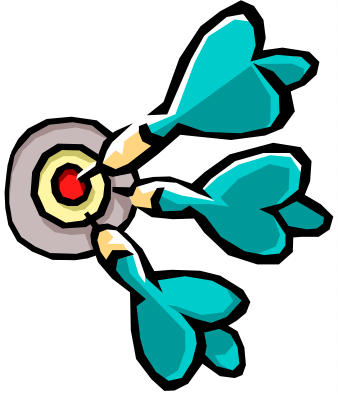
IPC - Linux

Programmazione di Sistema

A.A. 2016-17

Argomenti



- Identificativi
- Message Queue
- Pipe
- Shared Memory
- Memory Mapped File
- Semafori

Identificativi

- Ciascuna struttura IPC è identificata nel S.O. da un intero non negativo
  - All'atto della creazione di un oggetto IPC, si fornisce una chiave
  - Il S.O. converte questa chiave nell'ID associato
- Un processo può creare una nuova struttura IPC con chiave diversa da O
  - Tutti i processi che conoscono la chiave possono ottenere l'ID corrispondente

Condivisione della chiave

- Se, all'atto della creazione, come chiave viene indicato IPC_PRIVATE (0)
  - Non sarà possibile ad altri ottenere la corrispondenza con l'ID
  - L'ID generato dovrà essere condiviso in altri modi
- Un processo può creare una nuova struttura IPC specificando come chiave un valore predefinito
  - Gli altri processi devono conoscere questa chiave
  - Uso di file header

Message Queues

- I processi che intendono comunicare si accordano
  - Sul pathname di un file esistente
  - E su un project-ID (O-255)
- Tramite ftok() convertono questi valori in una chiave univoca
- L'ultimo processo che fa accesso ad una struttura dati di IPC deve occuparsi della rimozione della stessa
  - Altrimenti continua ad essere un oggetto kernel valido…

Message Queues

- Permettono lo scambio di messaggi tra processi
- I messaggi sono composti da un tipo e da un payload
  - Il processo che riceve può specificare il tipo dei messaggi a cui è interessato
- I messaggi sono puntatori a strutture

```
struct message {
  long type ;
  char messagetext
[ MESSAGESIZE ];
};
```


Creazione/accesso

```
int msgget(key_t key, int msgflg)
```

- Crea una nuova message queue
- Ottiene l'id della message queue associata alla chiave specificata

Invio di un messaggio

```
int msgsnd(int msqid,
           const void *msgp,
           size_t msgsz,
           int msgflg)
```

- msqid: id della coda
- msgp: puntatore al messaggio
- Il mittente deve avere il permesso di scrittura sulla coda

Lettura di un messaggio

```
ssize_t msgrcv(int msqid, void *msgp,
                    size_t msgsz,
                    long msgtyp, int
msgflg)
```

- Copia nel buffer msgp un messaggio della coda identificata da msqid
- mesgsz indica la dimensione del corpo del messaggio
- msgtyp indica il messaggio di interesse
  - 0 viene restituito il primo messaggio della coda
  - >0 viene restituito il primo messaggio del tipo indicato
  - <0 il messaggio con il più basso valore del campo tipo

Controllare una coda

```
int msgctl(int msqid, int cmd,
           struct msqid_ds *buf)
```

- Esegue l'operazione di controllo specificata da cmd sulla coda msqid
- Con comando IPC_RMID rilascia le risorse

Pipe

- Permettono la IPC tra processi padre-figlio
  - Creati con la funzione pipe()

```
int pipe_fds[2];
int read_fd, write_fd;
pipe (pipe_fds);
read_fd = pipe_fds[0];
write_fd = pipe_fds[1];
```

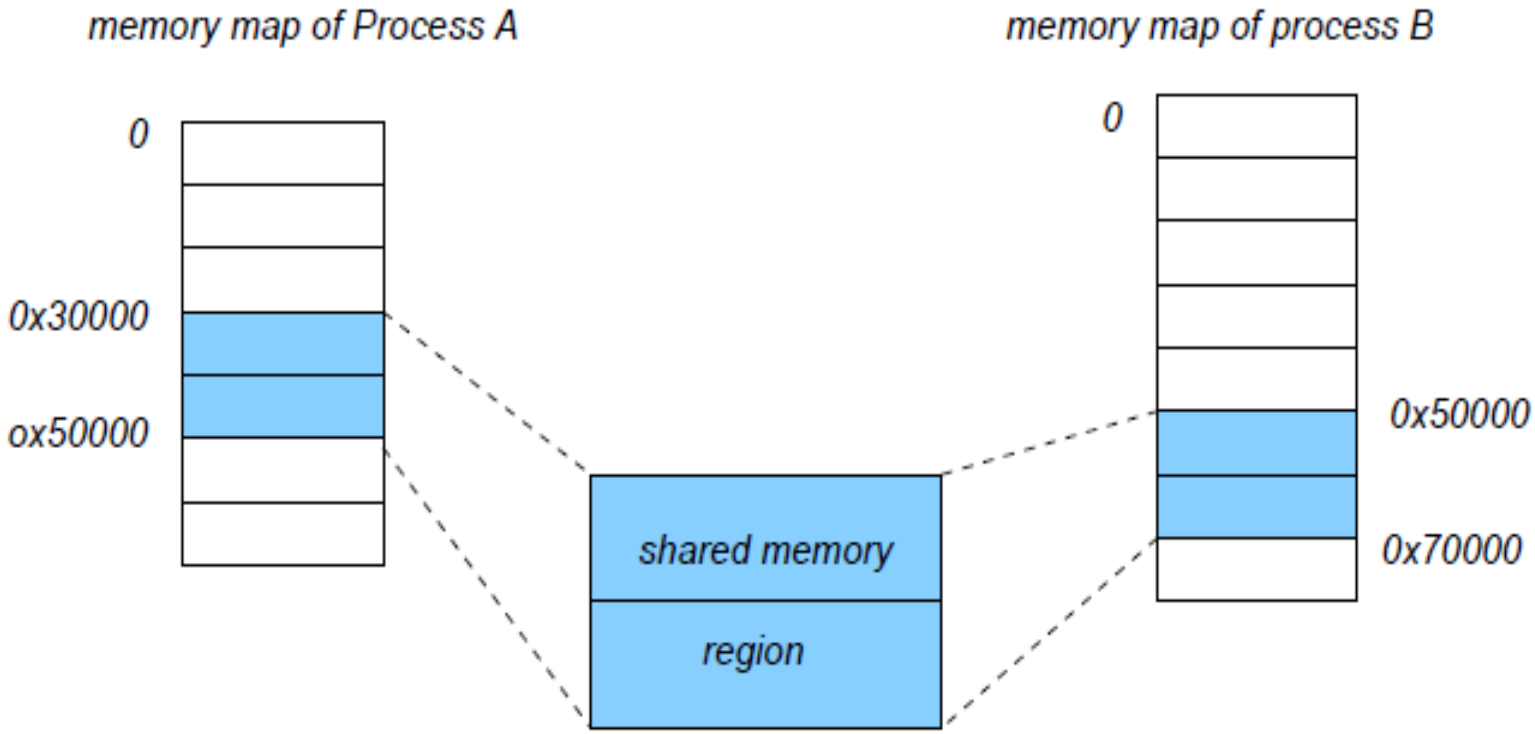

Esempio

```
int fds[2];
pid_t pid;
pipe (fds);
pid = fork ();
if (pid == (pid_t) 0) {
  /* processo figlio */
  close (fds[1]);
  // ... Lettura dalla pipe
  close (fds[0]);
}else {  /* processo padre*/
  close (fds[0]);
  // ... Scrittura sulla pipe
  close (fds[1]);
}
```

Named pipe: FIFO

- Permette la comunicazione tra processi generici
- Create con la funzione int mkfifo(const char *path, mode_t mode);
- Si accede come ad un file «normale»

Shared Memory



Creazione/accesso

```
int shmget(key_t key,
           size_t size,
           int shmflg)
```

- Restituisce l'id del segmento condiviso associato alla chiave key
- Eventualmente alloca il segmento in base ai parametri shmflag
- La dimensione restituita da shmget è uguale alla dimensione effettiva del segmento, arrotondata ad un multiplo di PAGE_SIZE

Operazioni di controllo

```
int shmctl(int shmid, int cmd,
           struct shmid_ds *buf)
```

- Ottenere informazioni sul segmento
- Impostare i permessi, il proprietario e il gruppo
- Rilasciare le risorse associate

Possibili operazioni

- IPC_STAT
  - Copia le informazioni dalla struttura dati del kernel associata alla memoria condivisa all'interno della struttura puntata da buf
- IPC_SET
  - Scrive i valori contenuti nella struttura dati puntata da buf nell'oggetto kernel corrispondente alla memoria condivisa specificata
- IPC_RMID
  - Marca il segmento come da rimuovere. Il segmento viene rimosso dopo che dell'ultimo processo ha effettuato il detach

Connessione e

```
void *shmat(int shmid,
            const void *shmaddr,
            int shmflg)
```

- Connette il segmento identificato da shmid allo spazio di indirizzamento del processo chiamante

```
int shmdt(const void *shmaddr)
```

- Disconnette il segmento specificato dallo spazio di indirizzamento del chiamante

Memory mapped file

- Porzioni di file mappate in memoria
  - Per condividere il contenuto del file tra processi in lettura/scrittura
  - Semplificano operazioni tipo fseek()

```
void* mmap ( void* start, size_t n,

             int prot, int flags,
             int fd, off_t off);
```

  - Mappa n byte
  - Del file specificato da fd
  - A partire dall'offset off
  - Preferibilmente all'indirizzo start

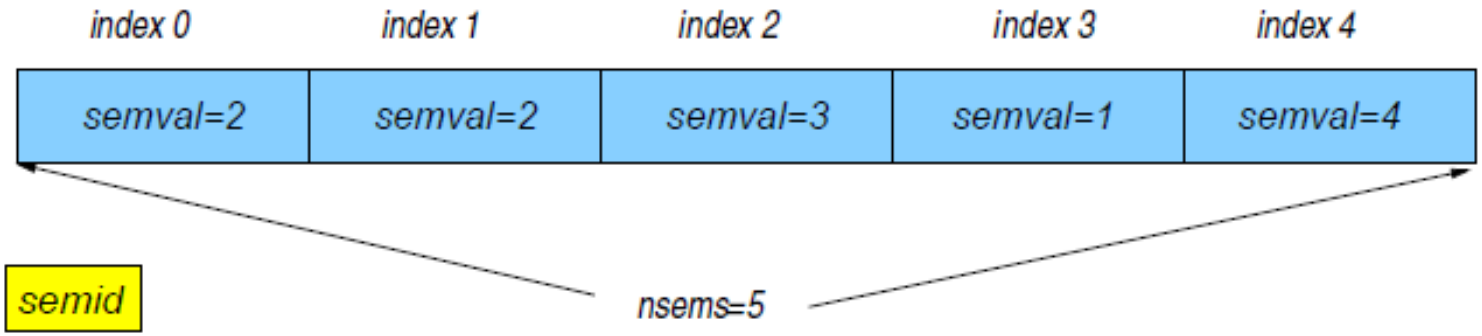
Rilascio

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

- Rilascia il mapping specificato dai parametri
- Invalida gli indirizzi corrispondenti
- La regione viene rilasciata automaticamente quando il processo termina
  - La chiusura del file descriptor non causa il rilascio della regione

Semafori System-V

- I semafori System-V vengono utilizzati per la sincronizzazione di thread di processi differenti
  - Contrariamente all'utilizzo dei semafori posix
- Sono costituiti da un array di contatori
  - Se è necessario proteggere più risorse, si



Creazione/accesso

- int semget(key_t key, int nsems, int semflg)
  - Restituisce l'id dell'insieme di semafori associati alla chiave

```
int sid = semget( mySemKey, 1,
                  IPC_CREAT|0700
)
```


Operare su un semaforo

- int semop(int semid, struct sembuf *sops, unsigned n)
  - semid: id del semaforo
  - sops: operazioni da compiere
  - n: numero elementi del semaforo

Struttura sembuf

```
struct sembuf {
  unsigned short sem_num ;
  //indice del semaforo su cui
operare

  short sem_op ;
  // operazione da effettuare
  // >0 per acquisire
  // <0 per rilasciare

  sem_flg ;
  // solitamente 0
};
```


Esempio

```
struct sembuf  sem_lock;

sem_lock.sem_num =  0;
sem_lock.sem_op    = -1;
sem_lock.sem_flg    =  0;

if (semop(sid, &sem_lock, 1) == -1) {
   perror("semop ");
   exit(-1);
}
```

Operazioni di controllo

- int semctl(int semid, int i, int cmd, [union semun arg])
  - Effettua l'operazione cmd sull'i-esimo semaforo del set identificato da semid

Union semun

```
union semun {
  int val ;
  /* Value for SETVAL */
  struct semid_ds * buf ;
  /* Buffer for IPC_STAT , IPC_SET */
unsigned short * array ;
  /* Array for GETALL , SETALL */
struct seminfo * __buf ;
  /* Buffer for IPC_INFO */
};
```


Struttura di semid_ds

```
struct semid_ds {
   struct ipc_perm sem_perm ;
     /* Ownership and permissions */
time_t sem_otime ;
     /* Last semop time */
time_t sem_ctime ;
     /* Last change time */
unsigned short sem_nsems ;
     /* No . of semaphores in set */
};
```


Possibili operazioni

- IPC_STAT
  - Copia le informazioni dalla struttura dati del kernel associata al semaforo all'interno della struttura semid_ds puntata da arg.buf
- IPC_SET
  - Scrive i valori contenuti nella struttura dati semid_ds puntata da arg.buf nell'oggetto kernel corrispondente all'insieme di semafori
  - Aggiorna sem_ctime
- IPC_SETALL
  - Imposta semval per tutti i semafori del set utilizzando arg.array
  - Aggiorna sem_ctime
- IPC_GETALL
  - Restituisce i semval correnti di tutti i semafori

Rimozione di un semaforo

- Per rilasciare le risorse relative ad un semaforo si utilizza la funzione semctl con parametro IPC_RMID

```
semctl( sid, 0, IPC_RMID, 0 );
```


Spunti di riflessione

- Si confrontino i meccanismi di comunicazione tra processi offerti da Linux con quelli di Windows e si provi a compilare una scaletta riassuntiva che mostri similitudini e differenze