18S2 COMP3331 Lab02

Haibo Wang

z5135009

Exercise 3:

Q1:

The status code is **200**, response **phrase** is OK.

Q2:

Last-Modified: Tue, 23 Sep 2003 05:29:00 GMT. and it also contains a header filenamed date.

Date is when the resource was originated.

Last modified date is that the origin sever believes the resource was last modified/accessed in that day.

Q3:

The connection is persistent, because the protocol is HTTP1.1, which persistents with pipelining.

Q4:

73 bytes.

Q5:

Text and some html code.

Exercise 4:

Q1:

No.

Q2:

No.

Q3:

Yes,

Last modified time: **Tue, 23 Sep 2003 05:35:00 GMT**

IF-NONE-MATCH: **1bfef-173-8f4ae900**

Q4

Status code: 304

Phase: Not Modified

No, in client sevrver the source's cache copy has up-to-date.

Q5:

e-tag: **1bfef-173-8f4ae900**, it is an identifier for a specific version of resource, it allows caches to be more efficient, and saves bandwidth, as a web server does not need to send a full response if the content has not changed. .

It hasn't changed.

Exercise 5:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/timeb.h>
#include <errno.h>

#define PACKETS 10

double min(double *a, int len);
double max(double *a, int len);
double average(double *a, int len);

int main(int argc, char* argv[]){
    // criteria to filter the correct sockaddr
    struct addrinfo hints;

    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_INET; // use IPv4
    hints.ai_socktype = SOCK_DGRAM; // use UDP

    // store the result addrinfo list
    struct addrinfo *res;

    if(getaddrinfo(argv[1], argv[2], &hints, &res) != 0){
        perror("getaddrinfo");
        exit(1);
    }

    int s;
    // create socket
    if((s = socket(AF_INET, SOCK_DGRAM,0)) < 0){
        perror("socket");
        exit(1);
    }

    // set receive timeout
    struct timeval timeout;
    memset(&timeout, 0, sizeof(timeout));
    timeout.tv_sec = 1;
    setsockopt(s, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));

    char *ip = inet_ntoa(((struct sockaddr_in *)(res->ai_addr))->sin_addr);

    char buf[100];
    int i;
    struct timeval start, end;
    int count = 0;
    double nums[10];
    for(i = 0; i < PACKETS; i++){
        // get current time
        gettimeofday(&start, NULL);
        double t1 = start.tv_sec*1000LL + start.tv_usec/1000;
        // send packet
        sprintf(buf, "PING %d %.0lf\r\n", i+1, t1);
        if(sendto(s, buf, strlen(buf), 0, res->ai_addr, res->ai_addrlen) < 0){
            perror(NULL);
            exit(1);
        }
        // receive packet
        receive:
        if(recv(s, buf, sizeof(buf), 0) != -1){
            int seqNum;
            sscanf(buf, "PING %d %lf\r\n", &seqNum, &t1);
            if(seqNum == i+1){
                // get current time and cal delay
                gettimeofday(&end, NULL);
                double t2 = end.tv_sec*1000LL + end.tv_usec/1000;
                double rtt = t2 - t1;
                nums[count++] = rtt;
                printf("ping to %s, seq = %d, rtt = %.0lf ms\n", ip, i+1, rtt);
            } else{
                goto receive;
            }
            sleep(1);
        // timeout
        } else if(errno == EAGAIN || errno == EWOULDBLOCK){
            printf("ping to %s, seq = %d, timeout!!!!\n", ip, i+1);
        } else{
            perror(NULL);
            exit(1);
        }
    }
    double small, large, ave;
    small = min(nums, count);
    large = max(nums, count);
    ave = average(nums, count);
    printf("rtt min/avg/max = %.0lf/%.0lf/%.0lf ms\n", small, ave, large);
    close(s);
    return 0;
}

double min(double *a, int len){
    int i;
    double small = a[0];
    for(i = 1; i < len; i++){
        if(small > a[i]){
            small = a[i];
        }
    }
    return small;
}

double max(double *a, int len){
    int i;
    double large = a[0];
    for(i = 1; i < len; i++){
        if(large < a[i]){
            large = a[i];
        }
    }
    return large;
}

double average(double *a, int len){
    double sum = 0;
    int i;
    for(i = 0; i < len; i++)
        sum += a[i];
    return sum / len;

}
```