT	2
Segmentación acceso desde parte oculta del RS	1
Paginación acceso a LDT	5
Paginación acceso a GDT	4
Paginación acceso a TLB (¿Cuanta como acceso la TLB?)	2

Cache de traslación de página TLB

Para que el procesador en cada acceso de memoria no tenga que pasar por los dos niveles de tabla, el procesador almacena las entradas de tablas de páginas usadas recientemente (PDE y PTE), en una cache llamada TLB, esta almacena hasta 32 entradas de tabla de página. Cuando una dirección lineal de 32-bits es obtenida por la unidad de segmentación. Los 20-bit superiores son comparados con las 32 entradas del TLB para determinar alguna coincidencia, si es así los 32-bits de la dirección física son calculados y es almacenada en el bus de direcciones.

Protección

El procesador utiliza mecanismos de *protección de nivel de segmentos y nivel de privilegio*

Protección por segmentación

Cuando queremos acceder a un segmento, el procesador chequea en la tabla de descriptores si contiene un descriptor para ese selector si no contiene se produce una interrupción.

Si se cumplen las condiciones de protección entonces la información del descriptor es copiada en la parte oculta del registro de segmento. Una vez copiado el procesador chequea el bit P para ver si el segmento para ese descriptor está presente. Después mas controles son hechos los cuales son *verificación de tipo* (usar segmentos en modo no correcto) y *chequeo limite* (offset mas la base no exceda el límite).

Protección de niveles de privilegio

Tiene 4 niveles de privilegio, el nivel 0 representa el nivel del privilegio más alto.

Nivel de privilegio del descriptor al cual queremos acceder (**DPL**)

Nivel de privilegio solicitado donde quiero ir (**RPL**)

Nivel de privilegio actual que es el descriptor almacenado en la parte oculta de la cache (**CPL**)

Para poder acceder a mi descriptor de segmentos la CPU compara estos 3 niveles de privilegios los cuales tienen que cumplir la siguiente condición $(CPL, RPL) \leq DPL$ (Numericamente) para poder acceder al segmento.

Call Gate

La necesidad de usar call gate es que yo necesito saltar de un nivel de privilegio menor a uno mayor.

Cuando hago un CALL mi selector ya no apunta a un descriptor de segmentos si no que apunta a un descriptor de puerta el cual contiene un selector y un offset, para acceder a dicho descriptor de puerta se tiene que cumplir que $(CPL, RPL) \leq DPL \text{ de Gate (Numericamente)}$, si se cumple esto entonces el descriptor se carga en la porción oculta del registro de segmentos y el selector en la parte visible por lo cual ahora mi **CPL = DPL de Gate** y mi

RPL = DPL Objetivo, por ejemplo si quiero ir de un PL = 3 a PL = 0 mi **DPL de Gate = 3 o 4** para poder acceder al descriptor de puerta, entonces una vez que accedí se carga mi selector del descriptor de puerta en la parte visible con un **RPL = 0** debido a que apunto a un descriptor de segmento que está en PL = 0 y en la parte oculta se copia mi descriptor de puerta por lo tanto mi **CPL = 3 o 4** y el **DPL Objetivo = 0** del descriptor de segmento que quiero acceder si la comparación de la CPU cumple que $DPL \text{ Objetivo} \leq (CPL, RPL) \leq DPL \text{ de Gate (Numericamente)}$.

Entonces accedo a mi descriptor en el cual obtengo la dirección base lineal y se la sumo al offset que contiene el descriptor de puerta, pudiendo así acceder a mi segmento de código que una vez que se ejecutó todo me retorna un valor a mi programa en el cual se hizo la llamada.

Multitarea

Segmento de estado de tareas (TSS)

Tipo especial de segmento usado para gestionar las tareas, el procesador lo usa como un block de notas, ósea que aquí se almacenan el contenido o contexto de una tarea.

TSS Descriptor

Como otros segmentos el TSS está definido por un descriptor llamado TSS descriptor.

Registro de tareas (TR)

Especifica la tarea en ejecución actual apuntando al descriptor de TSS el cual contiene a TSS.

Cambio de tareas sin task gate

Es cuando quiero ir por ejemplo de un PL = 1 a PL = 2 y ejecuto una instrucción JMP antes de cargar el nuevo selector en el TR, lo que se hace es con la información del descriptor que esta en la parte oculta la cual contiene el limite y la base para acceder al **TSS actual** en el cual guardo los registros internos del procesador, hecho esto el selector de la tarea actual se guarda en el TSS como un selector de enlace de regreso (bank link) y ahora si con LTR cargo el nuevo selector en la parte visible de mi TR para que apunte a un nuevo descriptor de TSS el cual se guarda en la parte oculta de TR actualizándola accediendo a mi **TSS nueva** para así cargar a los registros físicos del procesador con los datos de TSS para continuar con esta tarea y el bit B del descriptor de tareas nuevo es marcado como ocupado.

Cambio de tareas usando task gate

Método indirecto para el cambio de tareas. El cambio se hace a través de saltos o llamadas a task gate el cual la puerta de tareas se puede encontrar en la GDT, LDT, IDT entonces cualquier de estas puertas apunta a un descriptor de TSS para acceder al segmento TSS. Es igual que en el caso anterior solo que ahora se compara el DPL del descriptor de puerta de tareas con el CPL y RPL en ves del DPL del descriptor de TSS si se cumple la condición $(CPL, RPL \text{ de Puerta}) \leq DPL \text{ de Task Gate}$ entonces en vez de cargar el selector en TR para el descriptor de TSS, en nuestro programa nos referimos a CALL o JMP con lo cual especifica un selector que apunta a un descriptor de puerta en cualquiera de las tablas que contiene solo un selector que va a cargarse en la porción visible del TR el cual va a especificar nuestro descriptor de TSS de mayor privilegio el cual se carga en la parte oculta de TR y con la base y el limite accedemos a nuestro TSS.

Manejo de interrupciones en modo real y protegido

Interrupciones: Son usadas para el manejo de eventos externos asíncronos para el procesador.

Excepciones: Manejan las condiciones detectadas por el procesador mismo en el curso de las ejecuciones de las instrucciones.

Modo real

Hay una tabla la cual contiene punteros que definen el comienzo de ISR. Cada puntero representa dos palabras. La palabra que tiene la dirección de memoria mas alta contiene la dirección base del segmento y la que contiene la dirección en memoria más baja contiene el offset. Cada vez que una interrupción ocurre el procesador multiplica el numero de la interrupción por 4 para generar un índice en la IDT.

Modo Protegido

Ahora la IDT es un arreglo de descriptores de 8-byte y cuando una interrupción ocurre, su identificador que es un numero se multiplica por 8 y se suma a la dirección base de IDT almacenada en IDTR. El resultado me apunta a un descriptor de puerta en la IDT que puede ser de 3 tipos (interrupt gate, trap gate, task gate). Este descriptor contiene un selector y un offset, de modo que el selector apunta al descriptor de segmento que queremos acceder obteniendo la dirección de base lineal que le sumamos el offset de la puerta para así obtener la dirección lineal y acceder al segmento donde se encuentra la ISR (rutina de servicio de interrupciones) que se ejecuta y luego se retorna al programa.

Resumen Unidad 2

Sistemas operativos

El sistema operativo se ejecuta en modo kernel, el resto del software se ejecuta en modo usuario. Tiene dos funciones básicas proporcionar a los programadores un conjunto abstracto de recursos simples (maquina extendida) y administrar estos recursos de hardware (Multiplexación en el **tiempo** -> Asignación de la CPU | y en el **espacio** -> Cada cliente tiene una parte del recurso).

Llamada a sistema

Metemos los parámetros en la pila
Ejecutamos la función read()
Llamamos al procedimiento de biblioteca
Colocación del número de syscall en el registro de procedimiento
Lectura del registro de procedimiento y salto a la rutina de atención
Se ejecuta una instrucción TRAP
Se ejecuta la rutina de nivel privilegiado
Retornamos a la biblioteca
Luego retornamos a la instrucción siguiente de mi programa

Diferencias TRAP y llamada a un procedimiento

- TRAP cambia a modo Kernel mientras la llamada a procedimiento no.
- TRAP no puede saltar a una dirección arbitraria si no a una fija mientras la llamada a procedimiento va a una dirección relativa o absoluta.

Estructuras de un sistema operativo

Sistemas monolíticos

Esta escrito en un solo programa y se ejecuta en modo kernel. Son procedimientos enlazados entre si en los cuales cada uno tiene la libertad de llamar a cualquier otro.

Estructura

- Programa principal -> invoca procedimiento de servicios
- Procedimiento de servicios -> Invoca a llamadas del sistema
- Conjunto de procedimientos utilitarios -> ayudan a los procedimientos de servicios

Sistemas de capas

Es más lento porque cada tarea tiene que usar un servicio de la capa anterior hasta llegar a la capa de asignación de proceso y multiprogramación.

Consiste en ordenar el sistema operativo en una jerarquía de capas consta de 6 capas.

- 5_ El operador (Se localiza el proceso que opera)
- 4_Programas de usuario (No importa la gestión del kernel)
- 3_Administración en la entrada salida (Guarda en buffer el flujo entre ellos)
- 2_Comunicación operador-proceso
- 1_Administración de memoria (Administración de memoria para los procesos)
- 0_Asignación del procesador y multiprogramación

Estructura Microkernel

El objetivo es hacer el microkernel lo más liviano posible y cada vez que necesite gestionar memoria lo pido como un servicio a un proceso en modo usuario.

La idea es sacar las funcionalidades del kernel y llevarlas a modo usuario. El microkernel solo se encarga de la planificación de hilos y no de procesos esto posibilita multitarea. Siendo así el microkernel el que planifica todo el código que corre el sistema.

Estructura de Máquinas virtuales

Serian como dos capas:

- Capa de multiprogramación -> Hardware en donde se ejecuta el kernel
- Capa de maquina extendida -> Abstracción del hardware

Entonces yo tengo un proceso que usa la parte bonita del sistema operativo y cree que está solo, es decir aparentando que cada terminal (proceso) posee su propia maquina real la cual es proporcionada por la capa de hardware que se encarga de multiprogramar muchas máquinas virtuales sobre una maquina física.

Procesos

Un proceso es una entidad abstracta, definida por el kernel, para que los recursos del sistema se localicen de modo que la CPU ejecute un programa es decir una instancia de un programa en ejecución y el estado de los registros. Los procesos se ejecutan en pseudo paralelo. Debido a esto aprovecho tiempos muertos y hago posible que demore menos la ejecución de tareas haciendo rápida la conmutación de la CPU esto se define como multiprogramación. Diferenciando programa entidad pasiva y proceso entidad activa. Varios procesos pueden estar en distintas partes de mis programas.

Creación de un proceso

Tenemos solo una llamada a sistemas para crear un proceso esta es `fork()`, indica al sistema operativo que cree un proceso y le indica cual programa debe ejecutarlo. Su función es crear un clon exacto del proceso que hizo la llamada. En general el proceso hijo ejecuta después a `execve` o una llamada al sistema **similar** para cambiar su imagen de memoria y ejecutar un nuevo programa.

Terminación de un proceso

- Salida normal (voluntaria)
- Salida por error (voluntaria)
- Error fatal (involuntaria)
- Eliminado por otro proceso (involuntaria)

Jerarquías de procesos

El proceso hijo puede crear por si mismo mas procesos haciendo una bifurcación de procesos, formando una jerarquía de procesos.

Estado de un proceso

Nuevo a listo: Cuando cree un proceso y estoy buscando espacio en la memoria RAM y creando una entrada en la tabla de procesos, luego lo pongo en la lista de procesos que espera la CPU.

Ejecución a terminado: Dejo de ejecutarse, no se ejecuta mas porque no tiene espacio en memoria.

Listo a ejecución: Una rutina del sistema operativo se va a estar corriendo en el procesador que es el planificador que es parte del programa, lo que hace es elegir los procesos que están listos y va a cambiar en la tabla de procesos el estado de listo a ejecución y automáticamente hago un cambio de tareas usando task gate a ese PID *pudiendo retomar el estado en el que había quedado gracias a que cargue el selector de bank link en el registro de tareas volcando la información de TSS en el procesador para poder retomar donde había quedado.*

Ejecución a listo: No tengo forma de correr dos procesos a la vez, es decir el planificador (rutina del sistema operativo) y el proceso que se está ejecutando y si tengo un solo núcleo o sea una CPU por proceso esto no se puede. Lo que pasa es que se produce una interrupción de timer, la cual cuando espira el tiempo ejecuta una interrupt gate y se deja de ejecutar el proceso en ejecución y se atiende a la rutina de interrupciones del sistema operativo la cual cambia de estado y después llama a otra rutina del sistema operativo que llama al planificador para luego poder pasar de listo a ejecución y continuar donde se había quedado el proceso.

Ejecución a bloqueado: Si hay un proceso en ejecución y todavía no viene la interrupción de timer y este proceso hace una llamada al sistema en vez de esperar que me retorne lo que fue a buscar y aprovechar el tiempo porque que no se cuanto valla a tardar lo pongo en bloqueado y luego llama a la rutina del planificador para de nuevo pasar

un proceso en el estado listo a ejecución y reiniciando la interrupción de timer, para asegurarme que tengo todo el tiempo.

Bloqueado a listo: Si me viene una interrupción de hardware del Periférico que mi driver había programado entonces mi proceso en ejecución se suspende y se atiende la rutina de interrupción de hardware y se cambia el estado al proceso de bloqueado a listo, *luego se hace una task gate y el selector del bank link se carga en el registro de tareas para volver en donde se había quedado cuando se estaba ejecutando.*

Señales

Notificación asíncrona entregada a un proceso a partir de la ocurrencia de un evento.

Sería algo así como una interrupción por software. Yo puedo hacer que mientras se está ejecutando un proceso pasen 3 cosas:

- Si viene la señal que ejecute el manejador de esa señal que viene por defecto
- Si viene la señal puedo ignorarla y no hacemos nada
- Si viene la señal puedo poner un manejador para esta señal que haga una rutina.

IPC

La necesidad de usar IPC es que podemos implementar una tarea dividiéndola en subtareas y cada una de estas subtareas la podemos implementar como un proceso. Por lo tanto, voy a tener a varios procesos que están haciendo parte de la tarea que quiero hacer y en algún momento van a tener que compartir información. Es decir que si tendría más de un núcleo sería mucho más rápido realizar una tarea ya que estos procesos trabajan en simultaneo me permite modularidad que me permite remplazar un proceso por otro respetando su función en dicha tarea.

En común Fifo y Pipe

- Orientado a flujo de bytes (no segmentados)
- Procesos comparten información que residen en el kernel
- No tiene persistencia en el kernel
- Se necesita una llamada a sistema
- Persistencia de proceso
- Unidireccional
- Se pueden abrir con la configuración bloqueante o no bloqueantes
- No discrimina que proceso escribió los datos que se escribió en la tubería
- Lee el primer proceso el dato y el resto no ya que se quita del buffer

Pipe

Son tuberías sin nombres

Comunicación entre procesos relacionados

Para crear y abrir una tubería pipe()

Fifo

Persiste con un nombre en el sistema de archivos.

Son tuberías con nombres

Comunicación entre procesos no necesariamente relacionados

Para crear y abrir una fifo mkfifo() seguido de open

Cola de mensajes

- Procesos comparten información que residen en el kernel
- Se necesita una llamada a sistema
- Se pueden abrir configuraciones como bloqueantes y no bloqueantes
- Unidireccional

No se identifica quien envía cada mensaje para cada receptor

A cada mensaje se le asigna una prioridad

Lee el mensaje por antigüedad dependiendo la prioridad

Persistencia de kernel
Mensajes segmentados

Memoria compartida

Persistencia del sistema de archivos (archivos de memoria)
Persistencia de kernel (objetos de memoria)
Requiere sincronización
No requieren llamadas a sistemas para intercambiar datos

Hilos

M:N cuantos hilos quiero que se vean en usuario y cuantos en kernel

Flujo de ejecución dentro de un proceso

Descomponer una aplicación en varios hilos secuenciales que se ejecutan en cuasi paralelo realiza un procesos más rápido.

Con los hilos agregamos un nuevo elemento: la habilidad de las entidades en paralelo de compartir un espacio y todos sus datos entre ellos (variables globales). Son de 10 a 100 veces más rápidos que los procesos. Lo que agregan los hilos al modelo de procesos es permitir que se lleven a cabo varias ejecuciones en el mismo entorno del proceso. Cuando se ejecuta un proceso con multihilamiento en un sistema con una CPU, los hilos toman turnos para ejecutarse como en un proceso y se aprovechan los tiempos muertos. Un hilo puede leer, escribir o incluso borrar la pila de otro hilo.

Lo que estamos tratando de lograr con el concepto de los hilos es la habilidad de que varios hilos de ejecución compartan un conjunto de recursos, de manera que puedan trabajar en conjunto para realizar cierta tarea. Las transiciones entre los estados de los hilos son las mismas que las transiciones entre los estados de los procesos.

Por lo general, cada hilo llama a distintos procedimientos y por ende, tiene un historial de ejecución diferente. Esta es la razón por la cual cada hilo necesita su propia pila.

Un parámetro para `thread_create` especifica el nombre de un procedimiento para que se ejecute un nuevo hilo, este nuevo hilo se ejecuta en el mismo espacio de direcciones del hilo que lo creo. A menudo todos los hilos son iguales.

- **`thread_create`** crea un nuevo hilo
- **`thread_exit`** un hilo termina su trabajo.
- **`thread_join`** bloquea al hilo llamador hasta que un hilo (específico) haya terminado.
- **`thread_yield`** permite a un hilo entregar voluntariamente la CPU para dejar que otro hilo se ejecute

Implementación de hilos a nivel de usuario

M:1 muchos hilos en usuario y el kernel ve 1

Cuando un hilo hace una llamada a sistema se bloquean todos

Un paquete de hilos puede implementarse en un sistema operativo que no acepte hilos.

Los hilos se ejecutan encima de un sistema en tiempo de ejecución, el cual es una colección de procedimientos que administran hilos.

- Cada proceso necesita su propia tabla de hilos privada para llevar la cuenta de los hilos en ese proceso.
- La tabla de hilos es administrada por el sistema en tiempo de ejecución.
- La conmutación de hilos es más rápida que una instrucción TRAP
- El procedimiento que guarda el estado del hilo y el planificador son sólo procedimientos locales.
- Permiten que cada proceso tenga su propio algoritmo de planificación personalizado.

Implementación de hilos a nivel del kernel

Puede realizar la conmutación entre hilos y solo se bloquea el hilo en una llamada al sistema

- No tenemos tablas de hilos en cada proceso si no que el kernel tiene una tabla de hilos de todo el sistema.
- Llamada al sistema para crear o destruir hilos en la tabla de hilos del kernel.
- Los hilos del kernel no requieren de nuevas llamadas al sistema no bloqueantes
- Cuando un hilo se bloquea, el kernel, según lo que decida, puede ejecutar otro hilo del mismo proceso (si hay uno listo) o un hilo de un proceso distinto.

Planificación

Necesidad

Poder realizar multitarea debido a que aprovecho los tiempos muertos de la CPU y no la dejo ociosa. La función que cumple el planificador es la siguiente:

Cuando un proceso se está ejecutando y se produce una interrupción externa **no se ejecuta el planificador** si no lo que ocurre es que se pone el timer (el que hace que se produzca la interrupción de tiempo para que se ejecute la rutina que llama el planificador) en **pausa** y se ejecuta la rutina que atiende a mi interrupción que es la que lee el dato de un disco por ejemplo y lo pone en el buffer del proceso que lo estaba esperando y estaba bloqueado luego de esto se sigue ejecutando mi proceso actual hasta que expira el timer (quantum) y ahí recién **se ejecuta mi planificador** que realiza el cambio de contexto de tareas, guarda el contenido actual de mi proceso en el TSS y pasa recién ahí de bloqueado a listo el proceso que estaba esperando una interrupción de E/S que luego si le toca en la cola pasa a ejecución por el planificador.

Categoría de planificadores

Expropiativo: selecciona un proceso para que se ejecuta un tiempo fijo (quantum), si sigue en ejecución después de este tiempo se produce una interrupción de reloj debido a que expiro el timer para que la CPU le regrese el control al planificador. Por esta funcionalidad estoy siendo **más equitativo** con mis procesos limitados a E/S ya que los procesos CPU bonded no se van a ejecutar hasta que decidan ceder la CPU o se bloqueen si no cuando espire el timer, esto hace que mis **procesos limitados a E/S se terminen mucho más rápido**.

No expropiativo: selecciona un proceso y este se ejecuta hasta que el mismo se bloquea o hasta que libere la CPU en forma voluntaria y en las interrupciones de reloj no se toman decisiones de planificación. Este planificador **no es equitativo** ya que los procesos de E/S se van a ejecutar y si la tercera instrucción necesita un dato de disco se bloquea y se ejecuta el proceso CPU bonded por un tiempo muy largo y luego se vuelve a ejecutar mi proceso de E/S nuevamente tiene una instrucción que necesita un dato de disco y se bloquea por lo tanto los procesos de E/S se ejecutan muy lento mientras que los de CPU bonded terminan más rápido.

Sistemas por lotes

Se utilizan para hacer tareas periódicas en las que no interactúan con usuarios por lo tanto se usan algoritmos no expropiativo.

Algoritmos de planificación

- ✓ **Primero en entrar primero en ser atendido (FCFS):** la CPU se asigna de acuerdo al orden que van llegando los procesos y si un proceso se bloquea esperando E/S programa su periférico y luego se pasa al final de la cola. La desventaja es que no es equitativo con los procesos I/O bonded.
- ✓ **El trabajo más corto primero (SJF):** selecciona el trabajo mas corto primero sólo óptimo cuando todos los trabajos están disponibles al mismo tiempo y se los tiempos que tardan de antemano (tarea periódica). La desventaja es que no puedo saber cuanto demoran mis tareas de antemano a menos que sea periódica, debo tener mis tareas en el tiempo 0 para poder acomodarlas y las tareas más largas tardan más tiempo en ejecutarse debido a que si entran tareas más cortas seguirán ejecutándose estas. La ventaja es que disminuyen el turnaround.
- ✓ **El menor tiempo restante a continuación (SRTN):** el planificador selecciona el proceso que le falte menos tiempo para terminar su ejecución. La ventaja es que no necesito tener todas mis tareas en el tiempo cero debido a que mi planificador es expropiativo debido a que si llega tarde un proceso y tiene el tiempo restante de ejecución mas corto este se va a ejecutar, también mejora el throughput. La desventaja es que las tareas más largas tienen inanición y conocer tiempo de antemano.

Sistemas iterativos

El planificador expropiativo es esencial para evitar que el proceso no acapare la CPU y niegue el servicio a los demás ya que los usuarios esperarían una respuesta rápida entonces lo más importante es minimizar el tiempo de respuesta es decir el tiempo que transcurre un comando y obtener el resultado (proceso E/S).

Algoritmos de planificación

- ✓ **Planificación por turno circular (Round Robin):** a cada proceso se le asigna un quantum este no tiene que ser muy chico (se desperdicia el tiempo que se da la CPU) ni muy grande (se tarda en dar la CPU). En este método no tenemos el problema de que los procesos I/O bonded no tardan mucho ya que el tiempo que tarde en atender al periférico es mucho mas grande que el tiempo que tarda toda la lista de procesos en estado listo en ejecutarse gracias al quantum del planificador.
- ✓ **Planificación por prioridad:** a cada proceso se le asigna una prioridad y el proceso ejecutable con la prioridad más alta es el que se puede ejecutar, si tengo varios procesos en esa prioridad (CPU bonded) se usa Round Robin y como esto puede estar **indefinidamente** se disminuye la prioridad a medida que se van ejecutando para así bajar a la cola de estado listo en la prioridad más baja y que se ejecuten procesos de prioridad menor siendo más equitativo pero esto produce que si entra un nuevo proceso este se ejecute antes que los que estaban en el final de la cola y produce inanición.
- ✓ **El proceso más corto a continuación (SFJ, SPN):** se considera cada comando una tarea separada, se basa en realizar estimaciones del comportamiento anterior a través de un algoritmo de envejecimiento que realiza sumas ponderadas, si la tarea nunca se realizo antes no tengo forma de saberlo. El promedio ponderado que se obtiene le da mas peso a lo que se demoro la ultima vez que a lo que se demoro primero **contrario** a SFJ.
- ✓ **Planificación garantizada:** se da una proporción de la CPU $1/n$, donde n son los usuarios, y el proceso que se ejecuta primero es el que tiene la menor ratio que se calcula:

$$ratio = \frac{\text{Tiempo consumido}}{\text{Tiempo total (en funcion de la CPU)}}; \text{Tiempo consumido} = \frac{\text{Tiempo de inicio}}{N} \dots$$

El problema que tengo es que tengo que tener siempre la misma cantidad de usuarios.

- ✓ **Planificación por sorteo:** se selecciona al azar ya que el planificador sortea un boleto y el proceso que lo tenga se ejecuta, los procesos mas importantes pueden obtener boletos adicionales para incrementar su probabilidad de obtener la CPU más tiempo.
- ✓ **Planificación por partes equitativas:** se asigna la CPU en partes equitativas en función de los usuarios pero no en función de los procesos ya que si un usuario tiene mas procesos no importa se ejecuta un proceso en un usuario y después otro proceso en el otro usuario.

NOTA: cuando hacemos una llamada a sistema hacemos un cambio de contexto y ejecutamos una rutina del SO que es por ejemplo programar el controlador de un disco pero este no es el algoritmo de planificación después que programe el controlador el SO pone al proceso en estado bloqueado porque sabe que va obtener ese dato en un futuro lejano y recién ahí **llamo a la rutina para elegir otra nueva tarea que es el planificador.**

Sincronización

Para propósitos de sincronización no hay diferencias entre el modelo paralelo de CPU y el modelo de multihilamento.

Relación entre eventos

Concurrencia: esto ocurre cuando tengo multitarea debido a que mas de un evento puede ocurrir a la vez y mirando el programa no se sabe cual evento ocurre primero (no determinista).

Serialización: en la serialización un evento se debe producir después que otro, pudiendo este utilizar recursos modificados del primero que se ejecutó y no los recursos antes que se ejecutara,

Punto de encuentro: Un evento debe esperar por otro evento e inversamente en un punto de la ejecución, y ninguno de los dos puede continuar hasta que ambos hayan llegado. Si tenemos mas de uno se llama barrera. Esto posibilita que ambos eventos muestren el mismo dato requiere restricciones.

Exclusión mutua: dos eventos no deben producirse al mismo tiempo y los mismos modifiquen recursos compartidos, siendo el resultado dependiente del orden de corrida por lo tanto se deben usar restricciones para que el evento se ejecute de manera atómica.

Herramientas para sincronizar

Semáforo: se inicializa con un numero entero y si un hilo lo decrementa y este da negativo el hilo se boquea y cuando se incrementa aquel hilo que estaba esperando primero se desbloquea. Si el valor es positivo, representa el número de hilos que pueden disminuir sin bloqueo. Si es negativo, representa el número de hilos que se han bloqueado y están esperando. Si el valor es cero, significa que no hay hilos esperando, pero si un hilo intenta disminuirlo, se bloqueará.

Mutex: mutex garantiza que solo un hilo a la vez accede a la variable compartida. Un mutex es como un símbolo que pasa de un hilo a otro permitiendo que avance uno a la vez. *Es decir*, para acceder a la variable compartida debe obtener el mutex.

Solución a problemas

Poner bandera soluciona el problema de serialización, pero **gasto mucho recurso de la CPU** y no aseguro que sea una instrucción TSL es decir que no es atómica el cambio de bandera por lo tanto la relación de eventos se soluciona a través de:

Serialización

```
Hilo 1()
deposito plata
pthread_mutex_unlock(&mtx)

Hilo 2()
pthread_mutex_lock(&mtx)
saco plata

main()
Inicio el mutex (inicia en 1)
pthread_mutex_lock(&mtx)
```

Exclusión mutua

```
Hilo 1()
pthread_mutex_lock(&mtx)
zona critica (variable global)
pthread_mutex_unlock(&mtx)

Hilo 2()
pthread_mutex_lock(&mtx)
zona critica (variable global)
pthread_mutex_unlock(&mtx)

main()
inicio el mutex (inicia en 1)
pthread_mutex_lock(&mtx)
```

Punto de encuentro

```
Hilo 1()
encuentro
pthread_mutex_unlock(&mtx2)
pthread_mutex_lock(&mtx1)
Mismo dato H2

Hilo 2()
encuentro
pthread_mutex_unlock(&mtx1)
pthread_mutex_lock(&mtx2)
Mismo dato H1

main()
inicio los mutex (inician en 1)
pthread_mutex_lock(&mtx1)
pthread_mutex_lock(&mtx2)
```

Interbloqueo

```
Hilo 1()
pthread_mutex_lock(&mtx2)
pthread_mutex_unlock(&mtx1)

Hilo 2()
pthread_mutex_lock(&mtx1)
pthread_mutex_unlock(&mtx2)

main()
Inicio los mutex (inician en 1)
pthread_mutex_lock(&mtx1)
pthread_mutex_lock(&mtx2)
```

Gestión de memoria

La memoria principal (RAM) es un recurso que se debe administrar. Los sistemas operativos crean abstracciones de la memoria y las administran. Básicamente llevan el registro de cual es partes de la memoria están en uso, asignan memoria a los procesos cuando lo necesiten y desasignan cuando estos terminen.

Administrador de Memoria

Necesidad:

- ✓ Se ejecutan programas **solo** en memoria principal
- ✓ Se mejora la utilización de la CPU usando multiprogramación
- ✓ Por lo tanto, se deben mantener **varios** procesos almacenados en memoria principal simultáneamente

Funciones:

- ✓ Saber que partes están en uso y cuales no
- ✓ Asignar más memoria a los procesos si las necesitan
- ✓ Recuperar la memoria cuando los procesos terminan
- ✓ Manejar el intercambio entre disco y memoria (cuando no entran todos los procesos en memoria)

Monoprogramación y multiprogramación particiones fijas

Monoprogramación sin abstracción: Cada programa veía únicamente la memoria física por lo tanto no era posible tener simultáneamente dos programas debido a que uno podía escribir en el otro. El sistema operativo debe guardar todo el contenido de la memoria en un archivo en el disco, para que después se traiga a memoria y se ejecute el siguiente programa. Si sólo hay un programa a la vez en la memoria, no hay conflictos. Este concepto es el swapping

Multiprogramación sin abstracción: pero añadiendo cierto hardware es posible la multiprogramación sin swapping, pero tendríamos dos problemas:

- ✓ **Protección:** esto se logra particionando la memoria en 2Kb y a cada parte se le asigna una llave de 4bits, por lo tanto, cualquier intento por acceder a una parte de memoria con distinta llave se genera un trap evitando que los usuarios modifiquen otro programa.
- ✓ **Reubicación de memoria:** si tenemos dos programas en memoria principal los dos programas hacen referencia a la memoria física absoluta. Lo que deseamos es que cada programa haga referencia a un conjunto privado de direcciones locales para él, si no una instrucción podría especificar un salto al otro programa tenemos una solución que es la **reubicación estática** que modificaba el programa al instante a medida que se cargaba en memoria, no es muy conveniente debido tengo que saber cuáles palabras so reubicables y cuáles no pero no se utiliza porque es lento y complicado. Por lo tanto, recurrimos a la abstracción.

Multiprogramación con abstracción: la reubicación de memoria se resuelve con el espacio de direcciones que es el conjunto de direcciones que puede utilizar un proceso para direccionar la memoria. Cada proceso tiene su propio espacio de direcciones, independiente de los que pertenecen a otros procesos. Podemos asociar el espacio de direcciones de cada proceso sobre una parte distinta en memoria gracias al **registro base y limite**, es decir proporcionamos a cada proceso su espacio de direcciones privado.

Multiprogramación con particiones variables

Gracias a la evolución el SO pudo reasignar mas memoria cuando a un proceso le faltaba gracias al espacio de direcciones con esto las particiones se pueden hacer variables.

Intercambio

La memoria principal RAM es a menudo más chica que la que necesitan todos los procesos, entonces lo que se hace es llevar cada proceso completo a memoria y lo ejecuta durante cierto tiempo y después lo regresa al disco. Para esto debemos tener una reubicación dinámica de los procesos que están en la memoria que se hace posible gracias a los registros base y limite. Pero ahora tenemos dos problemas:

- ✓ **Fragmentación externa:** es todo espacio de memoria vacío (hueco) que no se le puede reasignar a ningún proceso debido a que no entra. Esta se puede evitar con compactación pero lleva tiempo y es lento.
- ✓ **Fragmentación interna:** es aquel espacio que se le asigna a un proceso y esta demás.

Administración dinámica de la memoria

Cuando la memoria se asigna en forma dinámica, el sistema operativo debe administrarla. En términos generales, hay dos formas de llevar el registro del uso de la memoria:

- ✓ **Mapa de bits:** aquí la memoria se divide en unidades de asignación. Para cada unidad de asignación hay un bit correspondiente en el mapa de bits, que es 0 si la unidad está libre y 1 si está ocupada. Cada bit puede valer o 16K de memoria RAM o 1 byte, pero si es muy grande estaría desperdiciando memoria, si se la tengo que asignar a un proceso chico y si es muy chico el mapa de bits sería muy grande, lo conveniente es que cada bit sea 1K.
Desventaja: dificultad en buscar espacio debido a que tengo que ir bit por bit y concatenar con los demás 0.
Ventaja: más fácil actualizar espacios vacíos.
- ✓ **Lista enlazada:** Cada entrada en la lista especifica un hueco (H) o un proceso (P), la dirección en la que inicia, la longitud y un apuntador a la siguiente entrada
Desventaja: más lento el actualizar los espacios vacíos debido a que tengo que sacar un elemento y sumarlo con otro para que me del total del espacio vacío que se me acaba de generar.
Ventaja: facilidad en buscar espacios vacíos ya que se fija si hay un hueco y si no salta y sigue buscando espacio.

Se pueden usar diferentes algoritmos:

- ✓ **First Fit:** el primer hueco en el que entra es el que usa.
- ✓ **Next Fit:** conveniente para usar parte final de la memoria, es como que deja una referencia en el último hueco que se usó para usar el que sigue.
- ✓ **Best Fit:** usa el hueco que mejor le quede analizando primero todos los huecos (genera mucha fragmentación externa).
- ✓ **Worst Fit:** los fragmentos externos que me queden sean muy grandes para que me entre un nuevo programa.
- ✓ **Quick Fit:** tengo varias listas con distintos tamaños de huecos. Dependiendo del espacio que necesite mi proceso elijo de donde sacar un hueco y asignarlo a mi proceso.

Memoria Virtual

Definición: se basa en hacerle creer al programador que el programa tiene una dirección consecutiva en cada línea de este, cuando en realidad esta en cualquier lado cuando se **traslada** al hardware y no se donde va a parar debido a que hay una caja negra en el medio que la ordena como le conviene al sistema operativo y al hardware y el espacio de direcciones es distinto al de la memoria física.

Cuando pongo la MMU (caja negra) por así decirlo el bus de direcciones ya no es más un cable, **no es directo** si no que hay una traslación y esta se encarga de ponerlo donde le convenga al hardware o al SO y si no tiene lugar en la memoria principal lo conecta al disco o a una memoria secundaria y después hace intercambio.

Paginación

Espacio de direcciones virtuales: estas direcciones se pueden generar usando indexado, registro base, registro de segmentos y otras formas, se divide en tamaños fijos llamados **páginas** y las unidades correspondientes en la memoria física (RAM) se llaman **marcos de página**. La MMU se encarga de hacer la traslación de direcciones virtuales (dirección lineal) a direcciones físicas. El proceso va a poder acceder al espacio de dirección virtual que es toda la memoria (incluyendo al disco) que yo puedo direccionar por ejemplo 64K y siempre va a ser menor a la memoria física por ejemplo 32K, es decir siempre voy a tener más páginas que marcos de página.

Fallo de página: cuando un programa se esta ejecutando y la pagina a la que necesita no está asociada lo que ocurre es que se produce un fallo de pagina el sistema operativo selecciona un marco de página que se utilice poco el bit $M = 1$ significa que lo tengo que copiar de nuevo en el disco por que se modificó la página y si el bit $M = 0$ entonces la desalojo directamente. Después obtiene la página que se acaba de referenciar en el marco de página que se acaba de liberar, cambia la asociación Presente/Ausente = 1 y reinicia la instrucción que originó el trap.

Tabla de páginas: el propósito de la tabla de páginas es asociar páginas virtuales a los marcos de página. Hablando en sentido matemático, la tabla de páginas es una función donde el número de página virtual es un argumento y el número de marco físico es un resultado.

Entrada de una tabla de página tiene los siguientes campos bits:

- ✓ Presente/Ausente: si nunca use antes mi pagina para cargarla en la memoria física este bit está en cero, por lo tanto, si uso la pagina que tenga en la entrada de la tabla de pagina este bit en 0 ocurre el fallo de página.y cuando la saco de la RAM con algún algoritmo se pone en 0 de nuevo.
- ✓ Protección: tipo de acceso permitido
- ✓ Referenciada: se utiliza para saber que pagina voy a sacar de la RAM para que entre otra.
- ✓ Modificada: se pone en 1 cuando la página se escribió automáticamente.

El número de página se utiliza como **índice en la tabla de páginas**, conduciendo al número del marco de página que corresponde a esa página virtual. Si el bit es Presente/Ausente = 1, el número del marco de página encontrado en la tabla de páginas se copia a los 3 bits de mayor orden del registro de salida, junto con el desplazamiento de 12 bits, que se copia **sin modificación** de la dirección virtual entrante. En conjunto forman una dirección física de 15 bits. Después, el registro de salida se coloca en el bus de memoria como la dirección de memoria física.

El problema con paginación es que tengo un espacio de direcciones unidireccional entonces si yo quisiera modificar algún segmento en memoria tengo que modificar todo mis espacios en memoria.

Algoritmo de remplazo de paginas

La asociación de una dirección virtual a una dirección física debe ser rápida por lo tanto se implementan los siguientes algoritmos:

- ✓ **No usadas recientemente (NRU)**: el bit R se borra en forma periódica (en cada interrupción de reloj) para diferenciar las páginas a las que no se ha hecho referencia recientemente de las que si se han referenciado. Cuando ocurre un fallo de página, el sistema operativo inspecciona todas las páginas y las divide en 4 categorías con base en los valores actuales de sus bits R y M:

Clase	R	M
0	0	0
1	0	1
2	1	0
3	1	1

Se selecciona la menor clase cuando se produce el fallo de página.

- ✓ **Primera en entrar, primera en salir (FIFO)**: en un fallo de pagina se elimina la página que esta primera y la nueva página se agrega a la parte final de la lista. El problema es que si mi programa la vuelve a necesitar la tengo que volver a poner.
- ✓ **Segunda oportunidad**: una modificación simple al algoritmo FIFO que evita el problema de descartar una página de uso frecuente es inspeccionar el bit R de la página más antigua. Si es 0, la página es antigua y no se ha utilizado, por lo que se sustituye de inmediato. Si el bit R es 1, **el bit se borra**, la página se pone al final de la lista de páginas y su tiempo de carga se actualiza, como si acabara de llegar a la memoria. La desventaja es que si mi programa es muy grande no voy a tener las suficientes paginas cargadas en memoria a tiempo y va a ser lento.
- ✓ **Reloj**: es una lista circular (Lista doblemente enlazada) igual que en el caso anterior se fija el bit R si es cero se remplaza la nueva página que se quiere usar y si no se pone el bit R = 0 y el puntero sigue a la siguiente página. Esto evita tener que andar copiando en la cola los elementos.
- ✓ **Menos usadas recientemente (LRU)**: se crea una matriz NxN donde N es el número de páginas. El objetivo es saber en todo momento cual es la página que hace más tiempo que no se usa. Esto se determina viendo la página que se usó tachando la fila y la columna de esa página y viendo en que fila tengo más cero es la página que menos se ha usado
- ✓ **WSClock**: Determina qué conjunto de trabajo (El conjunto de páginas que utiliza un proceso en un momento dado) debería sacar misma analogía del reloj con grupo de páginas. El objetivo de este algoritmo es que no ocurra fallos de página.

Segmentación

Lo que se necesita realmente es una forma de liberar al programador de tener que administrar las tablas en expansión y contracción. Una solución simple es proporcionar la máquina con muchos espacios de direcciones por completo independientes, llamados segmentos. Debido a que cada segmento constituye un espacio de direcciones separado, los distintos segmentos pueden crecer o reducirse de manera independiente, sin afectar unos a otros.

Para especificar una dirección en esta memoria segmentada o **bidimensional**, el programa debe suministrar una **dirección en dos partes**, un número de segmento y una dirección dentro del segmento. Esto soluciona el problema de que si tengo que sacar algún segmento y después ponerlo basta solo cambiar el segmento independiente y no tendría que modificar todo el programa si por ejemplo es stack aumenta.

Segmentación Paginada

Como en el caso de segmentación los segmentos no son todos iguales por lo tanto quedan pedacitos desocupados que no se pueden ocupar lo cual genera fragmentación externa.

Soluciona tanto la fragmentación interna como la externa es que particiono mis segmentos en partes de 4K, entonces voy a cargar en memoria principal RAM una porción del segmento de 4K cuando lo necesite y debido a esto van a ser todas mis partes iguales evitando fragmentación externa.

Si pongo páginas de segmento en la memoria me estoy evitando fragmentación interna debido a que me evito asignarle más espacio a los segmentos por las dudas que crezcan.

Sistemas de tiempo real

No hacemos cambio de contexto no tenemos multitarea.

Necesidad

Poder hacer que el tiempo entre la **valides lógica** y el instante que se produce (valides temporal) sea lo más rápido posible para garantizar el tiempo de respuesta. También que el sistema sea determinístico es decir que el tiempo de ejecución a de estar acotado en el caso más desfavorable de forma de **asegurar las restricciones temporales**.

Tener un sistema de tiempo real no significa que voy a poder cumplir los tiempos estrictos, sino que se me hace **más fácil** y va a depender si puedo cumplirlos.

Procesamiento secuencial

Consiste en ejecutar tareas consecutivamente en un bucle infinito, estas mismas **no pueden bloquearse** a la espera de un evento externo ya que los eventos externos son asíncronos con el funcionamiento del sistema por lo que el tiempo que va a tardar la tarea que espera el evento **no está acotado**, en consecuencia, no puede garantizarse el tiempo que va a tardar el bucle scan.

Es fácil obtener un tiempo de muestreo exacto siempre y cuando el **periodo sea mayor que la suma de los tiempos máximos de ejecución de las tareas** este se logra con un temporizador y si en ciertos periodos no se llega al tiempo máximo se agrega otra tarea que se llama **idle** que se ejecuta hasta que finalice el temporizador y luego esta lo reinicia devolviendo el control al programa principal ejecutando de nuevo el bucle scan, de esta forma se consigue **un periodo de muestreo independiente del tiempo de ejecución de las tareas y del hardware**.

Cuando tenemos tareas que tardan mucho en ejecutarse puede que otras tareas no se alcancen a terminar y por lo tanto no cumplamos las restricciones temporales para esto se recurre a **dividir la tarea** o cambiar por un **procesador más rápido**.

Latencia

El tiempo máximo que transcurre entre un evento y el comienzo de la ejecución de la tarea que lo procesa. Para este tipo de sistemas la latencia es igual al tiempo máximo que tarda el bucle scan debido a que el **caso más desfavorable** se da cuando el suceso externo ocurre justo **después** de su **comprobación** por parte de la tarea esto es T_s .

Ventajas:

- ✓ Facilidad de implementación
- ✓ Debido al orden fijo es fácil compartir información entre tareas
- ✓ No se pierde tiempo realizando cambio de contexto.

Desventajas:

- ✓ Latencia igual al tiempo que tarda el bucle scan
- ✓ Dificultad implementar tareas que no sean múltiplo del periodo de muestro.
- ✓ Dificultad de escalar el programa o modificar, es decir agregar una nueva funcionalidad ya que si hago esto tengo que cambiar todo.
- ✓ Si tengo que dividir muchas tareas para cumplir el tiempo de muestro resulta imposible

Sistemas Foreground/Background

Se basa en tener dos tipos de tareas, unas de primer plano (foreground) y otras de segundo plano (Background).

Latencia

En este tipo de sistema la latencia disminuye considerablemente debido a que ahora no tengo que esperar a mi bucle scan que termine cuando mi suceso ocurre después de la comprobación si no que se atiende a penas llega una interrupción y se deja la tarea de primer plano que se estaba ejecutando, entonces ahora la **latencia es el tiempo máximo que tarda una tarea de background** ya que si llega otra interrupción tengo que esperar a que termine una tarea de background. El problema que trae esto es el de los datos compartidos entre estos dos planos, ya que si se deja a medio hacer la tarea de primer plano y por ejemplo se le paso a la mitad los argumentos a una función para imprimir en pantalla lo que sucede es que imprime mal. La solución a esto es inhabilitar las interrupciones en lo que se llama a la zona critica que es donde se usa el recurso compartido, pero si el tiempo que tarda esta zona critica es grande **aumenta la latencia** de todo el sistema debido a que si llega una interrupción tengo que esperar a que termine de ejecutarse esta zona critica por lo tanto tiene que tardar el menor tiempo posible.

Ventajas:

- ✓ Baja latencia entre tareas
- ✓ No necesitan ningún software adicional como los RTOS
- ✓ No existe cambio de contexto entre tareas

Desventajas:

- ✓ Las tareas de segundo plano tienen que poder asociarse a una interrupción y están limitadas a cuantos temporizadores tengo en microcontrolador.
- ✓ Es mas complejo de programar debido a que aparecen problemas de concurrencia por la naturaleza asíncrona de mis interrupciones.

Sistemas operativos en tiempo real

Es mas parecido a una librería de funciones que se enlazan con una aplicación y esta toma el control cuando arranca y ejecuta al planificador que es el que le da el control al sistema operativo.

Realizo cambio de contexto y tengo multitarea gracias al planificador ya que es el encargado de cambiar las tareas de primer plano y usa la información como su prioridad o su limites temporales para decidir cual ejecutar, y las tareas de segundo plano las manejan las interrupciones.

Planificador FreeRTOS

En los RTOS son mas brutos debido a que si tengo una tarea de mayor prioridad no la termina de ejecutar hasta que esta se termine, entonces por lo general se suele bloquear esta tarea de mayor prioridad ejecutando una tarea que la haga demorarse para que el planificador la pase al estado bloqueado y así poder ejecutar la de menor prioridad. Tampoco no espera el quantum si aparece una tarea de mayor prioridad y se esta ejecutando una de menor.

Ventajas

- ✓ Multitarea: simplifica la estructura del código.
- ✓ Escalabilidad: se pueden agregar tareas sin tener que modificar tanto el programa.
- ✓ Reusabilidad de código: si las tareas tienen poca dependencia es fácil incorporarlas a otras aplicaciones.

Desventajas

- ✓ Usamos tiempo en el cambio de tareas, esta rutina que hace este cambio necesita estar en memoria entonces ocupa espacio.
- ✓ El programador NO decide cuando ejecutar cada tarea si no el planificador.
- ✓ No protege contra errores de programación.

Tareas

La tarea está definida por una función simple de C, lo único especial que tiene es su propósito, que debe devolver un void y recibir un puntero a void como parámetro.

Tipos de tareas:

- ✓ **Tareas periódicas:** con una frecuencia determinada. La mayor parte de su tiempo están en el estado bloqueado y esperan un tiempo de bloqueo para pasar al estado listo. Alta prioridad para tener baja latencia
- ✓ **Tareas aperiódicas:** con frecuencia indeterminada. Tareas inactivas bloqueadas hasta que ocurre un evento de interés.
- ✓ **Tareas continuas:** régimen permanente. Deben tener menor prioridad que el resto.

Métodos para proteger recursos compartidos

Inhabilitar las interrupciones:

- ✓ Desventajas: si la zona crítica tarda mucho tiempo en ejecutarse aumenta la latencia tanto de foreground como background.
- ✓ Ventaja: si la zona crítica tarda poco tiempo es un método rápido debido a que se realiza con pocas instrucciones de máquina.

Inhabilitar las conmutaciones de tareas:

Se realiza con vTaskSuspendAll

- ✓ Desventajas: sólo afectará a los tiempos de respuesta de las tareas de primer plano, que normalmente no son tan estrictos como los de las rutinas de interrupción es decir aumenta la latencia en badground
- ✓ Ventaja: la tarea se ejecuta hasta que termine y no cambia su contexto por otras tareas de mayor prioridad por lo tanto no tenemos problemas de concurrencia.

Semáforos:

- ✓ Desventajas: al ser un mecanismo más complejo, su gestión por parte del sistema operativo presenta una mayor carga al sistema
- ✓ Ventaja: sólo afecta a las tareas que comparten el recurso

Inversión de prioridad

La inversión de prioridad se da cuando dos tareas comparten el mismo semáforo y tenemos una tarea en metió. Si tenemos 3 tareas A, B, C y las tareas A y C comparten el semáforo, si primero se ejecuta C y luego se ejecuta B y luego cuando se ejecute A y quiera tomar el semáforo no va a poder por que la tarea C no lo libero por lo tanto se va a bloquear y va a pasar a ejecutar B y hasta que esta no termine no va a pasar a ejecutar a C para que termine y libere el semáforo pudiendo luego ejecutar a la tarea A, por lo tanto **la tarea B se ejecuto en vez de la tarea A.**

La solución para esto es usar planificadores no apropiativos y si no herencia de prioridad (Mutex no soluciona pero disminuye). En este caso lo que haría, es que cambiaría la prioridad de las tareas C y A para que cuando la tarea A pida el semáforo y se bloquee termine de ejecutar la C para poder liberarlo y poder ejecutar la de mayor prioridad A **en vez de tener que esperar a la tarea B que termine.**

Problema con interrupciones

Los RTOS no saben si las funciones las invocan desde alguna tarea del primer plano o de la ISR. Esto es porque la llamada a sistema se puede realizar de los dos lados y por ejemplo esta llamada abre una cola para proporcionar el dato que se estaba esperando de una tarea por lo tanto pueden ocurrir dos cosas:

- ✓ **Que se ejecute la tarea de alta prioridad:** este caso seria el que no tendría que pasar ya que es mas importante seguir atendiendo a la ISR por que si no mi latencia se haría muy grande ya que tengo que terminar de atender mi tarea de mayor prioridad para que se siga ejecutando la ISR

- ✓ ***Que se siga ejecutando la ISR:*** esto es de mayor prioridad y se puede lograr poniendo ***FromISR*** en la llamada a sistema que abre la cola para proporcionar el dato que espera la tarea de mayor prioridad entonces podemos saber quien hizo esa llamada a sistema y como es la ISR esta va a continuar en vez de la tarea de alta prioridad. Luego si se estaba ejecutando una tarea de menor prioridad se hace un cambio de contexto y se ejecuta la tarea de mayor prioridad

Redes

Hardware: tenemos dos tipos de tecnología:

- ✓ Tecnología de transmisión:
 - ❖ **Enlaces de difusión:** las máquinas **comparten** un mismo canal y el paquete se les envía a todas.
 - ❖ **Enlaces punto a punto:** conectan pares **individuales** de máquina.
- ✓ Escala:
 - ❖ **PAN:** 1m, ejemplo bluetooth
 - ❖ **LAN:** 10m-1km, propiedad privada se divide:
 - ✦ **Inalámbrica:** usan enlaces de **difusión** donde cada computadora se comunica con un dispositivo en el techo llamado **AP** (punto de acceso) que **transmite** paquetes entre ellas y también entre estas e internet (**Wifi**).
 - ✦ **Alámbrica:** usan enlaces **punto a punto** tienen un estándar llamado **Ethernet** en donde cada computadora se comunica a través del protocolo Ethernet y se conecta a un **switch** con un enlace punto a punto en donde los paquetes se difundirán por un solo **cable lineal**. Para redes mas grande se conectan los switch entre si mediante sus puertos.
 - ✦ **VLAN:** que es una LAN virtual es decir una **LAN logica** que surge de la division de una LAN **fisica** y es como que si tuvieramos dos **LAN distintas** en donde cada puerto del switch tiene colores diferentes y cada uno recibe unicamente por su puerto de color.
 - ❖ Tanto las redes inalámbricas como las alámbricas se pueden dividir en diseños
 - ✦ **Estático:** consiste en dividir el tiempo en **intervalos discretos** y utilizar un algoritmo por turno rotatorio (round-robin). Esta desperdicia la capacidad del canal cuando una máquina no tiene nada que transmitir durante su intervalo asignado.
 - ✦ **Dinámico:** puede ser **centralizada**, donde hay una sola entidad (estación base) que determina el turno de cada uno. En el método de asignación de canal **descentralizado** no hay una entidad central; cada máquina debe **decidir por su cuenta** si va a transmitir o no.
 - ❖ **MAN:** cubre toda una ciudad (10Km): acceso inalámbrico a Internet de alta velocidad han originado.
 - ❖ **WAN:** son redes compuestas (**telefonía celular**). Para conectar de host a host se utiliza una **subred** que es una combinación de enrutados y líneas de transmisión la cual transmite paquetes. Una WAN es parecida a una **LAN alámbrica extensa** solo con las siguientes diferencias:
 - ✦ Host y subred pertenecen a distintas personas
 - ✦ Los enrutadores conectan distintos tipos de red
 - ✦ A la subred se conectan diferentes cosas. Tenemos dos tipos de WAN
 - ✦ **VPN:** la red privada virtual consiste en enlaces virtuales y **no renta** líneas de transmisión.
 - ✓ **ISP:** la subred es una red ISP en donde el operario sería el proveedor de servicios. Esta red siempre se conecta con otros clientes siempre que **pagan**.
 - ❖ **Interredes:** a una colección de redes interconectadas se le conoce como **interred o internet**. La subred tiene más sentido a cuando pertenece al **operador**. También podríamos describir a una subred como una red ISP, o bien como una interred como una red. Una red se forma al combinar una subred y sus hosts.

Puerta de enlace (**Gateway**): es una máquina que realiza una **conexión** entre dos o más redes y provee la **traducción** necesaria tanto en términos de hardware como de software. Conviene usar puerta de enlace para la **capa en la parte media** que resulta ser la **ideal**, se le denomina capa de red; **un enrutador es una puerta de enlace** que conmuta paquetes en la capa de red.

Software

Aspectos de diseño para las capas:

- ✓ **Confiabilidad:** se basa en 3 aspectos código de detección y corrección de errores, la red debe tomar una ruta automática (enrutamiento), identificación de emisores y receptores en las distintas capas (direccionamiento).
- ✓ **Asignación de recursos:** se basa en compartir los recursos de manera dinámica sin que se sature el emisor y en caso de que se produzca se utilice realimentación para lograr un control de flujos, otros de los aspectos es evitar la congestión.
- ✓ **Calidad de servicio:** se basa en **proveer servicios** al mismo tiempo tanto para aplicaciones que lo requiere en tiempo real y que quieren un alto rendimiento.
- ✓ **Seguridad:** confidencialidad, integridad, autenticación.

Cada capa pasa los **datos** y la **información de control** a la capa inmediatamente **inferior**, hasta alcanzar a la capa más baja. Debajo de la capa 1 está el medio físico. Las capas piensan conceptualmente en su comunicación como si fuera **horizontal** aun cuando estos procedimientos se comunican con las capas inferiores a través de la interfaz de estas, no con el otro lado.

Protocolo: es un **acuerdo** entre capas adyacentes que se basa en **reglas y convecciones** para que las capas se comuniquen entre sí.

Servicios: se puede especificar como un **conjunto de primitiva**. En el caso que los protocolos se encuentren en S.O los servicios serían un conjunto de **llamadas al sistema** que hacen un salto a modo kernel para luego volverle el control al S.O y que este envíe los paquetes.

Interface: la interfaz de una capa indica a los procesos superiores como pueden acceder a ella.

La relación entre servicios y protocolos: mi protocolo es como doy servicios a mi capa superior y a su vez el protocolo depende del servicio de la capa de abajo. La relación entre estos es que mi protocolo hace que se cumpla mi servicio para la capa de arriba no importa que protocolo use siempre proporcione el mismo servicio en la capa que se implementa el protocolo, ejemplo el entramado lo aplico siempre no importa que protocolo tenga. Los **servicios** se relacionan con las **interfaces** entre capas. Mientras que los **protocolos** se relacionan con los **paquetes** que se envían entre las entidades pares de distintas máquinas.

Modelos de referencias

- ✓ **Modelo OSI** (interconexión de sistemas abiertos): se basa en los siguientes principios, que son las capas deben tener diferentes niveles de abstracción, funciones bien definidas que se eligen con protocolos estandarizados, límites para menor flujo de información, deben ser suficientes para no agrupar varias funciones en una misma.
Las capas son las siguientes:
 - ✦ **Capa física:** transmite bits por el medio de enlace.
 - ✦ **Capa de enlace:** enmascara los errores para tener una línea libre de errores.
 - ✦ **Capa de red:** encamina los paquetes y hace interconexiones entre redes heterogéneas, como así también se encarga del manejo de congestión.
 - ✦ **Capa de transporte:** aceptar datos de la capa superior y determina el tipo de servicio de la capa de sesión.
 - ✦ **Capa de sesión:**
 - ✦ **Capa de presentación:** se enfoca en la **sintaxis** y la **semántica** de la información transmitida.
 - ✦ **Capa de transporte:** contiene los protocolos que los usuarios necesitan.

✓ **Modelo TCP/IP:**

- ✦ **Capa de enlace:** una interfaz entre los hosts y los enlaces de transmisión y describe que enlace se deben usar para cumplir con las necesidades de esta capa de interred sin conexión.
- ✦ **Capa de interred:** la principal tarea de esta capa es entregar paquetes IP de manera independiente hacia sus destinos. EL ruteo y la congestión son los principales aspectos a considerar.
- ✦ **Capa de transporte:** permite que las entidades pares lleven a cabo una conversación. Se definieron dos protocolos de transporte de extremo a extremo:
 - ❖ **TCP** (protocolo de control de transmisión): es un protocolo *confiable orientado a la conexión* que discretizando mensajes hace posible la entrega de un *flujo de bytes sin errores* y maneja el *control de flujo*.
 - ❖ **UDP** (protocolo de datagrama de usuario): es un protocolo *sin conexión* y se usa donde es mas importante una entrega *oportuna* que una entrega precisa (voz o video).
- ✦ **Capa de aplicación:** contiene todos los protocolos de alto nivel.

Comparación de los modelos de referencia OSI y TCP/IP

Modelo OSI:

1. El modelo OSI hace **explícita** la distinción entre interfaz, protocolo y servicio.
2. Los protocolos están **ocultos** de una mejor forma y se pueden remplazar a medida que la tecnología avance.
3. El modelo OSI se ideó antes que existieran los protocolos por lo tanto **no sabían** bien en que capa ponerlos.
4. El OSI soporta la comunicación *sin conexión* y la orientada a la *conexión* en la **capa de red**, pero sólo la orientada a la conexión en la capa de transporte, en donde es más importante.

Modelo TCP/IP:

1. **No** tenían una **distinción** clara entre los 3 conceptos básicos.
2. Era un modelo que describía los **protocolos existentes** ya que los protocolos **llegaron primero**.
3. El modelo no encajaba en ninguna otra pila de protocolos, por lo que no era útil para describir otras redes si no **únicamente** la TCP/IP.
4. El TCP/IP sólo soporta el modo sin conexión en la capa de red, pero soporta ambos en la capa de transporte, lo que es muy importante para los protocolos simples de petición–respuesta.

Capa de Enlace

Este estudio se enfoca en los algoritmos para lograr una comunicación confiable y eficiente de unidades completas de información llamadas **tramas** (en vez de bits individuales, como en la capa física) entre máquinas adyacentes. Es importante saber en este punto que los canales de comunicación cometen errores. Además, sólo tienen una tasa de transmisión de datos finita y hay un retardo de propagación distinto de cero entre el momento en que se envía un bit y el momento en que se recibe.

Cuestiones de diseño de la capa de enlace de datos

Esta capa utiliza los servicios de la capa física para enviar y recibir bits a través de los canales de comunicación. Tiene varias funciones:

- ✓ Proporcionar a la capa de red una interfaz de servicio bien definida.
- ✓ Manejar los errores de transmisión.
- ✓ Regular el flujo de datos para que los emisores rápidos no saturen a los receptores lentos.

Para cumplir con estas metas, la capa de enlace de datos toma los paquetes que obtiene de la capa de red y los encapsula en tramas para transmitirlos. **El manejo de las tramas es la tarea más importante de la capa de enlace de datos.**

Servicios proporcionados a la capa de red

El servicio principal es la transferencia de datos de la capa de red en la máquina de origen, a la capa de red en la máquina de destino. Es más fácil pensar en términos de dos procesos de la capa de enlace de datos que se comunican mediante un protocolo de enlace de datos, pero no es la transmisión real.

Los servicios ofrecidos a la capa de red pueden ser:

- ✓ Servicio **sin** conexión **ni** confirmación de recepción (Ethernet): la máquina origen envía **tramas independientes** a la máquina de destino sin que esta confirme recepción. Si se pierde una trama **no** se realiza ningún intento por detectar la pérdida o recuperarse de ella, esto es apropiado cuando la tasa de error es muy baja y el tráfico en tiempo real donde es **peor el retraso que el error**.
- ✓ Servicio **sin** conexión **con** confirmación de recepción: se confirma de manera **individual** la recepción de cada trama enviada. Este servicio es útil en canales no confiables como Wifi, proporcionar confirmaciones es tan solo una **optimización**, nunca un requerimiento. En este caso si no llega una contestación después de un tiempo retraso. La desventaja es que para enviar otra trama tengo que esperar un tiempo de propagación de ida y vuelta para que me llegue el acuse de recibo y recién ahí enviar otra trama.
- ✓ Servicio orientado a conexión **con** confirmación de recepción: para resolver el problema de arriba se establece una conexión antes de transferirse los datos. Cada trama esta enumerada y la capa de enlace garantiza que cada trama enviada llegara a su destino en el orden correcto y solo **una vez**. Este servicio ofrece a los procesos de la capa de red el equivalente a un flujo de bits confiable. Su uso es apropiado en enlaces **largos y no confiables**.

Tenemos los siguientes servicios:

✦ **Entramado**

Es el servicio **fundamental** de la capa de enlace, esta se encarga de armar una unidad llamada **trama** con la información de 0 y 1 de la capa física y es más fácil gestionar ya que la capa de enlace divide el flujo de bits en tramas **discretas**, **empaquetando** estos 0 y 1 en tramas. A demás de separar el flujo de bytes para tener un tamaño **cuantificable** de 0 y 1 lo que hace es agregar información **adicional** esto me permite hacer **gestión de errores y control de flujo**. Con esto la capa de enlace calcula la **suma de verificación** para cada trama, e incluya esa suma en la trama al momento de transmitirla. Cuando una trama llega al destino, se recalcula la suma de verificación. Si ésta es distinta de la contenida en la trama, la capa de enlace sabe que ha ocurrido un error y toma las medidas necesarias para manejarlo.

Con el entramado se dónde empiezan mis datos y donde terminan

- ❖ **Conteo de bytes:** un campo en el encabezado especifica el número de bytes en la trama. Cuando la capa de enlace de datos del destino ve el conteo de bytes, sabe cuántos bytes siguen y, por lo tanto, donde concluye la trama. El problema es que si cambia el bits que especifica el tamaño de bytes de la trama **perderá la sincronía** y entonces será incapaz de localizar el inicio correcto de la siguiente trama. Rara vez se utiliza este método.

- ❖ **Bandera de bytes:** cada trama inicia y termina con un **bytes de bandera** evitando el problema de **sincronizar**. Pero puede pasar que el byte de bandera este en el **payload**, entonces cuando se tope con el siguiente byte de bandera que marcaría el final de la trama resulta que este es un **dato** y lo que sigue hasta el **próximo** byte de bandera se **pierde** debido a que el próximo que era el final pasa a ser el **inicio**. Para esto se agrega el byte de relleno (stuffing) que es un byte especial **ESC** (escape) que cuando este se lee se sabe que el que viene no es una bandera de fin de trama si no que es un dato.

En el caso de que en el payload venga un byte de ESC se coloca un byte de escape para el escape. Estos bytes en la capa de enlace del receptor quitan el byte de escape antes de pasarle los datos a la capa de red. El problema es que a medida que más de mis datos coincidan o con el byte de ESC o con el byte de bandera es que me va a **disminuir el rendimiento** y no aprovecho la capacidad máxima del canal.

- ❖ **Bandera de bits:** resuelve una desventaja de bandera de bits debido a que tiene que ocupar siempre **8 bytes** para los bytes especiales, en este caso se puede realizar el entramado a **nivel de bits** esto también lo hace mas **barato**. Cada trama empieza y termina con un **patrón de bits especial** que es 0111110 o 0x7E, este patrón es un byte bandera. Si el receptor pierde la pista de dónde está, sólo tiene que buscar la secuencia de banderas ya que sólo pueden ocurrir en los límites de las tramas y nunca dentro de los datos. El cero que agregamos cada 5 unos es el **bit de**

escape. El problema es que se tiene que realizar manejo de bits y para el programador es mucho mas trabajo.

- ❖ **Violaciones de codificación de la capa física**: utiliza un atajo desde la capa física. Estamos usando **violaciones de código** para delimitar tramas. Con este método aseguro que no voy a tener que agregar un relleno debido a que es un patrón especial que no son datos. El problema es que estoy agregando más información envié bits demás.

- ★ **Control de errores**: una vez resuelto el problema de marcar el inicio y el fin de cada trama, llegamos al siguiente dilema: cómo asegurar que todas las tramas realmente se entreguen en el **orden** apropiado a la capa de red del destino.

La manera normal de asegurar la entrega confiable de datos es proporcionar **retroalimentación al emisor**. Así, el receptor devuelve tramas de control que contienen confirmaciones de recepción positivas o negativas de las tramas que llegan tenemos siguiente posibilidades:

- ❖ **Sin conexión sin confirmación**: no garantizan nada
- ❖ **Sin conexión con confirmación**: puede que se pierda la trama de ida o la confirmación de vuelta. Debido a esto puedo tener **tramas duplicadas** ya que cuando se pierde mi confirmación no llega entonces espira el timer y vuelvo a retransmitir la misma trama que ya había recibido el receptor.
- ❖ **Orientada a confirmación con confirmación**: no tengo problema debido a que puedo enumerar la trama entonces el manejo de error es más **fácil**.

Para **detectar d** errores se necesita un código con distancia **$d+1$** ya que no hay manera de que errores de un bit puedan cambiar una palabra codificada válida a otra.

Para **corregir d** errores se necesita un código de distancia **$2d+1$** , pues así las palabras codificadas válidas están tan separadas que, aun con cambios, la palabra codificada original sigue estando más cercana que cualquier otra. Tenemos dos estrategias básicas para el manejo de errores:

- ❖ **Códigos de corrección de errores**: incluir suficiente información redundante para que el receptor pueda **deducir** cuáles debieron ser los **datos** transmitidos. Una trama consiste en **m** bits de datos (mensaje) y **r** bits redundantes (verificación) para esto tenemos diferentes **tipos** en donde la mayoría son códigos de bloque lineales sistemáticos:
 - ★ **Hamming**: se agregan bits de chequeo de paridad, estos están agregados a las potencias pares P1, P2, P4, P8, es decir que los bits que son potencias de 2 son bits de **verificación** que **obliga** a que el grupo de bits incluyéndolos sea de paridad par o (impar). El cálculo nos dice cual está mal, y en el caso que tengamos 2 mal no lo voy a poder corregir. Esto es debido a que cuando hago el chequeo de paridad me tienen que dar todos ceros para que no tenga error (en caso de paridad par), si me da alguno distinto de 0 entonces tengo que sumar los resultados del chequeo de paridad para saber cuál es el bits que tengo que cambiar esto lo vuelve hacer el receptor para saber si el resultado del chequeo de paridad esta bien y si el error es de un bit lo corrige si es de mas no lo puede diferenciar. La desventaja de este es que se hace en software.
 - ★ **Convolutional**: no es un código de bloque por lo tanto no tenemos un **límite** de codificación. La salida depende de los bits de entrada **actual y pasada** (el codificador tiene memoria). El número de bits pasados de los que depende la salida se denomina longitud de **restricción del código**. Por cada bits que entra salen 2 bits que son la sumas XOR de la entrada y los estados internos, estamos **duplicando** la información, entonces las salidas dependen de los estados anteriores y del bit de entrada.

- ✦ **Reed-Solomon**: son códigos de bloques lineales y con frecuencia también son sistemáticos. Éstos operan sobre símbolos de m bits y se basan en una ecuación polinómica de grado n .
- ✦ **Chequeo de paridad de baja densidad (LDPC)**: son códigos de bloques lineales. En estos códigos, cada bit de salida se forma sólo a partir de una fracción de los bits de entrada. Esto conduce a una representación matricial del código con una densidad baja de 1s las palabras codificadas recibidas se decodifican con un algoritmo de aproximación que mejora de manera reiterativa con base en el mejor **ajuste de los datos recibidos con una palabra codificada válida**. Usa justificaciones estadísticas y es para volúmenes de gran información (tramas grandes).
- ❖ **Códigos de detección de errores**: incluir sólo suficiente redundancia para permitir que el receptor sepa que ha ocurrido un error y entonces solicite la **retransmisión**.
 - ✦ **Paridad**: un código con un solo bit de paridad tiene una distancia de 2, ya que cualquier error de un solo bit produce una palabra codificada con la paridad incorrecta, por lo tanto, sólo puede detectar errores de un solo bit. Para detectar un solo bloque con 1 bit de error, basta con un bit de paridad por bloque. Un problema con este esquema es que un bit de paridad sólo puede detectar de manera confiable un error de un solo bit en el bloque. Si el bloque está muy confuso debido a una ráfaga de errores larga, la probabilidad de detectar ese **error se reduce a la mitad**. Es posible aumentar la probabilidad si cada bloque a enviar se trata como una matriz rectangular de n bits de ancho por k bits de alto. Si calculamos y enviamos un bit de paridad para cada fila, se detectarán de manera confiable errores de hasta k bits siempre y cuando haya, a lo sumo, un error por fila.
 - ✦ **Suma de verificación (checksum)**: la palabra **suma de verificación** se utiliza para indicar un grupo de bits de verificación asociados con un mensaje, sin importar cómo se calculen. Un **grupo de bits de paridad es una suma de verificación**. Generalmente, la suma de **verificación se coloca al final del mensaje**, como complemento de la función suma. Así, los errores se pueden detectar al sumar toda la palabra codificada recibida, tanto los bits de datos como la suma de verificación. Si el resultado es cero, no se ha detectado ningún error. Al agregar interleave al método por paridad, logro detecta errores de ráfaga con mayor facilidad debido a que hago el chequeo por paridad tanto en filas como en columna siendo una forma de suma de verificación cuando tengo que detectar errores en el receptor.
 - ✦ **Pruebas de Redundancia Cíclica (CRC)**: cuando se emplea este método de código polinomial, el emisor y el receptor deben acordar por adelantado un **polinomio generador, $G(x)$** . Para calcular el CRC para una trama con m bits, correspondiente al polinomio **$M(x)$** , la trama debe ser más larga que el polinomio generador. Divido a mi trama por $G(x)$ y el resto se lo resto a mi dato que quiero transmitir, entonces voy a transmitir este **checksum = $G(x)/(Dato - resto)$** y este va a ser **divisible** por el polinomio, entonces cuando el receptor recibe esto y lo divide al checksum por $G(x)$ si da 0 el resto es que no hubo error. Divido debido a que es más fácil dividir en binario.

Ni los códigos de detección ni los de corrección pueden manejar todos los posibles errores ya que los **bits redundantes** que ofrecen protección tienen la **misma probabilidad** de ser recibidos con **errores** que los bits de **datos**.

- ✦ **Control de flujo**: otro punto muy importante en el diseño de esta capa (y de las superiores) es qué hacer con un emisor que quiere transmitir tramas de manera sistemática y a mayor velocidad que aquella con que puede aceptarlos el receptor. En este caso, el receptor, comenzará a perder algunas tramas. Tenemos dos métodos para solucionar esto:

- ❖ **Control de flujo basa en realimentación:** el receptor regresa información al emisor para autorizarle que envíe más datos o para indicarle su estado.
- ❖ **Control de flujo basado en tasa:** limita la tasa a la que el emisor puede transmitir los datos, sin recurrir a la retroalimentación por parte del receptor.

Protocolo de enlaces de datos

La mayoría de la **infraestructura** de redes de área amplia se basa en las líneas de punto a punto. Aquí veremos los protocolos de enlace de datos que se encuentran en las líneas de punto a punto de Internet, en dos situaciones comunes:

- ✦ **Enlace SONET:** se envían paquetes de enlaces de fibra óptica en redes de área amplia.
- ✦ **Enlace ADSL:** operan en el lazo local de la red telefónica, en un extremo de Internet.

Un protocolo estándar llamado PPP (Protocolo Punto a Punto) se utiliza para enviar paquetes a través de estos enlaces. PPP se ejecuta en **enrutadores** IP para lograr el entramado para diferenciar los paquetes ocasionales del flujo de bits continuo en el que se transportan. Para transportar paquetes a través de estos enlaces, PPP provee tres **características** principales:

- ✓ **Método de Entramado:** delinea sin ambigüedades el final de una trama y el inicio de la siguiente. El formato de trama también maneja la detección de errores.
- ✓ **Protocolo de control de enlace (LCP):** activa las líneas, las prueba, negocia opciones y las desactiva cuando no son necesarias.
- ✓ **Mecanismo para negociar:** esto sirve para negociar opciones de la capa de red con **independencia** del protocolo de **red** que se vaya a utilizar. El método elegido debe tener un NCP (Protocolo de control de Red) distinto para cada capa de red soportada.

Formato de un protocolo PPP

- ✓ **Bandera de byte:** se rellena con byte de escape.
- ✓ **Dirección:** siempre se establece en 11111111 para indicar que todas las estaciones deben aceptar la trama. Así, evitamos tener que asignar direcciones en la capa de enlace de datos.
- ✓ **Control:** su valor predeterminado es 00000011. Este valor indica una trama no numerada (en Internet, generalmente se usa un modo “no numerado” para proveer un servicio sin conexión ni confirmación de recepción).
- ✓ **Protocolo:** indica la clase de paquete que está en el campo “Carga útil”. Se definen códigos que empiecen con un bit 0 para la versión 4 de IP (protocolo de capa de red), la versión 6 de IP y otros protocolos de capa de red que se podrían usar. [Los códigos que comienzan con un bit 1 se utilizan para los protocolos de configuración de PPP](#), entre los cuales está el LCP y un NCP diferente para cada protocolo de capa de red soportado.
- ✓ **Carga útil:** si la longitud no se negocia mediante LCP durante el establecimiento de la línea, se usa una longitud predeterminada de 1500 bytes
- ✓ **Checksum:**

PPP es un mecanismo de entramado que puede transportar los paquetes de varios protocolos (IP, de red) a través de muchos tipos de capas físicas. Este protocolo es sin conexión y sin confirmación.

Saber que la carga útil de PPP se mezcla aleatoriamente antes de insertarla en la carga útil de SONET. La aleatorización aplica operaciones XOR a la carga útil con una secuencia pseudoaleatoria larga antes de transmitirla ya que el flujo de bits de SONET necesita transiciones frecuentes de bits para la sincronización. Con la aleatorización, la probabilidad de que un usuario pueda ocasionar problemas al enviar una larga secuencia de ceros es muy baja.

Antes de transportar las tramas PPP a través de líneas SONET, es necesario establecer y configurar el enlace PPP.

El enlace inicia en el estado **MUERTO**, lo que significa que no hay conexión en la capa física. Al establecer una conexión en la capa física, el enlace pasa a **ESTABLECER**. Aquí, los iguales de PPP intercambian una serie de paquetes LCP para seleccionar las opciones PPP para el enlace.

Si la negociación de opciones LCP tiene éxito, el enlace llega al estado **AUTENTIFICAR**. Ahora las dos partes pueden verificar las identidades una de la otra, si lo desean. Si la autenticación tiene éxito, el enlace entra al estado RED y se envía una serie de paquetes NCP para configurar la capa de red. Por ejemplo, para el IP la posibilidad más importante es la asignación de direcciones IP a ambos extremos del enlace.

Una vez que el enlace llega a **ABRIR**, se transportan los paquetes IP en tramas PPP a través de la línea SONET. Al terminar el transporte de datos, el enlace pasa al estado **TERMINAR** y de ahí se regresa al estado **MUERTO** cuando se desactiva la conexión de la capa física.

Subcapa MAC (control de acceso al medio)

Es la parte inferior de la capa de enlaces de datos. Se aplica para redes de difusión en donde tengo un canal compartido y da la forma en la que las estaciones van a compartir el medio cuando se compite por él.

Tenemos dos tipos de asignación asignaciones:

- ✓ **Asignación estática:**
 - ❖ **FDM** (multiplexación por división de frecuencia): el BW se divide en N usuarios (Partes fijas CDU)
 - ✦ **Ventajas:** banda de frecuencias privada no tengo interferencia y eficiente (pocos usuarios).
 - ✦ **Desventaja:** se desperdicia el canal cuando es variable la cantidad de usuarios ya que o no puedo asignar una parte del BW (N grande) y desperdicio BW (N chico) . Pobre desempeño debido al retardo promedio.
 - ❖ **TDM** (multiplexación por división de tiempo) : se le asigna a cada usuario una N-ésima ranura de tiempo. No es muy flexible al igual que el FDM pero en vez de ancho de banda es ranura de tiempo.
- ✓ **Asignación dinámica:**
 - ❖ **Colisiones observables:** dos tramas se traslapan en el tiempo y se debe retransmitir.
 - ❖ **Tiempo continuo o ranurado:** continuo es que la trama se transmite en cualquier momento y ranurado significa que le doy un quantum de tiempo para que empiece a transmitir.
 - ❖ **Detección de portadora:** las estaciones pueden saber si el canal está en uso antes de intentar usarlo.

Protocolos de accesos múltiples

- ✓ **ALOHA**
 - ❖ **Puro:** cuando tengo el dato se transmite en una trama hacia el AP con voluntad propia, en donde este vuelve a difundir a todas las estaciones para verificar si llega bien es que la puede escuchar, en el caso de que haya una colisión, es decir al AP le llega la suma de ambas tramas, entonces la estación si no escucha su trama porque no la entiende, espera un tiempo aleatorio y se vuelve a retransmitir las tramas que colisionaron. Como G es la cantidad de veces que yo tengo que retransmitir para que una trama se envíe correctamente y $S = GP_0$ es la velocidad del canal y $P_0 = e^{(-2G)}$ la probabilidad de que una trama no se inicie en el periodo de otra, yo voy a tener mayor throughput cuando intente solo una vez y se transmita correctamente y a mayor G tengo menos probabilidad de colisiones. Con carga alta tengo bajo throughput debido a que tengo más probabilidad de colisión y $G \uparrow$ por lo tanto $S \downarrow$ y con carga baja tengo $S \uparrow$ debido a que tengo no tengo muchas colisiones por lo tanto $G \cong N$. Puedo colisionar de más de una manera.
 - ❖ **Ranurado:** se divide en intervalos discretos que son ranuras de tiempo y corresponden a una trama. Consiste en tener menos colisiones gracias a la sincronización de las estaciones que pueden transmitir solo después de un quantum (ranura de tiempo) y tienen que esperar obligada las demás estaciones. Esta sincronización se las da el AP que emite una señal especial para el comienzo de cada quantum o ranura. Es mas lento, pero puede colisionar de una **única manera** que es cuando se transmite a la vez. Se reduce el periodo de vulnerabilidad a la mitad.

Protocolos de accesos múltiples con detección con portadora

- ✓ **CSMA persistente -1:** primero escucho el canal antes de transmitir y transmito 1 **si o si** (probabilidad 1) si el canal esta desocupado. Si ocurre una colisión se transmite de nuevo luego de un tiempo aleatorio. Las colisiones pueden ocurrir cuando dos estaciones **transmiten a la vez**. Otro problema es el retardo ya que puede que justo cuando cense el canal todavía no le llega la trama a la estación que escucho entonces lo va a tomar como inactivo y va a transmitir generando una colisión.
- ✓ **CSMA no persistente:** primero se fija si esta ocupado el canal, si esta ocupado espero un tiempo aleatorio y no me quedo **censando** cuando se acaba el tiempo me vuelvo a fijar si esta ocupado, si es así, vuelvo a esperar un tiempo aleatorio posibilitando menos colisiones, pero es más lento.
- ✓ **CSMA persistente -p:** censo el canal si esta inactivo transmito con una probabilidad p y con una probabilidad $q = 1 - p$ espero hasta la siguiente ranura. Si de nuevo esta inactiva hago lo mismo hasta que transmita , pero si empieza a transmitir otra estación espera un tiempo aleatorio y empieza de nuevo (Actúa como colisión).

Una conclusión que se puede sacar de este grafico es que a medida que es menos persistente tenemos más throughput por más que tengamos más tráfico.

- ✓ **CSMA/CD** (detección de colisiones): ahorraríamos tiempo y BW. Este es un proceso **analógico** y solo es para redes alámbricas ya que en inalámbrica sufro atenuaciones no diferenciando la señal colisionada de la original. Debido a que estoy censando mientras estoy transmitiendo, cuando hay una colisión libero el canal a penas detecto diferentes niveles de tensión y espero un tiempo aleatorio y **no transmito toda la trama**. Para que una estación este segura de que ha podido tomar el canal sin tener colisiones, tiene que poder haber **transmitido** por un tiempo de 2τ , este es el tiempo de propagación de ida y vuelta de la estación más lejana (ALOHA ranurado con una ranura de 2τ).

Diferencias entre CSMA y ALOHA ranurado es que uno AP sincroniza y otro detecta la portadora.
Cuando hay una colisión no interrumpo la transmisión y en el otro si.

Protocolo libre de colisiones

Se resuelven las colisiones incluso en el periodo de contención.

- ✓ **Mapa de bits**: cada periodo de contención consiste en N ranuras. Se basa en colocar un bits en la ranura del canal correspondiente que quiere transmitir una trama en 1. Estos reservan la propiedad del canal por anticipado y evitan colisiones. No es muy flexible a la hora de tener varias estaciones ya que si tengo muchas mi trama de contención puede ser mas grande que los datos que quiero trasmitir.
- ✓ **Paso de token**: consiste en enviar una trama token por las estaciones y el que la tiene se fija si tiene para transmitir y si es así, se la guarda y puede transmitir y si no pasa el token a la siguiente estación. Mientras más información a transmitir tenga mas rendimiento y si tengo poca desaprovecho el canal. Las diferencia con el mapa de bits es que no tengo parcialidad en las estaciones y los periodos de contención están entremezclados en las tramas. **Si circula la información no circula el token y si circula el token no circula información.**
- ✓ **Conteo descendente binario**: en los anteriores la sobrecarga era de 1 bits por estación, por lo que no se **escala** bien para miles de estaciones. Para esto usamos **direcciones binarias** para las estaciones con un canal que combine las transmisiones. Entonces si una estación quiere usar el canal difunde su dirección. A todos los bits en cada posición de dirección de las diferentes estaciones se les aplica un **OR booleano** por el canal cuando se envían al mismo tiempo. Tan pronto como una estación ve que una posición de bit de orden alto, cuya dirección es 0, ha sido sobrescrita con un 1, se da por vencida. El proceso se repite con el resto de los bits. Después de ganar la contienda, la estación vencedora puede transmitir una trama, después de lo cual comienza otro ciclo de contienda.

Protocolos de contención limitada

Los protocolos de contención **limitada** usan contención cuando la carga es **baja** (para obtener un retardo bajo) y usan una técnica libre de colisiones cuando la carga es **alta** (para lograr una buena eficiencia de canal). Es decir, los CSMA con baja carga al no agregar trama de control aprovecho mejor el canal. Divido las estaciones en función de la carga, es decir poca estaciones canal compartido con todas y a medida que aumentan las estaciones las separo en grupos para que en cada grupo tengamos menos estaciones listas y tengamos más probabilidades de transmisión. Esto lo logramos con el siguiente protocolo:

- ✓ **El protocolo de recorrido de árbol adaptable**: cada ranura de bits está asociada a un nodo específico del árbol, siguiendo la analogía de arriba en el nodo uno tendríamos CSMA donde todos comparten el canal y para una **carga baja** aprovecho mejor el canal, pero si tengo **mucha carga** me conviene separar los canales en grupos los cuales no se podrían transmitir todos a la vez, por lo que le doy una ranura (nodo) a cada grupo teniendo menos estaciones listas que compitan por el canal a medida que tengo más colisiones posibilitando que cada grupo pueda tener mayor probabilidad de transmitir debido a que estos grupos van teniendo menos estaciones por ranura a medida que voy bajando en el árbol y una vez que se transmite la trama voy pasando de nodo en nodo desocupando el canal para cada ranura dada.

Protocolos de LAN inalámbrica

Estos protocolos no pueden detectar colisiones debido a que se puede tener mucha atenuación en la señal y no se note una **superposición** en las tramas. Por esto se utilizan **confirmaciones de recepción**. En este caso lo que importa en la recepción es la **interferencia** en el receptor, no en el emisor.

CSMA no tiene mucho sentido que cense el que va a transmitir debido a que a mi me interesa si hay portadora en el receptor ya que censar el canal no se puede debido a que sufre atenuación la información. También no tiene mucho sentido debido a los siguientes problemas:

- ✓ **Terminal oculto**: cuando dos estaciones se quieren comunicar, pero están muy lejos y en el medio hay una estación que hace que se mezcle la señal y se pierda.
- ✓ **Terminal expuesto**: cuando tenemos dos estaciones de transmisión juntas y con direcciones opuestas y una de ellas quiere transmitir si el canal está ocupado por la otra, esta no va a transmitir cuando sí podía.

Para solucionar esto se utiliza lo siguiente:

- ✓ **Protocolo MACA** (acceso múltiple con prevención de colisiones): cuando se transmite RTS (solicitud de envío) a las estaciones cerca de A se tienen que quedar calladas durante un tiempo que permita a A recibir CTS de B y las estaciones cerca de B se tienen que quedar calladas un tiempo que permita a B recibir la trama grande. La solicitud de envíos contiene la longitud de la trama de datos que seguirá después de la trama corta de advertencia para las demás estaciones. Pueden ocurrir colisiones en donde dos estaciones envíen RTS al mismo tiempo para una misma estación. Éstas chocarán y se perderán. En el caso de una colisión, un transmisor sin éxito (uno **que no escucha un CTS** en el intervalo esperado) espera un tiempo aleatorio y vuelve a intentar más tarde.

Ethernet

Existen dos tipos de Ethernet: **Ethernet clásica**, que resuelve el problema de **acceso múltiple** mediante el uso de las técnicas ya vistas; y **Ethernet conmutada**, en donde los switches se utilizan para conectar distintas computadoras. Las tenemos dos tipos de tramas Ethernet o IEEE 802.3 que contienen los siguientes campos:

Preámbulo de 8 bytes: la codificación Manchester produce un patrón de bits que equivale a una señal cuadrada para sincronizar al reloj del emisor con el del receptor y también para que las demás estaciones sepan que alguien va a transmitir.

Dirección de destino y dirección de origen (MAC) longitud de 6 bytes en donde los primeros 3 se los pone el fabricante y los otros 3 se le ponen al dispositivo. El origen lo mando para que me puedan contestar que le llegó la trama. Si no pongo este campo, como todas las estaciones la van a recibir, les llega una interrupción de hardware y el SO deja lo que estaba haciendo para atenderla, llama la rutina de atención que se da cuenta que es una trama de ethernet como **no sabe si es para ella** se lo va a pasar a la capa de red y recién ahí me voy a enterar si es para esta estación haciendo una AND con la dirección IP y la máscara de subred y si no da el nombre de red la descarto las demás estaciones también procesan hasta ese nivel y la descartan. Destino primero porque directamente después que detectan la señal cuadrada y se sincronizan saben que viene un dato y lo empiezan a comparar con su dirección MAC y si no es su dirección MAC siguen y **no generan una interrupción de hardware en el SO**.

Tipo o Longitud (trama Ethernet o IEEE 802.3): Ethernet usa el campo **Tipo** para indicar al receptor a qué protocolo de la capa de red va la trama. El IEEE 802.3 decidió que este campo transportaría la longitud de la trama, ya que para determinar la longitud de Ethernet había que ver dentro de los datos. Para esto se usó el protocolo LLC (control de enlace lógico en donde se encuentra el **tipo**). En la actualidad los números menores o iguales a 1500 son longitudes y los mayores son tipo.

Datos: estos deben cumplir una longitud mínima de 46 bytes para hacer que las tramas tarde 2τ para enviarse ya si ocurre una colisión Ethernet genera una ráfaga de ruido de 48 bits y como el tiempo que tarda llegar el ruido al emisor es 2τ , la trama corta llegará primero tomando como exitosa. Debo tener entre los datos y el relleno 46 bytes para asegurar que cuando me llegue la ráfaga de ruido siga transmitiendo y detecte este cambio en el voltaje y pueda volver a retransmitir.

Suma de verificación: es un CRC de 32 bits.

El protocolo de subcapa MAC de la Ethernet clásica

- ✓ **CSMA/CD con retroceso exponencial binario:** el tiempo se divide en ranuras discretas cuya longitud es igual al tiempo de propagación de ida y vuelta 2τ . Cuando tenemos i colisiones se elige un número aleatorio entre 0 y $2^i - 1$, y se salta ese número de ranuras. Este algoritmo asegura que cuando tenga mucha carga, es decir muchas colisiones se resuelva en un intervalo razonable y cuando tengo carga baja pocas colisiones tengo un retardo pequeño.

Ethernet conmutada

Ethernet avanzó de un **solo** cable extenso, ahora podían ser par trenzado debido a que se conectaban a un hub. En esta configuración es más simple agregar o quitar una estación. Estos hubs tenían los siguientes problemas:

- ✓ No incrementaban la capacidad del canal, al agregar estaciones se le daba una porción menor de este.
- ✓ Todas las estaciones estaban en el mismo dominio de colisión.
- ✓ Deben usar el algoritmo CSMA/CD para programar sus transmisiones
- ✓ Cualquier máquina puede ver el tráfico transmitido por las demás.

Surgió otra forma de tratar con el aumento de carga que fue **Ethernet conmutada** esta usaba switch que:

- ✓ Envía tramas a los puertos para los cuales están destinadas.
- ✓ Cada puerto es su propio **dominio de colisión independiente** mejor aislamiento.
- ✓ La capacidad se utiliza, se eficiencia.
- ✓ Tiene un búfer para que pueda almacenar la trama que está esperando salir para su puerto.
- ✓ Es posible usar a los puertos como **concentradores**.

Fast ethernet: cambió un poco de cosas, pero lo hizo trabajar a más frecuencia.

Gigabit Ethernet: debido a que tiene mayor frecuencia se podía propagar como máximo 10m lo que se hizo fue poner 512bytes en la capa física para no modificar el protocolo. También si se tenía varias tramas para enviar lo que se hacía era concatenarla en una sola y no hiciera falta el relleno de 512bytes.

Capa de Red

Se encarga de transportar paquetes de origen a destino eligiendo la ruta indicada proporcionando un mecanismo de conmutación de paquetes de almacenamiento y envío que se basa en almacenar los paquetes que van llegando en el payload de una trama en la capa de enlace, se espera el procesamiento de esta hasta comprobar la suma de verificación y luego se reenvía la trama por un enlace hasta el router más cercano y así hasta llegar al destino.

- ✓ **Implementación del servicio sin conexión:** los paquetes (datagramas) se envían por separado y se enrutan independientemente gracias a una tabla de enrutamiento la cual consiste en un destino y una línea de salida para ese destino. En caso de congestión cambia esta tabla y lo reenvía por otro lado.
 - ❖ **Ventaja:** se recupera más rápido ante caídas.
- ✓ **Implementación de servicios con conexión:** necesitamos una red de circuitos virtuales la cual establece la ruta a tomar y los datos se deben enviar después de establecer el circuito virtual. Esto se configura en el momento de la conexión y se almacena en una tabla dentro de los enrutadores. Para generar esta tabla se utiliza un proceso llamado conmutación mediante etiquetas MPLS el cual asigna una **etiqueta** a la conexión con el algoritmo de reencaminar, esto es para que no se corra el algoritmo de ruteo en cada router y porque si tengo dos hosts el primer router podría diferenciarlos, pero el segundo ya no. Es como si ahora tengo una tabla, pero de etiquetas. Si cambio de páginas en internet tengo que rehacer un CV.
 - ❖ **Ventajas:** puedo asegurar calidad de servicio y ancho de banda ya que se porque routers se van a encaminar mis paquetes.
 - ❖ **Desventaja:** tengo que armar antes mi circuito virtual para poder enviar paquetes y si se me cae un router para poder enviar mi paquete voy a tener que establecer nuevamente todos mis circuitos virtuales que pasaban por este router, para que pase por un nuevo router y continuar debido a esto es muy **lento**.

Interconexión de redes

Se utiliza una capa en común para todos que es la capa de red para ocultar las diferencias existentes y que proporciona un paquete universal que todos los enrutadores reconocen.

Procedimiento: meto los datos en el payload de la capa de red y junto con el encabezado que genera la capa de red (IP) lo meto en el payload de la capa de enlace que luego termina el procesamiento y la suma de verificación la puedo mandar al siguiente router que recibe la trama con el encabezado del tipo de interface del router anterior. Debido a que pueden tener diferentes interfaces los router se desencapsula la trama que llega, sacando del payload el paquete que contiene la dirección IP y lo almaceno en la capa de red para luego encapsularlo en la trama con diferente interfaz, hecho esto lo vuelvo a reenviar y repito lo mismo con los routers de diferentes interfaces hasta llegar al destino en donde saco del payload de IP los datos para la capa de transporte, entonces es como si se estuviesen comunicando virtualmente las capas de transporte.

Con esto podemos decir que:

- ✓ **Enrutador** es el que extrae de la trama el paquete y de ahí se utiliza la dirección de red para decidir a donde mandarlo, también estos son los que se encargan de conectar distintas redes en la capa de red por lo tanto deben entender los protocolos de esta.
- ✓ **Switch** transporta la trama con base en su dirección MAC y estos no tienen que entender la capa de red y se encarga de conectar el mismo tipo de red en la capa de enlace.

Tunelización

Se usa cuando tenemos un tipo diferente de red en el medio, es decir no tengo forma de traducir de IPv6 a IPv4. Entonces cuando se genera el paquete IPv6 este se encapsula en una trama se envía al siguiente router que tiene que ser multiprotocolo (tiene ambas redes) entonces el paquete se desencapsula de la trama que se recibió y como del otro lado de este router no tengo una red IPv6 lo que se hace es pasar el paquete IPv6 a una capa de red superpuesta en la cual encapsulo este paquete IPv6 en el payload de un paquete IPv4, es decir encapsulo un paquete de red dentro de otro paquete de red y recién ahí lo encapsulo en una trama compatible con IPv4 con destino al siguiente router multiprotocolo que luego desencapsulo para seguir trabajando con IPv6.

- ✓ **Desventajas:** la información no le puede llegar a ninguno de los host del trayecto con la red que tiene IPv4 debido a que mi dato está dentro del payload de IPv6.

Fragmentación

Surge un problema de interconexión de redes cuando un paquete grande quiere viajar a través de una red cuyo tamaño máximo de paquete es muy pequeño para solucionar esto tenemos las siguientes soluciones:

- ✓ **Conocer el MTU de la ruta** (Unidad máxima de transmisión de la ruta):
- ✓ Permitir que los enrutadores dividan los paquetes en fragmentos y envíen cada uno como un paquete de red separado existen dos métodos para este:

- ❖ **Fragmentación transparente:** se fragmenta el paquete a la entrada del router y se reconstruye a la salida de este. Tiene los siguientes problemas:
 - ✦ Necesita contar las piezas y necesita un bit de fin de paquete.
 - ✦ No se puede tomar otras rutas debido a que todas las piezas tienen que salir por el mismo enrutador.
 - ✦ Tiene que colocar todas las piezas en un búfer debido a que no saben si van a llegar todos y tener que descartarlos.
- ❖ **Fragmentación no transparente:** la recombinación ocurre solo en el host destino y para el caso de IP a cada fragmento se le asigna un número de paquete. Entonces cuando llegan al destino usa el número de paquete y el desplazamiento del fragmento para colocar los datos en la posición correcta, y la bandera de fin de paquete para determinar cuándo tiene el paquete completo
 - ✦ Ventaja: los router tienen que trabajar menos.
 - ✦ Desventaja: Si se pierde un fragmento todo se pierde y se tienen que agregar bits para el control siendo una carga mas grande para los host.

Descubrimiento de MTU de la ruta

Cada paquete IP se envía con sus bits de encabezado establecidos para indicar que no se puede realizar ningún tipo de fragmentación. Si un enrutador recibe un paquete demasiado grande, genera un **paquete de error**, lo devuelve al origen y descarta el paquete. Cuando el origen recibe el paquete de error, usa la información interna para **volver a fragmentar el paquete** en piezas que sean lo bastante pequeñas como para que el enrutador las pueda manejar. Si un enrutador más adelante en la ruta tiene una MTU aún más pequeña, se repite el proceso.

- ✓ **Ventaja:** el origen sabe la longitud a la cual tiene que enviar el paquete, entonces podemos decir que la fragmentación se saca de la red y se la pasa a los hosts. También no hace falta enviar todos los fragmentos como antes.
- ✓ **Desventaja:** puede tener retardos adicionales solo por enviar el paquete.

Protocolo IPv4

Versión: lleva la versión del protocolo.

IHL: longitud del encabezado en palabras de 32bits

Servicio ofrecido: 6 bits superiores indican la clase de servicio y los 2bits inferiores llevan la notificación de congestión.

Longitud total: incluye tanto el encabezado como los datos y su longitud máxima es de 65535.

Identificación: para que el host sepa a qué paquete pertenece un fragmento recién llegado.

DF: es una orden para que los enrutadores no fragmenten el paquete.

MF: para marcar el último fragmento, ya que es el que no tiene establecido este bit los demás sí.

Desplazamiento del fragmento: indica a qué parte del paquete actual pertenece este fragmento.

Tiempo de vida: es un contador que si llega a cero el paquete se descarta, se decrementa al pasar un router.

Protocolo: indica a cuál proceso de transporte (TCP o UDP) debe entregar el paquete.

Suma de verificación de encabezado: se debe calcular en cada salto debido a que hay un campo que cambia (TTL).

Dirección de origen y Dirección de destino: indican la dirección IP de las interfaces de red de la fuente y del destino.

Opciones: permite que las próximas versiones incluyan información adicional en cuando al diseño original.

Pasos que hace un enrutador para enviar un paquete IPv4:

1. Almacenar el paquete en un buffer
2. Se inspecciona el paquete para fijarse la versión, si esta no es compatible lo descarta, si es, sigue adelante.
3. Verificamos el TTL (tiempo de vida). Si este es cero descartamos el paquete.
4. Luego se fija la longitud del encabezado.
5. Sabiendo la IHL puede calcular el checksum.
6. Con la dirección destino voy a la tabla. Esta me dice por qué interfaz enviarlo.
7. Decremento el tiempo de vida y volver a recalcular el checksum.
8. Reenvió el paquete.

En caso de que no entre el paquete en la longitud total lo que se tiene que hacer es fragmentar copiando el encabezado y generar paquetes los más largos que pueda cambiando el offset y recalculando el checksum y mandar a cada uno de los paquetitos y se hace con cada uno de ellos.

Protocolo IPv6

La diferencia es que tiene con respecto a IPv4 son:

- ✓ Un tamaño de encabezado fijo 40bytes.
- ✓ No tengo checksum porque antes la tasa de error era mucho más grande que ahora, entonces si hay un error que se encargue otra capa.
- ✓ Tampoco tengo la parte de fragmentación, o sea que siempre se trabaja con el mismo largo de paquetes, entonces no pasa por el medio que se quiere mandar se descarta y es problema de otro.
- ✓ Se agrando 4 veces dirección origen y dirección destino.

Los pasos para enviar una trama son:

1. Verifico la versión
2. Decremento si puedo, si no, descartamos el paquete
3. Miro la dirección y lo reenvío.

El next header es el que en IPv4 era el protocolo. Si llega a poner una opción la podría poner en el payload y esa opción después va a tener un next header que va a ser la capa de transporte a la que va a ir. Si tuviera varias opciones van a ir encadenadas y al final el next header va a ir al protocolo de capa de transporte al que le voy a mandar los datos.

Direcciones IP

Una dirección IP no se refiere a un host, sino a **una interfaz de red** que es una placa, dispositivo físico o lógico, por lo que, si un host está en dos redes, debe tener dos direcciones IP. En contraste, los enrutadores tienen varias interfaces y, por lo tanto, múltiples direcciones IP. Se puede tener más de una IP por interfaz.

Para reducir las tablas de enrutamiento todas las máquinas en una misma red van a tener la dirección IP con el mismo prefijo y van a diferir en los hosts únicamente.

Máscara subred

Debido a la mala granularidad de redes por bloques fijos se creó la máscara de subred que nos dice cómo van a variar las redes. Es decir, tengo una granularidad de un bit y podría saltar de redes con muchos hosts a redes con pocos hosts.

- ✓ Ventajas: en los routers solo voy a direccionar con la dirección de red y no de host.
- ✓ Desventajas: la dirección de red de 32 bits de un host va a depender de en qué red está, ya que si cambio de red va a mantener el host pero hay que cambiar el prefijo porque no va a saber cómo llegar a ese lugar.

Subredes

Para saber a qué subred pertenece un paquete con cierta dirección IP lo único que tengo que hacer es una AND entre esta dirección IP y la máscara de subred en el router y tomar los bits del prefijo que corresponde a la subred y compararlo con el prefijo de este resultado, si el resultado coincide con el prefijo de la subred el paquete se envía por ahí. La dirección de prefijo es el nombre de la red.

El enrutamiento por prefijo requiere que todos los hosts en una red tengan el mismo número de red. Si por ejemplo se empezó con el prefijo /16 y se agregaron más establecimientos lo que podría ocasionar problemas a medida que las redes aumentan su tamaño debido a que todos los hosts deben tener el mismo prefijo. La solución a esto es dividir el bloque de direcciones en varias partes para uso interno en forma de múltiples redes, pero actuar como una sola red para el mundo exterior. A estas partes de la red se les llama subredes.

Como yo quiero que vean una sola red divido /16 que son 65536 hosts en 3 redes por ejemplo y a una de esas redes le doy 32768 hosts cambiando la máscara de subred /17 y a otra le doy 16384 hosts también cambiando sus máscaras subred /18 y a otras dos le doy 8192 hosts cambiando la máscara de subred /19. En definitiva lo que hago es rebanar los bits de mis hosts. El router que interconecta las distintas subredes debería tener en vez de una sola tabla tiene 3. Este router es el que recibe la ruta global.

CIDR: Enrutamiento Interdominio sin Clases

Consiste en hacer que las tablas de enrutamiento tengan la menor cantidad de entradas posibles. Lo que hago es tomar varias redes adyacentes y las uno con un prefijo lo cual, al hacer una AND con esta y la máscara de subred, me da la dirección por la cual me tengo que enrutar, debido a esto no tengo cada una de las redes adyacente en la tabla de enrutamiento del emisor, sino que tengo una pudiendo llegar a las 3 redes con una misma dirección IP. El router que conecta estas 3 redes debe tener la información del prefijo que corresponde a cada subred. Esto se llama agregación de ruta (ruteo sin clase).

Direccionamiento con clases y especial

Éste es un diseño jerárquico, pero a diferencia de CIDR, los tamaños de los bloques de direcciones son fijos. En realidad, una dirección clase B es demasiado grande para la mayoría de las organizaciones. Para lidiar con estos problemas se introdujeron las subredes para asignar de manera flexible bloques de direcciones dentro de una organización. Después se agregó el CIDR para reducir el tamaño de la tabla de enrutamiento global. Actualmente, los bits que indican si una dirección IP pertenece a una red clase A, B o C ya no se utilizan.

Capa de transporte

El objetivo fundamental es hacer una multiplexación con las distintas aplicaciones que quieren usar la red. Poder de alguna manera que los procesos de aplicación usen el servicio de la capa de red y poder especificar el procesos del otro lado. El hardware o software que se encarga de esto se llama **entidad de transporte** que está en el kernel de SO el cual se encuentra en una biblioteca que pertenece a las aplicaciones de red.

La capa de transporte constituye el **límite principal** entre el proveedor de servicios de transporte (Capa 1 a 4) y el usuario del servicio de transporte (Capa superiores). La interfaz de transporte debe permitir que los programas de aplicación establezcan, usen y después liberen las conexiones.

Algo fundamental es que la capa de transporte considera todas las redes intermedias como si fuera una capa de enlace, esto quiere decir que aseguro el control de errores, pero de extremo a extremo incluyendo varios enlaces no uno solo como en capa de enlace, pero esto se aplica para algunos protocolos.

Así, los segmentos (intercambiados por la capa de transporte) están contenidos en paquetes (intercambiados por la capa de red). A su vez, estos paquetes están contenidos en tramas (intercambiadas por la capa de enlace de datos). Cuando llega una trama, la capa de enlace de datos procesa el encabezado de la trama y, si la dirección de destino coincide para la entrega local, pasa el contenido por el campo de carga útil de la trama a la entidad de red. Esta última procesa de manera similar el encabezado del paquete y después pasa el contenido de la carga útil del paquete a la entidad de transporte.

Direccionamiento

Dos servidores no pueden estar asociados a un mismo puerto, pero dentro de la misma red podría tener dos aplicaciones que usen el mismo puerto porque se diferencian con el NSAP (Punto de acceso al servicio de red).

Para realizar la conexión se lleva a cabo los siguientes casos:

1. Un proceso servido se enlaza con un TSAP(2) para esperar una llamada entrante
2. Un proceso de aplicación (correo) se enlaza con un TSAP(1) manda una solicitud e indica TSAP(2) como destino y como origen al TSAP (1). Esto provoca una conexión entre el proceso de aplicación y el servidor.
3. El proceso de aplicación envía el correo.
4. El servidor de correo responde para decir que entregará el mensaje.
5. Se libera la conexión de transporte

Establecimiento de una conexión

Protocolo de envió y parada: consiste en que si un segmento se pierde o llega mal se hace un acuse de recibo. Como el emisor tiene un timer, pasado este tiempo si vuelve a falla, lo va a reenviar n veces hasta que le llegue un acuse de recibo. Con esto aseguro una transmisión sin errores, pero desperdicio el canal y además puede pasar que el emisor reciba paquete duplicados si se pierde el acuse de recibo y el paquete se toma como nuevo debido a que no es la capa de red quien lo transmite si no la de transporte, entonces cuando lo reenvía no utiliza el mismo paquete IP y no tiene el mismo numero de secuencia que el anterior se utiliza el siguiente protocolo.

Protocolo de acuerdo de las 3 vías

En este caso se enumera el segmento con un número de secuencia y se hace acuse de recibo bidireccional. Consiste en que el emisor envía un CR (Connection Request) el cual lleva un numero de secuencia y el receptor responde con un ASK (acuse de recibo) y asigna su numero de secuencia en la solicitud para que la confirme el emisor y se pueda empezar a mandar los datos. Tenemos varios casos:

- ✓ CR 8tardío: cuando se llega al receptor una solicitud CR tardía, este va a responder, pero el N° de secuencia va ser distinto por lo tanto el emisor se da cuenta y lo rechaza debido a que no acordaron esto en un principio, entonces el receptor se a cuenta que fue engañado por un **segmento duplicado con retardo** y lo rechaza.
- ✓ CR y Datos tardío: mismo caso de arriba para el CR tardío y para los datos tardío va a pasar algo similar y es que el numero de secuencia del receptor no es el acordado con el emisor por lo tanto lo descarta .

Liberación de una conexión

El sistema operativo tiene una **máquina de estados** una vez que me conecto y guarda información de la conexión y en qué estado esta, entonces si dejo que pase el tiempo esta crece pudiendo llegar a quedar sin lugar, por lo tanto tenemos que liberar la conexión.

- ✓ **Liberación asimétrica:** si una corta la conexión, por más de que la otra estaba enviando datos se interrumpe.
- ✓ **Liberación simétrica:** trata la conexión como dos conexiones unidireccionales distintas y requiere que cada una se libere por separado. Esta tiene el problema de los dos ejércitos que es básicamente si ninguna está segura de la desconexión de la otra, sigue indefinidamente. Para solucionarlo se hace lo siguiente:
Se envían ambos usuarios DR (Disconnection Request) y el ultimo que lo recibe manda un ACK para la confirmación de que le llegó el DR y que el otro pueda liberar la conexión. Podemos tener 3 casos:
 - ❖ Se pierde el ACK: no le llega al receptor por lo tanto si expira un timer se libera la conexión.
 - ❖ Se pierde un DR: expira un temporizador y se vuelve a mandar.
 - ❖ Se pierden los DR indefinidamente: el emisor después de N intentos libera la conexión y expira el tiempo en el receptor también liberando la conexión.

Control de errores y almacenamiento en búfer

El emisor y receptor negocian el tamaño de la ventana y el tamaño es el que puede soportar el receptor. El protocolo que se utiliza son los siguientes:

- ✓ **Ventana deslizante variable:** tengo control de flujo y de congestión. Consiste en que el emisor es el que solicita una cantidad de espacio en el búfer y el receptor da los que puede, esto se avisa a través de los ACK de los segmentos que van llenando el buffer y le avisa cuantos lugares quedan en el al emisor. Cuando se acaban los lugares se espera a que se procesen los datos y la ventana se desliza la cantidad de lugares que tiene disponibles y esta va variando si muchas aplicaciones empezaron a utilizar la red.
- ✓ **Ventana deslizante:** esta es para el aprovechamiento del canal y se desliza en una proporción fija si todos los segmentos llegan bien, pero si alguno se pierde el receptor me manda la cantidad de acuses de recibo que me llegaron bien, pero del **ultimo** que me llegó **sin huecos** al medio anterior a este. Entonces como le llega acuses duplicados de este sabe que llegó bien y desplaza la ventana hasta el primer hueco de segmento que no recibió el receptor y vuelvo a mandar en el orden del ultimo al hueco por esto expira el timer y manda de nuevo un acuse de recibo del anterior ya que cree que sigue el hueco de antes hecho esto manda el ACK del último que llegó sin huecos en el medio para saber hasta dónde desplazar la ventana.

Multiplexación

Es un mecanismo para saber a cuál proceso asignar un segmento recién llegado. También si se necesita un mayor BW y confiabilidad de la que proporciona una red, existe la **multiplexación inversa** que distribuye el tráfico entre **varias trayectorias de red** por round-robin.

Recuperación de fallas

Si a mitad de la transmisión falla el servidor. Al reactivarse, sus tablas se reinician, por lo que ya no se sabe en dónde se encontraba. Para solucionar esto el servidor envía un segmento de difusión para solicitar a sus clientes que le informen el estado de sus conexiones, en este trayecto puede pasar que el servidor mande el ACK y le llegue al cliente (Pasa de S1 a S0) y luego ocurra la falla antes de completar la escritura en el servidor. Como el cliente está en S0 no retransmitirá y pensará que llegó el segmento erróneamente. La recuperación de una falla en la capa **N** sólo se puede llevar a cabo en la capa **N+1**, y esto es sólo si la capa superior retiene suficiente información del estado como para reconstruir la condición en la que se encontraba antes de que ocurriera el problema. La capa de transporte puede recuperarse de fallas en la capa de red, siempre y cuando cada extremo de una conexión lleve el registro del estado en el que se encuentra.

Los protocolos de transporte

Los segmentos que tengo que enviar para un protocolo con conexión son 8, 3 para el acuerdo de las 3 vías, 2 lo que voy a mandar y lo que voy a recibir y 3 más para desconexión. Por esta razón UDP es mucho más rápido y solo uso dos segmentos y si se me pierde expira el timer y la aplicación manda otro segmento. Se usa mucho para protocolos que son liviano y no requiere transmitir mucha información.

UDP (Protocolo de datagrama de usuarios): proporciona una forma para que las aplicaciones envíen datagramas IP encapsulados sin tener que establecer una conexión. Es decir, no hace nada más que enviar paquetes entre aplicaciones. UDP transmite segmentos que consisten en un encabezado de 8 bytes seguido del payload. Los dos puertos sirven para identificar los puntos terminales dentro de las máquinas de origen y destino. cuando llega un paquete UDP, su payload se entrega al proceso de aplicación que está conectado al puerto de destino en la capa de ampliación. Con los campos de puerto, la capa de transporte entrega el segmento incrustado a la aplicación correcta. Proporcionar una **interfaz para el protocolo IP** con la característica agregada de **demultiplexar** varios procesos mediante el uso de los puertos y a detección de errores extremo a extremo opcional.

Puerto de origen: se necesita cuando hay que enviar una respuesta al origen. Es opcional es decir si el puerto origen no está especificado los 16bits se pondrán en 0, por lo que el destinatario no podrá responder.

Longitud UDP: incluye el encabezado de 8 bytes y los datos. La longitud mínima es de 8 bytes y la máxima de 65515 bytes, lo cual es menor al número que cabe en 16 bits debido al límite de tamaño en los paquetes IP.

Suma de verificación: es opcional para proporcionar una confiabilidad adicional.

TCP (protocolo de control de transmisión): utiliza el protocolo de ventana deslizante variable, también utiliza el protocolo de 3 vías, con una distinción que en la conexión se espera recibir un segmento con incrementos $ACK = k+1$ y $SEC = x+1$, junto con este se manda el tamaño de ventana. Se desglosa en los siguientes campos:

Puerto origen y puerto destino: identifican los endpoint locales (IP más Puerto) de 48bits

Nº de secuencia y Nº de confirmación de recepción: el 2º especifica el **siguiente byte** en el

Longitud del encabezado: comienzo de los datos en el segmentos.

Bits sin uso: reservado para corregir errores.

Bit ECE: este bit si está en 1 indica que hay **congestión** y se usa ECN (Notificación de congestión) que le avisa al receptor que hay congestión y este envía la notificación al emisor que reduzca su velocidad.

Bit CWR: indica una reducción en la ventana de congestión en el receptor para que deje de mandar ECN.

Bit URG: si es 1 uso un puntero urgente que me da un offset en la parte de datos para ejecutar información.

Bit ACK: si está en 1 indica que el número de confirmación de recepción es válido, es 0 al iniciar la conexión.

Bit PSH: indica que no llene el búfer para poder pasar los datos.

Bit RST: restablecer de manera repentina una conexión

Bit SYN: hace una conexión ($SYN=1 - ASK=k$) ; ($SYN=1 - ASK=k+1$) ; ($SYN=0 - ASK=k+1$) $SYN=0$ es acuse nomas.

Bit FIN: liberar una conexión

Tamaño de ventana: indica la cantidad de bytes que se pueden enviar.

Suma de verificación: realiza la operación con el encabezado, los datos, y un pseudoencabezado (Nº protocolo y dirección IP origen y destino)

Opciones:

Para la transmisión de segmentos como es bidireccional ambos sentidos especifican las ventanas para así regular el contenido que manda tanto servidor como cliente.

TCP no enumera los segmentos si no los bytes, entonces por cada bytes tengo un numero de secuencia. Como TCP es con flujo de bytes el receptor almacena todo en un búfer y no se entera de en cuantas partes se envió.

Todas las conexiones TCP son full dúplex y de punto a punto.

TCP no soporta la multidifusión ni la difusión.

Cada máquina que soporta TCP tiene una **entidad de trasporte TCP:**

- ✓ Procedimiento de biblioteca.
- ✓ Proceso de usuario.

Establecimiento de una conexión

Se establecen mediante el acuerdo de tres vías:

- ✓ Se espera en forma pasiva del lado del servidor con las primitivas LISTEN y ACCEPT ya se creó el socket.
- ✓ El cliente ejecuta la primitiva CONNECT en donde especifica la dirección IP del Host y el puerto y el límite del segmento TCP. Este segmento tiene **SYN = 1 y ASK = 0** y espera una respuesta.
- ✓ Cuando llega al destino la entidad TCP se fija si hay algún proceso que ejecuto la primitiva LISTEN en el puerto destino que se indica.
- ✓ Si algún proceso está escuchando en el puerto, ese proceso recibe el segmento TCP entrante y puede entonces aceptar o rechazar la conexión. Si la acepta, se devuelve un segmento de confirmación de recepción.

Liberación de la conexión

Consiste en 4 pasos, en los cuales el emisor manda un segmento FIN = 1 y recibe la confirmación del receptor liberando su conexión, si el receptor tiene que seguir mandando datos lo hace y cuando termina envía un segmento FIN = 1 y espera el ACK para liberar su conexión.

Capa de Aplicación

Word wide web

Cada página puede tener vínculos a otras páginas en cualquier parte del mundo. La idea de hacer que una página apunte a otra se conoce como **hipertexto**. Las páginas se ven mediante un programa llamado navegador.

A una pieza de texto, icono, imagen u otro elemento asociado con otra página se le conoce como **hipervínculo**. Para obtener cada página, se envía una solicitud a uno o más servidores, los cuales responden con el contenido de la página. El protocolo de solicitud-respuesta para obtener páginas es un protocolo simple basado en texto que se ejecuta sobre TCP. Este protocolo se llama HTTP (Protocolo de Transferencia de HiperTexto).

El lado del cliente

Un navegador es un programa que puede mostrar una página web y capturar clics del ratón para los elementos de la página visualizada. Al seleccionar un elemento, el navegador sigue el hipervínculo y obtiene la página seleccionada. Pero para desplegar una página había que responder las siguientes preguntas:

- ❖ ¿Cómo se llama la página?
- ❖ ¿En dónde está ubicada?
- ❖ ¿Cómo se puede acceder a ella?

Para responder estas 3 preguntas a la vez, a cada página se le asigna una URL (localizador uniforme de recursos) estos tienen 3 partes:

<http://www.cs.washington.edu/index.html>

1. El protocolo **http**
2. Le nombre de DNS de la maquina en la que se encuentra la página **www.cs.washington.edu**
3. La ruta que indica de manera única la página específica. La interpretación depende del servidor. **index.html** este archivo esta en un directorio de UNIX en teoría debería ser HTML.

Pasos para obtener una página cuando se hace click en un hipervínculo:

1. El navegador determina el URL
2. El navegador pide al DNS la dirección IP del servidor
3. El DNS responde con la dirección IP
4. El navegador realiza una conexión TCP con esta dirección y el puerto 80.
5. Después envía una solicitud HTTP para pedir la página /index.html.
6. El servidor envía la página con una respuesta HTTP
7. El navegador despliega la página /index.html
8. Se liberan las conexiones TCP

El lado del servidor

Como vimos antes, cuando el usuario escribe un URL o hace clic en una línea de hipertexto, el navegador analiza el URL e interpreta la parte entre `http://` y la siguiente barra diagonal como un nombre DNS que debe buscar. Equipado con la dirección IP del servidor, el navegador establece una conexión TCP con el puerto 80 de ese servidor. Después envía un comando que contiene el resto del URL, que es la ruta a la página en ese servidor. A continuación, el servidor devuelve la página para que el navegador la despliegue en pantalla.

1. Aceptar una conexión TCP de un cliente
2. Obtener la ruta a la página, que viene siendo el nombre del archivo solicitado.
3. Obtener el archivo (del disco).
4. Enviar el contenido del archivo al cliente.
5. Liberar la conexión TCP.

Tenemos varios problemas:

- ❖ Las lecturas de disco son muy lentas en comparación con la ejecución de un programa. Para esto una mejora que utilizan todos los servidores web es mantener una caché en memoria de los archivos leídos más recientes.
- ❖ Otro problema es que sólo se procesa una solicitud a la vez. Para esto una estrategia es hacer al servidor multihilos. En un diseño el servidor consiste en un módulo de front-end que acepta todas las solicitudes entrantes y k módulos de procesamiento. El front-end pasa cada solicitud entrante al primer módulo disponible, que a su vez la atiende mediante el uso de algún subconjunto de los siguientes pasos. Esos pasos ocurren después de haber establecido la conexión TCP y algún mecanismo de transporte:
 1. Resuelve el nombre de la página web solicitada.
 2. Realiza el control de acceso en la página web.
 3. Verifica la cache.
 4. Obtiene del disco la página solicitada o ejecuta un programa para construirla.
 5. Determina el resto de la respuesta.
 6. Regresa la respuesta al cliente.
 7. Realiza una entrada en el registro

Generación de páginas web dinámicas del lado servidor

Cuando se envía una solicitud para un servidor, en esta se envía una URL especificada. Esta solicitud tiene datos que deben ser procesados por un programa. La URL que trae la solicitud especifica el programa que se debe ejecutar y los datos se proveen al programa en forma de entrada. La página devuelta por esta solicitud dependerá de lo que ocurra durante el procesamiento, es decir ser actualizada con los nuevos datos.

La forma en que un servidor ejecuta el programa se logra a través de APIs estándares para que los servidores web invoquen programas, la cuales son:

- ❖ **Primera API:** es un método para manejar solicitudes de páginas dinámicas CGI (Interfaz de Puerta de Enlace Común). CGI provee una interfaz para permitir que los **servidores** web se **comuniquen** con programas de soporte y secuencias de comandos que pueden aceptar entrada.
- ❖ **Segunda API:** incrusta pequeñas **secuencias de comandos** dentro de las páginas de HTML y hace que las **ejecute** el mismo **servidor** para generar la página. Para escribir estas secuencias puede usarse PHP (Preprocesador de Hipertexto).

Las secuencias de comandos de CGI y PHP resuelven el problema de manejar la entrada y las interacciones con bases de datos en el servidor. Pueden aceptar información entrante de formularios, buscar información en una o más bases de datos y generar páginas HTML con los resultados

Generación de páginas web dinámicas en el cliente

Debido a que se necesita responder a los movimientos del mouse del usuario e interactuar de manera directa con los usuarios. Para esto es necesario tener secuencias de comandos incrustadas en páginas HTML que se ejecuten en la máquina cliente y no en el servidor. Tales secuencias son posibles mediante la etiqueta.

Aunque PHP y JavaScript se ven similares en cuanto a que ambos incrustan código en archivos HTML, se procesan de una manera totalmente distinta.

- ❖ **PHP:** cuando el usuario hace click recolecta la información y la manda al servidor como una solicitud de una página PHP. El servidor carga el archivo PHP y ejecuta la secuencia de comandos de PHP que está incrustada para producir una nueva página HTML.
- ❖ **JavaScript:** Todo el trabajo se hace en forma local, dentro del navegador. No hay contacto con el servidor.

Conclusión: el resultado se despliega casi al instante con JavaScript, mientras que con PHP puede haber un retraso de varios segundos antes de que llegue el HTML resultante al cliente. PHP se utiliza cuando se requiere la interacción con una base de datos en el servidor. JavaScript se utiliza cuando la interacción es con el usuario en la computadora cliente.

HTTP: el Protocolo de Transferencia de HiperTexto

Es el protocolo de la capa de aplicación que se utiliza para transportar toda la información entre servidores web y clientes, es decir transfiere las páginas. Es un protocolo simple de solicitud-respuesta que por lo general opera sobre TCP.

Cookies

La cookie es una cadena con nombre muy pequeña que el servidor puede asociar con un navegador. Esta es una solución para HTTP debido a que en este no existe sesión y no tenemos forma de establecer una a través de IP porque podría tener más de un usuario con la misma IP, porque me la podrían robar y porque podría tener más de una máquina con la misma IP debido a NAT. La solución son las cookies ya que cuando me logie por primera vez e hice el saludo de 3 vías, puse mi usuario y mi clave y el servidor en su respuesta me mando un encabezado de se llama cookie y con información para que no tenga que establecer sesión por cada vez que cambio de página.

Las páginas web se puede categorizar en:

- ❖ **Páginas web estáticas:** son sólo archivos guardados en un servidor que se presentan a sí mismos de la misma forma cada vez que un cliente los obtiene y los ve
- ❖ **Páginas web dinámicas:** si se generan bajo demanda mediante un programa o aplicación, este puede correr del lado del cliente o servidor o ambos lados. Gracias a esto los clientes no necesitan actualizar ni instalar nada el servidor se encarga de pasar la aplicación o programa.

Conexiones

La forma más común en que un navegador contacta a un servidor es mediante el establecimiento de una conexión TCP por el puerto 80 en la máquina del servidor.

HTTP 1.1 implementa las conexiones persistentes. Con ellas es posible establecer una conexión TCP, enviar una solicitud y obtener una respuesta, para después enviar solicitudes **adicionales** y obtener respuestas **adicionales**. También es posible canalizar solicitudes; es decir, enviar la solicitud 2 antes de que haya llegado la respuesta a la solicitud 1. Tenemos los siguientes casos:

- ❖ **Conexión separada:** se cierra la conexión después de **cada** solicitud y volver hacer el saludos de 3 vías.
- ❖ **Conexión persistente:** se cierra la conexión después de **varias** solicitudes.
- ❖ **Conexión persistente con canalización:** se cierra la conexión después de varias solicitudes y se puede mandar solicitudes a la **vez** antes que me hayan llegado la respuesta.

Encabezados de mensaje

Cada solicitud obtiene una respuesta que consiste en una línea de estado, y posiblemente información adicional a las línea de solicitud le pueden seguir líneas adicionales que contienen más información. Estas se llaman encabezados de solicitud

Almacenamiento en caché

Al proceso de guardar y ocultar las páginas que se obtienen para usarlas después se les conoce como almacenamiento en caché.

- ❖ Ventaja: cuando se puede reutilizar una página en caché, no es necesario repetir la transferencia.
- ❖ Desventaja: las paginas se mantienen en el disco.

El problema con el almacenamiento en caché en HTTP es cómo determinar que una copia de una página previamente colocada en caché es la misma que se obtendría del servidor otra vez. HTTP utiliza dos estrategias para lidiar con este problema:

1. **Validación de páginas:** se consulta la cache con el encabezado expire que se obtuvo cuando se obtuvo la pagina cache y compararlo con la fecha y hora actual, si esta es valida no hay necesidad de obtenerla del servidor.
2. **Preguntar al servidor si la copia en caché sigue siendo válida:** cuando el cliente hace una solicitud, especificamos el método que usa, el recurso, y la versión de HTTP y encabezados adicionales. Se usa el método GET condicional junto con los encabezados expire y last modified . El servidor me devuelve la fecha de expiración y la última modificación y si no expiro ni se a modificado la mostramos directamente de la cache y si no se manda todo el archivo de vuelta.

Representación finita de números reales

- ✓ **Enteros sin signo:**
 - ✦ **Rango:** 0 a $2^N - 1$
- ✓ **Complemento a 2:** cuando uno opera en complemento a 2 puede usar un único sumador tanto para restar como para sumar.
 - ✦ **Rango:** -2^{N-1} a $2^{N-1} - 1$
El $N - 1$ es debido al bit de signo que se agrega.
- ✓ **Punto fijo:** se llama punto fijo debido a que una vez que definimos cuantos bits son para la parte decimal esto queda *fijo* y no se modifica.
 - ✦ **Notación:**
Donde $N = m + n + 1$; N : número de bits; m : parte entera; n : parte decimal
1: un bits extra para el signo de bits (debido a este bits es $N-1$).

La ALU no distingue si está operando números en formato **entero** o en formato **punto fijo**, no existe un registro para tal fin. **Ambas** variables se definen como un **entero**, pero se diferencian por un factor de escala que está en la cabeza del programador y depende de un formato que se define como **Qm.n** y que esta en la cabeza del programador, es decir un número en punto fijo Qm.n simplemente puede verse como un entero multiplicado por 2^{-n} que es el **factor de escala** el cual es igual a la precisión. También gracias a este formato esta la diferencia, ya que podemos tener diferentes formatos que dan el mismo valor en punto fijo (el formato Qm.n no toma el bit de signo).

- ✦ **Rango:**

$$-2^m a 2^m - 2^{-n}$$

Para el caso de $n = 0$, es decir sin parte fraccional mi precisión pasaría a ser 1 y estaría en el caso de **complemento a 2**.

- ✦ **Precisión:** es constante en todo el rango a representar 2^{-n}
- ✦ **Rango dinámico:** Nos da una idea de cuanto números podemos representar con una cantidad definida de bits esta dado por:

$$DR_{dB} = 6.02 * (N - 1) [dB]$$

- ✦ **Escala de representación:** esta tiene límites y se pueden producir dos variantes:
 - ❖ **Overflow:** esto se da cuando el resultado esta fuera del rango que posibilita el número de bits y esto se da cuando los dos últimos bits del acarreo son *distintos*. Para evitar el overflow tenemos que una de las maneras es con la **aritmética de saturación** que limita al valor máximo cuando se produce overflow. Otra manera de evitarlo es guardando el resultado de N bits en un acumulador de $N+1$, en donde el tamaño de $N = m + \log_2(s)$ (m : número de bits de los números a sumar y s cantidad números).
 - ❖ **Underflow:** se da cuando la precisión de el resultado de algún cálculo no esta al alcance de la precisión del formato Qm.n que estamos trabajando, por ejemplo cuando multiplicamos 2 números la precisión se **duplica** no pudiendo representar todos los bits de la parte fraccional por lo tanto se recurre a dos métodos truncación y roundoff.

- ✦ **Operaciones:**

$$1 \ll n = 1 * 2^n \quad \text{para } n = 4 \quad x = 1.0000$$

$$1 \ll (n - 1) = 1 * 2^{(n-1)} \quad \text{para } n = 4 \quad x = 0.1000.$$

A estos valores se llega debido a que con el formato Qm.n yo estoy **fijando el punto** por lo tanto el 1 yo lo represento antes de definir el formato como 000000001. y si corro n lugares el 1 me queda en la parte entera siendo el valor de 1 y si lo corro $n - 1$ me queda un lugar antes que es el primer bits de la parte fraccional por lo tanto es 0.5.

Pasar de Fx2Fp: multiplicar por $2^{-n} \rightarrow X / (1 \ll n)$

Pasar de Fp2Fx: multiplicar por $2^n \rightarrow X (1 \ll n)$

- ❖ **Desplazamientos:** desplazar hacia la derecha divide por 2 y desplazar hacia la izquierda multiplico por 2. Tenemos dos tipos de desplazamientos:
 - **Desplazamientos lógicos:** reemplaza por cero el bit más significativo cometiendo errores cuando el número es negativo.
 - **Desplazamientos aritméticos:** reemplaza el bit más significativo por el bit que tenía anteriormente evitando errores.

✦ **Esquemas de redondeo:**

- ❖ **Truncacion:** elimina los bits extra de la multiplicación y toma siempre el valor que esta hacia la izquierda, esto se lo denomina también redondeo hacia $-\infty$. Tiene el valor de **media** igual a **constante**
- ❖ **Round off:** este es el que menor error presenta y al valor obtenido se le suma la mitad de la precisión $2^{-n}/2$, entonces cuando se hace esto el resultado da a la derecha y luego cuando se aplica truncacion va a tomar el valor de la izquierda, pero en principio tomo el valor de la derecha. Otra manera de verlo es que se le suma un 1 en la posición n-1. Tiene el valor de **media** igual a **cero**.

- ✓ **Punto flotante:** surge del estándar IEEE 754 la cual contempla punto flotante en base 2 y en base 10 también el estándar cubre esquemas de redondeo operaciones trigonométricas, funciones aritméticas y el manejo de excepciones.

✦ **Notación:**

$$(-1)^S * F * r^E$$

S: bits de signo; F: parte fraccional; r: base del número en este apunte $r = 2$;

E: exponente

- ✦ **Representación:** tenemos varios tipos pero la mas usada es:

❖ **Precisión simple:**

F: 23bits representa potencias negativas de 2.

E: 8 bits no poseen bit de signo debido a que se hace un sesgo (bias) = 127 con esto hago número que vallan de

$$-126 \leq E - bias \leq 127$$

La única representación válida para esto formato **1.x**. El campo F tiene un **1 implícito** que no forma parte de los 32bits. Entonces en el campo F representa lo siguiente:

$$1. [2^{-1}2^{-2}2^{-3} \dots 2^{-23}]$$

A esta manera de representar un numero en punto flotante la IEEE la llama **forma normalizada**. Con el 1 implícito tenemos un problema que es que no se puede representar al cero. Para esto cuando $E = 0$ en lugar de un 1 implícito va a tener un cero implícito. A este formato se lo conoce como **forma desnormalizada**.

- ✦ **Precisión:** no es constante porque a medida que va incrementando el exponente esta va aumentando y eso es malo. Donde tenemos mayor precisión es cerca del cero.
- ✦ **Rango dinámico:** es un número que nos indica la cantidad de números que podemos representar en una computadora.

$$DR_{dB} = 6.02 * 2^{bits(E)}$$

- ✦ **Escala de representación:** Siempre que tengamos 2^E el número mínimo que podemos representar es 1.000 ... 0 y el máximo es 1.111 ... 1 y siempre que queramos saber esto tenemos que multiplicar $2^E * F * (-1)^S$ y a medida que pasamos los exponentes de + a - vemos que los tramos van creciendo. E incluye el sesgo.

✦ **Operaciones:**

Multiplicar x2 incrementamos en 1 el exponentes

Dividir x2 decrementamos en 1 el exponente

Esto es debido a que la mantisa se mantiene siempre igual lo que cambia es el exponente.

Es decir 2^1 o sea 2 por eso incrementar o decrementar % o X.

Los números de punto flotante son **autorango**, porque cuando variamos el exponente nos movemos por diferentes tramos de la recta de los números reales.

Cuando queremos hacer un cálculo y tenemos:

❖ **Mantisas iguales:** antes de sumar por ejemplo debemos hacer los siguiente:

1. Representar en punto flotante.
2. Normalizar.
3. Se calcula el mayor exponente entre los 2 números con $n = (ex1 - ex2)$ y se fuerza al número que tiene menor exponente a n . Cuando hacemos esto el resultado no resulta alterado porque cuando le sumamos n al exponente estamos $\times 2$ y cuando desplazamos hacia la derecha $\div 2$.
4. Se suman las mantisas.
5. Se normaliza y se verifica que este $-126 \leq x \leq 127$.
6. Se verifica si tenemos que hacer redondeo

❖ **Mantisas distintas:** en este caso cuando forzamos el mayor exponente al número con menor exponente **pierde precisión** porque se desplaza a la derecha n bits haciendo que los últimos bits no se consideren

✦ **Esquemas de redondeo:** tenemos que Ulp es la precisión en punto flotante y lo siguientes casos:

- ❖ **Truncacion:** redondeo hacia cero $\rightarrow 0 \leftarrow$ si es un valor (-) toma el de la DER y (+) el de la IZQ
- ❖ **Redondeo a $-\infty$:** siempre toma el número hacia la IZQ
- ❖ **Redondeo a $+\infty$:** siempre toma el número hacia la DER
- ❖ **Redondeo al más cercano:** por defecto la usa la FPU cuando arranca una computadora.

Tenemos que cuando:

$$f > f' + \frac{Ulp}{2} \rightarrow f''$$
$$f < f' + \frac{Ulp}{2} \rightarrow f'$$

Comparación PF y PX

PF: representa un gran rango dinámico, tiempo de desarrollo menor debido a que no tenemos que verificar si se produce un overflow o un underflow.

PX: la precisión es constante en todo el rango y los microprocesadores que solo soporta punto fijo son más baratos.

Procesamiento digital de señales

Cuando se desea trabajar con una señal analógica en forma discreta vamos a necesitar 3 **bloques**:

- ✓ Conversor analógico digital con $T = 1/fs$ $x_c(t) \rightarrow x[n]$
- ✓ Sistema de tiempo discreto (filtro digital típicamente). $x[n] \rightarrow y[n]$
- ✓ Conversor digital analógico con $T = 1/fs$ $y[n] \rightarrow r(t)$

Las etapas de DSP son:

- ✓ **Filtro antialiasing:** Es un filtro analógico y siempre esta antes que el CAD es típicamente pasa bajos. Su función es garantizar que la máxima señal que entra al CAD es Ω_N , entonces la frecuencia de corte del filtro es Ω_N . Alias viene de que vemos una frecuencia que no es si no cumplimos con el teorema de N/S es decir si no lo cumplimos siempre vamos a ver la diferencia $fs - fn$ V. Se implementa con circuitos acondicionadores de señal:
 - ✦ **Filtro pasa bajos:** solo filtro pasivo (circuito RC)
 - ✦ **Filtro pasa altos:** con un divisor de tensión para el offset para que el CAD vea los ciclos (+) y (-).
 - ✦ **Filtro compuesto:** el cual tiene offset y amplificación, alta impedancia y el mismo filtro (Filtro pasivo), por otra parte, tengo un filtro activo. Podemos elegir uno de los dos filtros para seleccionar la frecuencia de corte.

Oversampling: Como el filtro antialiasing que es un filtro pasa bajos tiene que ser muy abrupto para filtrar el ruido en el dominio del tiempo y solo dejar pasar a Ω_N , esto es muy difícil de obtener. Con oversampling lo que se **logro** es utilizar un filtro antialiasing más **simple** (circuito RC) combinado con un aumento en la frecuencia de muestreo $T = 1/(M * fs)$ del CAD, ya que si esto no se hace los espectro se solaparan con el ruido que pasa debido a que el filtro antialiasing es simple, y aumentando fs lo que se solapa es el ruido, después en el dominio de la frecuencia vamos a utilizar un filtro digital con una frecuencia de corte **abrupta** después si queremos podemos bajar la frecuencia original para que el resto del circuito siga trabajando con esta, pero esto es opcional.

✓ **Sample and hold y conversor A/D:**

Sample and hold: mantiene un tiempo determinado un valor de tensión eléctrica a la entrada del CAD porque la conversión del CAD toma un tiempo. Entonces S/H mantiene el tiempo determinado **finito** que tarda el CAD en tomar la muestra.

Conversor AD: lo vamos a dividir en 3 etapas:

- ✦ **CAD ideal:** toma la muestra de tensión $x[n]$
- ✦ **Cuantizador:** a $x[n]$ le da un valor posible dentro de los posibles valores que depende de los bits del conversor analógico digital.
- ✦ **Coder:** toma el string de bits y los acomoda a un tipo de sistemas de numeración (Punto fijo).

Vamos a llamar $Q(x)$ a la función del cuantizador donde su respuesta es una escalera donde un escalón equivale a: $\Delta = \frac{2V_p}{2^B}$. El error varía de Δ a $-\Delta$, el hecho de cuantizar a una señal le estamos **metiendo ruido**, esto se modela como $x[n] + e[n]$.

$$SNR_{ADC} = 10 \log_{10}(LF) + 4.77 + 6.02 \cdot B$$

Para aumentar la SNR LF (V_{rms}/V_p (ADC)) debe ser grande y para esto debe aumentar la potencia efectiva de la señal para que sea más grande que V_p (ADC), pero a veces no es posible debido a que los CAD son de baja potencia y la potencia efectiva va de la mano con la tensión pico, si crece una crece la otra. Esto nos dice que es **mala idea atacar** al conversor AD con una señal de **mV** debido a que baja la SNR del conversor. Para ver los bits que se dedica a ver el Ruido podemos calcularlo como:

$$B = \frac{SNR_{signal} - SNR_{ADC}}{6}$$

La cantidad de B que nos da determina cuantos bits van a ser dedicados para ver el ruido.

Se tiene que cumplir que:

$$SNR_{ADC} \geq SNR_{signal}$$

Si esto no se cumple entonces lo que da B es el número de bits extra que se requiere y el 6 es debido a que cada 6dB tengo un bits. Siempre tenemos que estar 6dB por encima de SNR_{signal} (los bits dedicados al ruido son eliminados).

- ✓ **Conversor D/A:** recibe un número binario en punto fijo y lo transforma en un valor de voltaje, para aumentar la resolución tenemos que aumentar el número de bits (A veces conviene que la frecuencia del DAC sea mayor que ADC). El DAC **decodifica** la secuencia de bits y los convierte en un tren de impulsos, este tren de impulsos contiene espectros duplicados con una distancia entre ellos de f_s . La señal original es reconstruida pasando estos impulsos a través de un filtro pasa bajos con $f_{cot} = f_s/2$.

A la salida del DAC tenemos otro ZOH que recibe los impulsos y los detiene hasta que se obtenga un nuevo valor, lo sostiene un periodo de muestreo. Lo que hace es **interpolar** (rellena hasta el siguiente muestreo). Como el zero order holder tiene una respuesta sinc estaría filtrando uno de los espectros espejados a la salida del DAC.

Resumiendo:

- ❖ DAC: a la salida tenemos un tren de impulsos
- ❖ ZOH: tenemos una señal analógica escalera
- ❖ Filtro reconstrucción

Como la convolución en el dominio del tiempo implica la multiplicación en el dominio de la frecuencia estas 3 respuestas se multiplican.

- ✓ **Filtro reconstrucción:** filtro pasa bajos con una respuesta que es la **inversa** de la caída que introduce el sinc del ZOH. Puede haber 3 maneras de construir el FR:
 - ❖ **Ignorar:** es decir tolerar la caída de los 3dB
 - ❖ **Precualizar:** antes de entrar al DAC implementamos un filtro de reconstrucción, esto compensa la caída de los 3 dB y la respuesta a la salida del ZOH ahora es casi plana.
 - ❖ **Postecualizar:** ponerlo a la salida filtro analógico, pero es muy difícil de construir

- ✓ **Oversampling:** consiste en aumentar la frecuencia de muestreo en el DAC para modificar la respuesta en frecuencia del ZOH. Esto es posible debido a que un impulso se duplicara 8 con igual amplitud dependiendo de f_s hasta su próximo impulso con diferente amplitud, esto hace que a la salida del ZOH que la interpolación que realiza entre impulsos es muy pequeña formando pulsos muy angostos dando una repuesta **sinc mucho más ancha**. Esto hace que podamos usar un filtro pasa bajos con una respuesta plana ya que al ensancharse el sinc no tenemos la atenuación que teníamos antes.

Filtros de respuesta finita al impulso (FIR)

Es un filtro pasa bajos ya que elimina componentes de alta frecuencia (Ruido) y la respuesta en fase de la banda pasante es lineal. Representa un cuadrado en el dominio del tiempo entonces su respuesta en frecuencia es un sinc. A medida que aumenta el orden del filtro lo que ocurre es:

- ✓ Aumenta el retraso a la salida del filtro de la señal original (El retraso es $N/2$)
- ✓ Disminuye la frecuencia de corte, es decir la respuesta en frecuencia se hace más selectiva.
- ✓ Aumenta el promedio que saca debido a que el N son los puntos que toma, si fuera todo el ancho seria un nivel de continua.

Cuando queremos filtrar en el dominio de la frecuencia queremos eliminar frecuencias y quedarnos con cierto rango y cuando queremos eliminar filtrar en el dominio del tiempo queremos eliminar ruido de la señal (suavizar la señal). Esto depende de donde este la información.

Como el FIR la información esta contenida en el dominio del tiempo, los filtros en el dominio del tiempo **solo** se pueden implementar con **filtros digitales**. Con un solo filtro no se puede tener un buen filtro tanto en el dominio de la frecuencia como en el dominio del tiempo.

Filtro Moving Average

Es un filtro que se emplea en el dominio del tiempo y se define como la convolución entre la señal de entrada y un pulso rectangular de área 1 (conocido como promedio local) su RF es un sinc debido a que es un rectángulo en el dominio del tiempo y esto da un mal filtro en el dominio de la frecuencia. El uso de este filtro introduce un retraso de $N/2$ muestras, entre la entrada y la salida. Para calcular a N sin que elimine componentes de alta frecuencia que son parte de la señal debido a que se vuelve muy selectivo a media que aumenta se calcula como:

$$N_{max} = \sqrt{\frac{0.885894^2 \cdot f_s^2}{f_{co}^2}} - 1$$

$$h[n] = \frac{1}{N} \quad 0 \leq n < N$$

$h[n]$ es el **kernel** del filtro (rectángulo de área 1 con amplitud $1/N$), es un vector de N elementos que contienen a los coeficientes del filtro M/A.

Podemos decir que es un filtro FIR porque $y[n]$ no depende de salidas pasadas podemos estar seguro que no hay realimentación.

Filtros por método de ventana

Estos filtros se implementan en el **dominio de la frecuencia**.




Cuando la información está contenida en el dominio de la frecuencia, esta información va a estar representada por la respuesta en frecuencia de la señal por la fase y amplitud de la misma.

Para el análisis en frecuencia se requieren muchas muestras y en el tiempo con una sola basta para obtener la información.

Como el sinc (dominio del tiempo), la representación requiere una implementación de un filtro con infinitos coeficientes esto no es posible ya que se requiere infinita memoria. Se llega a una solución de compromiso que es **truncar al sinc** quedándonos con m elementos, pero ahora la respuesta en frecuencia tiene ripple en las dos bandas y esto se lo conoce como el fenómeno de Gibbs, para mejorar esto se multiplica al sinc por una ventana, esto hace que el sinc ya no tenga mas un corte abrupto disminuyendo el ripple en el dominio de la frecuencia, pero ahora el filtro tiene una **banda de transición** (ancho de banda entre la banda pasante y la atenuada).

Tipos de ventanas:

- ✓ **Hamming**: tiene la ventana **mas angosta** un 20% respecto a Blackman
- ✓ **Blackman**: tiene -73dB **más de atenuación** respecto a Hamming en el primer lóbulo lateral
- ✓ **Kaiser**: es una ventana de ventanas. Se configura eligiendo el orden de la ventana m y el valor de β , con este β se modifica la ventana a distintas ventanas estándares. A medida $\beta \uparrow$ la ventana se va haciendo **mas angosta** y también va aumentando la atenuación, y si $M \uparrow$ vamos a ver que aumenta $fco \uparrow$

Ventanas	BWT	Ripley	A(dB)
Rectangular			
Hanning			
Hamming			
Blackman			

Respuesta al impulso de un filtro FIR se representa como:

$$H(z) = b_0 + b_1 Z^{-1} + \dots + b_M Z^{M-1}$$

Z^{-1} representan retrasos en el dominio del tiempo discreto y la potencia la cantidad de retrasos

Respuesta infinita al impulso (IIR)

Leakey integrator

$$\lambda = \frac{M-1}{M}$$

Ecuación en diferencias con coeficientes constantes:

$$Y[n] = \lambda y[n-1] + (1-\lambda)x[n]$$

Esta expresión no pertenece a un filtro FIR debido a las salidas pasadas. También esta expresión se la conoce como un **filtro recursivo de polo simple** y también representa un sistema lineal e invariante en el tiempo LTI. El sistema es estable cuando $|\lambda| < 1$ esto siempre se cumple debido a que $\lambda = (M-1)/M$

Leakey integrator es un filtro pasa bajos con una respuesta en fase no lineal. La respuesta en fase es lineal cuando aumenta la frecuencia y la señal sufre desfase negativo, las armónicas sufren un desfase lineal y esto hace que la respuesta a un pulso sea simétrica. Cuando es no lineal los cosenos y los senos (armónicos) están desfasados y la respuesta a un pulso es no simétrica.

Método de la trasformada bilineal

También conocido como método de tustin.

Vamos a comparar el dominio de Laplace con el de z entonces s nos da:

$$s \approx \frac{2}{T} * \left(\frac{1-z^{-1}}{1+z^{-1}} \right)$$

Esta ecuación es la trasformada bilineal. Si reemplazamos S por esta expresión la **digitalizamos**. Debido a esto vamos a tener dos frecuencias:

- ✓ **Analógica** $-\infty < \Omega < +\infty$
- ✓ **Digital** $-\pi < w < \pi$

Para saber la relación entre Ω y w :

Vamos a considerar $S = j\Omega$ donde $Z = r * e^{jw} \rightarrow r = 1$; $Z = e^{jw}$ Reemplazando en la transformada bilineal tenemos:

$$S = j \left[\frac{2}{T} * \tan\left(\frac{w}{2}\right) \right] \rightarrow S = j\Omega$$

$$\Omega = \frac{2}{T} * \tan\left(\frac{w}{2}\right)$$

$$w = \arctan\left(\frac{\Omega T}{2}\right)$$

No consideramos la parte real. Estas dos frecuencias tienen una relación **no lineal**.

Precombado

Se basa en elegir una frecuencia de corte para el filtro **analógico** Ω_c tal que cuando sea digitalizado con la transformada bilineal obtengamos en el dominio discreto la frecuencia de corte w_c que necesitamos. Los pasos son:

1. Elegimos un filtro analógico
2. La frecuencia w_c la normalizamos $w_c = (2\pi f_c)/f_s$
3. Precombamos la frecuencia analógica $\Omega_c = (2/T) * \tan(w_c/2)$ este valor si lo pasamos $f = \Omega_c/(f_s * 2\pi)$ a Hz nos da la frecuencia analógica que tenemos que introducir en el dominio analógico para luego en el dominio digital valga lo que yo quiero.
4. Reemplazamos por la transformada bilineal en nuestro filtro analógico para obtener $H(Z)$.
5. Antitrasformamos para obtener las ecuaciones diferenciales $y[n]$.

Forma directa 1

Representa a $H(Z)$ en bloques. Cada vez que tengo un retraso en un filtro yo tengo que guardar el valor de la variable.

Por cada bloque de Z^{-1} voy a necesitar una variable que me guarde el valor para que en el próximo instante de muestreo yo tenga valores pasados del sistema.

Forma directa 2

Por la propiedad conmutativa de la transformada z pasamos los bloques con los coeficientes **a** hacia la izquierda y los coeficientes **b** hacia la derecha.

Con esto reducimos la forma directa 1 y si unimos las 2 nuevas partes se forma la forma directa 2. Con esto se reduce las variables que necesitaba para guardar en los bloques Z^{-1} .

Esta estructura es importante porque es conveniente representar los filtros IIR separados en diferentes bloques, donde cada bloque es de orden 2. Es decir, el orden del filtro siempre va a hacer $2 * N$ ó sea par.