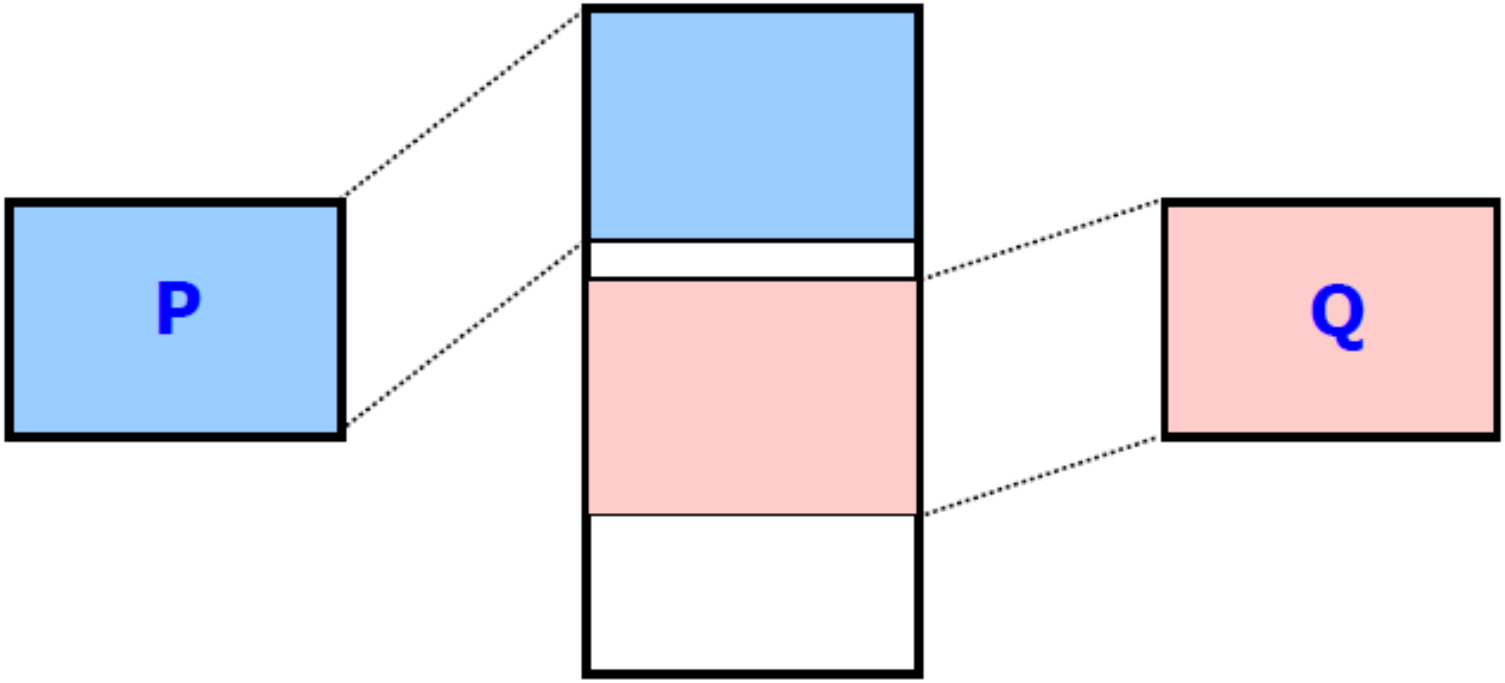
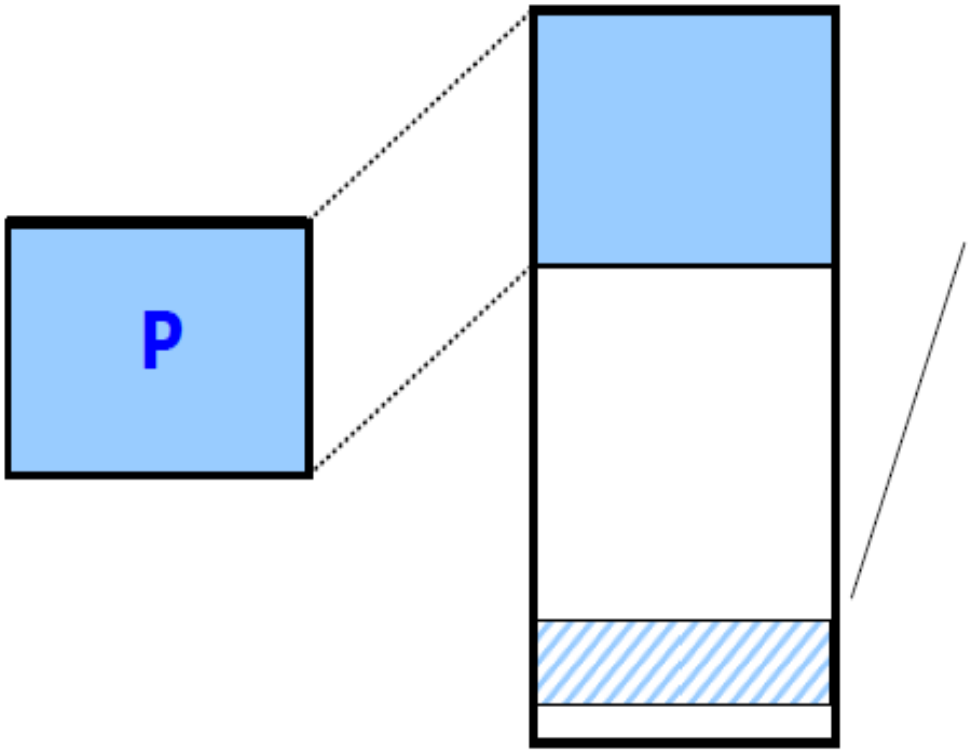
- Normalmente il meccanismo di gestione della memoria, qualunque esso sia, garantisce di allocare in RAM l'immagine di più processi in indirizzi non sovrapposti per evitare che uno danneggi l'altro per errore o per "malizia"
- Qui assumiamo, come meccanismo di base, uno semplice in cui l'immagine di ogni processo è allocata interamente in memoria, ed è allocata in modo contiguo, cioè in un intervallo di indirizzi della RAM:



- Nello standard POSIX si può creare un'area di memoria condivisibile fra processi con::

  shm_fd = shm_open(name, O_CREAT | O_RDWR, flags);

  ftruncate(shm_fd,size);

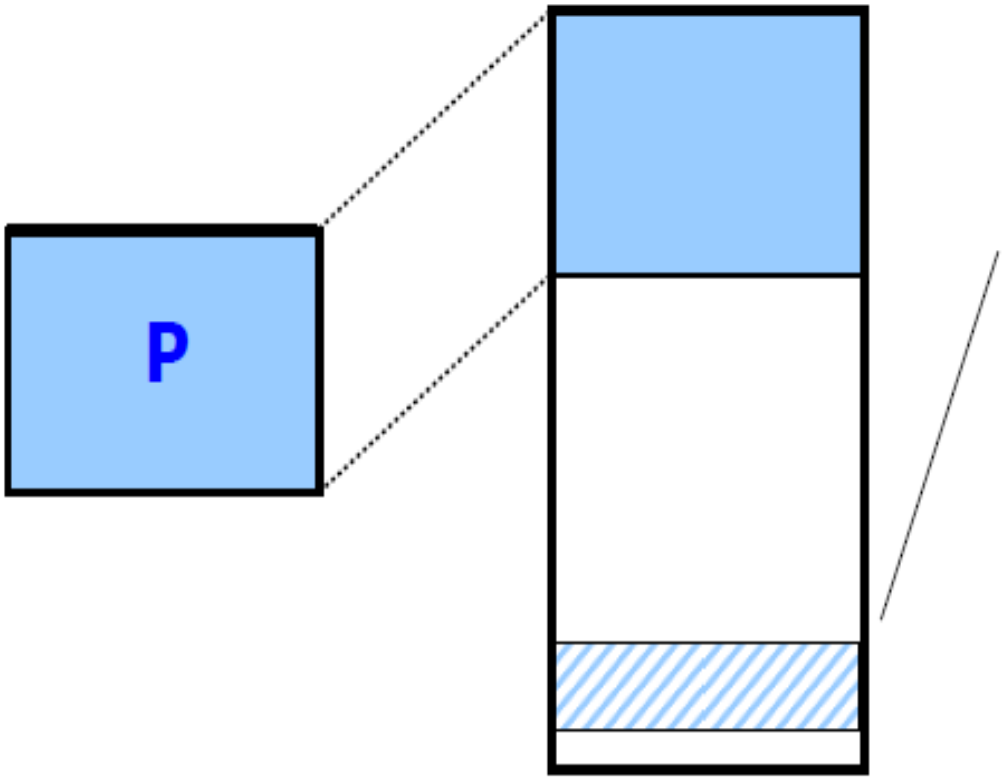


si ha uno «shared memory object»:
• identificato nel sistema da **name**, stringa che inizia con uno "/" e non ne contiene altri
• nel processo, identificato da **shm_fd** che è un file descriptor (!)
• usando **O_CREAT** viene creato se non esiste, se no solo «aperto»
• con diritti di accesso dati da **flags**
• di dimensione **size**

- il segmento può poi essere “collegato” allo spazio di indirizzamento di P con:

p = mmap(..., size, ..., MAP_SHARED, shm_fd, ...)

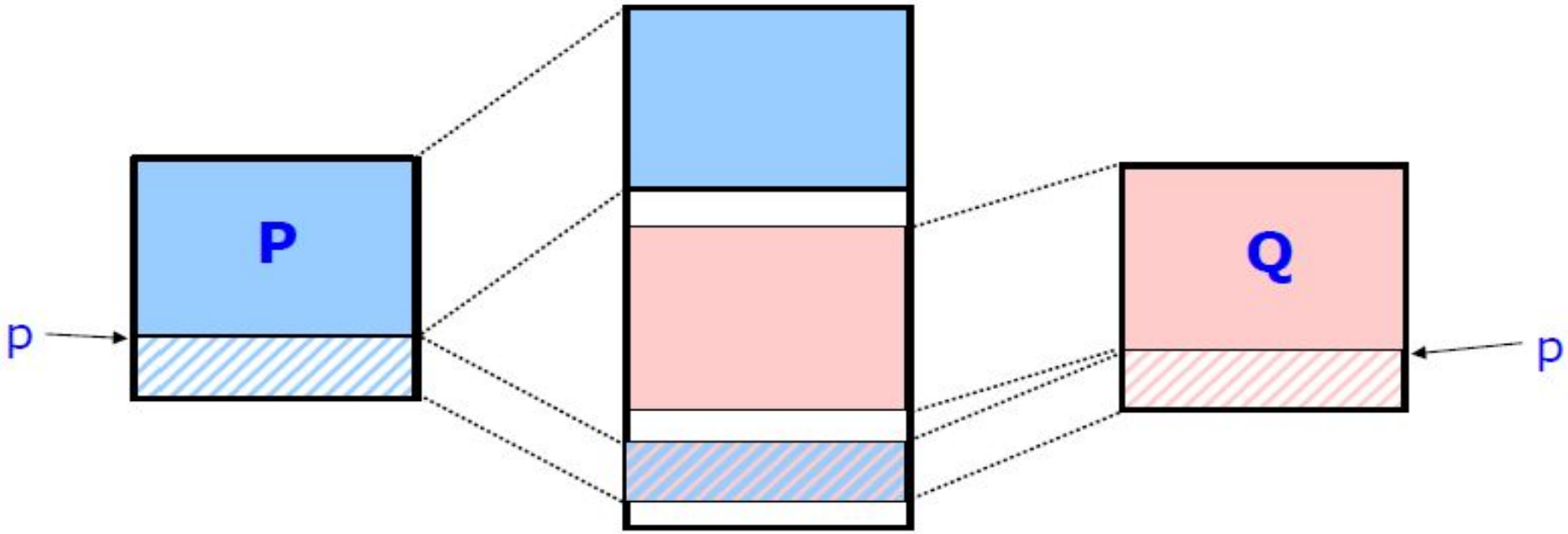
- mmap restituisce un puntatore, come malloc



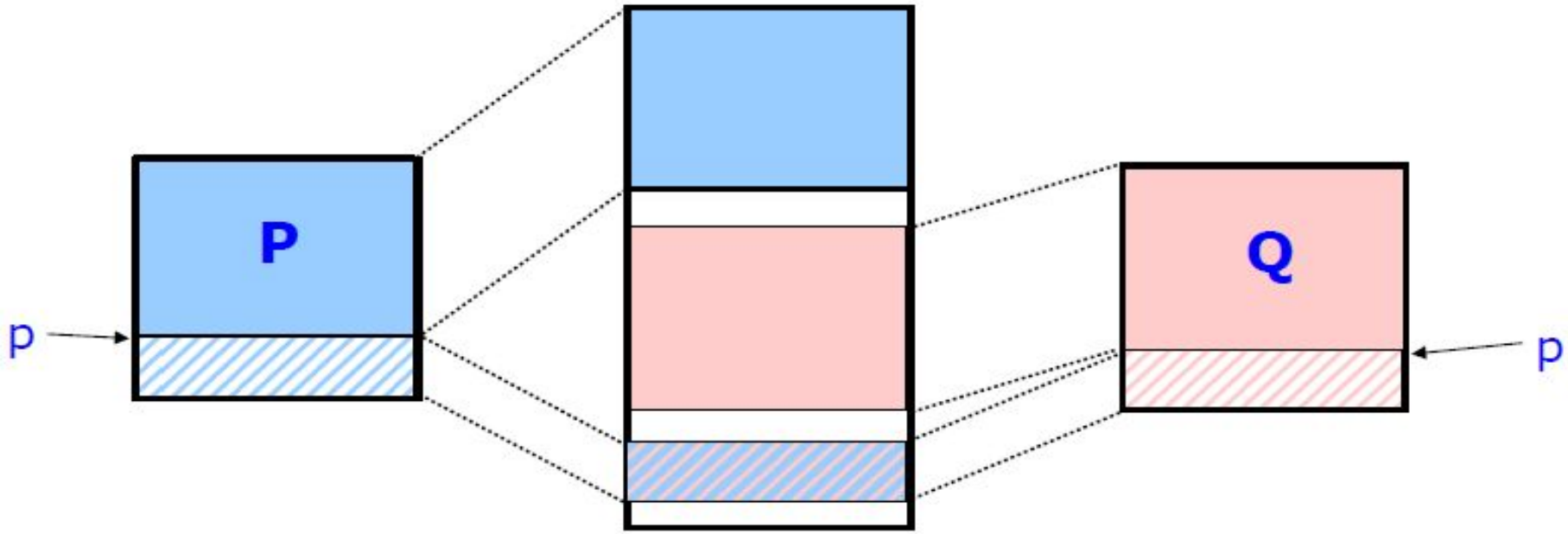
(**mmap** è nata per fare «I/O mappato in memoria» cioè leggere/scrivere su una porzione di file leggendo/scrivendo in memoria;la sua interfaccia viene riusata qui, visto che la memoria condivisa è identificata da un file descriptor)

creando nuovi processi con **fork()** dopo **mmap** abbiamo effettivamente memoria condivisa tra il processo e i suoi discendenti

processi non «parenti» in questo senso possono invece ripetere le operazioni usando lo stesso **name**(il primo che arriva «crea» la memoria condivisa, gli altri vi si collegano solo) o fare solo **mmap** se conosce **shm_fd**



Se le **mmap** sono fatte da ciascun processo, la memoria condivisa può essere «collegata» a indirizzi diversi in processi diversi



Esempio di utilizzo

```
int shm_fd1;

shm_fd1 = shm_open("/myshm1", O_CREAT|O_RDWR,0600);

ftruncate(shm_fd1,100*sizeof(int));

a1 = mmap(0 ,100*sizeof(int), PROT_READ|PROT_WRITE, MAP_SHARED,
shm_fd1,0);
```

Creazione, se non esiste

Permessi di lettura e scrittura al proprietario

Specifica dimensione in byte del segmento

L'indirizzo a cui attaccare il segmento viene scelto dal sistema. Scelta con maggiore portabilità

Specifica operazioni permesse

Specifica condivisione segmento con più processi

Variabili d'ambiente

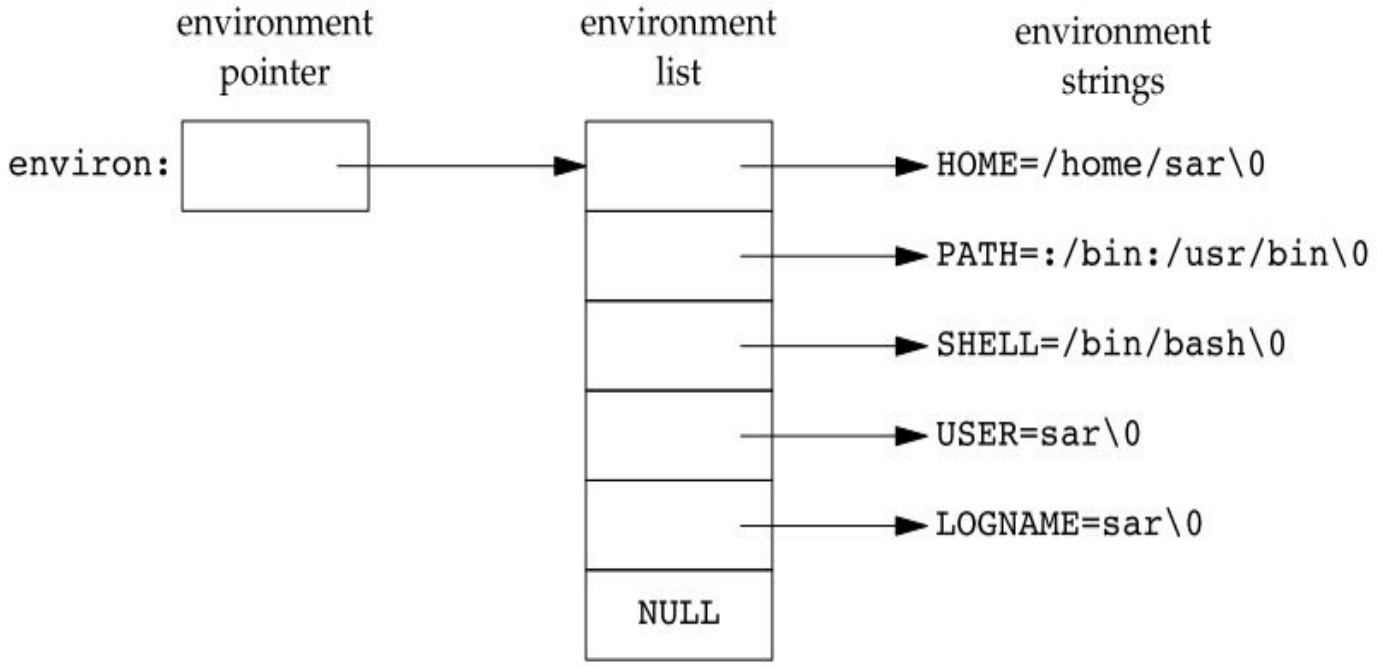
- Le variabili d'ambiente permettono di memorizzare informazioni utili per la configurazione dei programmi
  - Sono variabili specifiche per ogni processo
  - Sono accessibili tramite API fornite dal SO
  - Un processo *figlio* può ereditarne una copia dal processo *padre* che lo ha generato
- Nella shell possono essere
  - Elencate tramite il comando `env`
  - Definite con
    - `<NOME_VAR>=<VALORE>` - in tal caso sono definite solo per il processo corrente (la shell)
  - Esportate ai processi figli con
    - `export <NOME_VAR>`
  - Lette con
    - `$<NOME_VAR>`

Variabili d'ambiente

- Alcune variabili d'ambiente di sistema:
  - `HOSTNAME` il nome del computer
  - `HOME` la home directory dell'utente corrente
  - `PATH` il percorso di ricerca per i comandi
    - lista di directory separate da ; in cui la shell ricerca i comandi
  - `PWD` la directory attuale

Variabili d'ambiente

- Per operare sulle variabili d'ambiente in un programma C è sufficiente dichiarare:
  - *extern char **environ;*
  - La specifica *extern* indica al compilatore che la variabile è definita in altro file
  - Come per il passaggio di argomenti è il SO che si occupa di settare la variabile *environ* all'inizio del programma



Variabili d'ambiente: lettura e setting

- Per leggere le variabili d'ambiente:
  - *char *getenv(const char *name);*
  - Restituisce il puntatore al valore della variabile associata a *name, NULL* se non esiste;
- Per settare una variabile d'ambiente:
  - *int putenv(char *str);*
  - Prende una stringa di tipo *nome=valore* e l'aggiunge alla lista delle variabili d'ambiente