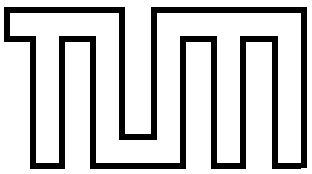


Technische Universität München

Fakultät für Informatik
Forschungs- und Lehreinheit Informatik VI

Übung zur Vorlesung Echtzeitsysteme

Aufgabe 3

Nadine Keddis
keddis@fortiss.org

Dominik Sojer
sojer@in.tum.de

Stephan Sommer
sommerst@in.tum.de

Michael Geisinger
geisinge@in.tum.de

Wintersemester 2011/12

Simple Network Time Protocol (SNTP)

Um in einem verteilen System Ereignisse zeitgenau auslösen bzw. registrieren zu können ist eine gemeinsame Zeitbasis notwendig. Das Problem dabei ist, dass Uhren, sebst wenn sie initial alle die gleiche Zeit anzeigen, im laufe der Zeit voneinander abweichen bzw. auseinander driften. Ausgelöst wird dies duch minimalste Frequenzunterschiede. Um trotzdem eine zuverlässige Zeitbasis zu erhalten, müssen die Uhren in verteilten System in geeigneten Abständen synchronisiert werden.

Um die Synchronisierung druchführen zu können, benötigt man geeignete Protokolle. Verbreitete Protokolle sind das *Precision Time Protocol* (PTP) und *Network Time Protocol* (NTP). Eine vereinfachte Variante von NTP ist SNTP. SNTP steht für *Simple Network Time Protocol* und ist in RFC 2030 spezifiziert (siehe http://www.faqs.org/rfcs/rfc2030.html).
(S)NTP verwendet UDP für die Nachrichtenübertragung (UDP Port 123). Die Genauigkeit der Zeitsynchronisierung beträgt 10 Millisekunden in einem globalen Netzwerk und 200 Mikrosekunden in einem lokalen Netzwerk. NTP wird meist mit einer UTC-Zeitskala eingesetzt. UTC (*Coordinated Universal Time*) beachtet Schaltsekunden im Gegensatz zu z.B. der TAI Zeitskala (*International Atomic Time*). Eine Übersicht zu unterschiedlichen Zeitskalen finden Sie unter http://stjarnhimlen.se/comp/time.html. Sowohl SNTP als auch NTP verwenden das gleiche Netzwerkpaketformat. Die Unterschiede bestehen vor allem darin, dass SNTP, im Gegensatz zu NTP, nur einen Zeitserver verwendet. Die Genauigkeit des SNTP Protokolls ist deshalb fast immer schlechter als die einer vollwertigen NTP Implementierung. Ausserdem kann SNTP Zeitsprünge aufgrund von Schaltsekunden nicht vermeiden.
Um die Grundlagen der Zeitsynchronisierung zu verdeutlichen, ist das SNTP Protokoll jedoch ausreichend.

Ziel

In der Übung wird ein SNTP Client implementiert, der die aktuelle Zeit von einem NTP Server (ntp.in.tum.de) bezieht und basierend darauf den Offset der eigenen Uhr berechnet. Neben der Funktionsweise eines (S)NTP Clients wird hierbei auch die Socket Schnitstelle (Schnitstelle zur Netzwerk-Kommunikation) betrachtet.

Die Aufgabe wird in der Programmiersprache C implementiert. Zum übersetzen wird Microsoft Visual Studio verwendet.

Entwicklungsumgebung

Zum erstellen eines Projekts gehen Sie wie folgt vor. Nach dem Starten von Visual Studio:

File → *New* → *Project* → *Win32* → *Win32 Console Application*

Project → *Properties* → *Configuration properties* → *C/C++* → *Advanced* → *Compile as „compile as C code“*

Anschliessend muss die Socket Bibliothek zum erstellten Projekt gelinkt werden:

Project → *Properties* → *Linker* → *CommandLine*

und tippen Sie „wsock32.lib“ ein.

Kopieren Sie nun das Rahmenprogramm aus der Datei `SNTPClient.c` in Ihre Projekt Sourcen.

Die Programmiersprache C bietet fast die gleiche Socket Schnittstelle unter Windows wie unter VxWorks. Die Aufgabe wird unter Windows gelöst, weil der Simulator in VxWorks nur begrenzte Netzwerk-Funktionalität hat.

Aufgabe 3: SNTP Client

Aufgabe 3.1: Netzwerk-Kommunikation

Um mit einem NTP Server kommunizieren zu können, muss zuerst die Kommunikation implementiert werden. Ergänzen Sie dazu die fehlenden Teile der Netzwerkfunktionalität des Rahmenprogramm aus `SNTPClient.c` and geeigneter Stelle.

Implementierungshinweis

Verwenden Sie für die UDP Kommunikation als Zielserver „ntp.in.tum.de“ und als Port 123.
Um eine Nachricht an den Server zu schicken benutzen Sie folgende Aufrufe:

- socket(); Erzeugt einen Socket für z.B. UDP
- sendto(); Schickt eine Nachricht zu einer angegeben Adresse
- ...
- closesocket(); Schliesst das angegebene Socket

Um eine Nachricht von dem Server empfangen zu können, sind folgende Funktionen nötig:

- socket(); Erzeugt ein Socket, das auf z.B. UDP gebunden ist
- recvfrom(); Empfängt eine Nachricht von einem Socket und speichert die Senderadresse
- ...
- closesocket(); Schliesst den angegebenen Socket

Zum Senden und Empfangen kann der gleiche Socket verwendet werden. Die (S)NTP Nachricht wird mit der Funktion *createNTPMessage*() erzeugt. Als Parameter benötigt diese Funktion unter anderem die Sendezeit des Paketes. Die aktuelle Zeigt erhält man under Windows durch den Aufruf von *time*(), der einen *time_t* Wert zurückliefert (http://msdn.microsoft.com/de-de/library/1f4c8f33%28VS.80%29.aspx).

Je nach Betriebssystem werden folgende Datentypen für die Zeitrepräsentation verwendet:

- *time_t* : stellt die Zeit als Anzahl von Sekunden (Windows, Linux, VxWorks)
- *struct timeval* : stellt die Zeit als 2 Werte dar: Sekunden und Mikrosekunden (Linux)
- *struct timespec* : stellt die Zeit als 2 Werte dar: Sekunden und Nanosekunden (VxWOrks)

Aufgabe 3.2: Erzeugen und Parsen von den NTP Nachrichten

Implementieren Sie die Funktion *decodeTimestamp()*, die das Parsen eines Zeitstempels einer (S)NTP Nachricht realisiert.

Implementierungshinweise decodeTimestamp

Die Zeitstempel von NTP werden als 64-bit Festkomma Werte codiert. Dabei umfasst der Ganzzahl Anteil 4 Byte und der Nachkommaanteil ebenfalls 4 Byte. Zu beachten ist, dass „Big Endian“ bei der Anordnung der Bytes verwendet wird. Dies bedeutet, dass das „most significant“ Byte die niedrigste Adresse hat.

Beispielscode zur Umrechnung eines Byte-Arrays im Big-Endian Format in einen double Datentyp:

```
//char pointer[] is the given array
int i; // variable to iterate in the array
short byte;
double base;
double timestamp = 0.0;
```

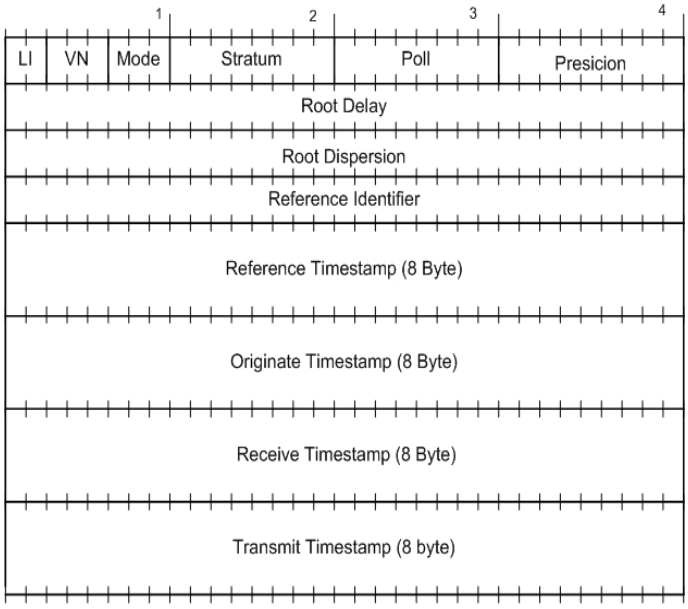
```
for (i = 0; i < 8; i++) {
        /* byte is a signed data type */
        byte = *(i + pointer);
        /* short is a signed data type */
        if (((*(i + pointer)) & 0x80) == 0x80)
                byte = (short) (128 + ((*(i + pointer)) & 0x7f));
        /* 2^24, 2^16, 2^8, 2^0, 2^-8, 2^-16, 2^-24, 2^-32 */
        base = pow((double)2, (double)(3 - i) * 8);
        timestamp = timestamp + (double) (byte * base);
}

return timestamp;
```

Testen Sie Ihre Implementierung von *decodeTimestamp()* mittels der vorgegebenen Funktion *int decodeTimestamp_test()*.

parseNTPMessage

Die Struktur eines NTP Paketes ist im Bild dargestellt. Eine Nachricht wird dabei als ein Array bestimmter Länge repräsentiert. Überlegen Sie sich eine mögliche Implementierung der Funktion *parseNTPMessage* und vergleichen Sie anschliessend Ihre Lösung mit der aus dem Rahmenprogramm.

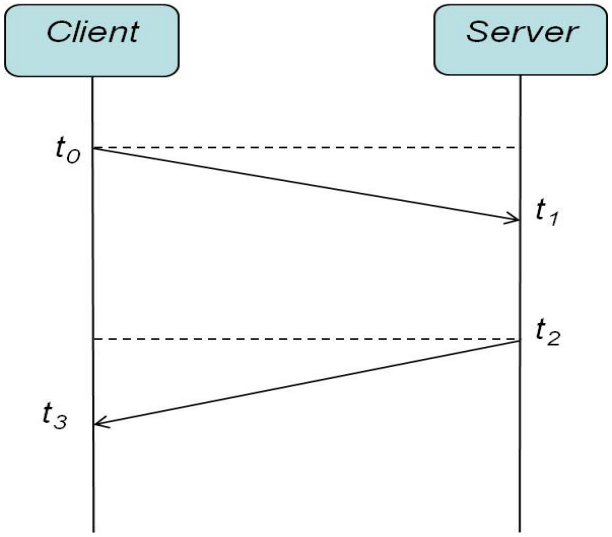


Genauere Informationen bezüglich des Aufbaus einer SNTP Message finden Sie unter `http://www.networksorcery.com/enp/protocol/sntp.htm`. Zum Versenden der SNTP Nachricht müssen folgende Felder gesetzt werden:

- Leap Indicator - LI
- Version Number - VN
- Mode
- Transmit Timestamp - ist nicht zwingend notwendig aber empfohlen

Aufgabe 3.3: Roundtrip Delay und Local Clock Offset

Bestimmen Sie den „Roundtrip Delay“ und den „Local Clock Offset“ und geben Sie diese Daten aus. Nähere Informationen hierzu können Sie der RFC2030 entnehmen.



Implementierungshinweis

Die Uhrzeit des lokalen Rechners kann man durch den *time*() Aufruf ermittelt werden. Merken Sie sich die aktuelle Uhrzeit in der Variable `localTimestamp` vor dem Senden des (S)NTP Packetes und übergeben Sie diese Uhrzeit der Funktion createNTPMessage(). Merken Sie die Empfangszeit des (S)NTP Paketes in der Variable `localTimestamp`, sie wird bei der späteren Berechnung benötigt.

Windows als auch Linux verwenden für (S)NTP Nachrichten eine UTC Zeitskala. Es ist jedoch zu beachten, dass UTC verschiedene „Epochen“ erlaubt. NTP zählt die Zeit seit dem 1. Januar 1900, Windows und Linux seit dem 1. Januar 1970. Um diese Zeitstempel umzurechnen müssen 2208988800 Sekunden addiert oder subtrahiert werden.

Wie sich der $RoundTripDelay$ bzw. der $Offset$ zur lokalen Uhr berechnen lassen, kann der $RFC2030$ entnommen werden. Einen Hinweis dazu finden Sie auch im Rahmenprogramm.