Universidad Nacional de San Antonio Abad del Cusco
Departamento Académico de Informática
ALGORITMOS PARALELOS Y DISTRIBUIDOS
Práctica Nº 1

Programación Multihilo en C++

1. **OBJETIVO.**
    - Comprender el concepto de hilo
    - Aprender a crear y manejar hilos en C++.
    - Entender el ciclo de vida de un hilo.
2. **BASE TEORICA COMPLEMENTARIA**
    - **MULTIPROCESAMIENTO**

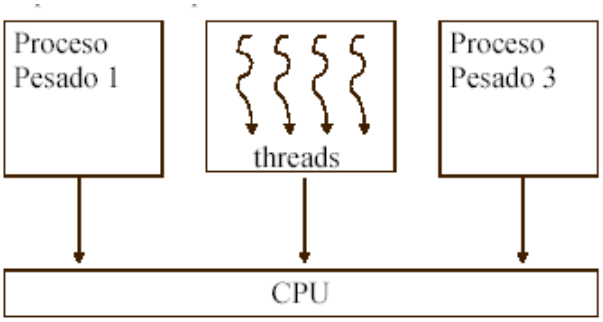
      El multiprocesamiento o multihilo o por su nombre en inglés multithreading, es la habilidad de un sistema computacional de ejecutar más de un hilo de ejecución a la vez. En una computadora moderna, el procesador no ejecuta los procesos de forma secuencial sino que cambia de contexto (de proceso o hilo) ejecuta unas pocas instrucciones, y vuelve a cambiar de contexto repitiendo ese ciclo de manera infinita.

      Esto nos da la sensación de que la computadora está ejecutando muchos procesos en paralelo pero eso es aparente, en realidad, el procesador solo puede ejecutar un proceso a la vez. En el caso de procesadores con varios núcleos, el procesador puede ejecutar tantos procesos o hilos en paralelo como núcleos disponga y cada núcleo trabaja de la misma forma.

    - **CONCEPTO DE HILO (Thread).**

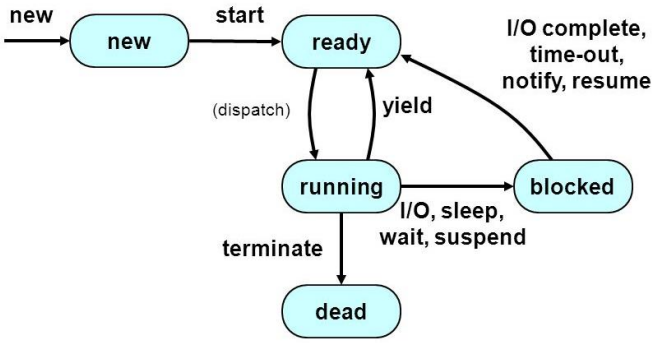
      Un *proceso* es un programa en ejecución que es localizado en la memoria por el sistema operativo. Un *hilo* es una ejecución o flujo de control en el espacio de direcciones de un proceso. Un proceso puede tener más de un hilo. Todos los hilos en un proceso tienen su propio contador de programa y su propia pila para variables locales y direcciones de retorno de invocación de procedimientos.

      En la figura observamos cómo sobre una CPU pueden estar ejecutándose distintos procesos pesados y uno de ellos puede estar compuesto por distintos flujos de ejecución (threads o hebras o hilos). Estos threads tendrán que "disputarse" el tiempo de ejecución que el S.O. le dé al proceso en el que residen.



- **CICLO DE VIDA DE UN HILO**

  Los hilos cumplen el siguiente ciclo de vida:



- **LA CLASE Thread en C++**

std::thread

Definido en la cabecera <thread>	
class thread;	(desde C++11)

La clase thread representa a un hilo en ejecución. Los hilos permiten que múltiples fragmentos de código se ejecuten de forma asíncrona y simultánea.

Tipos de miembros

Miembro de tipo	Definition
native_handle_type	' **Definido por la implantación**

Clases de miembros

id	representa el identificador' **de un hilo (miembro de clase público)**

Las funciones miembro

(constructor)	construye un nuevo thread (función miembro público)
(destructor)	destruye el objeto Thread, se debe haber ejecutado join() o detach previamente (función miembro público)
operator=	mueve el objeto thread a un nuevo identificador (función miembro público)
Los observadores	
joinable	comprueba si el objeto representa a un hilo actualmente en ejecución o a la espera de recibir un join() (función miembro público)
get_id	devuelve el *id* del hilo (función miembro público)
native_handle	devuelve el handle del sistema operativo relativo al hilo (función miembro público)
hardware_concurrency [estático]	devuelve el número de hilos simultáneos soportados por la implementación (función miembro público estático)
Operaciones	
join	espera a que el hilo termine su ejecución (función miembro público)
detach	desvincula el objeto del hilo en ejecución, ya no se puede hacer join y se puede destruir libremente (función miembro público)
swap	swaps dos objetos hilo

- **IMPLEMENTACIÓN DE HILOS EN C++**

```
void func()
{ //--función que ejecutará el hilo
  ..
  ..
  ..
  ..
}

int main()
{
      //-- Crea y ejecuta el hilo
      std::thread hilo1(func);
```

Función que ejecutará el hilo

Cuerpo del Hilo

Creación y planificación del hilo que ejecuta la función "func"

```
}
```

3. **DESARROLLO DE LA PRÁCTICA.**

Ejemplo 1.

```
#include <iostream>
#include <thread>
using namespace std;

void escribeMensaje()
{ //--funcion que ejecutará el hilo
        std::cout << "Hola mundo..." << '\n';
}

int main()
{
   //-- Crea y planifica el hilo para ejecutar escribeMensaje
   std::thread hilo1(escribeMensaje);

   getchar();
   return 0;
}
```

Ejemplo 2.

En este ejemplo se crea dos hilos y se lanzan a ejecución. Ambos hilos imprimen una palabra pero con distintos intervalos de tiempo. La palabra a imprimir y el intervalo de tiempo que duermen son parámetros del constructor de la clase. En este ejemplo se puede apreciar cómo se entremezcla la ejecución de ambos hilos de ejecución.

```
#include <iostream>
#include <thread>
#include <string>
using namespace std;

void imprime(string palabra, long tiempo)
{
   for (int i = 1; i <= 100; i++)
   {
        std::cout << palabra << '\n';
        _sleep(tiempo); //--duerme en milisegundos
   }
}

int main()
{
   // Crea y ejecuta hilo1 e hilo2
   std::thread hilo1(imprime, "tin", 10);
```

```
    std::thread hilo2(imprime, "ton", 5);

   getchar();
   return 0;
}
```

Ejemplo 3.
En el ejemplo anterior agregamos una instrucción que imprime un mensaje cuando los hilos terminaron.
¿Corresponde la ejecución del programa a ésta lógica?

```
#include <iostream>
#include <thread>
#include <string>
using namespace std;

void imprime(string palabra, long tiempo)
{
   for (int i = 1; i <= 100; i++)
   {
         std::cout << palabra << '\n';
         _sleep(tiempo); //--duerme en milisegundos
   }
}

int main()
{
   // Crea y ejecuta hilo1 e hilo2
   std::thread hilo1(imprime, "tin", 10);
   std::thread hilo2(imprime, "ton", 5);

   std::cout << "termino la ejecución de los hilos.." << '\n';
   getchar();
   return 0;
}
```

Agregar ésta instrucción para saber que terminaron los hilos

¿A qué se debe el resultado mostrado?

Ejemplo 4.
Ahora observe que ocurre cuando se ejecuta el siguiente ejemplo.

```
#include <iostream>
#include <thread>
```

```
#include <string>
using namespace std;

void imprime(string palabra, long tiempo)
{
   for (int i = 1; i <= 100; i++)
   {
         std::cout << palabra << '\n';
         _sleep(tiempo); //--duerme en milisegundos
   }
}

int main()
{
   // Crea y ejecuta hilo1 e hilo2
   std::thread hilo1(imprime, "tin", 10);
   std::thread hilo2(imprime, "ton", 5);


   hilo1.join();
   hilo2.join();

  std::cout << "termino la ejecución de los hilos.." << '\n';
  getchar();
  return 0;
}
```

Ejemplo 5.
En el siguiente ejemplo, el cuerpo del hilo imprime el contenido de un vector de enteros que se le pase como parámetro.

```
#include<iostream>
#include<vector>
#include<thread>

using namespace std;

void f(std::vector<int>& v);

int main()
{
   int s1;
   std::vector<int> pares = { 10, 20, 40, 60, 80 };
   std::vector<int> impares = { 1, 3, 5, 7, 9 };

   /* Lanza dos threads t1 y t2 */
   std::thread t1(f, pares);
   std::thread t2(f, impares);
```

```
   /* Espera que finalicen */
   t1.join();
   t2.join();

   getchar();
}

void f(std::vector<int>& v)
{
   for (std::vector<int>::iterator it = v.begin();
        it != v.end(); ++it)
      {
        std::cout << *it << std::endl;
       }
}
```

4. TAREA

Escribir un programa para que el cuerpo del hilo calcule la suma de los elementos de un vector de enteros el cual es pasado como parámetro. Cada hilo en ejecución devuelve la suma del vector al programa principal el cuál calcula la suma total y muestra el resultado.

5. BIBLIOGRAFIA

- Tanenbaum Andrew, 2009, “SISTEMAS OPERATIVOS MODERNOS” 3ra. Edición. Pearson. México.

- Herrera Josefina, 2011,”PROGRAMACIÓN EN TIEMPO REAL Y BASES DE DATOS . UN ENFOQUE PRÁCTICO”. Oficina de publicaciones digitales UPC. Barcelona

- https://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

- http://code0x66.blogspot.com/2017/01/c-threads-hilos.html

Cusco octubre de 2019.