ASI

Primera Parte: S.O.

3º Ingeniería de Telecomunicaciones — UPV/EHU

"Under-promise and over-deliver."

Javier de Martín – 2016/17

1. Introducción

1.1. Introducción

Un **sistema operativo** es el programa principal del sistema. Tiene como finalidad liberar complejidad del HW a los programadores y aumentar la eficiencia del sistema.

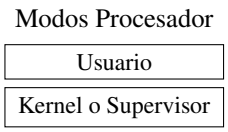


Figura 1: Modos del procesador

El **procesador corre en 2 modos**:

- **Usuario**: Dispone un conjunto limitado de instrucciones.
- **Kernel o Supervisor**: Tiene acceso completo a todo el hardware y puede ejecutar cualquier instrucción que la máquina puede ejecutar

El Sistema Operativo es software que corre en modo kernel.

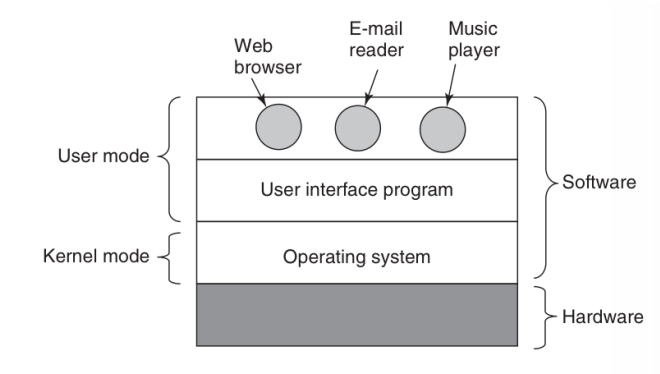


Figura 2: Dónde encaja el sistema operativo

1.1.1. Interface

La **interfaz de usuario** es el nivel más bajo de software ejecutable en modo de usuario.

El sistema operativo corre sobre el hardware y provee la base para cualquier otro software que se ejecute.

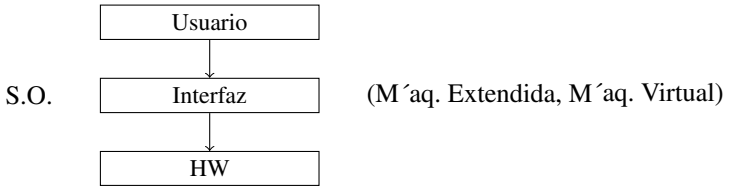


Figura 3: Modos del procesador

1.1.2. Gestor de Recursos

Ofrecer una asignación ordenada y controlada de los procesadores, memorias y dispositivos entre los diversos programas que compiten por ellos.

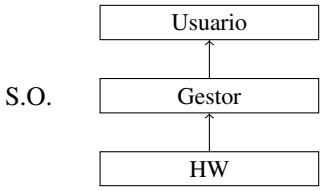


Figura 4: Modos del procesador

1.2. Historia

Aqui naaada

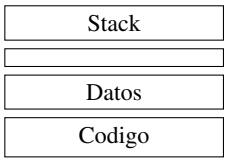
1.3. Conceptos de los S.O.

1.3.1. Introducción

Una **llamada al sistema** es un mecanismo utilizado por una aplicación para solicitar un servicio al sistema operativo.

1.3.2. Procesos

Un **proceso** es un programa en ejecución.



+ Entrada en la tabla de procesos

Figura 5: Espacio de direccionamiento de un proceso

1.3.3. Archivos

2. Gestión de Procesos

2.1. Introducción

Un **proceso** es una instancia de un programa en ejecución. Conceptualmente, cada proceso tiene su propia CPU virtual.

La diferencia entre un proceso y un programa es sutil pero crucial. La idea clave es que un proceso es algo similar a una actividad. Un programa es algo que puede ser guardado en disco no haciendo nada.

Si un programa corre dos veces cuenta como dos procesos.

2.1.1. Modelo

- El **paralelismo** es la ejecución de varias actividades simultáneamente en varios procesadores.
- El **pseudo-paralelismo** es la ejecución de varias actividades en un mismo procesador.

La **concurrencia** es la existencia de varias actividades ejecutándose simultáneamente que necesitan de sincronización para actuar en conjunto.

Cuatro principales eventos provocan la creación de procesos:

1. Inicialización del sistema
2. Ejecución de una llamada al sistema de creación de procesos por un proceso que está corriendo.
3. Una petición del usuario de crear un nuevo proceso.
4. Iniciación de una tanda de trabajos

Un proceso hijo puede formar más procesos, formando una **jerarquía de procesos**. A pesar de que cada proceso es una entidad independiente, puede estar en un estado.

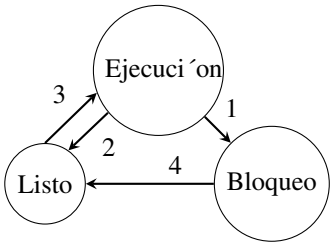


Figura 6: Modelo Básico de los 3 estados de un proceso

Un proceso está en **ejecución** cuando utiliza capacidad de procesamiento de la CPU en ese instante. Se **bloquea** y no puede ejecutarse hasta que un evento externo ocurre. Y está **listo** si ha parado temporalmente para dejar a otro proceso correr.

1. Running (Utilizando la CPU en ese instante).
2. Ready (runnable; temporalmente detenido para dejar correr a otro proceso)
3. Bloqueado[1] (imposible de correr hasta que un evento externo ocurre).

Aquellos procesos que se mantienen por detrás para gestionar actividad se llaman ***daemons***.

Un proceso **terminará** si:

1. Salida normal (voluntaria)
2. Error exit (voluntaria)
3. Error fatal (involuntario)
4. Muerto por otro proceso (involuntario)

La mayoría de procesos termina porque ha terminado su trabajo.

[1] Por ejemplo, cuando un proceso lee de un pipe y no hay nada disponible el proceso se bloquea automáticamente.

2.1.2. Implementación

El Sistema Operativo para implementar el modelo de proceso crea una **tabla de procesos**, que no es más que un array de estructuras (una por cada proceso). Se añaden campos de la estructura de cada proceso en la tabla de procesos y esqueleto de lo que hace el S.O cuando se produce una interrupción.

2.2. Comunicación entre Procesos

2.2.1. Condiciones de Carrera

En algunos sistemas operativos, procesos que trabajen juntos pueden compartir espacio común del cual pueden leer y escribir. Hay que prohibir a más de un proceso leer y escribir de la memoria compartida al mismo tiempo.

2.2.2. Secciones Críticas

La tarea del S.O es prohibir que más de un proceso acceda a una zona compartida de lectura/escritura al mismo tiempo $\rightarrow$ Operaciones "primitivas".

Una **sección crítica** son partes del programa donde se accede a las zonas compartidas.

¿Cómo evitar condiciones de carrera? Conseguir que no coincidan dos procesos dentro de sus regiones críticas a la vez. Condiciones que implican una buena solución:

1. Dos procesos no podrán estar a la vez dentro de sus secciones críticas
2. No se deberán asumir suposiciones sobre velocidades relativas de los procesos
3. Ningún proceso fuera de la sección crítica podrá bloquear otros procesos
4. Ningún proceso esperará indefinidamente para entrar en su sección crítica.

2.2.3. Exclusión Mutua con Espera Activa

La **exclusión mutua**, mientras un proceso está ocupado actualizando memoria compartida en su región crítica, ningún otro entrará en la suya correspondiente. Hay 5 formas:

1. **Inhabilitación de interrupciones**: Solución más simple para sistemas con un único procesador. Cada vez que un proceso entre en la región crítica deshabilitará todas la sinterupciones y las volverá a activar antes de salir. Con las interrupciones desactivadas, no pueden ocurrir interrupciones de reloj. No es óptima ya que no es 'sabio'dar a los procesos la posibilidad de deshabilitar interrupciones.
2. **Variables Candado**: Es una solución por software, se emplea una variable compartida `lock` inicialmente vale `0`. Cuando un proceso quiere acceder a la región crítica.
   - Mira el valor de la variable `lock`
   - Si `lock = 0`: El proceso la pone a 1 y entra en su región crítica.
   - Si `lock = 1` el proceso espera hasta que se cambie a `0`.

Cuando la variable está a `0` no hay ningún proceso en su región crítica y si `1` algún proceso está en su región crítica.

3. **Alternancia Estricta**: Solución por software, evita condiciones de carrera pero no cumple la condición 3.

```
  while(TRUE) {
   while(turn != 1)
critical_section();
turn = 0;
noncritical_section();
  }
```

4. **Solución Peterson**: Combina la idea de tomar turnos con la idea de variables candado.

```
  #define FALSE 0
  #define TRUE  1
  #define N     2

  int turn;
  int interested[N];

  void enter_region(int process) {

   int other;

other = 1 - process;
interested[process] = TRUE;
turn = process;

while(turn == process && interested[other] == TRUE)
  }

  void leave_region(int process) {

   interested[process] = FALSE;
  }
```

5. **Instrucción TSL** (TEST and SET LOCK): Solución por software con ayuda de harware. TSL es una operación indivisible. Copia el contenido de la variable en un registro.

2.2.4. Dormir y Despertar

Peterson y TSL son soluciones buenas pero tienen un defecto, espera activa e incumplen las reglas de planificación.

Las **primitivas de comunicación** son las operaciones para conseguir la condición de exclusión mutua. Bloquean el proceso cuando no se les permite entrar en su región crítica (no consumen tiempo de CPU).

- `SLEEP` es una llamada que suspende el proceso que la realiza hasta que otro proceso lo despierte
- `WAKEUP` es una llamada que tiene por parámetro el proceso a despertar.

El **problema productor-consumidor**. Por ejemplo dos procesos comparten un mismo buffer de tamaño fijo. El productor, pone información en el buffer y el otro, el consumidor la extrae. EL problema nace cuando el productor quiere poner un nuevo item en el buffer pero está lleno. La solución para el productir es dormirse y ser despertado cuando el consumidor haya liberado uno o más items. Similarmente, si el consumidor quiere eliminar uun item del buffer y ve que el buffer está vacío se duerme hasta que el productor pone algo en el buffer y lo despierta.

2.2.5. Semáforos

Solución atómica que resuelve el problema de pérdidas de llamadas de wakeup. Implementado con una variable entera que tiene dos operaciones:

- `DOWN`: Comprueba si el valor del semáforo > 0
  - SI: Decrementa una unidad su contenido
  - NO: Es cero, el proceso se duerme (bloquea)
- `UP`: Comprueba si el valor del semáforo > 0
  - SI: Incrementa una unidad su contenido
  - NO
    - Ningún proceso dormido - Pone a 1 el semáforo
    - Algún proceso dormido - Aleatoriamente se despierta alguno y el semáforo permanece a 0.

Con la implementación de `UP` y `DOWN` el S.O. inhabiblita las interrupciones durante las operaciones a un semáforo.

2.2.6. Contadores de Eventos

El problema productor/consumidor lo solucionan los semáforos con exclusión mutua. Los contadores de eventos sin solución mutua. Se define una variable entera con 3 operaciones:

- `READ (E)`: Devuelve el valor del contador de eventos `E`.
- `ADVANCE (E)`: Incrementa el valor de `E` en `1` (Atómicamente)
- `AWAIT (E, v)`: Espera hasta que `E` $\geq$ `v`

Como característica tiene que los contadores de eventos siempre crecen, nunca decrecen y empiezan desde `0`.

2.2.7. Mutexes

CUando la habilidad de contar de un semáforo no se necesita, se implementa una vesrión simplificada de los semáforos, los mutex. Un mutex es una variable compartida que puede estar en dos estados: bloqueada o desbloqueada.

2.2.8. Monitores

Los **monitores** son objetos destinados a ser usados sin peligro por más de un hilo de ejecución. Sus métodos son ejecutados con exclusión mutua. Lo que significa, que en cada momento en el tiempo, como máximo un hilo estará ejecutando cualquiera de sus métodos. Simplifica el razonamiento de implementar monitores en lugar de código a ser ejecutado en paralelo.

Los procesos no pueden acceder a una estructura de datos interna de un monitor desde una función declarada fuera del monitor. Pueden llamad a las funciones de un monitor siempre que quieran.

Propiedades:

1. Sólo un proceso puede estar activo en un monitor en cualquier instante $\rightarrow$ Exclusión mutua.
2. Permite el bloqueo de procesos dentro del monitor, espera activa, a partir de la utilización de variables de condición y las operaciones `WAIT` y `SIGNAL`.
   - `WAIT` sobre una variable de condición $\rightarrow$ El proceso se bloquea permitiendo a otro proceso entrar en el monitor.
   - `SIGNAL` sobre la variable de condición en la que se ha dormido un proceso $\rightarrow$ Despierta al proceso.

Hay dos formas de evitar que dos procesos permanezcan activos dentro del monitor después de una operación `SIGNAL`:

1. Hoare: Continúa el proceso recién despertado, suspendiendo el otro
2. Hansen: El proceso que realiza `SIGNAL` deberá abandonar el monitor (`SIGNAL` aparecerá únicamente al final de una función monitor.

Tiene un inconveniente, que utiliza lenguaje concepto y el compilador debe reconocerlo.

2.2.9. Transferencia de mensajes

Hay 2 primitivas de intercomunicación entre procesos:

1. `SEND (destino, mensaje)`: Envía un mensaje a un destino dado
2. `RECEIVE (origen, mensaje)`; Recibe un mensaje de un origen dado (Si no hay mensaje disponible el proceso receptor se bloquea hasta que llegue alguno).

Problema productor-consumidor:

1. Pipes: Todos los mensjaes del mismo tamaño, el S.O. almacena mensajes enviados y no recibidos.

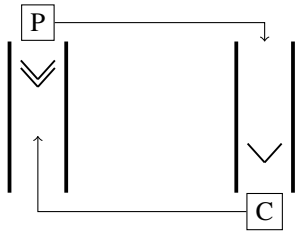


Figura 7: Transferencias de mensajes

2. Buzones (*mailbox*): Emplea un buzón que es una estructura de datos, es una zona de memoria donde caben un cierto número de mensajes. Con buzones, los parámetros de dirección empleados en las llamadas `SEND` y `RECEIVE` son buzones en lugar de procesos.

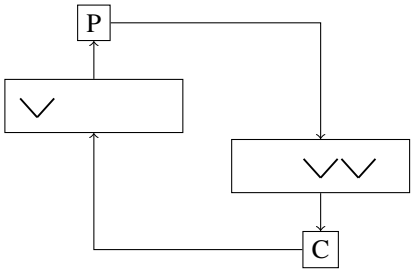


Figura 8: Transferencias de mensajes

   El almacenamiento, el buzón destino coge los mensajes que han sido enviados al proceso de destino pero que aun no han sido recibidos. Cuando un proceso intenta hacer un envío a un buzón que está lleno, es suspendido hasta que se extrae un mensaje.

3. Rendez Vous: Elimina todo el almacenamiento.
   - Si `SEND` antes que `RECEIVE` el proceso `SEND` se bloquea.
   - Cuando se produce `RECEIVE` se copia el mensaje directamente del proceso que lo envía al que lo recibe.

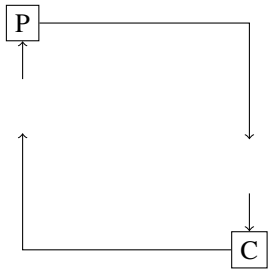


Figura 9: Transferencias de mensajes

2.2.10. Equivalencia de Primitivas

Hay más formas de intercomunicación entre procesos: secuenciadores, expresiones path y serializadores. Todas responden a esquemas similares.

Implementación de monitores a partir de semáfotos:

- Monitor: S(mutex) $\leftrightarrow$ 1Un sólo proceso dentro del monitor
- Variable condición S(c) $\leftrightarrow$ 0 Bloquear/despertar procesos dentro del monitor

La forma de implementar esto:

- Cuandos e llama a una subrutina monitor $\rightarrow$ DOWN (mutex)
- Cuando se abandona una zona monitor $\rightarrow$ UP (mutex)
- `WAIT` a una variable condición `c`:
  - UP (mutex)
  - DOWN semáforo asociado a variable c
  - DOWN (mutex)
- `SIGNAL` a una variable condición c:
  - UP semáforo asociado

Implementación de mensajes a partir de semáforos:

- Proceso: S(P) "0'Bloqueo cuando no se pueda completar `SEND`/`RECEIVE`
- Buzones: S(mutex) "1Ún único proceso inspeccionando los buzones
- Cola Send - Lista de procesos que intentan enviar un mensaje al buzón
- Cola Receive - Lista de procesos que intentan extraer un mensaje del buzón

Forma de interpretación:

1. DOWN (mutex)
2. Mirar variables celdas llenas/vacías
   - Si no vacío o lleno
     - Introducir/extraer mensaje
     - Actualizar variables y lincar
   - Si vacío o lleno
     - `RECEIVE` (vacío):
       - Meterse en la cola RECEIVE
       - UP (mutex) $\rightarrow$ 1 (entrada otros procesos)

        - `DOWN` a su semáforo (bloqueo)
        - DOWN (mutex)
    - `RECEIVE` (lleno)
        - Extraer mensaje, actualizar y lincar
        - Mirar cola SEND, si alguien duerme quitar el primero de la cola y up a su semáforo (desp)
    - `SEND` (lleno)
        - Meterse en la cola SEND
        - UP (mutex)
        - DOWN a su semáforo
        - DOWN (mutex)
    - `SEND` (vacío):
        - Introducir mensaje, actualizar y lincar
        - Mirar la cola `RECEIVE`, si alguien duerme quitar primero de la cola y up a su semáforo

3. UP (Mutex)

Implementación de semáforos a partir de monitores:

- Para un proceso una variable de condición
- Para un semáforo un contador y una lista encadenada

Forma de implementación:

- Procedure `DOWN`
    - if count = 0 → meter el proceso en la lista y WAIT (variable condición)
    - else → decrementar el contador
- Procedure `UP`:
    - if count = 0
        - Comprobar lista
            - ◇ if vacia → incrementar contador
            - ◇ else → eliminar primero de la lista
        - `SIGNAL` (variable condición)
    - `else` → Incrementar contador

Implementación de mensajes a partir de monitores

- Para un proceso una variable condición
- Par aun buzón ESTRUCTURA DE NO SE QUE MIERDAS CON MUCHAS COSAS

TERMINAR ESTA MIERDA DE SECCION QUE NO SE A DONDE ME LLEVARA

2.3. Planificación

CUando un ordenador es multiprogramado, tiene frecuentemente multitud de procesos o threads compitiendo por la CPU al mismo tiempo. Esta situación se produce cuando dos o más están simultáneamente en el estado listo. Si sólo una CPU está disponible, se ha de hacer una elección sobre qué proceso correr después, de esto se encarga el **planificador**.h¡ Características de los algoritmos de planificación:

1. **Imparcialidad**: Asegurarse que cada proceso tiene su parte justa de la CPU.
2. **Eficiencia**: Mantener la CPU ocupada el 100 % del tiempo.
3. **Tiempo de respuesta**: Minimizar el tiempo de respuesta para usuarios interactivos.
4. ***Turnaround***: Minimizar el tiempo que una tanda de usuarios tiene que esperar para obtener respuesta.
5. ***Throughput***: Maximizar el número de trabajos procesados por hora.

El **reloj interno** a cada interrupción el S.O. toma el control y decide el siguiente proceso que se ejecutará.

2.3.1. Round Robin

A cada proceso se le asigna un tiempo de ejecución → quantum.

2.3.2. Prioridades

A cada proceso se le asigna una prioridad, y el proceso ejecutable con la máxima prioridad es el que se ejecuta:

- Asignación de prioridades
    - Estática: No cambia, fácil implementación (poco *overhead*) y sin respuesta a cambios de entorno.
    - Dinámica: Dispone de mecanismos para su cambio, esquemas más difíciles de implementar que los estáticos (más overhead) y son sensibles a los cambios de entorno.

Hay veces que interesa agrupar los procesos por clases de prioridad, utilizando planificación de prioridades entre las clases y planificación *round robin* entre los procesos de cada clase.

2.3.3. Colas Múltiples

Es una variante de la planificación en clases de prioridad. La condición es que siempre que un proceso emplea la totalidad del quantum asignado se le baja una clase.

2.3.4. Primero Trabajo más Corto (SJF)

Algoritmo indicado para sistemas *batch* en los que son conocidos los tiempos de ejecución. Al ejecutar primero el trabajo más corto se produce el menor tiempo medio de respuesta. La generalización de SJF son los sistemas interactivos. Determinación del proceso ejecutable más corto, a partir de estimaciones basadas en el comportamiento anterior.

SJF es óptimo únicamente cuando todos los trabajos están disponibles simultáneamente.

2.3.5. Predeterminada

Consiste en adquirir un compromiso con el usuario y cumplirlo.

- Seguimiento del sistema:
    - tiempo CPU empleado por el usuario
    - Tiempo usuario activo en el sistema

La planificación determinada es similar a los sistemas de tiempo real → Planificación Deadline.

Dificultad:

1. El sistema ha de ejecutar el trabajo sin degradar el servicio a los otros usuarios.
2. El sistema debe planificar la asignación de recursos completamente hasta el deadline. Pueden llegar trabajos que introduzcan demandas que el sistema no puede predecir.
3. Varios trabajos deadline a la vez → introducir sofisticados métodos de optimización para cumplir plazos → overhead.

2.3.6. Planificación a Dos Niveles

Los tiempos de intercambio entre memoria principal y disco duro son uno o dos órdenes de magnitud superiores a los de intercambio entre los procesos que encuentran en memoria principal.

- El planificador a bajo nivel → Procesos en MP
- Planificador a alto nivel → Desplazamiento entre MP y Disco

Criterios de decisión de un planificador de alto nivel:

1. ¿Cuánto tiempo ha pasado desde que el proceso ha sido introducido o extraído?
2. ¿Cuánto tiempo de CPU ha consumido el proceso recientemente?
3. ¿Cómo de grande es el proceso?
4. ¿ Cómo de alta es la prioridad del proceso?

2.3.7. Evaluación

Existen diferentes métodos de evaluación:

- **Métodos Analíticos**: Consisten en aplicar el algoritmo seleccionado a una determinada carga del sistema y producir una fórmula o número que evalúa la eficacia del algoritmo para esa carga.
  - Modelo determinista: Se toma una carga particular establecida con anterioridad y se determina la eficacia de cada algoritmo para esa carga:
    - Produce números exactos y permite la comparación de algoritmos
    - Requiere números exactos en sus datos de entrada y los resultados obtenidos se pueden aplicar únicamente a esos datos.
  - Modelo de colas: Según este modelo, el ordenador se describe como un servidor, donde se determinan las distribuciones de probabilidad tanto del tiempo de llegada entre procesos como del tiempo de empleo de cPU. A partir de estas dos distribuciones se pueden calcular los valores medios de throughput, utilización, tiempo de espera...
    - Las distribuciones de llegadas y servicio se definen por relaciones matemáticas que muchas veces suponen hacer simplificaciones que restan exactitud al método, por ser éstas sólo una aproximación a los sistemas reales.
- **Simulación**: Permite conseguir una evaluación más precisa de los algoritmos de evaluación (aunque siempre nos referimos a niveles de confianza). Supone realizar e integrar una serie de tareas entre las que destacan:
  - Modelado y validación del sistema
  - Modelado y validación de los datos
  - Diseño de los experimentos
  - Validación de resultados
- **Implementación**: Única forma exacta de evaluar un algoritmo de planifiacción. Consiste en introducir el algoritmo en el S.O. para ver cómo responde bajo condiciones operativas reales.

2.4. Tratamiento de Procesos MINIX

2.4.1. Estructura Interna

1. **Capa 1**: Modelo de procesos secuenciales independientes que se comunican utilizando mensajes. Tiene 2 tareas
   - Captura de traps e interrupciones (lenguaje ensamblador)
   - Gestión de mensajes (lenguaje C)
2. **Capa 2**: Procesos de E/S → Tareas E/S → Drivers de dispositivo + System Taks. El kernel es capa 1 mas capa 2. Un único programa ejecutable que corre en modo kernel.
3. **Capa 3**: Servicio MM + Servicio FS. el S.O. se encarga de
   - Gestión de recursos - Kernel (capa 1 y 2)
   - Máquina extendida - MM y FS (capa 3)
4. **Capa 4**: Procesos de usuario → Shell, editores, compiladores...

2.4.2. Gestión de Procesos

MINIX - Jerarquía de procesos en forma de árbol donde la raíz es INIT

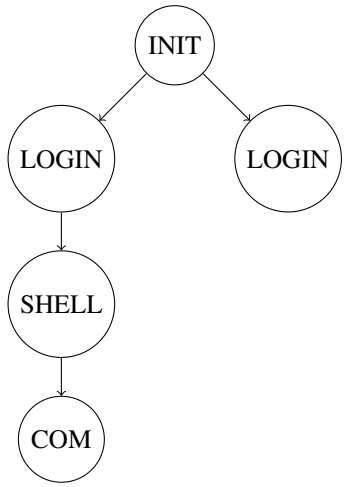


Figura 10: Árbol de procesos de MINIX

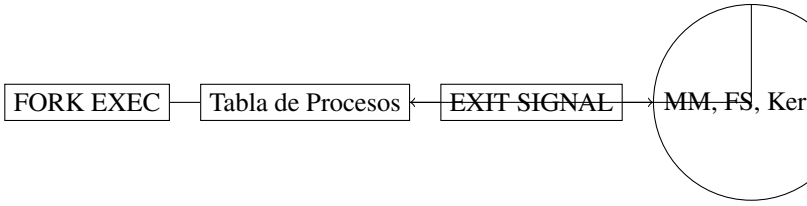


Figura 11: Transferencias de mensajes

2.4.3. Comunicación entre Procesos

Mediante el intercambio de mensajes de tamaño fijo (estructura mensaje), se utilizan 3 primitivas para el envío y recepción de mensajes:

- `send`
- `receive`
- `send_rec`

Cada proceso puede enviar y recibir mensajes de los procesos de su propia capa o de aquellos en capas contiguas.

MINIX utiliza el método rendez vous, no necesita gestión de almacenamiento.

2.4.4. Planificación de Procesos

Planificador de clases de prioridad con 3 niveles:

1. Nivel 0 (máxima prioridad) - Tareas capa 2
2. Nivel 1 (prioridad intermedia) - Tareas capa 3
3. Nivel 2 (mínima prioridad) - Tareas capa 4

Las tareas de los niveles 0 y 1 nunca son suspendidas, mientras que las correspondientes al nivel 2 (nivel de usuari) utilizan el algoritmo round robin.

2.5. Implementación de Procesos en MINIX

2.5.1. Organización MINIX

Minix se organiza ne los siguientes directorios:

- h: Archivos de encabezamiento
- Kernerl: Capas 1 y 2 (procesos, mensajes y drivers)
- mm: Gestión de memoria
- fss: Sistema de archivso
- lib: Funciones de libreria - llamadas al sistema
- tools: Programas necesairos para construir minix
- commands: Programas de utilidad
- include: Archivso de encabezamiento utilizados por los command
- test: Programas de comprobacion
- doc: Documetnacion y manuales

MINIX es un conjunto de 4 programas totalmente independientes (init, kernel, mm y fs) que se comunicacn a través de mensajes

2.5.2. Directorio h

BLA BLA BLA

2.5.3. Directorio Kernel - Archivos de encabezamiento

BLA BLA BLA

2.5.4. kernel main.c

BLA BLA BLA

2.5.5. kernel mpx88.s

2.5.6. RESUMEN

Funciones capa 1 minix:

- Iniciación del sistema
- GEstión de interrupciones
- GEstión de mensajes (envío y recepcion)
- Planificación

Mirate los códigos que hay por aqui

Para esconder los efectos de las interrupciones, los sistemas operativos proveen un modelo conceptual que consiste en procesos secuenciales corriendo en paralelo.

3. Gestión de E/S

3.1. HW E/S

3.1.1. Dispositivos E/S

Los dispositivos E/S pueden ser:

- **Dispositivos de bloque**: Almacenan información en bloques de tamaño fijo, cada uno con su propia dirección. Permiten leer o escribrir cada bloque con independencia de los demás. Son los discos.
- **Dispositivos de carácter**: Liberan o aceptan un conjunto de caracteres sin contemplar ninguna estructura de bloque. Por ejemplo terminales o impresoras.
- **Relojes**: No responden a estructura de bloques direccionables, no aceptan o generan conjuntos de caracteres y causan interrupciones en intervalos definidos de tiempo.

La E/S tiene dos puntos de vista:

- Funcional - Parte independiente - Sistema Archivos
- Estructural - Parte dependiente - Drivers del dispositivo

3.1.2. Controladores de Dispositivo

Los dispositivos de E/S:

- Componente mecánico - Dispositivo propiamente
- Componente electrónico - Drivers de dispositivo

El **controlador** es el responsable del control de los dispositivos exteriores y del intercambio de datos entre los dispositivos y la memoria principal y/o registros de la CPU.

Dibujicos que no se sin utiles

3.2. SW E/S

Estructura en capas:

- Capas inferiores: Esconden peculiaridades HW a las superiores
- Capas superiores: Presentan interfaz simple al usuario

3.2.1. Objetivos

- Software independiente de dispositivo
  - Tiene que existir la posibilidad de escribir programas que puedan ser utilizados con archivos que se encuentren en cualquier dispositivo, sin tener que modificarlos para acondicionarlos a las características de los distintos dispositivos.
- Nombramiento Uniforme (independiente del dispositivo)
  - Todos los archivos y dispositivos se direccionan de la misma forma, a partir del path name (sendero), sin que dependa este direccionamiento del tipo de dispositivo.
- Gestión de errores:
  - En general, los errores deben de ser corregidos tan cerca como sea posible, del lugar en que se producen
- Conversión de las transferencias asíncronas en síncronas
  - Hacer que aquellas operaciones que son activadas por interrupcioens aparezcan como bloqueadas a los programas usuario.
- Clasificación de dispositivos
  - Gestionar al mismo tiempo dispositivos “dedicados” y “compartidos”.

SW nivel usuario
SW independiente dispositivo
Drivers de dispositivo
Gesti´on de interrupciones

Figura 12: Capas del SW de E/S

3.2.2. Gestión de Interrupciones

Trata de ocultar las interrupciones. Siempre que se produzca un comando E/S y se espere una interrupción tendremos un proceso bloqueado. Cuando se produzca la interrupción se desbloquea el proceso previamente bloqueado:

1. UP semáforo
2. SIGNAL variable condición - monitor
3. SEND mensaje al proceso bloqueado

3.2.3. Drivers del Dispositivo

El driver es un programa dependiente de dispositivo, por eso cada tipo de dispositivo tiene un driver. El driver conoce los registros del controlador y los detalles del dispositivo. Su tarea consiste en aceptar órdenes del SW de la capa superior (independiente), traducirlas a términos concretos (establecer las operaciones que se tienen querealizar y en qué orden) y hacer que se realicen.

Una vez que haya emitido su comando o comandos se autobloqueará hasta que se produzca la correspondiente interrupción, momento en que se desbloqueará.

una vez desbloqueado, comprobará si se han producido errores y pasará los datos al SW independiente.

3.2.4. SW Independiente de dispositivo

Realiza las tareas E/S comunes a todos los dispositivos:

1. Provee una interfaz uniforma a nivel de usuario
2. Cuando se nombra un dispositivo se encarga de asignarle el driver correspondiente
3. Previene a los dispositivos de accesos no autorizados
4. Permite gestionar dispositivos abstractos que emplean un mismo tamaño de bloque, independientemente del tamaño físico real del sector.
5. Gesitona los problemas derivados del almacenamiento, tanto en los dispositivos de bloque como en los de carácter.
6. Asigna el espacio para nuevos archivos que se van a crear.
7. Realiza el tratamiento apropiado según la naturaleza del dispositivo, dedicado o compartido.
8. Gestiona los errores qu eno se han podido corregir en las capas inferiores.

3.2.5. SW Nivel de Usuario

Aunque la mayor parte del SW E/S se encuentra dentro del S.O., una pequeña parte corre fuera de éste.

1. Las llamadas a sistema (incluyendo las relacionadas con E/S), que se implementan a través de subrutinas de librería.
2. Algunas funciones que se implementan trambién a través de subrutinas de librería.
3. Algunos sistemas (procesos) se realizan desde nivel usuario.

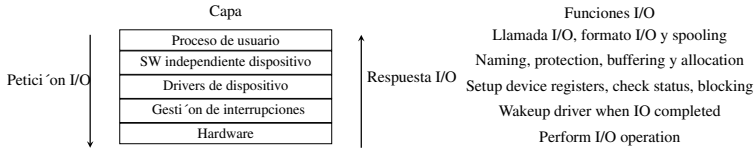


Figura 13: Resumen

3.3. Tratamiento E/S en MINIX

3.3.1. Gestión de Interrupciones

Cuando un driver de dispositivos comienza una operación de E/S, se bloquea en espera de un mensaje. Este mensaje es generado en la parte del S.O. que se encarga de la gestión de interrupciones (capa 1).

3.3.2. Drivers de Dispositivo

Hay un driver por cada tarea. Pertenecen al kernel, lo que permite un fácil acceso a la tabla de procesos y otras estructuras importantes.

Formas de estructurar la comunicación entre el usuario y el sistema:

- Process-structured system: 1-4 son mensajes de petición y respuesta entre 3 procesos independientes

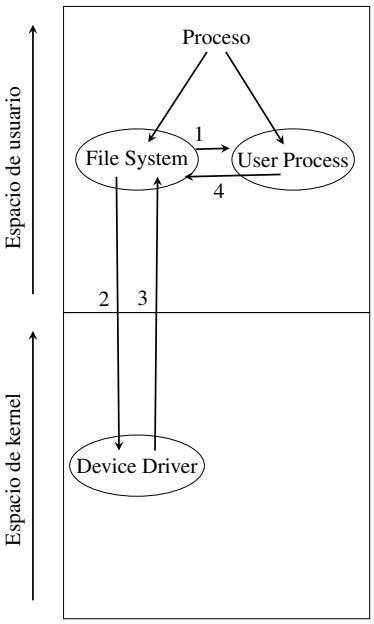


Figura 14: Resumen

- Sistema monolítico: La parte de espacio de usuario llama a la parte de kernel por trapping. El sistema de archivos llama al driver de dispositivo como un procedimiento. El S.O. entero corre como un único programa en modo kernel.

Estructura general de los drivers de dispositivo:

- Capturar un mensaje
- Ejecutar lo que indica el mensaje
- Devolver un mensaje de respuesta

3.3.3. SW independiente de dispositivo

Código E/S independiente del dispositivo → Sistema de archivo (tema 5).

3.3.4. SW Nivel de Usuario

MINIX dispone de funciones de librería para realizar llamadas al sistema y para convertir: binario a ASCII y al revés.

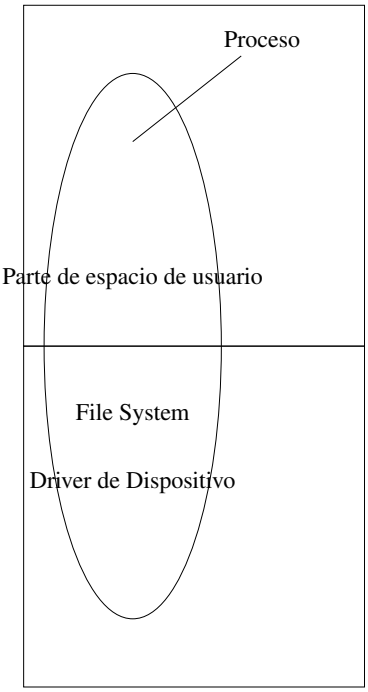


Figura 15: Sistema monolítico

3.4. Memoria RAM

3.4.1. HW Y SW

El disco RAM:

- Dispositivo de bloque más simple de todos, parte de la MP
- Tiene dos comandos: LEER bloque y ESCRIBIR BLOQUE

COsicas aqui

3.4.2. Driver Memoria RAM en MINIX

Número mayor de dispositivo es la memoria RAM. Número menor de dispositivo 0...3

1. `/dev/ram`
2. `/dev/mem` Superusuario
3. `/dev/kmem` Superusuario
4. `/dev/null` Tamaño cero

3.5. Discos

3.5.1. HW

Los datos son registrados y recuperados del disco a través de una bobina conductora (cabeza). Durante una operación de lectura/escritura, la cabeza permanece fija, girando el disco debajo de ella.

3.5.2. SW

Tiempo lectura/escritura de un bloque:

1. Tiempo de búsqueda (*seek time*
2. Tiempo de rotación (*latency time*)
3. Tiempo de transferencia (*latency time*)

La planificación del disco:

- Multiprogramación: Procesos generan solicitudes de acceso a disco (leer, escribir) más deprisa de lo que pueden ser atendidas (movimiento de la cabeza de disco).

- Planificación: Examen detallado de las solicitudes pendientes para determinar el modo más eficiente de atenderlas.

Planificación:

- Seek optimization
- Rotational optimization

Características:

1. Throughput: Maximizar el número de solicitudes atendidas por unidad de tiempo
2. Minimizar el tiempo de respuesta medio
3. Minimizar la varianza del tiempo de respuesta

Optimización del tiempo de búsqueda:

- FCFS (*First Come First Served*): No hay reordenamiento de la cola
- SSTF (*Shortest Seek Time First*): Sirve la solicitud con menor tiempo de búsqueda, aunque no sea la primera de la cola.
- SCAN: Opera como SSTF execepto que existe la solicitud de menor distancia de búsqueda en una dirección definida.
- N-Step Scan: El brazo de disco se mueve como el caso SCAN, escepto que únicamente da servicio a las solicitudes en estado de espera, cuando comienza un barrido.
- C-SCAN (Circular scan): Elimina la discriminación de las estrategias anteriores en relación con los cilindros internos y externos.

Tratamiento de errores comunes qu es e producen en los discos:

1. Error de programación: Petición de un sector inexistente
2. Error de checksum pasajero
3. Error de checksum permanente
4. Error de búsqueda
5. Error de controlador

Cache track-at-a-time. Leer 1 sector es leer 1 track

3.6. Relojes

3.6.1. HW

Dispositivo que genera interrupciones en intervalos definidos de tiempo.

3.6.2. SW

Tareas en las que interviene el reloj:

1. Mantener hora y día.
2. Prevenir que los procesos corran más tiempos del que han sido asignados.
3. Contar el tiempo de uso de la CPU.
4. Manejar la llamada al sistema `ALARM` hecha por procesos de usuario.
5. Proveer temporizadores watchdog para partes del sistema
6. Gestión de estadísticas.

Para simular múltiples relojes se usan los temporizadores watchdog.

3.7. Terminales

3.7.1. HW

Hay dos tipos de terminales en función de la forma en el que el S.O. se comunica con ellos:

- Interfaz RS-232 para comunicación serie
- Terminal Mapeado de Memoria

Igual añadir algo más aqui pero no sep

3.8. System Task

3.8.1. Tarea del sistema en MINIX

Es el SW encargado de comunicar el kernel con MM y FS vía mensajes en aquellos casos en que el kernel necesita estar informado de acciones que se desarrollan en capas superiores.

$$\text{Fork (MM)} \rightarrow \text{Planificación nuevo proceso (kernel)}$$

Igual añadir algo más aqui pero no sep

4. Gestión de Memoria

La gestión de memoria

- Seguimiento de zonas de memoria ocupada y desocupada
- Asignación/desasignación de memoria (procesos)
- Intercambio con el disco (cuando no hay capacidad suficiente en la MP)

4.1. G.M. sin Intercambio ni Paginación

Gestión de memoria **sin abstracción**

- Procesos van y vienen entre MP y disco (durante su ejecución, intercambio, paginación...
- Procesos se mantienen en MP

A pesar de no poder correr varios programas de manera simultáneamente, es posible hacerlo. El Sistema Operativo tendrá que guardar el contenido de la memoria a un archivo de disco, y entonces volverlos a traer y correr el siguiente programa. Siempre y cuando haya un único programa a la vez en memoria no habrá conflictos. A esto se le llama *swapping*.

4.2. Espacios de Dirección

No es posible dar acceso a los programas de usuario a todos los registros de memoria.

Para permitir a múltiples programas correr a la vez en memoria hay que resolver los siguientes problemas: protección y relocation.

Se puede resolver con la reubicación de los programas cuando son cargados, es una solución lenta y complicada.

La solución a esto es inventar una nueva abstracción para la memoria: el espacio de direcciones. Un **espacio de direcciones** es un conjunto de direcciones que un proceso puede utilizad para direccionar memoria.

Esta solución es una versión simple de la **reubicación dinámica**, mapea la dirección de cada proceso en una parte diferente de la memoria física de una forma sencilla.

Emplea los registros **base** y **límite**. Cuando un proceso se ejecuta, el registro base se carga con la dirección física donde el programa comienza en memoria ye l registro límite se carga con la longitud del programa.

El uso de estos registros es una forma sencilla de dotar a cada proceso de su propio espacio de direcciones porque cada dirección de memoria generada automáticamente tiene los contenidos del registro base añadidos antes de ser enviados a memoria.

Una desventaja de la reubicación utilizando registros base y límites es la necesidad de sumar y comparar cada referencia a memoria. Las comparaciones se realizan rápido pero las sumas pueden requerir mucho tiempo.

4.2.1. Swapping

Si la memoria física del ordenador es suficientemente grande para albergar todos los procesos no habrá problema, pero este no es el caso.

Hay dos approaches para manejar la sobrecarga de memoria:

- La estrategia más simple es el **swapping** que consiste en traer cada proceso entero, ejecutarlo durante un tiempo y devolverlo al disco. Los procesos en reposo, en su mayoría, se guardan en disco por lo que no consumen recursos cuando no están corriendo.
- La otra estrategia es la **memoria virtual**, permite a los programas ser ejecutados aun cuando sólo están parcialmente en la memoria principal.

Swapping tiene un problema, deja huecos en memoria. Es posible combinarlos en uno único moviéndolos tan abajo como sea posible, esta técnica se conoce como **compactación de memoria**.

SI los procesos son creados con un tamaño fijo que nunca cambia, la reubicación es simple. El sistema operativo reserva tanta memoria como es necesario, ni de más ni de menos.

4.3. Monoprogramación sin intercambio ni paginación

Sólo un proceso a la vez en ejecución, al que se le permite ocupar la totalidad de la memoria.

Modo de funcionamiento:

1. Usuario escribe comando en terminal $\rightarrow$ S.O. se encarga de cargar el programa del disco y ejecutarlo
2. Cuando el proceso termina, S.O. presenta un prompt en pantalla y espera un nuevo comando del terminal para cargar un nuevo proceso.

4.3.1. Multiprogramación y Empleo de Memoria

Multiprogramación $\rightarrow$ Mejor utilización de la CPU

4.3.2. Multiprogramación con Particiones Fijas

División de la memoria en `n` partes:

- Colas separadas: Cuando llega un proceso, se pone en la cola de la partición más pequeña donde quepa. El espacio partición no ocupado es espacio perdido. Cola partición grande vacía, colas particiones pequeñas llenas.
- Cola única: cuando queda una partición libre, se encarga el trabajo más cercano al comienzo de la cola que quepa en él. Trabajos pequeños es espacio perdido. La alternativa es coger de la cola el mayor de los trabajos que quepa. Discrimina trabajos pequeños.

Reubicación y protección. La reubicación necesita soporte de HW (data typing y recolocación dinámica).

- Data typing: Se registra el tipo de valor almacenado en cada localización de memoria.
- Recolocación: S.O. utilizando instrucciones privilegiadas examina el tipo de información del código y localiza cada apuntador que se ha de modificar.
    - Recolocación dinámica: Existen 2 registros privilegiados, registro base de recolocación y registro límite. En cada referencia a memoria, se añade automáticamente el contenido del registro base a la dirección efectiva.

La protección necesita soporte HW (candados y llaves y registro límite).

- Candados y llaves: La memoria se divide en bloques de un tamaño determinado y se le asigna un código de protección. Cuando un proceso intenta acceder a una posición de memoria con un código incorrecto se producirá una interrupción de protección.
- Registro límite: Se carga con la máxima dirección del espacio direccionable del proceso y si el preso intenta acceder a una dirección superior a la indicada en el registro límite se produce una interrupción de protección.

4.4. Intercambio (Swapping)

En aquellos sistemas de tiempo compartido en los que hay más usuarios que memoria donde mantener sus procesos, es necesario mantener algunos de los procesos, es necesario mantener algunos de los procesos en el disco e intercambios con los procesos de la memoria principal cada cierto tiempo.

4.4.1. Multiprogramación con Particiones Variables

El intercambio se puede realizar con particiones fijas. En la práctica, cuando la memoria es escasa, se pierde mucho espacio en los casos en que los programas son más pequeños que las particiones.

La alternativa son particiones variables, es un nuevo algoritmo de GM donde el número y tamaño de los procesos en memoria varían dinámicamente.

DIBUJICO

Particiones variables:

- Mayor flexibilidad, mejora el empleo de memoria.
- Complica asignación y desasignación de memoria
- Fragmentación externa
- Complica seguimiento de memoria.

Asignación del tamaño de memoria. Procesos:

- Mantienen mismo tamaño durante su ejecución. Asignación de la memoria que necesitan.
- No mantienen mismo tamaño durante su ejecución. Si se espera que la mayoría de los procesos vayan a crecer cuando se ejecutan, se les asigna una memoria "extra" en lugar de complicar el algoritmo de gestión con un mecanismo que resuelva estas situaciones.

Compactación, solución al problema de la fragmentación. Su objetivo es reunir toda la memoria libre en un único bloque. Supone reubicación de procesos, sólo es posible si se realiza en el tiempo de ejecución (reubicación dinámica).

4.4.2. G.M. con Mapa de Bits

División de la memoria en partes iguales (unidades de asignación). A cada unidad de asignación le corresponde un bit en el mapa.

- bit 0 - unidad libre (desocupada)
- bit 1 - unidad ocupada

El diseño depende del tamaño de las unidades de asignación:

- pequeño: mapa de bits grande
- grande: mapa de bits pequeño, se desperdicia memoria)

El seguimiento del estado de la memoria se realiza de forma sencilla, localizándolo en una zona fija de memoria, determianda por el tamaño de la memoria y el tamaño de la unidad de asignación.

Esto es un problema ya que la búsqueda de un bitmap de una longitud deterinada es una operación muy lenta. Es un argumento suficientemente importante para no usar bitmaps.

4.4.3. G.M. con listas enlazadas

Lista formada pro segmentos de memoria, tanto libres como asignados, todos ellos enlazados. Cada entrada en la lista especifica si es un hueco o un proceso.

Cuando los procesos y los huecos se mantienen en una lista ordenados por direcciones, se pueden utilizar diferentes algoritmos para reservar memoria para un proceso nuevo.

- **First fit**: El gestor de memoria busca en la lista de segmentos hasta que encuentra un hueco suficientemente grande. Este hueco se rompe en dos partes, la parte utilizada por el proceso y el hueco de memoria inutilizado. Es un algoritmo rápido.
- **Next fit**: Variación menor de first fit. Tiene en cuenta donde se hace la última asignación (no vuelve al principio). La siguiente búsqueda la realiza a partir de la última asignación en lugar de empezar por el principio como haría First Fit. Tiene un rendimiento ligeramente menor que First Fit.
- **Best fit**: Mira en toda la lista y coge el hueco más pequeño adecuado. Minimizando así el hueco desaprovechado. Es más lento que ferst fit y desperdicia más memoria que first fit.
- **Worst fit**: Idea opuesta a Best Fit. Asigna el mayor hueco disponible por lo que el nuevo hueco será lo suficientemente grande como para ser útil.
- **Quick Fit**: Mantiene listas separadas para los tamaños más solicitados. Encontrar un hueco de un tamaño determinado es muy rápido pero cuando un proceso termina o es extraído encontrar sus vecinos para ver si se pueden juntar es muy costoso.

Todos estos algoritmos pueden ser acelerados manteniendo listas separadas para procesos y huecos.

4.4.4. G.M. con Sistemas Buddy

Este sistema acelera la fusión de huecos adyacentes (cuando un proceso termina o es intercambiado) utilizando direccionamiento binario.

Forma de trabajo. La G.M. mantiene una lista de bloques libre de 2^k Bytes.

4.5. Memoria Virtual

Mientras que los registros base y límites pueden ser utilizados para crear la abstacción de los espacios de direcciones existe otro problema que hay que resolver, el manejo de bloatware.

El problema de manejar programas que son más grandes que la memoria ha estado presente desde el principio. Una solución adoptada para resolver este programa es dividir el programa en pequeñas piezas llamadas ***overlays*** .

A pesar de que el trabajo de extraer e introducir overlays lo realize el sistema operativo, el trabajo de realizar la división en piezas tiene que ser realizado manualmente por el programa.

Como esto no era una solución óptima se introdujo la **memoria virtual**. La idea básica es que cada programa tiene su propio espacio de direcciones, el cual es dividido en trozos que son **páginas**, siendo cada una un rango contiguo de direcciones. Estas páginas son mapeadas en la memoria física pero no todas las páginas tienen que estar en la memoria física a la vez para correr el programa. Cuando un programa referencia parte de su espacio de direcciones que está en la memoria física, el harware realiza el mapeo necesario al vuelo. Si el programa referencia parte de su espacio de direcciones que no está en dirección física, el sistema operativo es alertado para que vaya a recoger la parte que falta y reejecute la instrucción que falla.

4.5.1. Paginación

Imagen de MMU pag 195

La mayoría de sistemas de memoria virtual utilizan la técnica de **paginación**. El direccionamiento virtual genera unas direcciones llamadas **direcciones virtuales** que forman el **espacio de memoria virtual**. Cuando se utiliza memoria virtual, las direcciones virtuales no van directamente al bus de memoria. En su lugar, van a **MMU** (*Memory Management Unit*) que se encarga de mapear las direcciones virtuales en las direcciones físicas de memoria.

El espacio de direcciones virtuales consiste en unidades de tamaño fijo llamadas páginas. Las unidades que le corresponden en el espacio de memoria física son las **page frames**. Las páginas y pages frames, generalmente, son del mismo tamaño.

Hoy día, un bit de presente/ausente lleva la cuenta de qué páginas están físicamente presentes en memoria.

¿Qué pasa cuando un programa referencia una dirección sin mapear? Provoca que la CPU haga un trap al Sistema Operativo, una **falta de página**. El sistema operativo elige una page frame poco usada y escribe su contenidos de vuelta en el disco (si no está ya ahí). Entonces recupera (también desde el disco) la página que fue referenciada en la página que acaba de ser liberada, cambia el mapa y reinicia la trapped instruction.

El número de página es usado como índice en la **tabla de página**, siendo el número de la page frame correspondiente a la página virtual. Si el bit present/absent está a 0 se causa un trap al sistema operativo. Si el bit está a 1 el número de page frame encontrado en la tabla de página se copia al registro.
Las direcciones empleadas en un programa, son direcciones virtuales (lógicas) y forman el espacio de direccionamiento virtual, que es el que se puede direccionar la GPU.

- Sistemas sin M.V. $\rightarrow$ Direcciones cargadas directamente en el bus de direcciones (acceso directo memoria física o real)
- Sistemas con M.V. $\rightarrow$ Direcciones virtuales, MMU (*Memory Management Unit*) es el hardware que transforma la dirección virtual en dirección física.

Cuando se produce una falta de página, el S.O. decide expulsar una de las páginas reales y cargar una página virtual en la dirección que ha quedado libre, y además se encarga de actualizar la table.

Tablas de página

- Dirección virtual:
  - Número de página virtual
  - Offset

El propósito de la tabla de página es convertir páginas virtuales en páginas reales.

Alternativas de diseño:

1. Única tabla de página $\rightarrow$ Array de registros HW con una entrada por cada página virtual, ordenadas numéricamente. Durante la ejecución se evitan las referencias a la tabla de página. Resulta una solución cara en el caso en el que las tablas sean grandes. Se produce una pérdida de eficacia al tener que cargar diferentes tablas cada vez que se produzca un cambio de contesto.
2. Mantener la tabla de páginas en memoria principal, disponiendo de un registro que apunte al comienzo de la tabla. Un cambio de contexto supondrá únicamente cambiar el contenido del registro. En cambio, la ejecución de una instrucción requerirá una om ás referencias de memoria.

Las tablas de página multinivel evita mantener todas las tablas de página en memoria continuamente.

Si la página no está en memoria, se producirá una falta de página. Si la página está en memoria el número tomado de la tabla de página del segundo nivel se combinará con el offset para construir la dirección física. Una vez obtenida esta dirección se colocará en el bus y se enviará a la memoria.

4.5.2. Tablas de Página

En una implementación simple, el mapeo de las direcciones virtuales a direcciones físicas se puede resumir como:

- La dirección virtual se divide en:
  - Número de página virtual (bits más significativos), se utiliza como índice en la tabla de página para encontrar la entrada para esa página virtual.
  - Offset (bits menos significativos). Es el mismo para la dirección física y la direción virtual.

Así, la finalidad de la tabla de páginas es mapear páginas virutales en page frames.

Estructura de una entrada en la tabla de páginas.

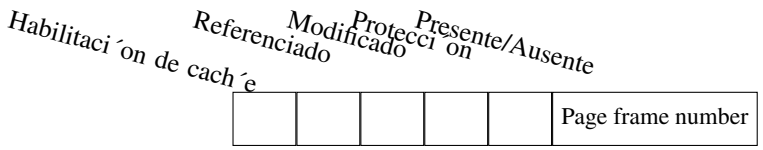


Figura 16: Entrada típica de tabla de página

El campo más importante es el Page Frame Number, después de todo, la principal meta del mapeo de páginas es sacar este valor. El bit de presente/ausente si está a uno indica que la entrada es válida y que puede ser utilizada, si es 0 la página virtual a la que la entrada pertenece indica que no está en estos momentos en memoria. Acceder a una entrada de page table con bit a 0 causa una falta de página. Los bits de protección indican que tipos de acceso son permitidos, si es 0 se puede leer/escribir y si es 1 solo lectura. El bit de Modificado y Referenciado lleva registro del uso de la página, cuando se escribe una página se pone el bit Modificado a 1. Si el SO reclama esta página y este bit ha sido modificado (está sucio) tiene que ser vuelta a escribir al disco y si no ha sido modificada (está limpia) puede ser abandonada ya que la copia del disco es válida. El bit referenciado se pone a 1 cuando una página es referenciada, tanto como por lectura o escritura. Las páginas que no han sido usadas son mejores candidatas que las páginas que han dido usadas en el reemplazo de páginas. Por último el último bit permite deshabilitar el cacheo de la página.

4.5.3. Acelerando la Paginación

En cualquier sistema de paginación hay que resolver dos principales problemas:

1. El mapeo de dirección virtuala dirección física tiene que ser rápido.
2. Si el espacio de direcciones virtuales es grande, la tabla de páginas será grande.

El primer punto es una consecuencia del hecho de que el mapeo de dirección virtual a física tiene que ser hecho en cada referencia a memoria.

Una solución son los **Translation Lookaside Buffers**, está basada en la observación de que la mayoría de los programas tienden a hacer muchas referencias a un pequeño número de páginas y no de la otra forma. De esa forma, una pequeña fracción de las entradas de las tablas de página son leídas; siendo el resto no leídas prácticamente.

Se proponen las TLB es equipar a los ordenadores con un pequeño dispositivo hardware para mapear las direcciones virtuale s adirecciones físicas sin tener que recorrer la tabla de página.

Cada entrada contiene información sobre una página, incluyendo el número de página virtual , un bit que se pone a 1 cuando la página es

modificada, el código de protección y la page frame física en la cual la página está ubicada. Estos campos tienen una correspondencia uno a uno con los campos en la tabla de página exceptuando el número de página virtual que no es necesario en la tabla de páginas.

¿Cómo funciona la TLB? Cuando se le pasa una dirección virtual a la MMU para traducirla, primero el hardware comprueba si el número de página virtual está presente en la TLB comparándolo con todas las entradas simultáneamente. Hacer esto requiere hardware especial, el cual todas las MMUs con TLBs tienen. Si se encuentra un match y el acceso no viola el bit de protección, el page frame se toma directamente de la TLB sin tener que ir a la table page. Si el número de página virtual está presente en la TLB pero la instrucción trata de escribir en un apágina de solo escritura se genera una falta de protección.

Hay un caso interesante cuando el número de página virtual no está en la TLB. La MMU detecta la pérdida y realiza una búsqueda en la tabla de páginas ordinaria. Entonces extrae una de las entradas de la TLB y la reemplaza con la tabla de página que acaba de buscar.

Los traps al Sistema Operativo sólo ocurren cuando una página no está en memoria.

Cuando se utiliza software para manejar la TLB hay que entender la diferencia entre diferentes tipos de misses (?????). Un **fallo leve** se produce cuando la página referenciada no está en la TLB pero está en memoria. Para solucionarlo hay que actualizar la TLB. Un **fallo grave** ocurre cuando la página no está ni en la TLB ni en memoria. Se soluciona accediendo al disco para traer la página.

4.6. Segmentación

Compilación en una memoria uni

FAAAALTAN

4.7. Algoritmos de Sustitución de Página

Cuando hay una falta de página, el sistema operativo tiene que elegir una página a expulsar (eliminar de memoria) para hacer hueco a la página que va a llegar. Si la página que va a ser eliminada ha sido modificada mientras estaba en memoria, tiene que voler a ser reescrita en el disco para que se actualice. Si no ha sido actualizada la copia que hay en el disco ya está actualizada por lo que no se necesita volver a reescribirlo. La página a ser leída sobreescribirá la página que va a ser expulsada.

Mientras que podría ser posible expulsar aleatoriamente cualquier página, no es óptimo.
La página a sustituir puede elegirse aleatoriamente corriéndose el riesgo de que la página expulsada sea empleada con mucha frecuencia, lo que supondría una pérdida de rendimiento debido al trabajo extra de gestión que hay que realizar.

4.7.1. Sustitución Óptima

Si cada página que está en memoria tuviera une etiqueta donde se indicara el número de insturcciones que han de pasar antes de que fuera referenciada alguna de las intrucciones contenidas en la página, el algoritmo sustituiría aquella página con el valor más alto en su etiqueta.

4.7.2. Sustitución NRU

Not Recently Used, emplea bits asociados a cada página en memoria virtual:

- `R` (bit referenciado): Se activa por HW, cada vez que la página se referencia (lectura o escritura).
- `M` (bit modificado): Se activa por HW cada vez que la página se escribe

Este par de bits están presentes en cualquier entrada de la tabla de páginas. Deben de ser actualizados cada vez que se hace una referencia en memoria por lo que serán modificados por hardware. Una vez un bit se pone a 1, se queda con ese valor hasta que el sistema operativo lo resetea.

Tan pronto como cualquier página es referenciada ocurrirá una falta de página. El sistema operativo pondrá entonces el bit `R` a uno, cambia la entrada de la tabla de página para apuntar a la página adecuada con modo READ ONLY y reinicia la instrucción.
Algoritmo de sustitución a partir de los bits R y M:

1. Cuando empieza un proceso → S.O. pone los bits R y M de todas las páginas a 0.
2. Periódicamente (cada interrupción de reloj) s epone a 0 el bit R (para distinguir aquellas páginas que han sido referenciadas receintemente de las qu eno lo han sido)
3. Cuando se produce una falta de página → S.O. inspecciona todas las páginas y las clasifica en 4 categorías en función de sus valores de los bits R y M
   - Clase 0 No Ref no mod
   - Clase 1 no ref, mod
   - Clase 2 ref, no mod
   - Clase 3 ref, mod

Algoritmo NRU, cambia aleatoriamente una página de la clase numerada más baja no vacía.

4.7.3. Sustitución FIFO

El S.O. mantiene una lista de todas las páginas en memoria, ordenadas por el tiempo que llevan en memoria, con la que más tiempo lleva en cabeza y la que menos en cola. Tiene un problema, cuando la página más antigua que es expulsada es muy utilizada.

Haciendo una variación de este algitmo nace **segunda oportunidad**, resuelve el problema de tener que expulsar una página que se usa mucho. todo ello se soluciona inspeccionando el bit `R` de la página más antigua. Si es `0` la página es vieja y no ha sido utilizada por lo que puede ser expulsada. Si el bit `R` es 1, el bit se pone a 0, se coloca la página al final de la lista de páginas y la búsqueda continúa.

Si todas las páginas han sido referenciadas este algoritmo degenera en simplemente FIFO.

4.7.4. Sustitución Clock Page

A pesar de que segunda oportunidad es un algoritmo útil es ineficiente ya que está continuamente desplazando páginas por la lista. Un mejor planteamiento es mantener todas las páginas en una lista circular como si fuera un reloj en la que la mano del reloj apunta a la página más antigua.

Cuando ocurre una falta de página, la página que está siendo apuntada por la manecilla se inspecciona. Si el bit `R` está puesto a `0` la página se expulsa y se inserta una nueva en su lugar avanzando la manecilla. En cambio, si el bit `R` está puesto a `1`, se limpia y la manecilla avanza a la siguiente página. Este proceso se repite hasta que se encuentre una página con `R = 0`.

4.7.5. Sustitución LRU

Least Recently Used:

- Localismo → Páginas que han sido muy utilizadas en las últimas referencias, probablemenet lo serán en las siguientes.
- Algoritmo → Cuando ocurra una falta de página se expulsa la página que no haya sido utilizada por más tiempo o la menos recientemente utilizada

- Implementación → Mantener una lista encadenada de todas las páginas de mmeoria, con la más recientemente utilizada al frente y la menos recientemente utiliada en la cola.

La actualización de la lista en cada referencia en memoria (encontrar página en la lista, suprimirla y ponerla la frente) es un proceso muy lento.

4.8. Principios de Diseño de Sistemas de Paginación

4.8.1. Localismo

Propiedad empírica más que teórica. Los procesos realizan referencias de forma no uniforme, referencias según patrones altamente localizados.

- Sistemas de paginación
  - Referencia subconjunto de páginas
  - Las páginas referenciadas tienden a ser adyacentes en el espacio de direccionamiento virtual
- Localismo
  - Temporal → Referencias recientes tienen muchas posibilidades de ser referenciadas en un futuro cercano.
  - Espacial → Una vez que se realiza una referencia a una localización, es muy probable que sus vecinos sean referenciados.

La consecuencia → ejecución de un programa verá mejorada su eficacia en la medida que el conjunto de páginas referenciadas se encuentre en la memoria principal.

4.8.2. Working Set

Los procesos son iniciados con ninguna de sus páginas en memoria. Tan pronto como la CPU intente recoger la primera isntrucción se producirá una falta de página provocando al sistema operativo que traiga la página que contiene la primera instrucción. Después de esto se producirán otras faltas de páginas para variables globales ausentes.

Después de un rato, elp roceso tendrá la mayoría de páginas que necesita y comenzará a ejecutarse con relativamente pocas faltas de páginas.

A esta estrategia se la conoce como **paginado bajo demanda** ya que las páginas se cargan únicamente bajo demanda y nunca por adelantado.

Gestión de memoria con working set

- Mantener en memoria principal los working set de los programas activos. La decisión de añadir un nuevo proceso al conjunto de procesos activos (incrementar el grado de multiprogramación) se tomará cuando haya espacio suficiente para acomodar el working set del nuevo proceso.
- W, ventana del working set. El working set de un proceso en el instante "t" es el número de página referenciadas por el proceso en el intervalo de tiempo.
- Determinación del tamaño de ventana → Operación crítica para la eficacia en el funcionamiento de la gestión de memoria de los working set.
- El working set varía con la ejecución del proceso.

Una vez que el proceso se estabiliza, el sistema compara las referencias de página en la ventana con las observadas en la ventana anterior y aumenta o disminuye el número de página en el working set.

- La implementación de una auténtica política de gestión de memoria con la working set puede suponer una sobrecarga en la información de gestión, debido principalmente a que la composición de los working set pueden variar rápidamente.

4.8.3. Algoritmo PFF

Page Fault Frequency es una medida de la eficacia en la ejecución de un proceso es la relación de faltas de página.

- Procesos con muchas faltas → No mantienen el working set en MP → Thrasing
- Procesos con pocas faltas → demasiadas páginas → Impiden progresar a otros procesos

El algoritmo PFF ajusta el conjunto de páginas residentes de un proceso en función de la frecuencia de aparición de faltas (o en función del tiempo entre faltas.

Realiza un ajuste dinámico del conjunto residente de página en respuesta al comportamiento variable del proceso.

El elemento clave para un funcionamiento eficiente de este algoritmo es el de establecer los umbrales con valores apropiados.

Ventaja PFF sobre los Working Set → El ajuste conjunto residente de página se realiza después de cada falta, mientras que el working set se realiza después de cada referencia de almacenamiento.

4.8.4. Demanda de página

Ninguna página es trasladada desde el almacenamiento secundario hasta que sea explicitamente referenciada por el proceso en ejecución.

Ventajas:

- Garantía de que las páginas en mp son las que el proceso necesita en ese momento.
- La carga de información de gestión para la determinación de la página que se va a llevar a memoria es mínima (contra técnicas de anticipación).

Problemas:

- La carga de página se realiza página a página → concepto de coste de espera.

4.8.5. Prepaginación

Paginaión anticipada. El SO intenta predecir las páginas que necesitará el proceso y las carga cuando hay espacio disponible.

Mientras el proceso se está ejecutando con sus páginas actuales, el sistema carga páginas nuevas para que estén disponibles cuando el proceso las necesite.

Las páginas referenciasdas se solicita por demanda y las páginas vecinas son prepaginadas.

4.9. Tratamiento GM en MINIX

TERMINA

5. Gestión de Archivos

5.1. Archivos

5.2. Nombramiento

Un archivo es un mecanismo de abstracción, cada archivo está identificado por su nombre.

5.3. Estructura

Hay 3 posibilidades:

1. Secuencia de bytes (sin estructura): El S.O. no sabe ni le importa lo que hay en el fichero.
2. Secuencia de registros de longitud fija (principio de estructura): Array de arrays.
3. Árbol de registros: Array de estructuras ordenadas según un campo de la estructura (utilizada en grandes ordenadores)

5.3.1. Tipos

1. Convencionales
2. Directorios
3. Especiales
   - Carácter
   - Bloque

5.3.2. Acceso

1. Secuencial
2. Aleatorio

5.3.3. Atributos

5.3.4. Operaciones

5.3.5. Mapeado de Archivos en Memoria

Otra forma de acceder a los archivos es volcar (mapear) los archivos dentro del espacio de dirección de un proceso (MULTICS).

El mapeo, dad una dirección virtual y el nombre de un archivo hace que el S.O. mapee el archivo dentro de lespacio de direccionamiento de la dirección virtual.

el mapeo de memoria trabaja mejor en los sistemas que soportan segmentación. De esta forma, cada arvhico puede ser mapeado en su propio segmento.

Problemas del mapeado de archivos en memoria:

1. Resulta difícil al sistema conocer la longitud exacta del archivo de destino (en el caso de que se tratarad e una segmentación paginada, podríamos conmocer el número más alto de la página que ha sido escrita pero no cuantos ytes han sido escritos en ella).
2. (Potencialmente) Puede ocurrir que si un archivo es mapeado por un proceso y también es abierto por una operación convencional de lectura, si el proceso modifica una página, ese cambio no se reflejará en el archivo hasta que la página sea expulsada (inconsistencia).
3. El archivo puede ser mayor que el segmento.

5.4. Directorios

5.4.1. Sistemas Jerárquicos de Directorios

Un directorio, típicamente contiene un número de entradas, una por cada archivo. Hay dos modalidades

- Atributos en la entrada del directorio
- Atributos en otro lugar

Abrir un archivo:

1. S.O. busca en el directorio hasta que encuentra el nombre del archivo.
2. Extrae los atributos y las direcciones del disco y las coloca en una tabla de la memoria principal.
3. Todas las referencias posteriores al archivo utilizan la información de la memoria principal.

5.4.2. Paths

Cuando el sistema de archivos está organizado como un árbol de directorios se necesita algunan forma de especificar los nombres del archivo:

1. Path absoluto (desde el directorio raíz)
2. Path relativo (directorio de trabajo actual)

5.4.3. Operaciones

Las operaciones (llamadas al sistema) para la

5.5. Implentación de un Sistema de Archivos

5.5.1. Implementación Archivos

La tarea principal en la implementación de un archivo es el seguimiento de los bloques de disco que pertenecen a ese archivo.

La **asignación contigua** consiste en asignar bloques contiguos de disco. Como **ventaja** tiene que es muy sencillo de implementar y es eficiente en las operaciones de acceso. Como **desventajas** tiene la asignación de espacio cuando no se sabe el tamaño final y la fragmentación.

La **lista enlazada** no esta mu definido.

Con los **i-nodos** cada archivo tiene una pequeña tabla que contiene una lista de atributos y de direcciones de bloques de disco donde se encuentran los datos. Cuando se abre un archivo se carga su i-nodo desde el disco en memoria. Para archivos grandes una de las direcciones del i-nodo es uan direcciónd e un bloque de disco llamado bloque de direccionamiento indirecto simple (contiene direcciones de disco). Si no es suficiente,otra direccion del i-nodo, bloque de direccionamiento indirecto doble(bloque que contiene una lista de bloques d direccionamiento indirecto simple). Si no fuera suficiente, existe un bloque de direccionamiento **indirecto triple**.

5.5.2. Implementación de directorios

La tarea principal de un sistema de directorios es convertir el nombre en ASCII de un archivo en la inforamción necesaria par alocalizar sus datos.

5.5.3. Archivos compartidos

La conexión entre un directorio y un archivo compartido se llama *link*.

Esto introduce algunos problemas. Si los directorios contienen una copiad e las direcciones de disco, si alguno de los archivos se modifica, esta modificación no se verá reflejada en el otro archivo. Esto se soluciona con i-nodos o enlaces simbólicos[2].

Inconvenientes de los i-nodos, la realización de un link nocambia el propietario, únicamente cambia el valor del campo contador de enlaces. Si un usuario trata de biorrar el archivo hay un problema.

Inconvenientes de los enlaces simbólicos, si un usuario C elimina el archivo, cuando B quiera acceder a él el sistema le informará de que el fichero no existe.

5.5.4. Gestión del Espacio de Disco

Estrategias de almacenamiento:

- Asignación consecutiva
- División del fichero en bloques

Si el tamaño de lbloque es grande se tiene poca eficacia en la utilización del disco, pero si espequeño habrá muchos bloques siendo el acceso muy lento.

Para el seguimiento de bloques libres hay dos métodos: lista enlazada y mapa de bits.

Con las **cuotas de disco** el administrador del sistema asigna a cada usuario un número máximo de archivos y de espacio, y el sistema operativo se asgura de que el usuario no sobrepasa esas cuotas.

5.5.5. Fiabilidad del S.A.

Hay varias tares para mejorar la seguridad del sistema de archivos:

- **Gestión de bloques en mal estado**:
  - El hardware dedica un sector del disco par aguardar la listad e bloques malos. Cuando el controlador se inicializa, lee la lista de bloquea malos y coge un bloque de repuesto para sustituir los defectuosos, registrando el mapeado en la lista de bloques malos. En lo sucesivo, todas las referencias a los bloques malos, utilizarán los de repuesto.
  - El software, el usuario construye un archivo que contenga los bloques malos y quita estos de la lista de bloques libres.
- **Backups**
- **Consistencia**: Correspondencia entre la información del S.A. (dirección donde se encuentra la información) y l ainformación propiamente dicha. Programas que analizan la consistencia: nivel bloque y nivel archivo. ¿Añadir algo más?

5.5.6. Eficacia

El acceso a disco es muchos órdenes más lentoq ue el acceso a memoria. Para reducir los accesos a disco se utiliza la caché[3]

Algoritmos de gestión de caché. El más sencillo consisste en comprobar todas las solicitudes de lectura para ver si el bloque que se necesita está en la caché. Si está, la solicitud puede ser atentidad sin acceso al disco. Si el bloque no está en la caché primero se lee dentro de la caché y después es copiadoa. donde sea necesario.

Cuando la caché está llena y hay que expulsar algún bloque para dar cabida a otro, cualquiera de los algoritmos de sustitución vistos en paginación es válido (FIFO, segunda oportunidad o LRU).

Las referencias a caché son relativamente infrecuentes y por tanto es viable mantener los bloques en exacto orden LRU (con una lista enlazada).

Problema de consistencia del sistema de archivos cuando els sistema cae. Si un bloque crítico, como el i-nodo, se lee dentro de caché y se modifica, pero no se actualiza en disco, si el sistema cae, dejará el SA en un estado inconsciente.

5.6. Tratamiento Sistemas de Archivos en MINIX

5.6.1. Mensajes

El sistema de archivos acepta 29 tipos de mensajes (todos excepto 2, llamadas al sistema).

5.6.2. Distribución S.A.

Distribución típica del Sistema de Archivos:

- Bloque Boot
- Super Bloque: Información de la distribución del Sistema de Archivos (su tarea principal es dar información del tamaño de las 6 partes que lo forman)
- Mapa de Bits de los i-nodos
- Mapa de bits de las zonas
- i-nodos
- Datos

Cuando se arranca MINIX, el superbloque del dispositivo raíz, se lee dentro de una tabla de memoria. De igual forma, cuando se montan otros Sistemas de Archivo, se cargan sus superbloques en memoria.

La tabla de superbloques tiene algunos campos no presentes en el disco, como son: el dispositivo del cual viene, si es sólo lectura o no y un capo que se pone a 1 siempre que la versión de memoria sufre alguna modificación.

Antes de que un disco pueda ser utilizado como Sistema de Archivos, se le debe dar la estructura anterior.

5.6.3. Mapas de Bits

En MINIX, el seguimiento de los i-nodos y de las zonas libres se realiza a través de dos mapas de bits.

Cuando se crea un archivo, el Sistema de Archivos busa a través de los bloques de mapas de bits hasta que encuentra un i-nodo libre. Si todos los i-nodos de unbloque están ocupados, la rutina de búsqueda devuelve un 0, por eso nunca se emplea el i-nodo 0.

Al usar zonas en lugar de bloques se mejora la eficacia cuando se realiza un acceso al fichero (zona es una localización contigua de bloques).

5.6.4. i-nodos

En MINIX un i-nodo ocupa 32 bytes mientras que en UNIX ocupan 64 bytes. Cuandos e abre un archivo se carga su i-nodo en memoria (tabla de i-nodo). La tabla de i-nodo tiene más campos que los que están en disco: Número de i-nodo, dispositivo y contador.

5.6.5. Bloque caché

TERMINAR

[2] Se crea un nuevo fichero de tipo `LINK` en el directorio que quiere compartir el archivo, que contenga únicamente el path del archivo al que es enlazado

[3] Colección de bloques que pertenecesn lógicamente al disco, pero que se encuentran en memoria, por razones de eficacia

5.6.6. Directorios y Paths

El Sistema de Archivos, a partir del path viaja a lo largo del árbol de directorios hasta encontral el i-nodo del fichero. El directorio en MINIX es un archivo con entradas de 16 bytes, 2 para el número de i-nodo y 14 para el nombre del archivo.

Dibujo como do scuadrados con las cosas

5.6.7. Descriptores de Archivo

Cuando se abre un archivo, se devuelve un descriptor de archivo (número asociado a un archivo abierto) al proceso usuario para utilizarlo en posteriores llamadas.

Al igual que en el kernel y en el GM el SA mantiene parte de la tabla de procesos. 3 de sus campos son de especial interés

1. Apuntador directorio raíz (i-nodo)
2. Apuntador directorio de trabajo (i-nodo)
3. Array de apuntadores ordenador poe el número del descriptor del archivo

El tercer campo podría parecer suficiente que el contenido de este array fueran apuntadores a los i-nodos de los respectivos descriptores de archivo.

Desafortunadamente, esta solución non funciona por el hecho de que los archivos pueden ser compartidos.

5.6.8. Pipes y Archivos Especiales

ESTO YA ESTABA

6. Gestión de Procesos

Un **proceso** es, básicamente, un programa en ejecución. Es definido como una entidad que representa la unidad básica de información implementada por el sistema.

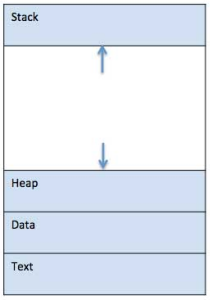


Figura 17: Modelo simplificado de un proceso en la memoria principal

Cuando un programa se carga en memoria y se convierte en proceso, puede ser dividido en 4 secciones:

- **Stack**: Contiene datos temporales, como parámetros de funciones, variables locales y direcciones de retorno.
- **Heap**: Memoria dinámicamente alojada para un proceso en su runtime.
- **Text**: Incluye la actividad actual representada por el valor del program counter y el contenido de los registros del procesador.
- **Data**: Contiene las variables globales y estáticas.

Todos los procesos tienen tres estados principales:

- **Bloqueo**: Un proceso espera un recurso externo o petición necesaria para su ejecución. No sigue la ejecución hasta que ocurre un evento externo.
- **Listo**: Estado en el que están los procesos que están preparados para ser ejecutados.
- **Ejecución**: Estado en el que están los procesos que se están ejecutando usando la CPU.

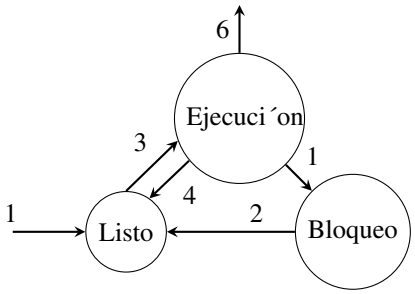


Figura 18: Modelo Básico de 3 estados

1. **Start**: Estado inicial cuando un proceso es creado/iniciado por primera vez.
2. **Listo**: El proceso espera ser asignado a un procesador por el SO y que puedan ejecutarse. Los procesos llegan a este estado después del estado de start o mientras lo están corriendo pero son interrumpidos por el scheduler para asignar la CPU a otro proceso.
3. **Ejecución**: Una vez que el proceso ha sido asignado a un procesador por el scheduler del SO, el estado cambia a running y el procesador ejecuta sus instrucciones.
4. **Bloqueo**: Un proceso se mueve al estado de bloqueo si tiene que esperar por un recurso, como la input del usuario, o a que un archivo esté disponible.
5. **Terminated**: Una vez que el proceso finaliza la ejecución o es terminado por el SO, se mueve a este estado donde espera a ser eliminado de la memoria principal.

Para que un **proceso** pueda ser **ejecutado** necesita varios requisitos:

- Una entrada en una tabla de procesos para que el planificador pueda gestionar su memoria y ejecución.
- Una zona de memoria reservada para este proceso.
- Una comunicación entre otros procesos para peticiones de datos.

En la tabla de procesos hay un array de estructuras con una estructura para cada proceso. Cada estructura de la tabla de procesos contiene cierta información y se llama **PCB** (*Process Control Block*). Es una estructura de datos mantenida por el SO para cada proceso. Es identificada por un entero que representa el ID del proceso (PID).

- **Estado del Proceso**: Estado actual del proceso, independientemente del estado.
- **Privilegios del Proceso**: Requerido para permitir o no acceso a los recursos del sistema.
- **Process ID**: Identificación única para cada proceso en el SO.
- **Puntero**: Puntero al proceso padre.
- **Contador de Programa**: Puntero a la dirección de la siguiente instrucción a ser ejecutada por el proceso.
- **Registros de CPU**: Donde el proceso necesita ser almacenado para ejecución en el estado 'listo'.
- **CPU Scheduling Info**: Prioridad de proceso y otra información que se necesite para schedule el proceso.
- **Información de gestión de memoria**: Page table, límites de memoria...
- **Accounting Information**: Incluye la cantidad de CPU utilizada para la ejecución del proceso, límite de tiempos...
- **IO Status Info**: Lista de dispositivos de IO allocated para el proceso.

Cuando se produce una **interrupción** se producen ciertas acciones:

- Se guardan los registros (PC).
- Se carga el PC de atención a interrupción de ese tipo.
- Se ejecuta en modo kernel y enmascaran las interrupciones.
- Se eejcuta el planificador.
- Se cargan los registros del proceso a ejecutar.
- Se pasa a modo usuario y se da el control al proceso a ejecutar.

6.1. Comunicación entre Procesos

Hay recursos a los que sólo puede acceder un único proceso a la vez, hay que evitar la **condición de carreras**.

La condición de carrera ocurre cuando dos o más procesos acceden a un recurso compartido sin control, de manera que el resultado del acceso depende del orden de llegada. Esa zona compartida se llama **sección crítica**. Para solucionar este problema el SO ha de prohibir el acceso a más de un proceso a una zona compartida al mismo tiempo.

Condiciones para una buena solución:

1. Dos procesos no podrán estar a la vez dentro de sus secciones críticas.
2. No se deben asumir suposiciones sobre velocidades relativas de los procesos.
3. Ningún proceso fuera de la sección crítica puede bloquear otros procesos.
4. Ningún proceso esperará indefinidamente para entrar en su sección crítica.

6.1.1. Exclusión Mutua con Espera Activa

Mientras un proceso está ocupado actualizando memoria compartida en su región crítica ningún otro entrará en la suya correspondiente.

Hay 5 formas:

1. **Inhabilitación de interrupciones**: Inhabilitar las interrupciones después de entrar en la región crítica y habilitarlas después de abandonarla para evitar que el planificador le quite el control. Esta solución es inviable debido a que no se le puede entregar el control de interrupciones a un proceso, es tarea del SO.
2. **Variables candado** (lock): Es una solución por software. Se utiliza una variable compartida `lock` (inicialmente `0`). Cuando un proceso quiere entrar en su sección crítica:
   - `lock = 0` → El proceso pone `lock = 1` y entra en la región crítica.
   - `lock = 1` → El proceso espera a que `lock = 0`.

   No cumple la condición 4, puede que tenga que esperar indefinidamente.
3. **Alternancia Estricta**: Solución por software. Cuando un proceso va a entrar en la sección crítica pasa el control a otro proceso para que bloquee los procesos que quieran entrar con una variable. Evita condiciones de carrera pero no cumple la condición 3.
4. **Solución Peterson**: Solución por software. Se definen 2 funciones de entrada y salida de la región crítica. Permite a dos o más procesos utilizar un recurso de único uso sin conflicto utilizando únicamente memoria compartida para comunicarse.
5. **Instrucción TSL** (*TEST and SET LOCK*: Solución por software con ayuda por hardware. Se utilizan 4 instrucciones de nivel ensamblador. E primer lugar, se copia el contenido de la variable en un registro y se pone la variable a 1. Se comprueba si el valor guardado es 1. Si es 0, se entra en la sección crítica. Si es 1, realiza de nuevo la comprobación. Debido a que se usan los registros, esta versión cumple todos los requisitos ya que si se diese el caso en el que en mitad de la ejecución de la reserva se produjese una interrupción dichos registros se guardarían.

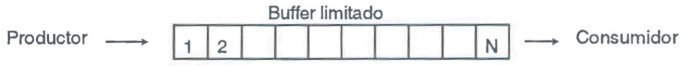
6.1.2. Dormir y Despertar

Peterson y TSL son soluciones buenas pero tiene como defectos la espera activa y otros más.

Primitivas de comunicación: Operaciones para conseguir la condición de exlusión mutua. Bloquean el proceso cuando no se les permite entrar en su región crítica sin consumir tiempo de CPU.
Para llevar a cabo este método son necesarias 2 primitivas de comunicación:

- **Sleep**: Llamada que suspende el proceso que la realiza hasta que otro proceso la despierte.
- **Wake-up**: Llamada que tiene por parámetro el proceso a despertar.

Surge el **problema productor-consumidor** (*ring buffer, bounded buffer*)



Intercambio de información entre processo por medio de un mecanismo de comunicación (buffer limitado de datos). Los productores se bloquean cuando el mecanismo (buffer) está lleno. Los consumidores se bloquean cuando el buffer está vacío.

Se da una situación de condición de carrera, el acceso a los elementos del buffer no está restringido → para solucionarlo añadir un bit de espera (wakeup).

6.1.3. Semáforos

Resuelven el problema de pérdidas de llamadas de Wake Up. Se utiliza una variable entera y definen dos operaciones:

- **DOWN**: Comprueba si el valor del semáforo `>0`. Si es así decrementa una unidad su contenido. Si es no (es cero) el proceso se duerme y queda bloqueado.
- **UP**: Comprueba si el valor del semáforo es `>0`. Si es así incrementa una unidad su contenido. Si es no y no hay ningún proceso dormido pone a `1` el semáforo. Si es no y hay algún proceso dormido aleatoriamente despierta a alguno y el semáforo permanece a `0`.

El SO inhabilita las interrupciones durante las operaciones a un semáforo. Para que este sistema funcione, las llamadas `UP` y `DOWN` deben de ser atómicas, el SO no puede quitar el control al proceso hasta que se ejecute.

Este método consigue evitar la condición de carrera y todas sus condiciones de implementación. El problema es que las llamadas `UP` y `DOWN` las hace el programa cliente. Si éste se equivoca de orden al realizar las llamadas puede llegar a provocar una excepción de bloqueo a todos los procesos que quieren acceder a la zona de acceso crítico. Para mejorarlo se definió el método de monitores.

6.1.4. Contadores de Eventos

Soluciona el problema de que el productor no puede escribir en el buffer ya que está lleno y se tiene que dormir. Se necesita saber el tamaño del almacén. No necesita exclusión mutua. utiliza un tipo especial de variable llamada contador de eventos y 3 operaciones:

- `READ (E)`: Devuelve el contador de eventos `E`.
- `ADVANCE (E)`: Incrementa el valor de `E` en `1` (atómicamente).
- `AWAIT (E,v)`: Espera hasta que $E \geq v$.

Los contadores de eventos **siempre crecen**, nunca decrecen y empiezan desde `0`.

Cuando el **productor** desea colocar un elemento en el buffer verifica si hay espacio por medio del `WAIT`. El **consumidor** espera a que el número de elementos que el productor ha colocado se hayan retirado.

6.1.5. Monitores

Permiten el bloqueo de procesos dentro del monitor utilizando operaciones atómicas como `WAIT` y `SIGNAL` y una variable de control. Sus **propiedades**:

1. En un monitor sólo un proceso puede estar activo en cualquier instante → exclusión mutua.
2. Permite el bloqueo de procesos dentro del monitor (porque no puedan proseguir → espera activa) a partir de la utilización de variables de condiciones y de las operaciones `WAIT` y `SIGNAL`:
   - `WAIT`: Sobre una variable de condición → el proceso se bloquea permitiendo a otro proceso entrar en el monitor. Ya no se necesitan llamadas a `UP` y `DOWN`.
   - `SIGNAL`: Sobre la variable de condición que ha dormido el proceso, despierta al proceso.

Forma de evitar que dos procesos permanezcan activos dentro del monitor después de una operación `SIGNAL`:

- **Hoare**: Continúa el proceso recién despertado, suspendiendo el otro.
- **Hansen**: El proceso que realiza `SIGNAL` deberá abandonar el monitor, `SIGNAL` aparecerá únicamente al final de una función monitor.

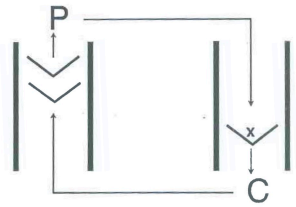
6.1.6. Transferencia de Mensajes

Existe un lugar donde almacenar los mensajes hasta que el consumidor los recoja. Hay dos primitivas de intercomunicación entre procesos:

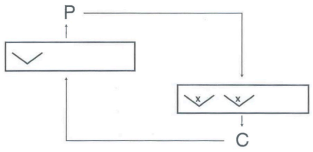
- `SEND (dest, msg)`: Envía un mensaje a un destino dado.
- `RECEIVE(origin, msg)`: Recibe un mensaje de un origen dado, si no hay ningún mensaje disponible el receptor se bloquea hasta que llegue uno.

Soluciones al **problema productor-consumidor**:

- **Pipes**: todos los mensajes son del mismo tamaño. El SO almacena mensajes enviados y no recibidos. Se puede bloquear si el consumidor no encuentra mensajes llenos y el productor no encuentra mensajes vacíos.

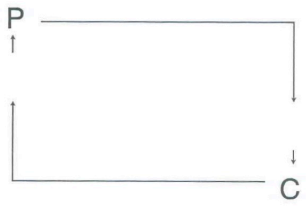


- **Buzón**: Estructura de datos. Zona de memoria donde cabe un cierto número de mensajes. Con buzones, los parámetros de dirección empleados en las llamadas de `SEND` y `RECEIVE` son buzones en lugar de procesos.



  El buzón de destino coge los mensajes que han sido enviados al proceso de destino pero que aún no han sido recibidos. Cuando un proceso intenta hacer un envío a un buzón que está lleno, es suspendido hasta que se extrae un mensaje.

- **Rendez vous**: Elimina cualquier forma de almacenamiento. Si se hace `SEND` antes que `RECEIVE` el proceso `SEND` se bloquea. Cuando se produce `RECEIVE` se copia el mensaje directamente del proceso que lo envía al que lo recibe.



6.2. Planificación

El **planificador** es la perte del SO que decide el orden de ejecución de los distintos procesos activos del sistema. Existen varios algoritmos de planificación y tienen que tener las siguientes **características**:

1. **Justo**: Asegurarse de que todos los procesos tengan un reparto justo de la CPU.
2. **Eficiencia**: Mantener la CPU activa el 100 % del tiempo.
3. **Tiempo de respuesta**: Minimizar el tiempo de respuesta para los procesos activos.
4. **Rendimiento**: Minimizar el tiempo que los usuarios deben esperar para la salida.
5. **Throughput**: Maximizar el número de trabajos procesados.

Para poder realizar estas funciones, es necesario un reloj interno que produzca interrupciones cada cierto tiempo para que el SO tome el control y decida cuál es el siguiente proceso a ejecutar.

6.2.1. Round Robin

A cada proceso la CPU le asigna un tiempo de ejecución. Es cíclico, la asignación del procesador es rotatoria. Con este algoritmo todos los procesos tienen la misma prioridad. A cada proceso se le asigna un tiempo determinado de ejecución (*quantum*).

En el caso de que finalize el quantum y el proceso no haya finalizado la ejecución se le quitará el control al proceso y se le dará el siguiente en la cola.

Los *quantum* pueden tener distintas duraciones:

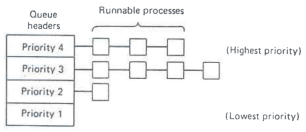
- Corto (20ms): Muchos cambios de contexto, da lugar a poco rendimiento.
- Largo (500ms): Respuesta pobre en entornos interactivos, puede tardar mucho en ejecutar un proceso si existen muchos en el sistema.

6.2.2. Prioridades

A cada proceso se le asigna una prioridad. El proceso con mayor prioridad es el que se ejecuta. Asignación de prioridades:

- **Estática**: No cambia en ejecución. Fácil de implementar. Es poco sensible a los cambios de entorno.
- **Dinámica**: Usa el algoritmo $1/f$, si un proceso tiene un *quantum* de 100ms y un proceso usa 2ms se le asigna una prioridad de 50. f es la fracción del último *quantum* utilizado en el proceso. Es bastante difícil de implementar.

Hay veces que interesa agrupar los procesos por clases de prioridad, utilizando planificación de prioridades entre las clases y planificación round robin entre los procesos de cada clase.



Si hay varios procesos con el mismo nivel de prioridad se suele usar un mecanismo adicional para decidir.

6.2.3. Colas Múltiples

Variante de la planificación en clases de prioridad de forma dinámica. Se desarrolla en los primeros sistemas CTSS en los que sólo se podía mantener un proceso en memoria. Los procesos de mayor prioridad son asignados con 1 quantum de ejecución, las demás clases inferiores tendrán 2^{n-1} quantums.

Con este mecanismo se consigue priorizar los procesos de ejecución rápida. Si un proceso consume la totalidad de su tiempo de ejecución se le baja una clase.

Hay 4 clases de prioridad:

- Terminal
- I/O
- Quantum Corto
- Quantum Largo

Algoritmo de planificación:

1. Proceso bloqueado en espera entrada terminal despertado → prioridad más alta.
2. Proceso bloqueado en espera I/O despertado → prioridad I/O
3. Proceso permanece ejecutable cuando se termina su quantum → quantum corto
4. Proceso emplea varias veces su quantum sin bloquearse por el terminal o por I/O → quantum largo

6.2.4. Primero Trabajo Más Corto

Algoritmo indicado para sistemas batch en los que son conocidos los tiempos de ejecución. Consiste en ejecutar los procesos de menor tiempo de ejecución en primer lugar. De esta manera, se consiguen menores tiempos medios de respuesta. Para saber el tiempo aproximado de ejecución se utilizan técnicas de *aging*. Se estima el siguiente valor tomando una medida ponderada de los valores medidos y uno estimado previamente. Tiene una dificultad, conocer a priori la duración del trabajo. Este sistema de planificación es óptimo únicamente cuando todos los trabajos están disponibles simultáneamente.

6.2.5. Predeterminada

Consiste en adquirir un compromiso con el usuario y cumplirlo. Se realiza una comparación entre el uso de la CPU y la CPU asignada al proceso. Después, el algoritmo se encarga de ejecutar los procesos que tengan una relación más baja hasta que posean una relación por encima de su competidor más próximo.

Tiene las siguientes dificultades:

1. El sistema ha de ejecutar el trabajo sin degradar el servicio a otros usuarios.
2. El sistema debe planificar la asignación de recursos completamente hasta el *deadline*. Pueden llegar trabajos que introduzcan demandas que el sistema no puede predecir.
3. Varios trabajos *deadline* a la vez → introducir sofisticados métodos de overhead.

6.2.6. Planificación a Dos Niveles

Los tiempos de intercambio entre memoria principal y disco son de uno o dos órdenes de magnitud superiores a los de intercambio entre los procesos que se encuentran en memoria principal.

Hay dos formas, una de ellas mantiene los procesos en la memoria principal mientras se ejecuta y otra es que los procesos se mantienen en el disco hasta que puedan ser ejecutados.

- Planificador de bajo nivel → Procesos en MP
- Planificador de alto nivel → Desplazamiento entre MP y disco

Criterios de decisión de un planificador de alto nivel:

1. ¿Cuánto ha pasado desde que el proceso fue intercambiado?
2. ¿Cuánto tiempo de CPU ha consumido el proceso recientemente?
3. ¿Cómo de grande es el proceso?
4. ¿Cómo de alta es la prioridad del proceso?

6.2.7. Evaluación

Métodos analíticos: Consisten en aplicar el algoritmo seleccionado a una determinada carga del sistema y producir una fórmula o número que evalúa la eficacia del algoritmo para esa carga.

- Modelo determinista: Se toma una carga particular establecida con anterioridad y se determina la eficacia de cada algoritmo para esa carga. Produce números exactor y permite la comparación de algoritmos. Requiere números exactos en sus datos de entrada y los resultados obtenidos se pueden aplicar únicamente a esos datos.
- Modelo de colas: Según este modelo el ordenador se describe como un servidor, donde se determinan las distribuciones de probabilidad tanto del tiempo de llegada entre procesos como el tiempo de empleo de CPU. A partir de estas dos distribuciones se pueden calcular los valores medios de *throughput*, utilización, tiempo de espera... Las distribuciones de llegada y servicio se definen por relaciones matemáticas que muchas veces suponen hacer simplificaciones que restan exactitud al método, por ser éstas sólo una aproximación a los sistemas reales.

Simulación: Permite conseguir una evaluación más precisa de los algoritmos de evaluación (siempre refiriéndose a los niveles de confianza). Supone realizar una serie de tareas.

Implementación: Única forma exacta de evaluar un algoritmo de planificación. Consiste en introducir el algoritmo en el SO para ver cómo responde bajo condiciones operativas reales.

6.3. Tratamiento Procesos MINIX

Comprobar si entra, en los apuntes de Carmen no está

7. Gestión E/S

7.1. HW E/S

7.1.1. Dispositivos E/S

- **Dispositivos Bloque**: Almacenan información en bloques de tamaño fijo, cada uno con su propia dirección. Permiten leer o escribir cada bloque con independencia de los demás. Discos.
- **Dispositivos Carácter**: Liberan o captan un conjunto de caracteres sin contemplar ninguna estructura de bloque. Terminales o impresoras.
- **Relojes**: No responden a ninguna estructura de bloques direccionables, no aceptan o generan conjuntos de caracteres y causan interrupciones en intervalos definidos de tiempo.

7.1.2. Controladores de Dispositivo

Un **controlador** es responsable del control de los dispositivos exteriores y del intercambio de datos entre los dispositivos y la memoria principal y/o registros de la CPU.

Existen 3 formas de acceso a E/S:

- **E/S programadas**: La CPU ejecuta la instrucción de E/S. Extrae o envía datos entre la MP y el controlador y espera a que el dispositivo termine. La CPU espera hasta que el dispositivo conteste y de forma general, como los dispositivos de E/S son mucho más lentos que la CPU, se pierde eficiencia.
- **Interrupciones**: La CPU envía el comando al controlador y sigue ejecutando sin esperar a que el controlador conteste. En el momento en el que el dispositivo contesta se produce una interrupción y el gestor de interrupciones maneja dicha interrupción para ejecutar la rutina de atención a dicha interrupción. No hay pérdida de eficiencia ya que la CPU no espera.
- **Acceso directo a memoria**: Parte del HW es capaz de controlar varios dispositivos de E/S por encima de los controladores. De esta forma se tiene un acceso directo desde la memoria al dispositivo, siendo éste quien genera las interrupciones para el SO.

7.2. SW E/S

Estructura en capas:

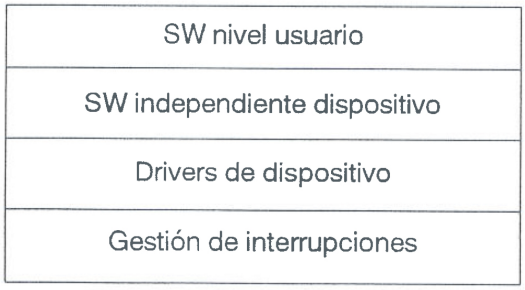
- **Capas inferiores**: Esconden las peculiaridades del HW a las superiores.
- **Capas superiores**: Presentan una interfaz simple al usuario.

7.2.1. Objetivos

- **SW independiente del dispositivo**: Tiene que existir la posibilidad de escribir programas que puedan ser utilizados con archivos que se encuentren en cualquier dispositivo, sin tener que modificarlos para acondicionarlos a las características de los distintos dispositivos.
- **Nombramiento uniforme (independiente del dispositivo**: Todos los archivos y dispositivos se direccionan de la misma forma, a partir del path name, sin que dependa este direccionamiento del tipo de dispositivo.
- **Gestión de errores**: Los errores deben de ser corregidos tan cerca como sea posible del lugar en el que se producen.
- **Conversión de las transferencias asíncronas en síncronas**: Hacer que aquellas operaciones que son activadas por interrupciones aparezcan como bloqueadas a los programas usuario.

- **Clasificación de dispositivos**: Gestionar al mismo tiempo los dispositivos dedicados y compartidos.

Dividir el SW en 4 capas:



7.2.2. Gestión de Interrupciones

Trata de ocultar las interrupciones. Siempre que se produzca un comando E/S y se espere una interrupción tendremos un proceso bloqueado. Cuando se produzca la interrupción se desbloquea el proceso previamente bloqueado.

7.2.3. Drivers de Dispositivo

Un **driver** es un programa dependiente de dispositivo, cada uno tiene el suyo. El driver conoce los registros del controlador y los detalles del dispositivo. Su tarea consiste en aceptar órdenes del SW de la capa superior (independiente), traducirlas a términos concretos (establecer las operaciones que se tienen que realizar y en qué orden) y hacer que se realicen.

Una vez que haya emitido su comando o comandos se autobloqueará hasta que se produzca la correspondiente interrupción, momento en el que se desbloqueará.

Una vez desbloqueado, comprobará si se han producido errores y pasará los datos al SW independiente.

7.2.4. SW Independiente del Dispositivo

Realiza las tareas E/S comunes a todos los dispositivos:

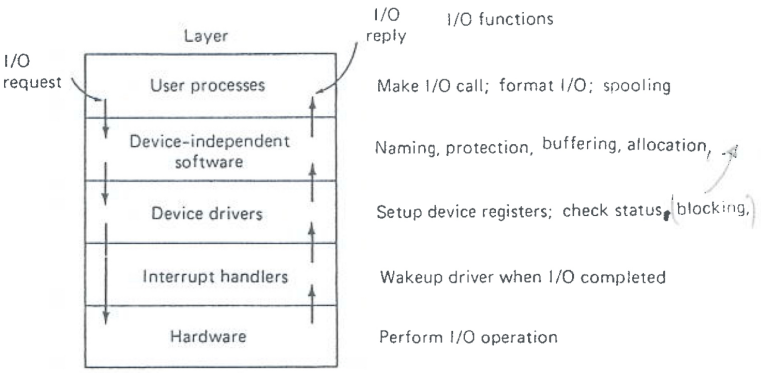
1. Provee una interfaz uniforma al nivel usuario.
2. Cuando se nombra un dispositivo se encarga de asignarle el driver correspondiente.
3. Previene a los dispositivos de accesos no autorizados.
4. Permite gestionar dispositivos abstractos que emplean un mismo tamaño de bloque, independientemente del tamaño físico real del sector.
5. Gestiona los problemas derivados del almacenamiento, tanto en los dispositivos de bloque como en los de carácter.
6. Asigna espacio para los nuevos archivos que se van a crear.
7. Realiza el tratamiento apropiado según la naturaleza del dispositivo, dedicado o compartido.
8. Gestiona los errores que no se han podido corregir en capas inferiores.

7.2.5. SW Nivel de Usuario

Aunque la mayor parte del SW E/S se encuentra dentro del SO una pequeña parte corre fuera de éste:

1. Las llamadas al sistema (incluyendo las relacionadas con E/S), que se implementan a través de subrutinas de librería.
2. Algunas funciones que se implementan también a través de subrutinas de librería.
3. Algunos sistemas (procesos) se realizan desde el nivel usuario.

Resumen:



7.3. Tratamiento E/S en MINIX

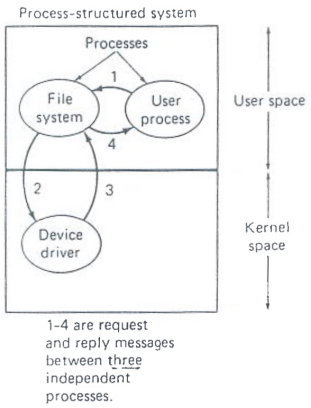
7.3.1. Gestión de Interrupciones

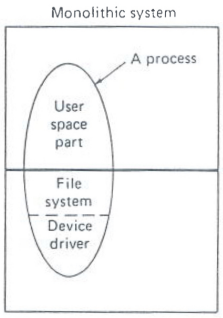
Cuando un driver de dispositivo comienza una operación de E/S, se bloquea en espera de un mensaje. Este mensaje es generado en la parte del S.O. que se encarga de la gestión de interrupciones (capa 1).

7.3.2. Drivers de Dispositivo

Un driver por cada tarea. Los drivers pertenecen al kernel, lo que permite un fácil acceso a la tabla de procesos y otras estructuras importantes. La comunicación entre usuario y sistema se realiza mediante llamadas al sistema que generan interrupciones que a su vez producirán mensajes para los procesos de nivel 2 (GM y SA). De la misma manera, éstos elegirán el driver a usar produciendo una interrupción para poder usar dicho driver desde el gestor de interrupciones.

Formas de estructurar la comunicación entre el usuario y el sistema:





The user-space part calls the kernel-space part by trapping. The file system calls the device driver as a procedure. The entire operating system is part of each process.

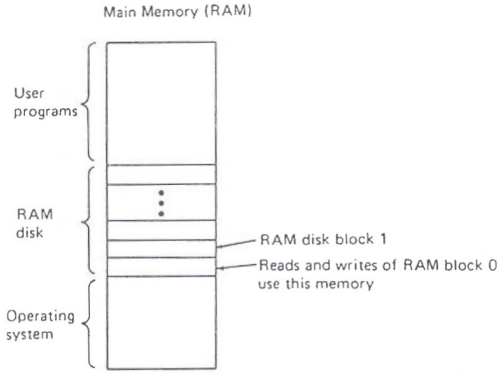
Estructura general de los drivers de dispositivo:

- Capturar un mensaje
- Ejecutar lo que indica el mensaje
- Devolver un mensaje de respuesta

7.4. Memoria RAM

7.4.1. HW y SW

Dispositivo más sencillo orientado a bloque. Dispone de dos comandos, lectura (bloque) y escritura (bloque). Parte de la MP de nuestro sistema.



7.4.2. Driver Memoria RAM en MINIX

Permite montar y desmontar fácilmente el sistema de archivos. Existen 2 números para la asignación de dispositivos:

- Número mayor: Designa el tipo de dispositivo, en este caso RAM.
- Número menor: Designa el tipo de dispositivo dentro del tipo anterior, puede ser de `0` a `3`.
  - `0`: `dev/ram`
  - `1`: `dev/mem` Superusuario
  - `2`: `dev/kmem` Superusuario
  - `3`: `dev/null` Tamaño cero

7.5. Discos

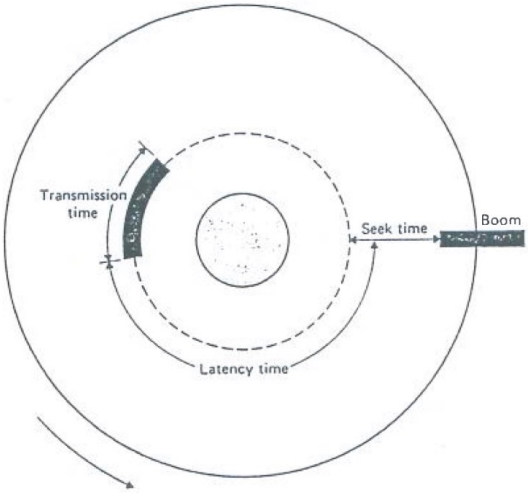
7.5.1. HW

Dispositivos orientados a bloque de almacenamiento masivo y no volátil. Los datos son registrados y recuperados del disco a través de una bobina conductora → cabeza. Durante una operación de lectura/escritura, la cabeza permanece fija, girando el disco debajo de ella. Los *tracks* son anillos concéntricos de la misma anchura que la cabeza.

7.5.2. SW

Tiempo de lectura/escritura de un bloque:

1. Tiempo de búsqueda (*seek time*)
2. Tiempo de rotación (*latency time*)
3. Tiempo de transferencia (*transmission time*)



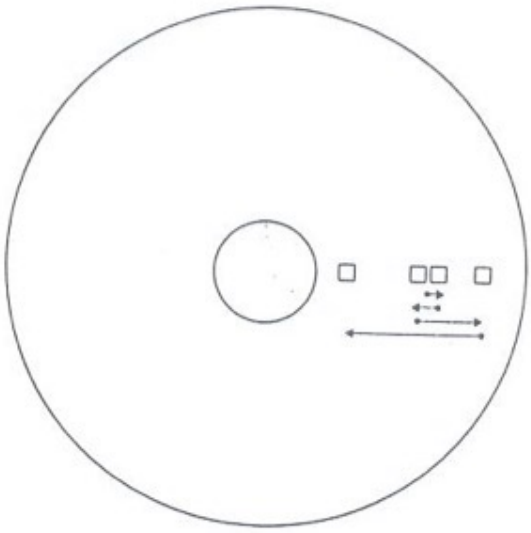
La **planificación de acceso al disco** determina el método más eficiente de atención a las peticiones. Tiene las siguientes características:

1. Throughput (Maximizar): Número de solicitudes atendidas por unidad de tiempo.
2. Tiempo de respuesta medio (minimizar)
3. Varianza del tiempo de respuesta.

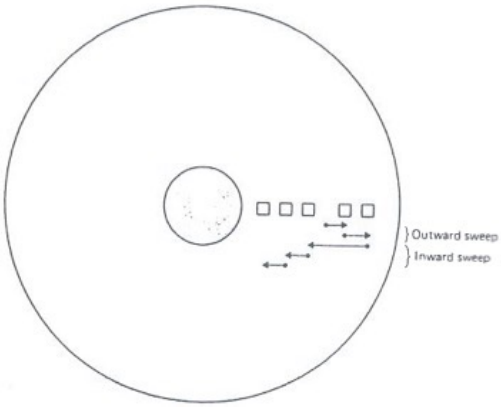
Optimización del tiempo de respuesta:

- **FCFS** (*First Come First Served*): No hay reordenamiento de la cola.

- **SSTF** (*Shortest Seek Time Fisrt*): Sirve la solicitud con menor tiempo de búsqueda, aunque no sea la primera en la cola.



- SCAN



Terminar

8. Gestión de Memoria

Terminar

9. Gestión de Archivos

Terminar

ASI

Segunda Parte: Seguridad en S.I.

3º Ingeniería de Telecomunicaciones — UPV/EHU

"Under-promise and over-deliver."

Javier de Martín – 2016/17

10. Parte 1

10.1. Introducción

The art of war teaches us to rely not on the likelihood of the enemy's not coming, but on our own readiness to receive him; not on the chance of his not attacking, but rather on the fact that we have made our position unassailable.

Ataque: Cualquier acción que comprometa la seguridad de cualquier componente de un SI de una organización.

Servicios de Seguridad: Un proceso o equipo que lo contiene diseñado para detectar, prevenir o recuperarse de un ataque.

Mecanismos de Seguridad: Servicio de procesado o comunicaciones que aumenta la seguridad de los sistemas de procesado o transmisión de información de una organización. Se crean para protegerse de ataques contra la seguridad, y se hace uso de uno o varios mecanismos para proporcionar cada servicio.

Los mecanismos de seguridad OSI están diseñados para:

- Prevenir
- Detectar
- Recuperarse

Los mecanismos pueden ser clasificados como preventivos, detectivos y recuperables.

10.2. Fundamentos de Seguridad en los SI

10.2.1. Ataques

Un **ataque pasivo** se basa en la monitorización y estudio del sistema y atentan contra la confidencialidad. Fáciles de detectar ya que no realizan modificaciones en el sistema.

- Divulgación de la información: Se difunde la información obtenida.
- Análisis de tráfico: Si la información transcurre encriptada, se puede obtener de ahí.

Un **ataque activo** implica la modificación o inserción de elementos en el sistema. Tienen características opuestas a los ataques pasivos.

- Usurpación de identidad
- Retransmisión
- Modificación de mensajes
- Denegación de servicios.

10.2.2. Servicios

Los **servicios de seguridad** en OSI son:

- Autenticación
- Control de Acceso
- Confidencialidad
- Integridad de datos
- No repudio

La **autenticación** entre entidades pares permite verificar que la entidad es quien dice ser, utilizado en las fases de establecimiento y transferencia de datos. La autenticación del origen de datos no proporciona protección frente a duplicación o modificación. Se emplea en tareas de autorización (concesión de derechos) y contabilidad (control de recursos).

El **control de acceso** permite proteger los recursos del sistema contra la utilización no autorizada. Para realizar el control de acceso es obligatorio identificarse. Define perfiles y granularidad.

Protección frente a revelaciones no autorizadas y ataques pasivos. Cuatro tipos:

- Orientado a conexión → datos transmitidos durante una conexión. TCP.
- No orientado a conexión → unidades simples. UDP.
- De campo selectivo → Campos específicos en una conexión o unidad de datos.
- Flujo de tráfico → Contra análisis de tráfico.

Protección contra modificaciones no autorizadas. Se dividen en función del objetivo a proteger (servicio de integridad orientado a conexión o no orientado a conexión) y en función al alcance de la herramienta (servicio con y sin recuperación[4]).

No repudio permite proteger contra las posibles negociaciones de acciones realizadas. Hay dos tipos:

- Con prueba de origen → Se puede asegurar que el mensaje ha sido enviado por el origen original.
- Con prueba de destino → Al emisor se le garantiza la recepción en el destino.

10.2.3. Mecanismos

Los mecanismos de seguridad de OSI son:

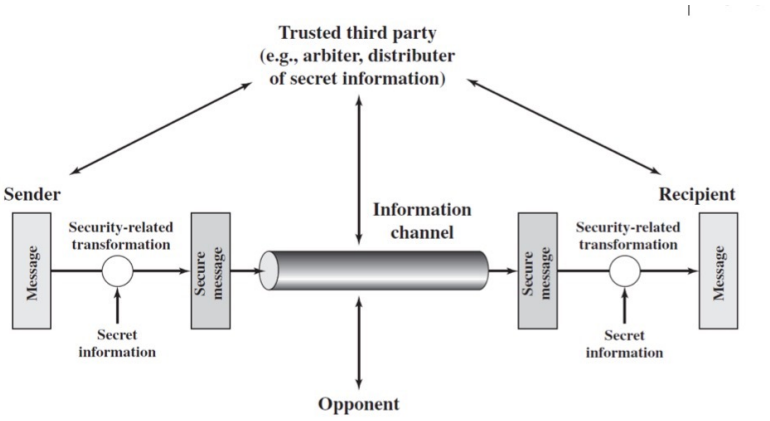
- Específicos: Técnicas destinadas a facilitar un servicio, por ejemplo: cifrado, firma digital, control de acceso, integridad, intercambio de autenticación, relleno de tráfico, control de encaminamiento y certificación.
- Generalizados: Pueden considerarse como aspectos de gestión de seguridad (relacionados directamente con el nivel de seguridad requerido), por ejemplo: confianza, etiquetas de seguridad, detección, recuperación...

También es interesante considerar mecanismos a distintos niveles:

[4] Si se nota que ha habido una modificación se puede pedir una retransmisión del mensaje original.

- Mecanismos a nivel de Red: Se autentica y cifra todo el tráfico de red, se protege a todas las aplicaciones, requieren la misma solución en todos los nodos, IPv6 e IPsec fueron diseñados con ello en mente. Hay soluciones parciales, entre routers de la organización VPN.
- Mecanismos a nivel de Aplicación: Dado que no hay mecanismos globales, se buscan soluciones para cada una de las aplicaciones que nos interesa.

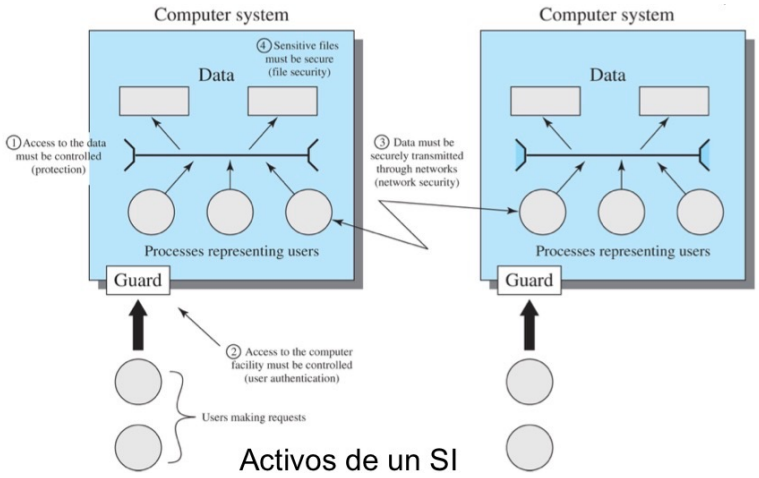
Modelo de seguridad de red



La NIST define la seguridad como la triada CIA: Confidentiality, Integrity y Availability:

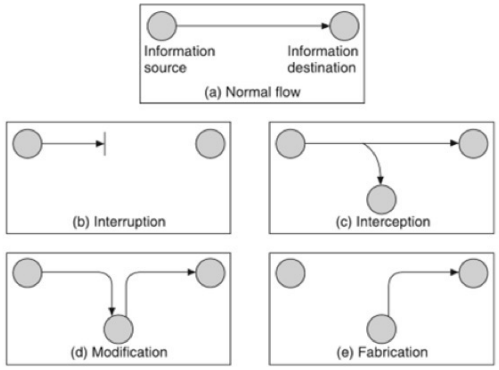
- Confidencialidad: Protección del acceso (o revelación) a la información solo para personas autorizadas, incluyebndo medios para proteger privacidad e información de propietario. Una pérdida de confidencialidad supone la revelación no autorizada de información
- Integridad: PRotección frente a modificación o destrucción de inforamción de forma no autorizada (incluye no repudio y autenticidad). Una pérdida de integridad supone la modificación o destrucción no autorizada de información.
- Disponibilidad: Asegurar el acceso y la utilización a tiempo y de forma confiable a la información. Una pérdida de disponibilidad supone la interrupción en el acceso o utilización de una información o sistema de información.
- Autenticidad: Propiedad de ser genuino y con capacidad para ser verificado y confiable. Confiabilidad en la validez de una transmisión. Que se pueda verificar que los usuarios son quienes dicen ser, o que las entradas a un sistema proceden de una fuente confiable.
- Contabilidad: Como no es posible que los sistemas sean completamente seguros, las partes autorizadas deberían disponer de mecanismos para trazar eventos de seguridad. Los sistemas deben almacenar registros de sus actividades para permitir análisis forenses posteriores para ayudar en la resolución de conflictos.
- Proporciona no repudio, disuasión, detección y prevención de intrusión, y posibilita reuperación y acciones legales posteriores.

Activos, amenazas y ataques



- **Amenazas**: "*Condición del entorno de sistema de información que, dada una oportunidad, podría dar lugar a que se produjese una violación de la seguridad*".
- **Ataque**: "*Un ataque es la realización de una amenaza*".

Una **amenaza a la seguridad** es la posibilidad de violación de seguridad, que existe cuando una entidad, circunstancia que puede causar daños. Un peligro que se deriva de una posible explotación de una vulnerabilidad. Para estudiar los tipos de amenazas se suele partir de la consideración que la función de un SI es proporcionar información, se considera que hay un flujo de información de una fuente a un destino.



Flujo normal de información y amenazas contra la seguridad

10.3. Técnicas Criptográficas

10.3.1. Introducción a la criptografía

- **Esteganografía** tiene como objetivo ocultar el mensaje.
- **Criptografía**: Objetivo es ocultar el significado del mensaje, el proceso se conoce como codificación.
- **Criptología**:; Ciencia que trata los problemas teóricos relacionados con la seguridad en el intercambio de mensajes en clave entre un emisor y un receptor a través de un canal de comunicaciones. Criptografía + Criptoanálisis.
- **Criptografía**: Técnicas de cifrado o codificado destinadas a alterar las representaciones lingüísticas de mensajes con el fin de hacerlos ininteligibles a receptores no adecuados. Tradicionalmente asociada sólo al cifrado.

Elementos clave del sistema: Texto en claro y texto cifrado, algoritmo de cifrado/descifrado y clave de cifrado (K_A) y descifrado (K_B).
Los sistemas criptográficos se caracterizan en función de:

- Tipo de operaciones utilizadas para transformar el plaintext en ciphertext: sustitución y transposición.
- Número de claves utilizadas: simétrica y asimétrica.
- Forma en la que se procesa el plaintext: Bloque y stream.

Tipos básicos de cifrado en función del tipo de operaciones:

- Sustitución: Las unidades de texto plano son sustituidas con texto cifrado siguiendo un sistema regular. Las unidades pueden ser una sóla letra o grupos más grandes. El receptor descifra el texto realizando la operación inversa. Hay varios tipos:
  - Simple: Opera sobre letras simples. Hay mono alfabético, si la sustitución es simple para todo el mensaje, y polialfabético, si utiliza diferentes sustituciones en diferentes partes de un mensaje.
  - Poligráfico: Opera sobre grupos de letras.
- Transposición: Las unidades del texto plano son cambiadas usando una ordenación diferente y normalmente bastante compleja, pero las unidades en sí mismas no son modificadas.

Criptografía de clave secreta o simétrica: Se utiliza la misma clave para cifrar y descifrar.

$$K_A = K_B = K_{AB} \quad K_{AB}(K_{AB}(m)))$$

La clave es **secreta** (K_{AB}) es conocida por los dos participantes de la comunicación.

Criptografía de clave pública o asimétrica: Cada usuario cuenta con un par de clave pública (K^+)/privada (K^-). Dada la clave pública, es computacionalmente inviable obtener la clave privada. La clave privada es secreta y por tanto, conocida únicamente por el propietario legítimo de la misma, mientras que la clave pública es pública, y por tanto conocida por todos.
Todo lo que se cifra con la clave pública únicamente puede ser descifrado con la correspondiente clave privada y vice-versa.

$$K^+(K^-(m))) = m \quad \text{ó} \quad K^-(K^+(m))) = m$$

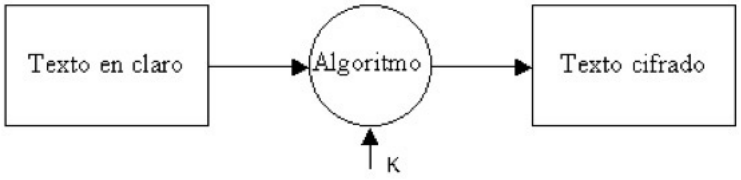
Para garantizar la confidencialidad de los mensajes enviados, todos los emisores han de cifrar los mensajes con clave pública del usuario de destino y sólo será el usuario de destino que podrá descifrar el mensaje.
Criptoanálisis es la parte de la criptología que se dedica al estudio de sistemas criptográficos con el fin de encontrar debilidades en los sistemas y romper su seguridad. Buscar romper o forzar el código. Hay varios tipos de ataques según el conocimiento previo del atacante:

- **Ataques *ciphertext-only***: Cuando sólo se dispone del texto cifrado. Por ejemplo: Fuerza bruta, análisis de frecuencias o método Kasiski.
- **Ataque *known-plaintext***: El atacante conoce el texto original. En ocasiones, el atacante conoce una parte del texto o alguna palabra probable.
- **Ataque *chosen-plaintext***: El atacante tiene la capacidad de definir un texto y obtener el correspondiente texto cifrado. El objetivo es poder descifrar textos que se descicfren e ese descifrador.

Si un sistema es seguro frente a ataques *chosen-plaintext* también es seguro frente a ataques *known-plaintext* y *ciphertext-only*.

10.3.2. Cifrado

El **cifrado** es la puesta en clave (*ciphertext*) de un texto (*plaintext*) mediante una función parametrizada por una clave. Previene ataques contra la confidencialidad.

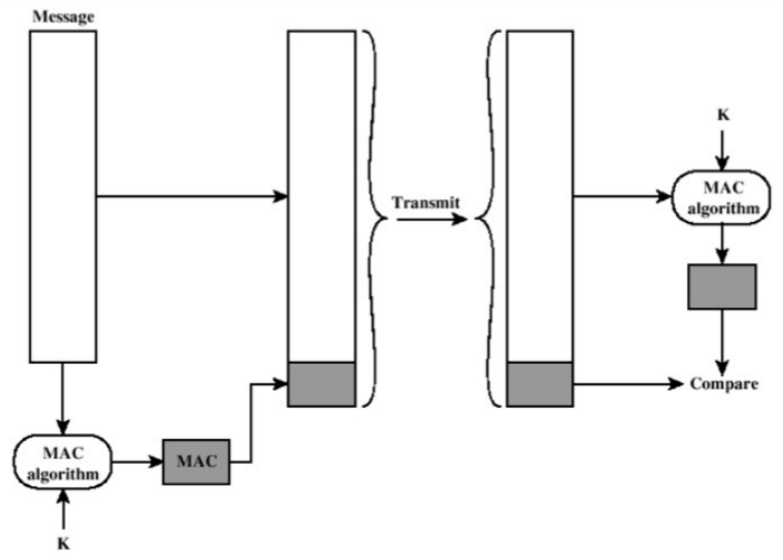


10.3.3. Códigos de Autenticación de Mensaje (MAC)

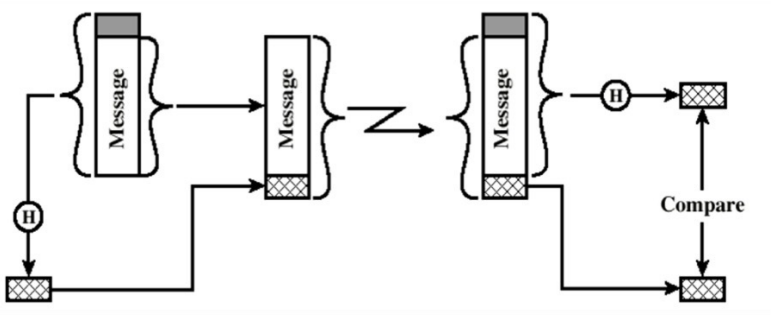
Permiten proteger la integridad y autenticidad de origen de los mensajes. Comprueban que el origen del mensaje es quién dice ser y que dicho mensaje no ha sido modificado durante su transmisión. Pueden considerarse sinónimos, si un mensaje es modificado durante su transmisión, ya no procede de su emisor légitimo sino de quien lo modificó.
A veces puede ser necesario garantizar la integridad del mensaje pero no su confidencialida.

Una **función *hash*** es una función que mapea cadenas de bits de longitud arbitraria a cadenas de longitud fija de n bits (huella dactilar), es una función resumen y es de un único sentido. Se puede ir del texto al resumen pero en sentido inverso es imposible.

MAC (*Message Authentication Code*) es un código que permite la autenticación de mensajes mediante técnicas criptográficas de clave simétrica. Los algoritmos de MAC toman como entrada dos parámetros (mensaje y clave simétrica) y generan una salida de longitud fija con la característica de que es computacionalmente inviable producir la misma salida sin conocer la clave.



Un **ejemplo de código MAC** es HMAC (*Hash-based Message Authentication Code*) y es un tipo específico de códigos MAC, el algoritmo utilizado para calcular el código MAC se basa en la utilización de funciones hash en lugar de, por ejemplo, algoritmos de cifrado.



Características de los códigos de autenticación de mensajes:

- Compresión: Mapean una entrada de longitud fija finita arbitraria a una salida de longitud finita fija.
- Facilidad de cómputo: Cual.

10.3.4. Firmas Digitales

10.3.5. Frescura de Mensajes

10.3.6. Distribución de Claves

10.4. Seguridad en SI

10.5. Situación Actual de la Seguridad

10.6. Gestión de la Seguridad

11. Parte 2

12. Parte 3

13. Parte 4

14. Hilos

Un **hilo** es una secuencia de tareas encadenada muy pequeña que puede ser ejecutada por un SO. Un hilo se **diferencia** de un proceso en que los últimos son, generalmente, independientes y actúan sólo a través de mecanismos de comunicación dados por el sistema.

15. Gestión de Procesos

15.1. Introducción

Se define un **proceso**

16. Ejercicios

16.1. Ejercicios búsqueda en tablas memorie TLB

TAE $= p_{acierto} \cdot (t_{busqueda} + TAM) + (1 - p_{acierto}) \cdot (t_{busquedayact} + t_{accesom}$

TAM: Tiempo de Acceso a Memoria
TAE: Tiempo de Acceso Efectivo a Memoria