Computación Concurrente **Flores González Luis Brandon**
4to Examen parcial

1. **En un sistema distribuido ¿Qué algoritmos existen para la escritura y lectura de relojes?**

   Los algoritmos para la sincronización dependen principalmente del reloj usado, entonces podemos categorizar estos de acuerdo al reloj:

   - **Relojes físicos:** Estos son los que disponen los propios computadores, son dispositivos electrónicos que cuentan las oscilaciones que ocurren en un cristal a una frecuencia definida. Para mantener estos relojes sincronizados entre sí podemos usar los siguientes algoritmos:
     - **Algoritmo de Cristian:** Se utiliza un servidor de tiempo, conectado a un dispositivo que recibe señales de una fuente de UTC, para sincronizar computadores externamente. Bajo solicitud, el proceso S proporciona el tiempo de acuerdo con su reloj.
       Sus principales características de este algoritmo son:
       - Es adecuado para sincronización con UTC.
       - El tiempo de transmisión del mensaje es (T1 - T0)/2
       - El tiempo en propagar el mensaje es (T1 - T0 - I)/2
       - El valor que devuelve el servidor se incrementa en (T1 - T0 - I)/2
       - Para mejorar la precisión se pueden hacer varias mediciones y descartar cualquiera en la que T1 - T0 exceda de un límite

       Donde I es el tiempo del manejador de interrupciones.
     - **Algoritmo de Berkeley:** Se elige un computador coordinador para actuar como maestro, este computador consulta periódicamente a los otros computadores cuyos relojes están para ser sincronizados, llamadas esclavos. Los esclavos le devuelven sus valores de reloj. El maestro estima sus tiempos locales de reloj observando los tiempos de ida y vuelta, y promedia los valores obtenidos.
       Sus principales características son:
       - El servidor de tiempo realiza un muestreo periódico de todas las máquinas para pedirles el tiempo.
       - Calcula el tiempo promedio e indica a todas las máquinas que avancen su reloj a la nueva hora o que disminuyan la velocidad.
       - Si cae el servidor entonces selecciona uno nuevo con el algoritmo de elección.
   - **Relojes lógicos:** Es un contador software que se incrementa monótonamente, cuyos valores no necesitan tener ninguna relación particular con ningún reloj físico. Dentro de estos relojes se encuentran los relojes vectoriales.
     Para mantener estos relojes sincronizados entre sí podemos usar el siguiente algoritmo:
     - **Algoritmo de Lamport:** Nos asegura que si el evento “a” ocurre antes del evento “b” entonces el valor del tiempo del evento “a” es menor al valor del tiempo del evento “b”. También nos asegura que el valor del tiempo del reloj siempre debe ser creciente. En otras palabras este algoritmo asigna tiempo a los eventos.

     Un caso más especial de estos relojes lógicos son los **relojes vectoriales**(vector de N enteros para un sistema de N procesos), estos implementan el siguiente algoritmo:
     - **Algoritmo de Mattern y Fidge:** Nos permite siempre evaluar si dos marcas de tiempo tienen o no relación de precedencia.

2. **En el área del cómputo concurrente ¿Qué es el concepto de "tolerancia a fallas benignas y bizantinas"? y ¿Qué algoritmos o métodos existen?**

Un punto importante a considerar son los posibles fallos que pueden presentarse en el sistema, estos pueden ocurrir en los procesos. En este caso se pueden presenciar dos:

- **Fallas benignas:** Es la imposibilidad de garantizar en un sistema asíncrono que una colección de procesos puedan ponerse de acuerdo en un valor compartido. También se puede decir que son los procesos que sufren un fallo por caída, es decir, no muestra fallos en ningun momento de la ejecución.
  Para tratar estos fallos existen el siguientes **algoritmo**:
  - **Detector de fallo:** Permite saber a los demás procesos mediante un servicio si un proceso ha fallado. Frecuentemente, se usa un *detector de fallo local* que se encuentra en cada proceso, este no es exacto(puede ser fiable o no) pero se encarga de evidenciar si el proceso pudo haber fallado o no. Más detalladamente el **algoritmo es**: Cada proceso *p* manda un mensaje al resto de procesos y lo hace cada T segundos. El detector de fallos utiliza una estimación del máximo tiempo de transmisión de una mensaje de *D* segundos. Si el detector de fallos local en un proceso *q* no recibe un mensaje de *p* dentro de los *T* + *D* segundos desde el último, entonces informa a *q* que *p* quizá tenga un fallo. Sin embargo, si después recibe un mensaje de p, entonces informa a *q* de que *p* no tiene un fallo.

- **Fallas bizantinas:** Los procesos tienen problemas para ponerse de acuerdo en un valor después de que uno o más de dichos procesos haya propuesto cuál debería ser el valor. En el caso de la exclusión mutua, los procesos acuerdan cuáles pueden entrar en la sección crítica. En el caso de una elección, los procesos acuerdan cuál es el proceso elegido.
  Los algoritmos para solucionar este tipo de acuerdo son:
  - **Algoritmo del consenso:**
    Cada proceso p_i comienza en el estado no decidido y propone un solo valor v_i de un conjunto de posibles valores D(i = 1, 2, …, N). Los procesos se comunican entre sí, intercambiando valores. Entonces, cada proceso fija el valor de una variable de decisión d_i. Al hacer esto pasa al estado decidido, en el cual ya no puede cambiar el valor de d_i (i = 1, 2, …, N).
    Los requisitos para un algoritmo de consenso son:
    - Terminación: cada proceso correcto fija su variable de decisión.
    - Acuerdo: el valor de decisión de todos los procesos correctos es el mismo: si p_i y p_j son correctos y están en el estado decidido, entonces $d_i = d_j (i, j = 1, 2, ..., N)$
    - Integridad: si todos los procesos correctos han propuesto el mismo valor, entonces cualquier proceso correcto en el estado decidido ha elegido dicho valor.
  - **Algoritmo de los generales bizantinos:**
    Informalmente, tres o más generales han de ponerse de acuerdo en si atacar o retirarse. Uno de ellos, el comandante, cursa la orden. Los otros, los tenientes del comandante, deben decidir si atacar o retirarse. Pero uno o más de los generales puede ser un traidor o fallar. Si el comandante es el traidor, manda atacar a un general y retirarse a otro. Si el traidor es un teniente, informa a uno de sus iguales que el comandante le mandó atacar mientras que a otro le dice que le ordenó retirarse.
    Este se distingue del de consenso en que un proceso destacado proporciona un valor en el que los otros han de ponerse de acuerdo, en vez de que cada uno proponga un valor. Los requisitos son:

        - Terminación: cada proceso correcto fija su variable de decisión.
        - Acuerdo: el valor de decisión de todos los procesos correctos es el mismo: si p_i y p_j son correctos y están en el estado decidido, entonces $d_i = d_j (i, j = 1, 2, ..., N)$
        - Integridad: si el comandante es un proceso correcto, entonces todos los procesos correctos han de decidir el valor propuesto por el comandante.
    - **Algoritmo de consistencia interactiva:**
      Es una variante del consenso, en el cual todo proceso propone un solo valor. El objetivo del algoritmo es que los procesos correctos se pongan de acuerdo en un vector de valores, uno para cada procesos. A este se le llamara el vector de decisión.
      Los requisitos son:
        - Terminación: cada proceso correcto fija su variable de decisión.
        - Acuerdo: el vector de decisión de todos los procesos correcto es el mismo.
        - Integridad: si p_i es correcto, entonces todos los procesos correctos deciden que v_i es la i-ésima componente de su vector.

3. ¿Qué es un sistema de computo en tiempo real? y ¿Qué algoritmos o métodos existen en dichos sistemas?

Un sistema en tiempo real es aquel que deberá procesar sus entradas y estado actual, con restricciones de seguridad, para entregar las salidas en un tiempo específico. Siendo la capacidad de cumplir o no con este término importante, lo que determina si el sistema es o no de tiempo real.

En este sistema se deben cumplir dos condiciones fundamentales:

- Que las respuestas o salidas sean las correctas.
- Que las respuestas o salidas sean en un tiempo determinado.

Se utilizan dos tipos de algoritmos basados en la localización de recursos para sistemas distribuidos en tiempo real asíncronos y tolerantes a fallas:

- Con base en el **análisis de la sobrecarga** del sistema: Un ejemplo de esta categoría sería el algoritmo periodico HILO para realizar el balanceo de cargas utilizando la métrica de disponibilidad del nodo para ello y así lograr un desempeño óptimo.
- Desde la perspectiva del **análisis de tiempos**.

Además se utiliza un **algoritmo de planificación** que permita manejar los recursos del sistema distribuido con base a dos características principales: el manejo de tiempos y el manejos de eventos. Algunos ejemplos de estos son:

- **“Rate-Monotonic” RM:** Asigna a cada tarea una prioridad fija dependiendo de su periodo de operación, es decir que la tarea con el periodo más corto tiene la mayor prioridad.
- **“Earliest-Dealine_First” EDF:** Asigna la prioridad de manera dinámica, siendo la tarea con mayor prioridad en un momento determinado, la que tenga su plazo(deadline) más cercano.

En ambos las tareas pueden ser “expulsadas” del procesador en cualquier momento y son óptimos.

Cabe destacar que hay una multitud de algoritmos para la planificación, y en general en un sistema de computo en tiempo real ya que este depende del modelo usado por lo que no se puede dar un listado completo.

Pregunta extra:

4. **En un sistema distribuido ¿Qué es exclusión mutua distribuida? ¿Qué algoritmos o métodos existen? y ¿Cuál es su utilización?**

   La exclusión mutua distribuida es el mecanismo de coordinación entre varios procesos concurrentes a la hora de acceder a recursos o secciones compartidas. La exclusión mutua se **utiliza** para acceder a la región critica(bloqueo), manipular los recursos compartidos y liberar el recurso(despierta los procesos en espera).

   Los siguientes algoritmos para este problema son:

   - **Algoritmos centralizados:** Se emplea un servidor que dé los permisos para entrar en la sección crítica, para esto un proceso envía un mensaje de petición al servidor y espera una respuesta por su parte donde esta respuesta constituye el permiso para entrar en la sección crítica.

   - **Algoritmos distribuidos:** En esta categoría tenemos al algoritmo basado en un anillo ya que de esta manera no se necesita un proceso adicional y para lograr esto basta con organizarlos en un anillo lógico. Para este algoritmo se requiere que cada proceso tenga un canal de comunicación hacia el siguiente proceso en el anillo. La exclusión se consigue obteniendo un permiso para entrar a la sección crítica mediante un mensaje que se pasa de un proceso a otro en única dirección alrededor del anillo. Si un proceso no requiere entrar en la sección crítica cuando recibe este permiso, entonces inmediatamente lo hace avanzar hacia su vecino. Un proceso que requiere este permiso espera hasta recibirlo y en este caso lo retiene. Cuando el proceso salga de la sección crítica enviara el testigo hacia el siguiente vecino.

   - **Algoritmos basados en marcas de tiempo:** Los procesos que necesitan entrar en una sección crítica envían un mensaje de petición mediante multidifusión(se envía la petición a múltiples destinos simultaneamente) y pueden entrar en ella solamente cuando el resto de los procesos haya respondido al mensaje. Las condiciones bajo las cuales un proceso responde a una petición se diseñan para asegurar que se cumplan las siguientes condiciones:
     - Solo un proceso puede estar ejecutándose cada vez en la sección crítica.
     - Las peticiones para entrar y salir de la sección crítica al final son concedidas.
     - Si una petición para entrar en la sección crítica ocurrio antes que otra, entonces la entrada a la sección critica se garantiza en ese orden.

Bibliografía

Coulouris, G., Dollimore, J. & Kindberg T.. (2001). *Sistemas Distribuidos Conceptos y Diseño*. Madrid: Pearson Educación. pp. 367-395, 399-439

Perez, F., Pérez, M. & Jose, M.. *Sistemas Operativos Distribuidos*. enero 03, 2018, de Universidad Politécnica de Madrid Sitio web: http://laurel.datsi.fi.upm.es/_media/docencia/asignaturas/sod/sincronizacion.pdf

Francisco, A. & Leonel, M.. (2009). *PLANIFICACIÓN DE SISTEMAS DISTRIBUIDOS EN TIEMPO-REAL*. México. pp. 11-38

(2004). Sistemas Informáticos de Tiempo Real. enero 03, 2018, de Universitat de València Sitio web: https://www.uv.es/gomis/Apuntes_SITR/Conceptos.pdf