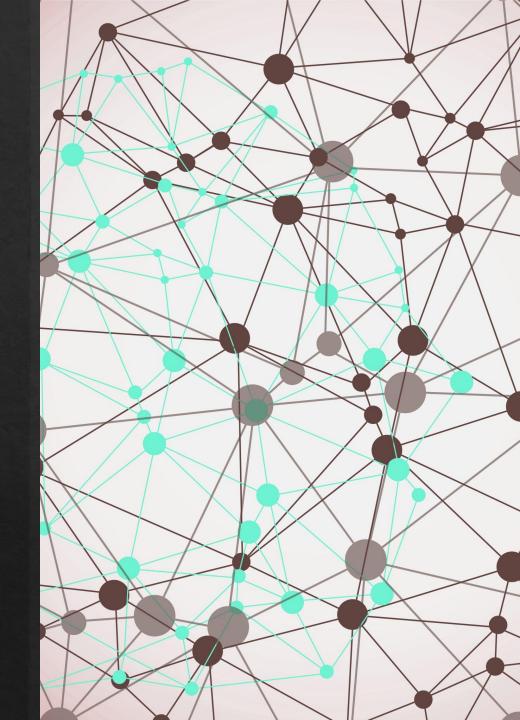
Sistemas Operativos

Introducción a pthreads

Pablo Jaramillo

Diapositivas basadas en las de José Astorga (2023-2)



Su aux

Su auxiliar

Su auxiliar (de la palabra auxilio)

Su auxiliar (de la palabra auxilio)

(al loco que le pueden gritar "auxilio auxiliar!")

Pablo



Auxiliar del ramo por tercera vez

Ayudante del ramo antes de eso

$Pablo_{\text{(Jara, si prefieren)}}$



Auxiliar del ramo por tercera vez

Ayudante del ramo antes de esc

Aquí para enseñar y ayudar!

Hagan preguntas!!

Si me ven en la U no duden en preguntar (y saludar)

Si necesitan ayuda escriban correo y manden su código completo

Threads en C

Procesos en paralelo

→ Procesos pesados

- Lo que vieron en PSS
- No comparten memoria
- Sirven para transferir data
- Costosos de instanciar
- Costosos para comunicarse entre ellos

Se presenta la necesidad de tener procesos menos costosos para aplicaciones más ágiles y versátiles

→ Procesos livianos

- Llamados threads (o hilos de ejecución)
- Pueden compartir memoria
- Son baratos de instanciar

Creación de threads

- → Lanza un nuevo thread que ejecuta la función start_routine.
- → start_routine puede tomar solo 1 argumento: arg.
- → El ID del thread corresponde a *thread.
- → attr corresponde a atributos de creación del thread (No los usaremos en este curso).
- → pthread_create retorna 0 si la creación del thread fue exitosa.

Término de un thread

- → Un thread termina si start_routine retorna.
- → Un thread puede terminarse llamando desde este a pthread_exit
 - int pthread_exit(void *return_value) // No confundir con exit()
- → El hilo que maneje la lógica coordinada debe encargarse de esperar el término de los threads, creados con pthread_create esto se hace con pthread_join ("enterrar un thread")

```
int pthread_join(pthread_t thread, void **return_value)
```

- → threads no enterrados se convierten en zombies y no devolverán sus recursos asignados
- → pthread_join retorna 0 en caso de éxito

Ejemplo:

```
#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;
```

¿Cómo puedo usar más argumentos?

¿Cómo puedo usar más argumentos?

Debemos usar una estructura para empaquetar todos los argumentos y entregar esta

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

Programación con pthreads, how to?

Diseño

- 1. Analizar cuáles partes del algoritmo puede ser **efectivamente** paralelizado.
- 2. Crear estructura Args para ingresar los argumentos necesarios.
- 3. Programar función a paralelizar (la función que lanza pthread_create).

Lógica

- 1. Lanzar threads con argumentos correspondientes.
- 2. Esperar a que el trabajo paralelizado se realice (Inclusive el del thread principal, si aplica).
- 3. Enterrar los threads lanzados y recolectar resultados.

*Nota:

Esto no es una receta, lo presente acá es una guía con pasos generales que pueden solaparse, repetirse o cambiar de orden.

Ejercicio: Buscar Factor

Paralelicemos esta función que busca *cualquier* factor de un número para acelerarla utilizando P cores

```
#include <pthread.h>

typedef unsigned long long ulonglong;
typedef unsigned int uint;

// busca un factor del número entero x en el rango [i, j]
uint buscarFactor(ulonglong x, uint i, uint j){
    for (uint k = i; k <= j; k++){
        if (x % k == 0)
            return k;
    }
    return 0;
}</pre>
```

Idea

^{*}Desafío: Lanzar P-1 threads y utilizar el principal en la búsqueda.

Ejercicio: Buscar Factor

Paralelicemos esta función que busca *cualquier* factor de un número para acelerarla utilizando P cores

```
#include <pthread.h>

typedef unsigned long long ulonglong;
typedef unsigned int uint;

// busca un factor del número entero x en el rango [i, j]
uint buscarFactor(ulonglong x, uint i, uint j){
    for (uint k = i; k <= j; k++){
        if (x % k == 0)
        return k;
    }
    return 0;
}</pre>
```

Idea

Dividir el intervalo de búsqueda en P partes

^{*}Desafío: Lanzar P-1 threads y utilizar el principal en la búsqueda.