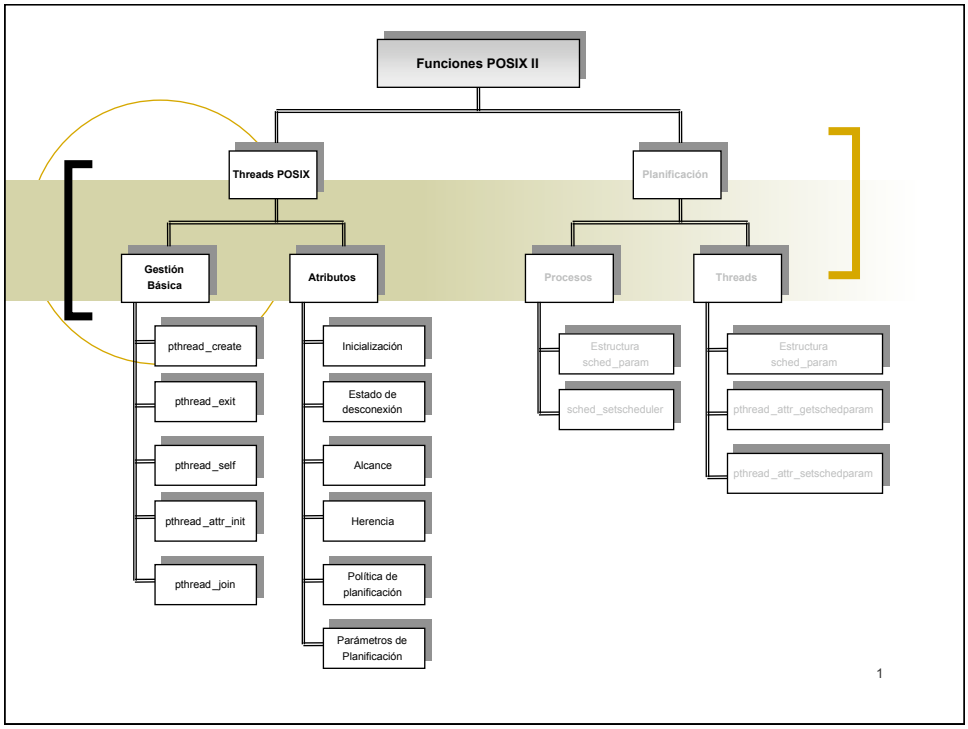

Threads POSIX

- Un SO que soporte threads debe contener un paquete que proporcione un sistema en tiempo real para manejo de threads
- Dos bibliotecas principales de gestión de threads: **Sun Solaris 2** y threads POSIX (***pthreads***)
- Ambos contemplan la creación y destrucción dinámica de threads
- **Sun Solaris 2**
  - Establecen explícitamente las propiedades de los threads
  - Ofrecen mayor control entre threads y recursos de procesadores
  - Normalmente sólo funcionan en Sun
- **Pthreads**
  - Se valen de objetos atributo para representar propiedades de los threads
  - Varios threads pueden estar asociados a un mismo objeto
  - Ofrecen un método más robusto de cancelación y terminación

Gestión básica de threads

- Funciones POSIX de gestión básica de threads

Función	Descripición
pthread_create	Crea un thread para ejecutar una función determinada
pthread_exit	Causa la terminación del thread que lo invoca
pthread_attr_init	Inicializa los atributos del thread a su valor por defecto
pthread_join	Hace que el thread que la invoca espere a que termine un thread determinado
pthread_self	devuelve la identidad del thread que lo invoca
pthread_cancel	Solicita la terminación de otro thread

- Un thread tiene asociados (entre otros) los siguientes parámetros
  - TID
  - Pila
  - Prioridad de ejecución
  - Dirección de inicio de la ejecución

Gestión básica de Threads (II)

- Un thread se representa mediante su TID de tipo *pthread_t*
- *pthread_t* es una estructura transparente al usuario y que depende de la implementación (pueden contener información adicional)
- Un pthread es dinámico ya que se puede crear en cualquier instante de tiempo
- En POSIX, cuando un thread se crea se coloca en una cola de threads preparados
- Normalmente las llamadas al sistema para gestión de threads devuelven 0 si se ha llevado a cabo con éxito y un valor no nulo en caso de error

Llamadas para gestión de Threads (I)

- Creación de threads: *pthread_create*

```
#include <pthread.h>
int pthread_create (pthread_t *tid, const pthread_attr_t
*attr, void *(*start_routine)(void*), void *arg);
```

- Parámetros
  - *tid*: apunta al thread que se crea
  - *attr*: atributos del thread creado (tipo *pthread_attr_t*)
  - *start_routine*: puntero a la función que debe ejecutar el thread
  - *arg*: puntero a los argumentos que se pasan a la función

Llamadas para gestión de Threads(II)

- Inicializar un objeto atributo por defecto: *pthread_attr_init*

```
#include <pthread.h>
int pthread_attr_init (pthread_attr_t *attr);
```

- Obtener identificador de un thread (TID): *pthread_self*

```
#include <pthread.h>
pthread_t pthread_self (void);
```

- Terminar un thread: *pthread_exit*

```
#include <pthread.h>
void pthread_exit (void *value_ptr);
```

Gestión básica de Threads: Ejemplo (I)

- La función que ejecuta el thread se pasa como un puntero a dicha función
- Los parámetros de la función se pasan como un puntero *void*
- Se declara un thread para la función *main* (*main* es implícitamente un thread cuando se trabaja con threads)
- Es necesario inicializar los atributos de los threads (aunque sea por defecto)

EJEMPLO:

Crear threads:

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

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

```c
//Programa que muestra cómo crear threads
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

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

pthread_t thread1, thread2, thmain;     /* Declaración de los threads */
pthread_attr_t attr;                    /* atributos de los threads */

/* Definición de las funciones func1 y func2 */
void *func1 (void *arg)
{
          pthread_t tid = pthread_self();  /* identificador de thread*/

          printf("Soy el thread 1 y voy a ejecutar func1 \n");
          sleep(2);
          printf("Soy el thread 1 y he terminado de ejecutar la función 1\n");
          pthread_exit(NULL); /* Provoca la terminación del thread*/
}

void *func2 (void *arg)
{
          pthread_t tid = pthread_self();  /* identificador de thread*/

          printf("Soy el thread 2 y voy a ejecutar func2 \n");
          sleep(5);
          printf("Soy el thread 2 y he terminado de ejecutar la función 2\n");
          pthread_exit(NULL); /* Provoca la terminación del thread*/
}
```


Ejemplo: Creación de threads

```c
/*Función main*/
int main(void)
{
     thmain = pthread_self();
     /*La propia función main es un thread*/

     /*inicializa los parámetros de los threads por defecto*/
     pthread_attr_init (&attr);

     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 dos threads y termino\n");
     pthread_exit (NULL);
}
```

```
[/u0/sitr/sitr001/threads]th1
Soy la función main y voy a lanzar los dos threads
Soy main: he lanzado los dos threads y termino
Soy el thread 1 y voy a ejecutar func1
Soy el thread 2 y voy a ejecutar func2
Soy el thread 1 y he terminado de ejecutar la función 1
Soy el thread 2 y he terminado de ejecutar la función 2
```

Llamadas para gestión de Threads(III)

- Dependencias entre threads: pthread_join, hace que el thread que la invoca espere a que termine un determinado thread

```
#include <pthread.h>
int pthread_join (pthread_t thread, void **value_ptr);
```

  - Permite que varios threads cooperen en una tarea
  - thread es el TID del thread que esperamos que termine
  - En value_ptr se almacena el código de salida del thread que termina (si es NULL no guarda nada)

EJEMPLO:

Threads: pthread_join

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

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

```c
// Ejemplo de utilización de pthread_join
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

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

pthread_t thread1, thread2, thmain;     /* Declaración de los threads */
pthread_attr_t attr;                    /* atributos de los threads*/

/* Definición de las funciones func1 y func2 */
void *func1 (void *arg)
{
          printf("Soy el thread1 y estoy ejecutando func1 \n");
          sleep(4);
          printf("Soy el thread 1 termino \n");
          pthread_exit (NULL);
}

void *func2 (void *arg)
{
          int err;
          printf("Soy el thread2 y esperando que thread1 termine \n");
          if (err =  pthread_join(thread1, NULL))
                    printf ("Error al esperar a thread1 \n");
          else
          {
            printf("Soy thread2, thread1 ha terminado y ejecuto func2 \n");
            sleep(2);
          }
          pthread_exit (NULL);
}
```


Ejemplo: pthread_join()

```c
/*Función main*/
int main(void)
{
          thmain = pthread_self();

          /*inicializa los parámetros de los threads por defecto*/
          pthread_attr_init (&attr);

          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 dos threads y termino
\n");     pthread_exit(NULL);
}
```

```
[/u0/sitr/sitr001/threads]th2
Soy la funci♦n main y voy a lanzar los dos threads
Soy main: he lanzado los dos threads y termino
Soy el thread1 y estoy ejecutando func1
Soy el thread2 y voy a esperar que thread1 termine
Soy el thread1 termino
Soy th2, th1 ha terminado y estoy ejecutando func2
[/u0/sitr/sitr001/threads]
```

EJEMPLO:

Paso de parámetros a threads:

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

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


```
//Programa que muestra el paso de parámetros en threads
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

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

// Estructura que contiene los datos a pasar como parámetros
// Un único parámetro se puede pasar directamente con el operador &
typedef struct
{
        int dato1,dato2;
}datos;

pthread_t thread1, thmain;    /* Declaración de los threads */
pthread_attr_t attr;          /*atributos de los threads*/

/* Definición de las funciones func1 */
void *func1 (void *arg)
{
        int a,b;

        datos *p= (datos *) (arg);

        pthread_t tid = pthread_self();  /*identificador de thread*/
        a=(p->dato1);
        b=(p->dato2);
        printf("Soy el thread 1 y voy a ejecutar func1 \n");
        printf("La multiplicación es %d\n",a*b);
        printf("Soy el thread 1 y he terminado de ejecutar la función 1\n");
        pthread_exit (NULL);
}
```

Ejemplo: paso de parámetros

```
/*Función main*/
int main (void)
{
        datos param;
        param.dato1=6;
        param.dato2=8;

        thmain = pthread_self();
        pthread_attr_init (&attr);

        printf("Soy la función main y voy a lanzar el thread \n");
        pthread_create (&thread1, &attr, func1, &param);

        printf("Soy main: he lanzado el thread y termino\n");
        pthread_exit (NULL);
}
```

```
[/u0/sitr/sitr001/threads]th3
Soy la función main y voy a lanzar el thread con parámetros 6x8
Soy main: he lanzado el thread y termino
Soy el thread 1 y voy a ejecutar func1
La multiplicación es 48
Soy el thread 1 y he terminado de ejecutar la función 1
[/u0/sitr/sitr001/threads]
```


Cancelar otro thread

- Solicita la terminación de otro thread:

```
#include <pthread.h>
int pthread_cancel (pthread_t target_thread );
```

- El thread objetivo debe estar en un estado '*cancelable*':
- Condiciones:
  - Configurar el thread para que sea cancelable:

```
int pthread_setcancelstate (int state, int * oldstate );
int pthread_setcanceltype (int type, int * oldtype );
```

state: **PTHREAD_CANCEL_ENABLE**, PTHREAD_CANCEL_DISABLE
type: **PTHREAD_CANCEL_DEFERRED**, PHREAD_CANCEL_ASYNCHRONOUS

  - Estar en estado suspendido (espera) o un punto de cancelación:

```
void pthread_testcancel();
```

EJEMPLO:

Cancelación de un thread

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

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


```
//Programa que muestra el paso de parámetros en threads
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

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


pthread_t thread1, thmain;    /* Declaración de los threads */
pthread_attr_t attr;          /*atributos de los threads*/

/* Definición de la función func1 */
void *func1 (void *arg)
{
          pthread_t tid;

          tid=pthread_self();  //se asigna un identificador de  thread

          // permite la cancelación del thread
          pthread_setcanceltype (PTHREAD_CANCEL_DEFERRED, NULL);
          pthread_setcancelstate (PTHREAD_CANCEL_ENABLE, NULL);

          printf("Soy el thread 1 [TID:%d] y voy a esperar\n",tid);

          sleep(10);                    // punto de cancelación
          pthread_testcancel();         // punto de cancelación

          printf("Soy el thread 1 [TID:%d] y he terminado\n", tid);
          pthread_exit (NULL);
}
```

Ejemplo: cancelación de threads

```
/*Función main*/
int main (void)
{
     thmain = pthread_self();   /*La propia función main es un thread*/
     pthread_attr_init (&attr);

     printf("Soy main [TID: %d] y voy a lanzar el thread\n", thmain);
     pthread_create (&thread1, &attr, func1, NULL);

     sleep(2);

     if (pthread_cancel(thread1)!=0)
         perror("Error cancelando thread");
     else
         printf("Soy main [TID: %d] he cancelado el thread [TID: %d] y termino\n",
                                        thmain, thread1);

     pthread_exit (NULL);
}
```

```
[/u0/sitr/sitr001/threads]th5
Soy main [TID: 1] y voy a lanzar el thread
Soy el thread 1 [TID:4] y voy a esperar
Soy main [TID: 1] he cancelado el thread [TID: 4] y termino
[/u0/sitr/sitr001/threads]
```


Atributos de los Pthreads (I)

- Los atributos de los threads en POSIX se encapsulan en un objeto tipo atributo (pthread_attr_t)
- Este mecanismo permite asociar a un grupo de threads los mismos atributos
- Cuando cambia una propiedad del objeto atributo, cambia para todos los threads asociados a él
- POSIX proporciona llamadas para crear, inicializar, configurar y destruir elementos de un objeto atributo

Atributos de los Pthreads (II)

- Algunas de las propiedades asociadas a los threads se muestran en la tabla siguiente

Propiedad	Función
Inicialización	pthread_attr_init pthread_attr_destroy
Tamaño de la pila	pthread_attr_setstacksize pthread_attr_getstacksize
Dirección de la pila	pthread_attr_setstackaddr pthread_attr_getstackaddr
Estado de desconexión	pthread_attr_setdetachstate pthread_attr_getdetachstate
Alcance	pthread_attr_setscope pthread_attr_getscope
Herencia	pthread_attr_setinheritsched pthread_attr_getinheritsched
Política de planificación	pthread_attr_setschedpolicy pthread_attr_getschedpolicy
Parámetros de planificación	pthread_attr_setschedparam pthread_attr_getschedparam

Atributos de los Pthreads (III)

- Excepto pthread_attr_init y pthread_attr_destroy todas las funciones para obtener o establecer atributos tienen dos parámetros

```
#include <pthread.h>
#include <sched.h>
int pthread_attr_setstacksize(pthread_attr_t *attr,
                      size_t stacksize);
```

- El primer parámetro es un puntero a un objeto pthread_attr_t y el segundo el valor del atributo que se desea cambiar

Algunos atributos de los pthreads(I)

- Inicialización y destrucción
  - pthread_attr_init: inicializa un objeto atributo a los valores por defecto
  - pthread_attr_destroy: hace que el valor del objeto sea no válido (POSIX no especifica el comportamiento del objeto una vez destruido)
- Dirección y tamaño de pila
  - pthread_attr_getstackaddr: obtiene dirección
  - pthread_attr_setstackaddr: establece dirección
  - pthread_attr_getstacksize: obtiene tamaño
  - pthread_attr_setstacksize: establece tamaño

Algunos atributos de los pthreads(II)

- Estado de desconexión: un thread puede estar en dos estados
  - estado de conexión (***undetached***): los demás threads son conscientes de su existencia y pueden esperarlo (*join*)
  - estado de desconexión (***detached***): los demás threads no tienen consciencia de su existencia (no se puede esperar)
  - **pthread_attr_getdetachstate**: examina el '*detachstate*' de un objeto atributo.Este puede ser
    - PTHREAD_CREATE_JOINABLE (no desconectado)
    - PTHREAD_CREATE_DETACHED (desconectado)
  - **pthread_attr_setdetachstate**: establece el '*detachstate*' a uno de los valores anteriores
  - Los threads desconectados invocan la llamada ***pthread_detach*** cuando terminan para liberar los recursos