Tema 3:
Seguridad y Protección

Grado en Ing. Informática

Curso 2024/25

Profesor: Manuel E. Acacio Sánchez

Índice

3.1 Introducción

- Aunque puedan parecer lo mismo, en el ámbito de los SSOO **seguridad** y **protección** se refieren a cosas diferentes:
  - **Protección**: Mecanismos que articula el SO para proteger la información, los usuarios, los procesos… → Tiene carácter INTERNO
  - **Seguridad**: concepto más amplio y general, que además de la protección incluye también seguridad física (p. ej. ¿quién tiene acceso a los edificios?), política de copias de seguridad etc.

Índice

3.3.1 Política y mecanismo

- **Protección**: Parte de la seguridad que corresponde al SO, es un **mecanismo** para controlar el acceso de procesos o usuarios a los recursos (físicos y lógicos) de un sistema de computación
- **Política** *vs*. **Mecanismo**
  - Las políticas establecen qué se va a proteger y qué procesos (y por lo tanto usuarios) van a poder acceder a qué recursos, etc. (ej. Los contables no pueden acceder a la base de datos de *marketing*)
  - Existen entonces una serie de mecanismos que ofrece el SO para poder implementar las políticas (permisos, ficheros, usuarios, etc.)
  - Las políticas pueden cambiar y además dependen de cada organización ⇒ Los SO deben ofrecer mecanismos generales y flexibles que permitan implementar las políticas

3.3.2 Dominios de protección

- Un sistema de cómputo se ve como un conjunto de **objetos** que necesitan protección. Tanto objetos hardware (CPU, memoria, impresoras…) como software (procesos, ficheros, bases de datos…)

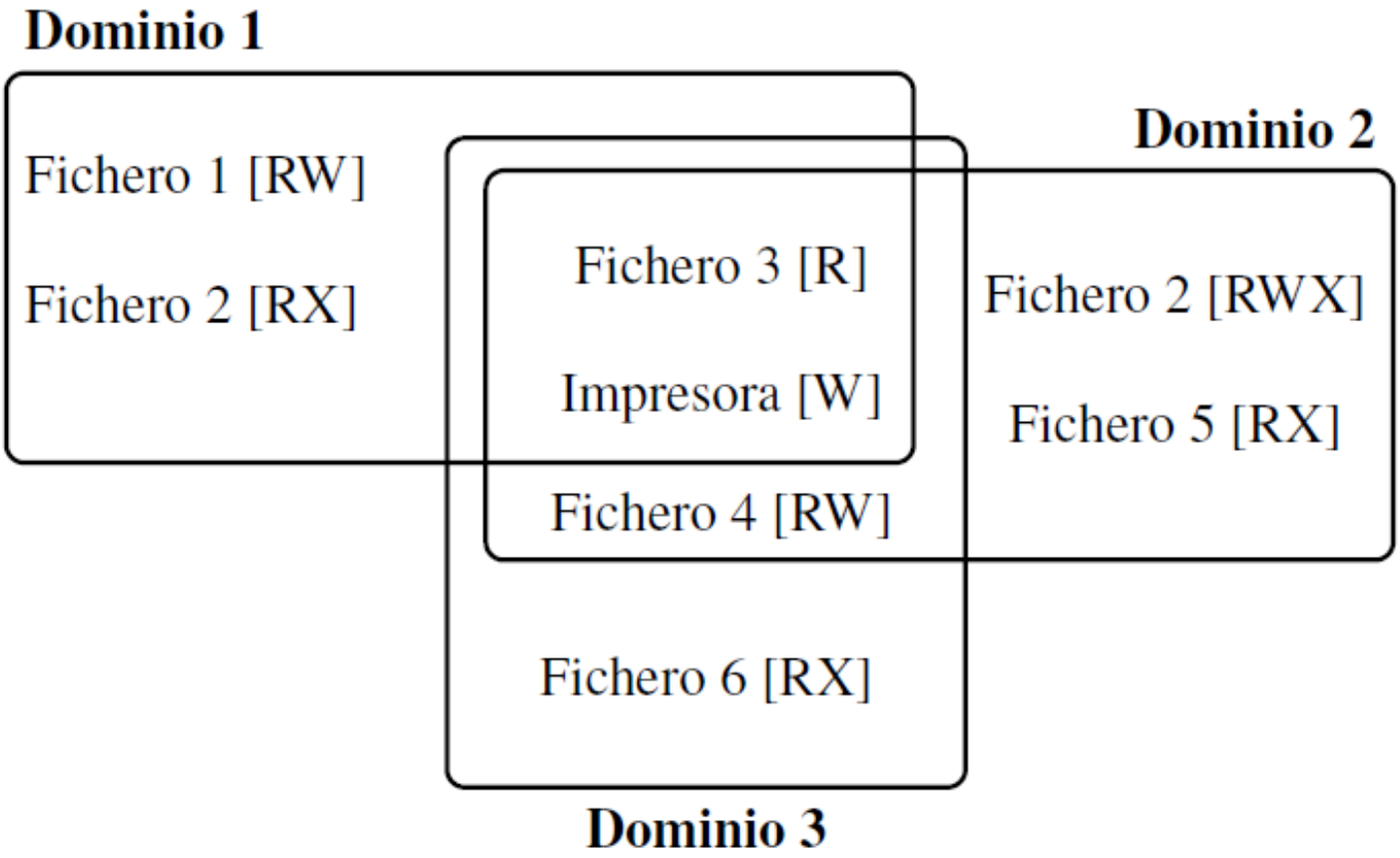
- Cada objeto tiene un nombre único (para referirse a él), y un conjunto de operaciones realizables sobre el mismo

- ¿Cómo especificamos qué le está permitido a cada proceso con cada objeto? Con un **dominio de protección**

- Un dominio de protección (o dominio) representa un conjunto de permisos sobre un conjunto de objetos:
  - Cjto. de parejas ⟨objeto, derechos⟩ ⇒ Derecho es el permiso para realizar cierta tarea

3.3.2 Dominios de protección

- **Ejemplo**:



- Los derechos son lectura (R), escritura (W) y ejecución (X)
- Cada proceso se ejecuta en alguno de los dominios de protección y durante su ejecución podría cambiar de dominio

3.3.3 Matriz de acceso o de protección

- ¿Cómo sabe el sistema qué objetos pertenecen a cada dominio?
- Podemos imaginar una gran **matriz** ⇒ **Filas** ≡ Dominios
  Columnas ≡ Objetos
- En cada casilla → **permisos**
- Sabiendo el dominio en el que se ejecuta el proceso el SO puede determinar los objetos a los que tiene acceso y para qué
- Si los procesos pueden cambiar de dominio → el dominio es un objeto con la operación **Entrar**
- Esta estructura se conoce como **matriz de acceso o de protección**

3.3.3 Matriz de acceso o de protección

- ¿Cómo sabe el sistema qué objetos pertenecen a cada dominio?

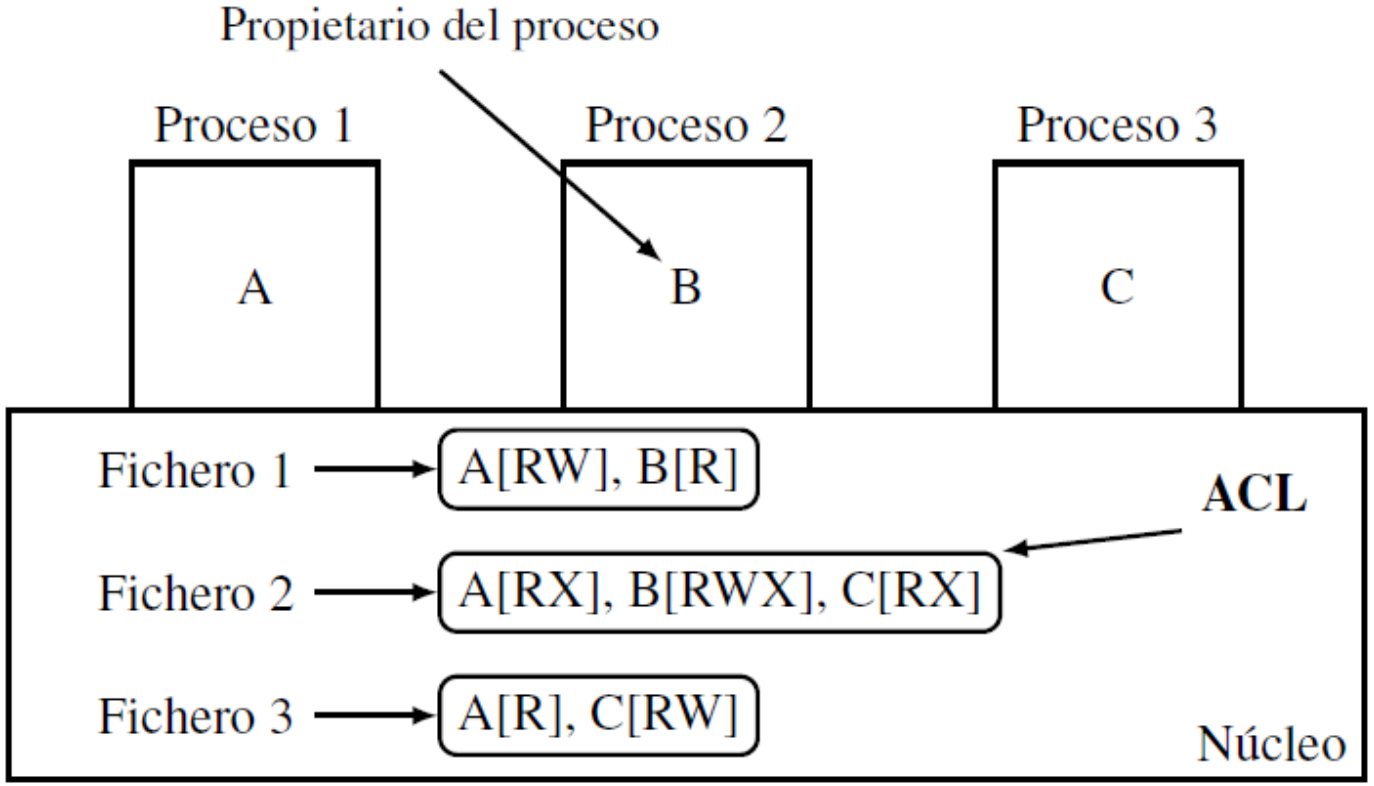
Dominios	Objetos									
	Fichero 1	Fichero 2	Fichero 3	Fichero 4	Fichero 5	Fichero 6	Impresora	Dominio 1	Dominio 2	Dominio 3
1	Leer Escribir	Leer Ejecutar	Leer				Escribir			Entrar
2		Leer Escribir Ejecutar	Leer	Leer Escribir	Leer Ejecutar		Escribir			
3			Leer	Leer Escribir		Leer Ejecutar	Escribir	Entrar		

- Dado que la mayoría de los dominios no tienen acceso a la mayoría de los objetos, no tiene sentido implementar tal cual la matriz de protección
  - La mayoría de las celdas estarían vacías

- Existen dos métodos prácticos que almacenan solo celdas no vacías: las listas de control de acceso (ACL) y las listas de posibilidades

3.3.4 Listas de control de acceso (ACL)

- Almacenan la matriz por columnas
- Por cada objeto → Lista con los dominios que pueden tener acceso a ese objeto y la forma de dicho acceso



- Las ACLs están protegidas por el núcleo del SO (los procesos de usuario no pueden manipularlas directamente)

3.3.4 Listas de control de acceso (ACL)

- Puede haber permisos adicionales aparte de RWX (copiar, borrar, ordenar…)
- Si el SO permite crear grupos de usuarios, también se aplican a grupos, pudiendo haber entradas con el grupo y sus permisos
  - Si en la ACL hay también una entrada para el usuario, el SO deberá decidir cuál se utiliza

- Con las ACL es muy sencillo revocar un permiso que se ha dado antes: se edita la ACL para reflejar el cambio.
  - **Problema**: Si la ACL se comprueba sólo cuando se abre un fichero (como sucede en UNIX), se podrá seguir accediendo como antes hasta que se cierre el fichero

3.3.5 Listas de posibilidades

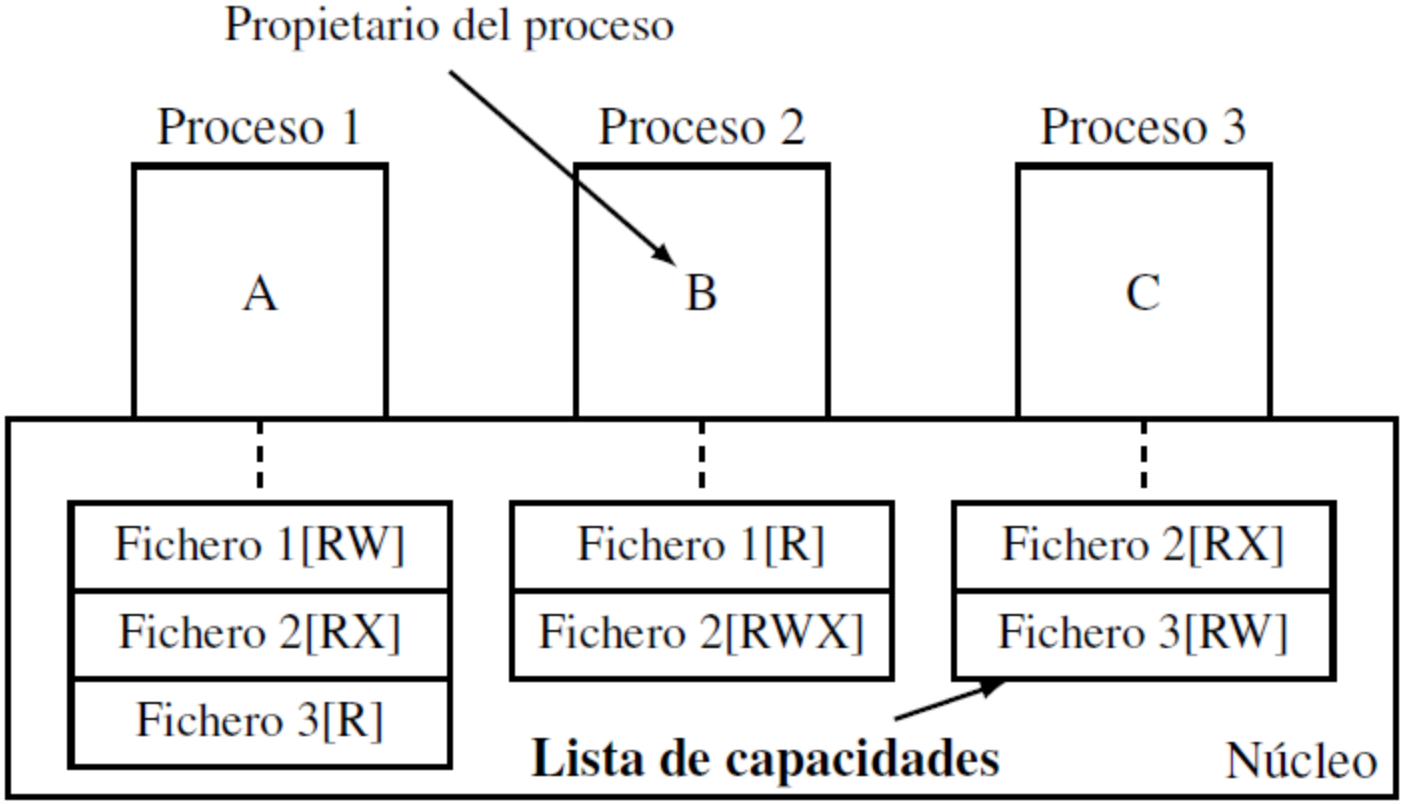
- Equivale a almacenar la matriz por filas: Para cada dominio hay un conjunto de objetos y operaciones permitidas sobre cada uno

- Suelen asociarse a procesos: A cada proceso se asocia una lista de objetos a los que puede tener acceso con una indicación de las operaciones permitidas, es decir, a cada proceso se le asocia su lista de posibilidades

- A cada elemento de la lista se le llama capacidad o posibilidad

3.3.5 Listas de posibilidades

- Cada capacidad otorga al propietario ciertos derechos sobre un objeto:contiene varios campos, ej. tipo de objeto, derechos, apuntador al objeto



- Aunque las listas se asocian a procesos, se mantiene en el núcleo por protección (un proceso no puede cambiar directamente su lista) → los procesos referencia a las posibilidades a través del número

3.3.5 Listas de posibilidades

- Un proceso quiere hacer operación sobre objeto:
  - Referencia según posición en la lista de capacidades, por ejemplo: "leer 1KiB del fichero al que apunta la capacidad 2"
  - Sucede en Unix cuando se hace una operación sobre un fichero a través de su descriptor de fichero (número de la posibilidad)
- Un proceso posee una capacidad: puede pasársela a otros procesos

- ¿Cómo se construye una posibilidad para un objeto dado que los procesos se están creando y destruyendo continuamente? La solución de UNIX es:
  - Combinar lista de posibilidades con las ACLs
  - Proceso accede primera vez a objeto (abre fichero con `open()`): el SO consulta la ACL:
    - Si no permiso → error
    - Si sí permiso → crea la posibilidad y se devuelve la posición en la lista (descriptor de fichero), que se usará para accesos
    - Al acabar, se destruye la posibilidad (e.j. con `close()`)

3.3.6 Comparación

- **Las ACL:**
  - Se corresponden directamente con las necesidades del usuario: cuando un usuario crea un objeto especifica los dominios y las operaciones permitidas
  - Es difícil determinar todos los objetos y derechos de un dominio
  - Suelen ser costosas de acceder: en cada acceso a un objeto hay que buscar en su ACL, que podría ser larga y/o estar almacenada en disco (ej. fichero)

3.3.6 Comparación

- **Las listas de capacidades:**
  - Son más complejas para el usuario pues no corresponden con sus necesidades de definición
  - Útiles para saber qué objetos puede ser accedidos por un proceso y con qué operaciones
  - ¿Dónde están guardadas cuando el ordenador se apaga?
    - La solución más común es tener asociada una ACL y consultarla en el momento en que se crea la capacidad

3.3.7 Cancelación

- **¿Qué pasa con la revocación de permisos?**
  - Es decir, ¿qué pasa si queremos quitar un derecho de acceso a un objeto?
  - La revocación de permisos de un objeto particular es costosa con las **listas de posibilidades**:
    - Dado que estos están distribuidas por el sistema (pueden estar almacenadas en las listas de posibilidades de muchos procesos), lleva tiempo determinar las posibilidades para un objeto y eliminarlas
    - Una forma de revocación general consiste en añadir un nivel de indirección: la posibilidad apunta a un objeto indirecto, que a su vez apunta al objeto real, revocamos el acceso sobre el objeto real *eliminando* el enlace entre el objeto indirecto y el objeto real
    - Pero sigue siendo difícil anular de forma selectiva (usuario a usuario)
  - Muy simple en la **ACL**: cambio la entrada de la ACL

Índice

3.4 Autenticación de usuarios

- Principal problema de seguridad para los SSOO
- Aunque existen varios métodos de autenticación lo más normal es utilizar **ID+contraseña**
  - El ID determina si se tiene acceso y a qué nivel (p.ej. root)
  - La contraseña autentica el ID
- En UNIX: cada contraseña se guarda cifrada (en `/etc/shadow`)
  - Se genera a partir de **(clave+base)** y el tipo de algoritmo de cifrado, llamando a la rutina de cifrado `crypt()`
  - La base ("salt"), se añade al comienzo de la clave encriptada en formato:
    - "**$**tipo**$**salt**$**password"
  - El "tipo" indica con un número si se usa **MD5**(1), **SHA256**(5) o **SHA512**(6).
  - La base es un valor relacionado con el momento en que se crea la contraseña:
    - Impide que contraseñas iguales se cifren igual

3.4 Autenticación de usuarios

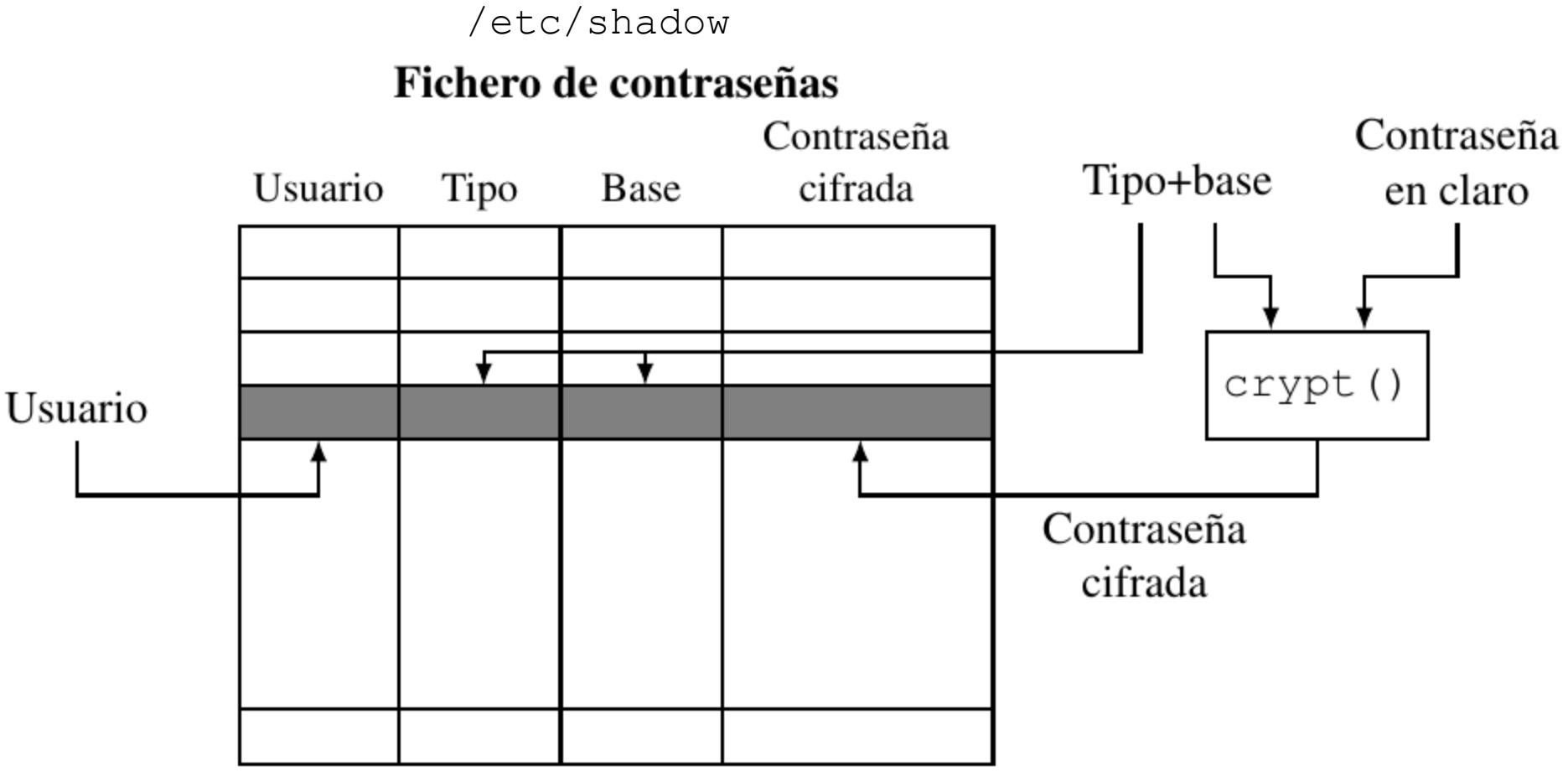
- Ejemplo de línea en `/etc/shadow`:

```
alumno:$6$o1n94O.a2xIBRHgT$PLeAm6eXFemW.wq3LQFWC2CO
l9.LchP8bOV4OBEjqXhVF8WeByg.bkqvjJ0F.Mz6PQ7VyImjem0De7a.0V
Wz1:18448:0:99999:7:::
```

- Ejemplo de línea en `/etc/passwd`:

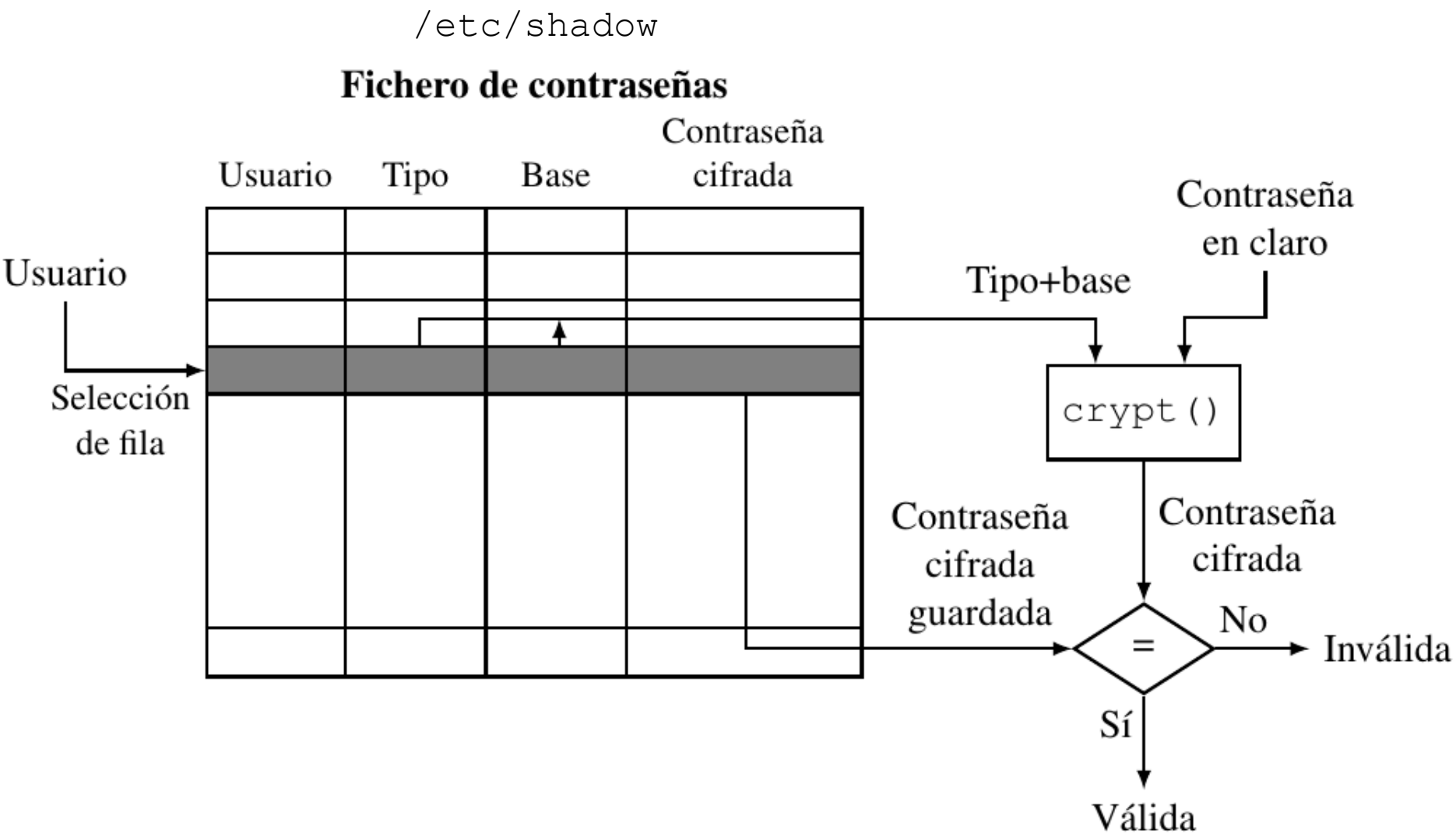
```
alumno:x:1000:1000:Alumno de ISO:/home/alumno:/bin/bash
```

3.4 Autenticación de usuarios



(a) Almacenamiento de una nueva contraseña para un usuario.

3.4 Autenticación de usuarios



3.4 Autenticación de usuarios

- Estrategias de elección de contraseñas
  - Las funciones *hash* empleadas para el cifrado no tienen función inversa y soportan ataques por adivinación de fuerza bruta
  - Aunque el problema real son los usuarios (longitud, palabras sencillas, etc.)
  - Técnicas para evitarlas:
    - Contraseñas largas, que no sean palabras de diccionario, introducir dígitos, letras en mayúsculas y minúsculas y caracteres especiales
    - Instrucción del usuario
      - Cambio frecuente de las contraseñas, no almacenarlas/anotarlas en lugares visibles, no comunicarlas a nadie, etc.
    - Inspección proactiva de contraseñas
      - El sistema permite al usuario elegir su propia contraseña siempre que esta cumpla ciertos requisitos que se comprueban al seleccionarla

Índice

3.5 Protección en UNIX

- Cada usuario tiene asignado un **Identificador de Usuario** (**UID**)
  - Aunque varios usuarios podrían tener el mismo número (UID)
- El «login» de un usuario lo asocia con su UID (a través de la entrada correspondiente en `/etc/passwd`)
- Además, los usuarios se organizan en torno a **Grupos de usuarios**, cada uno con su **GID** (de nuevo, un número):
  - Igualmente, varios grupos podrían tener el mismo GID
  - Además, un usuario podría pertenecer a varios grupos
- Así, **un par ⟨UID,GIDs⟩ forma un dominio en UNIX**
  - Esto es, define qué puede usar ese usuario y con qué permisos
- El UID 0 es especial (normalmente asociado al usuario «root») ya que tiene todos los privilegios:
  - Esto puede llevar a problemas de seguridad ⇒ Con la clave de root *¡¡se tiene acceso a todo el sistema!!*

3.5 Protección en UNIX

- En Linux, por defecto, cada fichero y dispositivo (mediante entradas de fichero especial en `/dev`) pertenece a un usuario y a un grupo
- Se implementan ACLs restringidas, ya que se especifica en cada fichero qué permisos tiene cada usuario sobre ese fichero
- Las ACLs están restringidas a tres «conjuntos de usuarios»:
  - Propietario: permisos para el usuario propietario del fichero
  - Grupo: permisos para los usuarios que pertenezcan al grupo al que pertenece el fichero
  - «Otros»: permisos para el resto de usuarios
- Cada uno de los grupos tiene asociados tres posibles permisos: lectura, escritura y ejecución
- Linux **también implementa ACLs completas** (permiten especificar con detalle qué usuarios y grupos pueden acceder a cada fichero) para aquellos SSFF que lo admitan. Simplemente tenemos que activarlas en las opciones de montaje

3.5 Protección en UNIX

- Por defecto, cada proceso se ejecuta con el dominio (UID,GID) del usuario que lo crea
- Excepcionalmente, los ejecutables pueden poseer en sus atributos los bits **SETUID** y **SETGID** para cambiar el dominio:
  - SETUID: Al ejecutarse toma el UID del usuario propietario del ejecutable
  - SETGID: Al ejecutarse toma el GID del grupo propietario del ejecutable
- Así, un proceso tiene dos parejas de identificadores: el usuario/grupo real ⟨UID,GID⟩ y el usuario/grupo efectivo ⟨EUID,EGID⟩
  - `getuid() / geteuid()`
  - `getgid() / getegid()`
  - Normalmente, ambos son iguales, salvo cuando están los bits SETUID y SETGID

3.5 Protección en UNIX

- SETUID/SETGID (continuamos…)
  - Por ejemplo, el usuario «pepito» del grupo «usuarios» ejecuta el siguiente programa:

    `-rwsr-xr-x  root root abr 20 12:00  passwd`
  - Al ejecutarlo, el proceso `passwd` tiene los siguientes usuarios
    - UID: pepito
    - GID: usuarios
    - EUID: root
    - EGID: usuarios
  - Tiene los permisos efectivos de «root», lo que le permite leer y escribir el contenido del fichero **`/etc/shadow`** donde se guardan las contraseñas cifradas

    `---------- root shadow 1880 abr 20 12:36 shadow`
  - ¿Podemos ver el contenido de **`/etc/shadow`** con, por ejemplo, la orden cat?

    `-rwxr-xr-x root root 50416 abr 20 12:00 cat`

3.5 Protección en UNIX

- UNIX por defecto implementa un sistema híbrido entre ACLs y listas de posibilidades:
  - Los ficheros tienen asociada una ACL restringida como se ha visto
  - Cuando un proceso quiere abrir un fichero, se comprueba la ACL para ver si tiene acceso
  - Si tiene acceso, se crea una entrada en la tabla de ficheros abiertos para ese proceso y se devuelve un índice denominado **descriptor del fichero**
  - La entrada **contiene los permisos** para ese proceso sobre ese fichero (leer, escribir, etc.)
  - Así, la información de permisos de la lista de ficheros abiertos es de hecho una **lista de capacidades del proceso**. Basta con pasar el descriptor como parámetro para que tengamos acceso al fichero
  - La **tabla de ficheros** abiertos la mantiene el SO, por lo que no puede ser manipulada por el proceso