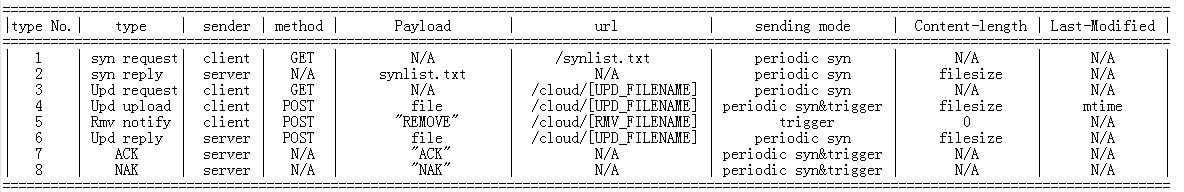
control thread (server side)

Create thread

Accept()

Wake up worker

Closesocket()

Bind()

Closesocket()

Accept()

Create thread

Bind()

Client side (monitor thread)

Monitor folder

File added /deleted /modified

Recv reply

Yes

Recv reply

Is updating == 1?

no

Yes

Send packet #5

Is it a remove msg?

no

Send packet #4

Client side (periodic synchronization thread)

Connect()

Connect() again for monitor thread

yes

has cookie?

no

read it.

Set updating = 1

Send packet #1

recv packet #2

Generate local list

Process synlist.txt, upload/delete/modified local files & handle conflict files

Set updating = 0

yes

First syn?

no

Create monitor thread

Sleep for 10 seconds

Worker threads (server side)

Sleep(100)

no

Send packet #8

Yes

More than 100 tries?

Yes

no

Successful?

Update requested file

Yes or not exist

yes

no

Is it newer?

Send packet #7

Sleep(100)

>0

0

no

no

Successful?

remove requested file

yes

Is packet #5?

0

0

Content-length?

Send packet #2

yes

No

Set&save cookie

cookie?

yes

Is it a POST?

Ignore it.

no

no

Is it a GET?

Recv()

yes

yes

Is packet #1?

Send packet #6

Protocol design and message exchanges:

|  |  |  |
| --- | --- | --- |
| Operation | When update on server | When update on other clients |
| Add | Immediately | Next synchronization |
| Modify | Immediately | Next synchronization |
| Delete | Immediately | Next synchronization |

**Client side:**

Client side is divided into two parts; one is monitor part while the other is periodic synchronization part. I will explain them in detail later.

1. **Monitor thread**

Monitor thread keeps an eye on the cloud folder and takes corresponding actions when there are changes in the monitored folder. When there’s file adding, modification or deletion, this thread will first check if it’s due to the periodic synchronization. If so, then it should do nothing but ignore this message and continue monitoring. If not, then an instant notification packet will be sent to the server immediately. The methods of packets sent by this thread are all POST.

The notification packet can be classified into two classes:

1. Update notification packet (packet type 4)

This packet is triggered when an addition or modification operation is detected. It contains the filename (specified in URL), length of the file (specified in Content-Length) together with the file itself to be the payload.

1. Remove notification packet (packet type 5)

This packet is triggered when a deletion operation is detected. It contains the deleted filename (specified in URL). It has 0 in Content-Length option, but also has payload (string “REMOVE”). This design is used to distinguish remove notification packet from an update notification packet with 0-size file (in this situation the payload is empty).

After sending out these packets, monitor will wait an ACK or NAK reply from the server, and after which, it will back to its normal routine.

1. **Periodic synchronization thread**

Periodic synchronization thread is created when user chooses to enter the client mode. It first connects to the server twice. One socket is occupied by itself; the other one is used by monitor thread, which avoiding data corruption if socket is shared between two different threads. Then it checks if it had received cookie from server before. If so it will fetch the cookie and insert it into every packet it sends out afterwards.

Next, it sets a flag (updating) to 1 which actually disables the monitor thread, and begins the first synchronization.

Synchronization includes the following step:

1. Send out synchronization request packet (packet type 1) attached with the cookie, if any.
2. Receive synlist.txt file from the server (packet type 2). This list contains the record of the fileinfo on server side. Each record has four parts: file name, file length, the last-modified time of the file (expressed using the total UNIX seconds) and the hash value.
3. Client then similarly generates a list of local file and then compare these two lists. It will mark some file (using member flag) for further operation.

**Noted that only when there’s no cookie on client will it post the files back which only exist on client side. Otherwise it will delete them.**

|  |  |
| --- | --- |
| Flag | condition |
| 0 | 1. **No cookie before (new client logins in) and file only exists on client side.** 2. File exists on both sides but contents are different and client has a newer copy. |
| 1 | 1. File only exists on server side. 2. File exists on both sides but contents are different and server has a newer copy. 3. File can be updated. |
| 2 | 1. File only exists on server side. 2. File exists on both sides but content is different and server has a newer copy. 3. File is occupied by another process thus can’t be updated. |
| 3 | 1. File exists on both sides and the hash value is consistent |
| 4 | 1. **Client has a cookie and file only exists on client side.** |

1. Then the thread retrieves local list and does the following operation accordion to the value of flag.

|  |  |
| --- | --- |
| Flag | Operation |
| 0 | Upload this file to server |
| 1 | Download this file to the client |
| 2 | Download this file to the client and rename due to confliction. Format: Conflict\_[filename] |
| 3 | Remain unchanged |
| 4 | Remove local file |

1. The first synchronization is done. Then the thread resets the updating flag to 0 and creates monitor thread. Last, the thread will sleep for 10 seconds and restart the next synchronization when it wakes up.

**Server side:**

1. **Master thread**

This thread is very simple: it just listens at port 80 and creates a new worker thread when there’s an incoming connection.

1. **Worker thread**

Firstly and most important thing is that, in my design, **server is totally passive, which means it will never initiatively contact with any client. All it does is to receive packet and respond correspondingly.**

1. Receiving synchronization request packet (packet type 1)

On receiving this kind of packet, the server will first check if this packet has cookie inside. If not it will generate a random cookie. Then server will send out the synchronization reply packet (packet type 2) together with the recently generated cookie, if any.

1. Receiving updating request packet (packet type 3).

On receiving this kind of packet, the server will first check if the requested file is really exist on server side. If so it will send this file using updating reply packet (packet type 6). Otherwise, only NAK (packet type 8) is sent out.

1. Receiving updating upload packet (packet type 4).

On receiving this kind of packet, the server will first check if the specified file is really exist on server side. If not then it creates a new file and fill it with what it just received, after which it sends an ACK (packet type 6) to indicate the operation is successfully executed. If so, the server will compare and see which one is newer. If server side is newer then it does nothing but send ACK (packet type 6) back, otherwise it will try to update the file (if file can’t be opened then it will sleep for 100ms and try again). After successfully updating it will send ACK (packet type 6) out. NAK (packet type 8) is sent when failure times reaches 100.

1. Receiving removing request packet (packet type 6).

On receiving this kind of packet, the server will first check if the requested file is really exist on server side. If so it will try to delete the file and send ACK (packet type 6). Otherwise, only NAK (packet type 8) is sent out. Also NAK (packet type 8) is sent when failure times reaches 100.

After one packet has been processed the server will return to the beginning, waiting to receive the next packet.