Programming Assignment 3:
Improving Performance with I/O Multiplexing

Part 1: November 12 (11:59 PM Hard deadline),

Part 2: November 19 (11:59 PM Hard deadline)

1. Introduction

The performance is one of the most critical points of high-end server applications. To improve the performance, modern operating systems provide interfaces to multiplex I/O between multiple file descriptors. The select that we learned in the class is the one example of them that provide I/O multiplexing. Popular web servers such as nginx and lighttpd also employ this approach and optimize their performance with event systems such as POSIX AIO, libevent, or epoll/kqueue.

In this assignment, you will improve the performance of the proxy that you have implemented in the previous assignments. You will change your threaded proxy to event-driven proxy using popular event interfaces, such as libevent. You will learn to use buffered I/O and completion-based event model. You are required to measure the performance of your proxy to compare it with the threaded one.

2. Part 1: Proxy with libevent

The previous approach was creating a new thread for new incoming connection to handle multiple connections at the same time. However, when there are thousands of concurrent connections, the cost for creating new threads or switching context between different threads becomes significant. In addition, since there could be concurrency issue on multiple threads, they need to use lock to protect some shared variables, which may degrade the performance of the applications. Due to that, modern high-performance web servers (e.g., nginx, lighttpd, …) take event-driven approach rather than threading.

In this part, you will work on libevent to implement your proxy. As learned in the class, libevent is a callback-based event interface that is based on event mechanisms such as epoll, kqueue,

select, and poll. It allows users to develop event-based applications with intuitive API without knowing the underlying layer. For example, applications can register an event and corresponding callback function with the event_set() and event_add(). Similarly, the events can easily be removed by event_del(). Event-driven applications mostly wait for the events using a loop. With libevent, you can simply call event_dispatch() for such event loop.

You will now use this to enable I/O multiplexing on your proxy. The listening socket will be registered to the libevent to get the notification of new incoming connections. After accepting the connection, the fd for it will be registered to wait for HTTP request. Later, new connection to the origin server will be made and you will wait for the write event. When write event comes, it means that the connection has been made and you can write data to the socket. After sending request to the server, you will wait for the response with read event. Finally, you will relay the response received from the server to the client. When sending response to the client, if you could not send all data at once due to limited TCP buffer, you should register write event and retry again after the write event comes.

Be sure to use nonblocking sockets when you are programming with libevent. (You can simply use fcntl() to make it nonblocking.) If not, your proxy could be blocked for the socket operations such as connect(), accept(), read(), and write().

Part 1: Implementing Basic Functionality and cache control

As we did in the Part 1, for this first part, you are required to test the basic functionality of your libevent-based proxy. We will test your proxy with several websites using modern web browsers. We will test several websites such as http://ina.kaist.ac.kr, http://www.yahoo.com, and http://www.naver.com. If you pass this functionality test, you will get 20 points in this part. In the 20 points, handling large file will be also included (5 points). Since you are required to do buffer management by yourself for large file download, we will explicitly test your proxy with large files that are basically larger than 10s MB.

HTTP/1.0 provides a simple caching mechanism. An origin server may mark a response, using the Expires header, with a time until which a cache could return the response without violating semantic transparency. Further, a cache may check the current validity of a response using what is known as a conditional request: it may include an If-Modified-Since header in a request

for the resource, specifying the value given in the cached response's Last-Modified header. The server may then either respond with a 304 (Not Modified) status code, implying that the cache entry is valid, or it may send a normal 200 (OK) response to replace the cache entry. HTTP/1.0 also includes a mechanism, the Pragma: no-cache header, for the client to indicate that a request should not be satisfied from a cache.

In this assignment, you need consider this cache control. Among the 20 points for the Part 1-1, this cache control takes 5 points. You need to concern about the “Expires” header to make the cache object expire after the expiration time. You also need to check the “Pragma” field. If it is set to “no-cache”, you should not cache the object. In your logging, add the cache entry as below.

Date: browserIP URL size cache-hit

where browserIP is the IP address of the browser, URL is the URL asked for, size is the size in bytes of the object that was returned, cache-hit is whether the object is “Hit”, “Miss”, or “No-cache”. For instance:

Sun 27 Oct 2002 02:51:02 EST: 128.2.111.38 http://www.cs.cmu.edu/ 34314 Miss
Sun 27 Oct 2002 02:51:02 EST: 128.2.111.38 http://www.cs.cmu.edu/ 34314 Hit
Sun 27 Oct 2002 02:51:02 EST: 128.2.111.38 http://www.cs.cmu.edu/no-cache.html 34314 No-cache

Part 2: Performance & Robustness Testing

Another 20 points will be given for the robustness and performance of your libevent proxy. We are planning to evaluate the following scenario. However, the testing scenario is not fixed. We recommend you to test your proxy on various conditions.

(1) ab with 1 concurrency 1,000 requests (10 points)
$ ab -n 1000 -X proxy_ip:proxy_port http://ina.kaist.ac.kr/~dongsuh/

(2) ab with 100 concurrency, 10,000 requests (10 points)
$ ab -c 100 -n 10000 -X proxy_ip:proxy_port http://ina.kaist.ac.kr/~dongsuh/

Increasing the concurrency can make you have connection reset from the remote server. If it seems that your proxy is refusing the connection, increase the listen backlog size to enable queueing more incoming connections. (Typically, 1000 would be enough for the second parameter of listen()). If this is not the case, the server is refusing the connection and there is no way to avoid it. As we tested, it had no problem when we use concurrency equal or less than 100.

The ab will print out the result of the benchmark. There are requests per second (#/sec) and transfer rate recorded. You will use these values to compare the performance between the different version of proxies. In your report, you need to contain the result in the following cases: (1) without proxy (direct testing to the server), (2) with threaded proxy, (3) with proxy (AIO or buffered event), and (4) with libevent proxy. This report is total 10 points. The 10 points include how much performance improvement you have with I/O multiplexing.

3. Extra-credit: (20pt) Due on the same day as part 2.

Implement a proxy using buffered event in the libevent library.

http://www.wangafu.net/~nickm/libevent-book/Ref6_bufferevent.html

4. Evaluation

(1) Part 1: Implementing basic functionality (20 pt)
- Testing simple web browsing (10 pt)
- Cache control (5 pt)
- Supporting large file transfer (5 pt)

(2) Part 2: Performance & robustness testing (20 pt)
- Testing with ab as 1 concurrency (10 pt)
- Testing with ab as many concurrency (10 pt)

(3) Report: Performance evaluation (10 pt)

5. Handin Instructions

Use your SVN repository. Handin directories are:

Part 1: tags/PA3/Part1

Part 2: tags/PA3/Part2

Extra credit: tags/PA3/Extra

6. References

Linux Programmer's Manual - AIO(7) ($ man aio)

Caching in HTTP/1.0 http://www8.org/w8-papers/5c-protocols/key/key.html

ApachBench (ab) http://httpd.apache.org/docs/2.2/programs/ab.html

Libevent http://libevent.org, http://www.wangafu.net/~nickm/libevent-book/

Buffered event http://www.wangafu.net/~nickm/libevent-book/Ref6_bufferevent.html

Notes about using ab:

In this part, you will make the proxy robust when there are many (100s ~ 1000s) concurrent connections. We will also see how your proxy improves the performance in this part. The ApacheBench (ab) will be used to evaluate the performance. ab generates 'n' HTTP requests while having 'c' concurrent connections that downloads a same file at the same time. You can specify arguments for the url to download, number of concurrency (-c option), number of total requests (-n option), and the proxy server address and port (-X option, address:port). For more detail, please enter the command "ab" to your shell to see its usage.

Again, your proxy should not suspend while doing the benchmark. After finishing benchmark, ab will print how many transactions were processed per second and whether the content of the file is legitimate. Your proxy should not print any errors in here. We will deduct points if there are errors found.

If the first part of your assignment was successful, you may start this part with basic following command. Since ab downloads same file over multiple connections, they will be served from the cache of the proxy, not from the actual server. If the cache is not implemented correctly, the performance may go down. While experiment, use remote proxy address (not localhost) so that your requests to be sent to the network.

(1) ab with 1 concurrency 1,000 requests

$ ab -n 1000 -X proxy_ip:proxy_port http://ina.kaist.ac.kr/~dongsuh/

If this evaluation is successful, we will give you 10 points. After this, move on to the more comprehensive test as following.

(2) ab with 100 concurrency, 10,000 requests

$ ab -c 100 -n 10000 -X proxy_ip:proxy_port http://ina.kaist.ac.kr/~dongsuh/

The ab will print out the result of the benchmark. There are requests per second (#/sec) and transfer rate recorded. You will use these values to compare the performance between the different version of proxies. In your report, you need to contain the result in the following cases: (1) without proxy (direct testing to the server), (2) with proxy (threaded), and (3) with your enhanced proxy. This report is total 10 points.