Los semáforos en Unix

- Semáforos multi-valuados, no binarios

- Se puede crear un arreglo de semáforos con una simple llamada

- Posibilidad de especificar una sola operación en un solo semáforo

- Posibilidad de especificar un conjunto de operaciones en un arreglo de semáforos

- En UNIX, cuando un proceso aborta (*exit()*) libera todos los candados, (*locks*), que retenía en sus semáforos

- Los semáforos tienen dueños, modos de acceso y *timestamps*

- Librerías:

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

Particularidades de los semáforos Unix

• Petición de *m* recursos del conjunto { R_1, R_2..... R_m }

Cada uno protegido por un semáforo (S_i)

Acceso de *m* recursos : R_1, R_2..... R_m

$P(S_1)$ $P(S_2)$..... $P(S_m)$

Atomicidad no garantizada:

Posibilidad de bloqueo de un proceso y de que que éste bloquee a otro

• UNIX: posibilidad de realizar conjunto de operaciones sobre un conjunto de semáforos. *Si una operación no es posible, ninguna se realiza.*

• Adquisición de *m* recursos del mismo tipo:

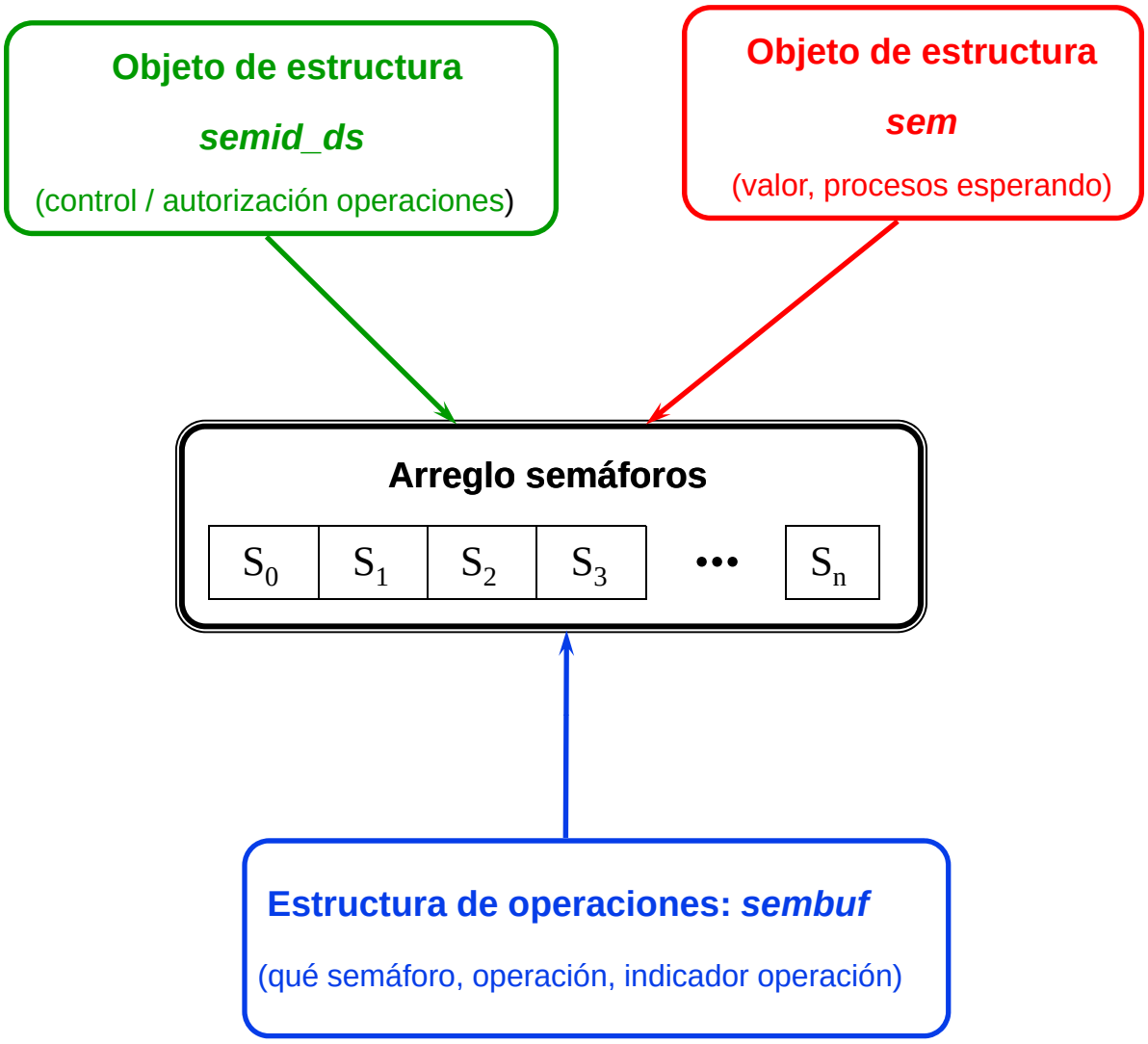
m operaciones P sucesivas del mismo semáforo

mismas consecuencias

• Despertar procesos en espera del valor de un semáforo:

UNIX despierta a *todos* los procesos que se encontraban en espera de un evento, dejando al administrador, que decida quien entra.

Estructuras datos semáforos



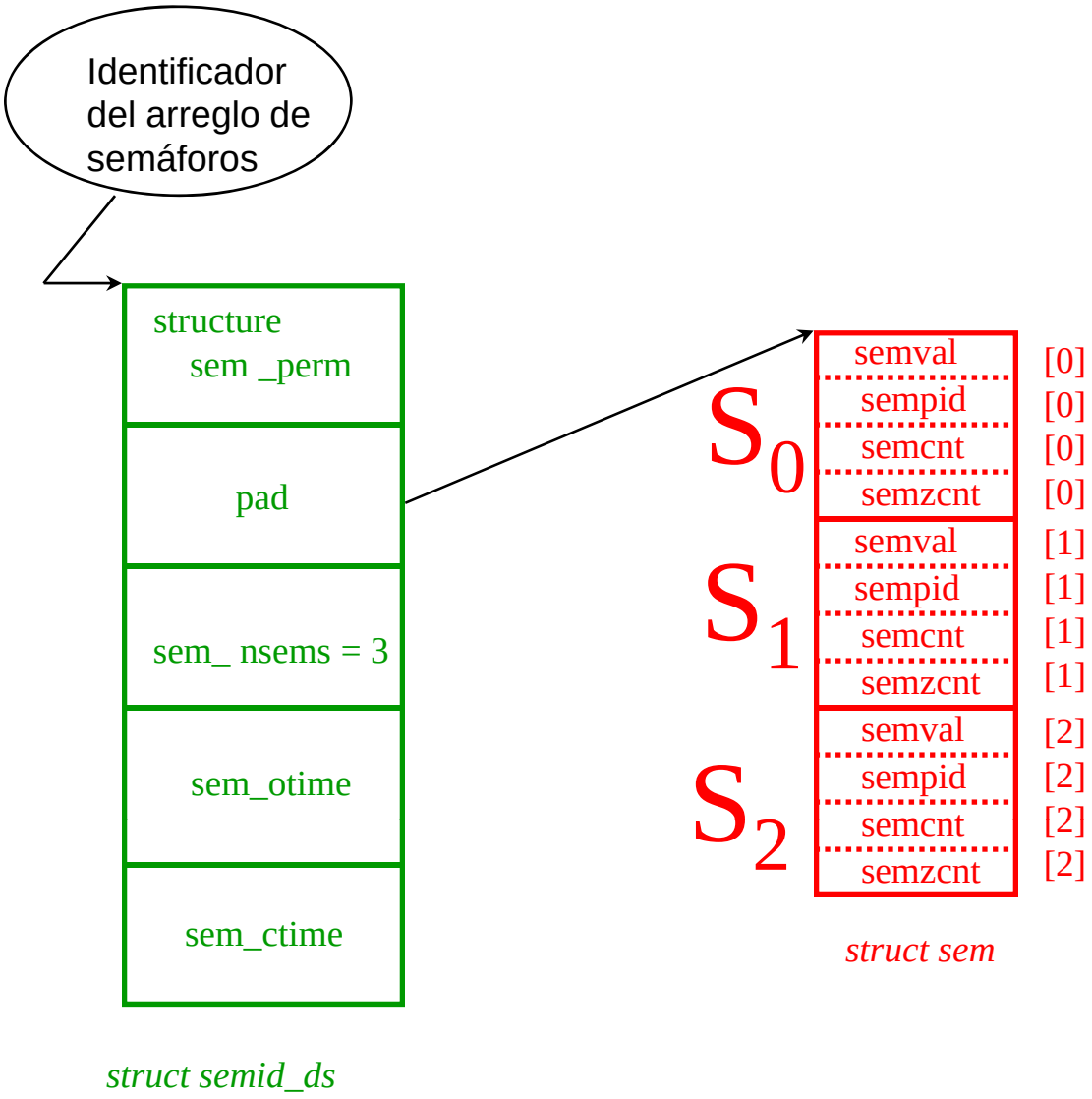
Estructuras datos semáforos

```
struct sem {
        ushort    semval        /* valor semáforo */
        short     sempid        /* último proceso que hizo un semop*/
        ushort    semcnt        /* # procesos esperan semval incremente su valor */
        ushort    semzcnt       /* # procesos esperan semval = 0 */
}
```

```
struct semid_ds {
   struct ipc_perm    sem_perm;          /* estructura permiso */
   int                *pad;              /* usado por sistema */
   ushort             sem_nsems;         /* número semáforos */
   time_t             sem_otime;         /* tiempo último semop */
   time_t             sem_ctime;         /* tiempo último cambio */
  }
```

```
struct sembuf{
  short   sem_num;     /* número semáforo */
  short   sem_op;      /* operación semáforo */
  short   sem_flg;     /* parámetro operación*/
  }
```

Ejemplo relación estructuras



semget(key, nsems, flag)

- Crea un arreglo de semáforos de tamaños *nsems*
- Obtiene un manejador de un conjunto de semáforos

id = semget(key, nsems, flag)

key_type key

llave de identificación del semáforo

int nsems

número de semáforos requerido en el segmento

int flag

si tiene el valor de:

IPC_CREAT	crea un semáforo con permisos especificados en perms:(IPC_CREAT | perms)
0	obtiene un manejador del semáforo existente

int id

manejador del semáforo

semop(id,ops,nops)

- Permite realizar atómicamente *nops* operaciones, definidas en ***ops*, sobre el arreglo de semáforos ubicado en *semid*
- Primitiva bloqueante

semop(id,ops,nops)

int id

manejador de semáforos, obtenido en semget()

struct sembuf **ops

definición de la operación a partir de la estructura:

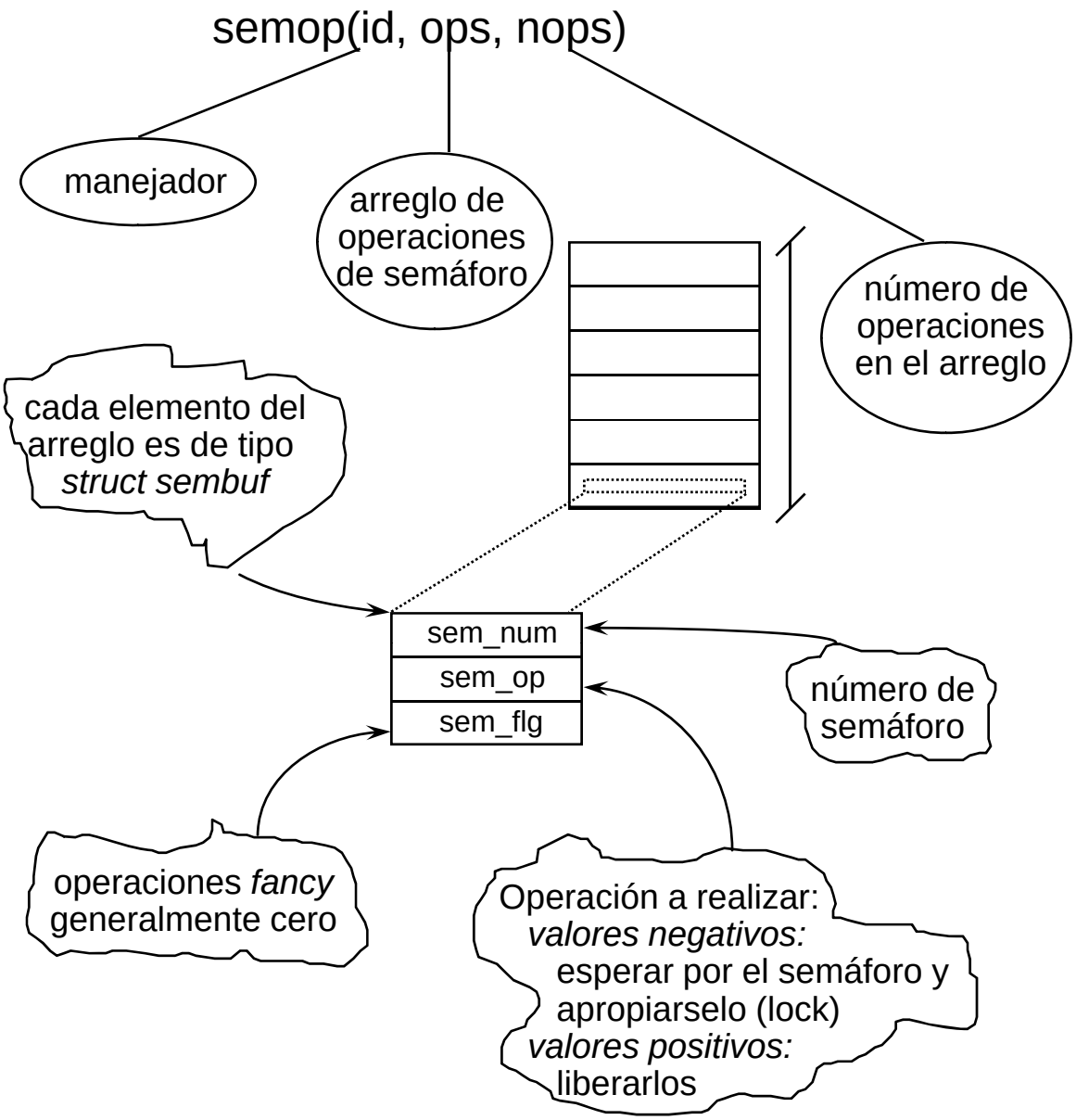
```
struct sembuf{
        short           sem_num;   /* número semáforo */
        short           sem_op;  /* operación semáforo */
        short           sem_flg;  /* parámetro operación */
 }
```

int nops

número de operaciones

semop(id, ops, nops)

+ Operaciones atómicas sobre semáforos



Naturaleza operaciones semop()

Operación elemental que engloba un número de semáforo y una operación. Esta puede tener 3 tipos de valor

1) Valor estrictamente *Negativo*

+ Especifica una operación tipo P(S)
+ Tiene por objeto disminuir el valor del semáforo(S)
+ Si la operación no es posible el proceso se duerme esperando que el valor del semáforo aumente.

2) Valor estrictamente *Positivo*

+ Especifica una operación tipo V(S)
+ Tiene por objeto aumentar el valor del semáforo(S)
+ Tiene el efecto de despertar todos los procesos en espera de que el valor del semáforo (S) aumente

3) Valor *0 (nulo)*

+ Permite probar si el valor del semáforo es 0
+ Si no es el caso el proceso es bloqueado

Definición operaciones en semop()

```
struct sem  {
  ushort        semval      /* val sem */
  short         sempid      /* último semop*/
  ushort        semcnt      /* incrementar  semal */
  ushort        semzcnt     /* semaval  = 0 */
}
```

```
struct sembuf {
  short sem_num; /* número semáforo */
  short sem_op;  /* operación semáforo */
  short sem_flg;  /* parámetro* oper. */
  }
```

if (*sem_op* < 0)

if (*semval* >= abs(*sem_op*) => *semval* = *semval* - *abs(sem_op)*

else

+ incrementar valor *semcnt* de la estructura *sem*

+ proceso bloqueado hasta que:

1) *semval* >= abs(*sem_op*)

2) arreglo semáforos desaparece

3) proceso recibe señal indica que *semcnt* es decrementada

if (*sem_op* > 0) => *semval* = *semval* + *sem_op*

(liberaración recursos que controlaba el semáforo)

if (*sem_op* = 0)

if (*semval* = 0) => regreso inmediato

else

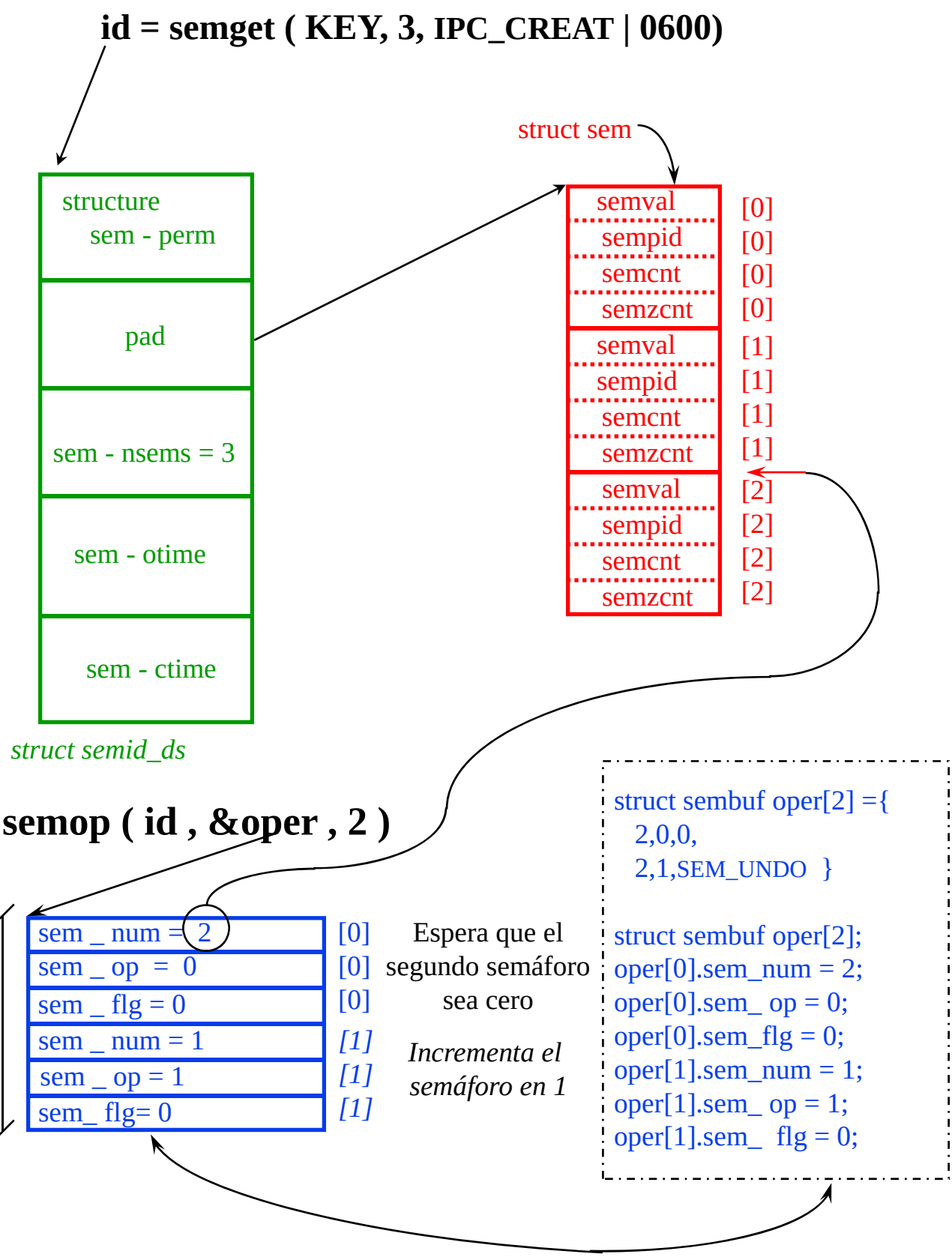
+ incrementar en 1 el campo *semzcnt* de la estructura sem

+ proceso suspendido hasta que:

1) *semval* = 0, (*semzcnt* es decrementado)

2) arreglo semáforos desaparece

3) proceso recibe señal


id = semget (KEY, 3, IPC_CREAT | 0600)
struct sem
structure
sem - perm
pad
sem - nsems = 3
sem - otime
sem - ctime
struct semid_ds
semval [0]
sempid [0]
semcnt [0]
semzcnt [0]
semval [1]
sempid [1]
semcnt [1]
semzcnt [1]
semval [2]
sempid [2]
semcnt [2]
semzcnt [2]
semop (id , &oper , 2)
ATOMICA
sem _ num = 2 [0]
sem _ op = 0 [0]
sem _ flg = 0 [0]
sem _ num = 1 [1]
sem _ op = 1 [1]
sem_ flg= 0 [1]
Espera que el segundo semáforo sea cero
Incrementa el semáforo en 1
struct sembuf oper[2] ={
2,0,0,
2,1,SEM_UNDO }
struct sembuf oper[2];
oper[0].sem_num = 2;
oper[0].sem_ op = 0;
oper[0].sem_flg = 0;
oper[1].sem_num = 1;
oper[1].sem_ op = 1;
oper[1].sem_ flg = 0;

Lock de un sémaforo P(S)

```
/* Poniendo un candado al semáforo 0
   Se asume que el arreglo solo cuenta con un semaforo */

int  P(s)
{
     int res;
     struct sembuf     sop;
          /* construcción arreglo operaciones de un elemento */
          sop.sem_num = n;
          sop.sem_op = -1;
          sop.sem_flg = 0;

          res = semop(sem, &sop, 1);
          if ( res < 0 )
               printf("Error en semop de P(S) \n");
               exit(1);
          }
          return res;
}
```

Unlock de un semáforo V(S)

```
/* Poniendo un candado al semáforo 0
   Se asume que el arreglo solo cuenta con un semaforo */

int  V(s)
{
     int res;
     struct sembuf     sop;
          /* construcción arreglo operaciones de un elemento */
          sop.sem_num = n;
          sop.sem_op = 1;
          sop.sem_flg = 0;

          res = semop(sem, &sop, 1);
          if ( res < 0 )
               printf("Error en semop de V(S) \n");
               exit(1);
          }
          return res;
}
```

semctl(id, num, cmd, arg)

- Permite realizar diferentes comandos, *cmd* , sobre un arreglo de semáforos apuntado por *id,* o sobre algun(os) semáforos del mismo arreglo.
- Inicialización, borrado, etc.

semctl(id, num, cmd, arg)

int id

manejador del semáforo

int num

especificación de los semáforos a procesar dentro del arreglo

int cmd

comando u operación a realizar

union semnum arg

argumento de la operación definida dentro de la unión *semnum*

El argumento de operación de semctl()

union semnum arg

arg es una unión definida de la siguiente forma:

```
union semnum {
        int             val;        /* usado en SETVAL */
        struct semid_ds *buf;       /* usado en IPC_STAT y IPC_SET*/
        ushort          *array;     /* usado en  GETALL y SETALL*/
}
```

la estructura de semid es:

```
        struct semid_ds {
            struct ipc_perm sem_perm;   /* estructura permiso */
            int             *pad;       /* usado por sistema */
            ushort          sem_nsems;  /* número semáforos */
            time_t          sem_otime;  /* tiempo último semop */
            time_t          sem_ctime;  /* tiempo último cambio */
        }
```

siendo la estructura del permiso:

```
        struct ipc_perm {
            ushort      uid;        /* id del usuario actual*/
            ushort      gid;        /* id del grupo actual */
            ushort      cuid;       /* id del usuario creador */
            ushort      cgid;       /* id del grupo creador */
            ushort      mode;       /* modo de acceso */
            short       pad1;       /* usado por el sistema*/
            long2       pad2;       /* usado por el sistema */
        }
```

Posibles operaciones en semctl()

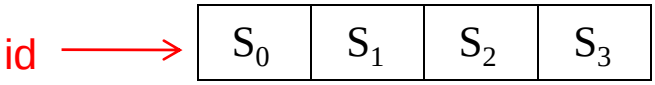
semctl(id, semnum, operacion, arg)

operación	semnum	Interpretación arg	Valor regreso (R), y efecto (E)
GETNCNT	Especificación de un número de semáforo	–	R: Número de procesos en espera de que el semáforo aumente su valor
GETZCNT	Especificación de un número de semáforo	–	R: Número de procesos en espera de que el semáforo tome un valor de cero
GETVAL	Especificación de un número de semáforo	–	R: Valor del semáforo
GETALL	Número, (cantidad), de semáforos	array	R: 0 si operación tuvo éxito y -1 si no. E: el arreglo contiene los valores de los "semnum" primeros semáforos
GETPID	Especificación de un número de semáforo	–	R: Identificación del último proceso que realizó una operación sobre el semáforo
SETVAL	Especificación de un número de semáforo	val	R: 0 si operación tuvo éxito y -1 si no. E: inicialización del semáforo a un valor dado
SETALL	Número, (cantidad), de semáforos	array	R: 0 si operación tuvo éxito y -1 si no. E: inicialización de los "sennum" primeros semáforos
IPC_STAT	_	buf	R: 0 si operación tuvo éxito y -1 si no. E: extracción estructura *semid_ds* de *id*, info es almacenada en *arg.buf*
IPC_SET	_	buf	R: 0 si la operación tuvo éxito y -1 si no E: modificación valores campos de la estructura de permisos de *semid_ds* de *id*, a partir de lo contenido en *arg.buf*
IPC_RMID	–	–	Supresión de la entrada en la tabla de semáforos

Ejemplo uso semctl()

Tomando en cuenta que se creó un arreglo de cuatro semáforos a través de la siguiente función:

id = semget(KEY, 4, IPCREAT | 0666)



r = semctl(id, 2, GETNCNT, 0)

Variable r contiene el número de procesos en espera de que el semáforo S_2 aumente su valor.

r = semctl(id, 1, GETZCNT, 0)

Variable r contiene el número de procesos en espera de que el semáforo S_1 sea igual a cero.

r = semctl(id, 3, GETVAL, 0)

Variable r contiene el valor del semáforo S_3 .

r = semctl(id, 2, GETALL, arg)

En arg.array se tienen los valores de semáforos S_0 S_1 y S_2 .

r = semctl(id, 3, SETVAL, arg)

Se asigna a S_3 el valor de arg.val

r = semctl(id, 2, SETALL, arg)

Se asigna a S_0 S_1 y S_2 el contenido de arg.array

r = semctl(id, 0, IPC_STAT arg)

En arg.buf se cuenta con los valores que describen al arreglo.

r = semctl(id, 0, IPC_SET, arg)

Se asigna a la descripción del semáforo los valores en arg.buf

r = semctl(id, 0, IPC_RMID, 0)

Borra el arreglo referenciado por id

Asignar un valor al semáforo

```
/* Asigna el valor x al semáforo 0 de sem
   Se asume que los arreglos son de tamaño 1, es decir que
   solo contienen un elemento */

  int asig_val.sem ( int sem,  int n )
{
      int r;
      union {
            int                 val;
            struct semid_ds     *buf;
            ushort              *array;
      } arg;

      arg.val = n;

      r = semctl(sem, 0, SETVAL, arg);
      if ( r < 0 )  {
          printf “Error en la asignación del valor \n”);
          exit(1);
      }
      return r;
}
```

Obtener el valor del semáforo

```
/* Regresa el valor del semáforo 0 de sem
   Se asume que los arreglos son de tamaño 1, es decir que
   solo contienen un elemento
*/


int   valsem_n ( int sem )
{

     int  r;

     r = semctl (sem,  0, GETVAL, 0);
     if ( r < 0 )  {
          printf “Error en la obtencion de valor \n”);
          exit(1);
     }
     return r;
}
```