Universidad Nacional Autónoma de México

Facultad de Ciencias

Computación Distribuida

Tarea 10

Johann Ramón Gordillo Guzmán
418046090

Tarea presentada como parte del curso de **Computación Distribuida** impartido por la profesora **M.C Karla Rocío Vargas Godoy**.

19 de Enero del 2021

Link al código fuente: https://github.com/JohannGordillo/

Índice

Página

1. Detectores de fallas en sistemas anónimos **2**

2. Sistemas parcialmente síncronos **3**

3. Dining Philosophers **4**

4. The choice coordination problem **6**

5. Autoestabilización **7**

6. Referencias **8**

1 Detectores de fallas en sistemas anónimos

1. ¿Por qué no es trivial asumir el conocimiento de membresía (knowledge membership) en los algoritmos que implementan los detectores de fallas que cumplen con integridad débil?

   Respuesta.

   Si la identidad de un proceso del sistema no es conocida por el resto de los procesos de ese sistema y además ese proceso falla antes de enviar algún mensaje, el resto de los procesos al no saber su identidad no tiene manera de saber que falló, por lo que jamás se va a sospechar de él y no se cumpliría la propiedad de integridad débil.

2. ¿Qué propiedad debe satisfacer cualquier sistema para poder implementar el algoritmo de detector de fallas Ω en una red de miembros desconocidos?

   Respuesta.

   Que para algún proceso p en el conjunto de los procesos correctos, a cada uno de los procesos q en el conjunto de los procesos correctos, se pueda encontrar una trayectoria desde p en $G(S)$, donde $G(S)$ es la gráfica dirigida generada por el sistema distribuido con el conjunto de procesos correctos como su conjunto de vértices y el conjunto de canales eventualmente oportunos como su conjunto de aristas.

2 Sistemas parcialmente síncronos

3. ¿Para qué sirve el GST?

   Respuesta.

   Denotemos con Δ a la cota superior en el tiempo que tienen los mensajes para ser entregados.

   Consideremos el caso en el que esta Δ es conocida, pero el sistema de mensajería entre los procesos no es confiable. Es decir, un sistema en el que a veces los mensajes se entregan tarde o simplemente no son entregados al proceso receptor.

   Notemos que con estas características, y sin restricciones adicionales, el sistema de mensajería anteriormente descrito es por lo menos tan malo como uno completamente asíncrono, por lo que aplicaría el resultado de imposibilidad FLP.

   Por esta razón es necesario agregar una restricción al sistema: Para cada ejecución existe un GST (*Global Stabilization Time*), que es un tiempo de estabilización global, desconocido para los procesos del sistema distribuido, tal que el sistema de mensajería respetará la cota Δ a partir de ese instante *GST*.

   Notemos además que la existencia del *GST* implica que el sistema es parcialmente síncrono, y que la condición de tener un *GST* es equivalente a tener las propiedades de *seguridad* (ningún par de procesos correctos debe estar en desacuerdo y ningún proceso correcto debe tomar una decisión contraria a las condiciones de validez) y *terminación* (cada proceso correcto eventualmente decide).

4. ¿Qué significa que un protocolo de consenso sea t-resistente?

   Respuesta.

   Un protocolo de consenso es t-resistente (o t-resilient, en inglés) si se puede resolver el problema del consenso con t fallas en el conjunto de procesos del sistema distribuido.

   Para fallas bizantinas sin autenticación, por ejemplo, el consenso t-resistente es posible si y sólo si $N \geq 3t + 1$, donde N es el número de procesos en el sistema.

3 Dining Philosophers

5. Define con tus propias palabras qué es un deadlock.

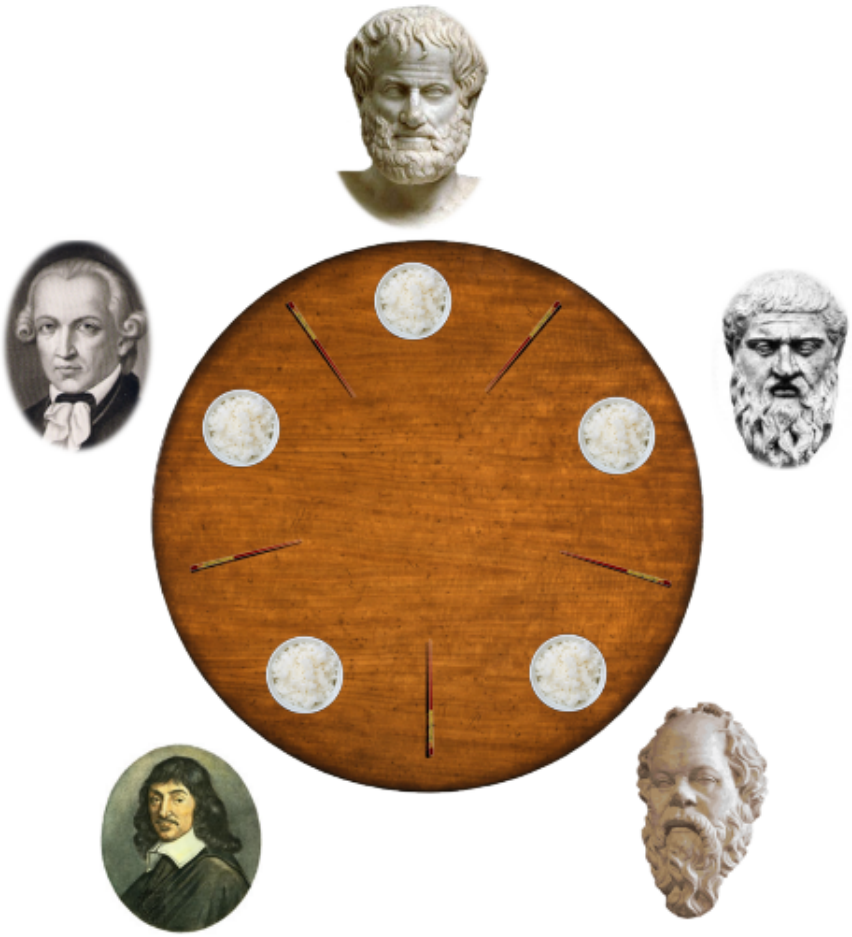
   Respuesta.

   Un conjunto de procesos sufre de un deadlock (o bloque mutuo) si cada uno de los procesos en el conjunto está esperando por un evento que solamente otro proceso del sistema puede causar.

   Usualmente este evento es la liberación de un recurso que está siendo utilizado por otro proceso.

   Cuando tenemos una situación de deadlock, hay un conjunto de procesos bloqueados, cada uno utilizando uno o más recursos y esperando al mismo tiempo un recurso que está siendo utilizado por otro proceso en el sistema.

   Un ejemplo de deadlock puede darse en el problema de los Dining Philosophers cuando los filósofos tienen hambre de manera simultánea y cada uno toma un tenedor de la mesa, porque todos los filósofos se quedarán esperando por el segundo tenedor de manera indefinida.



6. ¿Cuál es la ventaja de usar algoritmos simétricos?

Respuesta.

En el contexto del problema de los Dining Philosophers, los algoritmos simétricos son algoritmos donde las estrategias para todo los filósofos son idénticas. En la práctica, los algoritmos simétricos simplifican el diseño de los sistemas concurrentes.

La ventaja que tienen este tipo de algoritmos es que todos los procesos siguen la misma estrategia, por lo que no es necesario asignar diferentes estrategias basadas en un ordenamiento global. Además, se hace sencillo añadir o eliminar procesos del sistema sin afectar a los otros procesos.

El siguiente algoritmo, conocido como el *Algoritmo de los Filósofos Libres*, es un ejemplo de algoritmo simétrico probabilístico:

THE FREE PHILOSOPHERS ALGORITHM: Code for a philosopher

$R(\cdot)$ is the reflection function on *{right,left}*

```
1  repeat forever
2      think
3      become hungry
4      repeat
5          s ← a random element from {right,left}
6          await (s fork is free) and then take it
7          if R(s) fork is free then take R(s) else release s fork fi
8      until (holding both forks)
9      eat
10     release both forks
11 end
```

4 The choice coordination problem

7. ¿Por qué "lanzamos una moneda" en el algoritmo aleatorio?

   Respuesta.

   En algunos sistemas la terminación no se puede garantizar con certeza, por lo que otra propiedad es más apropiada: la terminación con probabilidad 1.

   Un ejemplo de dicha propiedad es el lanzamiento de una moneda. Eventualmente la cara de la moneda va a ser la ganadora de un lanzamiento.

   Una de las aplicaciones de los lanzamientos de monedas en los sistemas distribuidos es la ruptura de la simetría.

   Los algoritmos que hacen uso del lanzamiento de una moneda se conocen como *Algoritmos Probabilísticos* (o aleatorios).

   En el algoritmo visto en la exposición, cada turista tiene un notepad con un entero k escrito en él, y cuando un turista ve el entero K escrito en el tablón de anuncios del lugar que visita, compara el entero escrito en el tablón con el de su notepad. Si $k = K$, entonces el turista cambia K al siguiente impar más grande, para posteriormente lanzar una moneda. Si en la moneda obtiene cara, entonces lo incrementa de nuevo en 1. Finalmente, el turista actualiza el número en el tablón y en el notepad, para después ir al otro lugar.

   Usamos el lanzamiento de moneda para romper la simetría, ya que inicialmente aparece un 0 en todos los notepads y en los dos tablones de anuncios. Con el lanzamiento de la moneda aseguramos que eventualmente va a cambiar el número, por lo que eventualmente será escrito *here* en uno de los tablones.

   Así, el algoritmo termina con probabilidad 1 y al finalizar todos los turistas estarán dentro del mismo lugar.

8. En el algoritmo determinista, ¿cuántos símbolos distintos se pueden escribir en los Board de ambos lugares? Justifique la respuesta.

   Respuesta.

   En el algoritmo determinista, inicialmente el número escrito en los tablones de anuncios es 0 y el número de cada notepad es único y distinto de 0, de manera que ya no es necesario el uso de una moneda justa, como sí lo era en el algoritmo probabilístico.

   Usando este algoritmo para resolver el Choice Coordination Problem se pueden escribir $n+2$ símbolos distintos en los tablones, donde n es el número de turistas.

   La explicación detrás de esta afirmación es que los números que son escritos en los notepads al comienzo de la ejecución son únicos, y cada uno de estos n números puede ser escrito en los tablones de anuncios. Hasta este punto, tenemos que pueden ser escritos n símbolos distintos, sin embargo, al comienzo de la ejecución un 0 es escrito en ambos tablones y el 0 es distinto de todos los números de los notepads, por lo que el número de símbolos distintos aumenta a $n + 1$. Finalmente, la palabra *here* tambien es un símbolo, lo que completa la suma total de símbolos distintos a $n + 2$.

5 Autoestabilización

9. ¿Cuáles son las propiedades que debe cumplir un algoritmo con autoestabilización?

   Respuesta.

   Un algoritmo con autoestabilización debe cumplir las siguientes propiedades:

   - **Convergencia.**

     El sistema, iniciando desde cualquier configuración arbitraria, eventualmente llega a una configuración legal.

   - **Cerradura.**

     Si el sistema se encuentra en una configuración legal en adelante se mantiene en una configuración legal (siempre que no ocurra una falla).

   La noción de configuración legal depende del problema que intentemos resolver. En general diremos que un algoritmo es autoestabilizable si se puede recuperar de errores arbitrarios y si se puede mantener recuperado mientras no ocurran nuevos errores.

10. ¿Quién propuso por primera vez la autoestabilización y cuál fue uno de los algoritmos que exhibió?

    Respuesta.

    La primera persona en proponer la autoestabilización fue Edsger Dijkstra en Noviembre de 1974 dentro de su publicación *Self-stabilizing systems in spite of distributed control*.

    Propusó este concepto como solución a la incapacidad de los sistemas distribuidos de recuperar su comportamiento una vez que han caído en una configuración ilegal. Argumentaba que los sistemas deben ser lo suficientemente robustos para soportar fallas transitorias sin importar su naturaleza.

    En su publicación, Dijkstra consideró el problema de construir redes robustas con la arquitectura *Token Ring*.

    Dijkstra exhibió el algoritmo *Token Ring Circulation con $m > n$ estados*, donde considera gráficas conexas unidireccionales de n nodos con topología de anillo, donde cada nodo es un proceso y están enumerados de 0 a $n-1$. El algoritmo es el siguiente:

```
1 Code for process 0:
2 if ℓ_0 = ℓ_{n-1} then ℓ_0 ← ℓ_{n-1} + 1
3 Code for process i ≠ 0:
4 if ℓ_i ≠ ℓ_{i-1} then ℓ_i ← ℓ_{i-1}
5
```

6 Referencias

[1] Taubenfeld, G. (2018). *Distributed Computing Pearls*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers.

[2] Aspnes, J. (2017). *Notes on Theory of Distributed Systems*. Yale University.

[3] Jiménez, E., Arévalo, S. & Fernández, A. (2006). *Implementing unreliable failure detectorswith unknown membership*. Information Processing Letters. Elsevier.

[4] Dwork, C., Lynch, N. & Stockmeyer, L. (1988). *Consensus in the Presence of Partial Synchrony*. Journal of the Association for Computing Machinery, Vol. 35, No. 2. pp. 288-323.