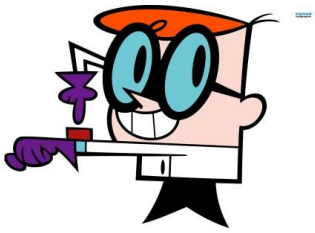
Algoritmo de Cristian

Elaborado por: Ukranio Coronilla

En UNIX uno de los comandos para controlar la fecha y hora del sistema es `timedatectl`. Al ejecutarlo podríamos obtener algo similar a lo siguiente:

```
      Local time: mar 2016-11-08 10:51:40 CST
  Universal time: mar 2016-11-08 16:51:40 UTC
        RTC time: mar 2016-11-08 16:51:40
       Time zone: America/Mexico_City (CST, -0600)
 Network time on: yes
NTP synchronized: yes
 RTC in local TZ: no
```



Para entender la salida del comando `timedatectl`, busque en Wikipedia el significado de los siguientes términos y razone la salida obtenida en su computadora:

Tiempo local CST - Central Standard Time

Universal Time - Coordinated Universal Time

RTC time - Real Time Clock

Time zone

NTP – Network Time Protocol

Es posible deshabilitar la sincronización NTP por red mediante el comando:

```
timedatectl set-ntp false
```

Observe los cambios que ocurren al ejecutar el comando `timedatectl .`

Ahora pruebe a inicializar el tiempo del sistema con el valor que usted quiera. Este es un ejemplo:

```
timedatectl set-time "2016-11-08 17:10:11"
```

Observe nuevamente los cambios que ocurren al ejecutar el comando `timedatectl .`

Es posible habilitar la sincronización NTP nuevamente mediante:

```
timedatectl set-ntp true
```

Ejercicio 1: Si tomamos al azar dos computadoras que se han sincronizado automáticamente por NTP, casi estarán sincronizadas con exactitud, y para probarlo elaboraremos un programa que emita un sonido beep cada que el segundero cambie. De modo que cuando ejecutemos el programa en computadoras distintas deberíamos escuchar casi simultáneamente el beep.

Para generar el sonido utilizaremos la librería de allegro que permite reproducir archivos wav dentro del código en C. Para ocuparla es necesaria su instalación mediante el siguiente comando:

```
sudo apt-get install liballegro4-dev
```

A continuación se muestra el código (tomado de la web) `allegro.c` para reproducir el archivo beep.wav:

```
#include <allegro.h>
#include <stdio.h>

int main() {

   SAMPLE *audio;
   int pos, vox;

   // Inicializa la librería de Allegro  "liballeg.so.4.4.2"  :
   if (install_allegro(SYSTEM_AUTODETECT, NULL, NULL)!= 0) {
      allegro_exit();
      printf("Error: no se puede inicializar la librería ''allegro' !");
      exit(-1);
   }

   // Instala el módulo de sonido:
   if (install_sound(DIGI_AUTODETECT, MIDI_NONE, NULL)!= 0) {
      allegro_exit();
      printf("Error: imposible la instalación del módulo de sonido !");
      exit(-1);
   }

   // Sube el archivo wav:
   audio = load_wav("beep.wav");
   if (!audio) {
      allegro_exit();
      printf("Error: imposible subir el archivo !");
      exit(-1);
   }

   //   Extrae información general del archivo wav:
   printf("Resolución de muestreo: %dbit\n", audio->bits);
   printf("Canales de salida: %d\n", audio->stereo*-1+1);
   printf("Frecuencia de muestreo: Hz %d\n", audio->freq);

   // Ejecuta el file wav:
   vox = play_sample(audio, 255, 128, 1000, 0);

   do {
      pos = voice_get_position(vox);
      printf("Ejecución de la muestra de audio n. %d\r", pos);
   } while (pos!=-1);


   // Terminar:
```

```
   destroy_sample(audio);
   remove_sound();

   return 0;

}
```

La compilación del archivo se llevaría a cabo como sigue:

```
gcc allegro.c -o allegro `allegro-config --libs`
```

No olvide descargar el archivo beep.wav de este servidor para ejecutar el programa.

Ejercicio 2: Suponga ahora que quiere sincronizar computadoras con mayor precisión (el servidor NTP normalmente está lejos), o computadoras que no se encuentran conectadas a Internet, o simplemente le interesan relojes lógicos más que físicos.

Implemente el algoritmo de Cristian visto en clase para llevar acabo la sincronización de las computadoras de su equipo.

Para inicializar el tiempo del sistema dentro de su programa con precisión, utilice la función `settimeofday()`; cuidando lo siguiente:

a) La sincronización NTP debe estar deshabilitada.
b) El programa debe ejecutarse como superusuario para que se realicen los cambios.

Nota: También es posible hacer cambios en la hora del sistema de manera gradual con la función `adjtime()`, lo cual sería más apropiado, pero con objeto de no demorar la práctica no lo utilizaremos.

Además de Internet, el algoritmo de Cristian se explica en el libro “Sistemas Operativos Distribuidos” de Andrew Tanenbaum (página 128) y se transcribe a continuación.

Algoritmo de Cristian

Comenzaremos con un algoritmo adecuado para los sistemas en los que una máquina tiene un receptor WWV y el objetivo es hacer que todas las máquinas se sincronicen con ella. Llamaremos a la máquina con el receptor WWV un **servidor del tiempo**. Nuestro algoritmo se basa en el trabajo de Cristian (1989) y trabajos anteriores. En forma periódica, en un tiempo que no debe ser mayor que $\delta/2\rho$ segundos, cada máquina envía un mensaje al servidor para solicitar el tiempo actual. Esa máquina responde tan pronto como puede con un mensaje que contiene el tiempo actual, C_{UTC}, como se muestra en la figura 3-6.

Como primera aproximación, cuando el emisor obtiene la respuesta, puede hacer que su tiempo sea C_{UTC}. Sin embargo, este algoritmo tiene dos problemas, uno mayor y otro menor. El problema mayor es que el tiempo nunca debe correr hacia atrás. Si el reloj del emisor es rápido, C_{UTC} será menor que el valor actual de C del emisor. El simple traslado de C_{UTC} podría provocar serios problemas, tales como tener un archivo objeto compilado justo después del cambio de reloj y que tenga un tiempo anterior al del archivo fuente, modificado justo antes del cambio del reloj.

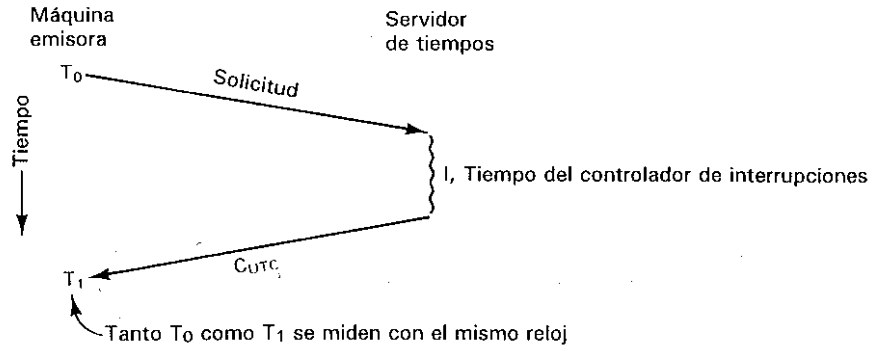


Figura 3-6. Obtención de la hora actual por medio de un servidor de tiempo.

Tal cambio se debe introducir de manera gradual. Una forma es la siguiente. Supongamos que el cronómetro se ajusta para que genere 100 interrupciones por segundo. Lo normal sería que cada interrupción añadiera 10 milisegundos al tiempo. Al reducir la velocidad, la rutina de interrupción sólo añade 9 milisegundos cada vez, hasta que se haga la corrección. De manera similar, el reloj se puede adelantar de manera gradual, sumando 11 milisegundos en cada interrupción en vez de saltar hacia adelante de una vez.

El problema menor es que el tiempo que tarda el servidor en responder al emisor es distinto de cero. Peor aún, este retraso puede ser de gran tamaño y varía con la carga de la red. La forma de enfrentar este problema por parte de Cristian es intentar medirlo. Es muy fácil para el emisor registrar de manera precisa el intervalo entre el envío de la solicitud al servidor de tiempo y la llegada de la respuesta. Tanto el tiempo de inicio, T_0, como el tiempo final, T_1, se miden con el mismo reloj, por lo que el intervalo será relativamente preciso, aunque el reloj del emisor esté alejado de UTC por una magnitud sustancial.

En ausencia de otra información, la mejor estimación del tiempo de propagación del mensaje es de $(T_1-T_0)/2$. Al llegar la respuesta, el valor en el mensaje puede ser incrementado por esta cantidad para dar una estimación del tiempo actual del servidor. Si se conoce el tiempo mínimo de propagación teórico, se pueden calcular otras propiedades de la estimación del tiempo.

Esta estimación se puede mejorar si se conoce con cierta aproximación el tiempo que tarda el servidor del tiempo en manejar la interrupción y procesar el mensaje recibido. Llamaremos al tiempo para el manejo de interrupción I. Entonces la cantidad del tiempo desde T_0 hasta T_1 dedicada a la propagación del mensaje es $T_1 - T_0 - I$, por lo que una mejor estimación del tiempo de propagación en un sentido es la mitad de esta cantidad. Existen sistemas en los que los mensajes de A a B siguen de manera sistemática diferentes rutas de B a A, por lo que tienen un tiempo de propagación distinto, pero aquí no consideraremos estos sistemas.

Para mejorar la precisión, Cristian sugirió hacer no una medición, sino varias. Cualquier medición en la que $T_1 - T_0$ exceda cierto valor límite se descarta, por ser considerada víctima del congestionamiento en la red y por tanto no confiable. Las estimaciones obtenidas de las demás pruebas se pueden promediar para obtener mejor valor. Otra alternativa es que el mensaje que regrese más rápido se considere como el más preciso, pues supuestamente encontraría menos tráfico y por lo tanto sería el más representativo del tiempo de propagación puro.