Sistemas Operativos

Semáforos

Se necesitan para

- Sincronizar Procesos
- Exclusión mutua
- Zonas críticas
- Abstracción de recursos

- Semáforo: una variable que permite o no el paso
- Sem = 0 disponible Sem >< 0 ocupado
- Una cola asociada

Una aproximación

- CIERRE(X)
- EXAM:
- If X = Ocupado
  Go to EXAM;
- X := 1;

APERTURA(X)
X = 0;

- Loop de uso procesador – Inseguro por interrupción

Modificaciones

- Test&Set (x)

- Wait (x) (duerme proceso en cola asociada) (FIFO)
- Signal (x) (depierta proceso de la cola asociada) (¿dónde lo pone?)

Nueva aproximación

CIERRE(X)	APERTURA(X)
EXAM: T.S.(X)	X = 0
If X = Ocupado	
then WAIT(X)	**SIGNAL(X)**
Go to EXAM	

- ¿por qué volver EXAM?

Dijkstra (semáforos contadores)

- P (x)

  x = x - 1 si x $<$ 0, wait (x)

- V(x)

  x = x + 1 si x $\leq$ 0, signal (x)

Modelo de Exclusión

- Init sem x = 1

P(x)

V(x)

¿cómo incide en la administración de procesos?

Modelo Productor Consumidor

- Caso del Productor-Consumidor

Modelo P/C un mensaje

- Init Espacio = 1 Mensaje = 0

- <u>EMISOR</u> <u>RECEPTOR</u>

  P(Espacio) P(Mensaje)

  . .

  Coloca inf. Retira inf.

  . .

  V(Mensaje) V(Espacio)

Modelo P/C con más de un espacio (emisor)

- E

  P(Espacio)

  VEC(i) = M

  i = (i+1) módulo 3

  V(Mensaje)

  Mensaje = 0

  Espacio = 3

  i = 0

Modelo P/C con más de un espacio (consumidor)

- R

  P(Mensaje) j = 0

  X = VEC(j)

  j = (j+1) módulo 3

  V(Espacio)

Problemas

- Siempre que hay cambio de información, por datos o por estructuras, es necesario excluir (serializar)
- Problemas de las exclusiones

Problemas

- **Productor**

  P(Exclu)

  P(Espacio)

  VEC(i) = M

  i = (i+1) módulo 3

  V(Mensaje)

  V(Exclu)

Problemas

- **Consumidor**

  P(Exclu)

  P(Mensaje)

  X = VEC(j)

  j = (j+1) módulo 3

  V(Espacio)

  V(Exclu)

Solución

- Productor

  P(Espacio)

  P(Exclu)

  VEC(i) = M

  i = (i+1) módulo 3

  V(Exclu)

  V(Mensaje)

Solución

- Consumidor

  P(Mensaje)

  P(Exclu)

  X = VEC(j)

  j = (j+1) módulo 3

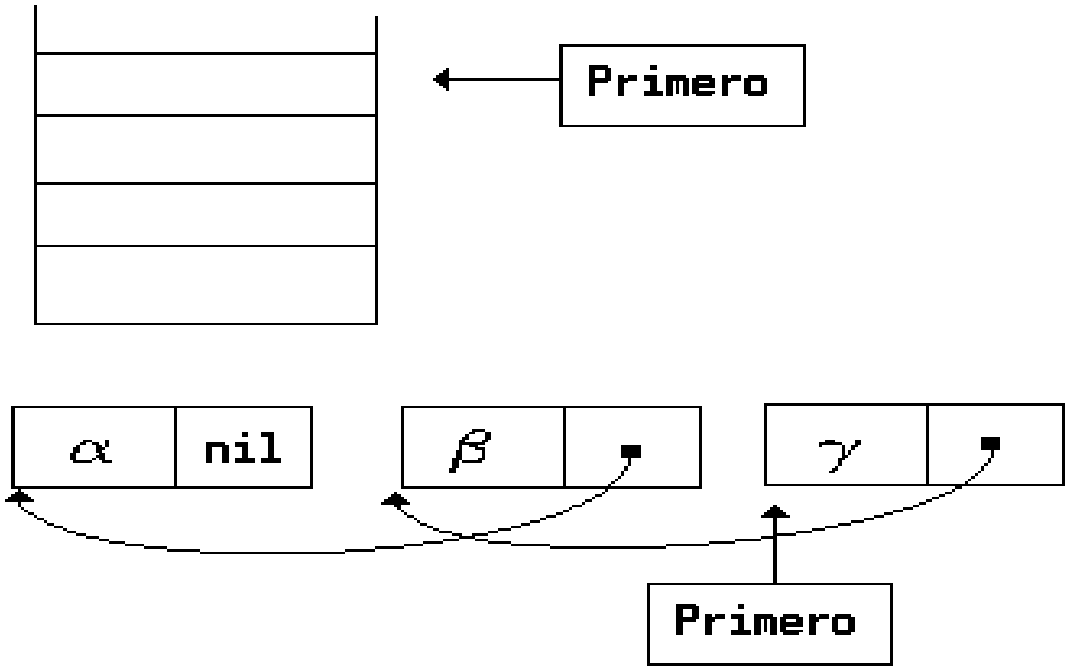
  V(Exclu)

  V(Espacio)

Implementado con stack

- E (semáforo) = MAX = 3 (cuenta el máximo de mensajes producibles)
- S (semáforo) = 0 (cuenta el número de mensajes disponibles para consumo)
- X (semáforo de exclusión) = 1

Stack



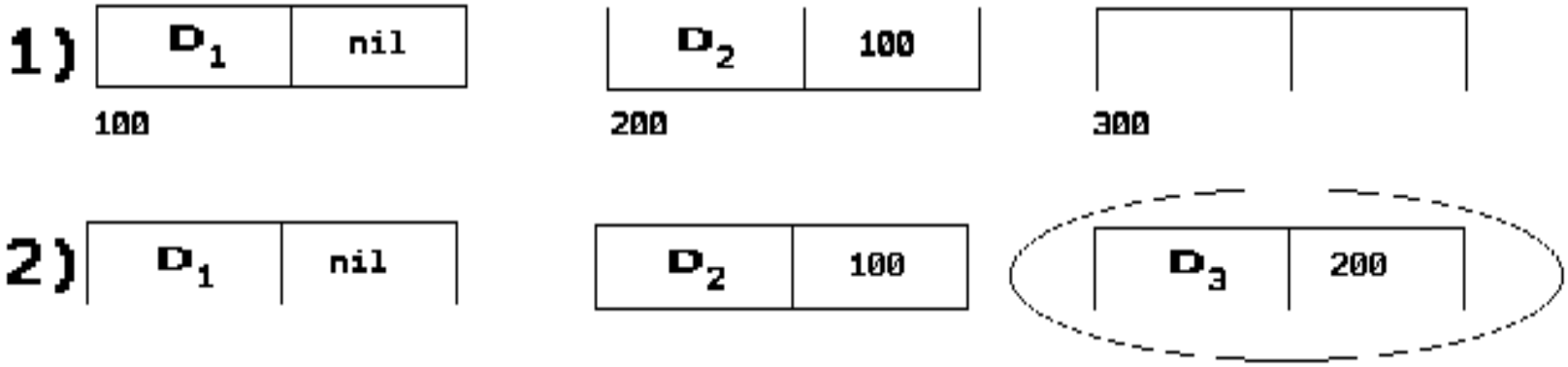
Prod/Cons Implementación Stack con exclusión bien puesta

- Las rutinas serán :

A (Productor)
P(E)
P(X)
[1] P = genera apuntador
P.Mensaje = Dato
[3] P. Próx = Primero
[4] Primero = P
V(X)
V(S)

B (Consumidor)
P(S)
P(X)
[2] C = Primero
[5] Primero=C.Próximo
Dato = C.Mensaje
Libera C
V(X)
V(E)

Si suponemos sin exclusión



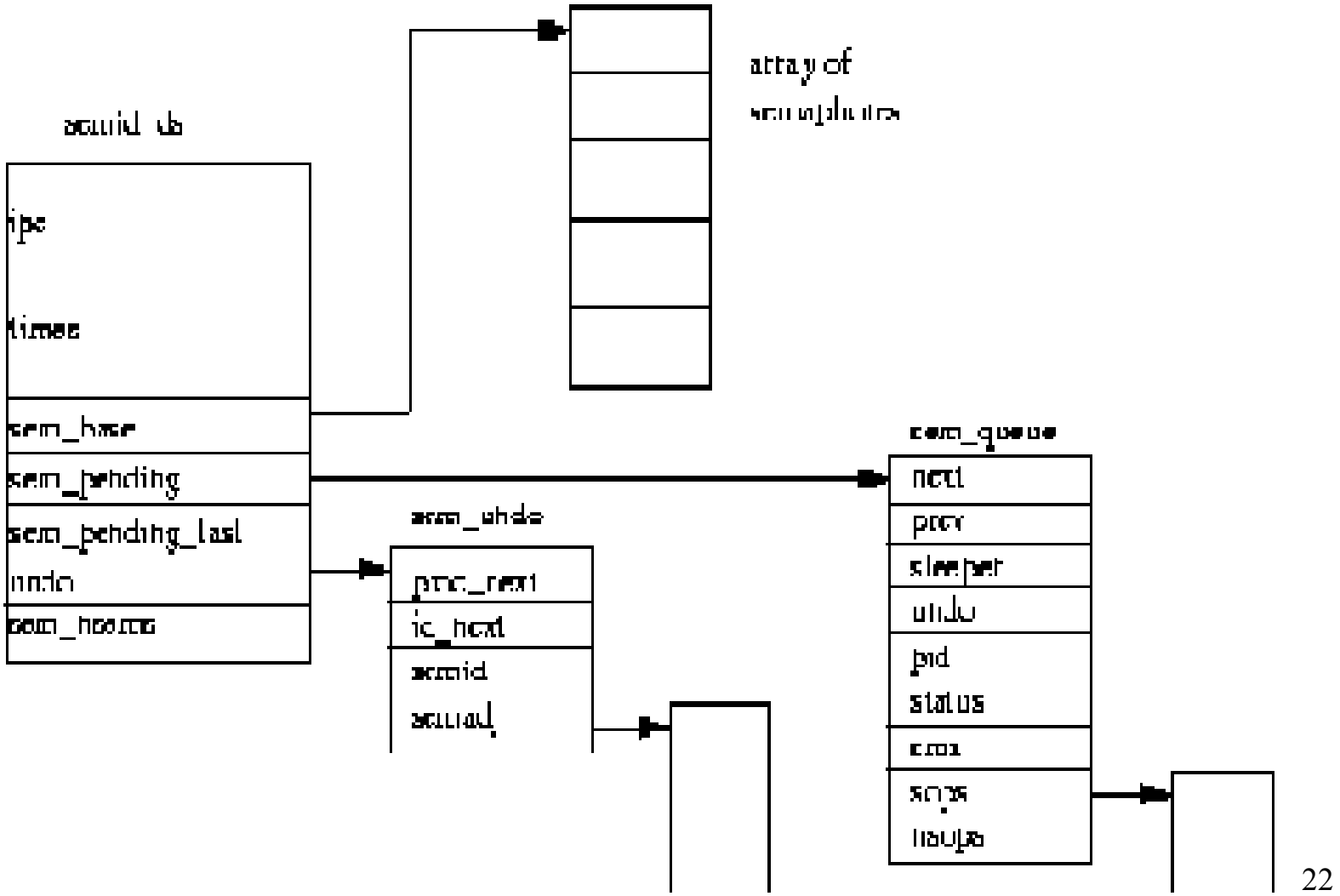
	Valores iniciales	1) A	B	A	B	A	Valores finales
Primero	nil	100	200			300	100		100
E	3	2	1	0			1		1
S	0	1	2		1			2	2
P	nil	100	200	300					
C	nil				200				

2)

Estudio de casos

- 0) A prod B Cons sec. ABBABB...
- 1) A Prod B y C Cons sec. ABCABC...
- 2) idem sec. A (BC) o A (CB)...
- 3) idem sec. ABCACBABCACB...

Implementación



Implementación

```
/* One semid data structure for each set of semaphores in the system. */
struct semid_ds {
    struct ipc_perm sem_perm;       /* permissions .. see ipc.h */
    time_t sem_otime;               /* last semop time */
    time_t sem_ctime;               /* last change time */
    struct sem *sem_base;           /* ptr to first semaphore in array */
    struct wait_queue *eventn;
    struct wait_queue *eventz;
    struct sem_undo *undo;          /* undo requests on this array */
    ushort sem_nsems;               /* no. of semaphores in array */
};
```

Implementación

- **sem_perm**
  - This is an instance of the ipc_perm structure, which is defined for us in linux/ipc.h. This holds the permission information for the semaphore set, including the access permissions, and information about the creator of the set (uid, etc).
- **sem_otime**
  - Time of the last semop() operation (more on this in a moment)
- **sem_ctime**
  - Time of the last change to this structure (mode change, etc)
- **sem_base**
  - Pointer to the first semaphore in the array (see next structure)
- **sem_undo**
  - Number of *undo* requests in this array (more on this in a moment)
- **sem_nsems**
  - Number of semaphores in the semaphore set (the array)

Implementación IPC (crea semáforo)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
#define KEY (1492)
void main()
{   int id; /* Number by which the semaphore is known within a program */
    ...
    id = semget(KEY, 1, 0666 | IPC_CREAT);
}
```

Implementación IPC (operación V)

```
#include <sys/types.h> ...
#define KEY (1492)
void main()
{    int id;
     id = semget(KEY, 1, 0666);
/* Which semaphore in the semaphore array : */
      operations[0].sem_num = 0;
/* Which operation? Add 1 to semaphore value : */
      operations[0].sem_op = 1;
 /* Set the flag so we will wait : */
      operations[0].sem_flg = 0;
/* So do the operation! */
     retval = semop(id, operations, 1);  (sobre retval se pregunta)
}
```

Implementación IPC (operación P)

```
#include <sys/types.h> ...
#define KEY (1492)
void main()
{    int id;
     id = semget(KEY, 1, 0666);
/* Which semaphore in the semaphore array : */
      operations[0].sem_num = 0;
/* Which operation? Substract 1 to semaphore value : */
      operations[0].sem_op = -1;
 /* Set the flag so we will wait : */
      operations[0].sem_flg = 0;
/* So do the operation! */
     retval = semop(id, operations, 1);  (sobre retval se pregunta)
}
```

Implementación POSIX

- ***P1003.1b***

Function	**Description**
`sem_close`	Deallocates the specified named semaphore
`sem_destroy`	Destroys an unnamed semaphore
`sem_getvalue`	Gets the value of a specified semaphore
`sem_init`	Initializes an unnamed semaphore
`sem_open`	Opens/creates a named semaphore for use by a process
`sem_post`	Unlocks a locked semaphore
`sem_trywait`	Performs a semaphore lock on a semaphore only if it can lock the semaphore without waiting for another process to unlock it
`sem_unlink`	Removes a specified named semaphore
`sem_wait`	Performs a semaphore lock on a semaphore

Implementación POSIX

- The following example creates a semaphore named `/tmp/mysem` with a value of 3

```
#include <sys/types.h>
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>

#include <sys/mman.h>
#include <semaphore.h>

#include <sys/stat.h>
```

Implementación POSIX

```
sem_t   *mysemp;
int     oflag = O_CREAT;
mode_t  mode = 0644;
const char semname[] = "/tmp/mysem"unsigned int value = 3;
int     sts;
    …
    mysemp = sem_open(semname, oflag, mode, value);
    if (mysemp == (void *)-1) {  perror(sem_open() " failed ");
}
```

- To access a previously created semaphore, a process must call the `sem_open` function using the name of the semaphore.