CI-0117 Laboratorio 6 Paralela y Concurrente
Rodrigo Vilchez Ulloa | B78292

1.a. Correr el programa Agua en C++

• Al ser utilizado el método fork, este va a crear "copias del programa" tantas veces como sea llamado, pero al estar este llamado dentro de un ciclo for, la cantidad de procesos creada por hijo es bastanta grande, por lo que no se crean N procesos como inicialmente se podría creer, es por esto que hay muchas salidas a consola que muestran creaciones de Oxígeno e Hidrógeno y de moléculas de agua. Es por eso que, por ejemplo, el proceso 4 se ve que imprime muchas veces en pantalla, pero es simplemente muchas iteraciones de i = 4 que se realizan a raíz de la creación procesos de forma descontrolada. Existe igualmente un problema de acceso a memoria compartida, pues se están creando moléculas de agua sin haber necesariamentelos átomos suficientes, esto porque no se está impidiendo correctamente que se dé un race condition para sumar o restar a la cantidad de átomos de oxígeno e hidrógeno.

• Demostración

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Semana 6/Agua$ make
g++ -c -g agua.cc
g++ -g agua.o Semaphore.o -o Agua
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Semana 6/Agua$ ./Agua
Se creo un atomo de oxigeno [1]
Se creo un atomo de oxigeno [2]
Se creo un atomo de oxigeno [2]
Se creo un atomo de hidrogeno [3]
Se creo un atomo de oxigeno [4]
Se creo un atomo de hidrogeno [3]
Molecula de agua creada por un H [3]
Se creo un atomo de hidrogeno [5]
Se creo un atomo de oxigeno [4]
Se creo un atomo de oxigeno [6]
Se creo un atomo de hidrogeno [3]
Molecula de agua creada por un H [3]
Se creo un atomo de hidrogeno [3]
Molecula de agua creada por un H [3]
Se creo un atomo de oxigeno [7]
Se creo un atomo de oxigeno [4]
Se creo un atomo de oxigeno [4]
Se creo un atomo de hidrogeno [5]
Molecula de agua creada por un H [5]
Se creo un atomo de oxigeno [7]
Se creo un atomo de oxigeno [6]
Se creo un atomo de hidrogeno [8]
Molecula de agua creada por un H [8]
Se creo un atomo de oxigeno [7]
Se creo un atomo de oxigeno [10]
```

1.b. Correr el programa Agua con semáforos y sin race condition

• Se utilizó la clase Semáforo implementada para laboratorios anteriores para que, cuando se modifique el valor de la cantidad de átomos de oxígeno o hidrógeno, sea solo un proceso el que pueda realizar esta acción, dependiendo de si está operando sobre la cantidad de oxígeno o hidrógeno, es por esto que se implementaron dos semáforos. Aún así, el problema de utlizar fork persiste pues se crean procesos de forma descontrolada.

• Demostración

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Semana 6/Agua$ ./Agua
Se creo un atomo de oxigeno [1]
Se creo un atomo de oxigeno [2]
Se creo un atomo de oxigeno [2]
Se creo un atomo de hidrogeno [3]
Se creo un atomo de hidrogeno [4]
Molecula de agua creada por un H [4]
Se creo un atomo de oxigeno [5]
Se creo un atomo de oxigeno [6]
Se creo un atomo de hidrogeno [7]
Se creo un atomo de hidrogeno [8]
Molecula de agua creada por un H [8]
Se creo un atomo de oxigeno [6]
Se creo un atomo de oxigeno [6]
Se creo un atomo de oxigeno [10]
Se creo un atomo de oxigeno [10]
Destruyendo los recursos de memoria compartida
Destruyendo los recursos de memoria compartida
Se creo un atomo de hidrogeno [7]
Se creo un atomo de oxigeno [5]
Se creo un atomo de oxigeno [9]
Se creo un atomo de oxigeno [9]
Se creo un atomo de oxigeno [9]
Se creo un atomo de hidrogeno [4]
Se creo un atomo de oxigeno [9]
Molecula de agua creada por un H [4]
```

1.c. Programa en Java

• Se utiliza la clase de semáforos proporcionada por Java, cada hilo crea una partícula, ya sea oxígeno o hidrógeno, y el hilo va a esperar hasta que su partícula sea utilizada para crear una molécula de agua, esto por como se distribuyen los semáforos en el programa. Como las partículas se crean de forma aleatoria en una cantidad de hilos dada (100 en este caso), es posible que se creen más partículas de hidrógeno u oxígeno de las que se necesiten, por lo que habrá hilos que van a quedarse esperando a que se utilice su partícula para crear agua, pero esto no va a suceder pues se determinó una cantidad de hilos para ser creados, entonces el hilo principal que ejecuta el join para unir los hilos tampoco se va a ejecutar y el programa va a quedar corriendo. Las condiciones de carrera se solucionan implementando AtomicInteger, que en el fondo son enteros, pero que java da la facilidad para que esas variables de ese tipo sean modificadas y accesadas por un hilo en un mismo momento, por lo que si un hilo está modificando esa variable, los demás hilos deberán esperar a que este termine para modificarla también.

- Demostración

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Semana 6/Agua$ java Agua
Starting O thread 67 ... cO(0), cH(0)
Starting O thread 92 ... cO(1), cH(0)
Starting H thread 55 ... cO(0), cH(0)
Starting H thread 52 ... cO(0), cH(0)
Starting O thread 54 ... cO(0), cH(0)
Starting O thread 48 ... cO(0), cH(0)
Starting O thread 47 ... cO(0), cH(0)
Starting H thread 2 ... cO(0), cH(0)
Starting H thread 49 ... cO(0), cH(0)
Starting H thread 50 ... cO(0), cH(0)
Starting O thread 88 ... cO(1), cH(0)
Starting O thread 89 ... cO(1), cH(0)
Starting O thread 90 ... cO(1), cH(0)
Starting H thread 16 ... cO(0), cH(0)
Starting H thread 85 ... cO(1), cH(0)
Starting O thread 84 ... cO(1), cH(0)
Starting H thread 26 ... cO(0), cH(0)
Starting H thread 3 ... cO(0), cH(0)
Starting H thread 79 ... cO(0), cH(0)
        H particle making water :) ... cO(9), cH(1)
        H particle making water :) ... cO(5), cH(1)
Starting H thread 87 ... cO(1), cH(0)
Starting H thread 86 ... cO(1), cH(0)
Starting H thread 25 ... cO(0), cH(0)
Starting H thread 28 ... cO(0), cH(0)
Starting O thread 82 ... cO(1), cH(0)
        H particle making water :) ... cO(9), cH(1)
```

1.d. Programa en Python

- Igual que en el programa en Java, se hace uso de los hilos implementados por el lenguaje, estos se sincronizan mediante semáforos para que la partícula en un hilo espere hasta que hayan las partículas suficientes para crear una molécula de agua. Las variables compartidas se implementan utilizando Locks de python, que son asignados a un hilo y por tanto, únicamente ese hilo puede acceder a las variables compartidas para modificarlas. Este lock se adquiere si, al momento de trabajar con la variable, no hay otro hilo esperando.

- Demostración

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Semana 6/A
gua$ python3 Agua.py
Starting O thread 0 ... cO(0), cH(0)
Starting O thread 1 ... cO(0), cH(1)
Starting O thread 2 ... cO(0), cH(2)
        O particle making water :) ... 2 ... cO(0), cH(2)
Starting O thread 3 ... cO(0), cH(0)
Starting O thread 4 ... cO(0), cH(1)
Starting O thread 5 ... cO(0), cH(2)
Starting O thread 6 ... cO(0), cH(2)
        O particle making water :) ... 5 ... cO(0), cH(2)
Starting O thread 7 ... cO(0), cH(2)
Starting O thread 8 ... cO(0), cH(3)
Starting O thread 10 ... cO(0), cH(2)
Starting O thread 11 ... cO(0), cH(2)
Starting O thread 9 ... cO(0), cH(1)
        O particle making water :) ... 8 ... cO(0), cH(2)
        O particle making water :) ... 11 ... cO(0), cH(3)
        O particle making water :) ... 9 ... cO(0), cH(3)
```

1.e. Programa Agua utilizando std::

• Se implementó la misma lógica que en los programas anteriores, solo que esta vez los hilos fueron creados con la clase *thread* proporcionadas por la std, se creo un array para una capacidad de N hilos, se implementaron semáforos de la clase que ya se ha creado para otros programadas, y se utilizaron mutex, también de la clase std para restringir el acceso a variables críticas como lo son la cantidad de hidrógeno y oxígeno. Correr el programa utilizando el comando **make**.

• Demostración

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Semana 6/A
gua/Agua std$ make
g++ -c -g agua.cc
g++ -g agua.o Semaphore.o -o Agua -lpthread
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Semana 6/A
gua/Agua std$ ./Agua
Se creo un atomo de oxigeno [0]
Se creo un atomo de oxigeno [1]
Se creo un atomo de oxigeno [2]
Se creo un atomo de oxigeno [3]
Se creo un atomo de oxigeno [4]
Se creo un atomo de hidrogeno [5]
Se creo un atomo de hidrogeno [6]
Se creo un atomo de oxigeno [7]
Molecula de agua creada por un H [6]
Se creo un atomo de oxigeno [8]
Se creo un atomo de oxigeno [9]
Se creo un atomo de hidrogeno [10]
Se creo un atomo de hidrogeno [11]
Se creo un atomo de hidrogeno [12]
Molecula de agua creada por un H [11]
Se creo un atomo de hidrogeno [13]
```

2.a. Monitor y Variables de Condición

• Un monitor es una estructura de datos que pertenece a los hilos que permite que no se den race conditions al momento en que algunos hilos quierna hacer uso o manipular datos o variables que sean compartidas. Una variable de condición permite a un hilo suspender su ejecución hasta que cierta condición se cumpla y permita que el hilo pueda seguir ejecutándose, está asociada a un mutex, y otro hilo puede indicarle al hilo suspendido que puede continuar.

2.b. Funcionamiento de SP y SV en la clase Semáforos

• La función **SP** crea un array con dos estructuras sembuf, que definen una operación en un semáforo, estas indican que a ambos semáforos se les va a aplicar un Wait, es por esto que en el método **semop**, se le envía todo el array P como segundo parámetro, para que aplique ambas operaciones a ambos semáforos. La función **SV** hace exactamente lo mismo que **SP**, solo que en lugar de ejecutar un Wait, ejecuta un Signal.

2.c. Programa philoFork-gv std::

• Para manipular la pantalla, el programa utilizaría la librería proporcionada por **curses.h**, esta contiene distintas funciones, con parámetros distintos, que permiten que en la consola se visualice ese cuadro de texto, que también es creado por el usuario, "a mano" en este caso, pues le va indicando línea por línea lo que debe mostrar la pantalla, y esta se va actualizando cada vez que ocurra un cambio en el estado de los pensadores. Las variables globales son un semáforo que representa la mesa y un mutex, lo que hace es sincronizar a un filósofo junto con otro que no esté adyacente al primero, mediante ese semáforo y hace uso de las funciones que anteriormente se explicaron para que dos filósofos se "duerman" o "activen" dependiendo de la ejecución aleatoria del programa. Se le asigna un tiempo aleatorio a cada pensador para la acción de pensar o de comer, cuando uno de ellos "quiera comer" puesto que se acabó su tiempo aleatorio para pensar, se llama al semáforo para que el programa sepa que hay alguien esperando por los palillos, es aquí donde se muestra el estado de "hambriento".

2.d. Programa philoFork-ngv std::

• La lógica de este programa es la misma que en el program anterior, solamente que se está prescindiendo del uso de variables globales, y como reemplazo se utilizan los datos del mutex y de la mesa como memoria compartida, utilizando los métodos de **shmget(), shmat(),** etc... y los semáforos pasan a estar como métodos en lugar de como una clase externa.

2.e. Versión con pthreads

• En está versión, se utiliza la librería proporcionada por *thread* y la proporcionada por mutex, para implementar bloqueos en secciones críticas. Se crea un array de hilos donde se almacena cada filósofo y se llama al método que aleatoriamente le proporciona un estado. Para sincroniarlo, se crea una clase *diningPh*, que contiene los métodos de coger un palillo, ponerlo en la mesa, etc. En el otro archivo, que contiene el main, se crea un objeto de esta clase y permite así sincronizar ambas etapas del programa. Utiliza mutex para otorgar acceso a un solo hilo a variables de uso compartido, así como de funciones para "dormir" y "despertar" a un hilo propias de la librería importada.

3.a. Programa Agua con OpenMP

• Para implementar el programa Agua utilizando OpenMP, se utilizaría la misma lógica que con std::thread o fork, donde en todas se establece una cantidad de hilos por crear. Una vez teniendo esta información, se utilizaría la sentencia **pragma omp parallel num threads(numHilos)** para crear todos los hilos necesarios, y el bloque en paralelo sería el que, aleatoriamente escoge si el hilo va a crear un átomo de oxígeno o uno de hidrógeno. Se utilizaría también **pragma omp critical** antes de cada operación en las variables globales, que son la cantidad de átomos de hidrógeno y oxígeno con las que cuenta el programa en cierto momento, estas operaciones serían de incrementar la cantidad de un átomo u otro, o de restarlas en caso de que sean utilizadas para crear una molécula de agua. Para implementar algo similar con los semáforos, es hacer uso del paso de mensajes que utiliza openMP entre los hilos creados, donde cada uno tiene una "cola de mensajes" a las cuales, el hilo creador de la molécula de agua les envía un mensaje para indicarles que su átomo ya ha sido utilizado y que puede terminar entonces. Hasta no recibir este mensaje, el hilo no termina su ejecución, por lo que la lógica detrás de los semáforos seguiría siendo la misma.