# Bloque I: Principios de sistemas operativos





## Notas:

**Tema 7. Gestión de memoria**

- Mecanismos de gestión de la memoria
- Mecanismos de memoria virtual
- Esquemas de gestión de memoria en aplicaciones de tiempo real
- Mecanismos de memoria compartida
- Ejemplo de memoria compartida

# 1. Mecanismos de gestión de la memoria

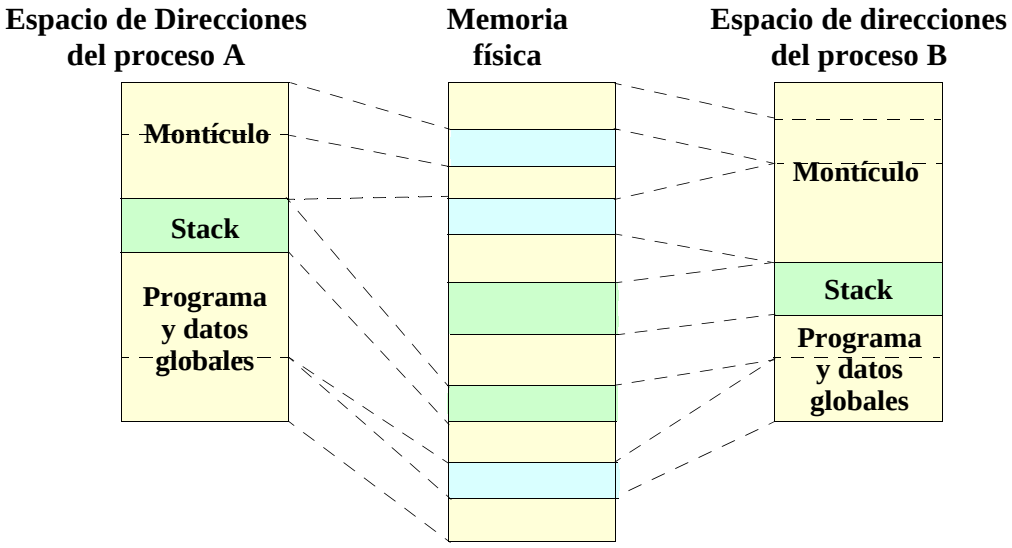
## Objetivos de la gestión de memoria:

- Dar a cada proceso un espacio de direcciones propio
- Protección entre procesos
- Proporcionar mapas de memoria grandes, independientes de la memoria física
- Maximizar el rendimiento
- Permitir que los procesos compartan memoria

## El principal mecanismo es la traducción o "mapeado" de posiciones de memoria lógicas a posiciones de memoria físicas



# Mapeado de memoria



# 2. Memoria virtual

**Permite mapear parte de las páginas de memoria en memoria secundaria**

- **para cada acceso a memoria:**
  - **si el dato está en memoria física se accede directamente**
  - **si no, se carga una página de memoria del disco, pasando en su lugar otra página al disco, si es necesario**

**Esto permite ofrecer un espacio de direcciones grande**

**La memoria lógica es independiente de la física**

**Con una buena gestión se maximiza el rendimiento**

- **política de asignación: FIFO, LRU (last recently used),...**



# Algunos conceptos básicos

**Página de memoria:**

- **granularidad de las zonas de memoria que se pueden mapear:**
  - **el tamaño de una zona mapeada es un número entero de veces el tamaño de la página**
  - **la dirección de comienzo de un objeto de memoria suele necesitar estar alineada con una página**

**Mapear en memoria:**

- **crear una asociación entre una zona del espacio de direcciones de un proceso y una zona de memoria física o de un objeto de memoria**

# Conceptos Básicos (cont.)

## Objeto de memoria compartida

- un objeto que representa memoria y que se puede mapear concurrentemente en el espacio de direcciones de varios procesos



# 3. Gestión de memoria en aplicaciones de tiempo real

## Alojar memoria dinámica

```
#include <stdlib.h>
void * malloc(int size)
```

- obtiene una nueva zona de memoria de tamaño `size`
- retorna un puntero a esa nueva zona de memoria

## Liberar memoria alojada dinámicamente

```
void free (void *addr);
```

- libera la zona de memoria que comienza en `addr`

## En la mayoría de los sistemas operativos estas operaciones tienen un tiempo de respuesta impredecible

- en sist. de tiempo real: usarlas sólo durante la inicialización

# Bloquear la memoria virtual

**Para evitar la impredecibilidad del tiempo de respuesta en sistemas de memoria virtual, hacer residentes todas las páginas del proceso:**

```
#include <sys/mman.h>
int mlockall (int flags);
```

- `flags` indica las opciones:
  - `MCL_CURRENT`: afecta a las páginas actuales
  - `MCL_FUTURE`: afecta a las páginas que se mapeen en el futuro

**Liberar las páginas actuales del proceso:**

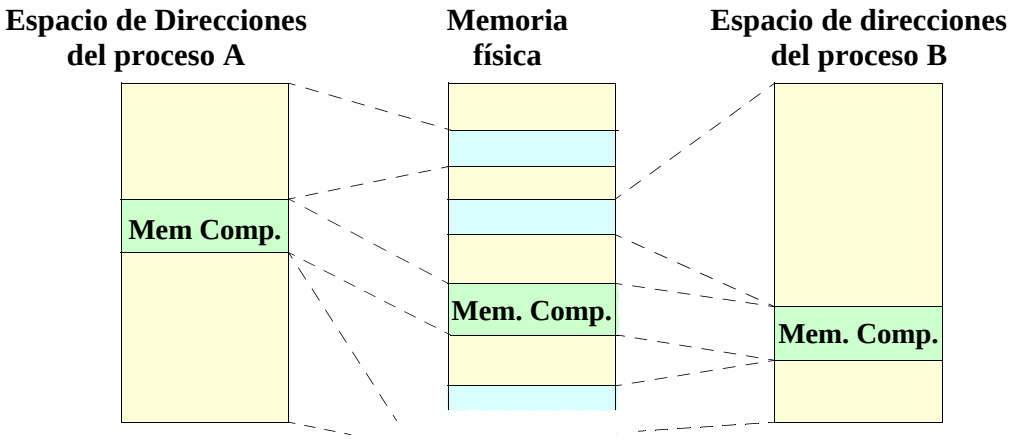
```
int munlockall (void);
```



# 4. Mecanismos de memoria compartida

**Los procesos tienen espacios de direcciones independientes**

**Un objeto de memoria compartida es una región de memoria que se puede mapear en el espacio de direcc. de un proceso**



# Objetos de Memoria Compartida

## Abrir un objeto de memoria compartida:

```
int shm_open (const char *name, int oflag,
              mode_t mode);
```

- establece una conexión entre un objeto de memoria compartida y un descriptor de fichero (que es el valor de retorno)
- para conseguir portabilidad, el nombre `name` debe tener la forma "`/nombre`"; no requiere estar en el sistema de ficheros
- `mode` indica los permisos si el objeto se crea



# Objetos de Memoria Compartida (cont.)

## Abrir un objeto de memoria compartida (cont.):

- `oflag` indica las opciones:
  - `O_RDONLY`: sólo lectura
  - `O_RDWR`: lectura y escritura
  - `O_CREAT`: si el objeto no existe, se crea
  - `O_EXCL`: si el objeto ya existía, error
  - `O_TRUNC`: si el objeto existe se trunca a tamaño 0

## El tamaño del objeto de memoria se puede fijar con:

```
int ftruncate (int fildes, off_t length);
```

## Borrar un objeto de memoria compartida:

```
int shm_unlink (const char *name);
```

# Mapear un Objeto de Memoria

## Función mmap():

```
void *mmap (void *addr, size_t len, int prot,
            int flags, int fildes, off_t off);
```

- mapea una parte del objeto de memoria representado por `fildes` en el espacio de direcciones
- la parte del objeto empieza en `off` y tiene `len` bytes
- la dirección inicial en el espacio de direcciones es el valor de retorno, y se obtiene de tres formas:
  - opción `MAP_FIXED`: la dirección es `addr` ¡Peligro!
  - sin opción `MAP_FIXED`, y `addr`≠NULL: la dirección es una función de `addr`, definida por la implementación
  - sin opción `MAP_FIXED`, y `addr`=NULL: el sistema elige la dirección; es la opción recomendada



# Mapear un Objeto de Memoria (cont.)

- `flags` especifica las opciones:
  - `MAP_FIXED`: interpretar `addr` de forma exacta
  - `MAP_SHARED`: los cambios son compartidos
  - `MAP_PRIVATE`: los cambios son privados
- `prot` especifica el tipo de acceso:
  - `PROT_READ`: se puede leer
  - `PROT_WRITE`: se puede escribir
  - `PROT_EXEC`: se puede ejecutar
  - `PROT_NONE`: no se puede acceder a los datos

## "Desmapear" direcciones:

```
int munmap (void *addr, size_t len);
```

# 5. Ejemplo de Memoria Compartida

## Fichero “mapfunc.h”:

```
void open_and_map (void **addr, int *fd);
void unmap_and_delete (void *addr, int fd);
```

## Fichero “mapfunc.c”

```
#include <sys/mman.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include "mapfunc.h"
```



# Ejemplo de memoria compartida (cont.)

```
void open_and_map (void **addr, int *fd)
{

  if ((*fd=shm_open ("/mem.int",O_RDWR|O_CREAT,
      S_IRUSR|S_IWUSR)) == -1)
  {
    perror ("error in shm_open\n");
  }
  if (ftruncate(*fd,100) == -1) {
    perror ("error in ftruncate\n");
  }
  if ((*addr=mmap(0,100,PROT_READ|PROT_WRITE,MAP_SHARED,*fd,0))
       == MAP_FAILED)
  {
    perror ("error in mmap\n");
  }
}
```

# Ejemplo de memoria compartida (cont.)

```
void unmap_and_delete (void *addr, int fd)
{
  if (munmap(addr,100)== -1) {
    perror ("error in munmap\n");
  }
  if (close (fd) == -1) {
    perror ("error in close\n");
  }
  if (shm_unlink("/mem.int") == -1) {
    if (errno != ENOENT) {
      perror("error in shm_unlink\n");
    }
  }
}
```



# Ejemplo de memoria compartida (cont.)

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include "mapfunc.h"

int main()
{
  pid_t childpid;
  int *p_addr;
  int p_fd;

  if ((childpid=fork()) == -1) {
    perror("can't fork");
    exit(1);
  } else if (childpid == 0) {
    // child process
    execlp("mem2","mem2",NULL);
    printf ("mem2 not found\n");
    exit (1);
```

# Ejemplo de memoria compartida (cont.)

```
  } else {
      // parent process
      open_and_map ((void *)&p_addr,&p_fd);
      *p_addr=0;
      do {
        *p_addr = *p_addr+2;
      } while (*p_addr % 2 == 0);
      unmap_and_delete (p_addr,p_fd);
      exit (0);
    }
}
```



# Ejemplo de memoria compartida (cont.)

```
#include <stdio.h>
#include <unistd.h>
#include "mapfunc.h"

// Muestra 20 veces en pantalla una variable compartida
// Luego, la hace impar

int main()
{
  int *c_addr;
  int c_fd;
  int i;

  open_and_map ((void *)&c_addr,&c_fd);
```

# Ejemplo de memoria compartida (cont.)

```
  *c_addr=0;
  for (i=1;i<20;i++) {
    printf("i=%d\n",*c_addr);
    sleep (1);
  }
  *c_addr=1;
  unmap_and_delete (c_addr,c_fd);
  exit (0);
}
```