

Memoria Compartida

Eduardo C. Garrido Merchán

Sistemas Operativos. Práctica 3. Semana 1.

Motivación

- ▶ Cada proceso tiene su espacio de direcciones, si queremos comunicar dos procesos por memoria, no podemos usar su espacio.
- ▶ Si un proceso intenta acceder a la memoria de otro, provocará una violación de segmento.
- ▶ Siempre podemos usar tuberías pero... Si queremos comunicar mensajes entre decenas de procesos se vuelve caótico.
- ▶ Siempre podemos usar ficheros o bases de datos... pero el acceso será muy lento.
- ▶ Para comunicar procesos entre sí sin estas adversidades usaremos **memoria compartida**

Motivación

- ▶ Cada proceso tiene su espacio de direcciones, si queremos comunicar dos procesos por memoria, no podemos usar su espacio.
- ▶ Si un proceso intenta acceder a la memoria de otro, provocará una violación de segmento.
- ▶ Siempre podemos usar tuberías pero... Si queremos comunicar mensajes entre decenas de procesos se vuelve caótico.
- ▶ Siempre podemos usar ficheros o bases de datos... pero el acceso será muy lento.
- ▶ Para comunicar procesos entre sí sin estas adversidades usaremos **memoria compartida**

Motivación

- ▶ Cada proceso tiene su espacio de direcciones, si queremos comunicar dos procesos por memoria, no podemos usar su espacio.
- ▶ Si un proceso intenta acceder a la memoria de otro, provocará una violación de segmento.
- ▶ Siempre podemos usar tuberías pero... Si queremos comunicar mensajes entre decenas de procesos se vuelve caótico.
- ▶ Siempre podemos usar ficheros o bases de datos... pero el acceso será muy lento.
- ▶ Para comunicar procesos entre sí sin estas adversidades usaremos **memoria compartida**

Motivación

- ▶ Cada proceso tiene su espacio de direcciones, si queremos comunicar dos procesos por memoria, no podemos usar su espacio.
- ▶ Si un proceso intenta acceder a la memoria de otro, provocará una violación de segmento.
- ▶ Siempre podemos usar tuberías pero... Si queremos comunicar mensajes entre decenas de procesos se vuelve caótico.
- ▶ Siempre podemos usar ficheros o bases de datos... pero el acceso será muy lento.
- ▶ Para comunicar procesos entre sí sin estas adversidades usaremos **memoria compartida**

Motivación

- ▶ Cada proceso tiene su espacio de direcciones, si queremos comunicar dos procesos por memoria, no podemos usar su espacio.
- ▶ Si un proceso intenta acceder a la memoria de otro, provocará una violación de segmento.
- ▶ Siempre podemos usar tuberías pero... Si queremos comunicar mensajes entre decenas de procesos se vuelve caótico.
- ▶ Siempre podemos usar ficheros o bases de datos... pero el acceso será muy lento.
- ▶ Para comunicar procesos entre sí sin estas adversidades usaremos **memoria compartida**

Definición

- ▶ La memoria compartida es una zona de memoria común gestionada por el sistema operativo.
- ▶ Actuará como una pizarra, en la que cualquier proceso, si consigue acceso, podrá leer y escribir.
- ▶ De este modo, por ejemplo, lo que escriba un proceso lo pueden leer varios.
- ▶ Cada proceso incluirá en su direccionamiento virtual dicha región de memoria, independiente al principio del resto de procesos.

Definición

- ▶ La memoria compartida es una zona de memoria común gestionada por el sistema operativo.
- ▶ Actuará como una pizarra, en la que cualquier proceso, si consigue acceso, podrá leer y escribir.
- ▶ De este modo, por ejemplo, lo que escriba un proceso lo pueden leer varios.
- ▶ Cada proceso incluirá en su direccionamiento virtual dicha región de memoria, independiente al principio del resto de procesos.

Definición

- ▶ La memoria compartida es una zona de memoria común gestionada por el sistema operativo.
- ▶ Actuará como una pizarra, en la que cualquier proceso, si consigue acceso, podrá leer y escribir.
- ▶ De este modo, por ejemplo, lo que escriba un proceso lo pueden leer varios.
- ▶ Cada proceso incluirá en su direccionamiento virtual dicha región de memoria, independiente al principio del resto de procesos.

Definición

- ▶ La memoria compartida es una zona de memoria común gestionada por el sistema operativo.
- ▶ Actuará como una pizarra, en la que cualquier proceso, si consigue acceso, podrá leer y escribir.
- ▶ De este modo, por ejemplo, lo que escriba un proceso lo pueden leer varios.
- ▶ Cada proceso incluirá en su direccionamiento virtual dicha región de memoria, independiente al principio del resto de procesos.

Programación en C de memoria compartida, descripción general

- ▶ Las funciones vistas residen en *sys/ipc.h*, *sys/shm.h* y *sys/types.h*. Las básicas para trabajar con memoria compartida son:
- ▶ *shmget*: Crea una nueva región de memoria compartida o devuelve una existente.
- ▶ *shmat*: Une lógicamente una región al espacio de direccionamiento virtual de un proceso.
- ▶ *shmdt*: Separa una región del espacio de direccionamiento virtual de un proceso.
- ▶ *shmctl*: Realiza funciones de gestión sobre la memoria compartida.

Programación en C de memoria compartida, descripción general

- ▶ Las funciones vistas residen en *sys/ipc.h*, *sys/shm.h* y *sys/types.h*. Las básicas para trabajar con memoria compartida son:
- ▶ *shmget*: Crea una nueva región de memoria compartida o devuelve una existente.
- ▶ *shmat*: Une lógicamente una región al espacio de direccionamiento virtual de un proceso.
- ▶ *shmdt*: Separa una región del espacio de direccionamiento virtual de un proceso.
- ▶ *shmctl*: Realiza funciones de gestión sobre la memoria compartida.

Programación en C de memoria compartida, descripción general

- ▶ Las funciones vistas residen en *sys/ipc.h*, *sys/shm.h* y *sys/types.h*. Las básicas para trabajar con memoria compartida son:
- ▶ *shmget*: Crea una nueva región de memoria compartida o devuelve una existente.
- ▶ *shmat*: Une lógicamente una región al espacio de direccionamiento virtual de un proceso.
- ▶ *shmdt*: Separa una región del espacio de direccionamiento virtual de un proceso.
- ▶ *shmctl*: Realiza funciones de gestión sobre la memoria compartida.

Programación en C de memoria compartida, descripción general

- ▶ Las funciones vistas residen en *sys/ipc.h*, *sys/shm.h* y *sys/types.h*. Las básicas para trabajar con memoria compartida son:
- ▶ *shmget*: Crea una nueva región de memoria compartida o devuelve una existente.
- ▶ *shmat*: Une lógicamente una región al espacio de direccionamiento virtual de un proceso.
- ▶ *shmdt*: Separa una región del espacio de direccionamiento virtual de un proceso.
- ▶ *shmctl*: Realiza funciones de gestión sobre la memoria compartida.

Programación en C de memoria compartida, descripción general

- ▶ Las funciones vistas residen en *sys/ipc.h*, *sys/shm.h* y *sys/types.h*. Las básicas para trabajar con memoria compartida son:
- ▶ *shmget*: Crea una nueva región de memoria compartida o devuelve una existente.
- ▶ *shmat*: Une lógicamente una región al espacio de direccionamiento virtual de un proceso.
- ▶ *shmdt*: Separa una región del espacio de direccionamiento virtual de un proceso.
- ▶ *shmctl*: Realiza funciones de gestión sobre la memoria compartida.

1. Creación de una nueva región de memoria compartida

- ▶ Es el primer paso para trabajar con memoria compartida.
- ▶ Usaremos la función `int shmget(key_t key, int size, int shmflg)`.
- ▶ El entero que retorna es el identificador de la región. Lo necesitarán los procesos para acceder a ella. -1 error.
- ▶ `key_t key`: El núcleo busca la región identificada por `key` (entero). Sino la encuentra y `shmflg` incluye `IPC_CREAT` crea una.
- ▶ `int size`: Número de bytes de la región.
- ▶ `int shmflg`: Opciones para la creación de la memoria.
Creación: `IPC_CREAT` y permisos en algunos sistemas: `SHM_W` y `SHM_R`.

1. Creación de una nueva región de memoria compartida

- ▶ Es el primer paso para trabajar con memoria compartida.
- ▶ Usaremos la función `int shmget(key_t key, int size, int shmflg)`.
- ▶ El entero que retorna es el identificador de la región. Lo necesitarán los procesos para acceder a ella. -1 error.
- ▶ `key_t key`: El núcleo busca la región identificada por `key` (entero). Sino la encuentra y `shmflg` incluye `IPC_CREAT` crea una.
- ▶ `int size`: Número de bytes de la región.
- ▶ `int shmflg`: Opciones para la creación de la memoria.
Creación: `IPC_CREAT` y permisos en algunos sistemas: `SHM_W` y `SHM_R`.

1. Creación de una nueva región de memoria compartida

- ▶ Es el primer paso para trabajar con memoria compartida.
- ▶ Usaremos la función `int shmget(key_t key, int size, int shmflg)`.
- ▶ El entero que retorna es el identificador de la región. Lo necesitarán los procesos para acceder a ella. -1 error.
- ▶ `key_t key`: El núcleo busca la región identificada por `key` (entero). Sino la encuentra y `shmflg` incluye `IPC_CREAT` crea una.
- ▶ `int size`: Número de bytes de la región.
- ▶ `int shmflg`: Opciones para la creación de la memoria.
Creación: `IPC_CREAT` y permisos en algunos sistemas: `SHM_W` y `SHM_R`.

1. Creación de una nueva región de memoria compartida

- ▶ Es el primer paso para trabajar con memoria compartida.
- ▶ Usaremos la función `int shmget(key_t key, int size, int shmflg)`.
- ▶ El entero que retorna es el identificador de la región. Lo necesitarán los procesos para acceder a ella. -1 error.
- ▶ `key_t key`: El núcleo busca la región identificada por `key` (entero). Sino la encuentra y `shmflg` incluye `IPC_CREAT` crea una.
- ▶ `int size`: Número de bytes de la región.
- ▶ `int shmflg`: Opciones para la creación de la memoria.
Creación: `IPC_CREAT` y permisos en algunos sistemas: `SHM_W` y `SHM_R`.

1. Creación de una nueva región de memoria compartida

- ▶ Es el primer paso para trabajar con memoria compartida.
- ▶ Usaremos la función `int shmget(key_t key, int size, int shmflg)`.
- ▶ El entero que retorna es el identificador de la región. Lo necesitarán los procesos para acceder a ella. -1 error.
- ▶ `key_t key`: El núcleo busca la región identificada por `key` (entero). Sino la encuentra y `shmflg` incluye `IPC_CREAT` crea una.
- ▶ `int size`: Número de bytes de la región.
- ▶ `int shmflg`: Opciones para la creación de la memoria.
Creación: `IPC_CREAT` y permisos en algunos sistemas: `SHM_W` y `SHM_R`.

1. Creación de una nueva región de memoria compartida

- ▶ Es el primer paso para trabajar con memoria compartida.
- ▶ Usaremos la función `int shmget(key_t key, int size, int shmflg)`.
- ▶ El entero que retorna es el identificador de la región. Lo necesitarán los procesos para acceder a ella. -1 error.
- ▶ `key_t key`: El núcleo busca la región identificada por `key` (entero). Sino la encuentra y `shmflg` incluye `IPC_CREAT` crea una.
- ▶ `int size`: Número de bytes de la región.
- ▶ `int shmflg`: Opciones para la creación de la memoria.
Creación: `IPC_CREAT` y permisos en algunos sistemas: `SHM_W` y `SHM_R`.

Ejemplo

```
#define FILEKEY "/bin/cat" /*Util para ftok */

#define KEY 1300
#define MAXBUF 10

int main (int argc, char *argv[]) {

    int *buffer; /* shared buffer */
    int key, id_zone;
    int i;

    char c;

    /* Key to shared memory */
    int key = ftok(FILEKEY, KEY);
    if (key == -1) {
        fprintf (stderr, "Error with key \n");
        return -1;
    }

    /* We create the shared memory */
    id_zone = shmget (key, sizeof(int)*MAXBUF, IPC_CREAT | IPC_EXCL
                     | SHM_R | SHM_W);
```

2. Unir lógicamente una región al espacio de memoria de un proceso

- ▶ Una vez creada la memoria, uniremos el espacio a todos los procesos que vayan a usar la memoria.
- ▶ Usaremos la función `char *shmat(int shmid, char *addr, int shmflg)`.
- ▶ Devuelve la dirección de memoria donde el núcleo ha unido la región.
- ▶ `shmid`: Valor que devuelve `shmget` para identificar la memoria.
- ▶ `addr`: Dirección donde queremos unir la memoria. Si ponemos 0, la asignará el núcleo.
- ▶ `shmflg`: Permisos de la región.

2. Unir lógicamente una región al espacio de memoria de un proceso

- ▶ Una vez creada la memoria, uniremos el espacio a todos los procesos que vayan a usar la memoria.
- ▶ Usaremos la función `char *shmat(int shmid, char *addr, int shmflg)`.
- ▶ Devuelve la dirección de memoria donde el núcleo ha unido la región.
- ▶ `shmid`: Valor que devuelve `shmget` para identificar la memoria.
- ▶ `addr`: Dirección donde queremos unir la memoria. Si ponemos 0, la asignará el núcleo.
- ▶ `shmflg`: Permisos de la región.

2. Unir lógicamente una región al espacio de memoria de un proceso

- ▶ Una vez creada la memoria, uniremos el espacio a todos los procesos que vayan a usar la memoria.
- ▶ Usaremos la función `char *shmat(int shmid, char *addr, int shmflg)`.
- ▶ Devuelve la dirección de memoria donde el núcleo ha unido la región.
- ▶ `shmid`: Valor que devuelve `shmget` para identificar la memoria.
- ▶ `addr`: Dirección donde queremos unir la memoria. Si ponemos 0, la asignará el núcleo.
- ▶ `shmflg`: Permisos de la región.

2. Unir lógicamente una región al espacio de memoria de un proceso

- ▶ Una vez creada la memoria, uniremos el espacio a todos los procesos que vayan a usar la memoria.
- ▶ Usaremos la función `char *shmat(int shmid, char *addr, int shmflg)`.
- ▶ Devuelve la dirección de memoria donde el núcleo ha unido la región.
- ▶ `shmid`: Valor que devuelve `shmget` para identificar la memoria.
- ▶ `addr`: Dirección donde queremos unir la memoria. Si ponemos 0, la asignará el núcleo.
- ▶ `shmflg`: Permisos de la región.

2. Unir lógicamente una región al espacio de memoria de un proceso

- ▶ Una vez creada la memoria, uniremos el espacio a todos los procesos que vayan a usar la memoria.
- ▶ Usaremos la función `char *shmat(int shmid, char *addr, int shmflg)`.
- ▶ Devuelve la dirección de memoria donde el núcleo ha unido la región.
- ▶ `shmid`: Valor que devuelve `shmget` para identificar la memoria.
- ▶ `addr`: Dirección donde queremos unir la memoria. Si ponemos 0, la asignará el núcleo.
- ▶ `shmflg`: Permisos de la región.

2. Unir lógicamente una región al espacio de memoria de un proceso

- ▶ Una vez creada la memoria, uniremos el espacio a todos los procesos que vayan a usar la memoria.
- ▶ Usaremos la función `char *shmat(int shmid, char *addr, int shmflg)`.
- ▶ Devuelve la dirección de memoria donde el núcleo ha unido la región.
- ▶ `shmid`: Valor que devuelve `shmget` para identificar la memoria.
- ▶ `addr`: Dirección donde queremos unir la memoria. Si ponemos 0, la asignará el núcleo.
- ▶ `shmflg`: Permisos de la región.

Ejemplo

```
/* we link the zone to share */
buffer = shmat (id_zone, (char *)0, 0);
if (buffer == NULL) {
    fprintf (stderr, "Error reserve shared memory \n");
    return -1;
}

printf ("Pointer buffer shared memory: %p\n", buffer);
```

3. Gestionar la memoria o separarla de un proceso

- ▶ Una vez unida, podemos escribir o leer de ella con los procesos a los que se una.
- ▶ Podemos hacer operaciones de gestión o separar la memoria de un proceso.
- ▶ Separar memoria de un proceso: *int shmdt(char *addr)*.
Donde *addr* es la dirección devuelta por *shmat*, con control de errores.
- ▶ *shmctl(int shmid, int cmd, struct shmid_ds shmstatbuf)*:
Busca el estado y establece parámetros para la memoria.
- ▶ *shmid* identifica la región, *cmd* controla la operación a realizar y *shmstatbuf* nos proporciona los permisos, tamaño/ no del segmento, creador de la memoria, etc...

3. Gestionar la memoria o separarla de un proceso

- ▶ Una vez unida, podemos escribir o leer de ella con los procesos a los que se una.
- ▶ Podemos hacer operaciones de gestión o separar la memoria de un proceso.
- ▶ Separar memoria de un proceso: *int shmdt(char *addr)*.
Donde *addr* es la dirección devuelta por *shmat*, con control de errores.
- ▶ *shmctl(int shmid, int cmd, struct shmid_ds shmstatbuf)*:
Busca el estado y establece parámetros para la memoria.
- ▶ *shmid* identifica la región, *cmd* controla la operación a realizar y *shmstatbuf* nos proporciona los permisos, tamaño/ no del segmento, creador de la memoria, etc...

3. Gestionar la memoria o separarla de un proceso

- ▶ Una vez unida, podemos escribir o leer de ella con los procesos a los que se una.
- ▶ Podemos hacer operaciones de gestión o separar la memoria de un proceso.
- ▶ Separar memoria de un proceso: *int shmdt(char *addr)*.
Donde *addr* es la dirección devuelta por *shmat*, con control de errores.
- ▶ *shmctl(int shmid, int cmd, struct shmid_ds shmstatbuf)*:
Busca el estado y establece parámetros para la memoria.
- ▶ *shmid* identifica la región, *cmd* controla la operación a realizar y *shmstatbuf* nos proporciona los permisos, tamaño/ no del segmento, creador de la memoria, etc...

3. Gestionar la memoria o separarla de un proceso

- ▶ Una vez unida, podemos escribir o leer de ella con los procesos a los que se una.
- ▶ Podemos hacer operaciones de gestión o separar la memoria de un proceso.
- ▶ Separar memoria de un proceso: *int shmdt(char *addr)*.
Donde *addr* es la dirección devuelta por *shmat*, con control de errores.
- ▶ *shmctl(int shmid, int cmd, struct shmid_ds shmstatbuf)*:
Busca el estado y establece parámetros para la memoria.
- ▶ *shmid* identifica la región, *cmd* controla la operación a realizar y *shmstatbuf* nos proporciona los permisos, tamaño/ no del segmento, creador de la memoria, etc...

3. Gestionar la memoria o separarla de un proceso

- ▶ Una vez unida, podemos escribir o leer de ella con los procesos a los que se una.
- ▶ Podemos hacer operaciones de gestión o separar la memoria de un proceso.
- ▶ Separar memoria de un proceso: `int shmdt(char *addr)`.
Donde `addr` es la dirección devuelta por `shmat`, con control de errores.
- ▶ `shmctl(int shmid, int cmd, struct shmid_ds shmstatbuf)`:
Busca el estado y establece parámetros para la memoria.
- ▶ `shmid` identifica la región, `cmd` controla la operación a realizar y `shmstatbuf` nos proporciona los permisos, tamaño/ no del segmento, creador de la memoria, etc...

3. Gestionar la memoria o separarla de un proceso

- ▶ Una vez unida, podemos escribir o leer de ella con los procesos a los que se una.
- ▶ Podemos hacer operaciones de gestión o separar la memoria de un proceso.
- ▶ Separar memoria de un proceso: `int shmdt(char *addr)`.
Donde `addr` es la dirección devuelta por `shmat`, con control de errores.
- ▶ `shmctl(int shmid, int cmd, struct shmid_ds shmstatbuf)`:
Busca el estado y establece parámetros para la memoria.
- ▶ `shmid` identifica la región, `cmd` controla la operación a realizar y `shmstatbuf` nos proporciona los permisos, tamaño/ no del segmento, creador de la memoria, etc...

Operaciones que se pueden hacer con el parámetro *cmd*

- ▶ El parámetro *cmd* de la función *shmctl* controla la operación a realizar en la memoria, algunas operaciones son:
- ▶ *IPC_STAT*: Copia información del kernel de la memoria identificada por *shmid* en *shmstatbuf*.
- ▶ *IPC_SET*: Al contrario, escribe información en la memoria identificada por *shmid* desde *shmstatbuf*.
- ▶ *IPC_RMID*: Elimina la estructura especificada por *shmid*.

Operaciones que se pueden hacer con el parámetro *cmd*

- ▶ El parámetro *cmd* de la función *shmctl* controla la operación a realizar en la memoria, algunas operaciones son:
- ▶ *IPC_STAT*: Copia información del kernel de la memoria identificada por *shmid* en *shmstatbuf*.
- ▶ *IPC_SET*: Al contrario, escribe información en la memoria identificada por *shmid* desde *shmstatbuf*.
- ▶ *IPC_RMID*: Elimina la estructura especificada por *shmid*.

Operaciones que se pueden hacer con el parámetro *cmd*

- ▶ El parámetro *cmd* de la función *shmctl* controla la operación a realizar en la memoria, algunas operaciones son:
- ▶ *IPC_STAT*: Copia información del kernel de la memoria identificada por *shmid* en *shmstatbuf*.
- ▶ *IPC_SET*: Al contrario, escribe información en la memoria identificada por *shmid* desde *shmstatbuf*.
- ▶ *IPC_RMID*: Elimina la estructura especificada por *shmid*.

Operaciones que se pueden hacer con el parámetro *cmd*

- ▶ El parámetro *cmd* de la función *shmctl* controla la operación a realizar en la memoria, algunas operaciones son:
- ▶ *IPC_STAT*: Copia información del kernel de la memoria identificada por *shmid* en *shmstatbuf*.
- ▶ *IPC_SET*: Al contrario, escribe información en la memoria identificada por *shmid* desde *shmstatbuf*.
- ▶ *IPC_RMID*: Elimina la estructura especificada por *shmid*.

Ejemplo

```
for (i = 0; i < MAXBUF; i++)  
    buffer[i] = i;
```

```
/* The daemon executes until press some character */
```

```
c = getchar();
```

```
/* Free the shared memory */
```

```
shmdt ((char *)buffer);
```

```
shmctl (id_zone, IPC_RMID, (struct shmid_ds *)NULL);
```

```
return 0;
```

```
}
```

Resumen de las operaciones con memoria compartida

- ▶ 1. Crear una *key*: Es un identificador de IPC (Inter Process Communication). Se puede convertir una ruta del sistema en un identificador IPC. Necesario para crear la memoria virtual. Usamos *ftok()*.
- ▶ 2. Crear el segmento de la memoria compartida: *shmget()*.
- ▶ 3. Operar con la memoria compartida: *shmat()*.
- ▶ 4. Destruir la memoria compartida: 4a. Separarla del proceso *shmdt()* y 4b. destruirla *shmctl()*.
- ▶ Ojo, *shmctl()* no elimina el segmento, lo marca para ser eliminado. Cuando el sistema comprueba que no está unido a ningún proceso, el sistema lo elimina.

Resumen de las operaciones con memoria compartida

- ▶ 1. Crear una *key*: Es un identificador de IPC (Inter Process Communication). Se puede convertir una ruta del sistema en un identificador IPC. Necesario para crear la memoria virtual. Usamos *ftok()*.
- ▶ 2. Crear el segmento de la memoria compartida: *shmget()*.
- ▶ 3. Operar con la memoria compartida: *shmat()*.
- ▶ 4. Destruir la memoria compartida: 4a. Separarla del proceso *shmdt()* y 4b. destruirla *shmctl()*.
- ▶ Ojo, *shmctl()* no elimina el segmento, lo marca para ser eliminado. Cuando el sistema comprueba que no está unido a ningún proceso, el sistema lo elimina.

Resumen de las operaciones con memoria compartida

- ▶ 1. Crear una *key*: Es un identificador de IPC (Inter Process Communication). Se puede convertir una ruta del sistema en un identificador IPC. Necesario para crear la memoria virtual. Usamos *ftok()*.
- ▶ 2. Crear el segmento de la memoria compartida: *shmget()*.
- ▶ 3. Operar con la memoria compartida: *shmat()*.
- ▶ 4. Destruir la memoria compartida: 4a. Separarla del proceso *shmdt()* y 4b. destruirla *shmctl()*.
- ▶ Ojo, *shmctl()* no elimina el segmento, lo marca para ser eliminado. Cuando el sistema comprueba que no está unido a ningún proceso, el sistema lo elimina.

Resumen de las operaciones con memoria compartida

- ▶ 1. Crear una *key*: Es un identificador de IPC (Inter Process Communication). Se puede convertir una ruta del sistema en un identificador IPC. Necesario para crear la memoria virtual. Usamos *ftok()*.
- ▶ 2. Crear el segmento de la memoria compartida: *shmget()*.
- ▶ 3. Operar con la memoria compartida: *shmat()*.
- ▶ 4. Destruir la memoria compartida: 4a. Separarla del proceso *shmdt()* y 4b. destruirla *shmctl()*.
- ▶ Ojo, *shmctl()* no elimina el segmento, lo marca para ser eliminado. Cuando el sistema comprueba que no está unido a ningún proceso, el sistema lo elimina.

Resumen de las operaciones con memoria compartida

- ▶ 1. Crear una *key*: Es un identificador de IPC (Inter Process Communication). Se puede convertir una ruta del sistema en un identificador IPC. Necesario para crear la memoria virtual. Usamos *ftok()*.
- ▶ 2. Crear el segmento de la memoria compartida: *shmget()*.
- ▶ 3. Operar con la memoria compartida: *shmat()*.
- ▶ 4. Destruir la memoria compartida: 4a. Separarla del proceso *shmdt()* y 4b. destruirla *shmctl()*.
- ▶ Ojo, *shmctl()* no elimina el segmento, lo marca para ser eliminado. Cuando el sistema comprueba que no está unido a ningún proceso, el sistema lo elimina.