**Interacción y cooperación**.

Coordinación Distribuida.

Los procesos distribuidos necesitan coordinar sus actividades. Si un grupo de procesos comparten un conjunto de recursos normalmente se requiere la exclusión mutua para prevenir interferencias y asegurar la consistencia cuando se accede al recurso.

No nos sirve el tener variables compartidas, ni las utilidades proporcionadas por un núcleo central, se requiere una solución basada en el paso de mensajes.

Requisitos de la exclusión mutua.

* E1: Seguridad, solo un proceso accede a la sección crítica.
* E2: Vitalidad, las peticiones entrar/salir son atendidas en algún momento, no hay inanición y abrazo mortal.
* E3: Ordenación, establecer prioridades para el acceso a la sección critica.

Algoritmo basado en servidor central.

Un servidor central es el encargado de dar los permisos de acceso a la sección crítica.

Para acceder a la SC un proceso envía un mensaje al servidor y espera la respuesta de este. El servidor da los permisos en forma de un testigo, si ningún otro proceso tiene el testigo el servidor responde a este. Si el testigo ya fue entregado a otro proceso no contesta y pone la petición a la cola de espera. Al salir de la SC los procesos, devuelven el testigo al servidor.

Requisitos de la exclusión mutua.

* Se cumplen la seguridad y vitalidad.
* No cumple con la ordenación ya que atenderá las peticiones en función de cuando llegan al servidor.

Rendimiento.

* 2 mensajes para la entrega de la sección critican. Incluso cuando no hay ningún proceso ocupándola hay que enviar un mensaje de petición y otro de contestación. Esto retrasa al proceso que hizo la petición.
* 1 mensaje para salir de la sección critica.

Problemas.

Todas las solicitudes al mismo servidor, lo que produce cuello de botella.

Ante una caída o fallo del servidor centra, habría que asignar a otro servidor centra y todas las peticiones se perderían y tendríamos que reiniciarlas.

Caída o fallo de un proceso en la sección critica, lo que implica perdida del testigo.

Algoritmo basa en anillo.

Se crea un anillo lógico, la exclusión se logra por el paso de un testigo entre los nodos del anillo.

Si un proceso recibe el testigo y necesita entrar a la sección critica lo retiene y accede, una vez termina devuelve el testigo a su vecino. Si no quiere entrar pasara el testigo a su vecino.

Requisitos de exclusión mutua.

* Se verifica el requisito de seguridad pues solo hay un testigo para todo el anillo.
* La vitalidad también se cumple.
* La ordenación no se puede cumplir debido a que cada proceso puede coger el testigo cuando pasa por el aunque haya un nodo esperando ya por el.

Rendimiento.

* Los procesos envían mensajes aunque ninguno quiera acceder a la sección crítica.
* Retraso en los proceso que quieren entrar a la sección critica desde 0 mensajes a N mensajes.
* Retraso de sincronización entre 1 mensaje y N mensajes.

Problemas.

* Se carga la red, el mensaje de solicitud recorre el anillo aunque ningún proceso quiera acceder a la sección crítica.
* Cuando un proceso se cae necesita reconfigurarlo y si tenía el testigo regenerarlo. Asegurarse de que el proceso este caído por que podría darse el caso de tener dos testigos en la red.
* Desconexión o ruptura de la red, implica una pérdida del testigo.

Algoritmo basado en relojes lógicos.

Para los algoritmos basados en relojes lógicos se tiene que cumplir dos premisas, cada uno de los procesos debe conocer las direcciones de todos los demás y cada proceso tiene que tener un reloj lógico.

Ricart y Agrawala.

Todos los procesos piden y conceden acceso a la sección crítica. Si un proceso quiere entrar a la sección crítica les pregunta a todos los demás si puede entrar. Cuando todos los demás contestan entra.

El acceso se obtiene a través de un mensaje en el que se almacena el nombre del proceso y la hora. Cada proceso se guarda el estado de la sección critica “liberada, buscada o tomada”.

Funcionamiento:

Si un proceso no esta en la sección critica y no desea entrar en ella entonces contesta inmediatamente al emisor, si no este no responde a la petición y la encola.

Si un proceso no esta en la sección critica pero ha solicitado su acceso comprueba si lo hizo antes, si el otro proceso solicito antes entonces le contesta al emisor sino el receptor encola la petición y no responde nada.

Cuando un proceso sale de la sección crítica avisa a todos los demás y responde a cualquiera de las peticiones de su cola.

Requisitos de exclusión mutua.

* Se cumplen los tres requisitos, solo un proceso accede a la sección crítica, todas las peticiones de entrada y salida son atendidas en algún momento y con los relojes lógicos de Lamport nos aseguramos que todos entren en el instante que les corresponde.

Rendimiento.

* Sin soporte multicast 2(n-1) mensajes, donde n es el número total de procesos. El algoritmo fue refinado hasta conseguir N mensajes.
* Con soporte multicast N mensajes.

Problemas.

* Algoritmo muy costoso.
* Los procesos implicados reciben y procesan cada solicitud. Igual o peor congestión que el servidor central.
* El fallo de cualquier proceso bloquea el sistema, ya que el proceso no contesta a las peticiones y se interpreta que la sección critica esta ocupada. La solución es que el receptor envié mensaje de permiso o denegación. Si el emisor no recibe alguna de esta supondrá que la sección esta libre.

Conclusiones.

* Ninguno puede tratar el problema de la caída de un computador o proceso.
* El algoritmo de servidor central es el que tiene menor número de mensajes, pero supone un cuello de botella.
* Es preferible que el servidor que gestiona el recurso implemente también la exclusión mutua.

Algoritmo de elección.

Muchos de los algoritmos distribuidos necesitan que un proceso actúe como coordinador, supondremos que cada proceso conoce los identificadores del resto de procesos, pero desconoce si están activos o inactivos.

El objetivo de todos estos algoritmos es garantizar que un único proceso llega a ser coordinador, aunque varios procesos hayan lanzado el algoritmo concurrentemente.

Cumplen seguridad y vivacidad.

Anillo.

Todos los procesos comienzan siendo no candidatos.

Cualquiera de ellos se percata que el coordinador ha caído, se marca como candidato y lanza un mensaje de elección con un identificador.

Mensaje de elección.

Si el identificador del mensaje es mayor que el del proceso, transmite el mensaje a su vecinos.

Si el identificador es menor:

* Es candidato, no hará nada por que ya ha mandado su mensaje de elección.
* Es no-candidato, sustituye el identificador por el suyo, se marca como candidato y lo reenvía al vecino.

Si el identificador es el suyo, se marca como no-candidato y manda un mensaje de elegido con su identificador.

Mensaje elegido.

Si el identificador no es el suyo, se marca como candidato y lo manda a su vecino.

Peor caso, se da cuenta que el coordinador ha caído el nodo siguiente al que conseguirá ser coordinador. 3n-1 mensajes.

Mejor caso, se da cuenta de que el coordinador ha caído el futuro coordinador. 2n mensajes.

Bully (Algoritmo del matón).

En este algoritmo los procesos pueden caer durante la elección. Se seleccionara el proceso con mayor identificador.

Requisitos:

* Todos los miembros del grupo deben conocer las identidades de los demás miembros.
* Se supone comunicación fiable.

Hay 3 tipos de mensajes:

* Mensajes de elección para anunciar una elección.
* Mensaje de repuesta a un mensaje de elección.
* Mensaje de coordinador anuncia identidad de nuevo coordinador.

Mejor caso, se da cuenta el segundo más alto. (n-2) mensajes.

Peor caso, se da cuenta el más bajo. (n2) mensajes.

El algoritmo comienza cuando uno de los procesos se da cuenta de que el coordinador ha caído. En ese momento envía un mensaje de elección a todos los procesos con identificador mayor al suyo.

Si no recibe respuesta, se nombra coordinador y envía un mensaje de coordinación a todos los procesos de id mas bajo.

Si recibe repuesta, se limitara a esperar el mensaje del coordinador.

Si un proceso recibe un mensaje de elección, responderá a él y enviara un mensaje de elección.

Si un proceso recibe un mensaje de coordinador, guardara el id del mensaje y tratará a éste como coordinador.

Si un proceso se reiniciara, si tiene el id más alto se erigirá como nuevo coordinador, si no, lanzará un mensaje de elección.