





# Taller de Programación Paralela

Fernando R. Rannou  
Departamento de Ingeniería Informática  
Universidad de Santiago de Chile

March 31, 2011



Hebras

**Hebras**



# Hebra versus proceso

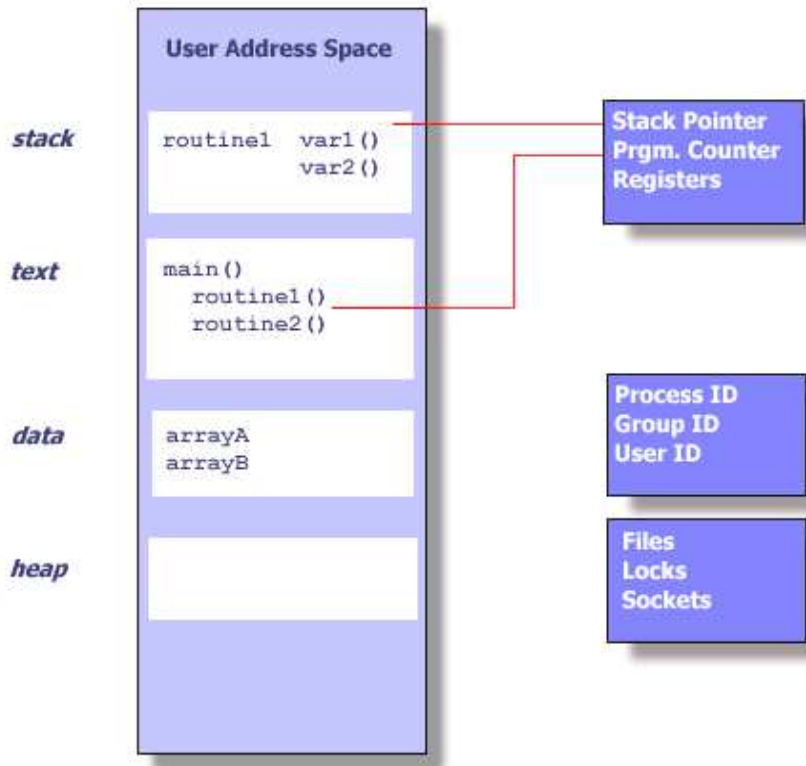
Hebras



- Recordemos que el concepto de proceso implica dos características:
  1. un programa en ejecución o la entidad que puede planificarse en el procesador
  2. la entidad que posee un programa ejecutable, area de datos globales, stack del usuario, etc.
- La primera característica define la hebra de un proceso
- La segunda, los recursos del proceso
- Desde este punto de vista todo proceso tiene al menos una hebra

# Los elementos de un proceso

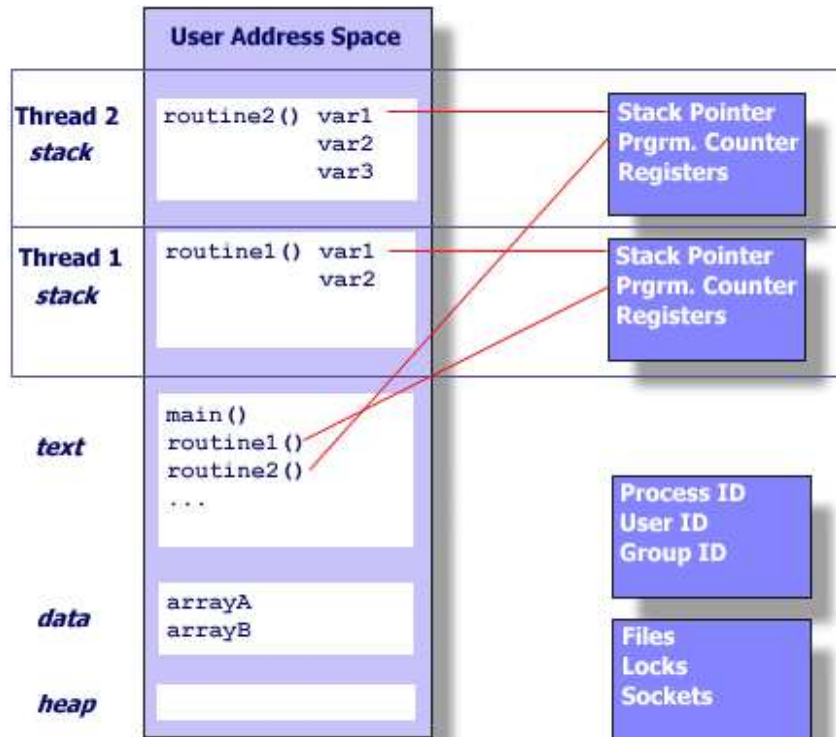
Hebras



- Un estado de ejecución (running, ready, blocked, etc)
- Contexto de proceso
- Programa ejecutable (texto)
- Área de datos globales (dato)
- Stack del usuario (y del sistema) (stack)
- Identificadores: proceso, usuario, dueño, grupo, etc

# Los elementos de una hebra

Hebras



- Un estado de ejecución
- Un contexto de hebra
- Almacenamiento estático de memoria para variables locales
- Un stack de ejecución
- Identificador de hebra
- Derecho a acceder a los datos globales y recursos del proceso al que pertenece



# Concepto de hebra

Hebras



- Una hebra existe “dentro” de un proceso y usa los recursos del proceso
- Tiene un flujo de control independiente
- Comparte con otras hebras del proceso recursos del proceso
- La hebra deja de existir cuando el proceso al que pertenece termina
- Es “de peso liviano” (lightweight) ya que la mayor parte del overhead en su creación es la creación del proceso mismo



# Beneficios de las hebras

Hebras



- Demora menos crear y eleminar una hebra que un proceso
- Demora menos hacer cambio de contexto entre hebras de un mismo proceso que entre dos procesos
- Ya que las hebras de un proceso comparten memoria y archivos, ellas se pueden comunicar sin necesidad de invocar rutinas del kernel
- Permitiría la ejecución paralela de hebras cuando hay varios procesadores



# Otro ejemplos

Hebras

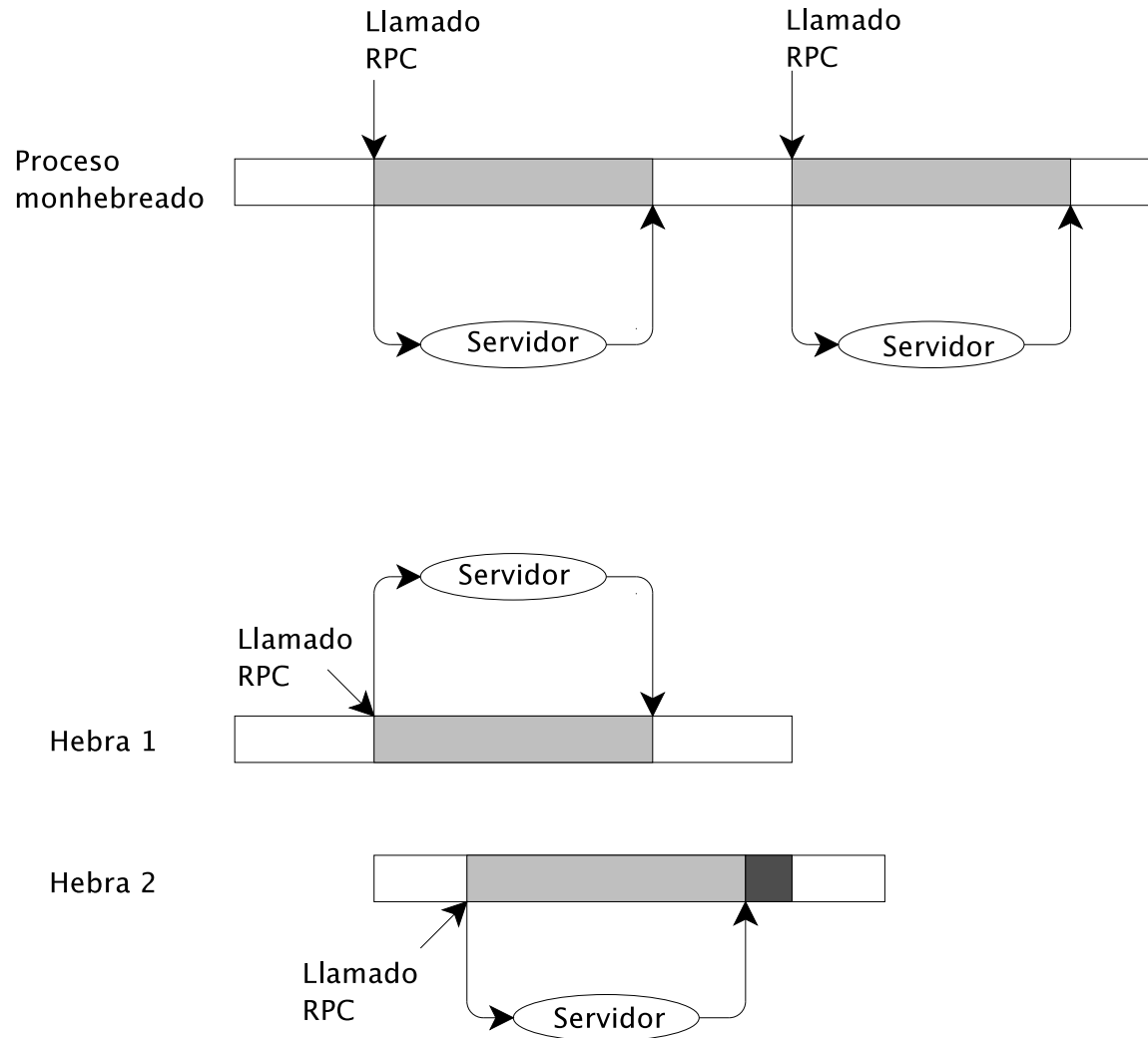


- Ejecución concurrente de tareas background y foreground
- Procesamiento asíncrono
- Aumento velocidad de ejecución
- Intercalar tareas de procesamiento con tareas de lectura
- Estructura modular del programa



# Multihebras en mono procesador

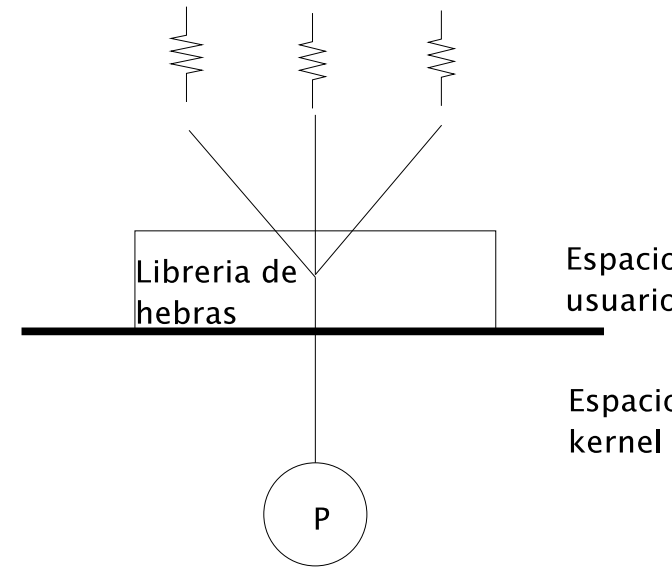
Hebras



# Hebras a nivel de usuario

Hebras

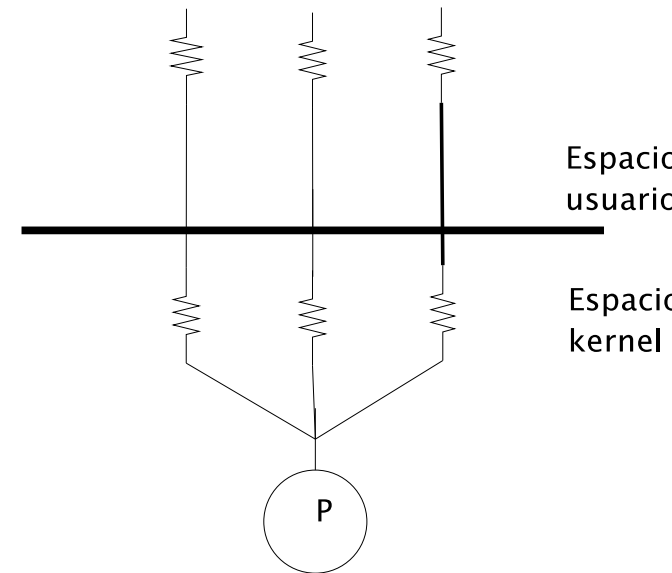
- Toda la administración de hebras la realiza la aplicación misma (proceso) o por librerías de manejo de hebras
- El kernel no conoce la existencia de hebras
- Scheduling es al nivel de procesos
- Ventajas:
  - ◆ Cambio de contexto es más rápido
  - ◆ Pueden ser implementada en cualquier SO y no requieren cambios al kernel
- Desventajas
  - ◆ No se puede explotar múltiples procesadores
  - ◆ Cuando una hebra se bloquea, el proceso entero se bloquea



# Hebras a nivel de kernel

Hebras

- El kernel mantiene información de contexto por el proceso y por la hebras del proceso
- La administración de hebras la realiza el kernel
- Scheduling es al nivel de hebra
- Ventajas:
  - ◆ Se puede explotar múltiples procesadores
  - ◆ Si una hebra se bloquea, el control de la CPU puede pasar a otra hebra
- Desventajas
  - ◆ Cambio de contexto entre hebras requiere la intervención del kernel
  - ◆ Si no usa una API estándar, los programas son menos portables





# Symmetric multiprocessing (SMP)

Hebras



- En un sistema SMP, varios procesadores residen en la misma tarjeta o el mismo chip (*multi core*).
- Todos comparten un mismo bus para acceder la memoria, dispositivos de I/O, etc. Sin embargo, cada uno tiene su propia memoria cache
- Un Sistema Operativo SMP permite utilizar dichos procesadores en forma simétrica:
  1. El SO puede correr en cualquiera de los procesadores
  2. El SO puede asignar hebras o procesos a cualquiera de los procesadores
  3. Es posible que el SO corra al mismo tiempo en dos o más procesadores
- Un sistema SMP junto con programación multihebras a nivel de kernel permite paralelismo
- A diferencia, un sistema no-simétrico, por ejemplo, es el modelo maestr-esclavo



# POSIX threads y Linux

Hebras



- IEEE POSIX 1003.1c es un estándar de programación de hebras en el lenguaje C
- LinuxThreads es el primer intento de hebras en Linux; no se adhiere al estándar POSIX
- Nuevo desarrollo NPTL *Native POSIX Thread Library* mejora sustancialmente deficiencias de LinuxThreads y conforma con el estándar POSIX
- LinuxThreads y NPTL implementan el model uno-a-uno, es decir una hebra de kernel soporta una hebra de usuario



# API Pthreads

Hebras



La API (*Application Programm Interface*) Pthreads define funciones en dos áreas:

1. **Administración de hebras:** creación, destrucción, *detaching*, *joining*, etc.
2. **Sincronización de hebras**
  - *Mutex* : creación, inicialización, locking, etc de *mutexes*, para exclusión mútua entre hebras.
  - *variables de condición*: mecanismo de comunicación entre hebras que comparten en mutex.

# Creación y destrucción de hebras

Hebras

```
#include <pthread.h>
```

```
int pthread_create(pthread_t *thread,  
                  const pthread_attr_t *attr,  
                  void *(*start_routine)(void*), void *arg);
```

- `thread` es el identificador de la nueva hebra (valor de retorno)
- `attr` es un argumento con los atributos de creación de la hebra (valor de entrada)
- `start_routine()` define la función que la hebra ejecuta al momento de creación
- `arg` es el argumento para `start_routine`
- En caso de éxito, `pthread_create()` retorna 0
- En caso contrario, retorna un valor de error distinto de 0



# Estructuras opacas

Hebras



- Los tipos `pthread_t` y `pthread_attr_t` son opacos, en el sentido que no conocemos su estructura interna
- Este tipo de argumentos son accesibles usando otras funciones de la librería
- Por ejemplo, un atributo *name* de una hebra pueden “setearse” y leerse usando `pthread_attr_setname` y `pthread_attr_getname`, respectivamente, donde *name* es el nombre del atributo.



# Atributos

Hebras

Algunos de los atributos de creación de hebras son:

Tipo y nombre	Descripción y macros
<code>size_t stacksize</code>	El tamaño del stack de la hebra $\geq$ <code>PTHREAD_STACK_MIN</code>
<code>void *stackaddr</code>	La dirección del stack de la hebra
<code>int detachstate</code>	El estado de <i>acoplamiento</i> acoplamiento de la hebra con su hebra padre <code>PTHREAD_CREATE_DETACHED</code> , <code>PTHREAD_CREATE_JOINABLE</code>
<code>int schedpolicy</code>	La política de scheduling de la hebra <code>SCHED_FIFO</code> , <code>SCHED_RR</code> , <code>SCHED_OTHER</code>
<code>struct sched_param</code> <code>schedparam</code>	Los parámetros de scheduling

# Funciones sobre atributos

Hebras

\*

## ■ Funciones para setear atributos:

```
int pthread_attr_getdetachstate(const pthread_attr_t *attr, int *detachstate);
int pthread_attr_getschedparam(const pthread_attr_t *attr,
                               struct sched_param *param);
int pthread_attr_getschedpolicy(const pthread_attr_t *attr, int *policy);
int pthread_attr_getstackaddr(const pthread_attr_t *attr, void **stackaddr);
int pthread_attr_getstacksize(const pthread_attr_t *attr, size_t *stacksize);
```

## ■ Funciones para leer atributos:

```
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
int pthread_attr_setschedparam(pthread_attr_t *attr,
                               const struct sched_param *param);
int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
int pthread_attr_setstackaddr(pthread_attr_t *attr, void *stackaddr);
int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

## ■ Inicializar atributos por defecto

```
int pthread_attr_init(pthread_attr_t *attr);
```

## ■ Destruir atributos

```
int pthread_attr_destroy(pthread_attr_t *attr);
```

# Término de una hebra

Hebras

- La vida de una hebra termina con los siguientes eventos:

1. La función `start_routine()` termina
2. La hebra invoca `pthread_exit()`
3. La hebra es cancelada por otra hebra con `pthread_cancel()`
4. El proceso al cual la hebra pertenece termina

```
void pthread_exit(void *status);  
int pthread_cancel(pthread_t thread);
```

- Cuando una hebra termina con `pthread_exit()`, puede retornar un valor de éxito (`status`) a la hebra padre.
- La hebra padre puede recuperar dicho valor con

```
int pthread_join(pthread_t thread, void **value_ptr);
```

# Hello world! multihebreado

Hebras

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

void *helloworld(void *param)
{
    printf("Hello World, soy hebra %d\n", param);
    pthread_exit((void *) 100);
}

int main(int argc, char *argv[])
{
    pthread_t tid;          // Identificador de la hebra
    pthread_attr_t attr;    // atributos de la hebra
    int i, status;

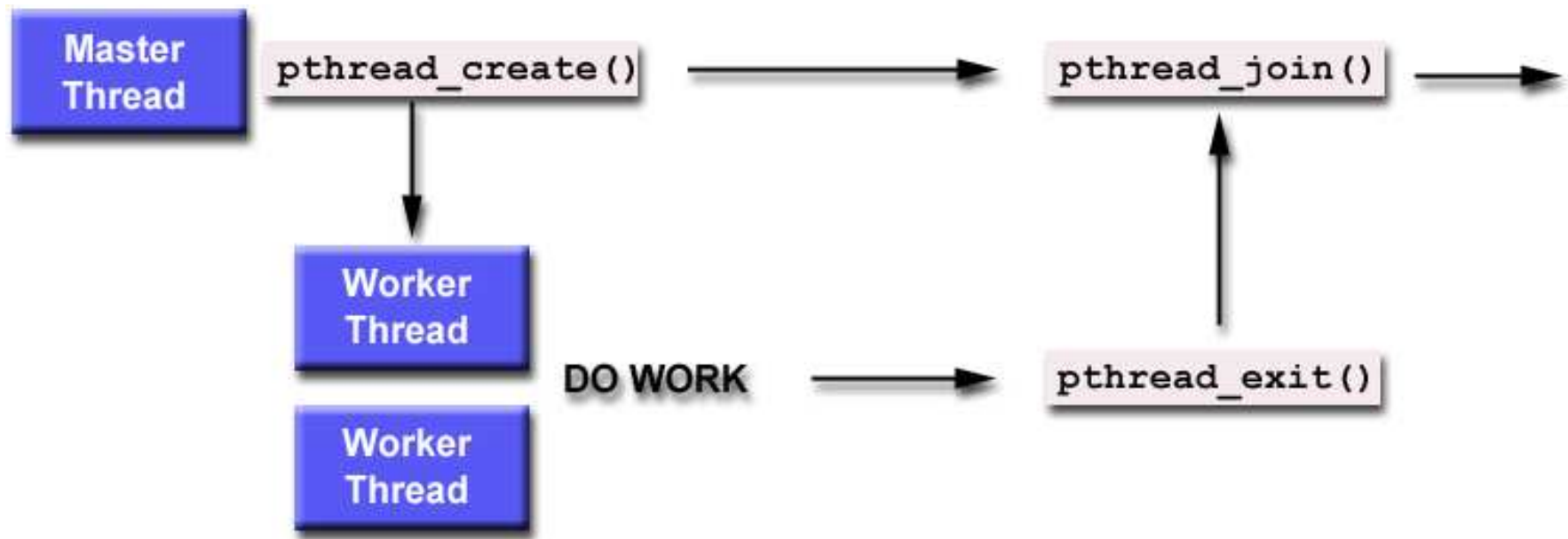
    pthread_attr_init(&attr);
    for (i=1; i<=5; i++) {
        pthread_create(&tid, &attr, helloworld, (void *) i);
        pthread_join(tid, (void **)&status);
        printf("Retorna hebra %d, con valor de retorno %d\n", i, status);
    }

    pthread_exit(0); // Por que es necesario?
}
```

# Atributo Joinable

Hebras

- Por defecto, una hebra es creada como “atachada” a la hebra padre
- Es decir, la hebra padre puede esperar y chequear el término de sus hebras trabajadoras
- Ya vimos que `pthread_join()` espera por una hebra en particular y recupera el valor de retorno



- También se dice que la hebra es *joinable*

# Atributo Detached

Hebras

- Es posible crear una hebra *no joinable* llamando a

```
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
```

antes de `pthread_create()`

- En este caso, la hebra padre NO puede esperar por dicha hebra, es decir no es *joinable*
- También es posible cambiar el atributo después de la creación de la hebra

```
int pthread_detach(pthread_t thread);
```

# Paso de parámetros

Hebras

- La función que ejecuta una hebra, en su inicialización, tiene sólo un argumento y es de tipo void \*
- Si se desea pasar varios argumentos, lo hacemos a través de una estructura
- El paso de parámetros es muy delicado, como lo demuestra el siguiente ejemplo:

```
#define NUMERO_DE_HEBRAS 10
int rc, t;

pthread_t tids[NUMERO_DE_HEBRAS];

for(t=0; t<NUMERO_DE_HEBRAS; t++)
{
    printf("Creando hebra %d\n", t);
    rc = pthread_create(&tids[t], NULL, helloworld, (void *) &t);
    ...
}
```

# Paso de parámetros (cont)

Hebras

```
struct thread_data{
    int  my_tid;
    int  suma;
    char *msg;
};

struct thread_data thread_data_array[NUMERO_DE_HEBRAS];

int main (int argc, char *argv[])
{
    ...
    thread_data_array[t].my_tid = t;
    thread_data_array[t].suma = sum;
    thread_data_array[t].msg = messages[t];
    rc = pthread_create(&tids[t], NULL, helloworld,
        (void *) &thread_data_array[t]);
    ...
}
```



## Paso de parámetros (cont)

Hebras

```
void *helloworld(void *arg)
{
    struct thread_data *my_data;
    ...
    my_data = (struct thread_data *) arg;
    taskid  = my_data->my_tid;
    suma    = my_data->suma;
    hello_msg = my_data->msg;
    ...
}
```