Practices on Computer Networks and Distributed Systems

Wall clock time and Distributed Systems

All rights reserved © 2015-21 by José María Foces Morán & José María Foces Vivancos

1. Sending and receiving ICMP timestamps in C with the Raw Sockets API

Raw Sockets allow programmatic access to the IP layer. In this practical we use Raw Sockets to write a program that sends and receives specific types of ICMP messages with the intention of retrieving the time of of day of another host that will ultimately allow it to have its clock synchronized. Synchronizing a clock entails being able to speed it up and to slow it down. The POSIX-compatible function that in Linux allows the clock to be sped up/slowed down in order for the clock to be synchronized is adjtime().

Exercise 1.

a. Check the type of clock synchronization client that is in use in your Linux

\$ systemctl status systemd-timesyncd

If your system is using the full fledged Linux NTP client, follow the relevant instructions in section 2, below.

- b. Assuming that your system is using systemd-timesyncd, read the resulting listing and confirm the IP address of the time server that your system was using until now. Check that the transport protocol is UDP and that the associated port is 123.
- c. Stop the NTP client so that the time synchronization program we wish to build doesn't conflict with it in taming the clock:
 - \$ systemctl stop systemd-timesyncd
- d. Print the current system's daytime with a μ s precision:
 - \$ date --rfc-3339=ns
- e. Set the clock time 2 minutes later than the current time; see this example:

\$ date -s '2021-10-22 09:20:00'

f. Our example time synchronization program is based on the ICMP protocol's Timestamp request and Timestamp response messages. In order for our program to send ICMP Timestamp request message, we will open a Raw Socket upon our stack's IP protocol which will allows to craft one such

message and later have it sent over the socket towards the Internet host that we want to use as reference for our local clock.

Download the raw sock icmptimestamp.c program from paloalto.junileon.es/ds/lab/icmptimestamp.c and test it. The program should contact the host which IP you entered into the program and, if the ICMP Timestamp request makes it to that host and the response is successfully delivered to our local host, then, the program will calculate the amount of µs necessary to set your local clock in sync with the remote system's clock. The process of updating the local time is achieved by calling Linux function adjtime(). That function will speed up or slow down the clock to achieve the synchronization without ever setting the clock backwards, which is absolutely forbidden. Run the program once and observe that in a few minutes of slowing it down, it will reach the target time. Use the following example as orientation:

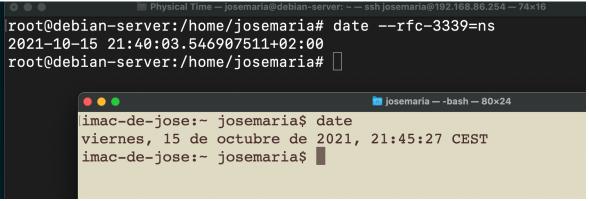


Fig. 1. The Linux clock is about 5 min behind the clock of the OS-X system

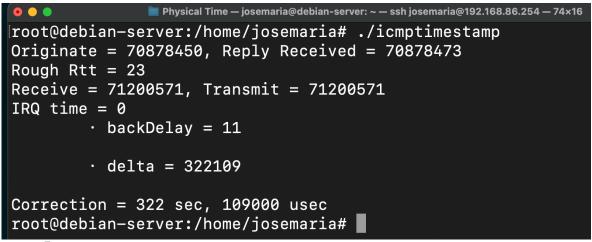


Fig. 2. The icmptimestamp program has sent an ICMP Timestamp request message to 193.146.101.46 and has received an ICMP Timestamp reply. The local clock is now being sped up to reach the clock time of that host.

v.1.6 - Oct 2021

\$ sudo su

./icmptimestamp

The instructor will provide you a guidelines to complete this exercise.

Exercise 2. Have icmptimestamp program emulate the behavior of the Christian's algorithm by repeatedly retrieving the time from another system, then calculating the average and finally *properly* using adjtime() for adjusting the clock. You will observe that repeated calls to the program time retrieving function fail, then, we will have to solve this problem by modifying the program so that it runs fine in repeated calls? Wireshark will be able to report to you the error condition that you will have to discover and which prevents the aforementioned repeated calls from functioning all right (Make sure you *sudo* the resulting program, for, Raw Sock programs need superuser privileges to be executed).

1. If using OS-X, you need to enable the following property:

\$ sysctl net | fgrep icmp | fgrep times
net.inet.icmp.timestamp: 0

2. Enable it:

```
$ sudo sysctl -w net.inet.icmp.timestamp=1
Password:
net.inet.icmp.timestamp: 0 -> 1
```

3. Check it:

\$ sysctl net | fgrep icmp | fgrep times
net.inet.icmp.timestamp: 1

4. Start Wireshark, set filter 'icmp'

5. Compile the icmp timestamp requester program (gcc ...: Ok) and run under Linux and have the program contact the MacBook Air at 192.168.2.? that we just reconfigured:

\$ sudo testicmpts 192.168.2.106

6. Monitor the icmp traffic with WS and check the program's output

2. Querying the state of a Unix/Linux NTP Client

As we reviewed in the Lectures, the Internet protocol used today for synchronizing clocks is NTP (Network Time Protocol), we can query its basic parameters by issuing the OS-X and Linux command ntpq and can estimate the Rtt with our NTP server by using ping and traceroute. Before practicing with these commands, let's peek at the RFC of the NTP protocol.

Exercise 3. Donwload and skim NTP's RFC and tell what transport it uses and the port number used by NTP servers.

Exercise 4. Monitor the NTP traffic from your host by running Wireshark, use a display filter 'ntp'; you may have to wait a few minutes before some NTP traffic appears. Confirm that the transport and the port you established in the preceding question are correct. Depict an NTP protocol graph.

Exercise 5. Decode fields in the NTP request packet sent by your client, then, respond to the following questions related to the NTP concepts developed on the lectures:

- a. What NTP stratum does your server belong in?
- b. What physical connection exists between your server and the physical atomic clock?
- c. Explain the meaning of the Reference, Origin and Receive Timestamps
- d. What is the field width of the aforementioned timestamps? They're represented with fixed-point numbers that represent a number of seconds, what's the smallest number of seconds representable? And the maximum?

Exercise 6. Issue the following command which queries your NTP client configuration, then, respond to the ensuing questions

\$ ntpq -pn

- a. What's the polling frequency used by your client?
- b. What is the delay between your client and the NTP server?
- c. What does jitter mean? Jitter represents the variability of the delay, does that make sense?
- d. What's the Rtt resulting from the execution of ping against the server? Is it equal to the ntpq's delay field?

3. Wall clock time and the ICMP protocol

One of the functions of the icmp protocol allows a system to find out another system's wall clock time so that the difference between both clocks can be calculated and thus, eventually, compensated for. The next experiment consists of two computers, Mac and Linux connected via Internet, the Mac use wants to discover how much its clock is skewed versus that of Linux, to that end, she executes the following Unix (OS-X) command:

```
$ sudo timedc
Passwd:
timedc> clockdiff 192.168.2.122
timedc: 192.168.2.122 will not tell us the date
time on 192.168.2.122 is 47161 ms. behind time on Mac-2.local
```

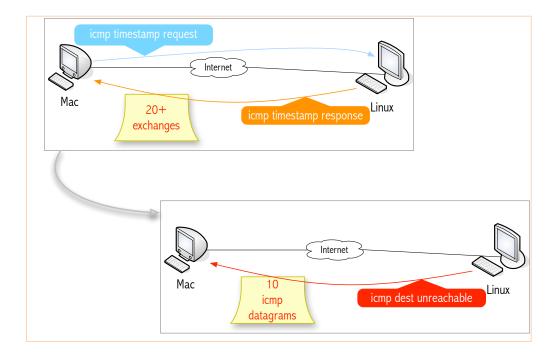


Fig. 3. The Mac computer sends several back-to-back icmp timestamp requests to the Linux computer, after that, the Mac receives 10 icmp destination unreachable messages from Linux

The timedc at Mac (192.168.2.106) command controls the UNIX timed daemon which sends icmp timestamp requests to Linux (192.168.2.122) and, for each one of them it expects to receive an icmp timestamp response; then, the command, interprets the contents of the icmp response and computes the time difference, which in this case results in Linux being 47161 ms *behind* Mac¹. See the following timed UNIX man page for more detail:

¹ Usually, instead of saying "Mac is behind", "Mac is slow" is used in the field of computer wall clock time.

● ○ ○	☆ Chema — less — 80×24	[™] ™			
TIMED(8)	BSD System Manager's Manual	TIMED(8)			
NAME timed time serv	er daemon				
SYNOPSIS timed [-dtM] [-i <u>network</u> -n <u>network</u>] [-F <u>host</u>]					
DESCRIPTION	is a time conver doomon which may be it	welled at beat			
time via launchd(8	<pre>is a time server daemon which may be in). (See launchd.plist(5) for more info</pre>	ormation about			
	s to run at boot.) It synchronizes the ther machines, which are also running t				
area network. The	se time servers will slow down the cloo	cks of some			
	up the clocks of others to bring them average network time is computed from	<u> </u>			
	using the ICMP timestamp request message				

Fig.2. The OS-X **\$ man timed** highlighting that **timed** is a server program capable of slowing down or speeding up other system's clocks to have them within average network time.

The exercises that follow illustrate the mentioned icmp protocol exchanges. Before starting to work the exercises, please, download the RFC where the ICMP protocol was documented by the IETF, it will be necessary to decode the WS (Wireshark) traces included alongside the exercises (Mac computer's host name is Brainstorm and the Linux computer's is Josephus).

Exercise 6. Observe the Request/Response sequence caused by the execution of timedc at Mac, what protocol encapsulates those ICMP R/R messages? (See fig. 3).

Exercise 7. Consult the RFC that you just downloaded and map the fields of the Type 13 message to those of the WS trace at Fig. 4.

Exercise 8. According to the WS trace of Fig. 5., how many bits make up any of the three relevant timestamp fields of the included ICMP Type 14 message.

Exercise 9. See Fig. 6 which contains the last message of the WS trace and provide a speculative explanation of it.

00	00		X Capturir	ng from Thunderbolt Eth	rnet: en1 [Wires	hark 1.10.3 (SVN Rev 53	3022 from /	trunk-1.10)]			
<u>F</u> ile	<u>E</u> d	it <u>V</u> iew <u>G</u> o <u>C</u>	apture <u>A</u> nalyz	e <u>S</u> tatistics Telep	hony <u>T</u> ools <u>I</u>	nternals <u>H</u>	elp					
۲	٢	🥂 🔲 🙇	B X	3 9, 4 🌳	🍋 ዥ 👱		⊕ ⊝	(H), ++	M 🕅	🖢 💥 🛛 🔯		
Filt	er: <mark>i</mark>	cmp		•	Expression	Clear App	ly Save					
No.		Time	Source	Destination	Protoco	Length I	nfo					4
	533	7 339.216820000	192.168.2.106	192.168.2.122	ICMP	62 T	imestamp	request	id=0x4903,	seq=25600/100	, ttl=64	
		8 339.216952000		192.168.2.106			imestamp			seq=25600/100		
		9 339.217013000		192.168.2.122			imestamp			seq=28160/110		
		0 339.217144000		192.168.2.106	ICMP	60 T	imestamp			seq=28160/110	, ttl=64	
		1 339.217206000		192.168.2.122					noran — timed			E M
		2 339.217338000 3 339.217401000		192.168.2.106	Drainstorm-2							
		4 339.217530000		192.168.2.106						SC, SIMPLEX, MU	ILTICAST>	mtu 1500
		5 339.217593000		192.168.2.122	opui				N_HWTAGGING	i, AV>		
		6 339.217723000		192.168.2.106	ethe	r 40:6c:8						
		7 339.217785000		192.168.2.122	inet					ixlen 64 scop		
		8 339.217916000		192.168.2.106	THE				tttttt00 br	oadcast 192.1	.68.2.255	
		0 339.218128000		192.168.2.106	nuo	options=1						
		0 341.219334000		192.168.2.106		a: autose	lect (10	00basel ·	<tull-duple< td=""><td>x,flow-contro</td><td>,energy</td><td>-efficient-</td></tull-duple<>	x,flow-contro	,energy	-efficient-
		2 343.220067000		192.168.2.106	ethernet~)							
		1 345.221209000		192.168.2.106		us: active		manant a	do timodo			
		9 347.222266000		192.168.2.106	Brainstorm-2				μαό τιπέας			
	559	0 349.223046000	192.168.2.122	192.168.2.106	$= 11 e_0C> C C C C C C C C C C$							
	562	0 351.224187000	192.168.2.122	192.168.2.106						n Brainstorm-	2 10001	
	562	2 353.225331000	192.168.2.122	192.168.2.106	timedc>	100.2.122	15 4/20	o ilis. Dei	iiiiu cine u	In Drainstorm-	-2. LUCal	
	565	2 355.226421000	192.168.2.122	192.168.2.106	cimenc>							
	568	8 357.227521000	192.168.2.122	192.168.2.106								
٩C												
Þ Fr	ame	5317 62 bytes	on wire (496 bi	ts), 62 bytes captu								
				:6c:8f:2d:67:c7), D								
		et Protocol Ver										
		et Control Mess		chema@Joseph								
			5	Linux Joseph								
				eth0 Li								
						160 2 12) Bear	+.102 1¢	0 0 055	Mask:255.255		
				in	et6 addr: fe						.255.0	
0000		3d 7e d5 e0 f6										
0010		30 22 e3 00 00 7a 0d 00 a2 3c			BROADCAST H							
0020		00 00 00 00 00 00			packets:118							
				17	packets:588			ea:0 ove	erruns:0 c	arrier:0		
)	Thu	underbolt Ether	net: en1: P		llisions:0 1			mar 1				
			1	RX	bytes:10995	08 (107.3	K1B)	TX bytes	3:79636 (7	/./ KIB)		
				-1								
				chema@Joseph	us:~\$ hostna	ame						
	,		-	Josephus			_					

Fig.3. timed daemons exchanging icmp timestamp messages to bring the slave's clock (Linux computer whose name is Josephus) into sync with the server (The MAC computer whose name is brainstorm).

5347 339.217785000 192.168.2.106 192.16 5348 339.217916000 192.168.2.122 192.16 5350 339.218128000 192.168.2.122 192.16 5360 341.219334000 192.168.2.122 192.16 5380 341.219334000 192.168.2.122 192.16 5382 343.220067000 192.168.2.122 192.16	B Frame 5347: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0 B Ethernet II, Src: Apple_2d:67:c7 (40:6c:8f:2d:67:c7), Dst: Micro-St_d5:e0:f6 (d4:3d: Compared by the state of the state o
5501 345.221209000 192.168.2.122 192.16 5559 347.222266000 192.168.2.122 192.16 5590 347.222266000 192.168.2.122 192.16 5500 349.223046000 192.168.2.122 192.16 5620 351.224187000 192.168.2.122 192.16 5622 355.226421000 192.168.2.122 192.16 5688 357.227521000 192.168.2.122 192.16 5688 357.227521000 192.168.2.122 192.16 ame 5347: 62 bytes on wire (496 bits), 62 byte hernet II, Src: Apple_2d:67:67 192.168.2.106 192.168.2.106 192.168.2.106 ternet Protocol Version 4, Src: 192.168.2.106 192.168.2.106 192.168.2.106 192.168.2.106	 Internet Control Message Protocol Type: 13 (Timestamp request) Code: 0 Checksum: 0x0c39 [correct] Identifier (BE): 18691 (0x4903) Identifier (LE): 841 (0x0349) Sequence number (BE): 38400 (0x9600) Sequence number (LE): 150 (0x0096) Originate timestamp: 16 hours, 33 minutes, 33.686 seconds after midnight UTC Receive timestamp: 16 hours, 13 minutes, 13.686 seconds after midnight UTC
d4 3d 7e d5 e0 f6 40 6c 8f 2d 67 c7 08 00 45 00 30 bd 34 00 00 40 01 00 00 c0 a8 02 6a c0 02 7a d 60 6c 40 03 40 03 66 60 03 9d 31 f6 66	

Fig.4. timed daemons exchanging icmp timestamp messages to bring the slave's clock (Linux computer whose name is Josephus) into sync with the server (The MAC computer whose name is brainstorm).

534/ 339.21//85000 192.108.2.10	ه 192.158.2.122 ICMP هک Ilmestamp request 1α=0x4903, seq=38400/150, TTL=64
5348 339.217916000 192.168.2.122	
5350 339.218128000 192.168.2.12	
5380 341.219334000 192.168.2.12	▷ Frame 5348: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
5382 343.220067000 192.168.2.12	Ethernet II, Src: Micro-St_d5:e0:f6 (d4:3d:7e:d5:e0:f6), Dst: Apple_2d:67:c7 (40:6c:8f:2d:67:c7)
5501 345.221209000 192.168.2.12	Internet Protocol Version 4, Src: 192.168.2.122 (192.168.2.122), Dst: 192.168.2.106 (192.168.2.106)
5559 347.222266000 192.168.2.12	▼ Internet Control Message Protocol
5590 349.223046000 192.168.2.12	Type: 14 (Timestamp reply)
5620 351.224187000 192.168.2.12	Code: 0
5622 353.225331000 192.168.2.12	Checksum: 0x93e0 [correct]
5652 355.226421000 192.168.2.12	Identifier (BE): 18691 (0x4903)
5688 357.227521000 192.168.2.12	
5666 557.227521000 152.100.2.12	Identifier (EE), our (oxosis)
	Sequence number (BE): 38400 (0x9600)
ame 5348: 60 bytes on wire (480	Sequence number (LE): 150 (0x0096)
hernet II, Src: Micro-St_d5:e0:f	Originate timestamp: 16 hours, 33 minutes, 33.686 seconds after midnight UTC
ternet Protocol Version 4, Src:	Receive timestamp: 16 hours, 32 minutes, 46.399 seconds after midnight UTC
ternet Control Message Protocol	Transmit timestamp: 16 hours, 32 minutes, 46.399 seconds after midnight UTC
Type: 14 (Timestamp reply)	
Codo: 0	
charal and a second for a second the	0000 40 6c 8f 2d 67 c7 d4 3d 7e d5 e0 f6 08 00 45 00 @l.g=~E.
	0010 00 28 1f a4 00 00 40 01 d4 fc c0 a8 02 7a c0 a8 .(@z
40 6c 8f 2d 67 c7 d4 3d 7e d5	0020 02 6a 0e 00 93 e0 49 03 96 00 03 8d al f6 03 8c .j
40 6C 81 20 67 C7 04 50 7e 05 00 28 1f a4 00 00 40 01 d4 fc	0030 e9 3f 03 8c e9 3f 00 00 00 00 00 00
00 20 11 84 00 00 40 01 04 10	

Fig.5. Timestamp reply, concrete ICMP type 14 message fields appearing in WS trace

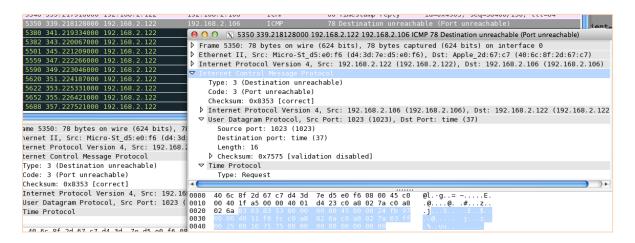


Fig.6. Last icmp message

Appendix A: Source code of the icmptimestamp.c program

```
/*
 * Parts of this source code were taken from the
 * W. Richard Stevens' book on Unix Network Programming
 * (Prentice-Hall 1998)
 * Refactorization, additional comments and adaptation
 * for the purposes of the CN Lab by José María Foces Moran 2014
 * Technical details about the structure of the ICMP datagram and IP
packets
 * may be obtained from RFC
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/param.h>
#include <sys/time.h>
#include <sys/file.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in systm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netdb.h>
#include <unistd.h>
#include <errno.h>
#define TRUE 1
/*
 * #Bytes of data, following ICMP header (Stevens' UNIX Net Programming
book)
 * Data that goes with ICMP echo request
 */
int datalen = 12;
struct sockaddr targetHost; //generic sock address for the target host
int rawSocket;
int nsent = 0;
int pid;
#define MAXIPHEADERLENGTH 60
```

```
#define MAXICMPPAYLOADLENGTH 76
#define MAXIPPACKETSIZE (65536 - 60 - 8)
#define MAGICSEQN 512
struct timeval originateTimeval, receiveTimeval;
long tsorig, tsrecv;
long tsdiff;
int initSocket(char *targetHostIpDDN) {
    int rawSock;
    struct sockaddr_in *inetTargetHost; //inet socket address for target
host
    struct protoent *protocol;
    bzero((char *) &targetHost, sizeof (struct sockaddr));
    inetTargetHost = (struct sockaddr in *) &targetHost;
    inetTargetHost->sin_family = AF_INET;
    inet_aton(targetHostIpDDN, &(inetTargetHost->sin addr));
    protocol = getprotobyname("icmp");
    rawSock = socket(AF_INET, SOCK_RAW, protocol->p_proto);
    return rawSock;
} //end of initSock()
unsigned char *createPacket(int *packlen) {
    unsigned char *packet;
    *packlen = datalen + MAXIPHEADERLENGTH + MAXICMPPAYLOADLENGTH;
    packet = (unsigned char *) malloc((unsigned int) *packlen);
    return packet;
}//end of createPacket()
/*
 * The source code of function internetChecksum are original from
 * W. Richard Stevens' book on Unix Network Programming
 * (Prentice-Hall 1998) in its entirety, no modification has been
 * carried out by JMFoces whatsoever except for the function name
 */
unsigned short internetChecksum(u short *addr, int len) {
    int nleft = len;
    int sum = 0;
   unsigned short *w = addr;
   unsigned short answer = 0;
    while (nleft > 1) {
        sum += *w++;
        nleft -= 2;
    }
```

```
if (nleft == 1) {
        *(unsigned char *) (&answer) = *(unsigned char *) w;
        sum += answer;
    }
    sum = (sum >> 16) + (sum \& 0xffff);
    sum += (sum >> 16);
    answer = ~sum;
    return (answer);
}
void sendRequest() {
    int len;
    struct icmp *icmp;
    unsigned char requestPacket[MAXIPPACKETSIZE];
    icmp = (struct icmp *) requestPacket;
    icmp_>icmp_type = ICMP_TSTAMP;
    icmp->icmp code = 0;
    icmp_seq = MAGICSEQN;
    icmp->icmp id = pid;
    gettimeofday(&originateTimeval, NULL);
    tsorig = (originateTimeval.tv sec % (24 * 3600)) * 1000 +
originateTimeval.tv usec / 1000;
    icmp_>icmp_otime = htonl(tsorig);
    icmp->icmp rtime = 0;
    icmp->icmp ttime = 0;
    len = datalen + 8;
    icmp->icmp cksum = internetChecksum((u short *) icmp, len);
    sendto(rawSocket, (char *) requestPacket, len, 0, &targetHost, sizeof
(struct sockaddr));
}
int processPacket(char *buf, int n, struct sockaddr in *from) {
    int headerLength;
    struct icmp *icmp;
    struct ip *ip;
    struct timeval delta;
    //Compute IP header length
    ip = (struct ip *) buf;
    headerLength = ip->ip hl << 2;</pre>
    //Subtract header length from n
    n -= headerLength;
    icmp = (struct icmp *) (buf + headerLength);
    /*
     * Discard all ICMP packets which ICMP type is not
     * RFC 792 type value 14 for timestamp reply message represented by
     * ICMP TSTAMPREPLY constant
     */
```

```
if (icmp->icmp type == ICMP TSTAMPREPLY) {
        if (icmp->icmp_seq != MAGICSEQN)
            printf("Spurious sequence received %d\n", icmp->icmp_seq);
        if (icmp->icmp id != getpid())
            printf("Spurious id received %d\n", icmp->icmp id);
        //Receive timestamp
        tsrecv = ntohl(icmp->icmp_rtime);
        //Difference between Receive timestamp and originate timestamp:
        tsdiff = tsrecv - tsorig; // ms
        printf("Originate = %ld, receive = %ld\n",
                ntohl(icmp->icmp_otime), ntohl(icmp->icmp_rtime));
        printf("Adjustment = %ld ms\n", tsdiff);
        delta.tv sec = tsdiff / 1000;
        delta.tv usec = (tsdiff % 1000) * 1000;
        printf("Correction = %ld sec, %ld usec\n", delta.tv_sec,
delta.tv_usec);
        /* adjtime() makes small adjustments to the system time,
         * as returned by gettimeofday(2), advancing or retarding it by
the time
         * specified by the timeval delta
         * See the man page for adjtime
         */
        adjtime(&delta, (struct timeval *) 0);
        return (0); //Timestamp reply
    } else
        return (-1); //Not timestamp reply
}
void receiveResponse(int packetLength, unsigned char *packet) {
   struct sockaddr_in from;
    int nbytes;
    int fromlen;
    while (TRUE) {
        fromlen = sizeof (from);
        nbytes = recvfrom(rawSocket, (char *) packet, packetLength, 0,
                (struct sockaddr *) &from, &fromlen);
        if (nbytes < 0) {
            printf("Bytes received < 0");</pre>
            fflush(stdout);
            if (errno == EINTR)
                continue;
            else
                perror("recvfrom error");
        }
        if (processPacket((char *) packet, nbytes, &from) == 0)
            exit(0);
```

```
}
}
int main() {
    unsigned char *packet;
    int packetLength;
    pid = getpid();
    rawSocket = initSocket("192.168.2.106");
    packet = createPacket(&packetLength);
    sendRequest();
    receiveResponse(packetLength, packet);
```

} //end of main()