Clase 7

Pasaje de mensajes sincrónicos (PMS)

Diferencia con PMA

La diferencia radica en la primitiva de transmisión `Send`. En PMS es bloqueante y la llamaremos `sync_send`.

- El transmisor queda esperando que el mensaje sea recibido por el receptor.
- La cola de mensajes asociada con un send sobre un canal se reduce a 1 mensaje.
- Naturalmente el grado de concurrencia se reduce respecto de la sincronización por PMA.

Si bien `send` y `sync_send` son similares, la semántica es diferente y las posibilidades de `deadlock` son mayores en comunicación sincrónica.

En el caso de **PMS**, el productor es capaz de realizar sus cálculos mucho más rápido que el consumidor en las primeras n/2 operaciones. Pero las siguientes, el productor se vuelve mucho más lento que el consumidor. n esta situación, los pares de envío y recepción de mensajes entre el productor y el consumidor se completarán asumiendo la demora del proceso que más tiempo consuma. Esto significa que el tiempo total para completar las operaciones será dominado por el proceso más lento.

En **PMA** el productor es más rápido que el consumidor, por lo que sus mensajes se acumulan en una cola. Luego, cuando el consumidor se vuelve más rápido, empieza a procesar la cola de mensajes del productor. Esto permite que la comunicación sea más eficiente, ya que el productor y el consumidor pueden trabajar a su propia velocidad sin esperar el uno al otro.

Conclusiones

- PMA tiende a tener mayor concurrencia, porque permite que los productores y consumidores trabajen independientemente.

- Si existen más de un productor y/o consumidor en el sistema, la dinámica puede volverse más compleja. En el caso de PMS, la relación de velocidad entre los productores y consumidores individuales puede variar. En PMA, la cola de mensajes puede tener mensajes de múltiples productores y ser consumida por múltiples consumidores.

CSP- Lenguaje para PMS

- CSP tiene la idea de que los procesos se comunican entre sí mediante la sincronización en vez de compartir datos directamente. Cada proceso es independiente y se ejecuta secuencialmente y la comunicación es mediante eventos que suceden sincronizadamente.
- La comunicación entre procesos se hace a través de canales, que son enlaces directos entre 2 procesos. Estos canales solo pueden transmitir datos en una dirección a la vez y también tienen identificadores.
- Las sentencias de entrada se representan con el símbolo `?` y las de salida con `!`. Por ejemplo

  ```
  process A { . . . B ! e; . . . }
  process B { . . . A ? x; . . . }
  ```

  El proceso A envía un evento `e` al evento B usando la sentencia de salida `!` y el proceso B recibe ese evento con la sentencia de entrada `?` y lo asigna a la variable X.
- Las sentencias de entrada y salida deben hacer "match", es decir, deben esperar que la otra parte realice una operación complementaria.
- La comunicación entre procesos a través de sentencias de entrada y salida tiene el efecto de una asignación distribuida, donde un valor se transfiere desde un proceso emisor a un proceso receptor, lo que permite la sincronización de eventos y la comunicación entre procesos de manera segura y controlada.

Sentencias

- `Destino ! port(e1..en)` un proceso Destino envía un mensaje compuesto por los valores e1,e2,..en a través del canal identificado por port.

- `Fuente ? port(x1,..xn)` un proceso Fuente espera y recibe un mensaje a través del canal identificado por “port” y almacena los datos en las variables x1,x2,…,xn.

Comunicación guardada

Está la posibilidad de que la comunicación con `!` y `?` puedan ser bloqueantes entonces se utiliza la comunicación guardada para evitar este suceso.

- Comunicación no sincrónica: es posible que un proceso quiera comunicarse con otros procesos, pero no conozca el orden en el que esos procesos desean comunicarse. La CG permite que un proceso espere hasta que una condición específica se cumpla antes de continuar la comunicación.
- Buffering y gestión de mensajes: CG servirá para implementar mecanismos de buffering, donde un proceso puede recibir o enviar mensajes cuando sea apropiado, en lugar de estar bloqueando esperando que otro proceso esté listo para la comunicación.
- Utilización de condicionales: las operaciones de comunicación guardada permiten el uso de condiciones para decidir cuándo se realiza la comunicación.

La comunicación con CG es no determinística, lo que significa que el resultado de la comunicación no se pueda predecir de ante mano.

- Una sentencia de comunicación guardada se expresa en la forma `B;C -> S` donde `B` es una condición booleana que determina si la comunicación se puede llevar a cabo, `C` representa la acción o el comando a ejecutarse como resultado de `B` y `S` es un conjunto de sentencias que se ejecutarán si la comunicación tiene éxito.
- La combinación de `B` y `C` en una sentencia de comunicación guardada se denomina "guarda”.
- La característica distintiva de la comunicación guardada es que puede bloquearse. Esto sucede cuando "B" es verdadero, pero la ejecución de "C" no puede llevarse a cabo debido a algún tipo de demora, como un bloqueo en otro proceso. En ese caso, la comunicación se suspende y el proceso que está esperando la comunicación queda bloqueado hasta que las condiciones permitan que se complete la comunicación.

Paradigmas de interacción entre procesos

En la programación concurrente hay varios paradigmas y modelos de interacción entre procesos para resolver determinados problemas.

1. **Algoritmos Heartbeat:** los procesos deben intercambiar información periódicamente usando mecanismos tipo send/receive. La idea es permitir que los procesos se mantengan informados sobre el estado de otros procesos.

   Es útil para soluciones iterativas que se quieren paralelizar. Se distribuirá la carga entre los workers.

2. **Algoritmos Pipeline:** se usan cuando la información debe pasar a través de una serie de procesos secuencialmente, donde cada proceso realiza alguna operación específica en la información y la pasa al siguiente proceso.

   Un pipeline es un arreglo lineal de procesos “filtro” que reciben datos de una entrada y entregan resultados por un canal de salida.

3. **Master/worker:** el master es responsable de coordinar el trabajo y administrar las tareas. Su función principal es dividir el trabajo en pequeñas unidades llamadas tareas y distribuidas a los trabajadores para su procesamiento. Por otro lado, los worker son procesos que realizan las tareas asignadas por el maestro. Cada trabajador es independiente y puede realizar su trabajo en paralelo con otros trabajadores.

   Se supone que un conjunto de workers comparten una “bolsa” con tareas independientes. Los workers sacan una tarea, la ejecutan y probablemente crean nuevas tareas en la bolsa.

4. **Probes (send) y echoes (receive):** la interacción entre procesos se hace mediante mensajes de envío y de recepción. Se suele usar para recorrer estructuras de datos como grafos o árboles.

5. **Algoritmos broadcast:** se utilizan para transmitir información desde un proceso emisor a múltiples procesos receptores en una arquitectura distribuida. Este paradigma permite lograr una distribución global de información en todo el sistema.

6. **Token passing:** la arquitectura distribuida recibe información global a través del paso de "tokens" de control o datos. Los "tokens" son elementos que circulan de un proceso a otro de manera secuencial. Este paradigma se utiliza para coordinar y sincronizar procesos en una red distribuida.

7. **Servidores replicados:** se manejan recursos compartidos, como dispositivos o archivos, mediante múltiples instancias de servidores distribuidos. La replicación de servidores garantiza la disponibilidad y la redundancia de recursos críticos. Si un servidor falla, otros servidores replicados pueden tomar el control y continuar proporcionando el servicio.