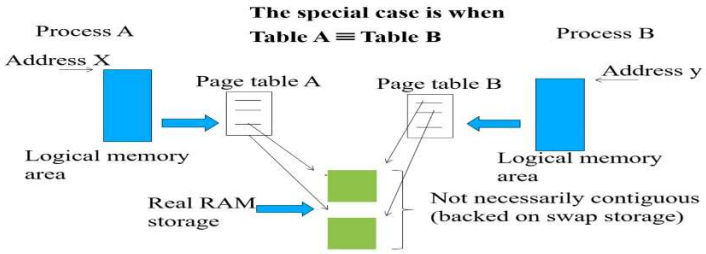
Memoria 9 Memoria condivisa UNIX

giovedì 20 novembre 2025 10:33

Vediamo ora la **memoria condivisa** : una tecnica che permette di utilizzare la memoria in modo condiviso (se qualcuno aggiorna altro legge queste info aggiornate). Di solito si ha nello scenario magari nel quale vi sono 2 thread della stessa applicazione (ricordiamoci mmap-ed e MAP_SHARED come in UNIX oppure virtual alloc WINDOWS più limitato). Vediamola in dettaglio :



In dettaglio :

int shmget(key_t key, int size, int flag)	
Descrizione	invoca la creazione di un'area di memoria condivisibile
Parametri	1) key: chiave per identificare la memoria condivisibile in maniera univoca nel sistema (IPC_PRIVATE e' un caso speciale) 2) size: taglia in byte della memoria condivisibile 3) flag: specifica della modalita' di creazione (IPC_CREAT, IPC_EXCL, definiti negli header file sys/ipc.h e sys/shm.h) e dei permessi di accesso
Descrizione	identificatore numerico per la memoria conivisa in caso di successo (descrittore), -1 in caso di fallimento

NOTA
Il descrittore indicizza questa volta una struttura unica valida per qualsiasi processo

La quale struttura viene chiamata **inter proccess comunication table** . Appena creata non è raggiungibile , in quanto non è istanziata. Per eliminarla invece si usa :

int shmctl(int ds_shm, int cmd, struct shmid_ds *buff)	
Descrizione	invoca l'esecuzione di un comando su una shared memory
Parametri	1) ds_shm: descrittore della memoria condivisa su cui si vuole operare 2) cmd: specifica del comando da eseguire (IPC_RMID, IPC_STAT, IPC_SET) 3) buff: puntatore al buffer con eventuali parametri per il comando
Descrizione	-1 in caso di fallimento

IPC_RMID invoca la rimozione della memoria condivisa dal sistema

Vediamo ora la struttura shmid_ds la quale viene usata in modo opportune se il flag è IPC_SET:

```
struct shmid_ds {                                    ↑
        struct   ipc_perm shm_perm;  /* operation perms */
        int  shm_segsz;          /* size of segment (bytes) */
        time_t   shm_atime;          /* last attach time */
        time_t   shm_dtime;          /* last detach time */
        time_t   shm_ctime;          /* last change time */
        unsigned short shm_cpid; /* pid of creator */
        unsigned short shm_lpid; /* pid of last operator */
        short    shm_nattch;         /* no. of current attaches */
};

struct ipc_perm
{
 key_t  key;
 ushort uid;   /* owner euid and egid */
 ushort gid;
 ushort cuid;  /* creator euid and egid */
 ushort cgid;
 ushort mode;  /* lower 9 bits of shmflg */
 ushort seq;   /* sequence number */
};
```

L'ultimo parametri rappresenta il numero degli attach correnti (usato per IPC_STAT): il numero di istanze della mappatura di pagine logiche su pagine fisiche tramite page table. **Per rimuovere una shared memory il numero degli attach deve essere zero.** Per eseguire un attach , quindi poter usare effettivamente la memoria condivisa si usa questa API:

void *shmat(int ds_shm, void *addr, int flag)	
Descrizione	invoca il collegamento di una memoria condivisa allo spazio di indirizzamento del processo
Parametri	1) ds_shm: descrittore della memoria condivisa da collegare 2) *addr: indirizzo preferenziale per il collegamento (NULL come default) 3) flag: modalita' di accesso (SHM_R, SHM_W, SHM_RW)
Descrizione	indirizzo valido per l'accesso alla memoria in caso di successo, -1 in caso di fallimento

int shmdet(const void *addr)	
Descrizione	invoca lo scollegamento di una memoria condivisa dallo spazio di indirizzamento del processo
Parametri	*addr: indirizzo della memoria da scollegare
Descrizione	-1 in caso di fallimento

Dove la seconda API permette di un-mappare la pagina dalla memoria condivisa (elimino collegamento tra pagine logiche ed info contenute nella shared memory). **Non si eliminano le pagine cosi**. Vediamo un esempio :

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>

#define PAGE (4096)
#define NPAGES (10)
#define SIZE PAGE*NPAGES

#define DISP 128
#define OK DISP-1

char messaggio[DISP];


void produttore(int id) {

   char *addr, *addr1;
   int ret;
   addr = shmat(id, 0 , SHM_W);
   if (addr == (void*)-1 ){
        printf("shmat error \n");
        exit(2);
   }
   addr1 = addr;
   printf("insert strings to write to shared memory ('quit' to close): \n");
   do{
     scanf("%127s", messaggio);
     memcpy(addr1, messaggio, DISP);
     addr1 = addr1 + DISP;
   }
   while( (strcmp(messaggio,"quit") != 0) && ((addr1 - addr) < (SIZE - DISP)));
   printf("shared memory update done\n");
   exit(0);
}
```

Si esegue la shared memory attach e ci scriviamo dentro .Nel ciclo (al più 127 caratteri), acquisisco stringhe e la copio nella pagina della shared memory.

```c
void consumatore(int id) {

   char *addr, *addr1;
   int ret;
   struct sembuf oper;

   addr = shmat(id, 0 , SHM_R);
   addr1 = addr;


   printf("shared memory keeps the following strings: \n");

    while( (strcmp(addr1,"quit") != 0) && ((addr1 - addr) < (SIZE - DISP))) {
         printf("%s \n", addr1);
         addr1 = addr1 + DISP;
    }
   exit(0);

}
```

Anche qui eseguo memory attach e nel ciclo vado a leggerele stringhe

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

   long id_shm, id_sem;
   int  ret, STATUS;
   long key = 16;
// long key = IPC_PRIVATE;//try with different configuraitons for the key
   void *point;


   id_shm = shmget(key, SIZE, IPC_CREAT|0666);

   if (id_shm == -1){
        printf("shmget error\n");
        return -1;

   }

   if(argv[1] &&(!strcmp(argv[1],"read"))) consumatore(id_shm);


   if(fork()) {
        wait(&STATUS);
   }
   else {
       produttore(id_shm);
   }

   if(fork()) {
        wait(&STATUS);
   }
   else {
       consumatore(id_shm);
   }

//   ret = shmctl(id_shm, IPC_RMID, point);

}/* end main */
```

Nel main viene creata la shared memory con u n codice , la quale se già esistente viene ritornato il descrittore. L'ultima istruzione commentata permette di elimnare la shared memory

Se lancio da terminale **ipcs** viene restituito un elenco di shared memory la quale chiave se è 0x00000000 significa che è privata IPC_PRIVATE).