Presentación

Objetivos

Esta práctica tiene como objetivo el diseño e implementación de una aplicación concurrente multi-hilo utilizando los threads de Java. Para su realización se pondrán en práctica muchos de los conceptos presentados en esta asignatura.

Presentación de la práctica

Hay que presentar los archivos con el código fuente realizado. Toda la codificación se hará exclusivamente en lenguaje Java.

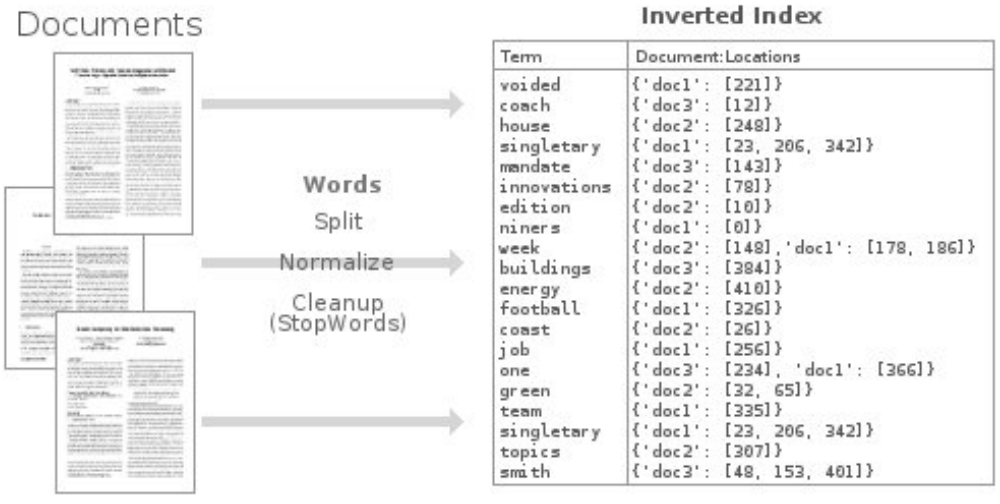
Enunciado de la práctica

El objetivo de la práctica es implementar la versión concurrente de una aplicación de indexación y búsqueda mediante un Índice Invertido.

En informática, un índice invertido (también conocido como archivo invertido) es un índice de base de datos que almacena una asignación del contenido, como palabras o números, a sus ubicaciones en una tabla, o en un documento o un conjunto de documentos. El propósito de un índice invertido es permitir búsquedas rápidas de texto completo, a costa de un mayor tiempo de procesamiento cada vez que se agrega un nuevo documento a la base de datos. El archivo invertido puede ser el archivo de la base de datos en sí, en lugar de su índice.

Hay dos variantes principales de índices invertidos: un índice invertido a nivel de registros (o índice de archivo invertido o simplemente archivo invertido) contiene una lista de referencias a documentos para cada palabra. Un índice invertido a nivel de palabra (o índice invertido completo) contiene adicionalmente las posiciones de cada palabra dentro de un documento. La última variante permite mayor funcionalidad (como búsquedas de frases), pero necesita más potencia de procesamiento y espacio para ser creada.

En la siguiente figura podéis ver un ejemplo de índice invertido para varios documentos, en los que la clave del índice es cada una de las palabras encontradas en el texto y los valores son los documentes y offsets en donde se localizan dichas palabras. A la hora de realizar una búsqueda por una palabra o conjunto de palabras, se accede a las palabras en el índice y se a partir de las ubicaciones se realiza un cálculo del porcentaje de similitud de la consulta y se devuelve la mejor/es.



Un índice invertido esta normalmente compuesto por dos procedimientos/programas. El primero se encarga de construir el índice invertido y el segundo implementa las consultas contra el índice.

La construcción del índice invertido consiste en leer secuencialmente el/los ficheros a indexar e ir añadiendo cada palabra o clave que se encuentre en un hash junto su ubicación (offset) dentro del fichero. Una vez procesado todos los ficheros, el hash resultante (el índice invertido) se puede salvar en disco o en una base de datos.

Para procesar una consulta contra el índice invertido, primero se descompone el texto de la consulta en claves (palabras) y con ellas se accede al índice, obteniendo las posiciones en los ficheros en los que aparecen dichas claves. Con los resultados obtenidos se calcula el porcentaje de similitud con la consulta de cada uno de las posibles ubicaciones y se devuelve la mejor.

En la práctica se os pide que implementéis una versión concurrente de una aplicación secuencial de consultas mediante un índice invertido. La aplicación consta de dos programas Indexing y Query para crear el índice y realizar las consultas respectivamente. En esta versión de índice invertido, las claves son los diferentes substrings de tamaño K (k-palabras) en los que se puede dividir el texto del documento o de la consulta.

En esta práctica, se os recomienda que os centréis la paralelizar la E/S, ya que es la parte que tiene más impacto en el rendimiento de la aplicación. También es posible ejecutar concurrentemente el procesamiento de la consulta, aunque esto último es opcional.

Al programa de Indexación se le pasará el fichero de entrada a indexar, el número de hilos que se deben utilizar durante la indexación y opcionalmente el tamaño de la clave (K) y la ruta del directorio en donde se salvarán los ficheros de índice:

Sintaxis: Indexing <TextFile> <Threads_Number> [<Key_Size>] [<Index_Directory>]

Al programa de consulta se le pasará una cadena con el texto de la consulta, la ruta del directorio que contiene los ficheros de índice, el fichero indexado original y opcionalmente el tamaño de la clave (K):

Sintaxis: Query <Query_String> <Index_Directory> <filename> <Threads_Number> [<Key_Size>]

Funciones involucradas.

En la práctica podéis utilizar las siguientes clases y métodos de java:

- *clase java.lang.Thread*
  - *run()*
  - *start()*
  - *join()*
  - *interrupt()*
  - *isAlive()*
  - *currentThread()*
  - *sleep(long millis)*

Análisis prestaciones

Calcular el tiempo de ejecución de la versión concurrente y compararlo con el de la versión secuencial. Discutir los resultados obtenidos. El objetivo de la práctica es que **la versión concurrente proporcione un mejor rendimiento que la versión secuencial**.

Analizar también, cómo evoluciona el tiempo de ejecución en función del número de hilos utilizados en la aplicación concurrente. Entregar un pequeño informe en donde se muestren (preferiblemente de forma gráfica) y expliquen los resultados obtenidos.

Indicar en el informe las características del hardware en donde habéis obtenido los resultados (especialmente el número de procesadores/núcleos).

Evaluación

La evaluación de la práctica se realizará en función del grado de eficiencia/concurrencia logrado.

Se utilizarán los siguientes criterios a la hora de evaluar las dos implementaciones de la práctica, partiendo de una nota base de 5:

- Aspectos que se evalúan para cada una de las versiones:
  - Ejecución Secuencial (Susp)
  - Ejecución no determinista ([-1.0p,-5.0p])
  - Prestaciones Servidor (-1.0p, +0.5p)
  - No mejora el tiempo de la versión secuencial (-2.0p)
  - Número incorrecto de hilos (-0.5p)
  - Join hilos (-0.5p)
  - Control Errores: cancelación hilos (-0.5p)
  - Condiciones de carrera (-0.5p)
  - No liberar memoria: (-0.25p,-0.5p)
  - Descomposición desbalanceada (Versión óptima) (-0.5p)
  - Descomposición parcial: falta asignar resto división (-0.5p)
- Evaluación del informe de la práctica:
  + Informe completo, con gráficos y conclusiones correctas (+0.25p,+1.0p)
  - Sin informe (-1.0p)
  - Informe con incoherencias en los tiempos sin justificar (-0.5p)
  - Informe: análisis poco riguroso (-0.5p)
  - No se explica el diseño de la solución propuesta (-0.5p)
  - Descripción de los problemas encontrados y como se han solventado.
- Estilo del código.

- Comentarios
- Utilizar una correcta identación
- Descomposición Funcional (Código modular y estructurado)
- Control de errores.
- Test unitarios para verificar la correcta ejecución de la lógica de la aplicación.

Se puede tener en cuenta criterios adicionales en función de la implementación entregada.

Versión Secuencial

Junto con el enunciado se os proporcionamos la versión secuencial en Java para implementar la creación del índice invertido y las consultas. Os adjuntamos un proyecto IntelliJ IDEA con la aplicación.

El proyecto tiene dos clases principales: Indexing y Query, que reciben los mismos parámetros que las versiones concurrentes, si exceptuamos el número de threads. Ambas clases están dentro de un paquete con la siguiente ruta eps.scp.

La sintaxis de la clase principal Indexing que crea el índice invertido es:

Sintaxis: Indexing <TextFile> [<Key_Size>] [<Index_Directory>]

Por ejemplo:

- java -cp Indexing.jar eps.scp.Indexing example2.txt 10 output/example2
- java -cp Indexing.jar eps.scp.Indexing pg2000.txt 10 output/quijote

La sintaxis de la clase principal Query que procesa la consulta:

Sintaxis: Query <Query_String> <Index_Directory> <filename> [<Key_Size>]

Por ejemplo:

- java -cp Indexing.jar eps.scp.Query "123456789012345" output/example2e/ example2.txt 10
- java -cp Indexing.jar eps.scp.Query "En un lugar de la Mancha" output/quijote/ pg2000.txt 10

En el directorio test del proyecto se os adjunta también varios ficheros de prueba de diferentes tamaños para probar la versión concurrente.

Esta versión secuencial, para construir el índice (método BuildIndex), va leyendo el fichero a indexar carácter a carácter hasta que forma una k-word con K caracteres. Entonces, añade esta k-word como clave a un HashMap, siendo su valor asociado la localización (offset) de la k-word en el fichero. Una vez calculado el índice invertido se salva a disco o se imprime por pantalla en función de si el usuario ha especificado el directorio para escribir el índice en disco. Para ello se distribuye las claves entre F ficheros (por defecto, F=1000) y se escribe cada conjunto de claves en un fichero dentro del directorio para el índice especificado por el usuario (el nombre de los ficheros tiene el patrón "*./IndexFileXXXX"*, siendo XXXX el número de fichero). Estos ficheros contienen por cada línea un clave del *HashMap*, junto con sus ubicaciones.

Para realizar una consulta, primero se necesita cargar el índice invertido en memoria, utilizando el método *LoadIndex*. A continuación, se procesa la consulta (método Query),

descomponiéndola primero en k-words, los cuales se utilizan para acceder en el índice invertido y obtener las ubicaciones de cada k-word en el fichero original. Para cada offset de una misma k-work, se le resta su desplazamiento dentro de la cadena de la consulta. Una vez se han obtenido todos las ubicaciones, se calcula la frecuencia que cada offset y se ordenan de mayor a menor. Solo aquellas localizaciones que tenga un porcentaje de matching superior al 80% se muestran por pantalla como resultado de la consulta.

Formato de entrega

MUY IMPORTANTE: La entrega de código que no compile correctamente, implicará suspender TODA la práctica.

No se aceptarán prácticas entregadas fuera de plazo (salvo por razones muy justificadas).

La entrega presencial de esta práctica es obligatoria para todos los miembros del grupo.

Comenzar vuestros programas con los comentarios:

```
/* ------------------------------------------------------------
Práctica 2.
Código fuente : gestor.c
Grau Informàtica
NIF i Nombre completo autor1.
NIF i Nombre completo autor2.
------------------------------------------------------------ */
```

Para presentar la práctica dirigiros al apartado de Actividades del Campus Virtual de la asignatura de Sistemas Concurrentes y Paralelos, ir a la actividad "Práctica 2" y seguid las instrucciones allí indicadas.

Se creará un fichero tar con todos los ficheros fuente de la práctica, con el siguiente comando:

```
$ tar -zcvf prac1.tgz fichero1 fichero2 ...
```

se creará el fichero "prac2.tgz" en donde se habrán empaquetado y comprimido lso ficheros "fichero1", fichero2, y ...

Para extraer la información de un fichero tar se puede utilizar el comando:

```
$ tar -zxvf tot.tgz
```

El nombre del fichero tar tendrá el siguiente formato: "Apellido1Apellido2PRA2.tgz". Los apellidos se escribirán sin acentos. Si hay dos autores, indicar los dos apellidos de cada uno de los autores separados por "_". Por ejemplo, el estudiante "Perico Pirulo Palotes" utilizará el nombre de fichero: `PiruloPalotesPRA1.tgz`

Fecha de entrega

Entrega a través de Sakai el 18 de noviembre, entrega presencial en la clase de grupo medio durante la semana del 18 de noviembre al 22 de noviembre de 2019.