ESD Electronic Systems Design
Synthesis Verification Testing Power Communication

Programmazione di sistema in Unix: IPC

N. Drago, G. Di Guglielmo, L. Di Guglielmo,
V. Guarnieri, M. Lora, G. Pravadelli, F. Stefanni

System call per la Comunicazione tra Processi (IPC)

System call per Meccanismi di IPC Avanzati

- Code di Messaggi
- Memoria condivisa
- Semafori

System call per la Comunicazione tra Processi (IPC)

System call per Meccanismi di IPC Avanzati

- Code di Messaggi
- Memoria condivisa
- Semafori

Introduzione

- UNIX e IPC
- Pipe
- FIFO (*named pipe*)
- Code di messaggi (*message queue*)
- Memoria condivisa (*shared memory*)
- Semafori

UNIX e IPC

- `ipcs`: riporta lo stato di tutte le risorse, o selettivamente, con le seguenti opzioni:
  - `-s` informazioni sui semafori;
  - `-m` informazioni sulla memoria condivisa;
  - `-q` informazioni sulle code di messaggi.
- `ipcrm`: elimina le risorse (se permesso) dal sistema.
  - Nel caso di terminazioni anomale, le risorse possono rimanere allocate
  - Le opzioni sono quelle `ipcs`
  - Va specificato un ID di risorsa, come ritornato da `ipcs`

UNIX e IPC - (cont.)

- Esempio:

```
host:user> ipcs
IPC status from /dev/kmem as of Wed Oct 16 12:32:13 1996
Message Queues:
T     ID     KEY        MODE       OWNER    GROUP
*** No message queues are currently defined ***

Shared Memory
T     ID     KEY        MODE       OWNER    GROUP
m    1300        0 D-rw-------      root    system
m    1301        0 D-rw-------      root    system
m    1302        0 D-rw-------      root    system

Semaphores
T     ID     KEY        MODE       OWNER    GROUP
*** No semaphores are currently defined ***
```

Pipe

- Il modo più semplice di stabilire un canale di comunicazione *unidirezionale e sequenziale* **in memoria** tra due processi consiste nel creare una *pipe*:

```
1 #include <unistd.h>
2
3 int pipe (int fildes[2])
```

- La chiamata ritorna zero in caso di successo, `-1` in caso di errore.
- Il primo descrittore (`[0]`) viene usato per leggere, il secondo `[1]` per scrivere.
- NOTA: L'interfaccia è quella dei file, quindi sono applicabili le system call che utilizzano file descriptor.

Pipe - Esempio - part. 1

```
1  /*************************************
2  MODULO: pipe.c
3  SCOPO: esempio di IPC mediante pipe
4  *************************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <unistd.h>
8  #include <sys/wait.h>
9
10 int main (int argc, char *argv[]) {
11   int status, p[2];
12   char buf[64];
13
14   pipe (p);
15   if ((status=fork()) == -1)   /* errore */
16     syserr (argv[0], "fork() fallita");
```

Pipe - Esempio - part. 2

```
1    else if (status == 0) { /* figlio */
2      close (p[1]);
3      if (read(p[0],buf,BUFSIZ) == -1)
4        syserr (argv[0], "read() fallita");
5      printf ("Figlio - ricevuto: %s\n", buf);
6      exit(0);
7    } else  {  /* padre */
8      close(p[0]);
9      printf("Padre - invio nella pipe:");
10     printf("In bocca al lupo\n");
11     write(p[1], "In bocca al lupo", 17);
12     wait(&status);
13     exit(0);
14   }
15 }
```

Pipe e I/O

- Non è previsto l'accesso random (no `lseek`).
- La dimensione fisica delle pipe è limitata (dipendente dal sistema – BSD classico = 4K).
- La dimensione è definita in `PIPE_BUF`.
- L'operazione di `write` su una pipe è *atomica*.
- La scrittura di un numero di Byte superiore a questo numero:
  - Blocca il processo scrivente fino a che non si libera spazio
  - la `write` viene eseguita a "pezzi", con risultati non prevedibili (es. più processi che scrivono)
- La `read` si blocca su pipe vuota e si sblocca non appena un Byte è disponibile (anche se ci sono meno dati di quelli attesi!)
- Chiusura prematura di un estremo della pipe:
  - Scrittura: le `read` ritornano 0.
  - Lettura: i processi in scrittura ricevono il segnale `SIGPIPE` (broken pipe).

Pipe e comandi - Esempio - part. 1

```
1  /*************************************************
2  MODULO: pipe2.c
3  SCOPO: Realizzare il comando "ps | sort"
4  *************************************************/
5  #include <sys/types.h>
6  #include <stdlib.h>
7  #include <unistd.h>
8
9  int main ()
10 {
11         pid_t pid;
12         int pipefd[2];
```

Pipe e comandi - Esempio - part. 2

```
1         pipe (pipefd);
2         if ((pid = fork()) == 0) { /* figlio */
3               close(1);       /* close stdout */
4               dup (pipefd[1]);
5               close (pipefd[0]);
6               execlp ("ps", "ps", NULL);
7         }
8         else if (pid > 0) { /* padre */
9               close(0); /* close stdin  */
10              dup (pipefd[0]);
11              close (pipefd[1]);
12              execlp ("sort", "sort", NULL);
13        }
14        return 0;
15  }
```

Pipe e I/O non bloccante

- E' possibile forzare il comportamento di `write` e `read` rimuovendo la limitazione del bloccaggio.
- Realizzato tipicamente con `fcntl` per impostare la flag `O_NONBLOCK` sul corrispondente file descriptor (0 o 1)

```
1 int fd[2];
2 fcntl(fd[0], F_SETFL, O_NONBLOCK);
```

- Utile per implementare meccanismi di polling su pipe.
  - Se la flag `O_NONBLOCK` è impostata, una `write` su una pipe piena ritorna subito 0, e una `read` su una pipe vuota ritorna subito 0.

Pipe - (cont.)

- Limitazioni
  - possono essere stabilite soltanto tra processi *imparentati* (es., un processo ed un suo "progenitore", o tra due discendenti di un unico processo)
  - Non sono permanenti e sono distrutte quando il processo che le crea termina
- Soluzione: assegnare un nome *unico* alla pipe: *named pipe* dette anche FIFO.
- Funzionamento identico, ma il riferimento avviene attraverso il nome anziché attraverso il file descriptor.
- Esistono fisicamente su disco e devono essere rimossi esplicitamente con `unlink`

Named Pipe (FIFO)

```
1 int mknod(const char *pathname,mode_t mode,dev_t dev);
2 // Esempio:
3 char* name = "my_fifo";
4 int result = mknod (name, S_IFIFO |
5                                 S_IRUSR | S_IWUSR, 0);
```

- Si creano con `mknod()`. L'argomento `dev` è ignorato.
- Valore di ritorno: 0 in caso di successo, -1 in caso di errore.
- Apertura, lettura/scrittura, chiusura avvengono come per un normale file. Quindi sono ereditate dai processi figlio.
- Possono essere usate da processi non in relazione, in quanto il nome del file è unico nel sistema.
- Le operazioni di I/O su FIFO sono atomiche.
- I/O normalmente bloccante, ma è possibile aprire (con `open` e flag `O_NONBLOCK`) un FIFO in modo non bloccante. In tal modo sia `read` che `write` saranno non bloccanti.

Named Pipe (FIFO) - (cont.)

```
1 #include <sys/types.h>
2 #include <sys/stat.h>
3 int mkfifo(const char *pathname, mode_t mode);
```

- A livello di libreria (non system call) esiste anche la funzione `mkfifo()`.
- Il valore di ritorno e i parametri sono gli stessi di `mknod()`.

Named Pipe (FIFO) - Example - part. 1

```
1  /*************************************
2  MODULO: fifo.c
3  SCOPO: esempio di IPC mediante named pipe
4
5  USO: Lanciare due copie del programma su due
6  shell separate, una con flag -0 e
7  l'altra con flag -1.
8  Lanciare prima quella con flag -1 che crea
9  la fifo.
10 La copia del programma con flag -0 leggera'
11 quanto scritto dalla copia con flag -1.
12 *************************************/
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <fcntl.h>
16 #include <sys/types.h>
17 #include <sys/stat.h>
18 #include <unistd.h>
```

Named Pipe (FIFO) - Example - part. 2

```
1  int main (int argc, char *argv[]) {
2    int i,fd;
3    char buf[64];
4
5    if (argc != 2) {
6      printf("Usage: fifo.x -[0|1]\n\t -0 to");
7      printf(" read\n\t -1 to write\n");
8      exit(1);
9    } else if (strncmp(argv[1], "-1", 2)==0){
10     if (mknod("fifo",S_IFIFO|0777,0)== -1) {
11       perror("mknod");
12       exit(1);
13     }
14     if ((fd = open("fifo",O_WRONLY))==-1){
15       perror("FIFO: -1");
16       unlink("fifo");
17       exit(1);
18     }
```

Named Pipe (FIFO) - Example - part. 3

```
1   } else if (strncmp(argv[1], "-O", 2)==0){
2     if ((fd = open("fifo",O_RDONLY))==-1){
3       perror("FIFO: -1");
4       unlink("fifo");
5       exit(1);
6     }
7   } else {
8     printf("Wrong parameter: %s\n", argv[1]);
9     unlink("fifo");
10    exit(1);
11  }
12  for (i=0;i<20;i++) {
13    if (strncmp(argv[1], "-1", 2)==0){
14      write(fd,"HELLO",6);
15      printf("Written HELLO %d \n", i);
16    } else {
17      read(fd,buf,6);
18      printf("Read %s %d\n",buf,i);}}}
```

System call per la Comunicazione tra Processi (IPC)

System call per Meccanismi di IPC Avanzati

- Code di Messaggi
- Memoria condivisa
- Semafori

Meccanismi di IPC Avanzati

- IPC SystemV:
  - Code di messaggi
  - Memoria condivisa
  - Semafori
- Caratteristiche comuni:
  - Una primitiva "`get`" per:
    - creare una nuova entry,
    - recuperare una entry esistente
  - Una primitiva "`ctl`" (control) per:
    - verificare lo stato di una entry,
    - cambiare lo stato di una entry,
    - rimuovere una entry.
  - Sono visibili e manipolabili tramite i comandi bash `ipcs` (anche per controllare le chiavi) e `ipcrm` (in caso di crash del programma).

Meccanismi di IPC Avanzati - (cont.)

- La primitiva "get" richiede la specifica di due informazioni:
  - Una *chiave*, usata per la creazione/recupero dell'oggetto di sistema
    - Valore intero arbitrario;
  - Dei *flag* di utilizzo:
    - Permessi relativi all'accesso (tipo `rwxrwxrwx`)
    - `IPC_CREAT`: si crea una nuova entry se la chiave non esiste
    - `IPC_CREAT` + `IPC_EXCL`: si crea una nuova entry ad uso esclusivo da parte del processo

Meccanismi di IPC Avanzati - (cont.)

- L'identificatore ritornato dalla "get" (se diverso da -1) è un descrittore utilizzabile dalle altre system call
- La creazione di un oggetto IPC causa anche l'inizializzazione di:
  - una struttura dati, che varia a seconda dei tipi di oggetto, contenente informazioni su
    - UID, GID
    - PID dell'ultimo processo che l'ha modificata
    - Tempi dell'ultimo accesso o modifica
  - una struttura di permessi `ipc_perm`:

```
struct ipc_perm {
    key_t key; /* Key. */
    uid_t uid; /* Owner 's user ID. */
    gid_t gid; /* Owner 's group ID. */
    uid_t cuid ; /* Creator 's user ID. */
    gid_t cgid ; /* Creator 's group ID. */
    unsigned short int mode ; /* R/W perm. */
}
```

Meccanismi di IPC Avanzati - (cont.)

- La primitiva "`ctl`" richiede la specifica di informazioni diverse in base all'oggetto di sistema
- In tutti i casi, la primitiva "`ctl`" richiede:
  - Un *descrittore*, usato per accedere all'oggetto di sistema
    - Valore intero ritornato dalla primitiva "`get`";
  - Dei *comandi* di utilizzo:
    - Cancellare
    - Modificare
    - Leggere informazioni relative agli oggetti

ftok()

```
1 #include <sys/types.h>
2 #include <sys/ipc.h>
3
4 key_t ftok(const char *pathname, int proj_id);
```

- `ftok()` è usata per ottenere chiavi probabilmente non in uso nel sistema.
- Utile per evitare possibili conflitti con altri processi. È migliore usare `ftok()` rispetto a delle costanti esplicite.
- I parametri sono un path ad un file esistente ed accessibile, e una costante espressa su un *byte* (sugli 8 bit meno significativi).
- Ritorna -1 in caso di errore, o una chiave univoca a parità di path e identificativo.

System call per la Comunicazione tra Processi (IPC)

System call per Meccanismi di IPC Avanzati

- Code di Messaggi
- Memoria condivisa
- Semafori

Code di Messaggi

- Un messaggio è una unità di informazione di dimensione variabile, senza un formato predefinito.
- Una *coda* è un oggetto di sistema, identificato da una chiave *key*, contenente uno o più messaggi.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

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

- Serve a ottenere l'identificatore di una coda di messaggi se trova una corrispondenza, altrimenti restituisce un errore;
- Serve a creare una coda di messaggi data la chiave `key` nel caso in cui:
  - `key` = `IPC_PRIVATE`, oppure
  - `key` $\neq$ `IPC_PRIVATE` e `flag & IPC_CREAT` è vero.

Code di Messaggi: Gestione

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

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

- La funzione permette di accedere ai valori della struttura *msqid_ds*, mantenuta all'indirizzo *buf*, per la coda specificata dalla chiave *id*. `id` è il descrittore ritornato da `msgget`
- Il comportamento della funzione dipende dal valore dell'argomento *cmd*, che specifica il tipo di azione da eseguire:
  - `IPC_RMID`: cancella la coda (`buffer` non è usato).
  - `IPC_STAT`: ritorna informazioni relative alla coda nella struttura puntata da `buffer` (contiene info su UID, GID, stato della coda).
  - `IPC_SET`: Modifica un sottoinsieme dei campi contenuti nella struct.

Code di Messaggi: Gestione - (cont.)

- `buf` è un puntatore a una struttura definita in `sys/msg.h` contenente (campi utili):

```
struct msqid_ds
{
  struct ipc_perm msg_perm; /* permissions (rwxrwxrwx) */
  __time_t msg_stime;       /* time of last msgsnd command */
  __time_t msg_rtime;       /* time of last msgrcv command */
  __time_t msg_ctime;       /* time of last change */
  unsigned long int __msg_cbytes;   /* current #bytes on queue */
  msgqnum_t msg_qnum;       /* #messages currently on queue */
  msglen_t msg_qbytes;      /* max #bytes allowed on queue */
  __pid_t msg_lspid;        /* pid of last msgsnd() */
  __pid_t msg_lrpid;        /* pid of last msgrcv() */
};
```

Code di Messaggi: Scambio di Informazione

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgsnd(int id, struct msgbuf *msg, size_t size, int flag)
```

- La funzione invia un messaggio sulla coda `id` (`id` è il descrittore ritornato da `msgget`)
- Il messaggio ha lunghezza specificata da `size`, ed è passato attraverso l'argomento `msg`
- La funzione restituisce `0` in caso di successo, oppure `-1` in caso di errore
- Struttura `msgbuf` dei messaggi:

```c
struct msgbuf {
    long    mtype;      /* message type > 0 */
    char    mtext[1];   /* message text */
};
```

- Da interpretare come "template" di messaggio!
- In pratica, si usa una `struct` costruita dall'utente

Code di Messaggi: Scambio di Informazione - (cont.)

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgrcv(int id, struct msgbuf *msg, size_t size, long type, int flag);
```

- Legge un messaggio dalla coda `id`, lo scrive sulla struttura puntata da `msg` e ritorna il numero di byte letti. Una volta estratto, il messaggio sarà rimosso dalla coda
- L'argomento `size` indica la lunghezza massima del testo del messaggio
- Se il testo del messaggio ha lunghezza superiore a `size` il messaggio non viene estratto e la funzione ritorna con un errore.

Code di Messaggi: Scambio di Informazione - (cont.)

- `flag`:

`IPC_NOWAIT`	(`msgsnd` e `msgrcv`)	non si blocca se non ci sono messaggi da leggere
`MSG_NOERRROR`	(`msgrcv`)	tronca i messaggi a `size` byte senza errore

- `type` indica quale messaggio prelevare:

0	Il primo messaggio, indipendentemente dal tipo
> 0	Il primo messaggio di tipo `type`
< 0	Il primo messaggio con tipo più "vicino" al valore assoluto di `type`

Code di Messaggi: esempi di gestione del messaggio

```c
typedef struct {     /* Example 1: fixed length string */
    long mtype;
    char mtext[256];
} Msg1; Msg1 msg1;
const char * text = "Hello!";
strcpy(msg1.mtext, text); msg1.mtype = 1;
msgsnd(mq_id, &msg1, sizeof(Msg1) - sizeof(long), 0);
```

```c
typedef struct Msg2{ /* Example 2: multiple types */
    long mtype;
    int id;
    char s[22];
    int i;
} Msg2; Msg2 msg2; msg2.mtype = 2;
msgsnd(mq_id, &msg2, sizeof(Msg2) - sizeof(long), 0);
```

```c
typedef struct Msg3{ /* Example 3: variable length */
    long mtype;
    char mtext[1];
} Msg3; Msg3 * msg3;
const char * text = "Hello!";
msg3 = (Msg3) malloc(sizeof(Msg) + strlen(text)*sizeof(char));
strcpy(msg3->mtext, text); msg3->mtype = 3;
msgsnd(mq_id,msg3,
  sizeof(Msg)+strlen(text)*sizeof(char)-sizeof(long),0);
```

Code di Messaggi: Esempio (Server - part 1)

```
1  /*************************************
2    PROCESSO SERVER
3  *************************************/
4  #include <sys/types.h>
5  #include <unistd.h>
6  #include <sys/wait.h>
7  #include <sys/ipc.h>
8  #include <sys/msg.h>
9  #include <stdio.h>
10 #include <stdlib.h>
11
12 #define MSGKEY  75
13 #define MSGTYPE  1
14
15 int main (int argc,char *argv[]) {
16   key_t msgkey;
17   int msgid, pid;
```

Code di Messaggi: Esempio (Server - part 2)

```
1   struct msg {
2     long mtype;
3     char mtext[256];
4   } Message;
5
6   if ((msgid = msgget(MSGKEY,(0666|IPC_CREAT|IPC_EXCL))) == -1) {
7     perror(argv[0]);
8   }
9   const unsigned msg_size = sizeof(msg) - sizeof(long);
10  /* leggo dalla coda, aspettando il primo messaggio */
11  msgrcv(msgid,&Message,msg_size,MSGTYPE,0);
12  printf("Received from client: %s\n",Message.mtext);
13
14  /* scrivo in un messagio il pid e lo invio*/
15  pid = getpid();
16  sprintf(Message.mtext,"%d",pid);
17  Message.mtype = MSGTYPE;
18  msgsnd(msgid,&Message,msg_size,0);   /* WAIT */
19 }
```

Code di Messaggi: Esempio (Client - part 1)

```
1  /*************************************
2    PROCESSO CLIENT
3  *************************************/
4  #include <sys/types.h>
5  #include <unistd.h>
6  #include <sys/wait.h>
7  #include <sys/ipc.h>
8  #include <sys/msg.h>
9  #include <stdio.h>
10 #include <stdlib.h>
11
12 #define MSGKEY  75
13 #define MSGTYPE  1
14
15 int main (int argc,char *argv[]) {
16   key_t msgkey;
17   int msgid, pid;
```

Code di Messaggi: Esempio (Client - part 2)

```
1   typedef struct msg {
2     long mtype;
3     char mtext[256];
4   } msg;
5   msg Message;
6
7   if ((msgid = msgget(MSGKEY,0666)) == -1) {
8     perror(argv[0]);
9   }
10  /* invio il PID in un messagio */
11  pid = getpid();
12  sprintf(Message.mtext,"%d",pid);
13  const unsigned msg_size = sizeof(msg)-sizeof(long);
14  Message.mtype = MSGTYPE;
15  msgsnd(msgid,&Message,msg_size,0);   /* WAIT */
16
17  /* Ricevo il messaggio del server -- wait */
18  msgrcv(msgid,&Message,msg_size,MSGTYPE,0);
19  printf("Received message from server: %s\n",Message.mtext);
20 }
```

Code di Messaggi: Esempio (Controller - part 1)

```c
/*****************************************
MODULO: msgctl.c
SCOPO: Illustrare il funz. di msgctl()
*****************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <time.h>

void do_msgctl();
char warning_message[] = "If you remove read permission for"
                         "yourself, this program will fail frequently!";

int main(int argc, char* argv[]) {
  struct msqid_ds buf; /*buffer per msgctl()*/
  int cmd;   /* comando per msgctl() */
  int msqid; /* ID della coda da passare a msgctl() */
```

Code di Messaggi: Esempio (Controller - part 2)

```
1   if (argc!=2){
2     printf("Usage: msgctl.x <msgid>\n");
3     exit(1);
4   }
5
6   msqid = atoi(argv[1]);
7
8   fprintf(stderr, "\tIPC_RMID = %d\n", IPC_RMID);
9   fprintf(stderr, "\tIPC_SET = %d\n", IPC_SET);
10  fprintf(stderr, "\tIPC_STAT = %d\n", IPC_STAT);
11  fprintf(stderr, "\nScegliere l'opzione desiderata: ");
12
13  scanf("%i", &cmd);
```

Code di Messaggi: Esempio (Controller - part 3)

```
1   switch (cmd) {
2   case IPC_SET:
3     fprintf(stderr, "Prima della IPC_SET,
4                      controlla i valori correnti:");
5     /* notare: non e' stato inserito il break, quindi di seguito
6     vengono eseguite le istruzioni del case IPC_STAT */
7
8   case IPC_STAT:
9     do_msgctl(msqid, IPC_STAT, &buf);
10    fprintf(stderr, "msg_perm.uid = %d\n", buf.msg_perm.uid);
11    fprintf(stderr,"msg_perm.gid = %d\n", buf.msg_perm.gid);
12    fprintf(stderr, "msg_perm.cuid = %d\n", buf.msg_perm.cuid);
13    fprintf(stderr, "msg_perm.cgid = %d\n", buf.msg_perm.cgid);
14    fprintf(stderr, "msg_perm.mode = %#o, ", buf.msg_perm.mode);
15    fprintf(stderr, "access permissions = %#o\n",
16                    buf.msg_perm.mode & 0777);
17    fprintf(stderr, "msg_cbytes = %d\n", buf.msg_cbytes);
18    fprintf(stderr, "msg_qbytes = %d\n",buf.msg_qbytes);
19    fprintf(stderr, "msg_qnum = %d\n", buf.msg_qnum);
20    fprintf(stderr, "msg_lspid = %d\n", buf.msg_lspid);
21    fprintf(stderr, "msg_lrpid = %d\n", buf.msg_lrpid);
```

Code di Messaggi: Esempio (Controller - part 4)

```
1       if (buf.msg_stime) {
2         fprintf(stderr, "msg_stime = %s\n", ctime(&buf.msg_stime));
3       }
4       if (buf.msg_rtime) {
5         fprintf(stderr, "msg_rtime = %s\n", ctime(&buf.msg_rtime));
6       }
7         fprintf(stderr, "msg_ctime = %s", ctime(&buf.msg_ctime));
8
9       /* se il comando originario era IPC_STAT allora esco
10         dal case, altrimenti proseguo con le operazioni
11         specifiche per la modifica dei parametri della coda.
12      */
13      if (cmd == IPC_STAT)
14        break;
```

Code di Messaggi: Esempio (Controller - part 5)

```
1       /* Modifichiamo alcuni parametri della coda */
2       fprintf(stderr, "Enter msg_perm.uid: ");
3       scanf ("%hi", &buf.msg_perm.uid);
4       fprintf(stderr, "Enter msg_perm.gid: ");
5       scanf("%hi", &buf.msg_perm.gid);
6       fprintf(stderr, "%s\n", warning_message);
7       fprintf(stderr, "Enter msg_perm.mode: ");
8       scanf("%hi", &buf.msg_perm.mode);
9       fprintf(stderr, "Enter msg_qbytes: ");
10      scanf("%hi", &buf.msg_qbytes);
11      do_msgctl(msqid, IPC_SET, &buf);
12      break;
13
14    case IPC_RMID:
15    default:
16      /* Rimuove la coda di messaggi. */
17      do_msgctl(msqid, cmd, (struct msqid_ds *)NULL);
18      break;
19    }
20    exit(0);
21  }
```

Code di Messaggi: Esempio (Controller - part 6)

```
1  void do_msgctl(int msqid, int cmd, struct msqid_ds* buf) {
2    int rtrn; /* per memorizzare il valore di ritorno della msgctl() */
3
4    fprintf(stderr, "\nmsgctl: Calling msgctl(%d, %d, %s)\n",
5            msqid, cmd, buf ? "&buf" : "(struct msqid_ds *)NULL");
6    rtrn = msgctl(msqid, cmd, buf);
7
8    if (rtrn == -1) {
9      perror("msgctl: msgctl failed");
10     exit(1);
11   } else {
12     fprintf(stderr, "msgctl: msgctl returned %d\n", rtrn);
13   }
14 }
```

Analisi Esempi

Per provare gli esempi precedenti:

- lanciare il programma server
- lanciare il programma client su una shell separata
  - client e server si scambieranno un messaggio e termineranno lasciando la coda di messaggi attiva.
- eseguire il comando `ipcs -q` per verificare che effettivamente esista una coda attiva.
- lanciare il programma `msgctl.c` passandogli come parametro il valore *msqid* visualizzato dal comando `ipcs -q`
- provare le varie opzioni del programma `msgctl.c`, in particolare usare `IPC_SET` per variare le caratteristiche della coda e `IPC_RMID` per rimuovere la coda

System call per la Comunicazione tra Processi (IPC)

System call per Meccanismi di IPC Avanzati

- Code di Messaggi
- Memoria condivisa
- Semafori

Memoria Condivisa

- Due o più processi possono comunicare anche condividendo una parte del loro spazio di indirizzamento (virtuale).
- Questo spazio condiviso è detto *memoria condivisa* (*shared memory*), e la comunicazione avviene scrivendo e leggendo questa parte di memoria.

```
#include <sys/shm.h>
#include <sys/ipc.h>

int shmget(key_t key, size_t size, int flags);
```

- I parametri hanno lo stesso significato di quelli utilizzati da `msgget`.
- `size` indica la dimensione in byte della regione condivisa.

Memoria Condivisa – (cont.)

- Una volta creata, l'area di memoria non è subito disponibile.
- Deve essere *collegata* all'area dati dei processi che vogliono utilizzarla.

```
#include <sys/shm.h>
#include <sys/ipc.h>

void *shmat (int shmid, void *shmaddr, int flag)
```

- `shmaddr` indica l'indirizzo virtuale dove il processo vuole `attaccare` il segmento di memoria condivisa. Tipicamente è `NULL`.
- Il valore di ritorno rappresenta l'indirizzo di memoria condivisa effettivamente risultante.
- La memoria è ereditata dai processi creati con `fork()`, ma non con `exec()`.
- In caso di errore ritorna `(void *) -1`.

Memoria Condivisa – (cont.)

- In base ai valori di `flag` e di `shmaddr` si determina il punto di attacco del segmento:
  - shmaddr == NULL: memoria attaccata in un indirizzo a scelta del sistema operativo.
  - shmaddr != NULL && (flag & SHM_RND): attaccata al multiplo di `SHMLBA` più vicino a shmaddr, ma non maggiore di shmaddr.
  - shmaddr != NULL && !(flag & SHM_RND): shmaddr deve essere allineato ad una pagina di memoria.
- Il segmento è attaccato in lettura se `flag & SHM_RDONLY` è vero, altrimenti è contemporaneamente in lettura e scrittura.
- Un segmento attaccato in precedenza può essere "staccato" (*detached*) con `shmdt`

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

- `shmaddr` è l'indirizzo che individua il segmento di memoria condivisa.
- Non viene passato l'ID della regione perchè è possibile avere più aree di memoria identificate dallo stesso ID (cioè attaccate ad indirizzi diversi).

Memoria Condivisa: Gestione

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmctl (int shmid, int cmd, struct shmid_ds *buffer);
```

- `shmid` è il descrittore ritornato da `shmget`
- Valori di `cmd`:
  - `IPC_RMID`: cancella il segm. di memoria condivisa.
  - `IPC_STAT`: ritorna informazioni relative all'area di memoria condivisa nella struttura puntata da `buffer` (contiene info su UID, GID, permessi, stato della memoria).
  - `IPC_SET`: modifica un sottoinsieme dei campi contenuti nella struct (UID, GID, permessi).
  - `SHM_LOCK`: impedisce che il segmento venga swappato o paginato.
  - `SHM_UNLOCK`: ripristina il normale utilizzo della memoria condivisa

Memoria Condivisa: Gestione – (Cont.)

- `buffer` è un puntatore a una struttura definita in `sys/shm.h` contenente:

```
struct shmid_ds    {
    struct ipc_perm shm_perm; /* operation permission struct */
    size_t shm_segsz;         /* size of segment in bytes */
    __time_t shm_atime;       /* time of last shmat() */
    __time_t shm_dtime;       /* time of last shmdt() */
    __time_t shm_ctime;       /* time of last change by shmctl() */
    __pid_t shm_cpid;         /* pid of creator */
    __pid_t shm_lpid;         /* pid of last shmop */
    shmatt_t shm_nattch;      /* number of current attaches */
```

Memoria Condivisa: esempi accesso

```
1  int shmid1 = shmget(key1,sizeof(int),0666|IPC_CREAT);
2  int * i1 = (int*) shmat(shmid1, NULL, 0);
3
4  int shmid2 = shmget(key2,sizeof(int) * 10,0666|IPC_CREAT);
5  int * i2 = (int*) shmat(shmid2, NULL, 0);
6
7  typedef struct Data { /* Layout: memoria contigua! */
8    int i1;
9    char buf[20];
10   int i2;
11 } Data;
12 int shmid3 = shmget(key3,sizeof(Data),0666|IPC_CREAT);
13 Data * d = (Data*) shmat(shmid3, NULL, 0);
14
15 typedef struct Msg { /* As mqueue */
16   long type;
17   char buf[1];
18 } Msg;
19 int shmid4 = shmget(key4,sizeof(Msg)+sizeof(char)*256,0666|IPC_CREAT);
20 Msg * d = (Msg*) shmat(shmid4, NULL, 0);
```

Memoria Condivisa: xmalloc

- Esempio di come creare delle funzioni simili a `malloc()` e `free()`, per la memoria condivisa.

```
1  typedef struct XMem {
2    key_t key;
3    int shmid;
4    char buf[1];
5  } XMem;
6
7  void * xmalloc( key_t key, const size_t size) {
8    const int shmid = shmget(key,size+sizeof(XMem)-sizeof(char),
9      0666|IPC_CREAT);
10   if (shmid == -1) return NULL;
11   XMem * ret = (XMem*) shmat(shmid, NULL, 0);
12   if (ret == (void*)-1) return NULL;
13   ret->key = key;
14   ret->shmid = shmid;
15   return ret->buf;
16 }
```

Memoria Condivisa: xmalloc – (cont.)

```
1  void xfree(void * ptr) {
2    XMem tmp;
3    XMem * mem = (XMem *)(((char*)ptr)
4      - (((char*)&tmp.buf) - ((char*)&tmp.key)));
5    const int shmid = mem->shmid;
6    shmdt(mem);
7    shmctl(shmid, IPC_RMID, NULL);
8  }
9
10 int main(int argc, char * argv[]) {
11   int * xi = (int*) xmalloc(ftok(argv[0], 'a'), sizeof(int) * 8);
12   ...
13   xfree(xi);
14 }
```

Memoria Condivisa: Esempio (Controller - part 1)

```
1  /*****************************************************
2  MODULO: shmctl.c
3  SCOPO: Illustrare il funz. di shmctl()
4  USO: Lanciare il programma e fornire l'ID
5       di un segmento di memoria condivisa
6       precedentemente creato.
7       Usare il comando della shell ipcs per vedere
8       i segmenti di memoria condivisa attivi
9  *****************************************************/
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <sys/types.h>
13 #include <sys/ipc.h>
14 #include <sys/shm.h>
15 #include <time.h>
16
17 void do_shmctl();
```

Memoria Condivisa: Esempio (Controller - part 2)

```
1  int main(int argc, char *argv[]) {
2    int cmd; /* comando per shmctl() */
3    int shmid; /* ID dell'area di memoria condivisa*/
4    struct shmid_ds shmid_ds; /* struttura per il controllo
5                                 dell'area di memoria condivisa*/
6
7    fprintf(stderr,"Inserire l'ID del segmento di memoria condiviso:");
8    scanf("%i", &shmid);
9
10   fprintf(stderr, "Comandi validi:\n");
11   fprintf(stderr, "\tIPC_RMID =\t%d\n", IPC_RMID);
12   fprintf(stderr, "\tIPC_SET =\t%d\n", IPC_SET);
13   fprintf(stderr, "\tIPC_STAT =\t%d\n", IPC_STAT);
14   fprintf(stderr, "\tSHM_LOCK =\t%d\n", SHM_LOCK);
15   fprintf(stderr, "\tSHM_UNLOCK =\t%d\n", SHM_UNLOCK);
16   fprintf(stderr, "Scegliere il comando desiderato: ");
17   scanf("%i", &cmd);
```

Memoria Condivisa: Esempio (Controller - part 3)

```
1   switch (cmd) {
2     case IPC_STAT:
3       /* Le informazioni sullo stato della memoria condivisa
4       vengono recuperate con la chiamata alla funzione
5       do_shmctl(shmid, cmd, &shmid_ds) eseguita al termine
6       del case. Le informazioni saranno inserite nella
7       struttura shmid_ds*/
8       break;
9     case IPC_SET:
10      /*visualizzazione dello stato attuale della memoria */
11      do_shmctl(shmid, IPC_STAT, &shmid_ds);
12      /* Lettura da tastiera dei valori */
13    /* di UID, GID, e permessi da settare */
14      fprintf(stderr, "\nInserire shm_perm.uid: ");
15      scanf("%hi", &shmid_ds.shm_perm.uid);
16      fprintf(stderr, "Inserire shm_perm.gid: ");
17      scanf("%hi", &shmid_ds.shm_perm.gid);
18      fprintf(stderr, "N.B.: Mantieni il permesso
19              di lettura per te stesso!\n");
20      fprintf(stderr, "Inserire shm_perm.mode: ");
21      scanf("%hi", &shmid_ds.shm_perm.mode);
22      break;
```

Memoria Condivisa: Esempio (Controller - part 4)

```
1       case IPC_RMID: /* Rimuove il segmento */
2         break;
3       case SHM_LOCK: /* Esegui il lock sul segmento */
4         break;
5       case SHM_UNLOCK: /* Esegui unlock sul segmento */
6         break;
7       default: /* Comando sconosciuto passato a do_shmctl */
8         break;
9     }
10    /* La funzione do_shmctl esegue il comando scelto dall'utente */
11    do_shmctl(shmid, cmd, &shmid_ds);
12    exit(0);
13  }
14
15  void do_shmctl(int shmid, int cmd, struct shmid_ds* buf) {
16    int rtrn; /* valore di ritorno della shmctl */
17
18    fprintf(stderr,"shmctl: Chiamo shmctl(%d, %d, buf)\n", shmid, cmd);
```

Memoria Condivisa: Esempio (Controller - part 5)

```
1   if (cmd == IPC_SET) {
2     fprintf(stderr, "\tbuf->shm_perm.uid == %d\n",
3                   buf->shm_perm.uid);
4     fprintf(stderr, "\tbuf->shm_perm.gid == %d\n",
5                   buf->shm_perm.gid);
6     fprintf(stderr, "\tbuf->shm_perm.mode == %#o\n",
7                   buf->shm_perm.mode);
8   }
9   if ((rtrn = shmctl(shmid, cmd, buf)) == -1) {
10    perror("shmctl: shmctl fallita.");
11    exit(1);
12  } else {
13    fprintf(stderr, "shmctl: shmctl ha ritornato %d\n", rtrn);
14  }
15  if (cmd != IPC_STAT && cmd != IPC_SET)
16    return; /* ritorno perche' il comando e' stato eseguito e non
17                  devo visualizzare nessuna informazione sullo stato */
```

Memoria Condivisa: Esempio (Controller - part 6)

```
1   /* Stampa lo stato corrente del segmento */
2   fprintf(stderr, "\nCurrent status:\n");
3   fprintf(stderr, "\tshm_perm.uid = %d\n", buf->shm_perm.uid);
4   fprintf(stderr, "\tshm_perm.gid = %d\n", buf->shm_perm.gid);
5   fprintf(stderr, "\tshm_perm.cuid = %d\n", buf->shm_perm.cuid);
6   fprintf(stderr, "\tshm_perm.cgid = %d\n", buf->shm_perm.cgid);
7   fprintf(stderr, "\tshm_perm.mode = %#o\n", buf->shm_perm.mode);
8   fprintf(stderr, "\tshm_perm.key = %#x\n", buf->shm_perm.__key);
9   fprintf(stderr, "\tshm_segsz = %d\n", buf->shm_segsz);
10  fprintf(stderr, "\tshm_lpid = %d\n", buf->shm_lpid);
11  fprintf(stderr, "\tshm_cpid = %d\n", buf->shm_cpid);
12  fprintf(stderr, "\tshm_nattch = %d\n", buf->shm_nattch);
13  if (buf->shm_atime)
14    fprintf(stderr, "\tshm_atime = %s", ctime(&buf->shm_atime));
15  if (buf->shm_dtime)
16    fprintf(stderr, "\tshm_dtime = %s", ctime(&buf->shm_dtime));
17  fprintf(stderr, "\tshm_ctime = %s", ctime(&buf->shm_ctime));
18 }
```

Memoria Condivisa: Esempio (Shm1 - part 1)

```
1  /*************************************************************
2  NOME: shm1.c
3  SCOPO: ``attaccare'' due volte un'area di memoria condivisa
4         Ricordarsi di rimuovere la memoria condivisa al termine
5         del programma lanciando shmctl.c oppure tramite il comando
6         della shell ipcrm.
7  *************************************************************/
8  #include <stdlib.h>
9  #include <stdio.h>
10 #include <sys/types.h>
11 #include <sys/ipc.h>
12 #include <sys/shm.h>
13 #define K 1
14 #define SHMKEY 75
15 #define N 10
```

Memoria Condivisa: Esempio (Shm1 - part 2)

```
1  int shmid;
2  int main (int argc, char *argv[]) {
3    int i, *pint;
4    char *addr1, *addr2;
5
6    /* Creao il segmento condiviso di dimensione 128*K byte */
7    shmid = shmget(SHMKEY, 128*K, 0777|IPC_CREAT);
8    /* Attacco il segmento in due zone diverse */
9    addr1 = shmat(shmid,0,0);
10   addr2 = shmat(shmid,0,0);
11
12   printf("Addr1 = 0x%x\t Address2 = 0x%x\t\n", addr1,addr2);
```

Memoria Condivisa: Esempio (Shm1 - part 3)

```
1   /* scrivo nella regione 1 */
2   pint = (int*)addr1;
3   for (i=0;i<N;i++) {
4     *pint = i;
5     printf("Writing: Index %4d\tValue: %4d\tAddress: 0x%x\n",
6                         i,*pint,pint);
7     pint++;
8   }
9   /* leggo dalla regione 2 */
10  pint = (int*)addr2;
11  for (i=0;i<N;i++) {
12    printf("Reading: Index %4d\tValue: %4d\tAddress: 0x%x\n",
13                        i,*pint,pint);
14    pint++;
15  }
16 }
```

Memoria Condivisa: Esempio (Shm2 - part 1)

```
1  /*************************************************************
2  NOME:   shm2.c
3  SCOPO:  ``attaccarsi'' ad un area di memoria condivisa
4  USO:    lanciare prima il programma shm1.c per creare la memoria
5          condivisa.
6          Ricordarsi di rimuovere la memoria condivisa al termine
7          del programma lanciando shmctl.c oppure tramite il comando
8          della shell ipcrm.
9  *************************************************************/
10 #include <sys/types.h>
11 #include <sys/ipc.h>
12 #include <sys/shm.h>
13 #include <stdio.h>
14 #define K 1
15 #define N 20
16 #define SHMKEY 75
17
18 int shmid;
```

Memoria Condivisa: Esempio (Shm2 - part 2)

```c
int main (int argc, char *argv[]) {
  int i, *pint;
  char *addr;

  /*mi attacco alla regione creata dal programma shm1.c*/
  shmid = shmget(SHMKEY, 128*K, 0777);
  addr = shmat(shmid,0,0);
  printf("Address = 0x%x\n", addr);
  pint = (int*) addr;
  /* leggo dalla regione attaccata in precedenza */
  for (i=0;i<N;i++) {
    printf("Reading: (Value = %4d)\n",*pint++);
  }
}
```

Memoria Condivisa: Esempio (Server - part 1)

```
1  /*************************************************************
2  MODULO: shm_server.c
3  SCOPO: server memoria condivisa
4  USO: lanciare il programma shm_server.c in una shell
5       e il programma shm_client.c in un'altra shell
6       Ricordarsi di rimuovere la memoria condivisa creata
7       dal server al termine del programma lanciando shmctl.c
8       oppure tramite il comando della shell ipcrm.
9  *************************************************************/
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <sys/types.h>
13 #include <sys/ipc.h>
14 #include <sys/shm.h>
15 #include <stdio.h>
16 #define SHMSZ 27
```

Memoria Condivisa: Esempio (Server - part 2)

```
1  int main(int argc, char *argv[]) {
2    char c;
3    int shmid;
4    key_t key;
5    char *shm, *s;
6    key = 5678;
7
8
9    /* Creo il segmento */
10   if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0) {
11     perror("shmget");
12     exit(1);
13   }
14   /* Attacco il segmento all'area dati del processo */
15   if ((shm = shmat(shmid, NULL, 0)) == (void *) -1) {
16     perror("shmat");
17     exit(1);
18   }
```

Memoria Condivisa: Esempio (Server - part 3)

```
1     s = shm;
2     for (c = 'a'; c <= 'z'; c++)
3       *s++ = c;
4
5     *s = NULL;
6     while (*shm != '*')
7       sleep(1);
8
9     printf("Received '*'. Exiting...\n");
10    exit(0);
11  }
```

Memoria Condivisa: Esempio (Client - part 1)

```
1  /*************************************************************
2  MODULO: shm_client.c
3  SCOPO: client memoria condivisa
4  USO: lanciare il programma shm_server.c in una shell
5       e il programma shm_client.c in un'altra shell
6       Ricordarsi di rimuovere la memoria condivisa creata
7       dal server al termine del programma lanciando shmctl.c
8       oppure tramite il comando della shell ipcrm.
9  *************************************************************/
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <sys/types.h>
13 #include <sys/ipc.h>
14 #include <sys/shm.h>
15 #include <stdio.h>
16 #define SHMSZ 27
```

Memoria Condivisa: Esempio (Client - part 2)

```
1  int main(int argc, char *argv[]) {
2    int shmid;
3    key_t key;
4    char *shm, *s;
5    key = 5678;
6
7    /* Recupero il segmento creato dal server */
8    if ((shmid = shmget(key, SHMSZ, 0666)) < 0) {
9      perror("shmget");
10     exit(1);
11   }
12
13   /* Attacco il segmento all'area dati del processo */
14   if ((shm = shmat(shmid, NULL, 0)) == (void *) -1) {
15     perror("shmat");
16     exit(1);
17   }
```

Memoria Condivisa: Esempio (Client - part 3)

```
1    /* Stampo il contenuto della memoria */
2    printf("Contenuto del segmento condiviso con il server:");
3    for (s = shm; *s != NULL; s++)
4      putchar(*s);
5
6    putchar('\n');
7    sleep(3);
8    /* ritorno * al server affinche' possa terminare */
9    *shm = '*';
10   exit(0);
11 }
```

Sincronizzazione tra Processi

- I semafori permettono la sincronizzazione dell'esecuzione di due o più processi
  - Sincronizzazione su un dato valore
  - Mutua esclusione
- Semafori SystemV:
  - piuttosto diversi da semafori classici
  - "pesanti" dal punto di vista della gestione
- Disponibili varie API (per es. POSIX semaphores)

Semafori

- I semafori sono differenti dalle altre forme di IPC.
- Sono contatori usati per controllare l'accesso a risorse condivise da processi diversi.
- Il protocollo per accedere alla risorsa è il seguente
  1. testare il semaforo;
  2. se > 0, allora si può usare la risorsa (e viene decrementato il semaforo);
  3. se $== 0$, processo va in *sleep* finchè il semaforo non ridiventa > 0, a quel punto *wake up* e goto step 1;
- Quando ha terminato con la risorsa, incrementa il semaforo.
- Se il semaforo è a 0 significa che si sta utilizzando il 100% della risorsa.
- Un semaforo binario (valori possibili 0 e 1) è detto *mutex*.

Semafori - (cont.)

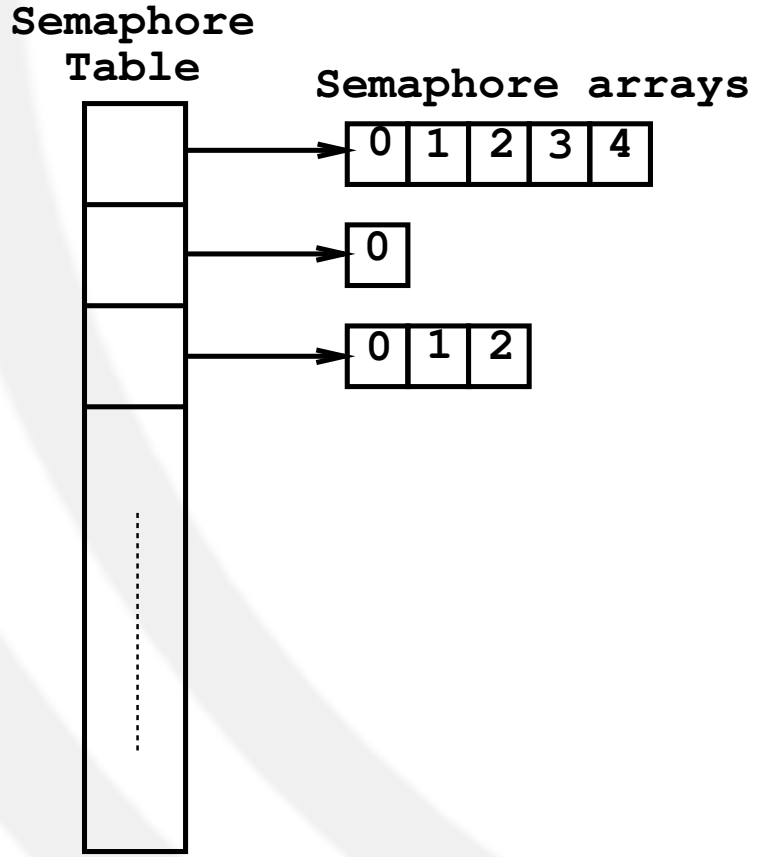
- È importante notare che per implementare correttamente un semaforo l'operazione di verifica del valore del semaforo ed il decremento/incremento devono costituire una operazione atomica.
- Per questa ragione i semafori sono implementati all'interno del kernel.

Semafori - (cont.)

- Il semaforo binario è la forma più semplice.
  - Controlla un'unica risorsa ed il suo valore è inizializzato a 1.
  - E.g., stampante capace di gestire 1 sola stampa alla volta.
- Forma più complessa.
  - Inizializzato ad un valore positivo indicante il numero di risorse che sono a disposizione per essere condivise.
  - E.g., stampante capace di gestire 10 stampe alla volta.

Semafori (System V API)

- Non è possibile allocare un singolo semaforo, ma è necessario crearne un insieme (vettore di semafori)
- Struttura interna di un semaforo



Semafori (SystemV API)

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semget(key_t key, int nsems, int semflg);
```

- I valori di `key` e di `semflg` sono identici al caso delle code di messaggi e della shared memory
- `nsems` è il numero di semafori identificati dal `semid` (quanti semafori sono contenuti nel vettore).
  - Il primo semaforo del vettore ha indice 0
- Esempi
  - `semid = semget(key, 1, IPC_CREAT|IPC_EXCL|0666);`
    crea un insieme con un solo semaforo
  - `semid = semget(key, 0, 0666);`
    recupera un semaforo esistente
- NOTA: anche se Linux inizializza i semafori a zero, una applicazione portabile non deve fare affidamento su questo comportamento.

Operazioni su Semafori

```
/* 4o parametro opzionale di tipo semun (args) */
int semctl(int semid, int semnum, int cmd, ...);
union semun {
  int val; /* SETVAL */
  struct semid_ds* buffer; /* IPC_STAT, IPC_SET */
  unsigned short *array; /* GET_ALL, SET_ALL */
  struct seminfo *__buf; /* IPC_INFO solo per Linux */
};
```

- Esegue l'operazione di controllo specificata da *cmd* sull'insieme di semafori specificato da *semid* o sul *semnum*-esimo semaforo dell'insieme. A seconda del comando il quarto parametro è opzionale.
- Operazioni (`cmd`):

`IPC_RMID`	Rimuove il set di semafori
`IPC_SET`	Modifica il set di semafori
`IPC_STAT`	Statistiche sul set di semafori
`GETVAL`	legge il valore del semaforo `semnum` in `args.val`
`GETALL`	legge tutti i valori in `args.array`
`SETVAL`	assegna il valore del semaforo `semnum` in `args.val`
`SETALL`	assegna tutti i semafori con i valori in `args.array`
`GETPID`	Valore di PID dell'ultimo processo che ha fatto operazioni
`GETNCNT`	numero di processi in attesa che un semaforo aumenti
`GETZCNT`	numero di processi in attesa che un semaforo diventi 0

Operazioni su Semafori

- `buffer` è un puntatore ad una struttura `semid_ds` definita in `sys/sem.h`:

```
struct semid_ds    {
    struct ipc_perm sem_perm;  /* operation permission struct */
    time_t sem_otime;          /* Last semop time */
    time_t sem_ctime;          /* Last change time */
    unsigned short sem_nsems;  /* No. of semaphores */
  };
```

- La union `semun` deve essere definita nel codice del processo che chiama la `semctl()`
- `ipc_perm` è una struttura definita in `sys/ipc.h`:

```
struct ipc_perm {
  key_t          __key; /* Key supplied to semget(2) */
  uid_t          uid;   /* Effective UID of owner */
  gid_t          gid;   /* Effective GID of owner */
  uid_t          cuid;  /* Effective UID of creator */
  gid_t          cgid;  /* Effective GID of creator */
  unsigned short mode;  /* Permissions */
  unsigned short __seq; /* Sequence number */
};
```

Operazioni su Semafori

La `semctl()` ritorna valori diversi a seconda del comando eseguito :

- `GETNCNT`: il numero di processi in attesa che il semaforo `semnum` venga incrementato
- `GETPID`: il PID dell'ultimo processo che ha effettuato `semop()` sul semaforo `semnum` dell'insieme
- `GETVAL`: il valore del semaforo `semnum` dell'insieme
- `GETZCNT`: il numero di processi in attesa che il semaforo `semnum` diventi 0

Semafori: Esempio (Controller - part 1)

```
1  /*************************************************************
2  MODULO: semctl.c
3  SCOPO:  Illustrare il funz. di semctl()
4  USO:    prima di lanciare semctl.c() creare un semaforo usando un
5          altro programma (ad esempio sem.x)
6  *************************************************************/
7  #include   <stdlib.h>
8  #include   <stdio.h>
9  #include   <sys/types.h>
10 #include   <sys/ipc.h>
11 #include   <sys/sem.h>
12 #include   <time.h>
13
14 struct semid_ds semid_ds;
```

Semafori: Esempio (Controller - part 2)

```
1  /* explicit declaration required */
2  union semun {
3    int val;
4    struct semid_ds* buf;
5    unsigned short int *array;
6    struct seminfo *__buf;
7  } arg;
8
9  void do_semctl(int semid, int semnum, int cmd, union semun arg);
10 void do_stat();
11 char warning_message[] = "If you remove read permission for yourself,
12                           this program will fail frequently!";
13
14 int main(int argc, char *argv[]) {
15   union semun arg; /* union to pass to semctl() */
16   int cmd;         /* command to give to semctl() */
17   int i;
18   int semid;       /* semid to pass to semctl() */
19   int semnum;      /* semnum to pass to semctl() */
```

Semafori: Esempio (Controller - part 3)

```
1   fprintf(stderr, "Enter semid value: ");
2   scanf("%i", &semid);
3
4   fprintf(stderr, "Valid semctl cmd values are:\n");
5   fprintf(stderr, "\tGETALL = %d\n", GETALL);
6   fprintf(stderr, "\tGETNCNT = %d\n", GETNCNT);
7   fprintf(stderr, "\tGETPID = %d\n", GETPID);
8   fprintf(stderr, "\tGETVAL = %d\n", GETVAL);
9   fprintf(stderr, "\tGETZCNT = %d\n", GETZCNT);
10  fprintf(stderr, "\tIPC_RMID = %d\n", IPC_RMID);
11  fprintf(stderr, "\tIPC_SET = %d\n", IPC_SET);
12  fprintf(stderr, "\tIPC_STAT = %d\n", IPC_STAT);
13  fprintf(stderr, "\tSETALL = %d\n", SETALL);
14  fprintf(stderr, "\tSETVAL = %d\n", SETVAL);
15  fprintf(stderr, "\nEnter cmd: ");
16  scanf("%i", &cmd);
```

Semafori: Esempio (Controller - part 4)

```
1   /* Do some setup operations needed by multiple commands. */
2   switch (cmd) {
3     case GETVAL:
4     case SETVAL:
5     case GETNCNT:
6     case GETZCNT:
7     /* Get the semaphore number for these commands. */
8       fprintf(stderr, "\nEnter semnum value: ");
9       scanf("%i", &semnum);
10      break;
11    case GETALL:
12    case SETALL:
13      /* Allocate a buffer for the semaphore values. */
14      fprintf(stderr,"Get number of semaphores in the set.\n");
15      arg.buf = &semid_ds;
16      /* when IPC_STAT is called the second argument of semctl()
17         is ignored. IPC_STAT is called to retrieve info semid_ds
18         on the semaphore set */
19      do_semctl(semid, 0, IPC_STAT, arg);
20      if (arg.array = (u_short *)malloc(
21                (unsigned) (semid_ds.sem_nsems * sizeof(u_short)))) {
22         /* Break out if you got what you needed */
23         break;
24      }
```

Semafori: Esempio (Controller - part 5)

```
1       fprintf(stderr,"semctl: unable to allocate spacefor %d values\n",
2               semid_ds.sem_nsems);
3       exit(2);
4   }
5
6   /*Get the rest of the arguments needed for the specified command.*/
7   switch (cmd) {
8     case SETVAL:
9       /* Set value of one semaphore. */
10      fprintf(stderr, "\nEnter semaphore value: ");
11      scanf("%i", &arg.val);
12      do_semctl(semid, semnum, SETVAL, arg);
13      /* Fall through to verify the result. */
14      fprintf(stderr,
15          "Executing semctl GETVAL command to verify results...\n");
16    case GETVAL:
17      /* Get value of one semaphore. */
18      arg.val = 0;
19      do_semctl(semid, semnum, GETVAL, arg);
20      break;
```

Semafori: Esempio (Controller - part 6)

```
1     case GETPID:
2       /* Get PID of the last process that successfully completes a
3       semctl(SETVAL), semctl(SETALL), or semop() on the semaphore. */
4       arg.val = 0;
5       do_semctl(semid, 0, GETPID, arg);
6       break;
7     case GETNCNT:
8       /* Get number of processes waiting for semaphore value
9          to increase. */
10      arg.val = 0;
11      do_semctl(semid, semnum, GETNCNT, arg);
12      break;
13    case GETZCNT:
14      /* Get number of processes waiting for semaphore value to
15         become zero. */
16      arg.val = 0;
17      do_semctl(semid, semnum, GETZCNT, arg);
18      break;
```

Semafori: Esempio (Controller - part 7)

```
1     case SETALL:
2       /* Set the values of all semaphores in the set. */
3       fprintf(stderr, "There are %d semaphores in the set.\n",
4           semid_ds.sem_nsems);
5       fprintf(stderr, "Enter semaphore values:\n");
6       for (i = 0; i < semid_ds.sem_nsems; i++) {
7         fprintf(stderr, "Semaphore %d: ", i);
8         scanf("%hi", &arg.array[i]);
9       }
10      do_semctl(semid, 0, SETALL, arg);
11      /* Fall through to verify the results. */
12      fprintf(stderr,"Do semctl GETALL command to verify results.\n");
13    case GETALL:
14      /* Get and print the values of all semaphores in the set.*/
15      do_semctl(semid, 0, GETALL, arg);
16      fprintf(stderr, "The values of the %d semaphores are:\n",
17          semid_ds.sem_nsems);
18      for (i = 0; i < semid_ds.sem_nsems; i++)
19        fprintf(stderr, "%d ", arg.array[i]);
20      fprintf(stderr, "\n");
21      break;
```

Semafori: Esempio (Controller - part 8)

```
1     case IPC_SET:
2       /* Modify mode and/or ownership. */
3       arg.buf = &semid_ds;
4       do_semctl(semid, 0, IPC_STAT, arg);
5       fprintf(stderr, "Status before IPC_SET:\n");
6       do_stat();
7       fprintf(stderr, "Enter sem_perm.uid value: ");
8       scanf("%hi", &semid_ds.sem_perm.uid);
9       fprintf(stderr, "Enter sem_perm.gid value: ");
10      scanf("%hi", &semid_ds.sem_perm.gid);
11      fprintf(stderr, "%s\n", warning_message);
12      fprintf(stderr, "Enter sem_perm.mode value: ");
13      scanf("%hi", &semid_ds.sem_perm.mode);
14      do_semctl(semid, 0, IPC_SET, arg);
15      /* Fall through to verify changes. */
16      fprintf(stderr, "Status after IPC_SET:\n");
17    case IPC_STAT:
18      /* Get and print current status. */
19      arg.buf = &semid_ds;
20      do_semctl(semid, 0, IPC_STAT, arg);
21      do_stat();
22      break;
```

Semafori: Esempio (Controller - part 9)

```
1       case IPC_RMID:
2         /* Remove the semaphore set. */
3         arg.val = 0;
4         do_semctl(semid, 0, IPC_RMID, arg);
5         break;
6       default:
7         /* Pass unknown command to semctl. */
8         arg.val = 0;
9         do_semctl(semid, 0, cmd, arg);
10        break;
11    }
12  exit(0);
13 }
14
15 void do_semctl(int semid, int semnum, int cmd, union semun arg) {
16   int i;
17
18   fprintf(stderr, "\nsemctl: Calling semctl(%d, %d, %d, ",
19         semid, semnum, cmd);
```

Semafori: Esempio (Controller - part 10)

```
1  void do_semctl(int semid, int semnum, int cmd, union semun arg) {
2    int i;
3
4    fprintf(stderr, "\nsemctl: Calling semctl(%d, %d, %d, ",
5        semid, semnum, cmd);
6    switch (cmd) {
7      case GETALL:
8        fprintf(stderr, "arg.array = %#x)\n", arg.array);
9        break;
10     case IPC_STAT:
11     case IPC_SET:
12       fprintf(stderr, "arg.buf = %#x)\n", arg.buf);
13       break;
14     case SETALL:
15       fprintf(stderr, "arg.array = [");
16       for (i = 0;i < semid_ds.sem_nsems;i++) {
17         fprintf(stderr, "%d", arg.array[i]);
18        if (i < semid_ds.sem_nsems)
19           fprintf(stderr, ", ");
20       }
21       fprintf(stderr, "])\n");
22       break;
23   }
```

Semafori: Esempio (Controller - part 11)

```
1    /* call to semctl() */
2    i = semctl(semid, semnum, cmd, arg);
3    if (i == -1) {
4      perror("semctl: semctl failed");
5      exit(1);
6    }
7    fprintf(stderr, "semctl: semctl returned %d\n", i);
8    return;
9  }
10
11 void do_stat() {
12   fprintf(stderr, "sem_perm.uid = %d\n", semid_ds.sem_perm.uid);
13   fprintf(stderr, "sem_perm.gid = %d\n", semid_ds.sem_perm.gid);
14   fprintf(stderr, "sem_perm.cuid = %d\n", semid_ds.sem_perm.cuid);
15   fprintf(stderr, "sem_perm.cgid = %d\n", semid_ds.sem_perm.cgid);
16   fprintf(stderr, "sem_perm.mode = %#o, ", semid_ds.sem_perm.mode);
17   fprintf(stderr, "access permissions = %#o\n",
18       semid_ds.sem_perm.mode & 0777);
19   fprintf(stderr, "sem_nsems = %d\n", semid_ds.sem_nsems);
20   fprintf(stderr, "sem_otime = %s",
21       semid_ds.sem_otime ? ctime(&semid_ds.sem_otime) : "Not Set\n");
22   fprintf(stderr, "sem_ctime = %s", ctime(&semid_ds.sem_ctime));
23 }
```

Operazioni su Semafori

```
int semop(int semid, struct sembuf* sops, unsigned nsops);
```

- Applica l'insieme (array) `sops` di operazioni (in numero pari a `nsops`) all'insieme di semafori `semid`.
- Le operazioni, contenute in un array opportunamente allocato, sono descritte dalla `struct sembuf`:

```
struct sembuf {
  short sem_num;
  short sem_op;
  short sem_flg;
};
```

  - `sem_num`: semaforo su cui l'operazione (i-esima) viene applicata
  - `sem_op`: l'operazione da applicare
  - `sem_flag`: le modalità con cui l'operazione viene applicata

Operazioni su Semafori

- Valori di `sem_op`:

< 0	equivale a `P` si blocca se `sem_val - |sem_op|` $<$ `0` altrimenti decrementa il semaforo della quantità `ops.sem_op`
$= 0$	In attesa che il valore del semaforo diventi 0
> 0	equivalente a `V` Incrementa il semaforo della quantità `ops.sem_op`

- Valori di `sem_flg`:

`IPC_NOWAIT`	Per realizzare `P` e `V` non bloccanti (comodo per realizzare *polling*)
`SEM_UNDO`	Ripristina il vecchio valore quando termina (serve nel caso di terminazioni precoci)

Operazioni su Semafori

NOTA BENE: L'insieme di operazioni inserite in una chiamata alla system call `semop()` viene eseguito in modo atomico. Se una delle operazioni non può essere eseguita, il comportamento della system call `semop()` dipende dalla flag `IPC_NOWAIT`:

- se `IPC_NOWAIT` è settato, `semop()` fallisce e ritorna -1;
- se `IPC_NOWAIT` non è settato, il processo viene bloccato;

Semafori: Esempio (sem - part 1)

```
1  /*************************************************
2  MODULO: sem.c
3  SCOPO:  Illustrare il funz. di semop() e semctl()
4  USO:    Lanciare il programma sem.x e quindi
5          semop.x o semctl.x in una shell separata
6  *************************************************/
7
8  #include     <stdio.h>
9  #include     <stdlib.h>
10 #include     <sys/types.h>
11 #include     <sys/ipc.h>
12 #include     <sys/sem.h>
13
14 #define KEY 14
```

Semafori: Esempio (sem - part 2)

```
1  int main(int argc, char *argv[]) {
2
3    int semid;  /* identificatore dei semafori */
4    int i;
5    /* struttura per le operazioni sui semafori */
6    struct sembuf * sops =
7              (struct sembuf *) malloc (sizeof(struct sembuf));
8
9    /*
10     Creazione di due semafori con permessi di lettura e scrittura
11     per tutti. Le flag ICP_CREAT e IPC_EXCL fanno si che la
12     funzione semget ritorni errore se esiste già un vettore di
13     semafori con chiave KEY. Vedere man semget.
14   */
15   if((semid = semget(KEY, 2, IPC_CREAT | IPC_EXCL | 0666)) == -1) {
16     perror("semget");
17     exit(1);
18   }
19
20   /* Inizializzo i due semafori a 1 */
```

Semafori: Esempio (sem - part 3)

```
1   /*
2     Per eseguire operazioni ATOMICHE sui semafori si usa la funzione
3     semop(). Vedere man semop.
4     Alla funzione semop() vengono passati 3 argomenti:
5     - l'identificatore dell'array di semafori su cui eseguire
6     l'operazione
7     - il puntatore alla struttura sembuf necessaria per eseguire
8     le operazioni
9     - il numero di operazioni da eseguire
10
11    Per ogni operazione da eseguire è necessario creare una
12    struttura di tipo sembuf. La struttura contiene 3 campi:
13    - il numero del semaforo da utilizzare. Ricordare che la semget
14    ritorna array di semafori.
15    - un intero N che rappresenta l'operazione da eseguire.
16      Se l'intero N e' > 0 il valore del semaforo viene incrementato
17      di tale quantità. Se N = 0 la semop blocca il processo in
18      attesa che il valore del semaforo diventi 0. Se N < 0 la semop
19      blocca il processo in attesa che il valore del semaforo meno N
20      sia maggiore o uguale a 0.
21    - una eventuale flag (IPC_NOWAIT o SEM_UNDO)
22      IPC_NOWAIT serve per avere semafori non bloccanti
23      SEM_UNDO serve per ripristinare il vecchio valore del semaforo
24               in caso di terminazioni precoci.
25  */
```

Semafori: Esempio (sem - part 4)

```
1     sops->sem_num = 0;   /* semaforo 0 */
2     sops->sem_op = 1;    /* incrementa di uno il valore del
3                             semaforo 0 */
4     sops->sem_flg = 0;   /* nessuna flag settata */
5     /* esecuzione dell'operazione sul semaforo 0 */
6     semop(semid, sops, 1);
7
8     sops->sem_num = 1;   /* semaforo 1 */
9     sops->sem_op = 1;    /* incrementa di uno il valore del
10                            semaforo 1 */
11    sops->sem_flg = 0;
12    /* esecuzione dell'operazione sul semaforo 1 */
13    semop(semid, sops, 1);
14
15    printf("I semafori sono stati inizializzati a 1.\n");
16    printf("Lanciare il programma semctl.x o semop.x su un'altra
17            shell e fornire semid=%d\n", semid);
```

Semafori: Esempio (sem - part 5)

```
1       while(1) {
2
3         sops->sem_num = 0;   /* semaforo 0 */
4         sops->sem_op = 0;    /* attende che il semaforo valga zero */
5         sops->sem_flg = 0;
6         /* quando viene eseguita questa operazione il codice si
7            blocca in attesa che il valore del semaforo 0 diventi 0 */
8         semop(semid, sops, 1);
9         /* Quando il semaforo diventa 0 stampo che il processo
10           e' stato sbloccato */
11        printf("Sbloccato 1\n");
12
13        sops->sem_num = 1;   /* semaforo 1 */
14        sops->sem_op = 0;    /* attende che il semaforo valga zero */
15        sops->sem_flg = 0;
16
17        /* quando viene eseguita questa operazione il codice
18         si blocca in attesa che il valore del semaforo 1 diventi 0 */
19        semop(semid, sops, 1);
20        /* Quando il semaforo diventa 0 stampo che il processo
21           e' stato sbloccato */
22        printf("                Sbloccato 2\n");
23
24      }
25  }
```

Semafori: Esempio (semop - part 1)

```
1  /*****************************************
2  MODULO: semop.c
3  SCOPO: Illustrare il funz. di semop()
4  *****************************************/
5  #include     <stdio.h>
6  #include     <stdlib.h>
7  #include     <sys/types.h>
8  #include     <sys/ipc.h>
9  #include     <sys/sem.h>
10
11 union semun {
12   int               val;    /* Valore per SETVAL */
13   struct semid_ds  *buf;    /* Buffer per IPC_STAT, IPC_SET */
14   unsigned short   *array;  /* Array per GETALL, SETALL */
15   struct seminfo   *__buf;  /* Buffer per IPC_INFO (specifico
16                                di Linux) */
17 };
18
19 int ask(int* semidp, struct sembuf **sopsp);
```

Semafori: Esempio (semop - part 2)

```
1  static struct semid_ds  semid_ds;  /* stato del set di semafori */
2  static char error_mesg1[] = "semop: Non posso allocare spazio per un
3                               vettore di %d valori.\n";
4  static char error_mesg2[] = "semop: Non posso allocare spazio per %d
5                               strutture sembuf. \n";
6
7  int main(int argc, char* argv[]) {
8    int     i;
9    int     nsops;          /* numero di operazioni da fare */
10   int     semid;          /* semid per il set di semafori */
11   struct sembuf *sops; /* puntatore alle operazioni da eseguire */
12
13   /* Cicla finche' l'utente vuole eseguire operazioni chiamando la
14      funzione ask */
15   while (nsops = ask(&semid, &sops)) {
16     /* Inizializza il vettore di operazioni da eseguire */
17     for (i = 0; i < nsops; i++) {
18       fprintf(stderr,
19       "\nInserire il valore per l'operazione %d di %d.\n",i+1,nsops);
20       fprintf(stderr,"sem_num(i valori validi sono 0<=sem_num<%d): ",
21                semid_ds.sem_nsems);
22       scanf("%d", &sops[i].sem_num);
```

Semafori: Esempio (semop - part 3)

```
1       fprintf(stderr, "sem_op: ");
2       scanf("%d", &sops[i].sem_op);
3       fprintf(stderr,"Possibili flag per sem_flg:\n");
4       fprintf(stderr, "\tIPC_NOWAIT =\t%#6.6o\n",IPC_NOWAIT);
5       fprintf(stderr, "\tSEM_UNDO =\t%#6.6o\n",SEM_UNDO);
6       fprintf(stderr, "\tNESSUNO =\t%6d\n",0);
7       fprintf(stderr, "sem_flg: ");
8       /* controllare cosa fa %i su man scanf */
9       scanf("%i", &sops[i].sem_flg);
10    }
11
12    /* Ricapitola la chiamata da fare a semop() */
13    fprintf(stderr,"\nsemop: Chiamo semop(%d, &sops, %d) with:",
14            semid, nsops);
15    for (i = 0; i < nsops; i++) {
16      fprintf(stderr, "\nsops[%d].sem_num = %d, ",i,sops[i].sem_num);
17      fprintf(stderr, "sem_op = %d, ", sops[i].sem_op);
18      fprintf(stderr, "sem_flg = %o\n",sops[i].sem_flg);
19    }
```

Semafori: Esempio (semop - part 4)

```
1       /* Chiama la semop() e riporta il risultato */
2       if ((i = semop(semid, sops, nsops)) == -1) {
3         perror("semop: semop failed");
4       } else {
5         fprintf(stderr, "semop: valore di ritorno = %d\n", i);
6       }
7     }
8   }
9
10
11  int ask(int *semidp, struct sembuf **sopsp) {
12    static union semun arg;      /* argomento per semctl() */
13    static int nsops = 0;        /* dimensione del vettore di sembuf */
14    static int semid = -1;       /* semid del set di semafori */
15    static struct sembuf *sops; /* puntatore al vettore di sembuf */
16    int i;
17
18    if (semid < 0) {
19      /* Prima chiamata alla funzione ask()
20         Recuperiamo semid dall'utente e lo stato corrente del set di
21         semafori */
22      fprintf(stderr,
23              "Inserire semid del set di semafori su cui operare: ");
24      scanf("%d", &semid);
```

Semafori: Esempio (semop - part 5)

```
1     *semidp = semid;
2
3     arg.buf = &semid_ds;
4
5     /* chiamata a semctl() */
6     if (semctl(semid, 0, IPC_STAT, arg) == -1) {
7       perror("semop: semctl(IPC_STAT) fallita.");
8       /* Notare che se semctl() fallisce, semid_ds viene riempita
9          con 0, e successivi test per controllare il numero di
10         semafori ritorneranno 0.
11         Se invece semctl() va a buon fine, arg.buf verra'
12         riempito con le informazioni relative al set di semafori */
13
14      /* allocazione della memoria per un vettore di interi la cui
15         dimensione dipende dal numero di semafori inclusi nel set */
16    } else if ((arg.array = (ushort *)malloc(
17                    sizeof(ushort) * semid_ds.sem_nsems)) == NULL) {
18      fprintf(stderr, error_mesg1, semid_ds.sem_nsems);
19      exit(1);
20    }
21  }
```

Semafori: Esempio (semop - part 6)

```
1   /* Stampa i valori correnti dei semafori */
2   if (semid_ds.sem_nsems != 0) {
3     fprintf(stderr, "Ci sono %d semaphores.\n", semid_ds.sem_nsems);
4     /* Chiama la funzione semctl per recuperare i valori
5        di tutti i semafori del set. Nel caso di GETALL il secondo
6        argomento della semctl() viene ignorato e si utilizza il
7        campo array della union semun arg */
8     if (semctl(semid, 0, GETALL, arg) == -1) {
9       perror("semop: semctl(GETALL) fallita");
10    } else {
11      fprintf(stderr, "I valori correnti dei semafori sono:");
12      for (i = 0; i < semid_ds.sem_nsems; i++)
13        fprintf(stderr, " %d", arg.array[i]);
14      fprintf(stderr, "\n");
15    }
16  }
```

Semafori: Esempio (semop - part 7)

```
1   /* Allocazione dello spazio per le operazioni che l'utente
2      desidera eseguire */
3   fprintf(stderr,
4       "Quante operazioni vuoi eseguire con la prossima semop()?\n");
5   fprintf(stderr, "Inserire 0 or control-D per uscire: ");
6   i = 0;
7   if (scanf("%d", &i) == EOF || i == 0)
8     exit(0);
9   if (i > nsops) {
10    if (nsops != 0) /* libero la memoria precedentemente allocata */
11      free((char *)sops);
12    nsops = i;
13    /* Allocazione della memoria per le operazioni da eseguire*/
14    if ((sops = (struct sembuf *)malloc(
15                        (nsops * sizeof(struct sembuf)))) == NULL) {
16      fprintf(stderr, error_mesg2, nsops);
17      exit(2);
18    }
19  }
20  *sopsp = sops;
21  return (i);
22 }
```