15640 Project 2: File Caching Proxy

1. Cache Consistency

   In this project, I use "Check on Use" as cache consistency policy.

   When the client open a file on proxy, it will first check the file version on server side and determine whether the local version is up-to-date or not. If not up-to-date, the proxy need to fetch the file from the server to cache.

   When the client close a file. If the file operation is not read only, the proxy will send the local version to server for update.

   Open-close semantics. To ensure the file opened will not be modified by other threads between open and close, we need to make copies of the original file and read/write on them. After close, if not read only, we need to delete the original file and rename the current copy to non-copy version.

   Copy and non-copy files: how to determine the path of copy file and non-copy file? The path of copy files are combined with the path of original file + file descriptor.

2. Messaging between Proxy and Server

   The messaging between proxy and server include:

   1) Proxy send request to server with file path and server return response of the last modified time
   2) Proxy read file content from server: Proxy send request, server reads the file and send back chunk of file contents. If the file size is too large, use loop to receive.

      Combine open with read on server side, if the reading start position is 0, open first.
   3) After close, if not read only, proxy send the current file to server. Use loop to send chunks of file contents to server, and server write the chunks to the file sequentially.
   4) Proxy unlink file: proxy send request to server and server delete the file within server storage.

   In a regular open-read-close session, if the cache version is up-to-date, the massaging between proxy and server only includes sending request to server to get the last modified time of the file on server side. Only one round trip is demanded. If the local version is out-of-date, the massaging includes get last modified time from server and read chunks of data from server. The times of transmission will be dependent to chunk size. If one chunk is enough to read file from server, then two round trips are needed.

   In open-read/write-close session, except above processes, after close local file, proxy needs to send chunks of the local file back to server.

3. Chunk Policy

   Define max chunk size equal to 100000,

   then use `int chunkSize = Math.min(cacheSize/10, MAXCHUNKSIZE);`

   to obtain the chunk size. It can fit the situation of a small cache size.

4. LRU Implementation

   Use a linked list representation of LRU. In the linked list, from head to tail, it is ranked from latest-used to least-used. If there has demand of cache eviction, it will evict from the tail of the linked list. Since there may be multiple copies of the same file. When open or close a file, move the non-copy version file to the first of linked list.

5. File Operations

   1) Open

First, when a client want to open a file, the proxy will first send a request to server and receive the latest modified time of such file on server side, which will then be used to compared with the modified time of the file in proxy cache.

After comparation,

a. If up-to-date, see whether the open operation is READ which means read only.
   If read only, get the last read copy in the cache. If last copy is latest as well, read count + 1 on that copy. If not, create a new read copy.
   If not read only, create a new rw copy of that file.
b. If the local cache has no such file or if the local version is out-of-date, open and read the file content from server to local cache. Create a new copy of that file.

LRU eviction:

a. When get the file content from server, if no old-version file exists, evict if demand to free up the new file size. If old file exists, evict if demand to free up new file size – old file size.
b. When create a new copy, if not read only, evict if demand to free up the size of the new copy in the cache. If read only, but cache need to evict for the new read copy, then rename the current non-copy version to copy version and read on that (read count++), remove the non-copy version from LRU list. It means it will evict it's own non-copy version file.
   If no eviction demand, create a new copy as normal.
c. After open, the non-copy version file in LRU list will be removed to first. (exclude the situation of read-only and need eviction)

2) Close
   a. If not read only, we need to send the current version to server for update.
      If the non-copy version exists, delete the non-copy version, rename the current copy to non-copy, move the corresponding block to the first of LRU list.
      If not (has been evicted), rename the current copy to non-copy, add the new non-copy to the first of LRU list.
   b. If read only, check the read count.
      If read count - 1 is equal to 0, and if the non-copy version exists, delete the current copy, and move non-copy version to the first of LRU list. If non-copy version not exist (evicted), rename the copy to non-copy version, add to the first of LRU.
      If read count - 1 is not equal to 0, read count -= 1.
3) Read/write/lseek
   Read/write/lseek on the copies in the cache.
   Write: Since writing may enlarge the file size, keep the cache size limit while writing is important. First, identify the new file size after writing and the increased size will be equal to new file size minus old file size. If the cache doesn't have enough space for it, eviction will be demanded.