Programación en memoria distribuida

Conceptos generales

En arquitecturas de memoria distribuida los procesadores tienen una memoria local y una red de comunicaciones. Para esto usaremos herramientas basadas en el intercambio de mensajes.

Programa distribuido: Programa concurrente comunicado por mensajes. Supone la ejecución sobre una arquitectura de memoria distribuida, aunque puedan ejecutarse sobre una de memoria compartida.

Primitivas de pasaje de mensajes: Interfaz con el sistema de comunicaciones 🡪 semáforos + datos + sincronización.

Los procesos SOLO comparten canales físicos o lógicos. Las canales se clasifican de maneras distintas:

* Cuantos o quienes pueden enviar o recibir datos por esos canales: Mailbox, input port, link
* Según la dirección en la que puede ir la información en un canal: Uni o bidireccionales
* Donde deben estar los procesos para establecer una comunicación: Sincrónicos o asincrónicos

Características

* **Los canales son lo único que comparten los procesos**: Las variables son locales a un proceso, la exclusión mutua no requiere mecanismo especial, los procesos interactúan comunicándose y son accedidos por primitivas de envío y recepción.
* **Mecanismos para el procesamiento distribuido**: Pasaje de mensajes sincrónicos/asincrónicos, llamado a procedimientos remotos (RPC), Rendezvous.
* **La sincronización de la comunicación interproceso depende del patrón de interacción**: Productores y consumidores, clientes y servidores, pares que interactúan.

Cada mecanismo es más adecuado para determinados patrones.

Relación entre mecanismos de sincronización

Iba a escribir, pero la diapo lo explica bien:

Diagrama

Descripción generada automáticamente

Los semáforos están relacionados con el pasaje de mensajes, principalmente con los asincrónicos, porque los mensajes asincrónicos al dejar las cosas en una cola y seguir es parecido a como funciona la sincronización con semáforos.

Pasaje de mensajes asincrónicos

Los PMA usan canales que funcionan como colas de mensajes enviados y aún no recibidos

Los canales se declaran como **chan** *ch (id : tipo, … , id: tipo)*

* **chan** entrada (char)
* **chan** acceso\_disco(INT cilindro, INT bloque, INT bloque, CHAR\* buffer)
* **chan** resultado[n] (INT)

Tienen dos operaciones:

* **Operación send 🡪** agrega un mensaje al final de la cola (“ilimitada”) de un canal que no bloquea al emisor. Es atómica, si hay dos haciendo send al mismo canal entran uno a la vez.
* **Operación receive 🡪** Recibe un mensaje del canal que demora( bloquea) al receptor hasta que en el canal haya al menos un mensaje, luego toma el primero y lo almacena en variables locales.

Las variables del receive deben tener los mismos tipos que la declaración del canal, y es una primitiva bloqueante, ya que produce un delay

Que sea bloqueante quiere decir que el proceso NO hace nada hasta recibir un mensaje en la cola correspondiente del canal. NO es necesario hacer polling.

El acceso a los contenidos de los canales es atómico y respeta el orden FIFO, el primero que entra es el primero que sale. En un principio son ilimitados, pero en implementaciones reales tienen un tamaño de buffer asignado.

Se supone que los mensajes NO se pierden ni modifican , porque todo mensaje enviado en algún momento puede ser leído.

La operación **empty(ch)** determina si la cola de un canal está vacía, lo cual es útil cuando el proceso puede hacer trabajo productivo mientras espera un mensaje, pero debe usarse con cuidado al tener más de un receptor para un mismo canal.

Si hay más receptores puede pasar que empty sea false y que no haya más mensajes cuando se sigue ejecutando o que empty sea true aunque haya un mensaje en el canal cuando se sigue la ejecución.

Los canales son declarados globales a los procesos ya que pueden ser compartidos y depende como se usan pueden ser:

* **Mailbox:** Cualquier proceso puede enviar o recibir por alguno de los canales declarados.
* **Input port:** Un solo receptor y varios emisores
* **Link:** Un único emisor y un único receptor. Provee un camino entre el emisor y sus receptores

Productores y consumidores: Red de ordenación

Un **filtro** es un proceso que recibe mensajes de uno o más canales de entrada y envía mensajes a uno o más canales de salida. La salida de un filtro es función de su estado inicial y de los valores recibidos.

La función del filtro puede especificarse por un predicado que relacione los valores de salida con los de entrada.

**Problema:** Ordenar una lista de N números de modo ascendente. Podemos pensar en un filtro sort con un canal de entrada y un canal de salida. Para saber cuando se consiguen todos los números es necesario conocer N, enviar N como el primer elemento enviado o cerrar la lista de N números con un valor especial.

Una solución más eficiente que una secuencial es una red de pequeños procesos que ejecutan en paralelo e interactúan para armar la salida ordenada. La **idea** es mezclar repetidamente y en paralelo dos listas ordenadas de N1 elementos cada una en una lista ordenada de 2 \* NI elementos.

Con PMA pensamos en 2 canales de entrada por cada canal de salida y un carácter especial que cierra cada lista parcial ordenada.

La red se hace con filtros merge donde cada merge recibe valores de 2 streams de entrada ordenados y produce un stream de salida ordenado out.

Los streams terminan en un carácter especial y el merge agrega ese carácter al final. El merge compara los próximos dos valores recibidos y envía el menor a out.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Esto puede programarse usando:

* **Static naming:** Es un arreglo global de canales y cada instancia de merge recibe desde 2 elementos del arreglo y envía a otro.
* **Dynamic naming:** Canales globales, parametrizar los procesos y darle a cada proceso 3 canales al crearlo. ( merges idénticos, pero necesitas un coordinador )

Los filtros podemos conectarlos de distintas maneras, solo necesitamos que la salida de uno cumpla con las suposiciones de entrada del otro, pueden reemplazarse si se mantienen comportamientos de entrada y salida.

Clientes y servidores: Monitores activos

Un servidor maneja pedidos de procesos clientes, cómo se implementa con PMA?

El pasaje de mensajes se puede simular con monitores y viceversa.

El monitor encapsula variables que registran el estado y provee un conjunto de procedures. Los simulamos usando procesos servidores y PM, como procesos activos en lugar de como conjuntos pasivos de procedures.

Un servidor es un proceso que maneja pedidos de otros procesos clientes. Un cliente que envía un mensaje a un canal luego recibe el resultado desde un canal de respuesta propio.

Naturalmente, el proceso servidor reside en un procesador físico y M procesos clientes residen en otros N procesadores.

Diagrama

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Se puede generalizar esta solución con una única operación para considerar múltiples operaciones.

El if del servidor será un case con las distintas clases de operaciones.

El cuerpo de cada operación toma datos de un canal de entrada en args y los devuelve al cliente adecuado en resultados.

Solución para múltiples operaciones:

Tabla

Descripción generada automáticamente con confianza baja

Clientes y servidores: Monitores con múltiples operaciones y variables condición

Para este caso los clientes se mantienen igual, solo cambia el servidor.

Consideramos un caso de un manejo de múltiples unidades de un recurso:

* Los clientes adquieren y devuelven unidades.
* Las unidades libres se insertan en un conjunto sobre el que se harán las operaciones insertar y remover.
* El número de unidades disponibles es lo que controla nuestra variable de sincronización por condición.

Cuando el servidor tiene 2 operaciones y no hay unidades disponibles, el servidor no puede esperar hasta responder al pedido, debe salvarlo y diferir la respuesta, cuando una unidad es liberada atiende un pedido salvado enviando la unidad.

Conclusiones

El monitor y el servidor muestran la dualidad entre monitores y PM, hay una correspondencia directa entre el mecanismo de ambos. La eficiencia de los dos depende de la arquitectura física de soporte.

Si tenemos memoria compartida conviene monitores pero en arquitecturas físicamente distribuidas son más eficientes los mecanismos de PM.

Tabla

Descripción generada automáticamente con confianza media

Clientes y servidores: Sentencia de alternativa múltiple

Mismo problema con sentencias de alternativa múltiple (do , if , etc), esta resolución tiene sus ventajas y desventajas a la solución simulando monitores.

El problema con esta solución es que puede producir busy waiting, cómo se usa un if de alternativa múltiple si todas las condiciones son falsas me quedo girando en el while true. No es algo bueno pero en el caso de PMA va a ser permitido cuando otro tipo de solución sea demasiado compleja de implementar.

Clientes y servidores: Continuidad conversacional

Existen A alumnos que hacen consultas a D docentes, el alumno espera a que el docente lo atiende y a partir de ahí comienza a realizar las diferentes consultas hasta que no le queden dudas.

Los alumnos son los procesos clientes y los docentes son los servidores. Los procesos servidores son idénticos y cualquiera de ellos que esté libre puede atender un requerimiento de un alumno. Todos los alumnos piden una consulta por un canal global y reciben una respuesta de un docente a través de un canal propio, esto es así para que cuando haya un docente libre pueda ver la consulta e iniciar una conversación con ese alumno particular.

Tabla

Descripción generada automáticamente

En este ejemplo la interacción entre clientes y servidores se denomina continuidad conversacional, atención es un canal compartido por el que cualquier Docente puede recibir, si cada canal puede tener un solo receptor se necesita otro proceso intermedio.

En la práctica no nos dejaban tener un canal de múltiples receptores si no recuerdo mal.

Pares interactuantes : intercambio de valores

**Problema:** Cada proceso tiene un dato local V y los N procesos deben saber cual es el menor y cual es el mayor de los valores.

Ejemplo donde los procesadores están conectados por tres modelos de arquitectura: centralizado, simétrico y en anillo circular.

La arquitectura centralizada es apta para una solución donde todos envían su dato local V al procesador central y el proceso central calcula el mínimo y máximo. Una vez que los calcula les manda el dato a todos los procesos.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Se podría usar un solo canal resultado si los procesos generan un único valor, si esto tuviese que iterarse en la segunda iteración un proceso puede agarrar un resultado de la primer iteración que era para otro y se arma un re kilombo porque el otro proceso agarra un resultado errado y todo eso.

En la **arquitectura simétrica** hay un canal entre cada par de procesos. Todos los procesos ejecutan el mismo algoritmo. Cada proceso transmite su dato local V a los N-1 procesos restantes. Luego recibe y procesa los N-1 datos que le faltan de modo que toda la arquitectura está calculando el mínimo y máximo y toda la arquitectura tiene acceso a los N datos.

Hay N.(N-1) mensajes en total, muchísimos más que el anterior.

Si tuviésemos una primitiva broadcast habría que mandar N mensajes.

Texto

Descripción generada automáticamente con confianza media

Se necesita si o si un canal para cada proceso porque si todo el mundo envía su valor al mismo canal y envia n-1 mensaje se rompe todo, se pierden datos , te agarras el mismo que mandaste, etc.

En la **solución de anillo circular** P[i] recibe mensajes de P[i-1] y envía mensajes a P[i+1] . P[n-1] tiene como sucesor a P[0]

Es un esquema de 2 etapas, en la primera los procesos reciben 2 valores y calculan resultados parciales transmitiendo el resultado a su sucesor. En la segunda etapa todos deben recibir la circulación del máximo y mínimo global.

P[0] debe ser algo diferente para arrancar el procesamiento.Escala de tiempo

Descripción generada automáticamente

Se requerirán (2N) -1 mensajes.

Si bien el número de mensajes es lineal los tiempos pueden ser muy diferentes con respecto a la solución centralizada. En los otros casos los mensajes se pueden enviar en forma concurrente, en esta son secuenciales, haciendo que el tiempo sea mucho mayor.

Ya no es simétrico, y eso no es bueno.

La simétrica es la más corta y sencilla de programar, pero usa el mayor número de mensajes sin broadcast. Pueden transmitirse en paralelo si la red soporta transmisiones concurrentes pero el overhead de comunicación agota el speedup. La centralizada y anillo en cambio tienen una cantidad lineal de mensajes, pero tienen distintos patrones de comunicación que llevan a distinta performance.