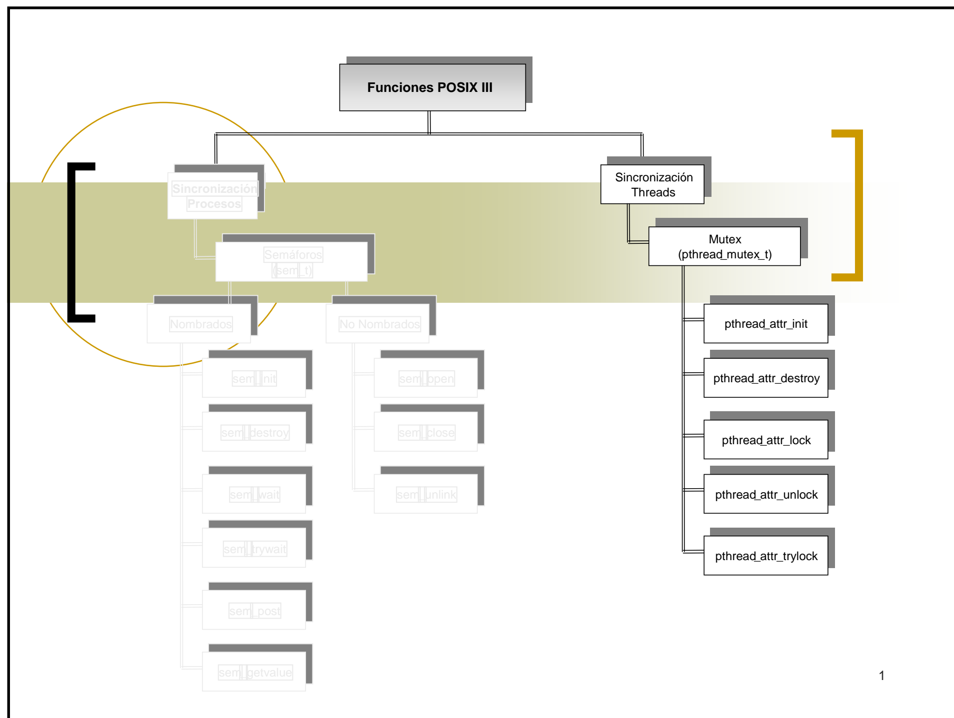



# Sincronización de Threads

- Sincronización de threads
  - Semáforos
  - Mutex (esperas a corto plazo)
  - Variables condicionales (esperas a largo plazo)
- Exclusión mutua
  - Mutex o candado: mecanismo de sincronización más sencillo y eficiente
  - Sirve para obtener acceso exclusivo a las variables o recursos
  - En POSIX.1c son variables de tipo pthread_mutex_t (estructura que almacena todos los datos concernientes al mutex)

## Funciones para *mutex* (I)

- Las funciones básicas para creación y manejo de mutex se muestran en la tabla siguiente

| Función | Descripción |
|---|---|
| `pthread_mutex_init` | Inicializa una variable de tipo `pthread_mutex_t` |
| `pthread_mutex_destroy` | Destruye una variable mutex |
| `pthread_mutex_lock` | Protege la sección crítica (operación P) |
| `pthread_mutex_unlock` | Libera la protección de la sección crítica (op. V) |
| `pthread_mutex_trylock` | Como `pthread_mutex_lock` pero sin bloqueo |



## Funciones para *mutex* (II)

- pthread_mutex_init: inicializa una variable de tipo pthread_mutex_t

```
#include <pthread.h>
int pthread_mutex_init (pthread_mutex_t *mutex,
                        const pthread_mutexattr_t *attr);
```

- Dos parámetros
  - Puntero a una variable pthread_mutex_t (mutex)
  - objeto atributo de mutex (attr)
  - Se inicializa por defecto haciendo el segundo parámetro NULL
  - Devuelve 0 si se ha podido inicializar el mutex.

```
#include<pthread.h>
pthread_mutex_t my_lock;
if (pthread_mutex_init(&my_lock, NULL)!=0)
       perror("No puedo inicializar my_lock");
```

# Funciones para *mutex* (III)

- pthread_mutex_destroy: destruye la variable mutex

```
#include <pthread.h>
int pthread_mutex_destroy (pthread_mutex_t *mutex);
```

  - El mutex no se destruye hasta que está completamente vacío (no hay threads en espera)
  - Devuelve 0 si se ha podido liberar el mutex.



# Funciones para *mutex* (III)

- pthread_mutex_lock: operación P de semáforos
  - Código de la sección de entrada que protege la sección crítica
  - Si el candado está cerrado, el thread que la invoca se suspende hasta que el candado quede libre
  - Devuelve 0 si se ha podido bloquear el mutex.

```
#include <pthread.h>
int pthread_mutex_lock (pthread_mutex_t *mutex);
```

- pthread_mutex_trylock: igual que pthread_mutex_lock pero no bloquea al proceso llamante si el candado está ocupado
  - Devuelve 0 si se ha podido bloquear el mutex.

```
#include <pthread.h>
int pthread_mutex_trylock (pthread_mutex_t *mutex);
```

## Funciones para *mutex* (IV)

- pthread_mutex_unlock: operación V de semáforos
  - Libera el candado cuando un thread termina su sección crítica
  - Si hay algún thread suspendido en la cola del candado, despierta al primero.
  - Devuelve 0 si se ha podido desbloquear el mutex.

```
#include <pthread.h>
int pthread_mutex_unlock (pthread_mutex_t *mutex);
```

- Ejemplo

```
#include<pthread.h>
pthread_mutex_t my_lock;
pthread_mutex_init(&mylock, NULL);

pthread_mutex_lock(&my_lock);
/* Sección crítica */
pthread_mutex_unlock(&my_lock);
```



## EJEMPLO:

Productor – Consumidor

:compilar con librerías: **–lposix4 -lpthread**

:compilar en **linux** con librerías: **–lrt -lpthread**

```
// 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
```

Buffer

Salida

Entrada



# Ejemplo *Mutex*

```
#include <pthread.h>
#define BUFSIZE 8
static int buffer[BUFSIZE]
static int entrada = 0;
static int salida = 0;
Static int contador =0;
static pthread_mutex_t candado = PTHREAD_MUTEX_INITIALIZER;

//Obtener el siguiente elemento del buffer y colocarlo en *itemp
void get_item (int *itemp)
{
  pthread_mutex_lock(&candado);
  contador=contador-1;
  *itemp = buffer[salida];
  salida = (salida + 1) % BUFSIZE;
  pthread_mutex_unlock(&candado);
  return;
}
```

# Ejemplo *Mutex* (cont.)

```
// Colocar un elemento en el buffer
void put_item(int item)
{
  pthread_mutex_lock(&candado);
  contador=contador+1;
  buffer[entrada] = item;
  entrada = (entrada + 1) % BUFSIZE;
  pthread_mutex_unlock(&candado);
  return;
}
```



# EJEMPLO:

## Uso de mutex en threads

:compilar con librerías: **–lposix4 -lpthread**

:compilar en **linux** con librerías: **–lrt -lpthread**

```
/* Programa que muestra el empleo de mutex con threads */
/* Compilar con: gcc -o mutexth mutexth.c -lpthread -lposix4 */

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sched.h>   /* Para la función sched_yield() */

#define MAXLONG 100

/* Prototipos de funciones que ejecutan los threads */
void *func1 (void*);
void *func2 (void*);

/* Declaración de los threads */
pthread_t thread1, thread2;

/* Declaración del objeto atributo */
pthread_attr_t attr;

/* Declaración del mutex */
pthread_mutex_t exmut;
```



```
/* Definición de la función func1 */
void *func1 (void *arg)
{
        char buffer[MAXLONG];
        char *c;

        sprintf (buffer, "Soy el thread 1 y estoy escribiendo un
mensaje en pantalla");
        c = buffer;
        /* Sección de entrada: operación P(S) */
        pthread_mutex_lock (&exmut);
        /* Sección crítica: Escritura en pantalla */
        while (*c != '\0')
        {
                fputc (*c, stdout);
                c++;
                /* Forzamos la expulsión de la CPU con sched_yield */
                sched_yield();
        }
        fputc('\n', stdout);
        /* Fin de la sección crítica*/
        /* Sección de salida: operación V(S) */
        pthread_mutex_unlock (&exmut);

        pthread_exit (NULL);
}
```

```
void *func2 (void *arg)
{
	char buffer[MAXLONG];
	char *d;

	sprintf (buffer, "Soy el thread 2 y estoy escribiendo un mensaje
en pantalla");
	d = buffer;
	/* Sección de entrada: operación P(S) */
	pthread_mutex_lock (&exmut);
	/* Sección crítica: Escritura en pantalla */
	while (*d != '\0')
	{
		fputc (*d, stdout);
		d++;
		/* Forzamos la expulsión de la CPU con sched_yield */
		sched_yield();
	}
	fputc('\n', stdout);
	/* Fin de la sección crítica*/
	/* Sección de salida: operación V(S) */
	pthread_mutex_unlock (&exmut);

	pthread_exit (NULL);
}
```



```
int main (int argc, char *argv[])
{
	/* Inicialización del objeto atributo */
	pthread_attr_init (&attr);

	/* Inicialización del mutex */
	pthread_mutex_init (&exmut, NULL);

	printf ("Soy la función main y voy a lanzar los dos
threads \n");
	pthread_create (&thread1, &attr, func1, NULL);
	pthread_create (&thread2, &attr, func2, NULL);
	printf ("Soy main: he lanzado los threads y termino \n");

	pthread_exit (NULL);

}
```

## Ejemplo: Secuenciación de Tareas

- Con mutex

```
Soy la función main y voy a lanzar los dos threads
Soy main: he lanzado los threads y termino
Soy el thread 1 y estoy escribiendo un mensaje en pantalla
Soy el thread 2 y estoy escribiendo un mensaje en pantalla
```

- Sin mutex

```
Soy la función main y voy a lanzar los dos threads
Soy main: he lanzado los threads y termino
SSooyy  eell  tthhrreeaadd  12  yy  eessttooyy  eesssccrriibbiieennddoo  uunn  mm
eennssaajjee  eenn  ppaanntta alllllaa
```



## Sincronización de Threads

Variables Condicionales

# Variables Condicionales

- Permiten manejar el acceso a secciones críticas múltiples (anidadas) evitando el interbloqueo.
- Ejemplo:
  - La tarea 1 accede a una sección crítica pero para ejecutarla precisa un dato adicional "producido" por otra tarea 2
  - La tarea 2 para generar el dato precisa acceder a la misma sección crítica por lo que se queda bloqueado.
- Funcionamiento:
  - Tienen asociado un **mutex** que controla el acceso a la sección crítica
  - Adicionalmente y de forma atómica permiten comprobar el estado de una variable (variable condicional)
  - Si el valor no es válido (***dato no disponible***) se desbloquea de forma atómica el mutex y el Thread se añade a la cola de espera.
  - La variable condicional tiene dos estados: **Señalizada** (unlock), **No Señalizada** (lock).



# Variables Condicionales

- Funciones manejo variables condicionales

```
int pthread_cond_init (pthread_cond_t *cond,
                       pthread_condattr_t *attr);
int pthread_cond_destroy (pthread_cond_t *cond);
int pthread_cond_wait (pthread_cond_t *cond,
                       pthread_mutex_t *mutex);
int pthread_cond_signal (pthread_cond_t *cond);
```

- **pthread_cond_wait** :
  - El mutex debe estar previamente bloqueado (sección crítica)
  - Si la variable condicional está "Señalizada" el thread continua con la sección crítica (el estado de la var. condicional se inicializa (lock))
  - Si la variable condicional no está "Señalizada" **desbloquea** el mutex y coloca al thread en la cola de espera
- **pthread_cond_signal:**
  - Desbloquea la variable condicional: despierta threads suspendidos e intentan bloquear el mutex

## EJEMPLO Variables Condicionales:

### Productor – Consumidor

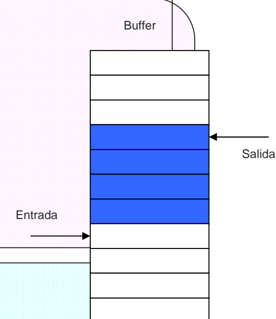
:compilar con librerías: **–lposix4 -lpthread**

:compilar en **linux** con librerías: **–lrt -lpthread**



```
// 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
```



## Ejemplo *Mutex+Var. Condicionales*

```
#include <pthread.h>
#define BUFSIZE 8
static int buffer[BUFSIZE]
static int entrada = 0;
static int salida = 0;
Static int contador =0;

static pthread_mutex_t candado;
static pthread_cond_t producido;
static pthread_cond_t consumido;

pthread_mutex_init(&candado, NULL);
pthread_cond_init(&producido, NULL);
pthread_cond_init(&consumido, NULL);


//..............
```



## Ejemplo *Mutex* +V.C. (cont.)

```
//Obtener el siguiente elemento del buffer y colocarlo en *itemp
void get_item (int *itemp)
{
  pthread_mutex_lock(&candado);
  while (contador == 0 ) /* si buffer VACIO */
        pthread_cond_wait(&producido, & candado); // espera

  contador=contador-1;
  *itemp = buffer[salida];
  salida = (salida + 1) % BUFSIZE;
  pthread_cond_signal(&consumido); //señaliza la extracción de datos
  pthread_mutex_unlock(&candado);
  return;
}
```

# Ejemplo *Mutex* +V.C. (cont.)

```
// Colocar un elemento en el buffer
void put_item(int item)
{
  pthread_mutex_lock(&candado);
  while (contador == BUFSIZE ) /* si buffer LLENO */
       pthread_cond_wait(&consumido, & candado); // espera

  contador=contador+1;
  buffer[entrada] = item;
  entrada = (entrada + 1) % BUFSIZE;
  pthread_cond_signal(&producido); //señaliza la presencia de datos
  pthread_mutex_unlock(&candado);
  return;
}
```