SISTEMAS OPERATIVOS:

Lección 9:
Gestión de Memoria

Jesús Carretero Pérez
Alejandro Calderón Mateos
José Daniel García Sánchez
Francisco Javier García Blas
José Manuel Pérez Lobato
María Gregoria Casares Andrés

ADVERTENCIA

- Este material es un simple guión de la clase: no son los apuntes de la asignatura.

- El conocimiento exclusivo de este material no garantiza que el alumno pueda alcanzar los objetivos de la asignatura.

- Se recomienda que el alumno utilice los materiales complementarios propuestos.

Objetivos

- Conocer las funciones de la gestor de memoria.
- Conocer las fases en la generación de un ejecutable y la estructura del mapa de memoria de un proceso.
- Comprender los esquemas de asignación contigua de memoria.
- Ser capaz de usar los servicios de gestión de memoria para archivos proyectados y bibliotecas dinámicas

Contenido

- Funciones del gestor de memoria.
- Generación de ejecutables y bibliotecas dinámicas.
- Formato de fichero ejecutable.
- Servicios del gestor de memoria.

Objetivos del Gestor de Memoria

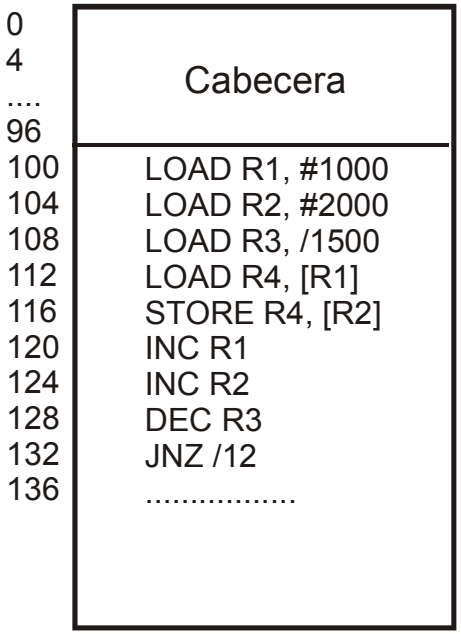
- S.O. multiplexa recursos entre procesos
  - Cada proceso cree que tiene una máquina para él solo.
  - Gestión de procesos: Reparto de procesador.
  - Gestión de memoria: Reparto de memoria.

- Objetivos:
  - Ofrecer a cada proceso un espacio lógico propio.
  - Proporcionar protección entre procesos.
  - Permitir que los procesos compartan memoria.
  - Dar soporte a las regiones del proceso.
  - Maximizar el grado de multiprogramación.
  - Proporcionar a los procesos mapas de memoria muy grandes.

Espacios lógicos independientes

- No se conoce posición de memoria donde un programa ejecutará.
- Código en ejecutable genera referencias entre 0 y N
- Ejemplo: Programa que copia un vector

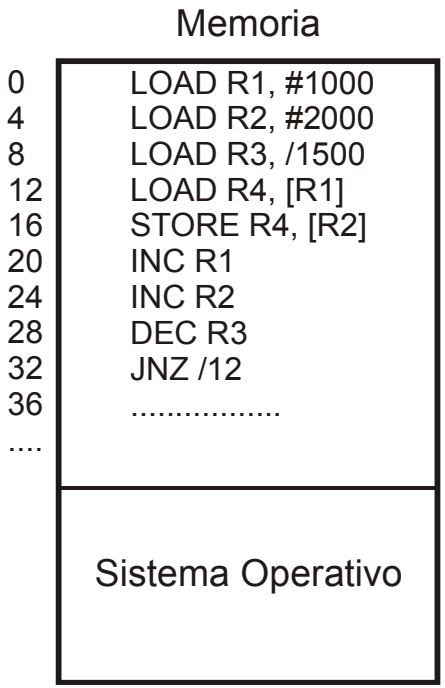
Fichero Ejecutable



- Vector destino a partir de dirección 2000
- Tamaño del vector en dirección 1500
- Vector origen a partir de dirección 1000

Sistemas Operativos monoprogramados

- S.O. en direcciones más altas
- Programa se carga en dirección 0
- Se le pasa el control al programa



Reubicación

- En un sistema operativo multiprogramado, no se pueden colocar todos los programas a partir de la misma dirección (p. ej. 0).
  - Necesidad de pode reubicar un programa a partir de una dirección de memoria.
  - Reubicación -> Traducción de direcciones lógicas en direcciones físicas.

- Dir. lógicas: direcciones de memoria generadas por el programa
- Dir. físicas: direcciones de memoria principal asignadas al proceso

- Función de traducción:

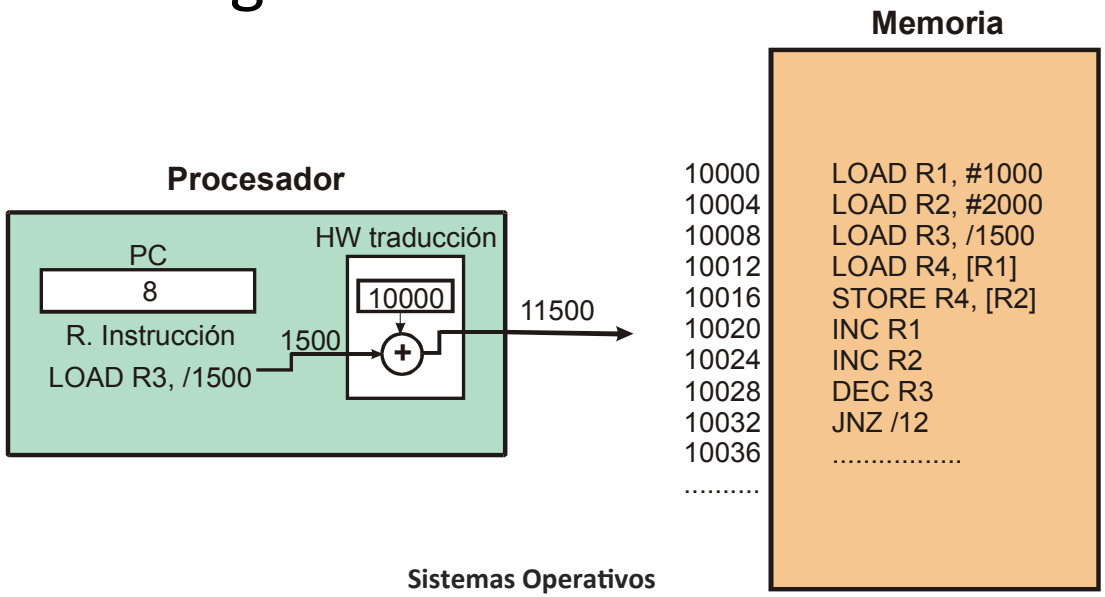
  Traducción(IdProc, dir_lógica) -> dir_física

Reubicación

- Reubicación crea espacio lógico independiente para proceso
  - El Sistema Operativo debe poder acceder a espacios lógicos de los procesos
- Ejemplo: Programa tiene asignada memoria a partir de 10000
  - Sumar 10000 a direcciones generadas
- Dos alternativas:
  - Reubicación hardware
  - Reubicación software

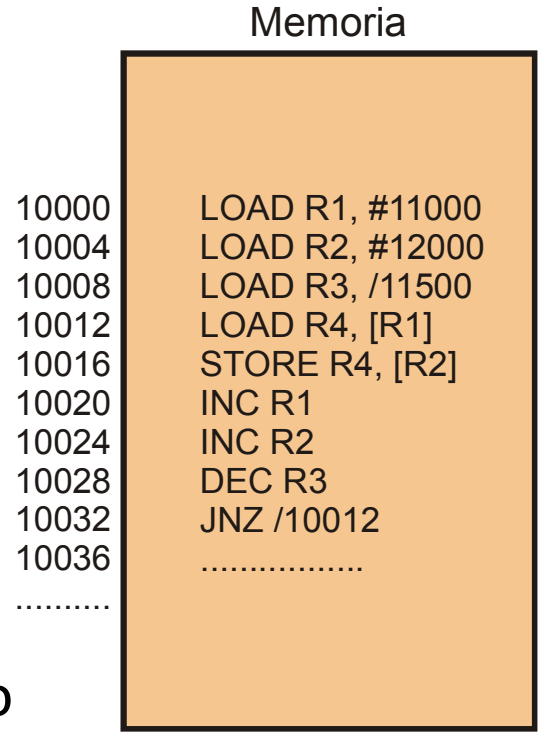
Reubicación hardware

- Hardware (MMU) encargado de traducción
- S.O. se encarga de:
  - Almacena por cada proceso su función de traducción
  - Especifica al hardware qué función aplicar para cada proceso
- Programa se carga en memoria sin modificar



Reubicación software

- Traducción de direcciones durante carga del programa
- Programa en memoria distinto del ejecutable
- Desventajas:
  - No asegura protección
  - No permite mover programa en tiempo de ejecución

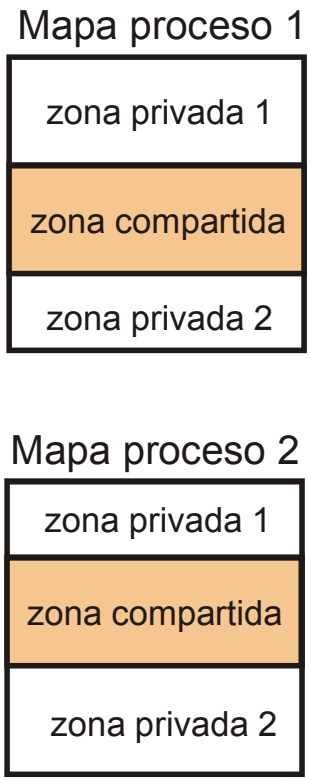


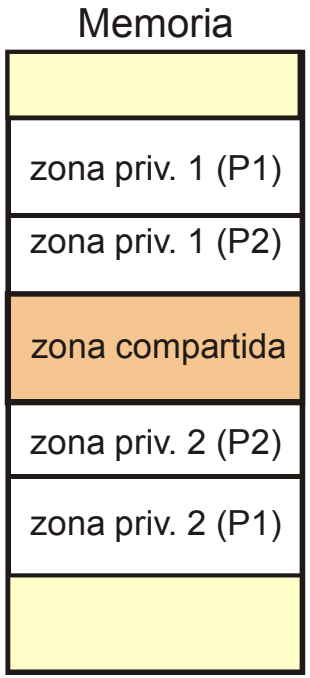
Protección

- **Monoprogramación**: Protección del SO
- **Multiprogramación**: Además procesos entre sí
- Traducción debe crear espacios disjuntos
- Necesario validar todas las direcciones que genera el programa
  - La detección debe realizarla el hardware del procesador
  - El tratamiento lo hace el SO
- En sistemas con mapa de E/S y memoria común:
  - Traducción permite impedir que procesos accedan directamente a dispositivos de E/S

Compartición de memoria

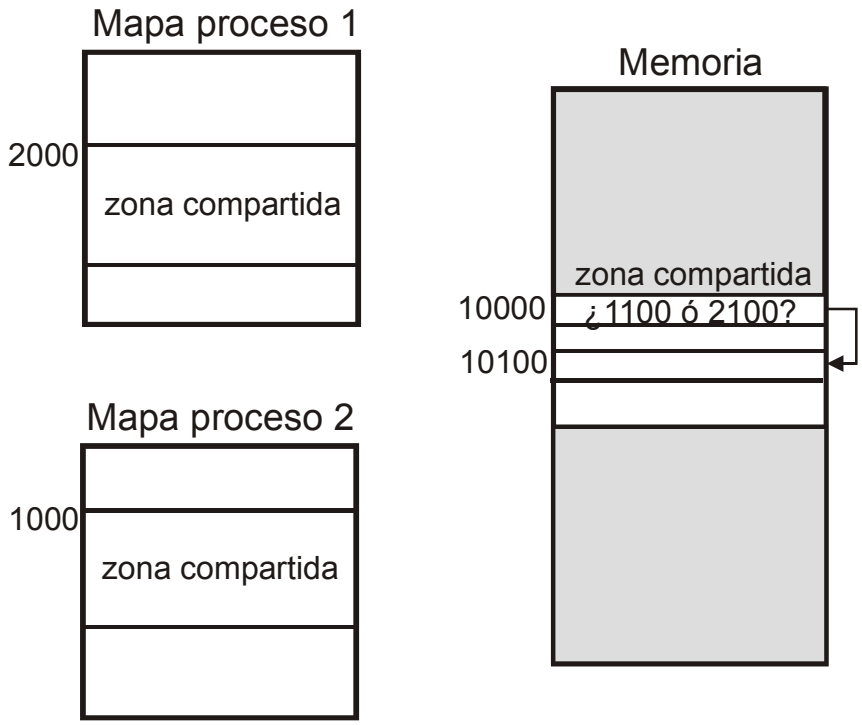
- Direcciones lógicas de 2 o más procesos se corresponden con misma dirección física
- Bajo control del S.O.
- Beneficios:
  - Procesos ejecutando mismo programa comparten su código
  - Mecanismo de comunicación entre procesos muy rápido
- Requiere asignación no contigua.





Problemas de la compartición de memoria

- Si posición de zona compartida contiene referencia a otra posición de la zona
- Ejemplo con zonas de código:
  - Zona compartida contiene instrucción de bifurcación a instrucción dentro de la zona
- Ejemplo con zonas de datos:
  - Zona contiene una lista con punteros



Soporte de regiones

- Mapa de proceso no homogéneo
  - Conjunto de regiones con distintas características
  - Ejemplo: Región de código no modificable
- Mapa de proceso dinámico
  - Regiones cambian de tamaño (p.ej. pila)
  - Se crean y destruyen regiones
  - Existen zonas sin asignar (huecos)
- Gestor de memoria debe dar soporte a estas características:
  - Detectar accesos no permitidos a una región
  - Detectar accesos a huecos
  - Evitar reservar espacio para huecos
- S.O. debe guardar una tabla de regiones para cada proceso

Maximización del rendimiento

- Reparto de memoria maximizando grado de multiprogramación
- Se "desperdicia" memoria debido a:
  - "Restos" inutilizables (fragmentación)
  - Tablas requeridas por gestor de memoria
- Menor fragmentación ➔ Tablas más grandes
- Compromiso: Paginación

- Uso de memoria virtual para aumentar grado de multiprogramación

Páginas de una dirección
- No hay fragmentación
- Irrealizable por tamaño de TP

Memoria

0	Dirección 50 del proceso 4
1	Dirección 10 del proceso 6
2	Dirección 95 del proceso 7
3	Dirección 56 del proceso 8
4	Dirección 0 del proceso 12
5	Dirección 5 del proceso 20
6	Dirección 0 del proceso 1
	....
N-1	Dirección 88 del proceso 9
N	Dirección 51 del proceso 4

Mapas de memoria muy grandes

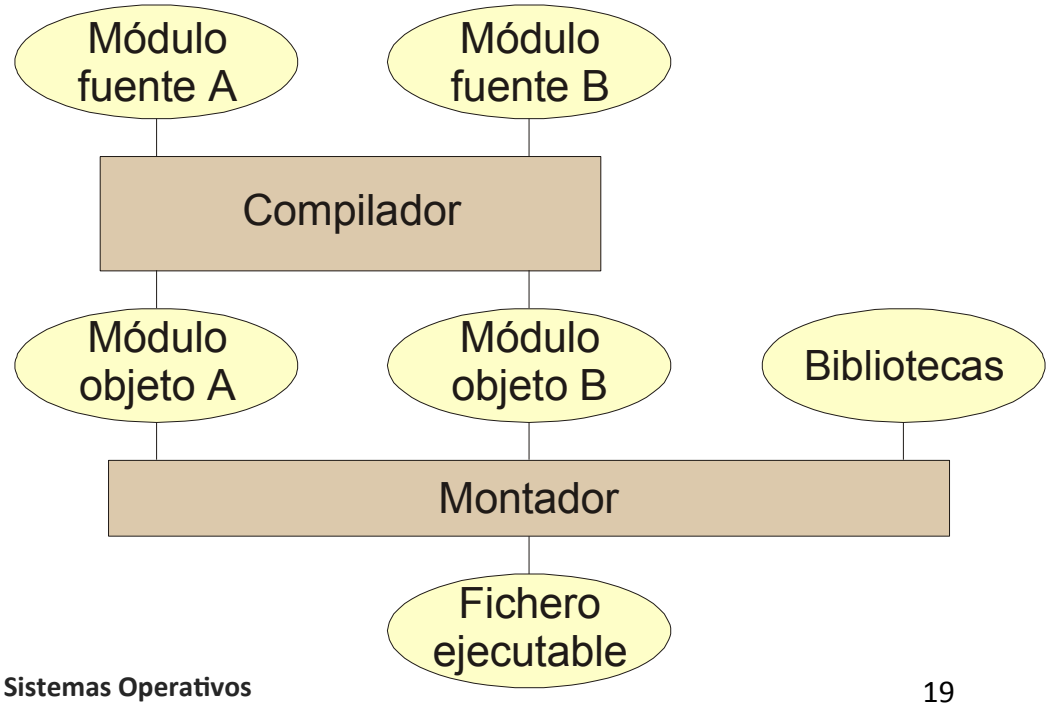
- Procesos necesitan cada vez mapas más grandes
  - Aplicaciones más avanzadas o novedosas
- Resuelto gracias al uso de memoria virtual
- Históricamente se han usado overlays:
  - Programa dividido en fases que se ejecutan sucesivamente
  - En cada momento sólo hay una fase residente en memoria
  - Cada fase realiza su labor y carga la siguiente
  - No es transparente: Toda la labor realizada por programador

Contenido

- Funciones del gestor de memoria.
- Generación de ejecutables y bibliotecas dinámicas.
- Formato de fichero ejecutable.
- Servicios del gestor de memoria.

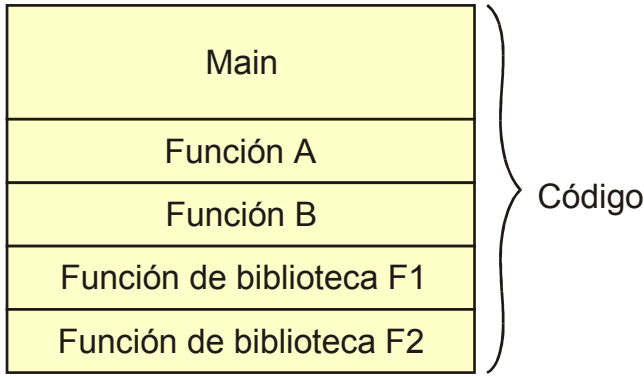
Fases de generación de un ejecutable

- Aplicación: conjunto de módulos en lenguaje de alto nivel
- Procesado en dos fases: Compilación y Montaje
- Compilación:
  - Resuelve referencias dentro cada módulo fuente
  - Genera módulo objeto
- Montaje (o enlace):
  - Resuelve referencias entre módulos objeto
  - Resuelve referencias a símbolos de bibliotecas
  - Genera ejecutable incluyendo bibliotecas



Bibliotecas de objetos

- Biblioteca: colección de módulos objeto relacionados
- Bibliotecas del sistema o creadas por el usuario
- Bibliotecas Estáticas:
  - Montaje: enlaza los módulos objeto del programa y de las bibliotecas
  - Ejecutable autocontenido
- Desventajas del montaje estático:
  - Ejecutables grandes
  - Código de función de biblioteca repetido en muchos ejecutables
  - Múltiples copias en memoria del código de función de biblioteca
  - Actualización de biblioteca implica volver a montar



Bibliotecas dinámicas

- Carga y montaje de biblioteca en tiempo de ejecución
- Ejecutable contiene:
  - Nombre de la biblioteca
  - Rutina de carga y montaje en tiempo de ejecución
- En 1ª referencia a símbolo de biblioteca en tiempo de ejecución:
  - Rutina carga y monta biblioteca correspondiente
  - Ajusta instrucción que realiza referencia para que próximas referencias accedan a símbolo de biblioteca
    - Problema: Se modificaría el código del programa
    - Solución típica: Referencia indirecta mediante una tabla

Bibliotecas dinámicas: Ventajas e inconvenientes

- Ventajas:
  - Menor tamaño ejecutables
  - Código de rutinas de biblioteca sólo en archivo de biblioteca
  - Procesos pueden compartir código de biblioteca
  - Actualización automática de bibliotecas: Uso de versiones
- Inconvenientes:
  - Mayor tiempo de ejecución debido a carga y montaje
    - Tolerable: Compensa el resto de las ventajas
  - Ejecutable no autocontenido
- Uso de bibliotecas dinámicas es transparente
  - Mandatos de compilación y montaje igual que con estáticas

Uso de bibliotecas dinámicas

- Forma de uso habitual:
  - Se especifica en tiempo de montaje qué biblioteca usar pero se pospone su carga y montaje a tiempo de ejecución
- Uso explícito:
  - Requerido por aplicaciones que determinan en tiempo de ejecución qué bibliotecas deben usar
  - No se especifica biblioteca en mandato de montaje
  - Programa solicita carga de bib. mediante servicio del sistema
    - dlopen en UNIX y LoadLibrary en Win32
  - Acceso "no" transparente a símbolos de la biblioteca
    - Programa necesita usar servicios del sistema para ello
    - dlsym en UNIX y GetProcAddress en Win32

Compartición de bibliotecas dinámicas

- Biblioteca dinámica contiene referencias internas
  - Problema de zona compartida con autoreferencias
- Tres posibles soluciones:
  1. A cada bib. dinámica se le asigna rango de direcciones fijo
     - Inconveniente: Poco flexible
  2. En montaje en t. de ejecución se reajustan autoreferencias
     - Inconveniente: Impide compartir código de biblioteca
  3. Crear biblioteca con código independiente de posición(PIC)
     - Se genera código con direccionamiento relativo a registro
     - Inconveniente (tolerable): dir. relativo menos eficiente

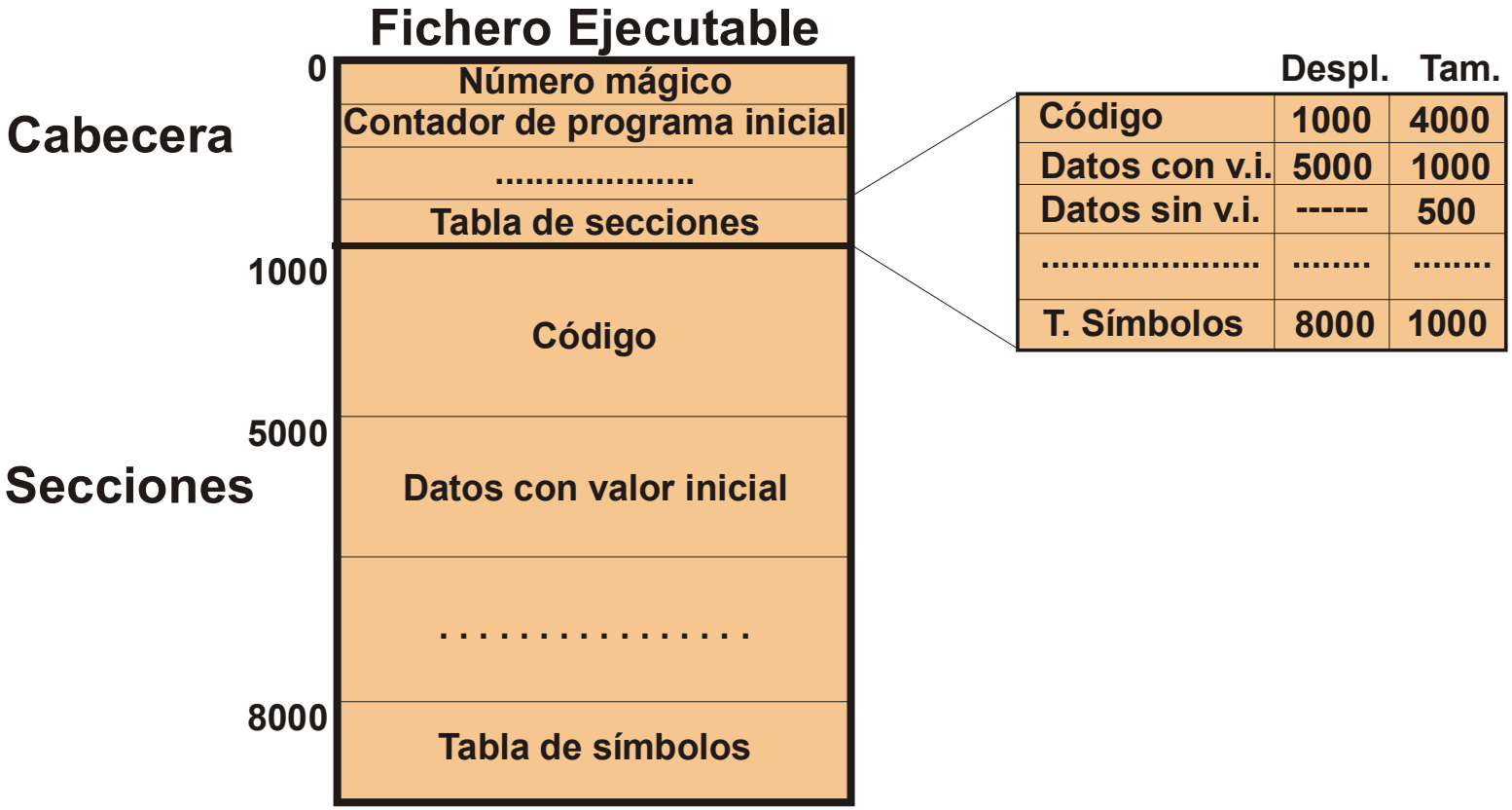
Contenido

- Funciones del gestor de memoria.
- Generación de ejecutables y bibliotecas dinámicas.
- Formato de fichero ejecutable.
- Servicios del gestor de memoria.

Formato de ejecutable

- Distintos fabricantes usan diferentes formatos
  - Ejemplo: En Linux Executable and Linkable Format (ELF)
- Estructura: Cabecera y conjunto de secciones
- Cabecera:
  - Número mágico que identifica a ejecutable
  - Punto de entrada del programa
  - Tabla de secciones

Formato de ejecutable



Secciones de un ejecutable

- Variedad de tipos de secciones. Ejemplo:
  - Tabla de símbolos para depuración
  - Lista de bibliotecas dinámicas usadas
- Más relevantes en mapa de memoria del proceso:
  - Código, Datos con valor inicial y Datos sin valor inicial
- Código (texto)
  - Contiene código del programa
- Datos con valor inicial
  - Variables globales inicializadas
- Datos sin valor inicial
  - Variables globales no inicializadas
  - Aparece en tabla de secciones pero no se almacena en ejecutable
- ¿Por qué no hay sección vinculada a variables locales?
  - → Pila o registros

Variables globales y locales

- Variables globales
  - Estáticas
  - Se crean al iniciarse programa
  - Existen durante ejecución del mismo
  - Dirección fija en memoria y en ejecutable
- Variables locales y parámetros
  - Dinámicas
  - Se crean al invocar función
  - Se destruyen al retornar
  - La dirección se calcula en tiempo de ejecución
  - Recursividad: varias instancias de una variable

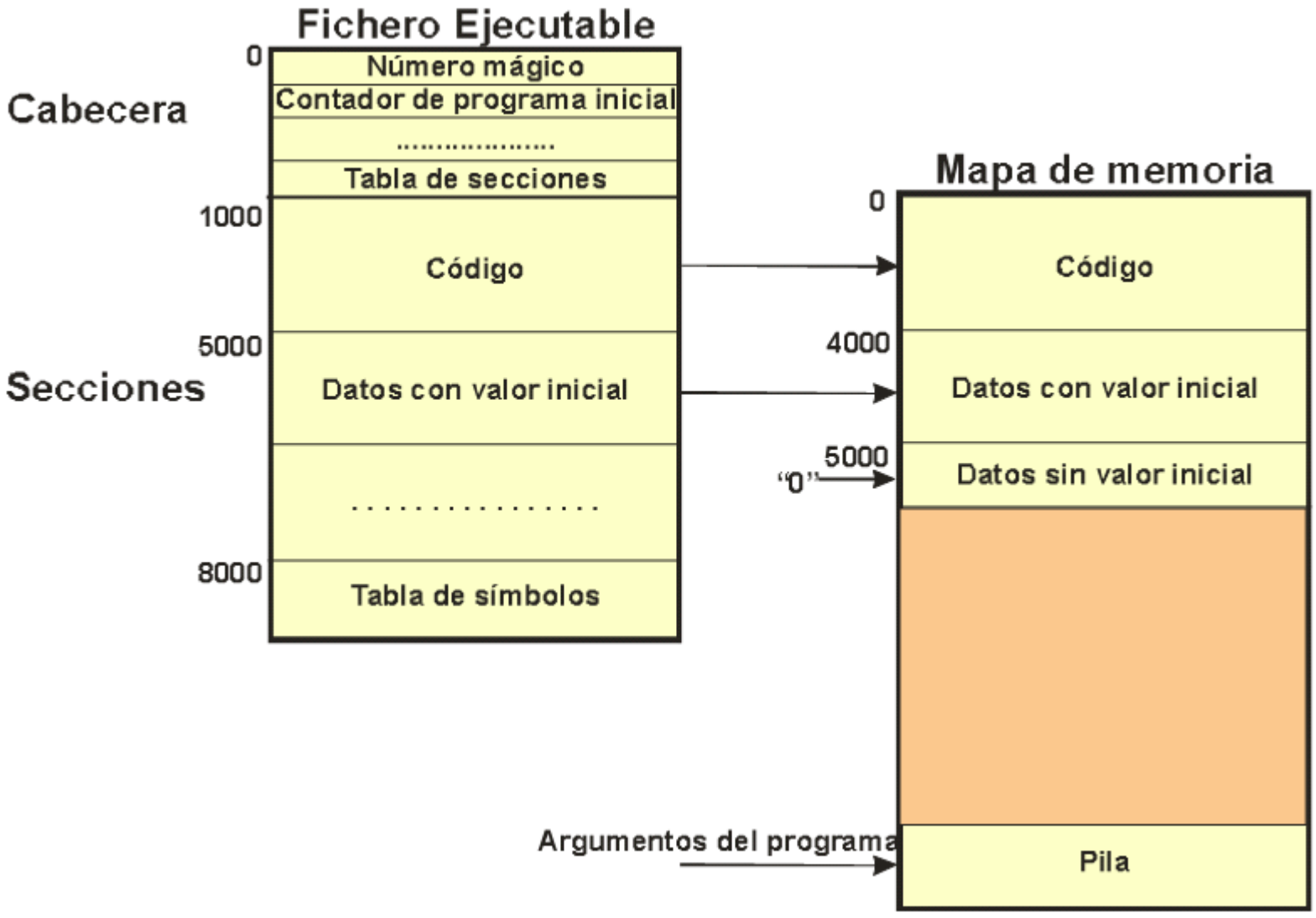
Mapa de memoria de un proceso

- Mapa de memoria o imagen del proceso: conjunto de regiones
- Región:
  - Tiene asociada una información (un "objeto de memoria")
  - Zona contigua tratada como unidad al proteger o compartir
- Cada región se caracteriza por:
  - dirección de comienzo y tamaño inicial
  - soporte: donde se almacena su contenido inicial
  - protección: RWX
  - uso compartido o privado
  - tamaño fijo o variable

Creación de mapa de memoria inicial a partir de ejecutable

- Ejecución de un programa: Crea mapa a partir de ejecutable
  - Regiones de mapa inicial → Secciones de ejecutable
- Código
  - Compartida, RX, T. Fijo, Soporte en Ejecutable
- Datos con valor inicial
  - Privada, RW, T. Fijo, Soporte en Ejecutable
- Datos sin valor inicial
  - Privada, RW, T. Fijo, Sin Soporte (rellenar 0)
- Pila
  - Privada, RW, T. Variable, Sin Soporte (rellenar 0)
  - Crece hacia direcciones más bajas
  - Pila inicial: argumentos del programa

Creación de mapa de memoria inicial a partir de ejecutable



Otras regiones

- Durante ejecución de proceso se crean nuevas regiones
  - Mapa de memoria tiene un carácter dinámico
- Región de Heap
  - Soporte de memoria dinámica (malloc en C)
  - Privada, RW, T. Variable, Sin Soporte (rellenar 0)
  - Crece hacia direcciones más altas
- Archivo proyectado
  - Región asociada al archivo proyectado
  - T. Variable, Soporte en Archivo
  - Protección y carácter compartido o privado especificado en proyección

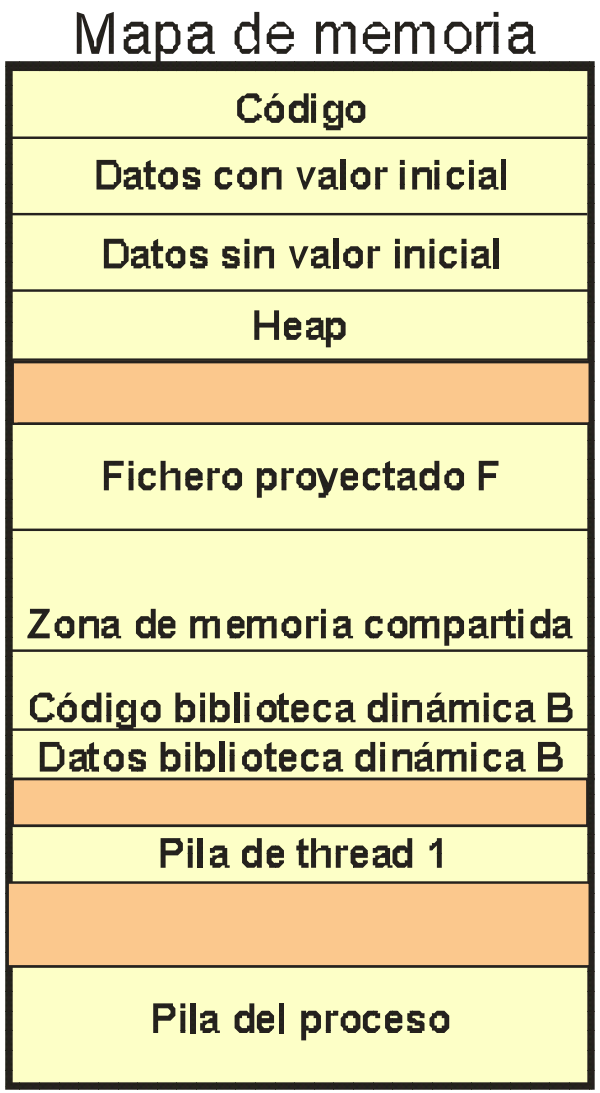
Otras regiones

- Memoria compartida
  - Región asociada a la zona de memoria compartida
  - Compartida, T. Variable, Sin Soporte (rellenar 0)
  - Protección especificada en proyección
- Pilas de threads
  - Cada pila de thread corresponde con una región
  - Mismas características que pila del proceso
- Carga de biblioteca dinámica
  - Se crean regiones asociadas al código y datos de la biblioteca

Características de regiones

Región	Soporte	Protección	Comp/Priv	Tamaño
Código	Fichero	RX	Compartida	Fijo
Dat. con v.i.	Fichero	RW	Privada	Fijo
Dat. sin v.i.	Sin soporte	RW	Privada	Fijo
Pilas	Sin soporte	RW	Privada	Variable
Heap	Sin soporte	RW	Privada	Variable
F. Proyect.	Fichero	por usuario	Comp./Priv.	Variable
M. Comp.	Sin soporte	por usuario	Compartida	Variable

Posible mapa de memoria de un proceso



Operaciones sobre regiones

- Para estudiar evolución del mapa de memoria se pueden distinguir las siguientes operaciones:
  - Crear región
    - Implícitamente al crear mapa inicial o por solicitud del programa en t. de ejecución (p.ej. proyectar un archivo)
  - Eliminar región
    - Implícitamente al terminar el proceso o por solicitud del programa en t. de ejecución (p.ej. desproyectar un archivo)
  - Cambiar tamaño de la región
    - Implícitamente para la pila o por solicitud del programa para el heap
  - Duplicar región
    - Operación requerida por el servicio FORK de POSIX

Archivos proyectados en memoria

- Generalización de memoria virtual.
  - Entradas de TP referencian a un archivo de usuario.
- Programa solicita proyección de archivo (o parte) en su mapa.
  - Puede especificar protección y si privada o compartida.
- S.O. rellena entradas correspondientes con:
  - No residente, Cargar de Archivo.
  - Privada/Compartida y Protección según especifica la llamada.
- Cuando programa accede a posición de memoria asociada a archivo proyectado, está accediendo al archivo.

Contenido

- Funciones del gestor de memoria.
- Generación de ejecutables y bibliotecas dinámicas.
- Formato de fichero ejecutable.
- Servicios del gestor de memoria.

Servicios de gestión de memoria

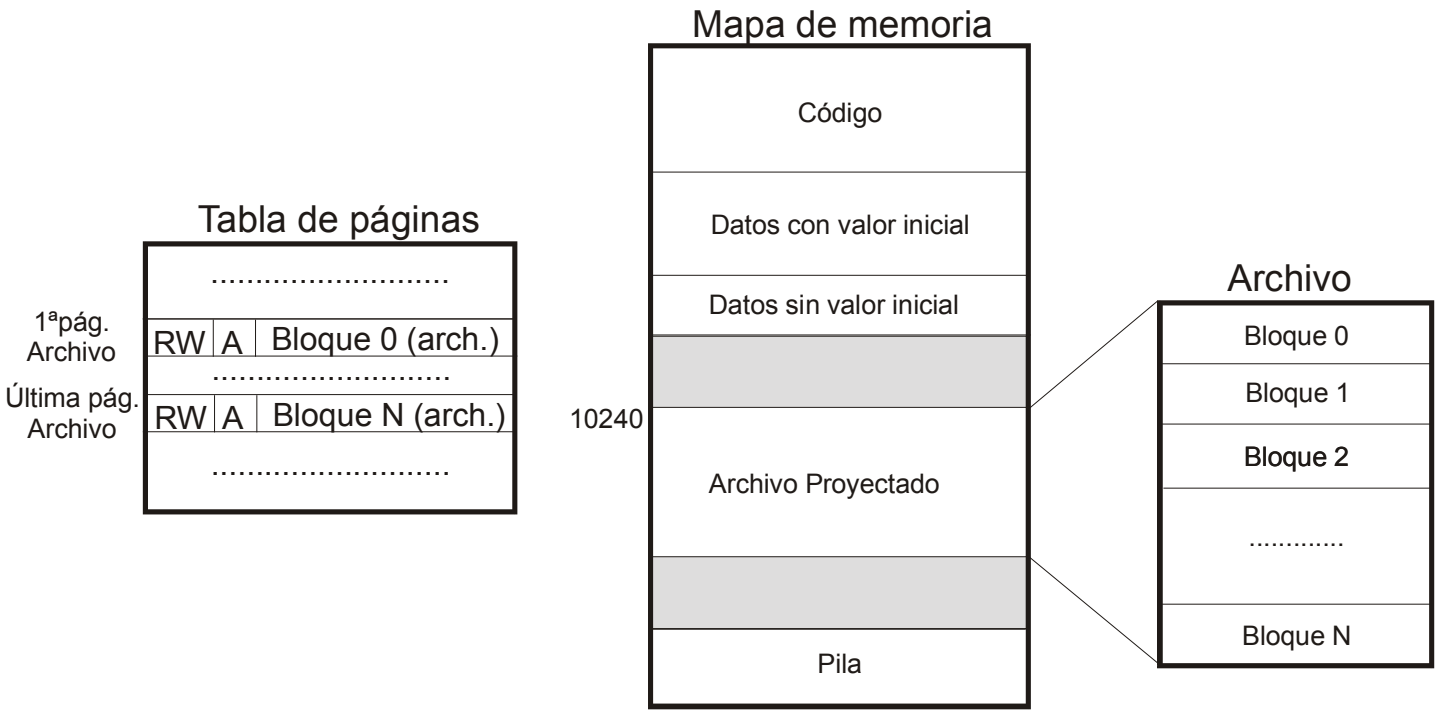
- Gestor de memoria realiza funciones internas.
- Ofrece pocos servicios directos a aplicaciones.

- Servicios:
  - POSIX.
    - Gestión de la proyección de archivos: mmap, munmap.
    - Gestión de bibliotecas dinámicas: dlopen, dlsym, dlclose.
  - Win32
    - Gestión de la proyección de archivos: CreateFileMapping, MapViewOfFile, UnmapViewOfFile.
    - Gestión de bibliotecas dinámicas: LoadLibrary, GetProcAddress, FreeLibrary.

Archivos proyectados en memoria

- Forma alternativa de acceso a archivos frente a read/write
  - Menos llamadas al sistema
  - Se evitan copias intermedias en cache del sistema de archivos
  - Se facilita programación ya que una vez proyectado se accede a archivo como a estructuras de datos en memoria
- Se usa para carga de bibliotecas dinámicas
  - La zona de código se proyecta como compartida
  - La zona de datos con valor inicial se proyecta como privada

Proyección en memoria



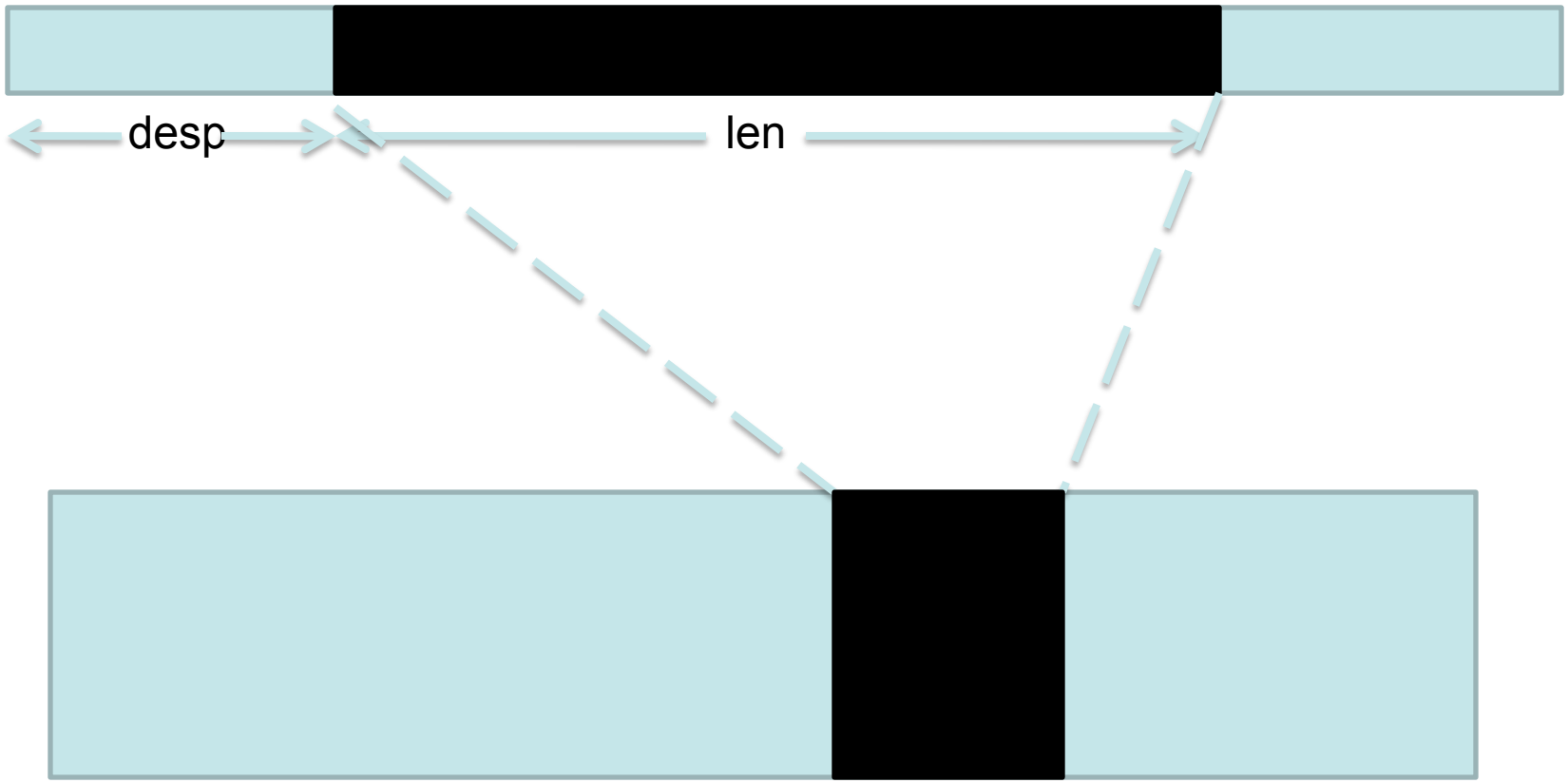
Proyección en POSIX

- `void *mmap(void *direc, size_t lon, int prot, int flags, int fd, off_t desp);`
- Establece proyección entre espacio de direcciones de un proceso y un archivo.
  - Devuelve la dirección de memoria donde se ha proyectado el archivo.
  - direc: dirección donde proyectar. Si NULL SO elige una.
  - lon: especifica el número de bytes a proyectar
  - prot: Protección para la zona (se pueden combinar con |).
  - flags: Propiedades de la región.
  - fd: Descriptor del fichero que se desea proyectar en memoria.
  - desp: Desplazamiento inicial sobre el archivo.

Proyección POSIX: mmap

- Tipos de protección:
  - PROT_READ: Se puede leer.
  - PROT_WRITE: Se puede escribir.
  - PROT_EXEC: Se puede ejecutar.
  - PROT_NONE: No se puede acceder a los datos.
- Propiedades de una región de memoria:
  - MAP_SHARED: La región es compartida. Las modificaciones afectan al fichero. Los procesos hijos comparten la región.
  - MAP_PRIVATE: La región es privada. El fichero no se modifica. Los procesos hijos obtienen duplicados no compartidos.
  - MAP_FIXED: El fichero debe proyectarse en la dirección especificada por la llamada.

Proyección POSIX



Desproyección en POSIX

- **`void munmap(void *direc, size_t lon);`**
  - Desproyecta parte del espacio de direcciones de un proceso desde la dirección direc hasta direc+lon.

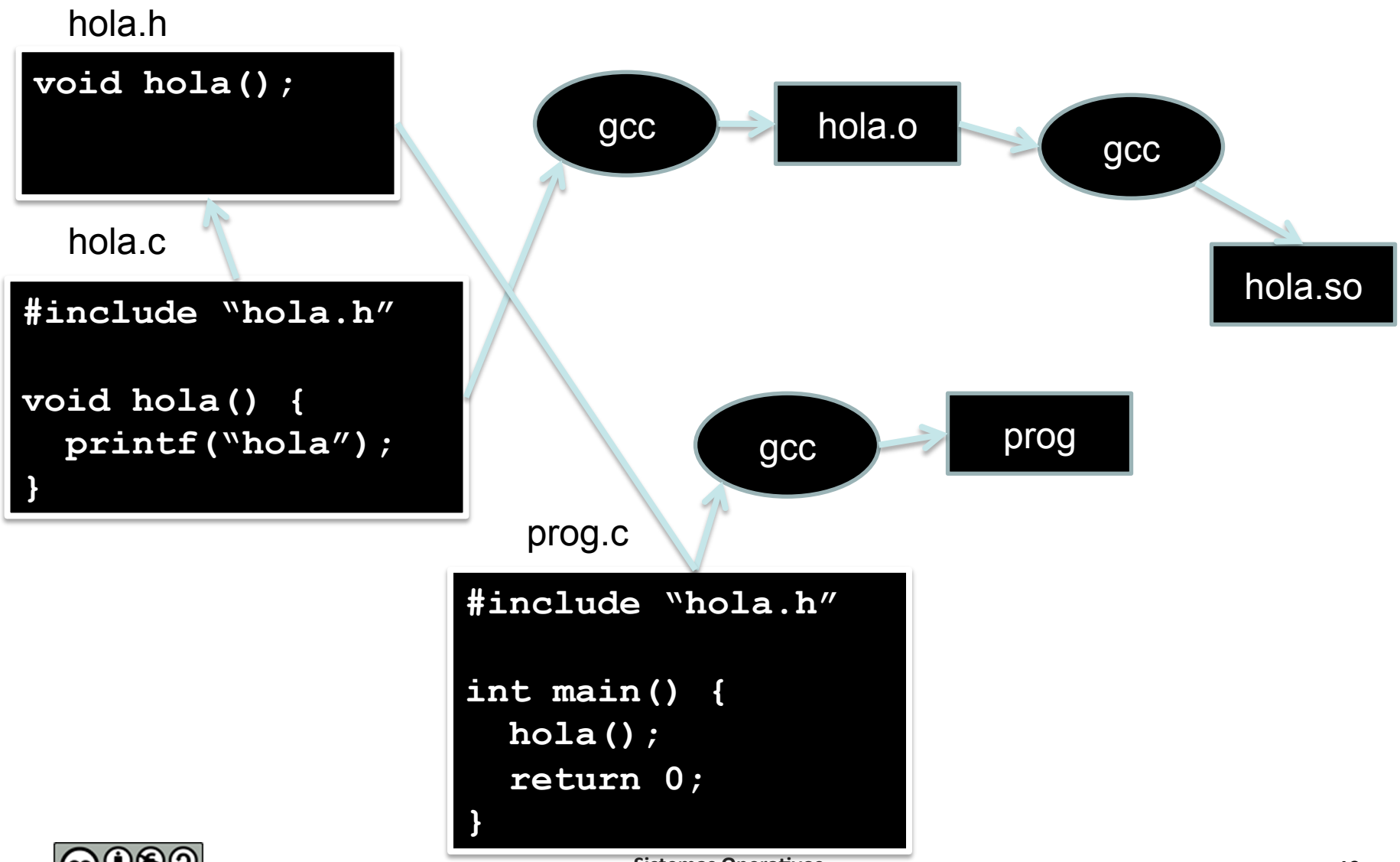
Ejemplo: Contar el número de blancos en un fichero

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main() {
  int fd;
  struct stat dstat;
  int i, n;
  char c,
  char * vec;

  fd = open("datos.txt",O_RDONLY);
  fstat(fd, &dstat);
  vec = mmap(NULL, dstat.st_size, PROT_READ, MAP_SHARED, fd, 0);
  close(fd);
  c =vec;
  for (i=0;i<dstat.st_size;i++) {
    if (*c==' ') {
      n++;
    }
    c++;
  }
  munmap(vec, dstat.st_size);
  printf("n=%d,\n",n);
  return 0;
}
```

Ejemplo: Copia de un fichero

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main() {
  int i, fd1, fd2;
  struct stat dstat;
  char * vec1, *vec2, *p, *q;

  fd1 = open("f1", O_RDONLY);
  fd2 = open("f2", O_CREAT|O_TRUNC|O_RDWR,0640);
  fstat(fd1,&dstat);
  ftruncate(fd2, dstat.st_size);

  vec1=mmap(0, dstat.st_size,
    PROT_READ, MAP_SHARED, fd1,0);
  vec2=mmap(0, dstat.st_size,
    PROT_WRITE, MAP_SHARED, fd2,0);

  close(fd1); close(fd2);

  p=vec1; q=vec2;
  for (i=0;i<dstat.st_size;i++) {
    *q++ = *p++;
  }

  munmap(vec1, dstat.st_size);
  munmap(vec2, dstat.st_size);

  return 0;
}
```

Biblioteca dinámica: generación



Bibliotecas dinámicas

- Normalmente el enlace implícito con bibliotecas dinámicas es suficiente.

- Enlace explícito:
  - Hay que escribir el código para cargar y enlazar los símbolos de la biblioteca dinámica.
  - Ejemplo de utilidad:
    - Decidir en tiempo de ejecución entre dos bibliotecas dinámicas que implementan una misma API.

Bibliotecas dinámicas: servicios POSIX

- **`void * dlopen(const char * bib, int flags);`**
  - Carga una biblioteca dinámica y la enlaza con el proceso actual.
  - Devuelve un descriptor que puede usarse posteriormente con dlsym y dlclose.
  - bib: Nombre de la biblioteca.
  - flags: Opciones.
    - RTLD_LAZY: Resolución diferida de referencias.
    - RTLD_NOW: Resolución inmediata de referencias.

Bibliotecas dinámicas: Servicios POSIX

- void * dlsym(void * ptrbib, char * simb);
  - Devuelve un puntero a un símbolo de la biblioteca dinámica.
    - ptrbib: Es el descriptor de biblioteca obtenido mediante dlopen.
    - simb: Cadena con el nombre del símbolo a cargar.
- void dlclose(void * ptrbib);
  - Descarga la biblioteca dinámica del proceso.

Carga explícita de biblioteca dinámica

```
#include <stdio.h>
typedef void (*pfn)(void);

int main() {
  void * bib;
  pfn func;
  bib = dlopen("libhola.so", RTLD_LAZY);
  func=dlsym(bib,"hola");
  (*func)();
  dlclose(bib);
  return 0;
}
```

SISTEMAS OPERATIVOS:

Lección 9:
Gestión de Memoria