INSTITUTO POLITÉCNICO NACIONAL
Escuela Superior de Cómputo

ESCOM

Practica

"NTP"

Alumnos
Arcos Ayala Jonathan
Zepeda Ibarra Allan Ulises

Grupo
4CV3

0.1 Introduccion

En una red de ordenadores hay varios ordenadores que están conectados entre si por un cable. A través de dicho cable pueden transmitirse información. Es claro que deben estar de acuerdo en cómo transmitir esa información, de forma que cualquiera de ellos pueda entender lo que están transmitiendo los otros, de la misma forma que nosotros nos ponemos de acuerdo para hablar en inglés cuando uno es italiano, el otro francés, el otro español y el otro alemán. Una forma de conseguir que dos programas se transmitan datos, basada en el protocolo TCP/IP, es la programación de sockets. Un socket no es más que un "canal de comunicación" entre dos programas que corren sobre ordenadores distintos o incluso en el mismo ordenador.

Desde el punto de vista de programación, un socket no es más que un "fichero" que se abre de una manera especial. Una vez abierto se pueden escribir y leer datos de él con las habituales funciones de read() y write() del lenguaje C. Existen básicamente dos tipos de "canales de comunicación" o sockets, los orientados a conexión y los no orientados a conexión. Las estructuras son usadas en la programación de sockets para almacenar información sobre direcciones. La primera de ellas es struct sockaddr, la cual contiene información del socket.

```
 struct sockaddr
{
   unsigned short sa_family;  /* familia de la direcci n */
   char sa_data[14];          /* 14 bytes de la direcci n del
       protocolo */
};
```

Pero, existe otra estructura, struct sockaddr_in, la cual nos ayuda a hacer referencia a los elementos del socket.

```
 struct sockaddr_in
{
   short int sin_family;         /* Familia de la Direcci n
                    */
   unsigned short int sin_port; /* Puerto
                      */
   struct in_addr sin_addr;      /* Direcci n de Internet
                     */
   unsigned char sin_zero[8];    /* Del mismo tama o que struct
       sockaddr */
};
```

socket()

```
 #include <sys/types.h>
#include <sys/socket.h>

int socket(int domain,int type,int protocol);
```

bind()

```
 #include <sys/types.h>
#include <sys/socket.h>

int bind(int fd, struct sockaddr *my_addr,int addrlen);
```

connect()

```
 #include <sys/types.h>
#include <sys/socket.h>

int connect(int fd, struct sockaddr *serv_addr, int addrlen);
```

listen()

```
 #include <sys/types.h>
#include <sys/socket.h>

int listen(int fd,int backlog);
```

accept()

```
 #include <sys/types.h>
#include <sys/socket.h>

int accept(int fd, void *addr, int *addrlen);
```

send()

```
 #include <sys/types.h>
#include <sys/socket.h>

int send(int fd,const void *msg,int len,int flags);
```

recv()

```
 #include <sys/types.h>
#include <sys/socket.h>

int recv(int fd, void *buf, int len, unsigned int flags);
```

recvfrom()

```
 #include <sys/types.h>
#include <sys/socket.h>

int recvfrom(int fd,void *buf, int len, unsigned int flags
struct sockaddr *from, int *fromlen);
```

close()

```
 #include <unistd.h>

close(fd);
```

shutdown()

```
 #include <sys/socket.h>

int shutdown(int fd, int how);
```

gethostname()

```
 #include <unistd.h>

int gethostname(char *hostname, size_t size);
```

NTP define una arquitectura para un servicio de tiempo y un protocolo para distribuir la información del tiempo sobre Internet.

Los servidores primarios están conectados directamente a una fuente de tiempo como un radioreloj recibiendo UTC. Los servidores NTP se sincronizan entre sí en uno de los tres modos:

Multidifusión: está pensado para su uso en LAN de alta velocidad.

Llamada a procedimiento: es similar al método de Cristian . este modo es adecuado cuando se requieren precisiones mas altas que las obtenidas con el método de multidifusión.

Modo simétrico: está pensado para su uso en servidores que proporcionan información de tiempo en LANs y por los estratos más bajos de la subred de sincronización.

En el modo de llamada a procedimiento remoto y en el simétrico, los procesos intercambian pares de mensajes. Cada mensaje lleva marcas de tiempo de los sucesos del mensaje reciente:

- El tiempo local cuando el mensaje anterior NTP entre el par fue enviado $(Ti-3)$.
- El tiempo local cuando el mensaje anterior NTP entre el par fue recibido $(Ti-2)$.
- El tiempo local cuando el mensaje actual fue transmitido $(Ti-1)$.
- El tiempo local cuando el mensaje actual fue recibido (Ti).

Por cada par de mensajes enviados entre dos servidores NTP calcula una compensación (o), la cual es una estimación de la deriva actual entre dos relojes.

Si el desplazamiento del reloj en B con A es o ,y si los tiempos de transmisión actuales para m y m' son t y t' respectivamente, entonces:

$Ti-2 = Ti-3+t+o$

$Ti = Ti-1+t'+o$

Lo cual conduce a: $di = t+t' = Ti-2-Ti-3+Ti-Ti-1$

$$o = oi + \frac{t-t'}{2} \tag{1}$$

$$oi = \frac{Ti-2-Ti-3+Ti-1-Ti}{2} \tag{2}$$

0.2 Desarrollo

Primeramente nos dedicamos a investigar el funcionamiento de la funcion date:

```
 #> date
Sun Jun 24 18:50:23 CDT 2007
```

Para eso usaremos el comando date -s que nos permite cabiar la hora de nuestra computadora. Posteriormente nos dedicamos a poner en marcha el cliente y el servidor que comunicaremos por medio de sockets, logrando generar 8 mensajes para asi poder calcular la compensacion del reloj del cliente.

```
if (PROT == 2)
        {
                printf("Abriendo conexion .....\n");
                if(connect(s, (struct sockaddr *)&si_other, sizeof(
                    struct sockaddr))==-1){
```

```
                printf("Error en el connect\n");
                return 1;
        }
        struct sinc aux;
        time_t o=time(0);
        printf("Iniciando de transmicion ....\n");
        for (i = 0; i < 8; ++i)
        {
                if (recv(s, &aux, sizeof(aux),0) == -1){
                        printf("Error en recv\n");
                        return 1;
                }
                aux.T2=time(0);
                aux.T1=time(0);
                if (send(s, &aux, sizeof(aux),0) == -1){
                        printf("Error en send\n");
                        return 1;
                }
        }
        if (recv(s, &o, sizeof(o),0) == -1){
                printf("Error ne sendto\n");
                return 1;
        }
        printf("%ld\n", o);
o = time(0)+o;
        struct tm *tlocal = localtime(&o);
        strftime(inst,30,"date -s '%y-%m-%d %H:%M:%S'",
            tlocal);
        system(inst);
        printf("%s\n", inst);
        printf("Siconizacion terminada\n");
        delay(del);
}

close(s);
```

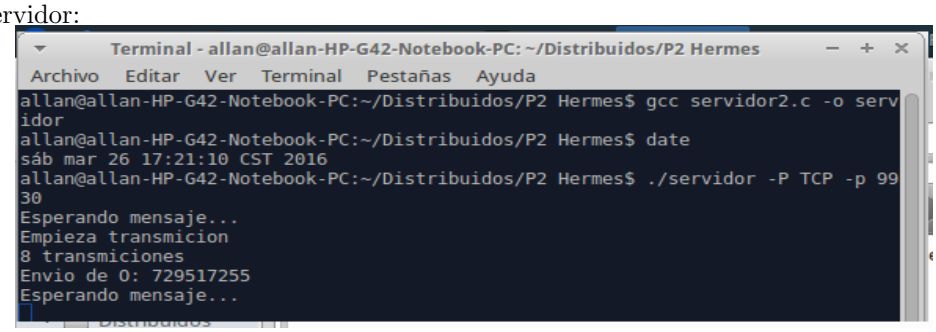
por ultimo se hace el calculo descrito en la introduccion para el algoritmo NTP y actualizamos la hora en el cliente.

```
for (i = 0; i < 8; ++i)
        {
            aux.T=0;
            aux.T1=0;
            aux.T2=0;
            aux.T3 = time(0);
            if (sendto(s, &aux, sizeof(aux),0,(struct sockaddr
                *)&si_other, sizeof(struct sockaddr)) == -1) {
                perror("Error en el envio sendto");
                return 1;
            }
            if (recvfrom(s, &aux, sizeof(aux), 0, (struct
                sockaddr *)&si_other, &slen)==-1){
                perror("Error en la recepcion recvfrom");
                return 1;
            }
            aux.T = time(0);
            tiempos[i].o = (aux.T2-aux.T3+aux.T1-aux.T)/2;
            tiempos[i].d = (aux.T2-aux.T3+aux.T-aux.T1);
            //printf("%ld\t%ld\t%ld\t%ld\n", aux.T3, aux.T2,
                aux.T1, aux.T);
            //printf("%ld\t%ld\n", tiempos[i].o, tiempos[i].d);
            if (i!=0)
            {
```

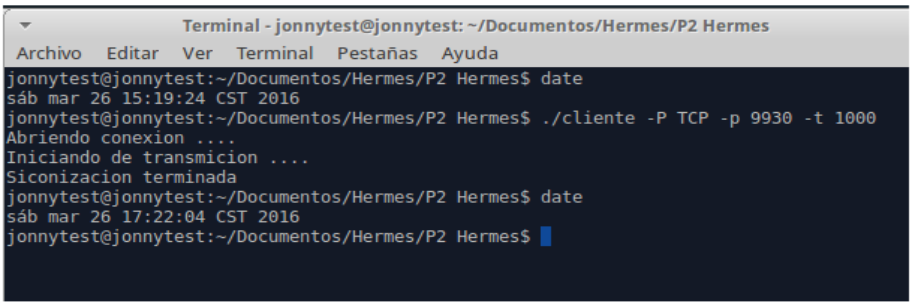
```
                    minD = minD<tiempos[i].d?minD:tiempos[i].d;
                    indiceTim = minD<tiempos[i].d?indiceTim:i;
                }else{
                    minD = tiempos[i].d;
                    indiceTim = i;
                }
            }
            o = (tiempos[indiceTim].o*-1)+(abs(aux.T2-aux.T3)-abs(
                aux.T-aux.T1))/2;
            //printf("Envio de tiempo: %ld\n", tiempos[indiceTim].o
                );
            //printf("Promedio de tiempo de envio: %d\n", ((abs(aux
                .T2-aux.T3)-abs(aux.T-aux.T1))/2));
            if (sendto(s, &o, sizeof(o),0,(struct sockaddr *)&
                si_other, sizeof(struct sockaddr)) == -1) {
                perror("Error en la recepcion recvfrom");
                return 1;
            }
            printf("Envio de O: %ld\n",o);
        }
        }
```

0.3 Pruebas

Servidor:



Cliente:



0.4 Conclusiones

Arcos Ayala Jonathan: El protocolo NTP nos ayuda a tener un cierto contro en el reloj logico entre las computadoras interconectadas entre si de una forma rápida y segura entre varias capas de una red

Zepeda Ibarra Allan Ulises: La sincronización no es cosa facil de lograr ya que manejar de manera correcta el tiempo de los sistemas operativos requiere de precicion de calculo, cualquier error te puede llevar a no poder ajustar el relojn de manera correcta por lo que hay que tener mucho cuidado en todos los calculos del algoritmo.