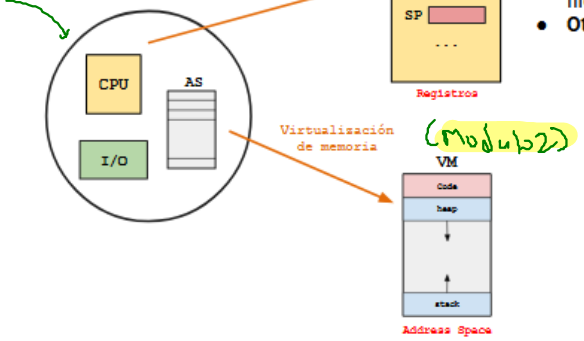
14/10/2024 - Sistemas Operativos (U de @)

1. Contextualización

Procesos y virtualización

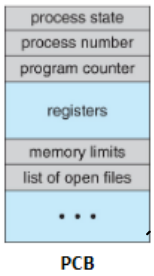
Proceso = CPU + Mem + I/O



(Modulo1)

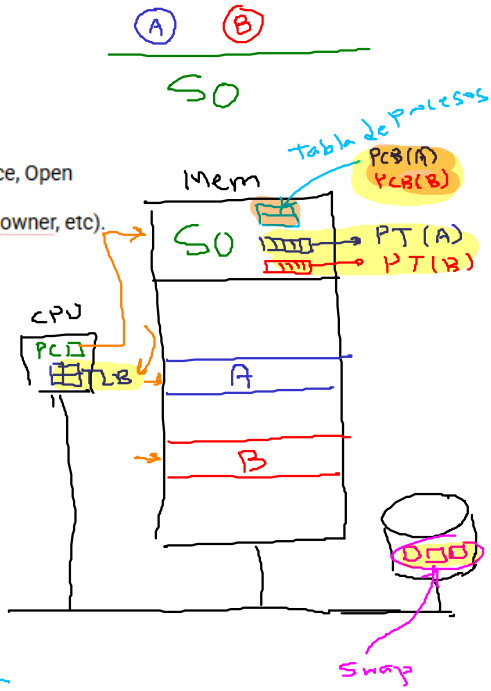
Un proceso incluye:

- **CPU context** (Registers)
- **OS Resources** (Address Space, Open files, etc).
- **Otra información** (PID, state, owner, etc).



PCB

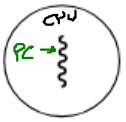
(Modulo2)



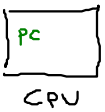
Procesos e hilos

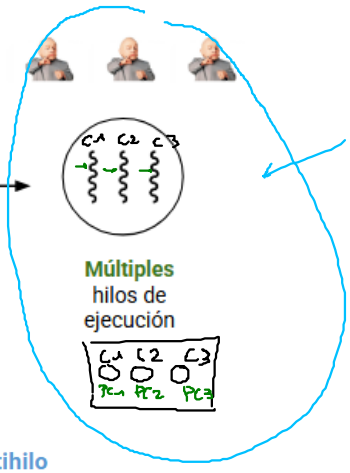
Proceso

Hilo



Un hilo de ejecución

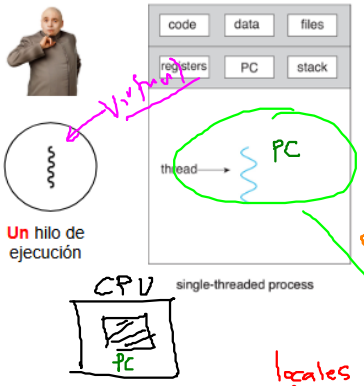




Aplicaciones multihilo

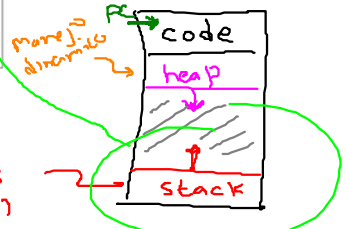
Como es el mapa de Memoria?

Proceso con un solo hilo

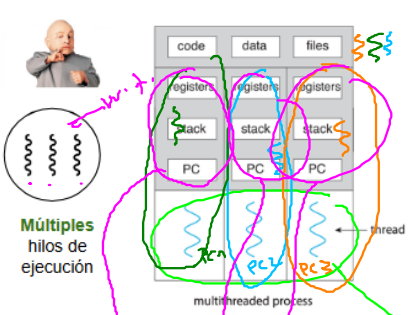


Un proceso con un solo hilo se caracteriza por que:

- Tiene un único hilo de ejecución (a cuyo estado se asocian los registros de la CPU: PC, SP, PGs, etc.).
- Un espacio de direcciones único (code, stack y heap).

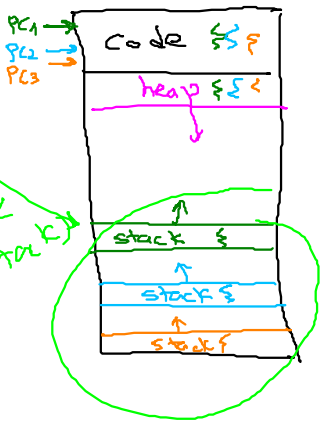


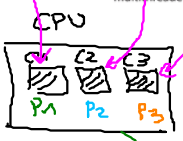
Proceso multihilo



Un proceso con varios hilos se caracteriza por:

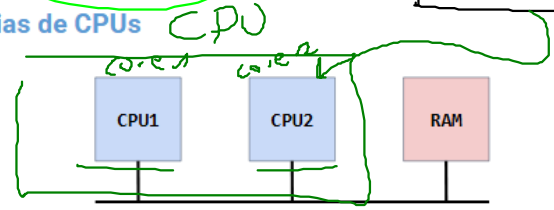
- Múltiples puntos de ejecución.
- Múltiples PC (Program Counter)
- Compartir el mismo espacio de direccionamiento entre hilos.

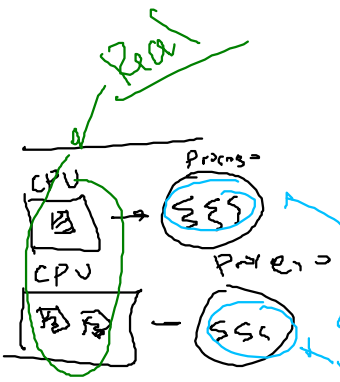




Varios stack (multihilo)

Tendencias de CPUs

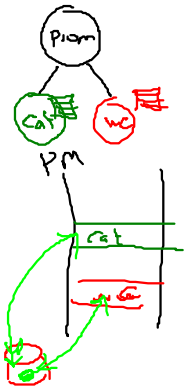


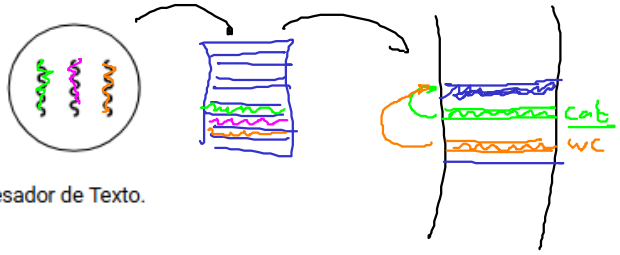


- **Tendencia:**
  - Igual velocidad.
  - Más núcleos.
- Programas más rápidos → Ejecución concurrente.
- **Objetivo**: Escribir aplicaciones que utilicen varias CPUs → **¿Oh y ahora quién podrá ayudarme?**

Desarrollo de aplicaciones concurrentes

Aproximación 1	Aproximación 2
• Construir aplicaciones a partir de procesos que se comunican entre sí.	• Usando una nueva abstracción conocida como thread (hilo - hebra - proceso ligero - miniproceso). • Los hilos son como los procesos excepto que, múltiples hilos de un mismo proceso comparten el mismo espacio de direcciones (AS).
• **Ejemplo**: Pestañas de un navegador.	• **Ejemplo**: Procesador de Texto.





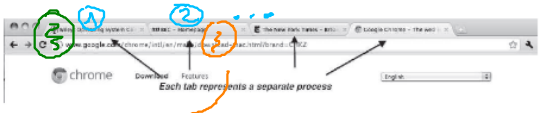
16/10/2025 – Sistemas Operativos (Ude@)

2. Aplicaciones que usan concurrencia.

a. Navegador

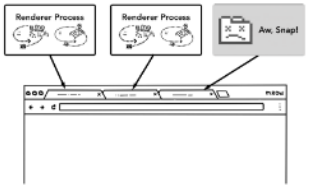
Ejemplos - Browser

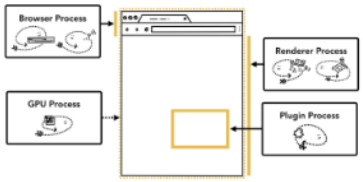
- Un browser es una aplicación que permite desplegar el contenido de un sitio web.
- Muchos navegadores modernos permiten navegar a través de diferentes páginas usando pestañas.
- Cada una de las pestañas del navegador representa un proceso separado.



- **Problema**:
  - Cuando el contenido dinámico (aplicaciones hechas en javascript, flash, html 5) de un website tiene bugs, este puede hacer ralentizar o incluso bloquear el browser.
  - Cuando una aplicación web bloquea una pestaña, el proceso entero (incluyendo todos los sitios webs desplegados en las demas pestañas) tambien se pueden bloquear.

Browser





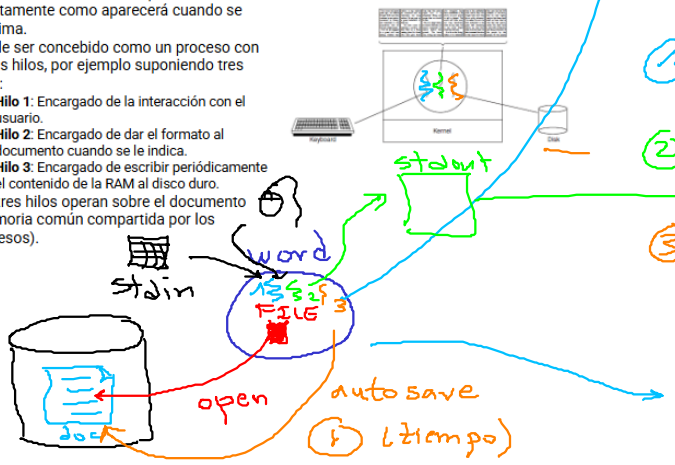
Link animación

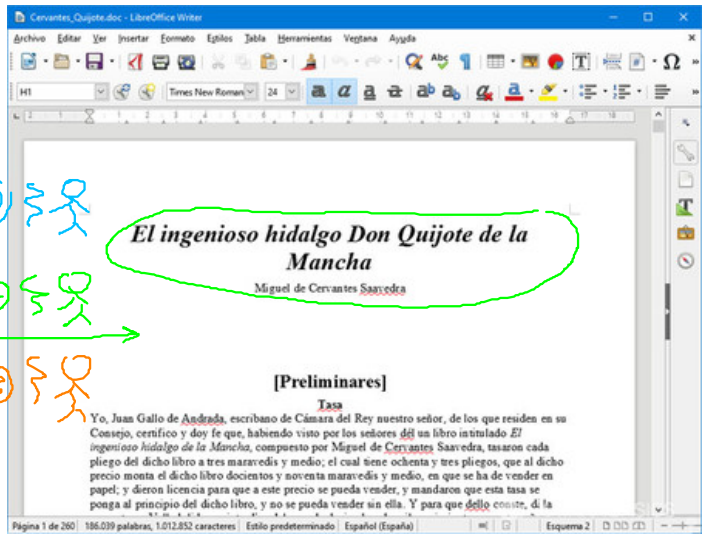
- **Solución:** Arquitectura multiproceso de google donde se identifican tres tipos de procesos (link):
  - Browser: Encargado de la interfaz de usuario, el disco y los datos de red.
  - Render: Contiene la lógica para renderizar las páginas web (html, javascript, imágenes, etc).
  - Plug-in: Proceso creado para cada tipo de pluging (flash, quicktime)

b. Procesador de texto

Ejemplo - Procesador de texto

- Un procesador de texto es un programa que muestra un documento en pantalla exactamente como aparecerá cuando se imprima.
- Puede ser concebido como un proceso con varios hilos, por ejemplo suponiendo tres hilos:
  - **Hilo 1**: Encargado de la interacción con el usuario.
  - **Hilo 2**: Encargado de dar el formato al documento cuando se le indica.
  - **Hilo 3**: Encargado de escribir periódicamente el contenido de la RAM al disco duro.
- Los tres hilos operan sobre el documento (memoria común compartida por los procesos).

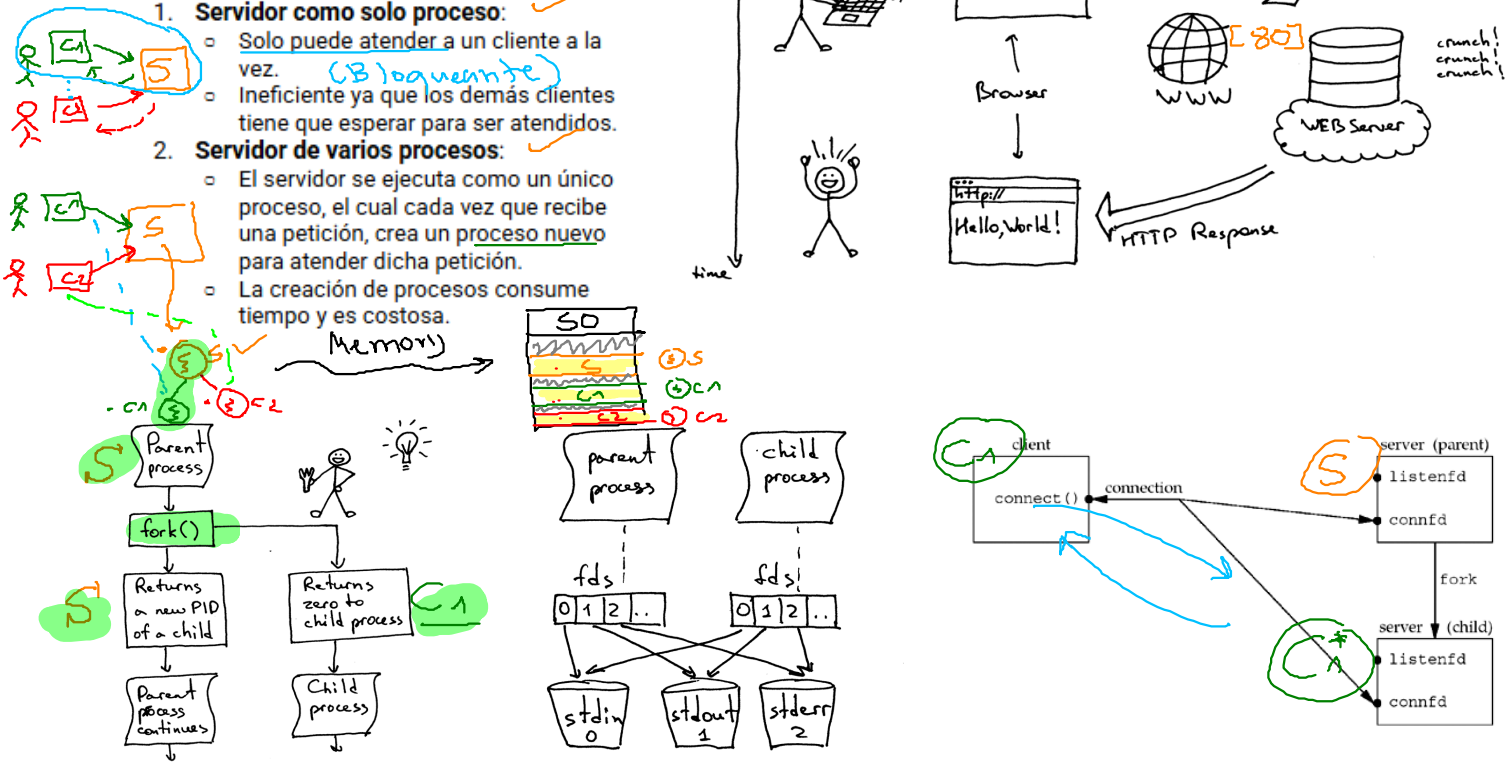




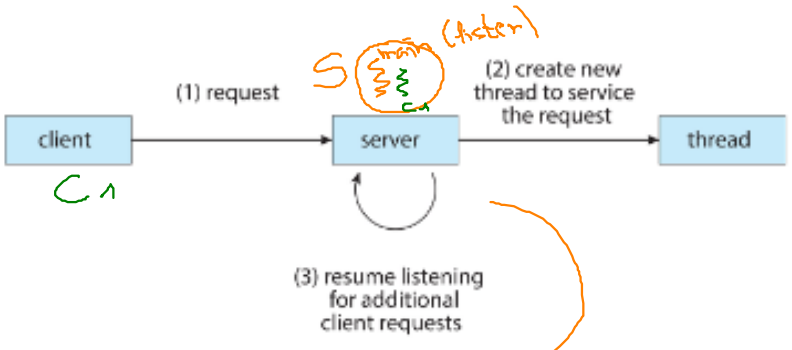
Ejemplo - Servidor

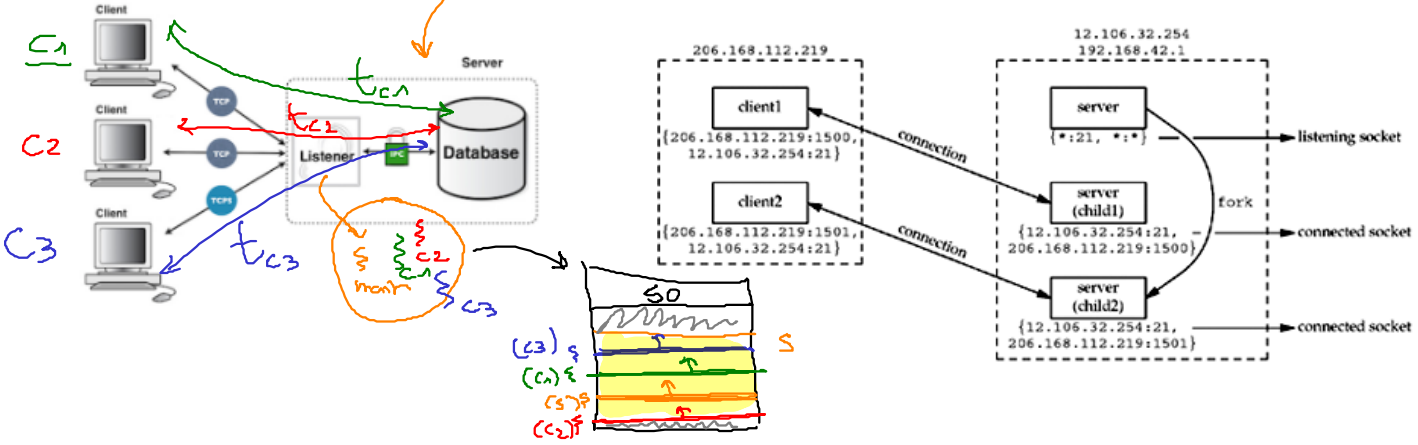
Un servidor acepta y atiende peticiones hechas por clientes:

1. **Servidor como solo proceso**:
   - Solo puede atender a un cliente a la vez. (Bloqueante)
   - Ineficiente ya que los demás clientes tiene que esperar para ser atendidos.
2. **Servidor de varios procesos**:
   - El servidor se ejecuta como un único proceso, el cual cada vez que recibe una petición, crea un proceso nuevo para atender dicha petición.
   - La creación de procesos consume tiempo y es costosa.



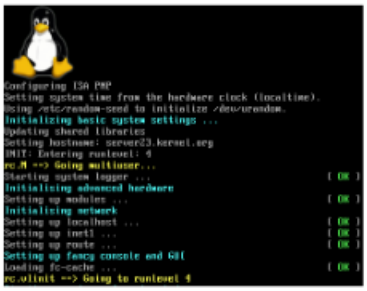
3. **Servidor Multihilo**:
   - El servidor se implementa usando un proceso con múltiples hilos. Para ello:
     - El servidor crea un hilo separado que escucha por las peticiones de los clientes.
     - Cuando una petición es recibida, el servidor crea un nuevo hilo para atender dicha petición y continúa pendiente de más peticiones.

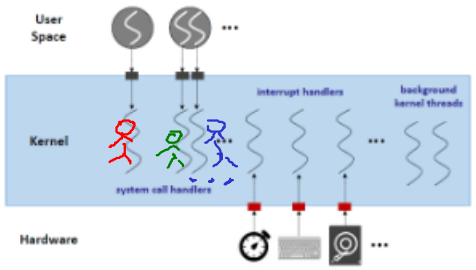


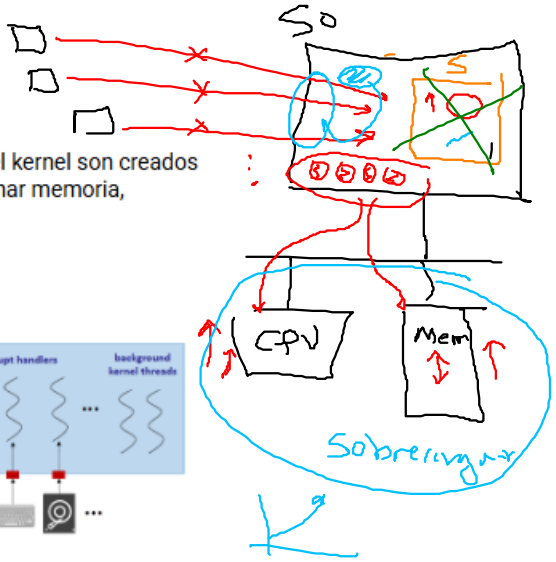


Ejemplo - Sistemas operativos

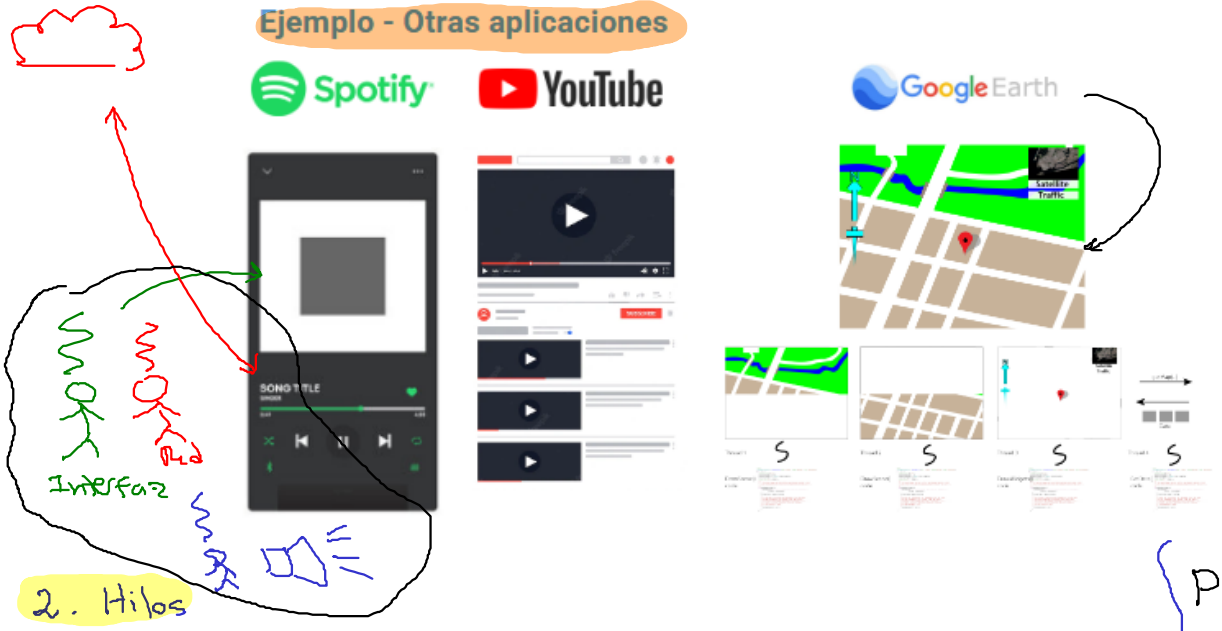
- Muchos sistemas operativos también son multihilo.
  - En los sistemas Linux, durante el tiempo de booteo, varios hilos del kernel son creados para realizar una tarea específica (administrar dispositivos, gestionar memoria, manejar interrupciones, etc).







Ejemplo - Otras aplicaciones

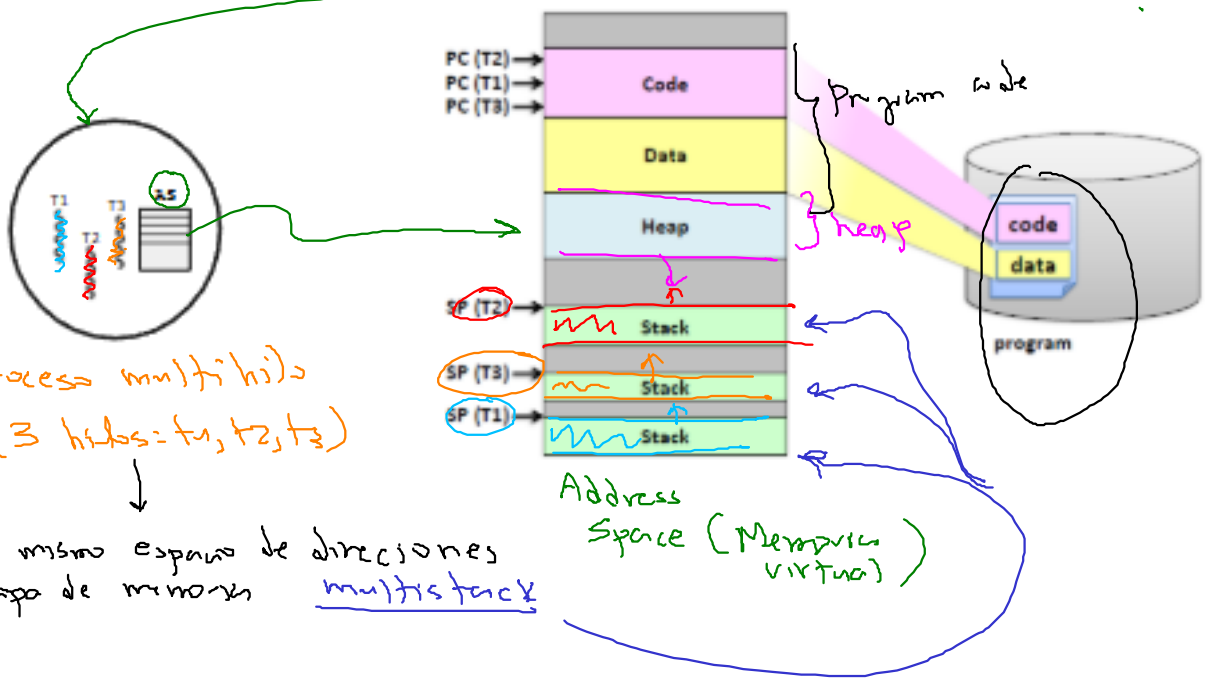


Abstracción → Proceso → Hilo

Aspectos importantes

- Un hilo de control hace referencia a una secuencia de instrucciones que están siendo ejecutadas en un programa.
- Cada hilo (thread) tiene:
  - Un Thread ID (TID)
  - Un conjunto de registros asociados, incluidos el PC y el SP.
  - Stack
- Cada hilo comparte (con otros los demás hilos si el proceso es multihilo):
  - Code. (Data + Text).
  - Heap.

¿Como es un proceso cuando tiene varios hilos?



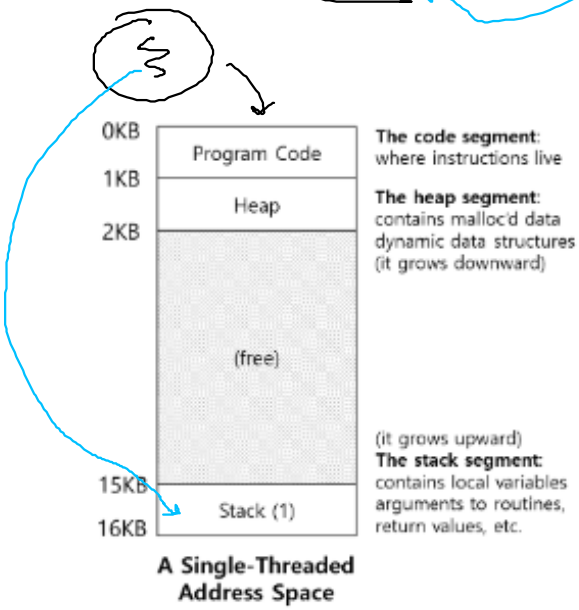
Proceso multihilo (3 hilos: t1, t2, t3)

- El mismo espacio de direcciones
- Mapa de memoria multistack

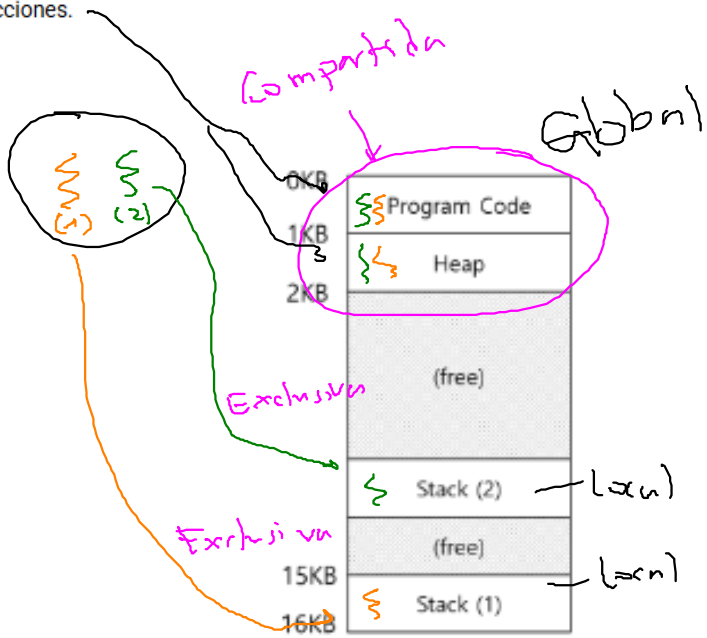
Address Space (Memoria virtual)

Hilos y espacios de direcciones - Address Space multi-stack

- Todos los hilos comparten los segmentos code y heap del espacio de direcciones.
- Cada hilo tiene su propio stack.



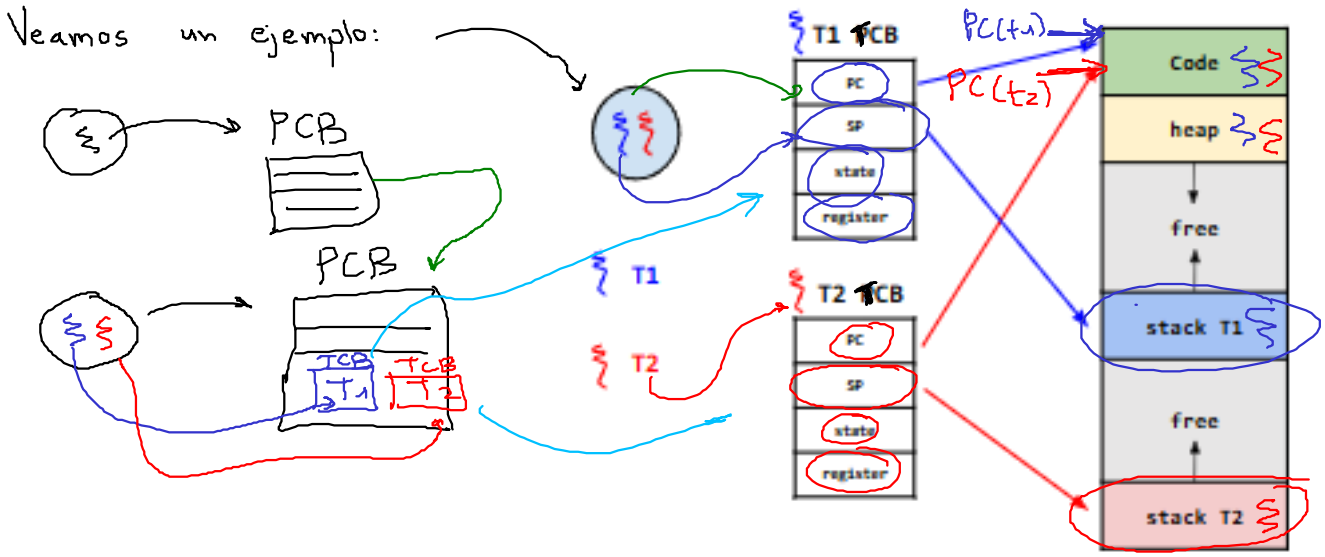
A Single-Threaded Address Space



Two threaded Address Space

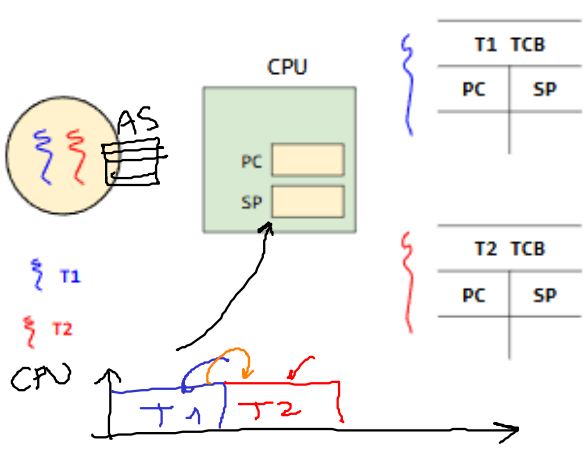
Thread control block

Estructura que almacena el estado de cada hilo.



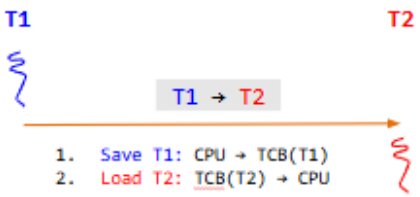
Cambio de contexto

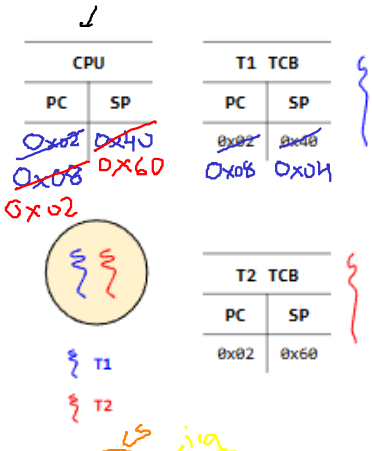
- El cambio de contexto entre hilos es similar al de los procesos. Sin embargo, a diferencia del caso para los procesos, en el cambio de contexto entre hilos, el espacio de direcciones no cambia.

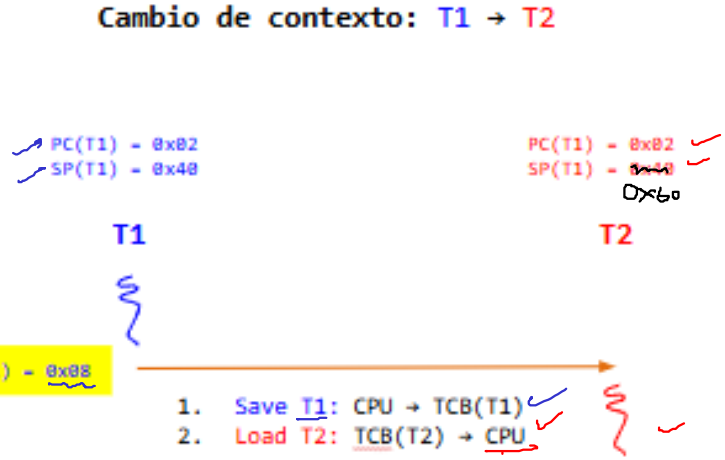


Cambio de contexto entre hilos T1 a T2:

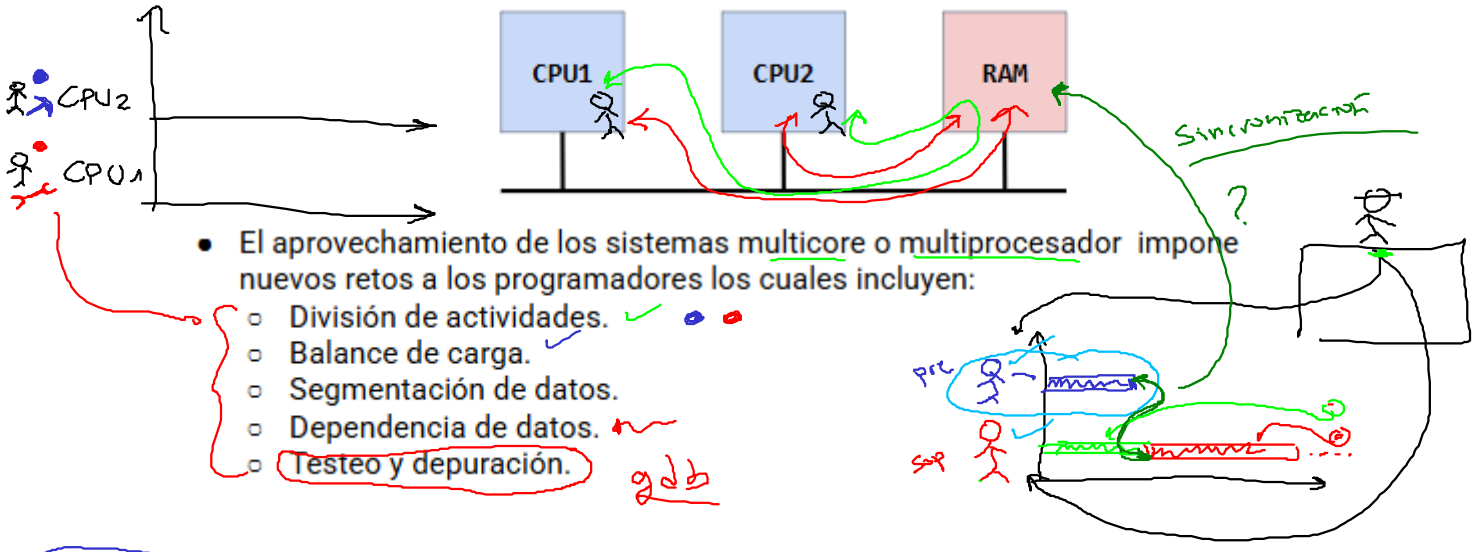
- Se almacenan los valores de los registros de T1.
- Se cargan los valores de los registros T2.
- Se mantiene el mismo espacio de direccionamiento.







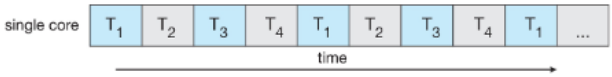
3. Concurrencia .vs. Paralelismo



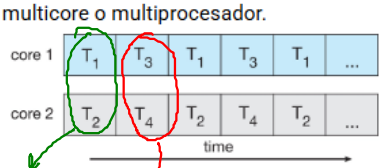
- El aprovechamiento de los sistemas multicore o multiprocesador impone nuevos retos a los programadores los cuales incluyen:
  - División de actividades.
  - Balance de carga.
  - Segmentación de datos.
  - Dependencia de datos.
  - Testeo y depuración.

gdb

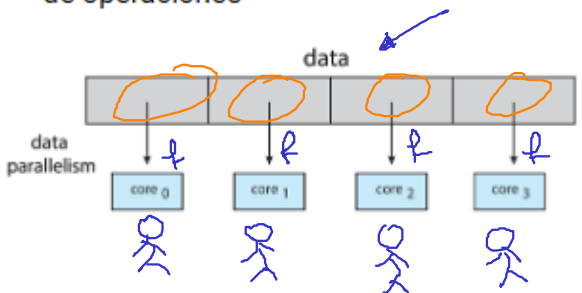
- **Concurrencia**:
  - Permite la ejecución de más de una tarea conforme transcurre el tiempo.
  - En un sistema con un solo procesador, el scheduler proporciona la concurrencia.



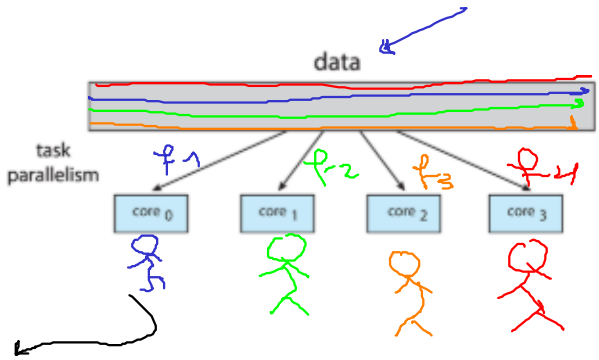
- **Paralelismo**:
  - Implica que un sistema puede hacer de manera simultánea (a la vez) más de una tarea.
  - Propio de sistemas multicore o multiprocesador.

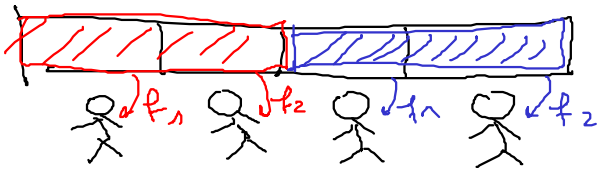


- **Paralelismo de datos**: Divide los datos en subconjuntos los cuales son procesados por diferentes cores realizando el mismo conjunto de operaciones



- **Paralelismo de tareas**: Distribuye hilos a través de los núcleos de tal manera que cada hilo realiza una única operación.





4. Uso de los hilos

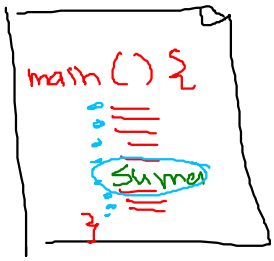
21/10/2025 - Sistemas Operativos (Vde@)

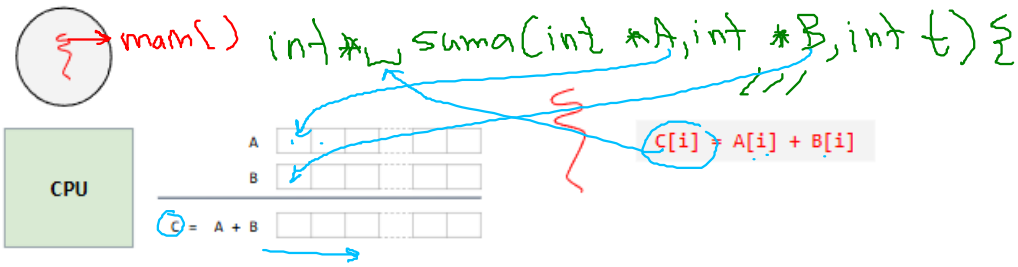
* Paralelización

¿Por qué se usan los hilos? - Paralelizar

Suponga que se desean sumar dos arreglos muy grandes (A y B) y llevar el resultado a un tercero. ¿Como se podría mejorar el desempeño de la aplicación?

Una sola CPU:

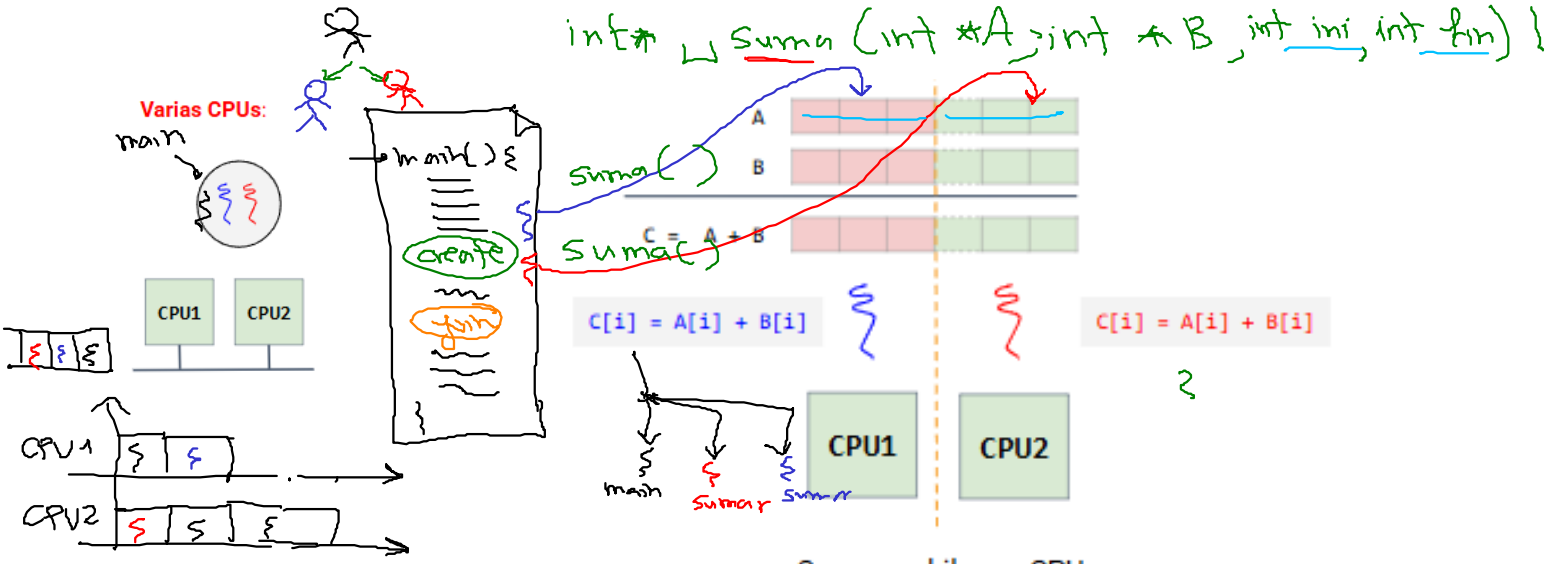




¿Por qué se usan los hilos? - Paralelizar

Suponga que se desean sumar dos arreglos muy grandes (A y B) y llevar el resultado a un tercero. ¿Como se podría mejorar el desempeño de la aplicación?

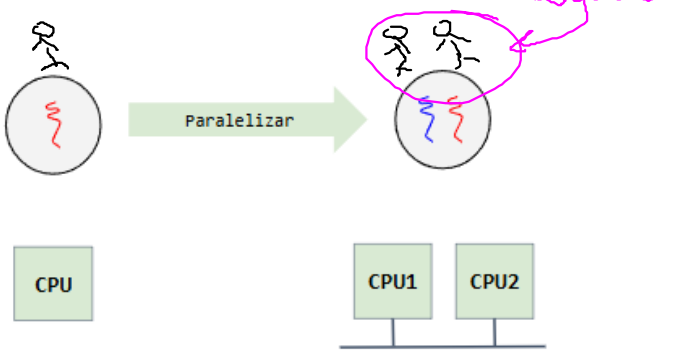
Varias CPUs:



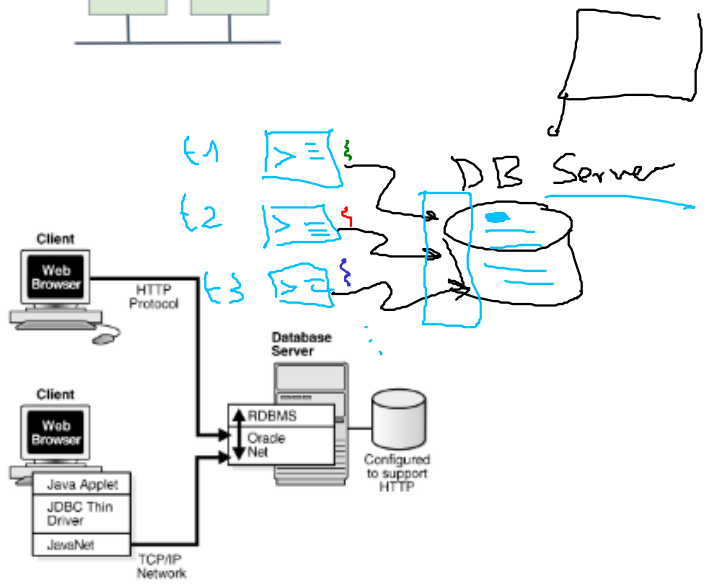
Se usa un hilo por CPU

¿Por qué se usan los hilos? - Paralelizar

Paralelizar consiste en transformar un programa estándar de un solo hilo a **varios hilos** de ejecución.



* Overlap:

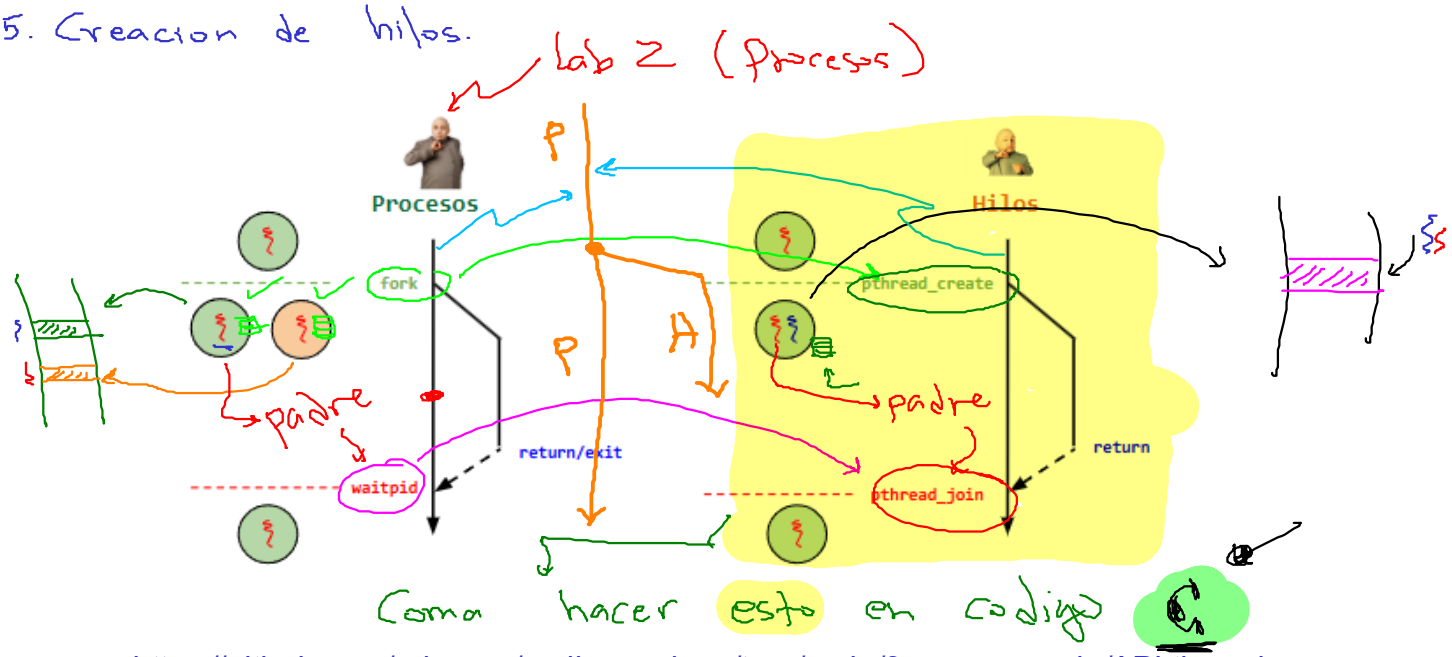


¿Por qué se usan los hilos? - Overlap

Evitar bloquear el progreso de un programa debido a una operación I/O más lenta pues al usar hilos:

- No se hace necesario que un programa tenga que esperar que una operación I/O (la cual es muy lenta) se complete de modo que es posible que la CPU realice otras tareas.
- Casos típicos: Aplicaciones tipo cliente - servidor (web server, DBMS, etc).

5. Creacion de hilos.



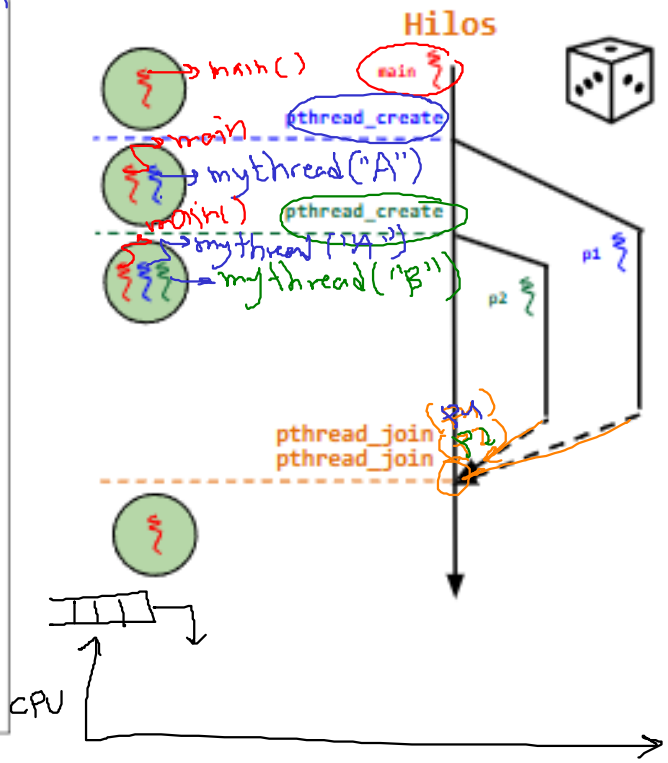
https://github.com/udea-so/codigos_clase/tree/main/3-concurrencia/API-thread

* Desafio 1: Orden de ejecución no predecible

Ejemplo - Orden de ejecución

- El siguiente programa crea dos hilos independientes (**p1** y **p2**) que ejecutan una tarea (mythread)
- **Pregunta**: ¿Cuál es el orden de ejecución de los hilos?

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

#include "common.h"
#include "common_threads.h"

void *mythread(void *arg) {
    printf("%s\n", (char *) arg);
    return NULL;
}

int main(int argc, char *argv[]) {
    if (argc != 1) {
    fprintf(stderr, "usage: main\n");
    exit(1);
     }

    pthread_t p1, p2;
    printf("main: begin\n");
    Pthread_create(&p1, NULL, mythread, "A");
    Pthread_create(&p2, NULL, mythread, "B");
    // join waits for the threads to finish
    Pthread_join(p1, NULL);
    Pthread_join(p2, NULL);
    printf("main: end\n");
    return 0;
}
```

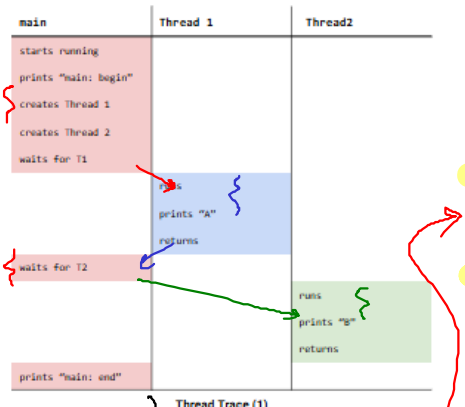


Ejemplo - Orden de ejecución

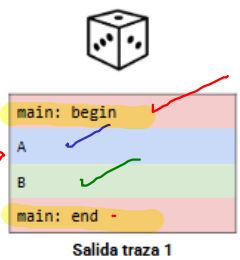
- **Pregunta**: ¿Cual hilo se ejecuta primero?
- **Respuesta**:
  - El orden es indeterminado (aleatorio).
  - Depende del scheduler.

La salida es no determinista (Aleatoria)

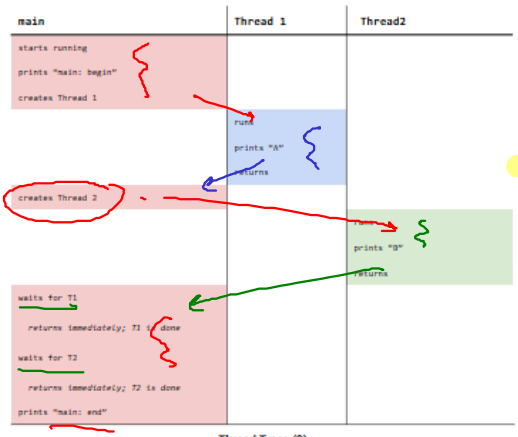
Ejemplo - Orden de ejecución - Traza 1



Thread Trace (1)

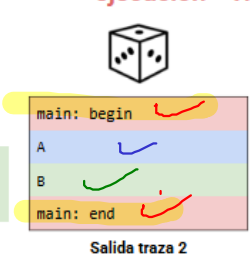


Salida traza 1

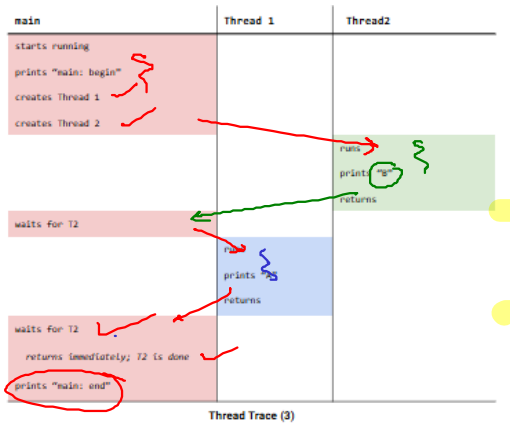


Thread Trace (2)

Ejemplo - Orden de ejecución - Traza 2



Salida traza 2

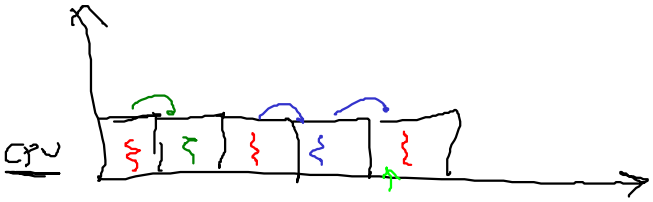


Thread Trace (3)

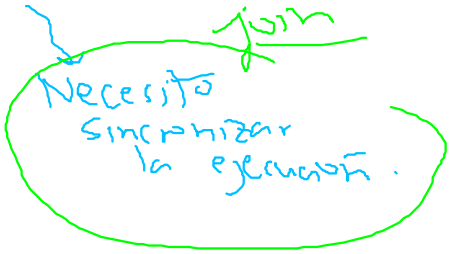
Ejemplo - Orden de ejecución - Traza 3

```
main: begin
B
A
main: end
```

Salida traza 3



Salida imprevisible

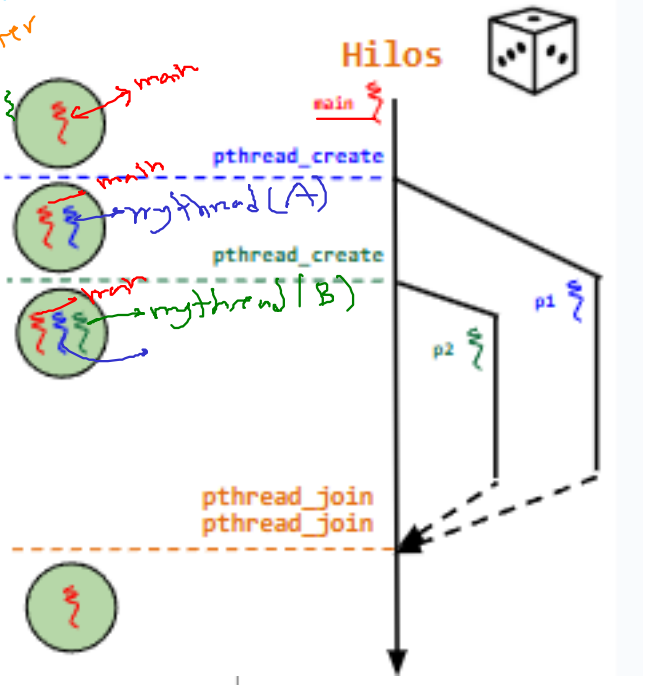


* Desafío 2: Manejo de datos compartidos

Ejemplo - Datos compartidos

- El programa crea dos hilos independientes (**p1** y **p2**) que ejecutan una tarea (**mythread**)
- Ambos hilos intentan actualizar una variable global (**counter**)
- **Pregunta**: ¿Qué implicaciones trae esto?


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

#include "common.h"
#include "common_threads.h"

int max;
volatile int counter = 0; // shared global variable

void *mythread(void *arg) {
  char *letter = arg;
  int i; // stack (private per thread)
  printf("%s: begin [addr of i: %p]\n", letter, &i);
  for (i = 0; i < max; i++) {
    counter = counter + 1; // shared: only one
  }
  printf("%s: done\n", letter);
  return NULL;
}

int main(int argc, char *argv[]) {
  if (argc != 2) {
    fprintf(stderr, "usage: main-first <loopcount>\n");
    exit(1);
  }
  max = atoi(argv[1]);

  pthread_t p1, p2;
  printf("main: begin [counter = %d] [%x]\n", counter,
         (unsigned int) &counter);
  Pthread_create(&p1, NULL, mythread, "A");
  Pthread_create(&p2, NULL, mythread, "B");
  // join waits for the threads to finish
  Pthread_join(p1, NULL);
  Pthread_join(p2, NULL);
  printf("main: done\n [counter: %d]\n [should: %d]\n",
         counter, max*2);
  return 0;
}
```



Ejecución 1 - Salida

```
prompt> gcc -o main main.c -Wall -pthread; ./main
main: begin (counter = 0)
A: begin
B: begin
A: done
B: done
main: done with both (counter = 20000000)
```

Ejecución 2 - Salida

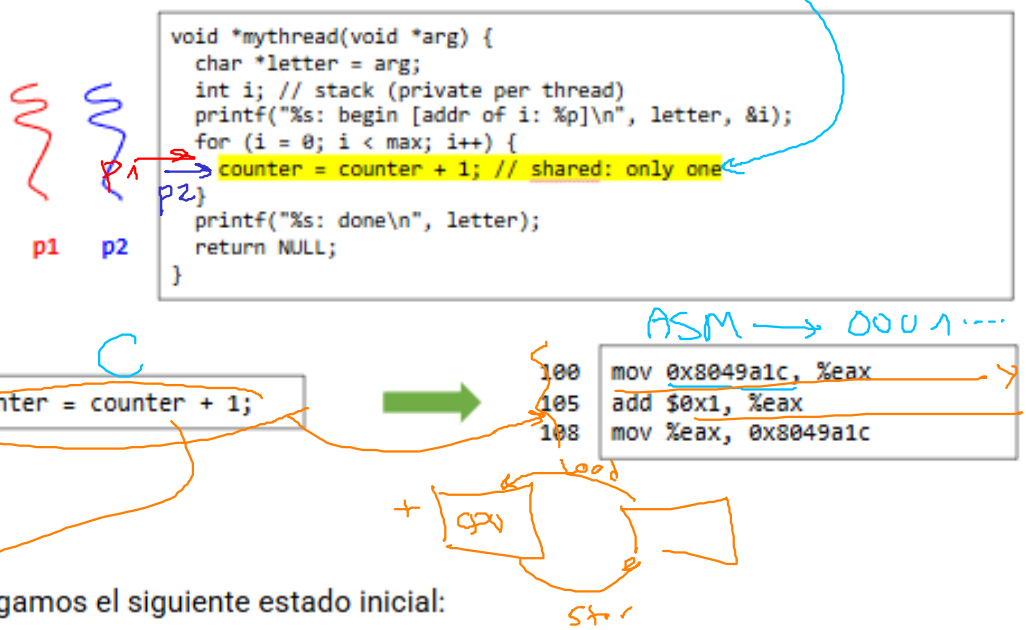
```
prompt> ./main
main: begin (counter = 0)
A: begin
B: begin
A: done
B: done
main: done with both (counter = 19345221)
```

Ejecución 3 - Salida

```
prompt> ./main
main: begin (counter = 0)
A: begin
B: begin
A: done
B: done
main: done with both (counter = 19221041)
```

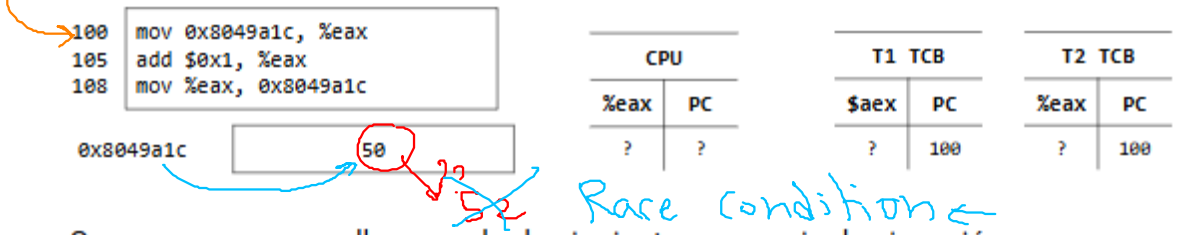
¿Por que estos resultados tan raros?

- Salida **no determinista**:
  - Diferente orden de ejecución para cada caso.
  - Diferentes valores desplegados en la variable **counter**.

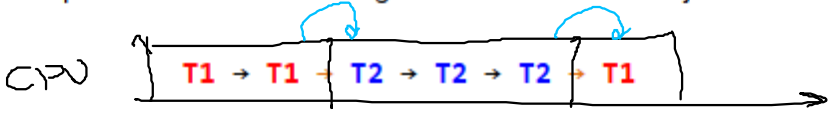


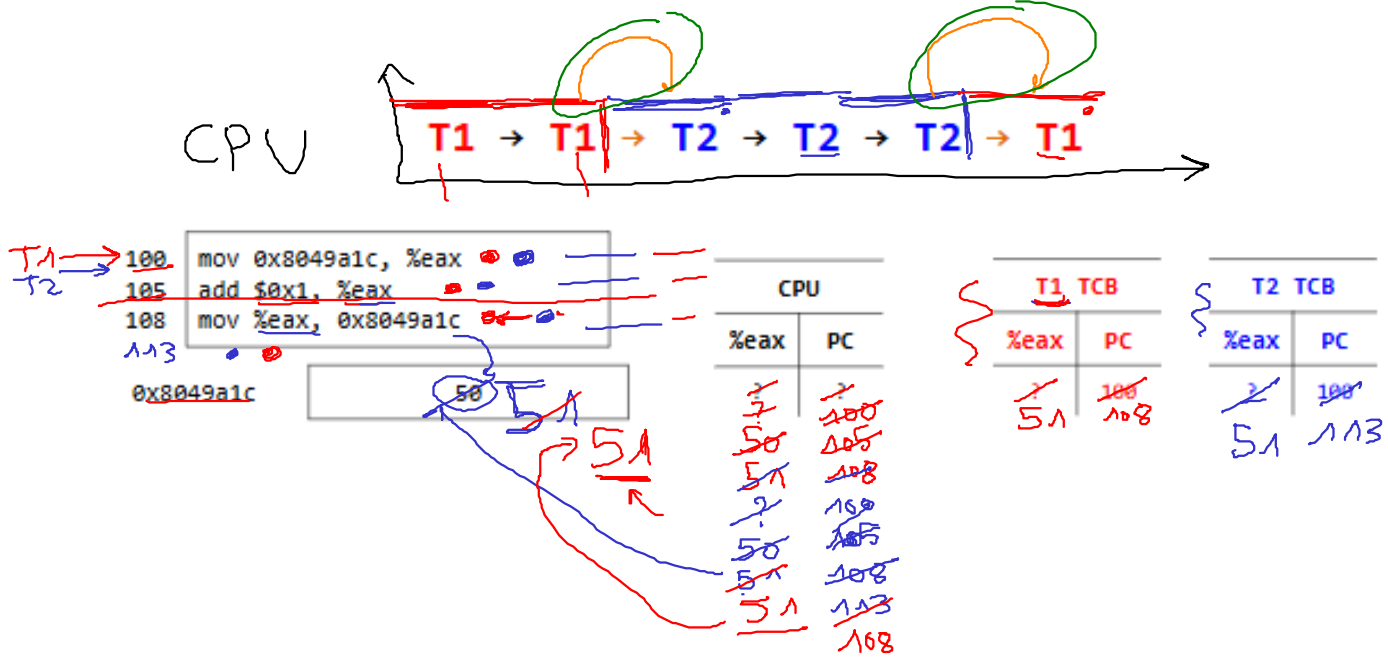
Análisis

- Supongamos el siguiente estado inicial:



- Supongamos que se lleva a cabo la siguiente secuencia de ejecución:





OS	Thread 1	Thread 2	PC	eax	counter (after instruction)
	before critical section		100	0	50
	mov 8049a1c,%eax		105	50	50
	add $0x1,%eax		108	51	50
interrupt					
save T1					
restore T2			100	0	50
		mov 8049a1c,%eax	105	50	50
		add $0x1,%eax	108	51	50
		mov %eax,8049a1c	113	51	51
interrupt					
save T2					
restore T1			108	51	51
	mov %eax,8049a1c		113	51	51 ✗

Conclusiones del análisis

- En el ejemplo con los dos hilos se analizó la instrucción:

```
counter = counter + 1
```

- Se asumió que la variable counter tenía un valor de `50` (por defecto).
- Tras la ejecución de esta instrucción por parte de ambos hilos se espera valor final de `52`, pero el resultado fue de **`51`**

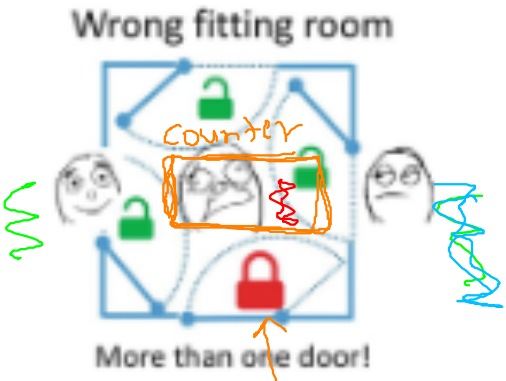
¿Que paso entonces?

Condición de carrera (Race condition - Data race)

- Los dos hilos compiten modificando variables compartidas.
- Resultados no deterministas (por eso no dio lo que esperamos).

Sección crítica

- Los dos hilos compiten modificando variables compartidas.
- Resultados no deterministas (por eso no dio lo que esperamos).



* Definición importante : Seccion critica -

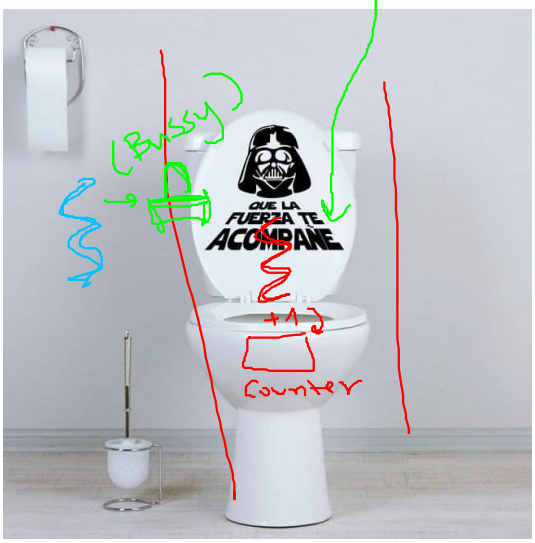
Sección crítica

Porción de código donde se accede a una variable compartida.

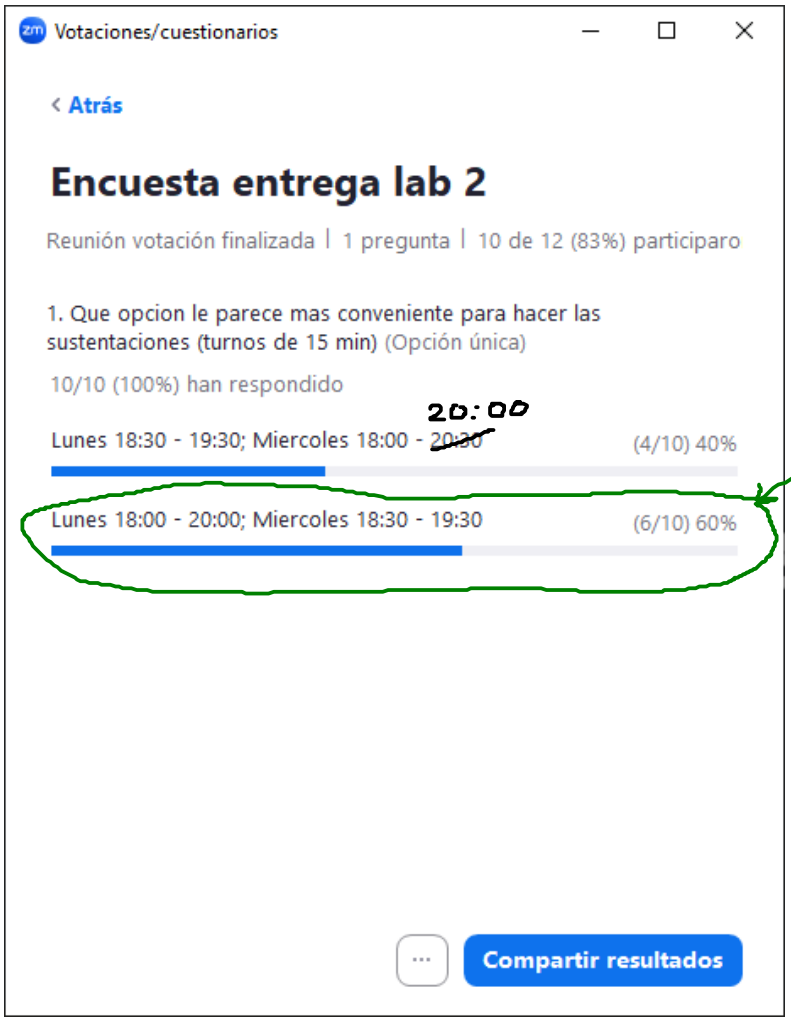
```
void *mythread(void *arg) {
  char *letter = arg;
  int i; // stack (private per thread)
  printf("%s: begin [addr of i: %p]\n", letter, &i);
  for (i = 0; i < max; i++) {
    counter = counter + 1; // sección critica
  }
  printf("%s: done\n", letter);
  return NULL;
}
```

```
counter = counter + 1;
```

```
100 mov 0x8049a1c, %eax
105 add $0x1, %eax
108 mov %eax, 0x8049a1c
```



23/10/2025 - Sistemas Operativos (Ude@)

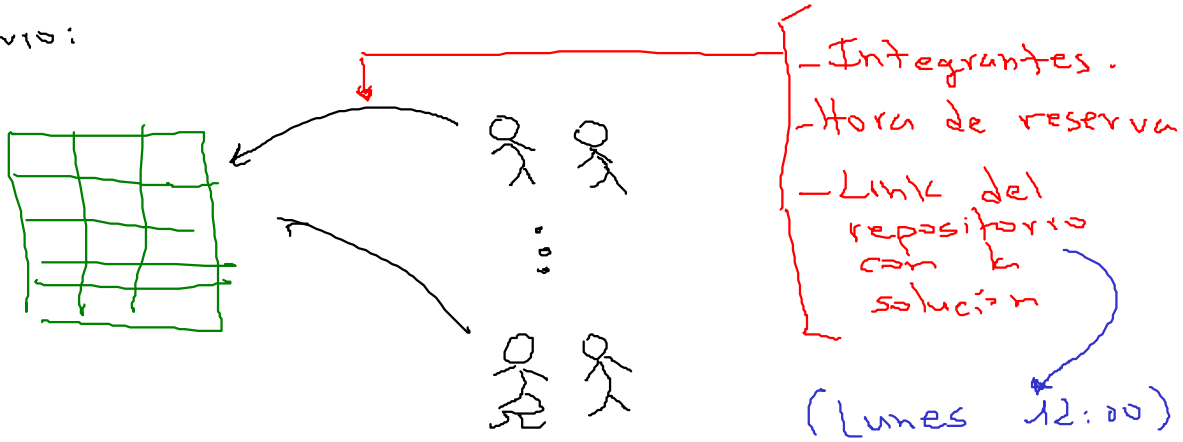


Winner:

- Lunes: 8
- Miercoles: Resto (4)

Pendiente:

1. Agenda por google meet para cada equipo
   (Se envia apenas se acabe la clase)
2. Formulario:



Sección crítica

- No debería ser accedida de manera concurrente:
  - Múltiples hilos ejecutando una región crítica pueden generar un **race condition**.
- Se requiere el **soporte de atomicidad** (instrucciones especiales que no pueden ser interrumpidas) para las regiones críticas
  - **Exclusión mútua**: Propiedad que garantiza que un solo hilo se encuentre ejecutando el código de la región crítica

No hay atomicidad

```
mov 0x8049a1c, %eax
add $0x1, %eax
mov %eax, 0x8049a1c
```

Instrucción atómica

```
memory-add 0x8049a1c, $0x1
```

6. API Threads

¿Como crear hilos?

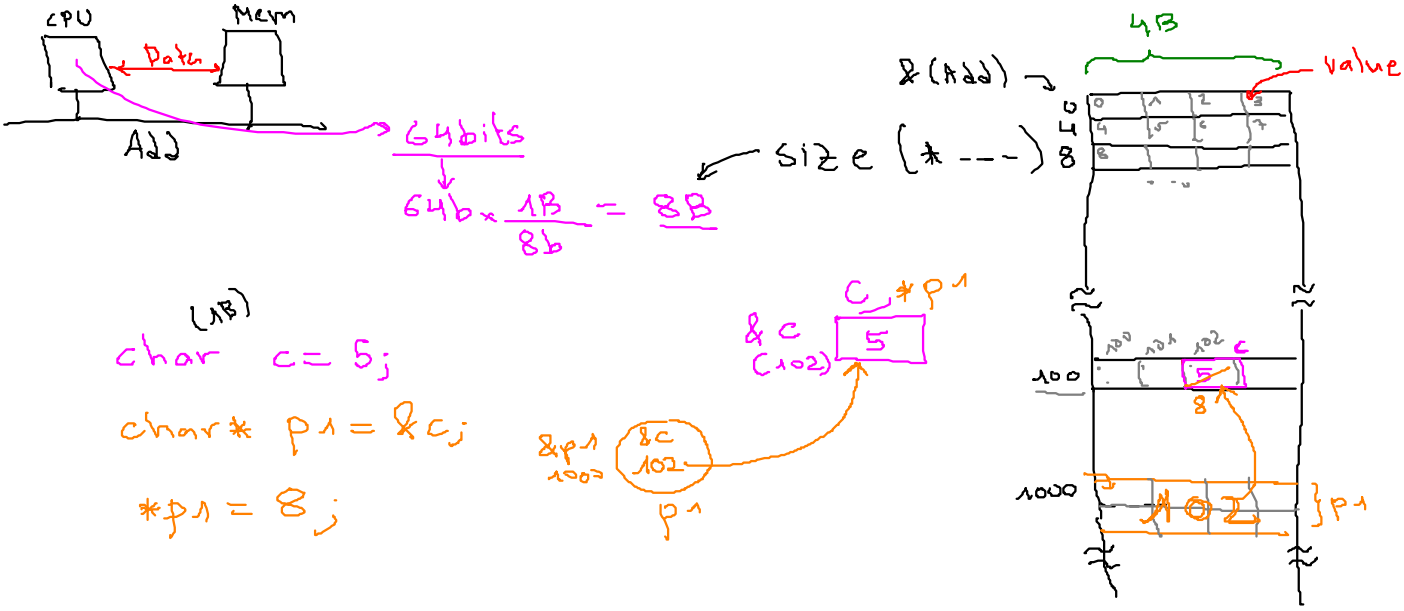
- Si **`start_routine`** requiere argumentos de diferente tipo, la declaración sería como se muestra a continuación:
  - Un ***argumento*** tipo entero (`int`)

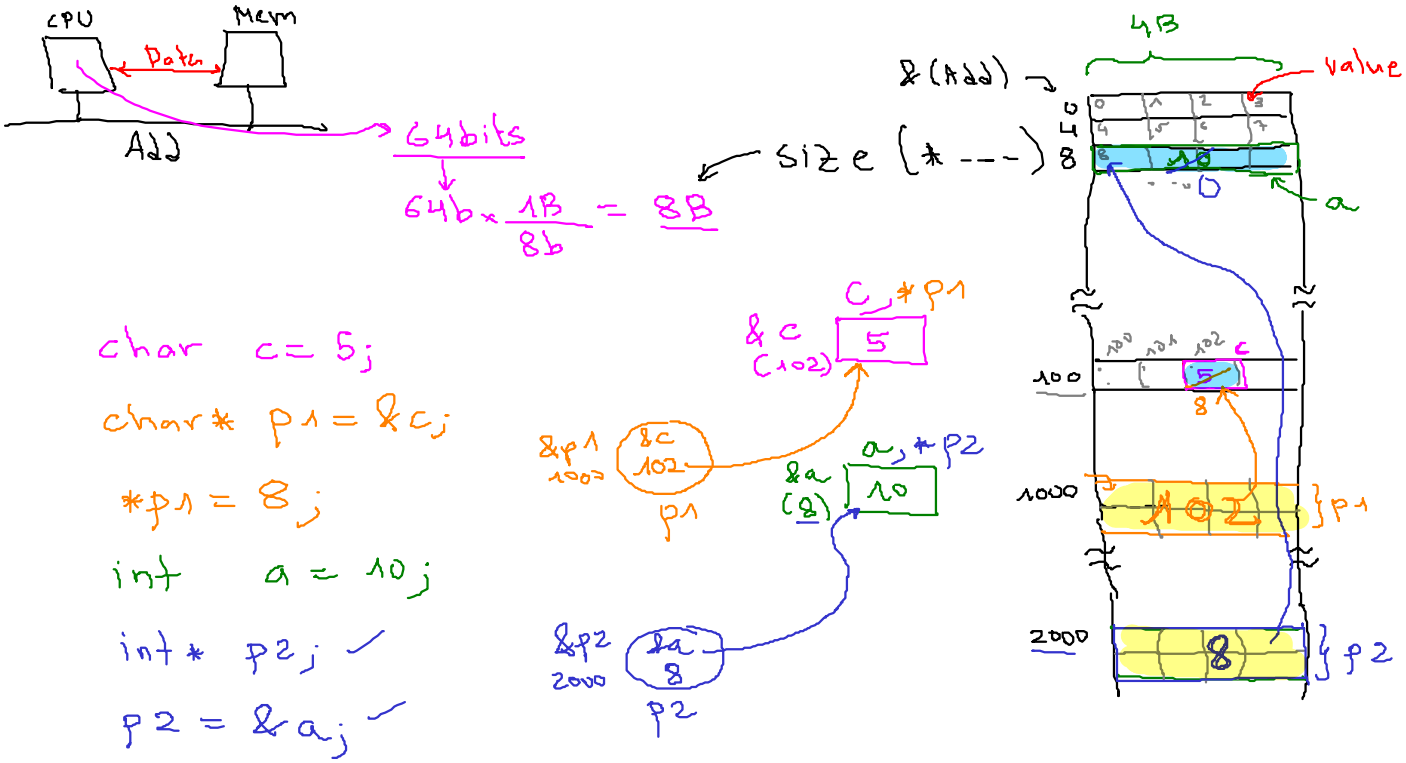
```
int pthread_create(..., // first two args are the same
                   void* (*start_routine)(int),
                   int arg);
```

  - Un ***retorno*** tipo entero (`int`)

```
int pthread_create(..., // first two args are the same
                   int (*start_routine)(void*),
                   void* arg);
```

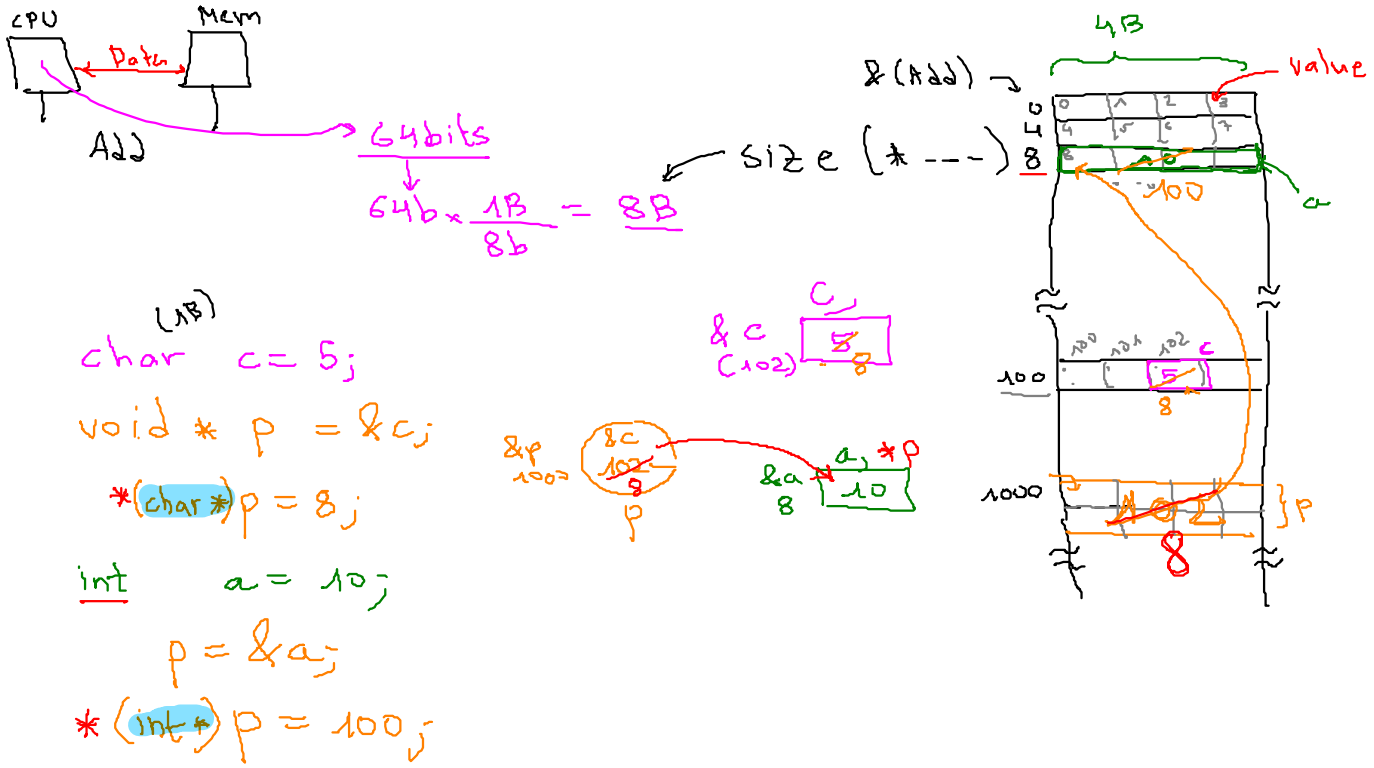
Importante: 1. Apuntadores a void.
2. Acceso a structs usando apuntadores





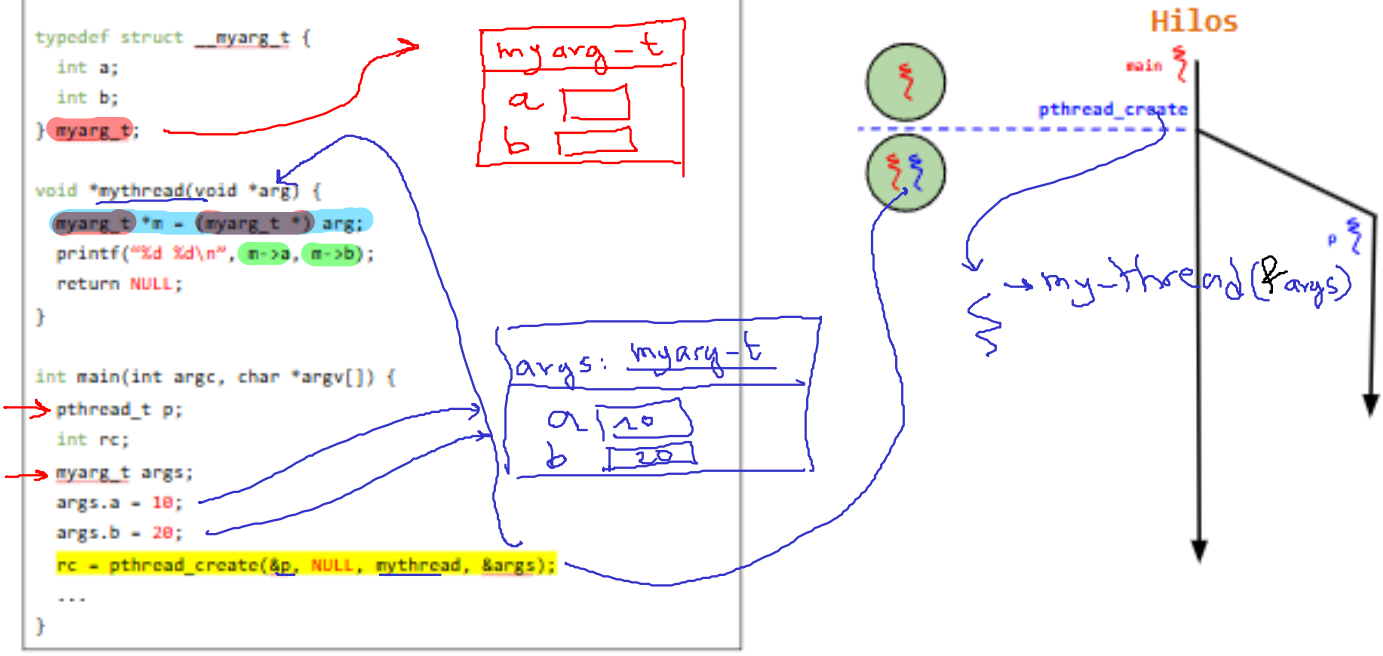

```
char c = 5;
char* p1 = &c;
*p1 = 8;
int a = 10;
int* p2;
p2 = &a;
*p2 = 0;
```

Apuntadores genericos (void*)




```
(1B)
char c = 5;
void * p = &c;
*(char*)p = 8;
int a = 10;
p = &a;
*(int*)p = 100;
```

```c
#include <pthread.h>

typedef struct __myarg_t {
  int a;
  int b;
} myarg_t;

void *mythread(void *arg) {
  myarg_t *m = (myarg_t *) arg;
  printf("%d %d\n", m->a, m->b);
  return NULL;
}

int main(int argc, char *argv[]) {
  pthread_t p;
  int rc;
  myarg_t args;
  args.a = 10;
  args.b = 20;
  rc = pthread_create(&p, NULL, mythread, &args);
  ...
}
```



¿Como crear hilos?

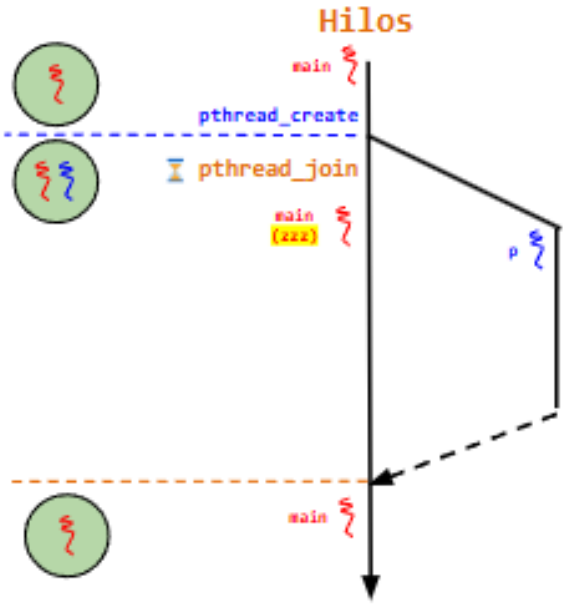
Para esperar hacer que un hilo espere la terminación de otro para acabar se emplea **`pthread_join`**:

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

Donde:

- **`thread`**: Especifica el hilo por el cual se va a esperar.
- **`value_ptr`**: un apuntador al valor de retorno.
  - Como **`pthread_join`** cambia (internamente) el valor de retorno, se debe ***pasar un apuntador*** a ese valor.

```
...
int main(int argc, char *argv[]) {
  int rc;
  pthread_t p;
  myret_t *m;

  myarg_t args;
  args.a = 10;
  args.b = 20;
  pthread_create(&p, NULL, mythread, &args);
  pthread_join(p, (void **) &m); // this thread has been
                                 // waiting inside of the
                                 // pthread_join() routine.
  printf("returned %d %d\n", m->x, m->y);
  return 0;
}
```

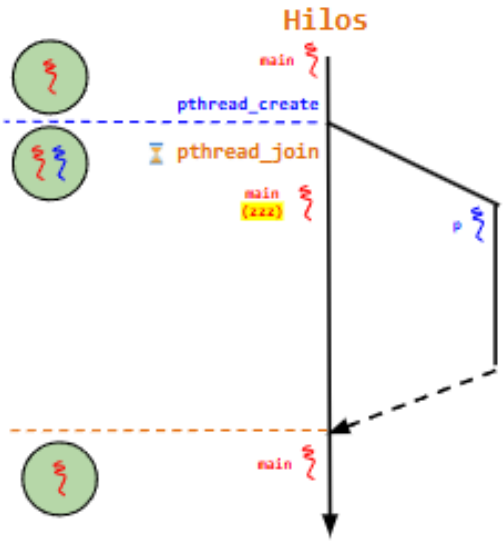


Ojo con esto:

Tenga cuidado con ***cómo se devuelven*** los valores desde un hilo.

```
void *mythread(void *arg) {
  myarg_t *m = (myarg_t *) arg;
  printf("%d %d\n", m->a, m->b);
  myret_t r; // ALLOCATED ON STACK: BAD!
  r.x = 1;
  r.y = 2;
  return (void *) &r;
}
```

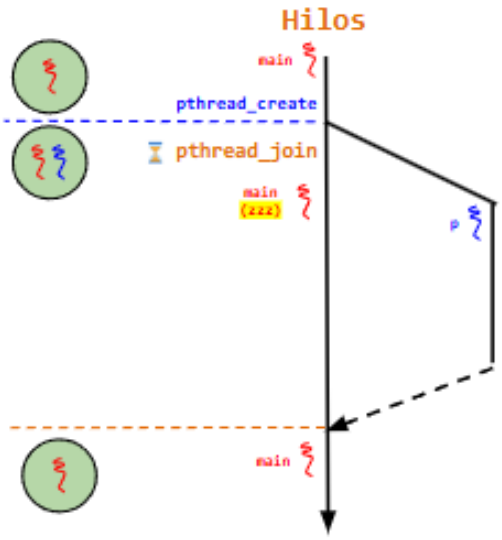
Cuando la variable **`r`** retorna, la memoria es **liberada automáticamente**.



Ejemplo - Pasando un valor simple a un hilo

Simplemente pasando un único valor.

```c
void *mythread(void *arg) {
  int m = (int) arg;
  printf("%d\n", m);
  return (void *) (arg + 1);
}

int main(int argc, char *argv[]) {
  pthread_t p;
  int rc, m;
  pthread_create(&p, NULL, mythread, (void *) 100);
  pthread_join(p, (void **) &m);
  printf("returned %d\n", m);
  return 0;
}
```



→ La parte de los locks se vera despues.

Recordatorio avances - Curso Red Hat

SISTEMAS OPERATIVOS Y L 2554485-1,2554842-1 (2025-2) • 6 respuestas

1. Cual es el porcentaje de avance del curso de System Administration de Red Hat (0-100) Respuesta 1 (Espacio para rellenar) *

(6/6) 100% respondido

Respuesta 1

48
38 100 0 55 15