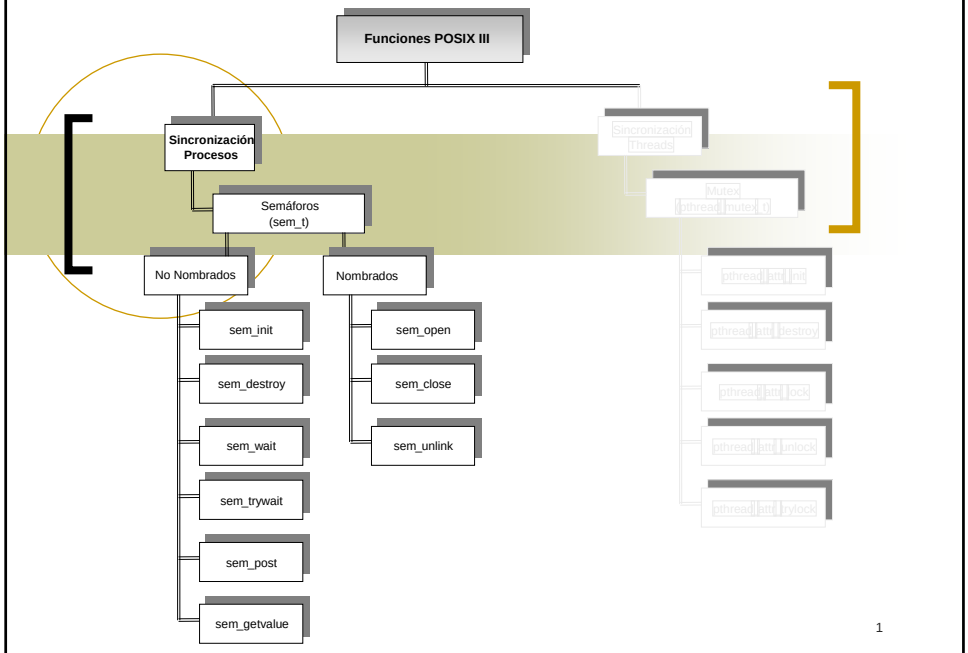


Semáforos POSIX (I)

- Pertenecen al estándar POSIX.1b y son relativamente recientes (1993)
- Variable de tipo *sem_t* sobre la cual se pueden realizar las funciones clásicas de los semáforos
- POSIX define dos tipos
  - **Semáforos nombrados**: se puede utilizar por el proceso que lo crea y por cualquier otro aunque no tenga relación con el creador
  - **Semáforos no nombrados**: se puede utilizar por el proceso que lo crea (threads). El uso desde otros procesos precisa de una región de memoria compartida.

```
#include <semaphore.h>
sem_t *semaforo;       //nombrados
sem_t semaforo;        // no nombrados
```

- Librerías:
  `-lposix4` **`-lrt`** **(linux)**

Semáforos POSIX (II)

- POSIX no especifica la naturaleza del tipo *sem_t*, basta con saber que almacena toda la información referente al semáforo
- Todas las llamadas de creación y manejo de semáforos devuelven -1 en caso de error y 0 (normalmente) en otro caso
- La única diferencia entre semáforos nombrados y no nombrado es la forma de crear y destruir el semáforo.
- Los semáforos POSIX son semáforos contadores que tienen valores no negativos y deben ser inicializados antes de ser usados
- Para ver los semáforos activos:

```
ipcs -sa
```


Operaciones con Semáforos (I)

- *sem_wait*: operación P sobre un semáforo

```
#include <semaphore.h>
int sem_wait (sem_t *sem)
```

  - Decrementa el contador del semáforo
  - Si el valor previo del semáforo es 0, *sem_wait* efectúa un bloqueo del proceso llamante hasta que el semáforo sea incrementado
  - Devuelve 0 si el semáforo pudo ser decrementado y -1 en caso de error
- *sem_trywait*: operación P no bloqueante
  - Si el valor previo del semáforo es 0 en lugar de bloquear al proceso, no hace nada y devuelve -1

```
#include <semaphore.h>
int sem_trywait (sem_t *sem)
```

Operaciones con Semáforos (II)

- *sem_post*: incrementa el valor del contador del semáforo (operación V)
  - Devuelve 0 si el semáforo pudo ser incrementado y -1 en caso de error

```
#include <semaphore.h>
int sem_post(sem_t *sem)
```

- *sem_getvalue*: obtiene en *sval* el valor del contador del semáforo

```
#include <semaphore.h>
int sem_getvalue (sem_t *sem, int *sval);
```


Semáforos no nombrados (I)

- Inicialización: la inicialización de un semáforo no nombrado en POSIX se realiza con la función *sem_init*

```
#include <semaphore.h>
int sem_init (sem_t *sem, int pshared, unsigned int value);
```

  - Inicializa *sem* para que contenga el valor *value* (>=0).
  - *value* indica el número de procesos que puede acceder al recurso que asociado al semáforo
  - *pshared :*
    - 0 indica que el semáforo sólo se puede utilizar por el proceso que lo crea (threads);
    - si es distinto de cero, el semáforo puede ser utilizado por cualquier proceso siempre que la estructura *sem* resida en una región de memoria compartida
  - Devuelve 0 si el semáforo pudo ser inicializado y -1 en caso de error

Semáforos no nombrados (II)

- *sem_destroy* : destruye un semáforo

```
#include <semaphore.h>
int sem_destroy (sem_t *sem)
```

  - Si un proceso intenta destruir un semáforo que tiene a otro proceso en espera, *sem_destroy* devuelve un mensaje de error
  - Devuelve 0 si el semáforo pudo ser liberado y -1 en caso de error

Semáforos nombrados (I)

- Sincronizan procesos que no tienen ninguna relación entre sí.
- Los semáforos nombrados poseen:
  - Nombre
  - Identificador de usuario
  - Identificador de grupo
  - Permisos de acceso
- El nombre de un semáforo es una cadena de caracteres (con las mismas restricciones de un nombre de fichero). Además
  - Si el nombre (ruta) es relativa, sólo puede acceder al semáforo el proceso que lo crea y sus hijos
  - Si el nombre comienza por "/", el semáforo puede ser compartido por cualquier proceso

Semáforos nombrados (II)

- *sem_open*: devuelve un identificador que es utilizado por *sem_wait, sem_trywait, sem_post* y *sem_getvalue* para referirse a dicho semáforo

```
#include <semaphore.h>
sem_t *sem_open (const char *name, int oflag);
sem_t *sem_open (const char *name, int oflag,
mode_t mode, unsigned int value);
```

- *oflag* determina si se accede a un semáforo ya creado o se debe crear uno nuevo
  - Si *oflag* es 0 se indica que se quiere acceder a un semáforo creado (si no existe, *sem_open* devuelve -1)
  - Si *oflag* tiene el valor de O_CREAT, o O_CREAT | O_EXECL se utiliza la segunda sintaxis presentada arriba

Semáforos nombrados (II)

  - Según *oflag*
    - **O_CREAT**: *sem_open* crea un semáforo si no existe. Si existe, accede al semáforo e ignora los parámetros restantes
    - **O_CREAT | O_EXECL**: *sem_open* crea un semáforo si no existe. Si existe se devuelve -1
  - En los casos necesarios, *mode* especifica los permisos con los que se crea el semáforo (iguales que en *open*) y *value* especifica el valor inicial del semáforo creado

- *sem_close*: libera los recursos que el sistema asigna a un proceso cuando trabaja con un determinado semáforo

```
#include <semaphore.h>
int sem_close (sem_t *sem);
```

  - No necesariamente elimina el semáforo, sólo lo hace inaccesible al proceso

Semáforos nombrados (III)

- *sem_unlink*: elimina un semáforo nombrado

```
#include <semaphore.h>
int sem_unlink (sem_t *sem);
```

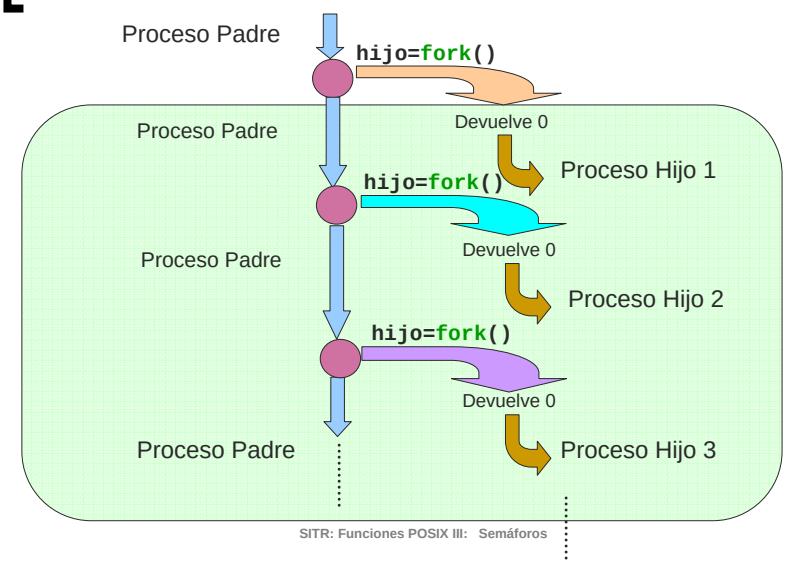
  - Si otros procesos hacen referencia al semáforo, *sem_unlink* pospone la destrucción.
  - Las llamadas *sem_open* con el mismo nombre de semáforo después de hacer *sem_unlink* se refieren a otro semáforo (aun cuando aun haya procesos utilizando el semáforo)

EJEMPLO: Sección Crítica

Semáforos Nombrados

- El Padre crea un semáforo no nombrado y n-1 procesos
- Cada proceso (padre e hijos) muestra una cadena carácter a carácter liberando la CPU entre cada carácter (*sched_yield()*)
- La sección crítica es la el uso de la salida por pantalla para mostrar la cadena.
- El proceso padre espera la terminación de los hijos y libera el semáforo

Creación de Procesos




```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <semaphore.h>
#include <sched.h>
#include <fcntl.h>

#define BUFFER_SIZE             500
#define NUM_PROCESOS            10
char buffer[BUFFER_SIZE];

int main (int argc, char *argv[ ])
{
        char *c;
        int i, n= NUM_PROCESOS;
        pid_t hijo;
        sem_t *semaforo;

        /* Crea el semáfono nombrado */
        if((sem_cont=sem_open("/semaforo", O_CREAT, 0644, 1))==(sem_t *)-1)
        {   perror("No se puede crear el semáforo"); exit(1); }

        /* Crea los procesos */
        for (i = 1; i<n; ++i)
        {   hijo = fork();
            if (hijo ==-1) { perror("No se puede crear el proceso"); exit(-1);}
            else if(hijo==0)  break; // si es un hijo terminar el for (solo el padre crea los hijos)
        }
        /* i actúa como identificador del proceso en el padre i==n */

        sprintf( buffer, "i:%d  proceso ID:%d  padre ID:%d\n",i,
                                getpid(), getppid());
        c = buffer;
// -> Continua
```

Id Semáforo

Valor Inicial

Bifurcación

```
/******** Sección de entrada ****/
if (sem_wait(semaforo) == -1)
{
        perror("Semáforo no válido");
        exit(1);
}
/**** Principio de la sección crítica *******/
while (*c != '\0')
{
        fputc(*c, stderr); fflush(stderr);
        c++;
        sched_yield(); /* libera la CPU */
}
/********** Fin de la sección crítica *******/

/*********** Sección de salida ***********/
if (sem_post(semaforo) == -1)
{
        perror("Semáforo no válido");
        exit(1);
}
/********* Sección Restante ***********/
sem_close(&semaforo)
// El Padre espera que terminen los hijos
if(i==n)
{  printf("Soy el Padre [PID:%d] y voy a esperar a los hijos\n", getpid());
   for(i=1;i<n;i++)  wait(0);
  sem_unlink("/semaforo");
}
exit(0);
}
```


Ejemplo Sección Crítica

- Con semáforos

```
i:1  proceso ID:18818  padre ID:18817
i:2  proceso ID:18819  padre ID:18817
i:3  proceso ID:18820  padre ID:18817
i:4  proceso ID:18821  padre ID:18817
i:5  proceso ID:18822  padre ID:18817
i:6  proceso ID:18823  padre ID:18817
i:7  proceso ID:18824  padre ID:18817
i:8  proceso ID:18825  padre ID:18817
i:9  proceso ID:18826  padre ID:18817
i:10  proceso ID:18817  padre ID:17106

Soy el Padre [PID:18817] y voy a esperar que terminen todos los hijos
```

- Sin semáforos

```
iiiiiiiii:::::::::3126154789     0             ppppp pppprrrrprrrrrooooorooooo
ccccoccccceeeeeceeeeessssessssoooooosoooo    o     IIIII IIIIDDDDIDDDDD:::::D::::
1111:1111188888188889999899999333339344353483769O1     2             ppppp pppp
aaaapaaaaaddddddadddrrrrdrrrrreeeeereeee    e     IIIII IIIIDDDDIDDDDD:::::D::::
1111:1111188888188889999799999333331333322222022222

6

Soy el Padre [PID:18932] y voy a esperar que terminen todos los hijos
```

EJEMPLO: Secuenciación de Tareas

Semáforos Nombrados

- Crear dos procesos (padre e hijo).
- El proceso padre debe mostrar los números impares del 1 al 20.
- El proceso hijo debe mostrar los números pares del 1 al 20.
- Sincronizar la ejecución concurrente de padre e hijo para que se impriman los números del 1 al 20 de forma correlativa.

EJEMPLO: Secuenciación de Tareas

Solución

- Asociaremos un semáforo a cada proceso
- En la sección de entrada cada proceso realizará una operación **P(S)** (bloqueo) sobre su semáforo.
- En la sección de salida cada proceso liberará **V(S)** el semáforo del otro proceso.
- El padre iniciará la ejecución mientras el hijo está suspendido (esto se consigue inicializando los semáforos forma correcta).

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <semaphore.h>
#include <errno.h>
#include <fcntl.h>

sem_t *sem_1, *sem_2; // declaramos un puntero para el identificador de los semáforos

int main(void)
{
        int i;
        pid_t hijo;
        int val;

        printf("Creando semáforos .....\n");

        /* comprueba si ya existe el semáforo  y sino lo crea desbloqueado (1)*/
        sem_1=sem_open ("/sem_1", O_CREAT, 0644, 1);
        if(sem_1==(sem_t *)-1)
                perror("Error creando semáforo 1");

        /* comprueba si ya existe el semáforo  y sino lo crea ya bloqueado (0)*/
        sem_2=sem_open ("/sem_2", O_CREAT, 0644,  0);
        if(sem_2==(sem_t *)-1)
                perror("Error creando semáforo 2");

// Continua ->
```

Padre
Valor Inicial 1
(Desbloqueado)

Hijo
Valor Inicial 0
(Bloqueado)


```
        printf("Creando proceso hijo .....\n");
        hijo=fork() ;
        switch(hijo)
        {
             case -1:
                        printf("error creando proceso hijo\n");
                                exit(0);

            case 0: /* estamos en el hijo, valores pares */
                        printf("Soy el hijo con PID:%d\n", getpid());
                        for (i=2;i<=20;i+=2)
                        {
                           sem_wait (sem_2);   /* espera que el padre libere el semáforo */
                              printf("hijo  valor:%d\n",i);
                              sleep(1);
                           sem_post (sem_1); /* activa al padre */
                        }
                        /* libero semáforos */
                        sem_close (sem_1);
                        sem_close (sem_2);
                        printf("Soy el hijo y termino.....\n");
                        break;

        // Continua ->
```

Bifurcación

```
        default: /*estamos en el padre, valores impares */
                printf("Soy el padre con PID:%d\n", getpid());
                for(i=1;i<=20;i+=2)
                {
                  sem_wait(sem_1);    /* espera que el hijo libere el semáforo */
                    printf("padre valor:%d\n",i);
                    sleep(1);
                  sem_post(sem_2); /* activa al hijo */
                }

                /* libero semáforos */
                sem_close(sem_1);
                sem_close(sem_2);

                printf("Soy el padre espero que termine el hijo .....\n");
                wait(0); /* Esperar que acabe el hijo */

                printf("Soy el padre destruyo los semáforos y termino.....\n");
                sem_unlink("/sem_2");
                sem_unlink("/sem_1");
        }
        exit(0);
}
```


Ejemplo: Secuenciación de Tareas

- Con semáforos

```
[/u0/sitr/sitr001/procesos]sem3
Creando semáforos .....
Creando proceso hijo .....
Soy el hijo con PID:28685
Soy el padre con PID:28684
padre valor:1
hijo  valor:2
padre valor:3
hijo  valor:4
padre valor:5
hijo  valor:6
padre valor:7
hijo  valor:8
padre valor:9
hijo  valor:10
padre valor:11
hijo  valor:12
padre valor:13
hijo  valor:14
padre valor:15
hijo  valor:16
padre valor:17
hijo  valor:18
padre valor:19
Soy el padre espero que termine el hijo .....
hijo  valor:20
Soy el hijo y termino.....
Soy el padre destruyo los semáforos y termino.....
[/u0/sitr/sitr001/procesos]
```

- Sin semáforos

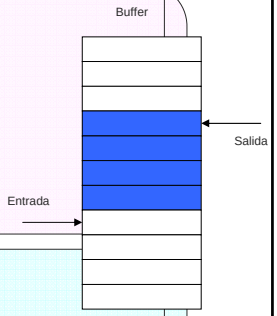
```
[/u0/sitr/sitr001/procesos]sem3r
Creando semáforos .....
Creando proceso hijo .....
Soy el hijo con PID:28187
hijo  valor:2
Soy el padre con PID:28186
padre valor:1
padre valor:3
hijo  valor:4
hijo  valor:6
padre valor:5
padre valor:7
hijo  valor:8
hijo  valor:10
padre valor:9
padre valor:11
hijo  valor:12
hijo  valor:14
padre valor:13
padre valor:15
hijo  valor:16
hijo  valor:18
padre valor:17
padre valor:19
hijo  valor:20
Soy el hijo y termino.....
Soy el padre espero que termine el hijo .....
Soy el padre destruyo los semáforos y termino.....
[/u0/sitr/sitr001/procesos]
```

EJEMPLO:

Productor-Consumidor


```
// Problema productor - consumidor
type item=.....;
var          buffer:array[0..n-1] of item
entrada, salida: 0..n-1;
contador: 0..n
entrada=0; salida=0; contador=0;
function productor;
          repeat
                    .....
                    // produce un item en la variable nuevo
                    .....
                    while contador==n do no-op
                    buffer[entrada]=nuevo;
                    entrada=(entrada+1)mod n;
                    contador=contador+1;
          until false
end productor

function consumidor;
          repeat
                    while contador==0 do no-op
                    nuevo=buffer[salida];
                    salida=(salida+1)mod n;
                    contador=contador-1;
                    .....
                    // consume el item almacenado en nuevo
                    .....
          until false
end consumidor
```



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <semaphore.h>
#include <sched.h>
#include <fcntl.h>

#define BUFFER_SIZE     10      //  tamaño del buffer
#define CICLOS          10      // numero de ciclos de ejecución

int buffer[BUFFER_SIZE];
sem_t *sem_cont, *sem_free;     // puntero para el identificador de los semáforos

int main (int argc, char *argv[])
{
        int i;
        pid_t hijo;
        int val;
        int entrada, salida;    // índices a las zonas de inserción y extracción
        entrada=salida=0;       // inicialización

        printf("Creando semáforos .....\n");
        /* semáforo del contador de productos sino lo crea inicializado(0)*/
        if((sem_cont=sem_open("/sem_cont", O_CREAT, 0644, 0))==(sem_t *)-1)
                perror("Error creando semáforo 1");

        /* semáforo del espacio libre y sino lo crea inicializado (BUFFER_SIZE)*/
        if((sem_free=sem_open("/sem_free", O_CREAT, 0644, BUFFER_SIZE))==
                                                        (sem_t *)-1)
                perror("Error creando semáforo 2");
// Continua ->
```

Contador Valor Inicial 0

Espacio Libre Valor Inicial BUFFER_SIZE


```
hijo=fork();
switch(hijo)
{
   case -1:
        printf("error creando proceso hijo\n");
        sem_unlink("/sem_cont");
        sem_unlink("/sem_free");
        exit(0);

   case 0:  /* estamos en el hijo -> consumidor */
        printf("Soy el hijo (consumidor) con PID:%d\n", getpid());
        sleep(1);

        for (i=0;i<=CICLOS;i++)
        {
          sem_wait(sem_cont);     /* espera datos en el buffer (contador>0) y dec.*/
                buffer[salida] = 0;         // consume un elemento
                salida= (salida+1) % BUFFER_SIZE;               // buffer circular
          sem_post(sem_free); /* incrementa el contador de espacio */

          sem_getvalue(sem_cont,&val); // valor del contador del semáforo
          printf("Soy el Consumidor: salida=%d, Estado %d productos\n", salida, val);
          sleep(2);
        }
        /* libero semáforos */
        sem_close( sem_cont );
        sem_close( sem_free );
        printf("Soy el hijo y termino.....\n");
      break;
```

Bifurcación

Espera que haya productos y consume un elemento

Incrementa el contador libre

```
default:    /*estamos en el padre-> productor */
            printf("Soy el padre (productor) con PID:%d\n", getpid());
            sleep(1);

            for (i=0;i<=CICLOS;i++)
            {
                sem_wait(sem_free); /* espera que haya espacio en el buffer y decrementa */
                        buffer[entrada] = i; // produce un elemento
                        entrada= (entrada+1) % BUFFER_SIZE;          // buffer circular
                sem_post(sem_cont); /* incrementa el contador del semáforo */

                sem_getvalue(sem_cont,&val); // valor del contador del semáforo
                printf("Soy el Productor: entrada=%d, Estado %d productos\n", entrada, val);
                sleep(1);
            }

                /* libero semáforos */
                sem_close(sem_cont);
                sem_close(sem_free);

                printf("Soy el padre espero que termine el hijo .....\n");
                wait(0); /* Esperar que acabe el hijo */

                printf("Soy el padre destruyo los semáforos y termino.....\n");
                sem_unlink("/sem_cont");
                sem_unlink("/sem_free");
    }
    exit(0);
}
```

Incrementa el contador de productos

Espera que haya espacio libre

Ejemplo: Productor - Consumidor

```
Creando semaforos .....
Creando proceso hijo .....
Soy el hijo (consumidor) con PID:18408
Soy el padre (productor) con PID:18407
Soy el Productor: entrada=1, Estado 1 productos
Soy el Consumidor: salida=1, Estado 0 productos
Soy el Productor: entrada=2, Estado 1 productos
Soy el Consumidor: salida=2, Estado 0 productos
Soy el Productor: entrada=3, Estado 1 productos
Soy el Productor: entrada=4, Estado 2 productos
Soy el Consumidor: salida=3, Estado 1 productos
Soy el Productor: entrada=5, Estado 2 productos
Soy el Productor: entrada=6, Estado 3 productos
Soy el Consumidor: salida=4, Estado 2 productos
Soy el Productor: entrada=7, Estado 3 productos
Soy el Productor: entrada=8, Estado 4 productos
Soy el Consumidor: salida=5, Estado 3 productos
Soy el Productor: entrada=9, Estado 4 productos
Soy el Productor: entrada=0, Estado 5 productos
Soy el Consumidor: salida=6, Estado 4 productos
Soy el Productor: entrada=1, Estado 5 productos
Soy el padre espero que termine el hijo .....
Soy el Consumidor: salida=7, Estado 4 productos
Soy el Consumidor: salida=8, Estado 3 productos
Soy el Consumidor: salida=9, Estado 2 productos
Soy el Consumidor: salida=0, Estado 1 productos
Soy el Consumidor: salida=1, Estado 0 productos
Soy el hijo y termino.....
Soy el padre destruyo los semaforos y termino.....
```