Concurrencia y Distribución (Prácticas)

2010/2011

Dr. Arno Formella
Dr. Miguel Reboiro Jato
María Stella Orozco Ochoa

Departamento de Informática
Universidad de Vigo

10/11

Comentarios

Las tareas de programación parecen simples leyendo su descripción, pero no son tan simples pensando en su realización.

Empezando I

1. Consigue el “Hola Mundo” en Java.
2. Consigue un “Hola Mundo, soy hilo ...” usando varios hilos (mira detenidamente en los manuales de Java la clase `Thread` y también la interfaz `Runnable`).
3. Mide cuantos hilos se puede lanzar y mantener vivos simultaneamente.
4. Mide el tiempo que un sólo hilo necesita para escribir por ejemplo 100000 veces "Hola Mundo", y cuanto tiempo necesitan por ejemplo 1000 hilos distribuyendo el trabajo entre ellos. Realiza un diagrama dibujando tiempo de ejecución frente a números de hilos.

Empezando II

5. Cambia el trabajo que realiza un hilo (escribir a consola) por algo que no tenga salida, observa las diferencias comparándolo con los resultados de antes (realiza los mismos gráficos que arriba).
6. Asegúrate que tu programa termina bien, es decir, que todos los hilos participantes lleguen a su último "}".

Describe de forma precisa todos tus observaciones (dependiendo del tipo de S.O., de la ocupación del ordenador, del trabajo por realizar, etc.).

PingPONG I

1. Implementa un `pingPONG` perfecto. Ten los siguientes detalles en cuenta:
   - Experimenta con los diferentes intentos presentados en los apuntes. (Observa y apunta sus propiedades.)
   - Desarrolla una solución con las siguientes propiedades:
     - Usa tres hilos (un hilo para el programa principal, o el árbitro, y un hilo para cada jugador).
     - El árbitro inicia el juego (mensaje previo a la pantalla).
     - Los jugadores producen sus `ping`s y `PONG`s alternamente.
     - El árbitro termina el juego después de cierto tiempo o cierto número de jugadas (con mensaje previo a la pantalla).
     - Los jugadores realizan como mucho un intercambio de pelota más.

PingPONG II

- Ambos jugadores/hilos terminan correctamente (con mensaje previo a la pantalla).
- El árbitro escribe el último mensaje.
- El programa termina correctamente.

- Observa la diferencia entre el uso de `notify()` y `notifyAll()`, sobre todo en respecto a los despiertos "inútiles" de hilos.

2. Amplía el programa para que genere tantos jugadores (hilos) como se desea (ya conoces el máximo de la práctica anterior) y genera una tabla de tiempos de ejecución incluyendo también el tiempo de ejecución con el caso del mismo programa usando un solo hilo (que entonces imprima solamente `ping`).

PingPONG III

3. ¿Cuál sería una implementación perfecta? (es decir una implementación en la cual se despierta solamente al hilo que tiene que jugar en este instante).
4. Implementa el `pingPONG` entre dos ordenadores.
   - Asume que los IPs estén conocidos antemano.
   - Duplica la salida en los tres ordenadores, cada uno pone un prefijo delante, por ejemplo, `arbitro:`, `jugador rojo:`, y `jugador azul`.

Planificación con prioridades I

Implementa una aplicación con por lo menos tres tipos de procesos/hilos con diferentes prioridades (llamados *A*, *B*, y *C* etc.) que quieren acceder a un recurso compartido.

1. ¿Cómo implementarías el control del planificador para que todos los procesos tengan acceso al recurso con la siguiente forma de justicia: dentro de la misma prioridad el acceso se realiza en orden de pedido y entre los diferentes prioridades se distribuye los accesos para que en los últimos *k* accesos (elegible como variable de configuración del planificador) por lo menos unos *a*% de los accesos son para los procesos de tipo *A*, *b*% para los del tipo *B*, y

Planificación con prioridades II

c % para los del tipo *C* etc.? Obviamente, los porcentages valen solamente si hay procesos de tal tipo esperando en tal momento y su suma no puede superar los 100 %. (Ayuda: un planificador sabe contar).

2. Razona si tu solución garantiza una espera *finita* para todos los procesos pidiendo acceso al recurso.

Planificación con prioridades I

Implementa una aplicación con por lo menos tres tipos de procesos/hilos con diferentes prioridades (llamados *A*, *B*, y *C* etc.) que quieren acceder a un recurso compartido.

1. ¿Cómo implementarías el control del planificador para que todos los procesos tengan acceso al recurso con la siguiente forma de justicia: dentro de la misma prioridad el acceso se realiza en orden de pedido y entre los diferentes prioridades se distribuye los accesos para que en los últimos *k* accesos (elegible como variable de configuración del planificador) por lo menos unos *a*% de los accesos son para los procesos de tipo *A*, *b*% para los del tipo *B*, y

Planificación con prioridades II

c % para los del tipo *C* etc.? Obviamente, los porcentages valen solamente si hay procesos de tal tipo esperando en tal momento y su suma no puede superar los 100 %. (Ayuda: un planificador sabe contar).

2. Razona si tu solución garantiza una espera *finita* para todos los procesos pidiendo acceso al recurso.

Estructuras de datos concurrentes I

1. Preparación:
   - Estudia detenidamente el paquete `java.util.concurrent`.
   - Estudia la implementación de una lista concurrente `http://trevinca.ei.uvigo.es/~formella/doc/cd06/ConcurrentList.tgz`.
2. Usa la lista concurrente para implementar una tabla de dispersión (*hashtable* o *hashmap* en inglés) de la siguiente manera:
   - Existe un array de tamaño fijo que contiene para cada clave (campo en el array) una lista concurrente que almacena a su vez las entradas con dicha clave.

Estructuras de datos concurrentes II

- Implementa por lo menos las funcionalidades: `insert` (se inserte un nuevo objeto en la tabla), `lookup` (se devuelve verdadero si el objeto se encuentra en la tabla, sino falso), y `delete` (se borra un objeto, si existe en la tabla).

3. Implementa un caso de uso de dicha tabla de dispersión lo suficientemente grande para realizar mediciones de tiempo de ejecución.
4. Compara tu implementación con la `ConcurrentHashMap` de Java respecto a tiempo de ejecución, uso de memoria, y lo que consideras interesante.