Asgn3: Design Document

Alexander Soe cruzid: asoe

CSE130, Fall 2019

Goal

The goal of this assignment is to add caching to our single threaded server. This means that if a file has been recently requested or put into our server, its file name and contents should be held in a cache stored in the memory of the program. Then if another client requests something in the cache, the server can just write directly from the cache to the client rather than reading from disk and then writing. If the cache goes beyond its max size, then the first file to be cached will be removed and then written to disk. Additionally, if anything from disk is requested by the client and there is no instance of that file in the cache, then the cache should be updated accordingly.

Assumptions

I'm assuming that the cache has to be a global cache as all functions need to have access to the cache. This is because both GET and PUT requests manipulate the cache in some way by either removing, or adding elements to the cache. Also the logging functions have to be changed such that they can tell you the presence of certain files in or not in the cache. This is going to change how the headers are going to look and it's also going to affect how you calculate the offset for the logfile. Lastly, there is going to need to be a way to check whether the size of the cache is no greater than size 4 as well as check if any of the files need to be updated in the cache. If an element is being added to the cache and the cache is already of size 4, then the first element that was added to the list will be removed.

Design

Since we are using our previous assignment, all the necessary things to set up the http server using sockets will be predominantly the same. Additionally the way the clients are handled will also be the same as prior assignments. Unlike the last assignment however, the multithreading option will be removed as there will be many added critical sections when the cache is implemented due to the shared resource of the cache. This will be done by changing the default thread limit to 1.

To create the cache I'm going to be using two deques: one for storing the actual file contents and the other to store the file names. Since the max size of the deques is going to be 4 there is no need to use a more complicated data structure such as an unordered map as the size is so small. Also the deque has the added benefit of keeping track of what order each element is added to queue as I'm trying to implement a FIFO cache where the first element the cache is the first one to get popped out. The key to making sure that the file names and file contents are matching is that whenever something is popped or pushed from deque, the other deque must also pop or push in tandem. That way each index of one deque should be connected to that same index in the other deque.

To deal with saving to the cache I'm going to be using C++ strings. When I'm reading from the client for a PUT request, I'm going to use a string constructor to construct a C++ string form

every buffer read in from the client. Then I'm going to concatenate all these buffers into a huge string and store it in my deck. When I get a GET request and the file does exist on the disk but does not exist in the cache, I'm going to read from the file into a char buffer and then again use a string constructor to create a string from the buffer which I will concat with all the other buffers to get the file contents that will need to be stored in the deque. When a client is trying to get anything from the cache I will check the cache for the file, and if it is there, I will write the entire string in one go. I will have flags everywhere to check if the "-c" option was typed such that all the cache manipulation will be ignored and the server will work as before writing to and from disk.
As for the logging, I'm going to create two global char messages that state whether the contents are in or not in the cache and then just add those to the headers and adjust the offset as such. Errors will be handled as they usually are as the cache will react essentially the same. The only thing to watch out for is that the cache must always be checked first for both PUT and GET requests when checking for errors or in general.

Pseudocode

```
String getFromCache(string fileName)
{
        for(loop through the cache)
        {
                Compare file names
                Return cache contents
        }
}

Void writeFromCache(string fileName, string fileContents)
{
        open(file);
        write(file)
}

Void cacheFile(string fileName, string fileContents)
{
        if(updateCache() == true)
        {
                return;
        }
        if(cacheSize == 4)
        {
                Pop off first element in queues
                Write old element to disk
                Push back new element
```

```
        }
        Else
        {
                Push back new element
        }
}

Struct context
{
        Pthread_cond_t cond;
        Pthread_mutex_t lock;
        Queue clientQueue;
}

//Create a lock to stop other threads from stealing each others space in the logfile
Size_t getOffSet(size_t logLength)
{
        lock(mutex);
        currLogPos  = offset;
        offSet +=  logLength;
        unlock(mutex);
        Return currLogPos;
}



Size_t printLogHeader(char fileName, size_t contLength)
{
        //Calculate the total offset using the ceiling function
        //Write "========\n" into the the last 9 spots in the offset for the logfile
        //Write the PUT header message
}

String convertCharToHex(char c)
{
        hex[3];
        sprintf(hex,  " %02X", c);
        String hexString = hex;
        Return hexString;
}

String convertLineNumber()
{
        //Same as convertCharToHex but with the lineNumbers
}
```

```
Size_t printPUTLog(size_t totalReadSize, size_t readSize, char fileContents, size_t startPosition)
{
	Size_t newPosition = startPosition;
	for(int i = 0…..)
	{
		if(totalReadSize % 20 == 0)
		{
			pwrite(convertLinenumber);
			startPosition += convertLineNumber.length();
		}
		pwrite(convertCharToHex);
		startPosition += converCharToHex.length();
		if(totalReadSize % 20 == 19)
		{
			pwrite(“\n”);
			startPosition++;
		}
	}
	Return newPosition;
}

Void printGetLog(char fileName)
{
	//print the getLog header
}
Void pringErrorLog(char fileName)
{
	//print error Log
}

String readHeader(int fd)
{
	while(read(fd) > 0)
	{
		if(strstr(“\r\n\r\n”) != nullptr)//check for end of header
		{
			break;
		}
	}
	if(n < 0)
	{
		close(fd);
		Return string
	}
```

```
        Return string
}

Int getContentLength(string header)
{
        strstr("Content-Length:") //find content length keyword
        if(Content-length is there)
        {
                sscanf("Content-Length: %d")
                If(contentLength > 0)
                {
                        Return length;
                }
        }
        Else
        {
                Return error
        }
        Return -1;
}

Bool isCorrectFileName(string filename)
{
        if (filename is greater than or less than 27)
        {
                Return false;
        }
        if(characters are the right ascii values)
        {
                Return false;
        }
        Return true;
}

String getFileName(string header)
{
        Vector tokenVector;
        Stringstream tokStream;
        String token;
        while(getline(tokstream))
        {
                tokenVector.push_back(token);
        }
        String filename = tokenVector[1];
        Return filename;
```

```
}

Void handlPut(filename, contentLength, client_fd)
{
	if(access(file access))
	{
		if (access(write access))
		{
			Write 403 response
		}
	}
	if(fileExists and there is contentlength)
	{
		Write 200 response
	}
	if(!fileExits and contentLength)
	{
		Write 201 response
	}


	if(contLength != -1)
	{
		while(contLength > 0)
		{
			read(client);
			printPutLog(...);
			String fileContents = buffer;
		}
	}
	Concatenate all the strings
	Else
	{
		read(client);
		printPutLog(...);
		write(till eof)
	}
	Cache the string and file name
}

Void handleGet(filename, client_fd)
{
	printGETLog(...);
	if (access(file exists))
	{
```

```
                Write 404;
        }
        if (access(read ok))
        {
                Write 403;
        }
        write(client_fd, header length);
        while(contentlength != 0)
        {
                Read(fileContents);
                write(client_fd);
        }
}

handle()
{
        while(there is a header)
        {
                //Call get
                //Call put
                //Check for errors
        }
}

Int main()
{
        //Do all the socket setup stuff
        //Call all the file
        while(1)
        {
                while(clientfd = accept >0)
                {
                        lock(mutex);
                        push(client fd);
                        Cond_signal;
                        unlock(mutex);
                }
        }
        Return 0;
}
```