UNIVERSIDAD EMPRESARIAL SIGLO 21



CARRERA: LIC. EN INFORMATICA Trabajo Práctico número 1

Materia: Algoritmos concurrentes y paralelos

Profesor: PIRAY, EDUARDO ENRIQUE

Modulo: 1

Alumno: Soler, Diego Francisco.

Fecha: 10/06/2024

Objetivos:

- Crear y manipular programas concurrentes en C++.
- Implementar programas concurrentes que requieran cierta sincronización.
- Adquirir conocimiento empírico del funcionamiento de threads en C++.

Consigna 1

Basado en la situación problemática, se debe analizar el comportamiento de los algoritmos presentados, estudiando sus particularidades desde el punto de vista de la implementación. Para llevar a cabo esta tarea es necesario ejecutar los algoritmos y luego de realizar diversas ejecuciones, documentar y explicar el comportamiento de tales programas.

Consigna 2

Con base en el enfoque presentado en el algoritmo 2, desarrollar una nueva versión del programa en el que se ejecuten 15 procesos, de manera que el proceso i-ésimo se pause un tiempo aleatorio (definir un rango en milisegundos que pueda ser perceptible), y que por pantalla se muestre el mensaje correspondiente ("Soy el proceso 1", "Soy el proceso 2", ..., "Soy el proceso n") un número aleatorio de veces.

Consigna 3

Tal como se puede observar, los algoritmos anteriores son claros cuando el comportamiento del proceso puede estar encapsulado en una única función. Pero termina siendo confuso cuando los procesos necesitan llevar a cabo tareas de mayor complejidad. Para ello, un enfoque más general consiste en encapsular (en una clase de objetos) el comportamiento de un proceso, creando las instancias necesarias, y de este modo lanzar la función deseada del objeto en un *thread*. Analice de qué manera podría implementar este enfoque, describa las características y esboce una implementación del algoritmo.

Consigna 1

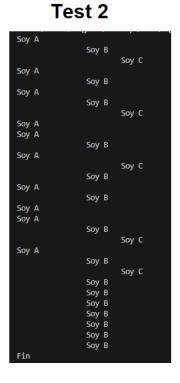
Algoritmo 1:

Análisis general:

El algoritmo uno lo que hace es crear 3 hilos, cada uno con un mensaje, delay y cantidad de repeticiones distintos, cada uno de estos hilos ejecutan la función misaludo de manera independiente respetando sus respectivos delays. Estos hilos están unidos mediante la función join, creando un hilo principal permitiendo asegurar que los 3 hilos terminen de ejecutarse antes de hacer un print del string "Fin".

El algoritmo en ejecución:

Ahora procedemos a ejecutar el algoritmo 6 veces para testear su funcionamiento:





Test 6 Test 5 Test 4 Soy B Soy A Soy A Soy B Soy C Soy C Soy B Soy A Soy A Soy A Soy B Soy B Soy B Soy A Soy A Soy A Soy C Soy B Soy B Soy B Soy C Soy A Soy A Soy A Soy A Soy A Soy B Soy B Soy B Soy A Soy A Soy A Soy C Soy B Soy C Soy B Soy B Soy A Soy A Soy A Soy A Soy A Soy A Soy B Soy B Soy B Soy A Soy A Soy A Soy C Soy B Soy B Soy B Soy C Soy A Soy A Soy A Soy B Soy B Soy B Soy C Soy C Soy B Soy B Soy C Soy B

Después de realizados los test podemos observar discrepancias en la ejecución del código, en cada uno de los test tiene un inicio distinto al anterior en los primeros 3 prints del programa. Este comportamiento es normal en los programas multihilo sin sincronización y se debe a que el sistema operativo es el encargado de asignar recursos para el procesamiento de cada hilo, dado que no estamos utilizando ningún método de sincronización el sistema operativo schedulea y ejecuta en cualquier orden cada uno de los hilos. En otras palabras, el sistema operative decide cuando y como se ejecutan los hilos en cada ejecución del programa y, por lo tanto, puede hacer variar el resultado en cada ejecución.

Algoritmo 2:

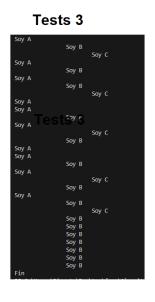
La diferencia del algoritmo 2 con respecto del algoritmo 1 es en la forma en la que se crean y se manejan los hilos. En este caso, utilizamos un array para almacenar cada uno de los hilos en lugar de realizar declaraciones individuales de los mismos. En cuestiones practicas ambos algoritmos trabajan de la misma forma, pero el algoritmo 2 al utilizar un array permite que el programa este mas organizado y sea escalable.

De igual forma, este algoritmo no cuenta con un método de sincronización, por lo tanto, tiene el mismo comportamiento que el algoritmo 1 dado que el sistema operativo sigue siguiendo el encargado de asignar y ejecutar los hilos. Esto se observa en las siguientes imágenes donde se realizaron 6 test.

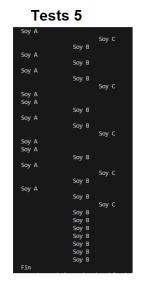
Tests 1

Soy B
Soy A
Soy B
Soy C
Soy A
Soy B
Soy C
Soy B
Soy B
Soy C
Soy B





Tests 4		
, ,	Soy B	1
Soy A		
		Soy C
Soy A	C D	
Soy A	Soy B	
Joy A	Soy B	
	20, 2	Soy C
Soy A		
Soy A		
	Soy B	
Soy A		
	Soy B	
Cou. A		Soy C
Soy A Soy A		
JUJ A	Soy B	
Soy A	20, 5	
,	Soy B	
		Soy C
Soy A		
	Soy B	
	Soy B	
	C D	Soy C
	Soy B	
	Soy B Soy B	
	Soy B	
	Soy B	
	Soy B	
Fin		





Consigna 2

Para esta consigna vamos a modificar el código del algoritmo 2 para cumplir con la consigna, sumado a que le vamos a permitir al usuario que elija que proceso desea pausar, este proceso será pausado por única vez por 600 milisegundos y luego volverá a su funcionamiento normal. Además, creamos la función generalAleatorio para generar números aleatorios en base a un rango dado, modificamos la función misaludo para que cumpla con la pausa mencionada y agregamos distintos loops para poder automatizar el proceso.

Todo el código se puede observar desde el archivo consigan2.cpp

Consigna 3

Ahora para la consigna 3 vamos a modificar el código de la consigna 2 pero utilizando programación orientada a objetos. En nuestro código vamos a crear la clase proceso, con los atributos id y procesoPausar (determina si es el proceso que se debe apusar). Esta clase tiene los métodos:

- generarAleatorio: Que genera un numero aleatorio dentro de n rango dado
- <u>saludo</u>: crea una variable que llama al método generarAleatorio, contiene un loop para imprimir los mensajes y el if que determina si se trata del hilo a ser pausado y si ya fue pausado una vez.

Por último, tenemos la función main, que crea una instancia de la clase proceso para cada hilo, hace el join para esperar a que todos los hilos terminen y por último imprime el string "Fin".

Todo el código se puede observar desde el archivo consigan3.cpp

Dificultades que me ocurrieron:

- En la consigna 2 y 3, cometí el error de querer usar el mismo loop para lanzar los hilos y hacer el join, generando que no ocurra la concurrencia dado que no se lanzaba otro hilo hasta que no terminara el anterior.
 Esto lo solucione separando los loops.
- Agregue un validador para el cin para que el usuario solo pueda ingresar un int del 1 al 15.