Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ciencias de la Computación e Informática

CI-0117 Programación Paralela y Concurrente
Grupo 01
I Semestre

I Tarea programada: Collatz Ascensor

Profesor:
Francisco Arroyo

Estudiantes:
Rodrigo Vílchez Ulloa | B78292

15 de mayo del 2020

Índice

1. Introducción

El tema de la tarea programada trata acerca del paralelismo y de cómo hacer uso de los procesos ejecutados de forma paralela en el procesador para ejecutar tareas más eficientes, en contraste con los programas seriales que no aprovechan la ventaja ofrecida por la capacidad de los núcleos del procesador para distribuir el trabajo, de la misma forma que, al implementar procesos en paralelo, se debe hacer un buen uso de la memoria compartida por estos, pues puede terminar causando un mal cálculo, una sobreescritura o cualquier manejo erróneo de los datos.

2. Objetivos

Collatz:

- Haga un programa en C o C++ que reciba un número como parámetro, el programa debe determinar cuáles la mayor cantidad de pasos que Collatz realiza sobre todos los elementos y en cuál elemento del conjunto ocurrieron.
- El programa debe indicar el tiempo que tarda en realizar las operaciones solicitadas, la cantidad de pasos y el número que los generó.

Ascensor:

- La sincronización, concurrencia y eficiencia en el programa.
- Crear al menos una clase para el .Ascensorz otra para las "Personas".

3. Descripción

Collatz:

Se debe crear un programa que debe implementar paralelamente la conjetura de Collatz para un intervalo de números determinado por el usuario. Esta conjentura indica que, mediante dos operaciones aritméticas, se puede llegar al número 1 sin importar cuál sea el número inicial. Se realiza la operación $n = n/2$ en caso de que el número sea par, y $n = 3 \cdot n + 1$ si es impar. Estas operaciones se aplican sucesivamente a partir de n hasta llegar al número 1. El programa debe indicar cuál fue el número al que le tomó más operaciones llegar al número 1, cuántos fueron los pasos totales y el tiempo en resolver el problema.

Ascensor:

Crear un programa que simule un ascensor, este tiene una capacidad máxima y ciertas reglas para que las personas suban y bajen de él. Cada persona debe llamar al ascensor para indicarle en cuál piso se encuentra y a cuál piso quiere ir. Tanto el ascensor como las personas se deben programar como si fueran procesos distintos, por lo que paralelamente el ascensor va a estar operando, mientras nuevas personas solicitan el servicio. El ascensor debe satisfacer todas las llamadas al finalizar el programa.

4. Diseño

Collatz:

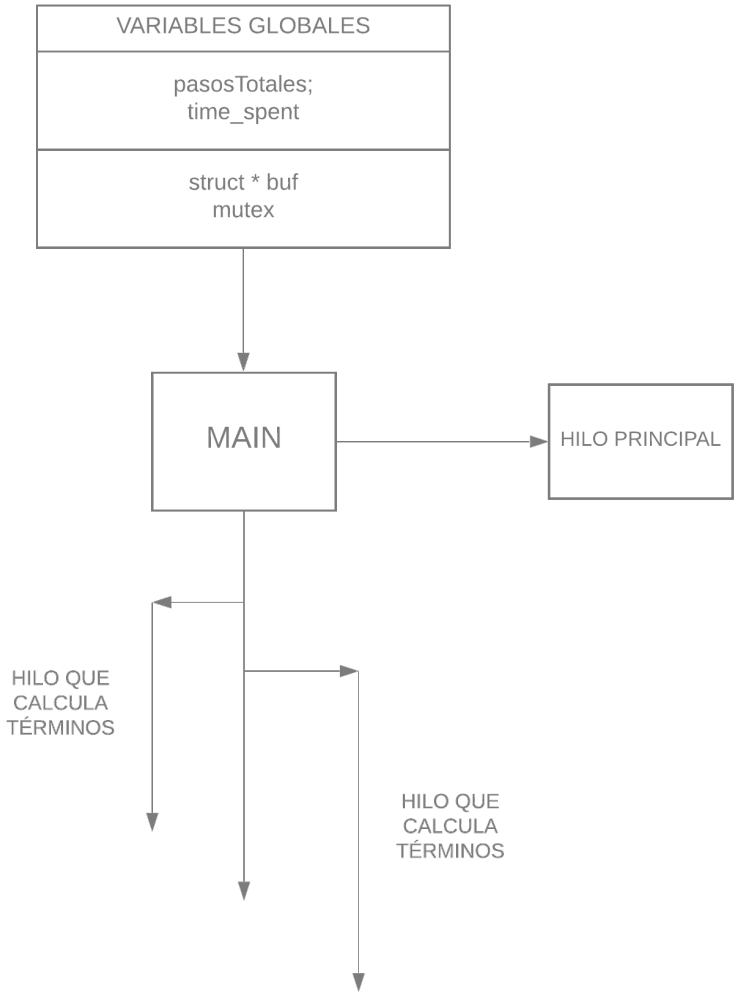

Figura 1: Diseño del programa Collatz.

Ascensor:

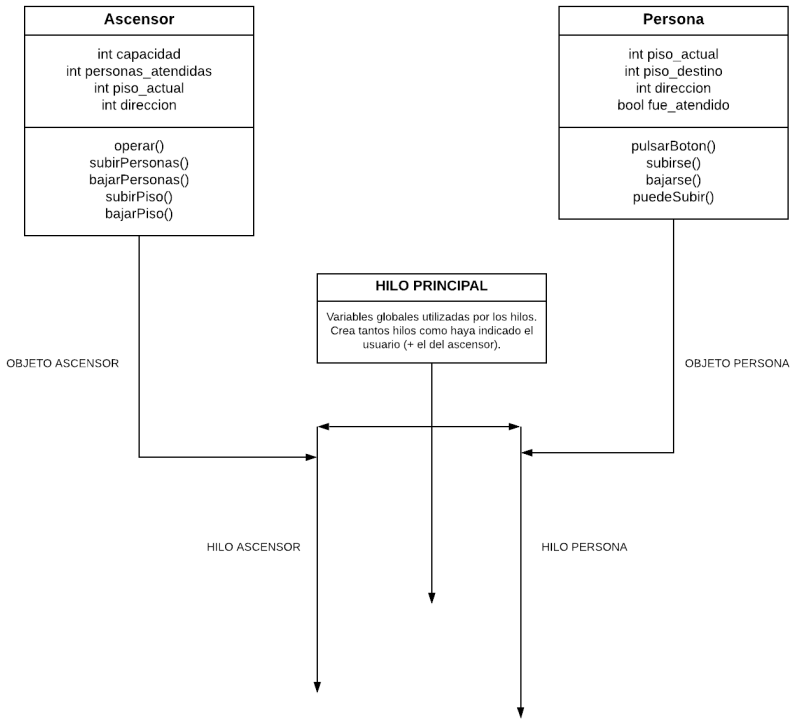

Figura 2: Diseño del programa Ascensor.

5. Desarrollo

Collatz

Para este problema se utilizaron hilos. El usuario debe establecer el intervalo de números que desea calcular, este irá de $[1, n]$, y también el número de hilos. Una vez que se tengan estos datos, el programa divide ese intervalo en intervalos más pequeños y le envía uno distinto a cada hilo para dividir el trabajo. Cada hilo recibe un struct que contiene el límite inferior y superior de su intervalo a calcular, así como un ID único para cada hilo, con esta información procede a ejecutar un método que, para cada número en ese intervalo, cuenta los pasos que le toma a cada uno llegar a 1. Cada hilo cuenta con dos variables locales, una que guarda el número que más pasos ha realizado hasta el momento y otra que guarda ese número de pasos, así que, en caso de que un número realice más pasos, estas variables se irán actualizando. Al final, el hilo envía un struct que contiene cuál fue su número que más pasos tardó así como la cantidad, este struct se guarda en un array que ha sido declarado como variable global para que fuera accesible por cada hilo, sin provocar problemas pues las posiciones del array están determinadas por el ID que le corresponde a cada hilo. De igual forma, los pasos de cada número se van sumando a otra variable global que acumula los pasos totales en todos los hilos, para esto se utilizó un mutex, que permite el acceso a un solo hilo cuando este vaya a sumar, a la variable global, los pasos totales de todos los números que tuvo que calcular . Una vez que todos los hilos hayan terminado y enviado su struct al array, el hilo principal recorrerá ese array buscando, entre todos los hilos, cuál fue el número que más pasos tomó hacer las operaciones. Algunas consideraciones son: el programa supone que el usuario ingresa un número que sea potencia de 10; para distribuir bien el intervalo, el programa verifica que la cantidad de hilos sea la adecuada para hacer una distribución pareja de los intervalos entre los hilos.

Ascensor

El problema fue resuelto utilizando hilos. Existe un hilo principal, que es el main, que es el encargado de asignarle espacio a cada hilo que se vaya a crear, así como de crearlos y finalizarlos (join). De manera conjunta, se crearon dos clases llamadas Ascensor y Persona. El hilo principal creará un segundo hilo que será el que le corresponde al objeto Ascensor, el cual es solo uno para todo el programa, y posteriormente creará todos los hilos que correspondan a las personas (un hilo por persona), la cantidad de personas será definida por el usuario al correr el programa desde la terminal, por lo que existen dos funciones que son la que ejecutan los hilos: la que crea un objeto ascensor y la que crea un objeto persona. El hilo encargado del ascensor ejecutará una función llamada *operar*, esta función consta de un ciclo que va a seguir ejecutándose hasta que todas las personas hayan sido atendidas por el ascensor, pero que igualmente manda a dormir al hilo si no existe una solicitud pendiente, para no gastar recursos. Dentro de la clase Ascensor, se implementaron otros métodos tales como subir de piso, bajar de piso, subir personas, bajar personas, entre otros. Dentro de la clase Persona, cuando un objeto es creado, este esperará una cantidad de segundos aleatoria para hacer la solicitud al ascensor, esto para que el ascensor no reciba todas las solicitudes al mismo momento, sino que opere recibiendo solicitudes en diferentes lapsos de tiempo, pudiendo recibir varias o pocas al mismo tiempo. Igualmente, cuando una persona es creada, se le asigna un piso aleatoriamente, así como un piso destino. Como fue especificado en el enunciado, el ascensor solo permite subir personas que vayan en la dirección del ascensor, es decir, que si el ascensor va bajando, solo podrán subir personas que también deseen bajar siempre que haya espacio en el elevador para subir, esta capacidad máxima es de ocho personas. Cuando el ascensor está vacío y no hay solicitudes pendientes, este esperará a que haya una y no gastará recursos. Existen dos variables compartidas por los hilos, las cuales son una cola de solicitudes y un array que indica cuántas personas están esperando por piso. Cada vez que un objeto/hilo vaya a modificar o consultar estas variables, deberá esperar que no haya ningún otro proceso leyendo o modificando esas variables, esto se implementó por medio de pthread-mutex, uno para cada variable. De igual forma, se implementaron semáforos para que el ascensor se despierte cuando este pase a tener solicitudes y cuando baje o suba personas, esto para que el ascensor no opere antes de que, paralelamente, un hilo persona aún no se haya bajado o subido.

6. Manual de usuario

Requerimientos de Software

- **Sistema Operativo:** Linux
- **Arquitectura:** 32/64 bits
- **Ambiente:** Terminal

Compilación

- **Collatz**
  Para compilar el programa, se utiliza `gcc` en la siguiente sentencia:

```
$ gcc  -o collatz collatz.c -lpthread $
```

  O simplemente se utiliza el makefile:

```
$ make $
```

- **Ascensor**
  Utilizando el comando make:

```
$ make $
```

Especificación de las funciones del programa

En el programa Collatz, para ejecutarlo es necesario indicar desde terminal primero el número hasta el cual se va a calcular, y segundo la cantidad de hilos.

```
$ ./collatz LIMITE_NUMERO CANTIDAD_HILOS $
```

En el programa Ascensor, se debe indicar, mediante la línea de comandos, la cantidad de personas máxima que se van a atender.

```
$ ./ascensor CANTIDAD_PERSONAS $
```

7. Casos de Prueba

Collatz

Prueba 1:

Prueba con 10^5 números y cuatro hilos.

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Tareas programa
atz$ ./collatz 100000 4
HILO #0          RANGO DE NUMEROS: [1, 25000]
HILO #1          RANGO DE NUMEROS: [25001, 50000]
HILO #3          RANGO DE NUMEROS: [75001, 100000]
HILO #2          RANGO DE NUMEROS: [50001, 75000]

Numero con mas pasos: 77031
Cantidad de pasos: 350
Tiempo entre todos los procesos: 0.290179
Tiempo real de ejecución: 0
Total de pasos: 10753840
```

Figura 3: Prueba con 10^5 términos y 4 hilos.

Prueba 2:

Prueba con 10^7 números y dos hilos.

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Tareas programada
atz$ ./collatz 10000000 2
HILO #0          RANGO DE NUMEROS: [1, 5000000]
HILO #1          RANGO DE NUMEROS: [5000001, 10000000]

Numero con mas pasos: 8400511
Cantidad de pasos: 685
Tiempo entre todos los procesos: 13.578358
Tiempo real de ejecución: 4
Total de pasos: 1552724831
```

Figura 4: Prueba con 10^7 términos y 2 hilos.

Prueba 3:

Prueba con 10^6 números y ocho hilos.

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Tareas programadas/Coll
atz$ ./collatz 1000000 8
HILO #0          RANGO DE NUMEROS: [1, 125000]
HILO #1          RANGO DE NUMEROS: [125001, 250000]
HILO #2          RANGO DE NUMEROS: [250001, 375000]
HILO #3          RANGO DE NUMEROS: [375001, 500000]
HILO #4          RANGO DE NUMEROS: [500001, 625000]
HILO #5          RANGO DE NUMEROS: [625001, 750000]
HILO #6          RANGO DE NUMEROS: [750001, 875000]
HILO #7          RANGO DE NUMEROS: [875001, 1000000]

Numero con mas pasos: 837799
Cantidad de pasos: 524
Tiempo entre todos los procesos: 6.380778
Tiempo real de ejecución: 0
Total de pasos: 131434424
```

Figura 5: Prueba con 10^6 términos y 8 hilos.

Ascensor

Prueba 1:

Prueba con 100 personas, mostrando el inicio y el final de la ejecución del programa.

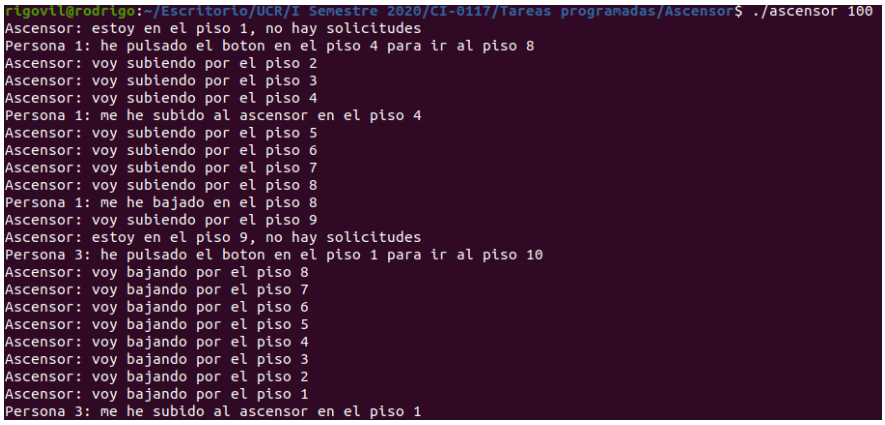

Figura 6: Prueba con 100 personas al comenzar el programa.

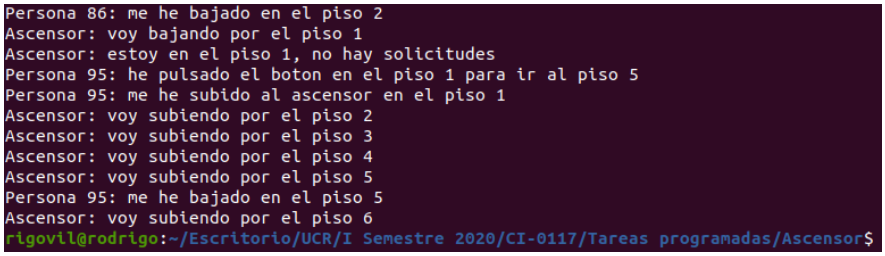

Figura 7: Prueba con 100 personas al terminar el programa.

FIN