SISTEMAS OPERATIVOS

3004610 - 1

German Sánchez Torres, I.S., M.Sc., Ph.D.

Profesor, Facultad de Ingeniería - Programa de Sistemas

Universidad del Magdalena, Santa Marta.

Phone: +57 (5) 4214079 Ext 1138 - 301-683 6593

Edificio Docente, Cub 3D401.

Email: sanchez.gt@gmail.com –gsanchez@unimagdalena.edu.co

Hilos de Ejecución

pthread

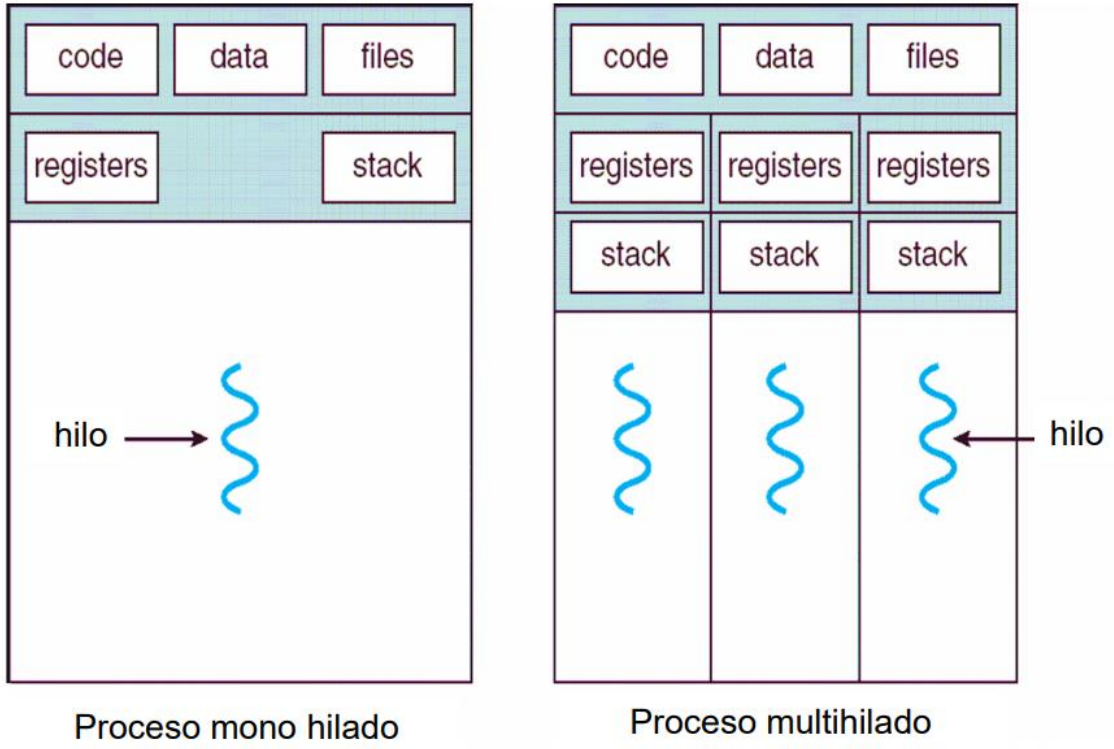
SISTEMAS OPERATIVOS

3004610 - 1

PROCESOS LIGEROS

INTRODUCCION

- Un proceso ligero (*thread o hebra*) es un programa en ejecución que usa los recursos de un proceso por lo que comparte la imagen de la memoria y otra información con otros procesos ligeros.



Proceso mono hilado

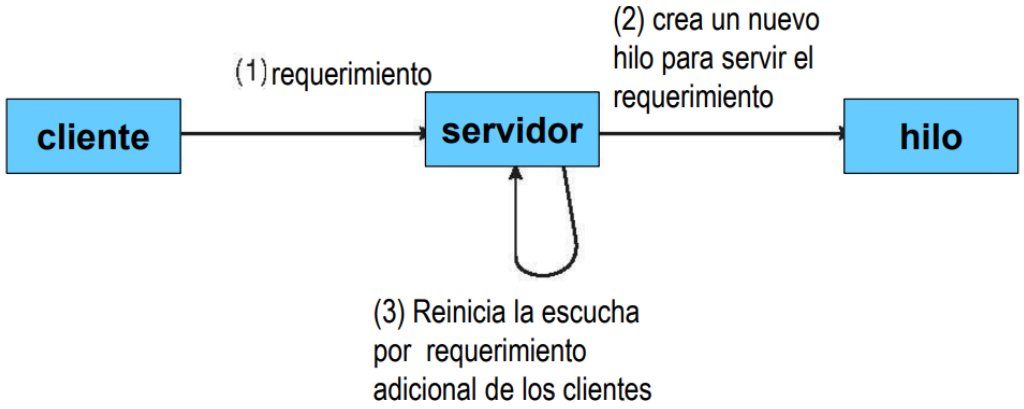
Proceso multihilado

Ventajas del modelo de múltiples hilos

Recursos compartidos: Facilita la comunicación entre procesos sin intervención del kernel.

Administración eficiente: Requiere menos costo computacional la creación, la computación de contexto y la terminación.

Aprovecha las arquitecturas multiprocesador: Permitiendo un uso mas eficiente de los recursos computacionales disponibles



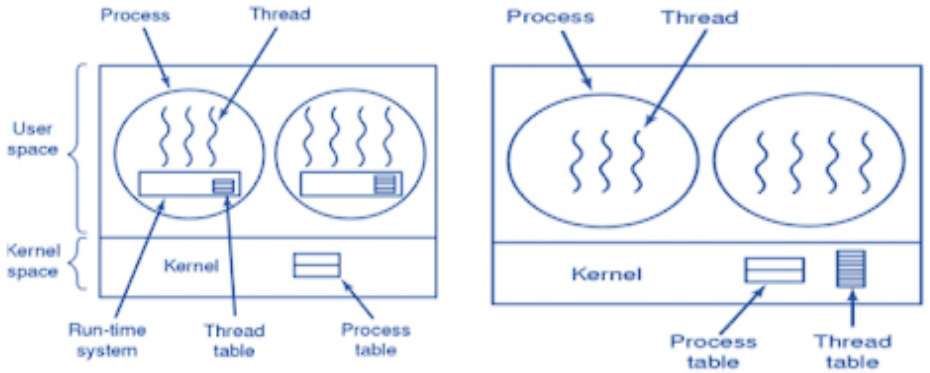
Modelo de hilos en el SO

Hilos de nivel de kernel (KLT- kernel-level thread)
Son administrados directamente por el sistema operativo.

- El núcleo conoce y administra todos los hilos.
- Una entrada en el bloque de control de proceso (PCP) para cada proceso.
- Una entrada en el bloque de control de hilos (TCB) por hilo en el sistema.
- El SO proporciona llamadas al sistema especificas para crear y administrar hilos.

Hilos de nivel de Usuario (ULT- User-level thread)
Son administrados sin soporte del kernel del SO.

- El kernel no conoce los hilos y son administrados como si fueran un único proceso.
- No permiten tener estados diferentes.
- No requieren modificación del SO (librerías de nivel de usuario)



(a) A user-level threads package. (b) A threads package managed by the kernel.

Información compartida y no compartida

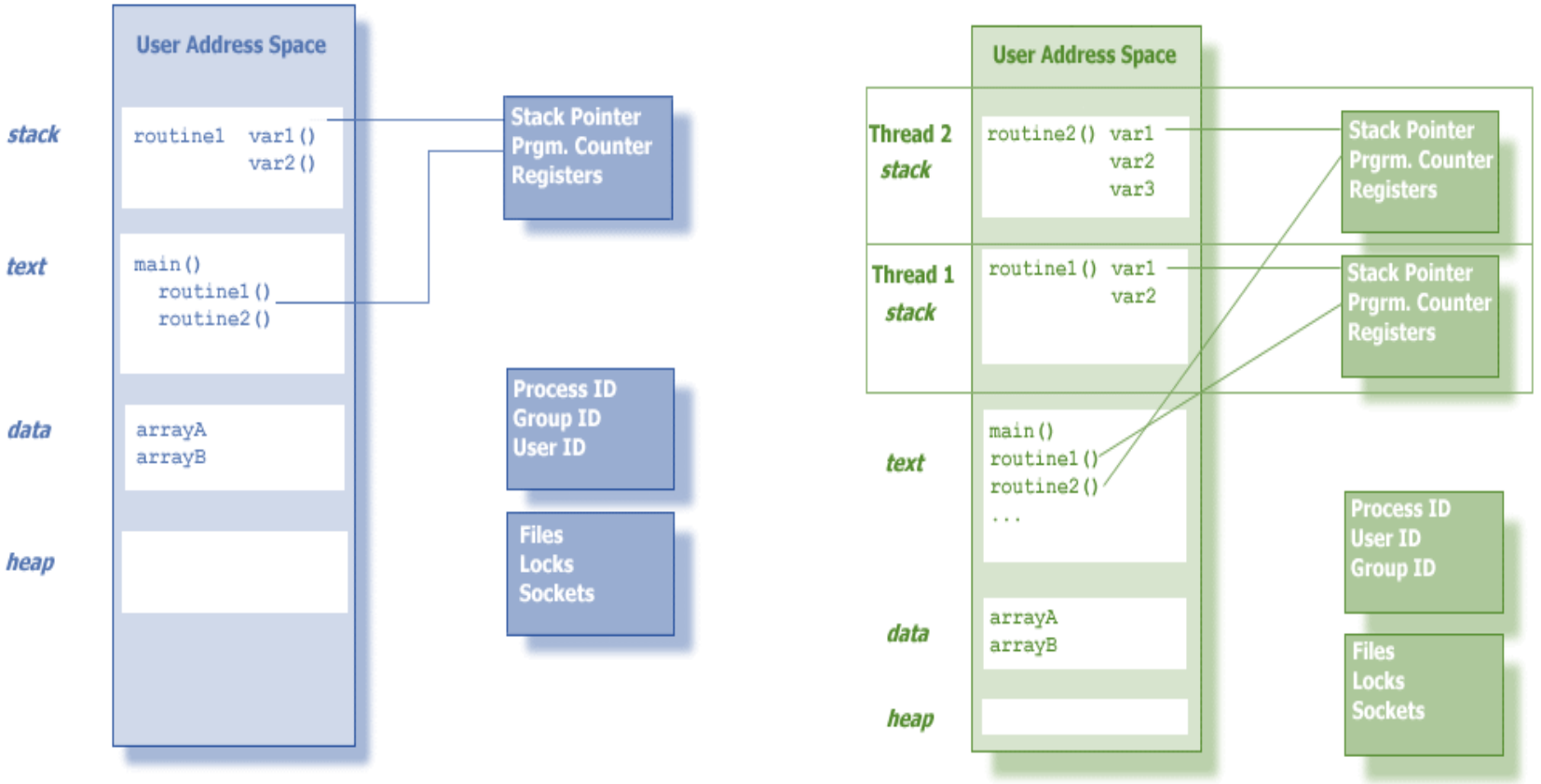
Información compartida

- **Espacio de memoria**
- **Variables globales**
- **Archivos abiertos**
- **Procesos hijos**
- **Temporizadores**
- **Señales y semáforos**
- **Contabilidad**

Información no compartida

- **Contador de programa**
- **Pila**
- **Registros del procesador**
- **Estado del proceso ligero**

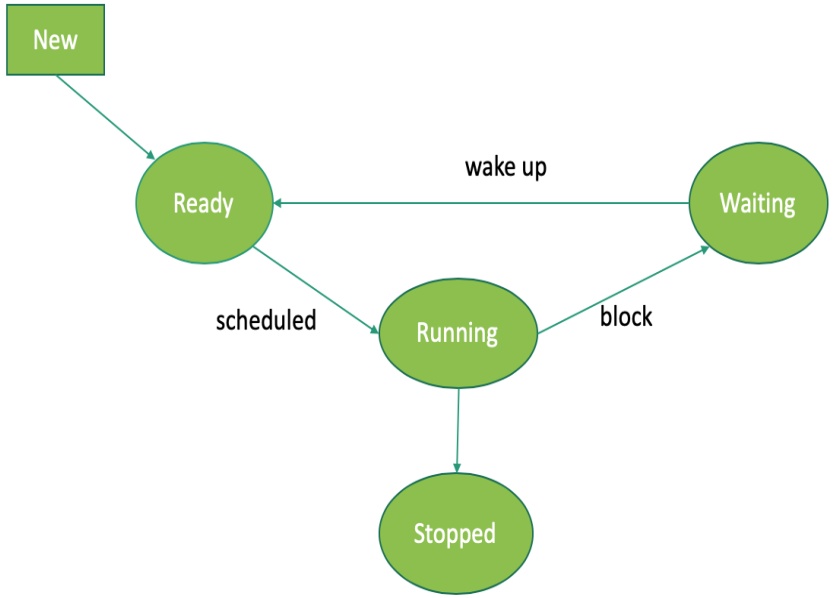
Información compartida y no compartida

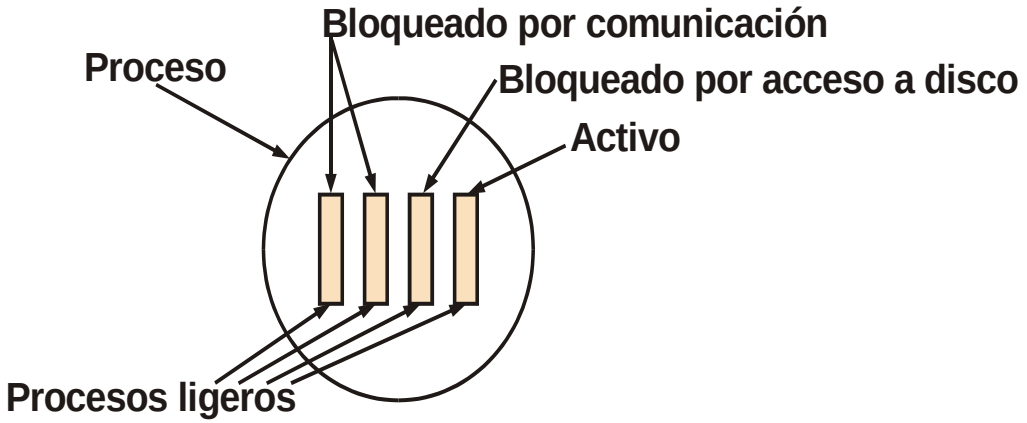


Estados de los hilos

Estado de los procesos ligeros

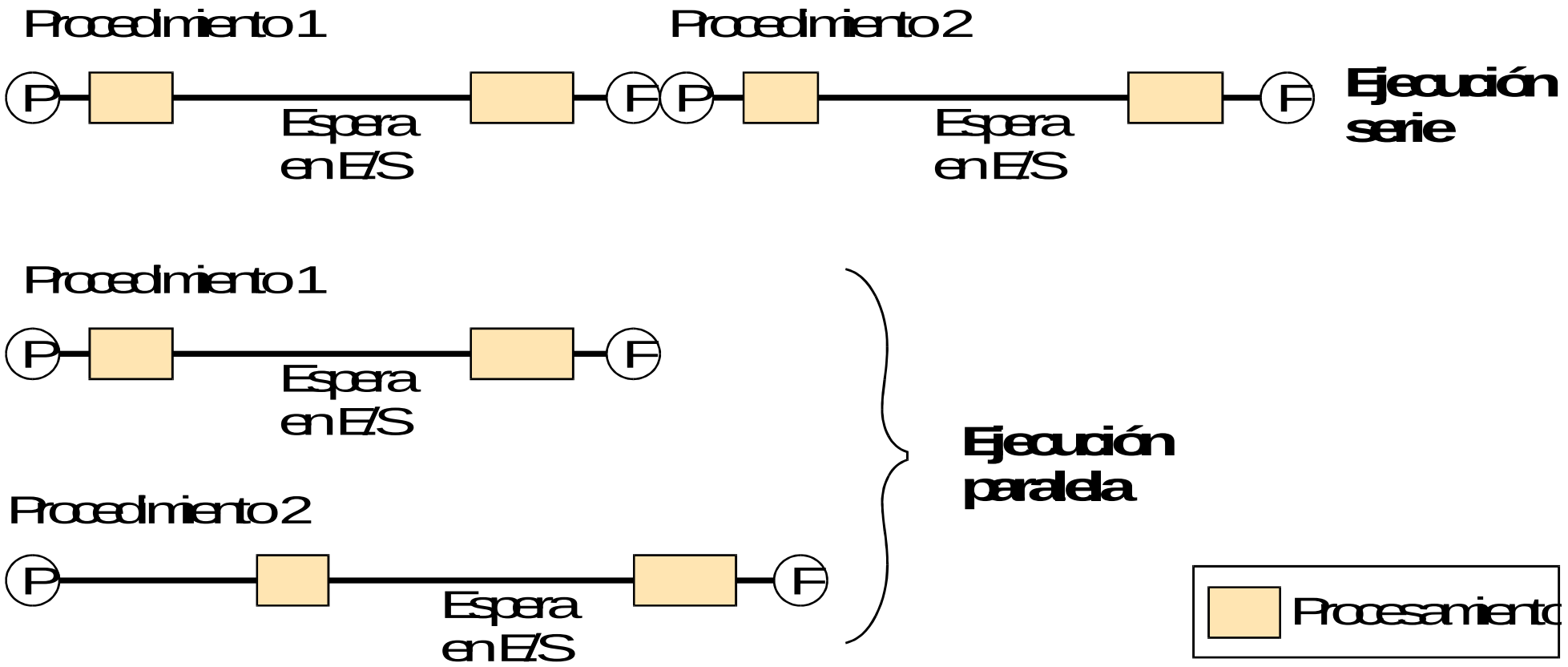
- Un proceso ligero puede estar ejecutando, listo o bloqueado.





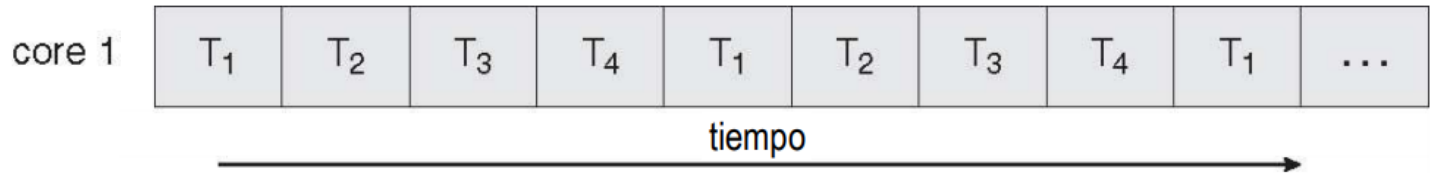
Paralelismo

Los procesos ligeros permiten paralelizar una aplicación.

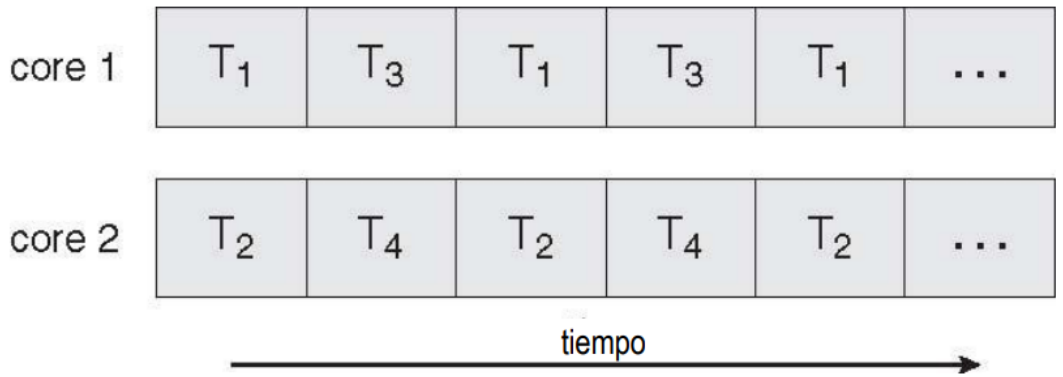


Paralelismo

Mono-Core



Multi-Core



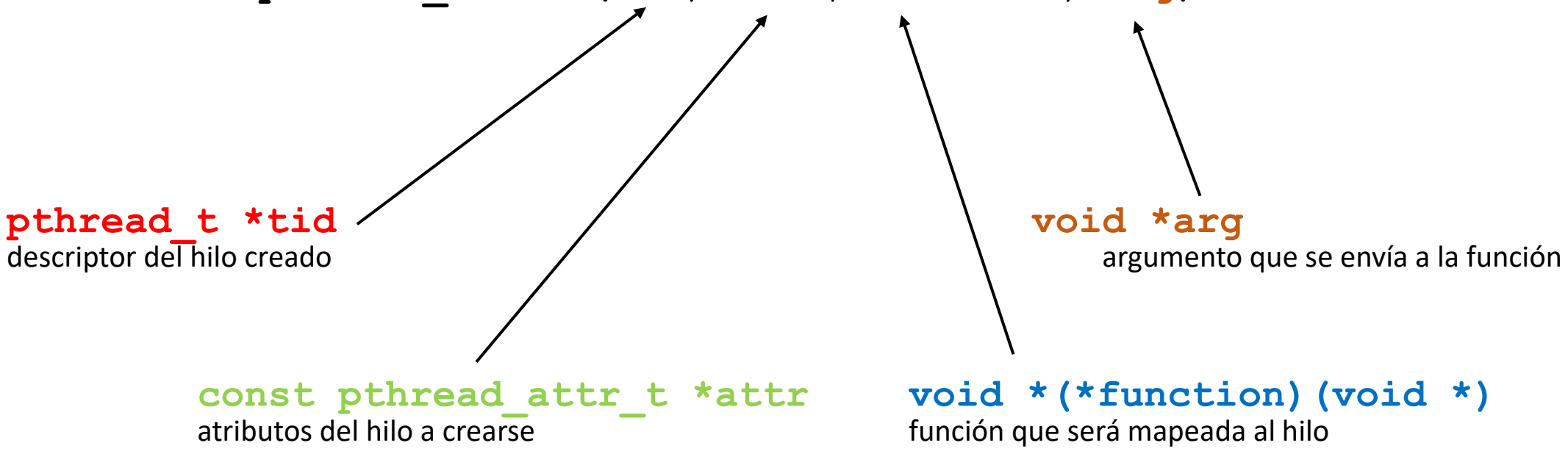
Librería Pthread

Pthread – hilos POSIX

- Es la implementación estándar para sistemas derivados de Unix.
- IEEE POSIX 1003.1c API para creación y sincronización.
- Tipos definidos en pthread.h para lenguaje C
- Los hilos Pthread tienen dos tipos: *Acoplado (joinable) Desacoplado (detached)*
- Por defecto los hilos son acoplados: Los recursos se mantienen hasta el **`pthread_join`**
- Los hilos desacoplados no pueden unirse (esperarse): Los recursos pueden reclamarse en la terminación, no se pueden resetear a ser *unibles.*

Pthread – hilos POSIX

```
int pthread_create(tid, attr, function, arg);
```



pthread_create()

```
int pthread_create(tid, attr, function, arg);
```

Inicia un hilo ejecutando la función **`function`**

Descriptor del hilo retornado por medio de la estructura **`pthread_t`**

Especifica *NULL* para usar los atributos por default

Un único argumento enviado a la función, si no tiene argumentos, especifica *NULL*

Se debe verificar los códigos de error!

`EAGAIN` – recursos insuficientes para crear el hilo
`EINVAL` – atributo inválido

Terminación de hilos

```
int pthread_create(tid, attr, function, arg);
```

El nuevo hilo termina mediante una de las siguientes maneras:

Llama a pthread_exit(), especificando un valor de estado de salida que es disponible para otro hilo en el mismo proceso que llama pthread_join().

Retornar de `function()`. Esto es equivalente a llamar pthread_exit con el valor proporcionado en la declaración de devolución.

Que sea cancelado pthread_cancel().

Cualquiera de los hilos en el proceso llama a la exit(), o el hilo principal retorna de main(). Esto provoca la terminación de todo hilos en el proceso.

Esperando un hilo

```
int pthread_join(tid, val_ptr);
```

pthread_t tid

Identificador de un hilo *a esperar*

```
int pthread_create(tid, attr, function, arg);
```

void **val_ptr

valor de salida devuelto por un hilo

Esperando un hilo

Un hilo espera a que un hilo con descriptor **`tid`** termine

- Solo espera a que un hilo se *una*
- El hilo debe ser *acoplado*

Un valor de salida se devuelve del hilo unido

- Tipo devuelto es **`(void *)`**
- Usar **`NULL`** si no se espera un valor de retorno

`ESRCH` - hilo (pthread_t) no encontrado
`EINVAL` - hilo (pthread_t) no unible

Ejemplos

Compilación y ejecución de programas con hilos POSIX

gcc –o PthreadExample PthreadExample.c –lpthread

./PthreadExample

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

/* Global variable:  accessible to all threads */
int thread_count;

void *Hello(void* rank);  /* Thread function */

int main(int argc, char* argv[]) {
   long       thread;  /* Use long in case of a 64-bit system */
   pthread_t* thread_handles;

   /* Get number of threads from command line */
   thread_count = strtol(argv[1], NULL, 10);

   thread_handles = malloc (thread_count*sizeof(pthread_t));
```

Pacheko Parallel Programming Ch 04 - Threads

```
   for (thread = 0; thread < thread_count; thread++)
      pthread_create(&thread_handles[thread], NULL,
          Hello, (void*) thread);

   printf("Hello from the main thread\n");

   for (thread = 0; thread < thread_count; thread++)
      pthread_join(thread_handles[thread], NULL);

   free(thread_handles);
   return 0;
}  /* main */


void *Hello(void* rank) {
   long my_rank = (long) rank;  /* Use long in case of 64-bit system */

   printf("Hello from thread %ld of %d\n", my_rank, thread_count);

   return NULL;
}  /* Hello */
```

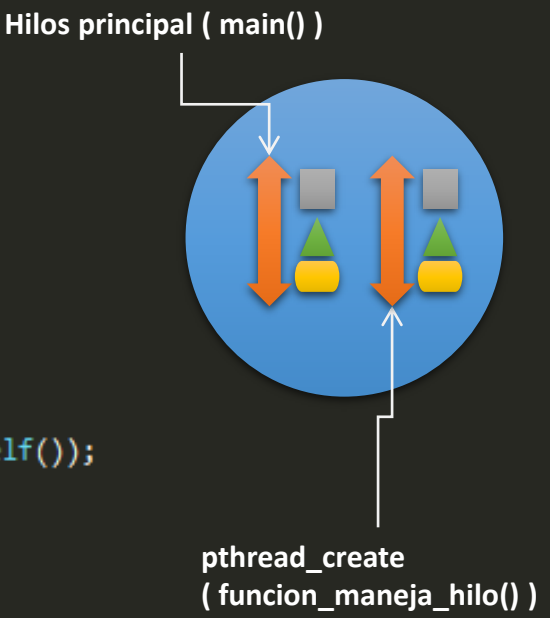
Pacheko Parallel Programming Ch 04 - Threads

```c
/*
Ejemplo Hilo1.c
*/
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

void* funcion_maneja_hilo(void *);


int main(){
    pthread_t pidhilo;
    pthread_create(&pidhilo, NULL, funcion_maneja_hilo, NULL);
    printf("Hilo principal (idthread -> [%ld])\n", (long int) pthread_self());
    pthread_join(pidhilo, NULL);
return 0;
}

void* funcion_maneja_hilo(void *param){
    printf("Hilo (idthread -> [%ld])\n",(long int) pthread_self());
    pthread_exit(0);
}
```



Registros de procesador

Program counter (*pc*)

Puntero de pila

```c
/*
Ejemplo Hilo2.c
Envío de parametros simple en la creacion de hilos
mediante cadena de caracteres
*/
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

void* funcion_maneja_hilo(void *);


int main(){
    pthread_t pidhilo;
    pthread_create(&pidhilo, NULL, funcion_maneja_hilo, (void*) "hola");
    printf("Hilo principal (idthread -> [%ld])\n", (long int) pthread_self());
    pthread_join(pidhilo, NULL);
return 0;
}

void* funcion_maneja_hilo(void *param){
    printf("Hilo param->%s  (idthread -> [%ld])\n",(char *)param, (long int) pthread_self());
    pthread_exit(0);
}
```

```c
/*
Ejemplo Hilo2.c
Envío de parametros a un hilo mediante el uso
de una estructura de datos
*/
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

void* funcion_maneja_hilo(void *);
typedef unsigned long int tipo_hilo;

struct nodo{
 int val;
};

int main(){
    pthread_t pidhilo;
    struct nodo Nodo;
    Nodo.val = 5;
    pthread_create(&pidhilo, NULL, funcion_maneja_hilo, (void*) &Nodo);
    printf("Hilo principal (idthread -> [%lu])\n", pthread_self());
    pthread_join(pidhilo, NULL);
return 0;
}

void* funcion_maneja_hilo(void *param){
    printf("Hilo param->%d  (idthread -> [%lu])\n",((struct nodo *)param)->val, pthread_self());
    pthread_exit(0);
}

```

```c
/*
Ejemplo Hilo2.c
Error en el Envío de parametros a multiples hilos mediante el uso
de una estructura de datos
*/
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

void* funcion_maneja_hilo(void *);
typedef unsigned long int tipo_hilo;

struct nodo{
 int val;
};

int main(){
    pthread_t pidhilo[2];
    struct nodo Nodo;
    int i;

    for(i=0; i<2; i++){
     Nodo.val = i;
     pthread_create(&pidhilo[i], NULL, funcion_maneja_hilo, (void*) &Nodo);
    }
    printf("Hilo principal (idthread -> [%lu])\n", pthread_self());
    for(i=0; i<2; i++){
     pthread_join(pidhilo[i], NULL);
    }
return 0;
}

void* funcion_maneja_hilo(void *param){
    printf("Hilo param->%d  (idthread -> [%lu])\n",((struct nodo *)param)->val, pthread_self());
    pthread_exit(0);
}
```

```c
/*
Ejemplo Hilo2.c
Correcto Envio de parametros a multiples hilos mediante el uso
de una estructura de datos
*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void* funcion_maneja_hilo(void *);
typedef unsigned long int tipo_hilo;

struct nodo{
 int val;
};

int main(){
    pthread_t pidhilo[2];
    struct nodo *Nodo;
    int i;

    for(i=0; i<2; i++){
     Nodo = (struct nodo *)malloc(sizeof(struct nodo));
     Nodo->val = i;
     pthread_create(&pidhilo[i], NULL, funcion_maneja_hilo, (void*) Nodo);
    }
    printf("Hilo principal (idthread -> [%lu])\n", pthread_self());

    //Esperar por la terminacion de los Hilos
    for(i=0; i<2; i++){
     pthread_join(pidhilo[i], NULL);
    }
return 0;
}

void* funcion_maneja_hilo(void *param){
    printf("Hilo param->%d  (idthread -> [%lu])\n",((struct nodo *)param)->val, pthread_self());
    pthread_exit(0);
}
```