Memory mapped files

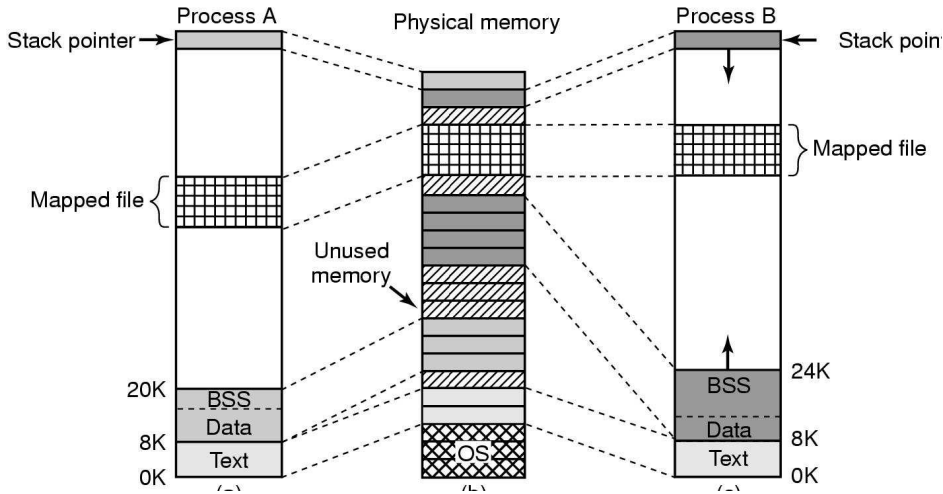
Permettono di "vedere" un file come un array di caratteri in memoria

Usi:

- usati internamente dal SO per caricare le librerie dinamiche
- nelle applicazioni, come alternativa a read(2)
  - è più veloce di lseek(2) + read(2)
  - soprattutto per file di grandi dimensioni
- per condividere dati fra processi

Il "backing store" per un mm file è nel file stesso

Memory mapped files



Memory mapped files

`void * mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset)`

*The **mmap** function asks to map length bytes starting at offset offset from the file specified by the file descriptor fd into memory, preferably at address start. This latter address is a hint only, and is usually specified as 0. The actual place where the object is mapped is returned by mmap, and is never 0.* — manuale di mmap(2)

Memory mapped files

`void * mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset)`

The *prot* argument describes the desired memory protection (and must not conflict with the open mode of the file). It is either PROT_NONE or is the bitwise OR of one or more of the other PROT_* flags.

- PROT_EXEC Pages may be executed.
- PROT_READ Pages may be read.
- PROT_WRITE Pages may be written.
- PROT_NONE Pages may not be accessed.

void * mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset)

The *flags* parameter:

MAP_FIXED Do not select a different address than the one specified. If the specified address cannot be used, mmap will fail. [...] Use of this option is discouraged.

MAP_SHARED Share this mapping with all other processes that map this object. Storing to the region is equivalent to writing to the file. The file may not actually be updated until msync(2) or munmap(2) are called.

MAP_PRIVATE Create a private copy-on-write mapping. Stores to the region do not affect the original file. It is unspecified whether changes made to the file after the mmap call are visible in the mapped region.

You must specify exactly one of MAP_SHARED and MAP_PRIVATE

Esempio: copia di un file con mmap(2) (a)

```
int main() {
  int n;
  int m;
  off_t filesize;
  struct stat info;
  char * input;

  /* otteniamo la dimensione del file di input con fstat(2) */
  /* nota: STDIN_FILENO è il filedescriptor dell'input (0) */
  if (fstat(STDIN_FILENO, &info) < 0) {
      perror(errore in fstat);
      exit(0);
  }
  filesize = info.st_size;

  /* Mappa il file di input in memoria; il puntatore risultante e'
     input
  mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset)
  */
  input = mmap(0, filesize, PROT_READ, MAP_PRIVATE, STDIN_FILENO, 0);
  if (MAP_FAILED == input) {
      perror(errore in mmap);
      exit(EXIT_FAILURE);
  }
```

Esempio: copia di un file con read(2)

```
#define BUF_SIZE 8192
char buf[BUF_SIZE];

int main() {
  int n;

  n = read(0, buf, BUF_SIZE);
  while (n > 0) {
    if (n != write(1, buf, n)) {
      perror(Errore in scrittura);
      exit(0);
    }
    n = read(0, buf, BUF_SIZE);
  }
  if (n < 0) {
    perror(errore in lettura);
  }
  exit(0);
}
```

Esempio: copia di un file con mmap(2) b)

```
/* copia l'input nell'output, a blocchi di BUF_SIZE caratteri.  */
  m = 0;
  while (m < filesize) {
    n = min(BUF_SIZE, filesize - m);
    if (n != write(1, input + m, n)) {
      perror(errore in scrittura);
      exit(EXIT_FAILURE);
    }
    m = m + n;
  }
  exit(0);
}
```

Rimuovere un MM file

```
int munmap(void *start, size_t length);
```

The munmap system call deletes the mappings for the specified address range, and causes further references to addresses within the range to generate invalid memory references. The region is also automatically unmapped when the process is terminated. On the other hand, closing the file descriptor does not unmap the region. — manuale di munmap(2)

Nota: posso mappare un file intero, e poi rimuovere un pezzo di mappa in mezzo!