Laboratorio 06

Estudiantes que realizaron el laboratorio:

Pablo Daniel Barillas Moreno. *Carné No.* 22193
André Emilio Pivaral López. *Carné No.* 23574

Instrucciones.

Esta actividad se realizará individualmente. Al finalizar los períodos de laboratorio o clase, deberá dejar constancia de sus avances en Canvas, según indique su catedrático. Al finalizar la actividad, adjuntar los archivos .pdf y .cpp para solucionar los ejercicios:

- Desarrolle el programa solución en C++.
- Incluir video corto con narración de funcionamiento del programa.

Ejercicio 01 (35 pts.)

Desarrolle un programa que utilice Pthreads de C++, que permita determinar los números de la serie Fibonacci que hay entre 0 y un número, el cual debe ser ingresado por el usuario. Indicar que el número debe de estar entre 0 y 100. El programa debe de evaluar cada uno de los números dentro del rango establecido.

Se debe calcular e imprimir el valor de la suma total de los números encontrados, en la rutina principal, debe implementar paso de parámetros por medio de estructuras (de ser necesario). La impresión debe ser amigable, para que pueda asociarse la iteración y el valor de la serie de Fibonacci obtenido.

La **serie de Fibonacci** es una secuencia de números en la cual cada número es la suma de los dos números anteriores. Comienza con 0 y 1, y a partir de ahí, cada número subsiguiente es la suma de los dos anteriores.

Ejemplo de los primeros números de la serie de Fibonacci:

1. F0=0
2. F1=1
3. F2=F1+F0=1+0=1
4. F3=F2+F1=1+1=2
5. F4=F3+F2=2+1=3
6. F5=F4+F3=3+2=5
7. F6=F5+F4=5+3=8
8. F7=F6+F5=8+5=13

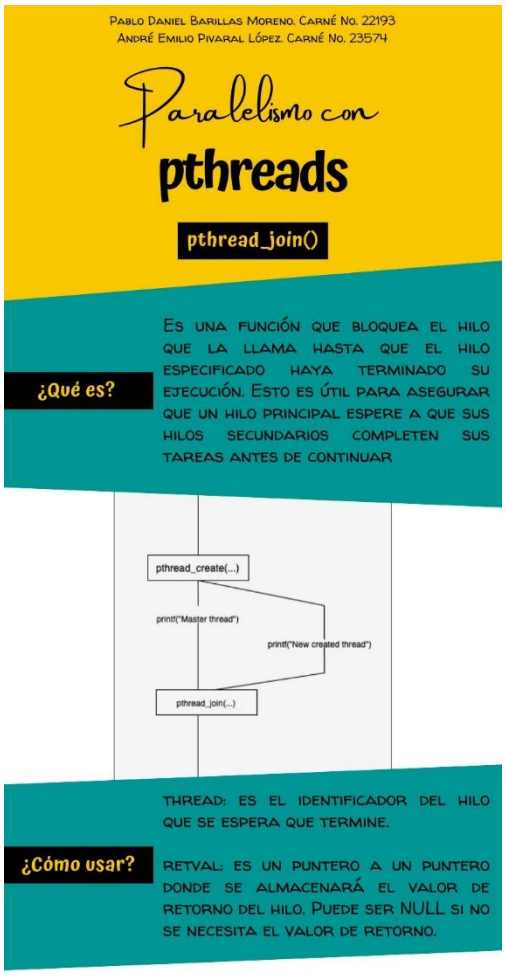
Y así sucesivamente.

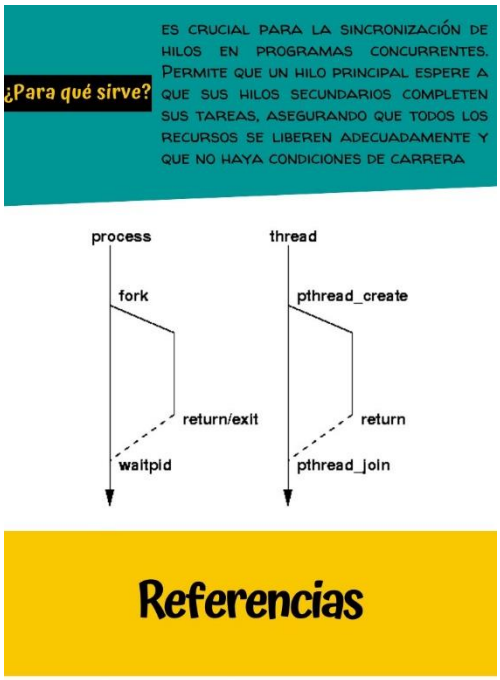
Ejercicio 02 (20 pts.)

a. Investigue qué es, cómo usar y para qué sirve pthread_join(). Deje evidencia de su investigación creando una infografía.

Enlace a la infografía:

https://www.canva.com/design/DAGPzf4NoVw/hn44kn8WO_uc_ahW8ps0cg/view?utm_content=DAGPzf4NoVw&utm_campaign=designshare&utm_medium=link&utm_source=editor





The Open Group. (2019). *pthread_join*. Threads. https://pubs.opengroup.org/onlinepubs/009695399/functions/pthread_join.html

Microsoft Learn. (2022). *Thread.Join*. Método (System.Threading). https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.join?view=net-8.0

The Open Group. (2018). *The Single UNIX Specification*. Version 4. https://pubs.opengroup.org/onlinepubs/9699919799/functions/pthread_join.html

GeeksforGeeks. (2023). Thread functions in C/C++. https://www.geeksforgeeks.org/pthread_join-in-c/

```c
#include <pthread.h>

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

Utilice el ejemplo de hello world con Pthreads y:

b. Modifique el programa para realizar un ciclo for para crear hilos y un ciclo for (separado) para hacer su respectivo join. Cada hilo debe de imprimir “Hello world thread No. X” colocando el número del hilo en lugar de la “X”.

c. Basándose en el mismo ejemplo, realice un ciclo for que cree el hilo y haga su respectivo join dentro del mismo ciclo. Cada hilo debe de imprimir “Hello world thread No. X” colocando el número del hilo en lugar de la “X”.

Responder las siguientes preguntas:

1. **¿Cuál fue la diferencia entre la impresión del primer programa y el segundo?**

R// En el primer programa (ciclo separado para crear hilos y hacer join), todos los hilos se crearon casi simultáneamente en el primer ciclo, y luego el programa esperó a que cada uno de ellos terminara en el segundo ciclo con la función pthread_join. Esto llevó a que los mensajes de los hilos ("Hello world thread No. X!") se imprimieran de manera no determinística o concurrente. Dado que todos los hilos estaban ejecutándose al mismo tiempo, el sistema operativo decidió cuándo asignar tiempo de CPU a cada uno, lo que significa que los mensajes pudieron haberse mostrado en un orden aleatorio dependiendo de cómo se programó cada hilo para ejecutarse. El orden de creación de los hilos no garantizaba el orden de ejecución o impresión, lo que resultó en que los mensajes no se imprimieran necesariamente en secuencia del hilo 0 al hilo 9.

En cambio, en el segundo programa (creación y join dentro del mismo ciclo), cada hilo fue creado y el programa esperó a que ese hilo específico terminara su ejecución antes de crear el siguiente hilo. Esto significaba que los mensajes de los hilos se imprimieron de manera **secuencial** y **determinística**. Cada hilo imprimió su mensaje antes de que el siguiente hilo fuera creado, lo que resultó en una secuencia de impresiones completamente predecible, desde "Hello world thread No. 0!" hasta "Hello world thread No. 9!". Dado que el sistema esperó a que cada hilo terminara antes de continuar, no hubo ejecución concurrente y los mensajes se imprimieron en el orden exacto de creación.

2. ¿A qué se debió el comportamiento descrito en la respuesta anterior?

R// El comportamiento en ambos programas se debió a la forma en que se manejaron los hilos y el uso de la función pthread_join para sincronizar su ejecución.

En el **primer programa**, los hilos se crearon en paralelo, es decir, todos fueron lanzados casi al mismo tiempo dentro del ciclo for. En este caso, no se hizo pthread_join inmediatamente después de crear cada hilo, sino que el programa permitió que todos los hilos comenzaran su ejecución antes de esperar a que todos terminaran en un ciclo separado. Este enfoque introdujo **concurrencia**, lo que significa que varios hilos estaban ejecutándose al mismo tiempo. Como el sistema operativo administra la ejecución de los hilos, es quien decide cuál de ellos recibe tiempo de CPU en cada momento, lo que introduce una **naturaleza no determinística** en la forma en que los mensajes se imprimen. El sistema operativo puede cambiar entre hilos en cualquier momento, lo que resulta en un **orden aleatorio** de los mensajes "Hello world thread No. X!" según cómo los hilos se programen para ejecutarse.

En contraste, en el **segundo programa**, cada hilo se creó y el programa utilizó pthread_join inmediatamente después de crear el hilo, lo que obligó al sistema a **esperar** a que ese hilo específico terminara antes de crear el siguiente. Este enfoque evita la concurrencia, ya que el sistema no permite que más de un hilo esté ejecutándose al mismo tiempo. Una vez que un hilo ha completado su tarea y ha impreso su mensaje, el programa crea el siguiente hilo. Esto produce un comportamiento completamente **determinístico** y **secuencial**, donde el orden de creación y ejecución de los hilos es idéntico al orden de impresión de los mensajes. En este caso, no hay sobreposición entre la ejecución de los hilos, lo que garantiza que cada hilo imprima su mensaje antes de que el siguiente hilo sea siquiera creado.

Ejercicio 03 (35 pts.)

Desarrolle un programa que utilice Pthreads de C++, que realice el cálculo de valor de convergencia para la siguiente serie geométrica:

a. $$\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$$

Condiciones:

- El valor máximo de “n” a evaluar en la serie debe ser ingresado por teclado.
- Se utilizará 1 pthread para evaluar cada valor de n en la serie. Es decir, la cantidad de hilos utilizados para evaluar la serie debe ser igual a n.
- La sumatoria e impresión del resultado se debe efectuar en la rutina principal. Debe implementar paso de parámetros por medio de pthread_join.

Enlace al repositorio de GitHub del laboratorio 6B

https://github.com/DanielBarillasM/Laboratorio-06_Daniel-Barillas_Microprocesadores.git