Sistemas Distribuidos

Práctica JavaDFS

Objetivo

- Diseño SFD usando tecnología Java RMI
- Similar SFD de Sprite (*Prot1* en el tema) pero muy simplificado
  - No hay directorios
  - Monousuario
  - Únicas operaciones: *read*|*write*|*seek*
  - Ficheros nunca se borran
  - *write-on-close*
  - Tamaño fichero y de ops. *read*|*write*|*seek* múltiplos tamaño de bloque
    - No tratar con fragmentos de bloque simplifica mucho el sistema
  - Propios de Java: 1 aplicación/JVM
  - Aplicación no concurrente y no abre mismo fichero varias veces
- Servicio con estado: protocolo de coherencia ya lo requiere

Uso de ficheros de acceso aleatorio

```
try {
        RandomAccessFile f = new RandomAccessFile(“fich”, "rw");
        byte[] b = new byte[1024];
        f.read(b);
        f.seek(0);
        f.write(b);
        f.close();
}
catch (FileNotFoundException e) {
        e.printStackTrace();
}
catch (IOException e) {
       e.printStackTrace();
 }
```

Uso de JavaDFS

```
try {
      DFSCliente dfs = new DFSCliente();
      DFSFicheroCliente f = new DFSFicheroCliente(dfs, “fich”, "rw")
       byte[] b = new byte[1024];
       f.read(b);   f.seek(0);  f.write(b);
       f.close();
}
catch (FileNotFoundException e) {
       e.printStackTrace(); }
catch (IOException e) {
      e.printStackTrace();  }
catch (Exception e) {
      e.printStackTrace();  }
```

Desarrollo incremental

- Acceso remoto a ficheros sin cache
- Acceso usando cache pero sin sesiones simultáneas
  - Sólo coherencia entre sesiones sucesivas
- Acceso usando cache con sesiones simultáneas

Acceso remoto a ficheros

- Basado en *RandomAccessFile* en el servidor
- Definición servicios RMI para crear/abrir, *read,write,seek,close*
- Modelo propuesto: fábrica de referencias remotas (ver guía)
  - *DFSServicio*: crea/abre fich. y genera ref remota para su acceso
  - *DFSFicheroServ*: *read, write, seek, close*
- Servicio con estado: toda la funcionalidad en el servidor
  - *DFSFicheroServ*. Estado: fichero abierto, modo y posición
    - 1 por cada open aunque sea del mismo fichero
- Cliente sólo pasarela de peticiones
  - *DFSFicheroCliente*: sus ops sólo llaman servicio remoto asociado
    - Cuidado con read
  - *DFSCliente*: encapsula acceso a *rmiregistry* y a variables entorno
    - Host y puerto del *rmiregistry* en variables de entorno

Acceso con cache restringida (1/2)

- Cache LRU de bloques ya programada (*LinkedHashMap*)
  - Con escritura diferida y *write-on-close* (≠ Sprite)
- Una cache en cliente por fichero accedido en algún momento
  - *DFSCliente*: estructura de datos relaciona fichero y su cache
  - *DFSFicheroCliente*: crea cache para fichero si no existe
- Lectura en cliente; por cada bloque consulta cache (*get*):
  - Si no en cache, se pide a servidor y se copia en cache (*put*)
  - Copia bloque de cache al *buffer* del *read*
- Escritura en cliente; por cada bloque:
  - Lo copia en cache (*put*) y lo marca como modificado
- Escrituras al servidor: bloques modificados en *close* y
  - Operación *put* expulsa bloque modificado

Acceso con cache restringida (2/2)

- Cambio radical en relación cliente/servidor
  - Cliente debe conocer/gestionar posición en el fichero
  - *read*|*write* al servidor incluyen posición
    - no se requiere *seek* en servidor
- Protocolo de coherencia entre sesiones:
  - Guardar en cliente fecha última modificación asociada a cache
  - Al empezar nueva sesión comprobar fecha modificación real
    - Si difieren eliminar bloques en cache

Acceso con cache sin restricciones

- Repaso protocolo de SFD Sprite
- Protocolo requiere conocer clientes lectores/escritores de fich
  - 1 *DFSFicheroServ*/f.abierto aunque usado por varios clientes
- Y requiere poder solicitarles volcados e invalidaciones
  - Esquema propuesto: Callbacks (véase guía)
  - Cliente envía objeto de tipo *callback* al abrir fichero
- *DFSServicio* gestiona estructura de datos que:
  - Relaciona: nombre de fichero → *DFSFicheroServ*
  - Por cada fichero: lectores/escritores y sus *callbacks*