Addendum sulla Shared Memory

EM

Spazi di indirizzamento dei processi

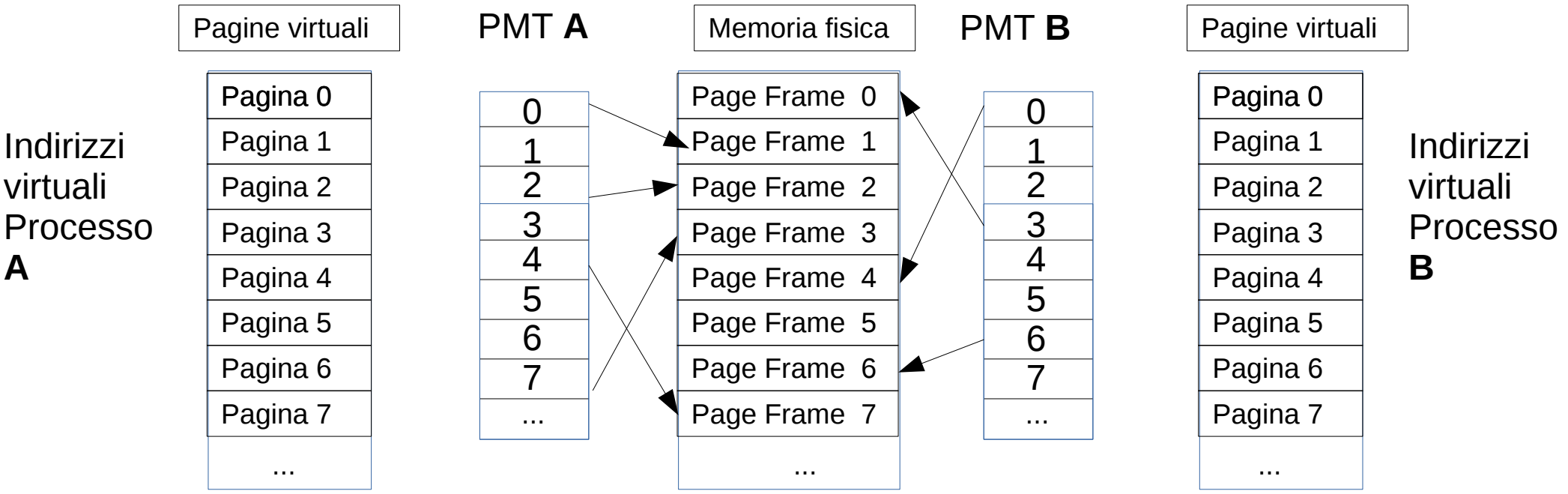
- Spazio di indirizzamento è il range

  indirizzo virtuale minimo – indirizzo virtuale massimo generato da un processo

- Page frames non possono essere condivise tra processi!

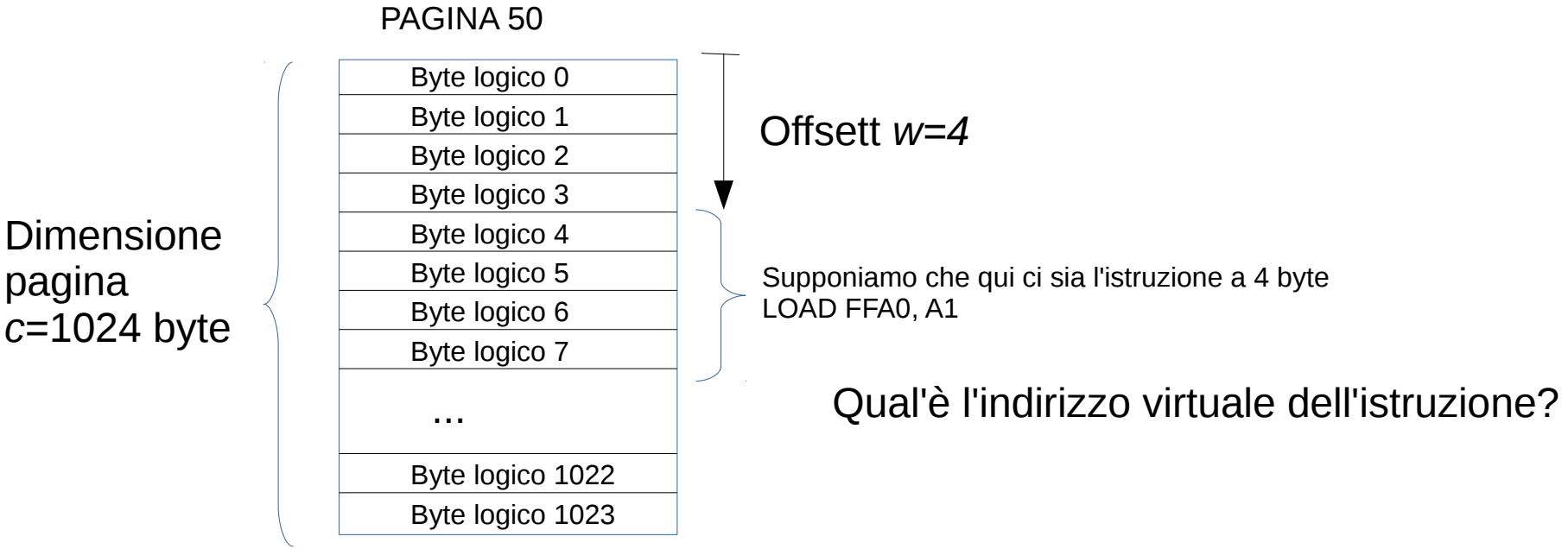
- Motivo: le Page Map Table convertono gli indirizzi virtuali in page frames diverse

- **La PMT contiene tanti elementi quante sono le pagine virtuali**



Spazi di indirizzamento dei processi

- Non ci può essere sovrapposizione di pagine virtuali semplicemente perché il kernel si accorge se una pagina fisica è allocata da un altro processo
- Gli Spazi di indirizzamento sono distinti per diversi processi!
- Cioè: **stessi indirizzi virtuali in processi diversi corrispondono a diverse locazioni di memoria fisica**
- Esempio: pagina virtuale *n*=50 di un processo. Supponiamo pagine di *c*=1024 byte

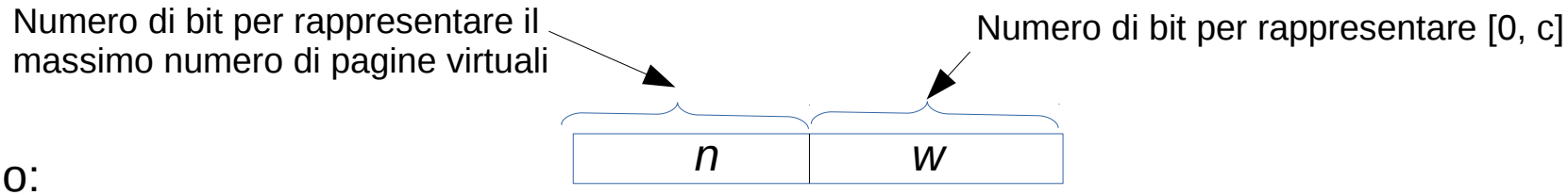


Spazi di indirizzamento dei processi

- Indichiamo l'indirizzo virtuale α di ogni istruzione in notazione posizionale con due soli termini, *n* e *w*:

$$\alpha = n*c^1+w*c^0 = n*c + w = 50*c + w = 51204 \qquad (w \in [0, c])$$

dove *n* è il numero di pagina virtuale, *c* è la dimensione della pagina virtuale, *w* l'offsett nella pagina virtuale.



- In binario:

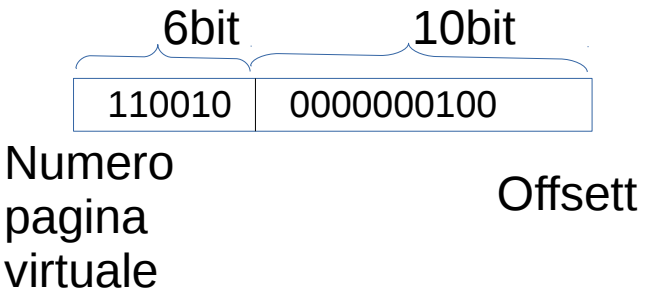
- Qual'è l'indirizzo in memoria fisica β corrispondente ad α?

$$\beta=\text{PMT}[n]*c+w = PMT[50]+w$$

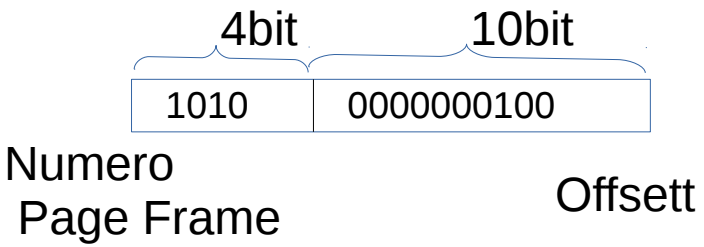
- Assumendo che PMT[*50*] = 10, β=10*c+w = *10244*

Spazi di indirizzamento dei processi

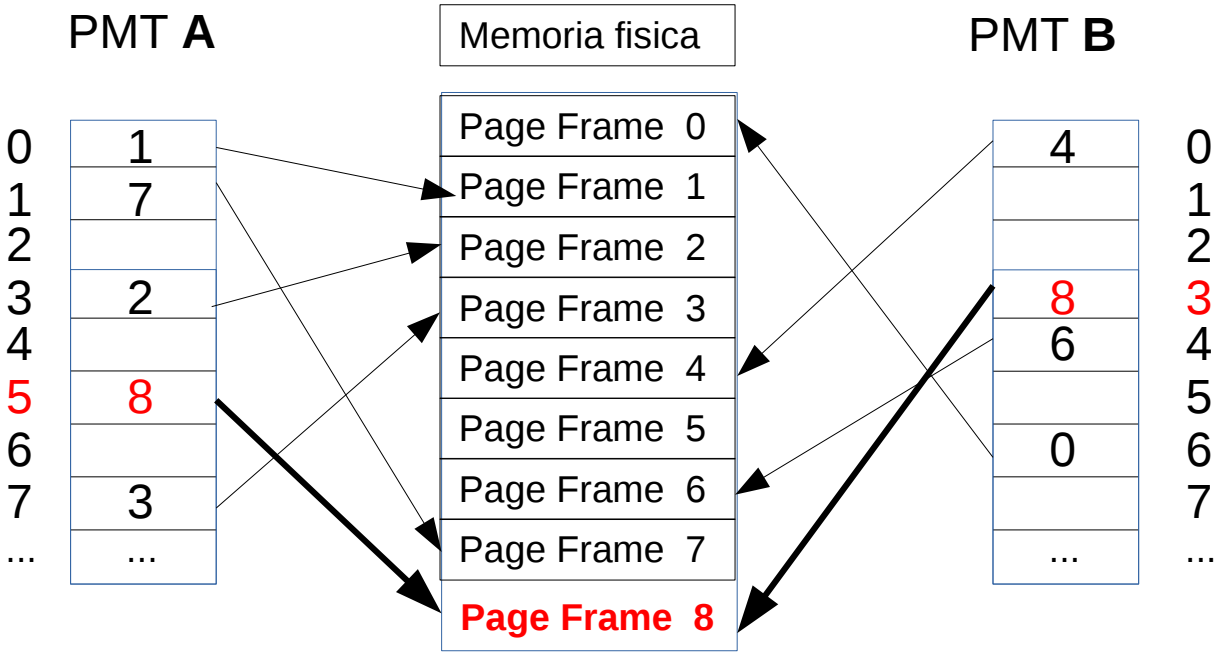
- Supponiamo di avere 64 pagine virtuali nel nostro processo → 6bit
- Supponiamo di avere pagine di 1024 byte → 10bit
- Indirizzo virtuale di 16 bit:



- Supponiamo di avere 16 page frame da 1024 bit (memoria di 16Kbyte)
- Indirizzo fisico di 14 bit:

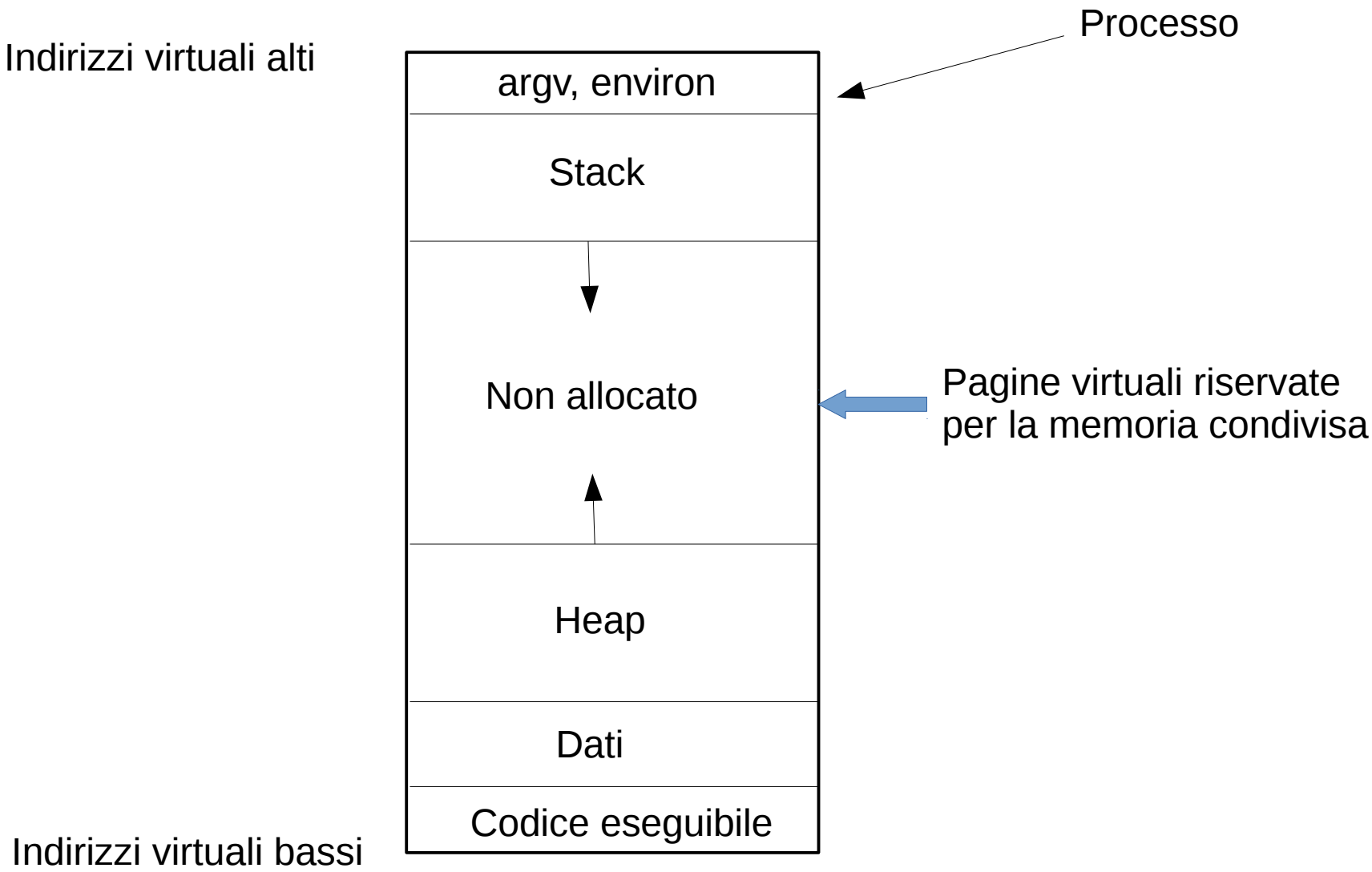


Pagine fisiche condivise tra processi



- Alcune pagine virtuali dei processi tra Heap e Stack vengono riservate
- Alcune page frame in memoria fisica vengono riservate
- Le relative page frame in page map table sono fissate:
  - Cioè: **ci sono diversi indirizzi virtuali in processi diversi che corrispondono alle stesse locazioni di memoria fisica**

Spazio di indirizzamento virtuale di un processo in Linux



Pagine fisiche condivise tra processi

- Chi sceglie le pagine virtuali all'interno dello spazio di indirizzamento dei processi?
  - La funzione *mmap* della Realtime Extensions Library di POSIX.1b, librt
  - Compilazione con librt: `$gcc source.c -o source -lrt`
  - API di librt: interfacce specificate dall'estensione Realtime POSIX.1b:
    - Memoria condivisa
    - Semafori
    - Passaggio di messaggi
    - Schedulazione di processi
    - I/O asincrono e I/O sincronizzato
    - Funzioni per la misura del tempo

- Principali API per la shared memory:
  - mmap, unmap → mappa la regione nello spazio di indirizzamento virtuale del processo
  - ftruncate → stabilisce la dimensione della regione
  - shm_open → apre o crea una regione condivisa
  - shm_unlink → rimuove la regione
  - fstat → ritorna una struttura stat per la regione
  - fchmod → cambia i permessi della regione

Definizione delle API per la Shared Memory

- mmap, unmap → mappa un file in memoria, elimina la mappatura
  - Mappa un file aperto nella memoria condivisa.

```
#include <sys/mman.h>

 void *mmap(void *addr, size_t length, int prot, int flags,
            int fd, off_t offset);
 int munmap(void *addr, size_t length);
```

- Argomenti:
  - addr → specifica l'indirizzo virtuale. Se addr è zero, l'indirizzo è scelto dal kernel
  - length → stabilisce la dimensione della memoria condivisa, o del file
  - prot → definisce la protezione della memoria. Può essere:

    PROT_EXEC, PROT_,READ_WRITE, PROT_NONE
  - flags → stabilisce se gli aggiornamenti alla SHM sono visibili ad altri processi, e se sono riportati nel file collegato. Principali flag:

    MAP_SHARED → visibili, riportati
    MAP_PRIVATE → no
    MAP_ANONYMOUS → non crea il file mappato in memoria
  - fd → descrittore del file aperto. Se MAP_ANONYMOUS, fd = -1
  - offset → offset nel file, normalmente 0

- mmap ritorna l'indirizzo virtuale della regione condivisa

Shared Memory per allocare spazio per un processo

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/mman.h>
#define SIZE 5*sizeof(int)
//mmap-3.c
int main (void)
{   pid_t pid;
    float* ptr_shm;
    float* ptr_shm1;
    int i_leggo, i_scrivo;

    ptr_shm=mmap(0, SIZE, PROT_READ|PROT_WRITE,MAP_ANONYMOUS|MAP_SHARED, -1, 0);
    ptr_shm1=ptr_shm;

    for(i_scrivo=0; i_scrivo<SIZE; i_scrivo++) ptr_shm[i_scrivo]=i_scrivo;

    for(i_scrivo=0; i_scrivo<SIZE; i_scrivo++) printf("ho scritto:%4.2f ",*ptr_shm++);

    printf("\n\n");

    sleep(1);

    for(i_leggo=0; i_leggo<SIZE; i_leggo++) printf("leggo:%4.2f ",*ptr_shm1++);
    printf("\n");

    munmap(ptr_shm, 5*sizeof(int));

    return (0);
}
```

Shared Memory per interi

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/mman.h>
#define SIZE 5*sizeof(int)
//mmap-5.c
int main (void)
{   pid_t pid;
    int* ptr_shm;
    int* ptr_shm1;
    int* ptr_shm2;
    int i_leggo, i_scrivo;

    ptr_shm=mmap(0, SIZE, PROT_READ|PROT_WRITE,MAP_ANONYMOUS|MAP_SHARED, -1, 0);
    ptr_shm1=ptr_shm; ptr_shm2=ptr_shm;

    for(i_scrivo=0; i_scrivo<SIZE; i_scrivo++) *ptr_shm++=i_scrivo;

    for(i_scrivo=0; i_scrivo<SIZE; i_scrivo++) printf("ho scritto:%d ",*ptr_shm1++);
    printf("\n\n");

    sleep(1);

    for(i_leggo=0; i_leggo<SIZE; i_leggo++) printf("leggo:%d ",*ptr_shm2++);
    printf("\n");

    munmap(ptr_shm, 5*sizeof(int));

    return (0);
}
```

Shared Memory tra due processi (primitivo)

```
#include <stdio.h>   #include <stdlib.h>  #include <sys/types.h>    #include <unistd.h>
#include <sys/mman.h>
#define SIZE sizeof(int)
//mmap-2.c
int main (void)
{   pid_t pid;
    float* shared_memory, scritto, letto ;
    int i_letto, i_scritto;
    shared_memory=mmap(0, SIZE, PROT_READ|PROT_WRITE,MAP_ANONYMOUS|MAP_SHARED, -1, 0);
    pid = fork();
    if (pid==0){
        sleep(1);
        letto = *shared_memory;
        printf ("\tprocesso generato. Leggo = %f\n", letto);
        sleep(2);
        i_letto = *(int*)shared_memory;
        printf ("\tprocesso generato. Leggo = %d\n", i_letto);
    }
    else{
        scritto=20;
        *shared_memory = scritto ;
        printf ("Processo padre. Ho scritto = %f\n", scritto );
        sleep(2);
        i_scritto=47;    *(int*)shared_memory = i_scritto ;
        printf ("Processo padre. Ho scritto = %d\n", i_scritto );

        wait(pid);
    }
    return (0);
}
```

API per la Shared Memory con nome

- shm_open, shm_unlink

```
#include <sys/mman.h>
#include <sys/stat.h>        /* For mode constants */
#include <fcntl.h>           /* For O_* constants */

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

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

- oflag
  - O_RDONLY, O_RDWR, O_CREATE
  - O_EXCL → errore se oflag=O_CREATE e la regione esiste già
  - O_TRUNC → se la regione esiste, trona a 0 byte

- mode
  - shm_mode ritorna un file descriptor che duplica il contenuto della shared memory. È il file mode del file creato. Se no file, mode=0

- shm_unlink
  - Se tutti I processi che condividono la regione hanno eseguito unmap della regione, elimina la regione

Shared Memory con nome per allocare spazio per un processo

```
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/mman.h>
#include <sys/stat.h>        /* mnemonici di mode */
#include <fcntl.h>           /* mnemonici di oflag */
//mmap-8.c
int main (void)
{   int fd,i, i_scrivo, i_leggo,size=5*sizeof(float);
    float* ptr_shm;
    float* ptr_shm1;
    float* ptr_shm2;

    fd = shm_open("mymem",  O_CREAT | O_EXCL |O_RDWR, S_IRUSR | S_IWUSR);
    ftruncate(fd,size);
    ptr_shm = mmap(NULL ,size, PROT_READ | PROT_WRITE, MAP_SHARED ,fd, 0);

    ptr_shm1=ptr_shm;
    ptr_shm2=ptr_shm;
    for(i_scrivo=0; i_scrivo<size; i_scrivo++) ptr_shm[i_scrivo]=i_scrivo;

    for(i_scrivo=0; i_scrivo<size; i_scrivo++) printf("ho scritto:%4.2f ",*ptr_shm1++);
    printf("\n\n");
    sleep(1);

    for(i_leggo=0; i_leggo<size; i_leggo++) printf("leggo:%4.2f ",*ptr_shm2++);
    printf("\n");

    munmap(ptr_shm, size);
    shm_unlink("mymem");
    return(0);
}
```

Scrivere una stringa nella Shared Memory con nome (server)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>
//mmap-9-srv.c
int main (int argc, char *argv[]) {
    const char* shm_name  = "mymem";
    const int SIZE = 4096;
    const char * message[] = {"Questo ","è un ","esempio di ","shared ","memory","\n"};
    int i, fd;
    void * ptr;
    fd = shm_open(shm_name, O_CREAT | O_RDWR, 0666);
    if(fd==1){
        printf("errore in shm_open\n"); exit(1);
    }

    ftruncate(fd, sizeof(message));
    ptr = mmap(0, SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
    if (ptr == MAP_FAILED) {
        printf("errore in mmap\n"); exit(1);
    }
/* scrive la stringa in memoria */
    for (i = 0; i < strlen(*message); ++i) {
        sprintf(ptr, "%s", message[i]);     ptr += strlen(message[i]);
    }

    munmap(ptr, SIZE);
    return 0;
}
```

Leggere una stringa scritta nella Shared Memory con nome da un altro processo (client)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>
//mmap-9-cli.c
int main (int argc, char *argv[]) {
    const char * shm_name  = "mymem";
    const int SIZE = 4096;
    int i, fd;
    void * ptr;

    fd = shm_open(shm_name, O_RDONLY, 0666);
    if(fd==1){
        printf("Errore in shm_open\n");   exit(1);
    }
    ptr = mmap(0, SIZE, PROT_READ, MAP_SHARED, fd, 0);
    if (ptr == MAP_FAILED) {
        printf("Errore in mmap\n");     return(1);
    }

    printf("%s", (char*) ptr);

    if(shm_unlink(shm_name)==1){
        printf("Errore in unlink %s\n", shm_name); exit(1);
    }
    return 0;
}
```