

---

---

# Auxiliar 1: Introducción pthreads

CC4302 - Sistemas Operativos  
José Astorga

---

---

# Threads en C

# Procesos

- Inicialmente, solo existían procesos pesados
  - ◆ No comparten nada de memoria
  - ◆ Para transferir datos: archivos
  - ◆ Requieren gran cantidad de recursos para crearse
  - ◆ Alto sobrecosto de la comunicación
- Necesidad de tener procesos más baratos o livianos

# Procesos livianos

- Threads, hebras o hilos de ejecución
- Procesos que comparten memoria
- Requieren poco costo en su creación (recursos y tiempo)

# Creación de threads

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

- Lanza un nuevo thread que ejecuta la función **start\_routine**.
- La función **start\_routine** recibe un solo argumento: **arg**.
- El ID del nuevo thread se almacena en **\*thread**.
- **attr** contiene atributos especiales para la creación de un thread (por ahora, no usaremos ninguno).
- **pthread\_create** retorna 0 si la creación del thread fue exitosa.

# Término de un thread

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

- Un thread termina si la función **start\_routine** retorna
- También, un thread puede terminar llamando a la función

- ◆ `void pthread_exit(void* return_value)`

- Esto es distinto a llamar a `exit()`

# Esperar el término de un thread

- Alguien tiene que esperar que un thread creado con `pthread_create` termine (“enterrar un thread”):
- Para enterrar un thread se debe invocar:
  - ◆ `int pthread_join(pthread_t thread, void **return_value)`
- Si un thread no es enterrado, se convierte en zombie y no liberará su identificador ni sus recursos utilizados !!
- `pthread_join` retorna 0 en caso de éxito

# Veamos ejemplo 1: Cómo lanzar threads

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

void *thread(void *ptr) {
    char* nombre = (char*) ptr;    // Castear argumento
    printf("Thread - %s\n", nombre); // Trabajo en paralelo
    return NULL;                   // Retorno
}

int main() {
    pthread_t pid_1, pid_2; // Guardar PID de los threads lanzados
    char* nombre_1 = "primero";
    char* nombre_2 = "segundo";

    pthread_create(&pid_1, NULL, thread, nombre_1); // lanzar thread1
    pthread_create(&pid_2, NULL, thread, nombre_2); // lanzar thread2

    pthread_join(pid_1, NULL); // esperar thread 1
    pthread_join(pid_2, NULL); // esperar thread 2
    return 0;
}
```



## ¿Qué hacer si queremos entregar más de un argumento al thread?

- Debemos crear una estructura que reúna todos los argumentos, y luego entregar a `pthread_create` un puntero a dicha estructura.

```
typedef struct {  
    unsigned long x;  
    unsigned int i;  
    unsigned int j;  
    unsigned int res;  
} Args;
```

# Pasos para programación con pthreads

1. Descubrir / diseñar qué parte del algoritmo podemos paralelizar efectivamente.
  2. Crear estructura Args para poder ingresar argumentos a la función a paralelizar.
  3. Programar la función a paralelizar (función que lanza pthread\_create).
- 
- A. Lanzar threads con argumentos correspondientes.
  - B. Esperar que el trabajo paralelo sea realizado (Quizás es necesario realizar trabajo en el thread principal).
  - C. Enterrar los threads lanzados y recolectar los resultados.

Nota: Es una pequeña guía, son pasos generales. A veces se solapan o no necesariamente se hacen en orden!

# Ejemplo: Buscar Factor

→ Suponemos que tenemos la función

```
uint buscarFactor(ulonglong x, uint i, uint j);
```

→ Con esta función podemos calcular fácilmente si un número es primo

```
uint raiz_x= (uint)sqrt((double)x);
```

```
int x_es_primo = buscarFactor(x, 2, raiz_x) == 0
```

# Ejemplo: Buscar Factor

Queremos programar

```
uint buscarFactorParalelo(ulonglong x, uint i, uint j);
```

- Realiza la búsqueda utilizando P cores
- Dividiremos el intervalo de búsqueda [i, j] en P partes

- Pequeño desafío: lanzar P-1 cores y realizar parte de la búsqueda en el thread principal.