THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Gestion du temps dans les ordinateurs et les systèmes distribués

Tran, Vien Ha

Award date:
1999

Awarding institution:
Universite de Namur

Facultés Universitaires Notre - Dame de la Paix
Institut d'Informatique

Gestion du temps
dans les ordinateurs et les systèmes distribués

Vien Ha TRAN

Version du 30 août 1999

Travail de fin d'études réalisé en vue de l'obtention
du titre de Licencié en Informatique

Année Académique 1998-1999

Résumé

Ce memoire aborde la gestion du temps dans un système non distribué (un PC) et celui dans un système distribué (un réseau des PCs).

Dans le cas du PC, le fonctionnement de l'horloge du temps réel – RTC (Real Time Clock) ,de l'horloge du système – SC (System Clock) et la relation entre elles sont discutés. A partir de cette étude, le problème hardware de l'an 2000 et les solutions possibles pour celui-ci sont envisagées.

Dans le cas du réseau des PCs le problème de la synchronisation entre les PCs et les algorithmes pour résoudre celui-ci sont envisagés.

Abstract

This thesis is about the management of time of a non-distributed system (a PC) and one of a distributed system (Network of PCs).

In the case of one PC, the fonctionnement of the Real Time Clock, of the System Clock and their relation are discussed. From this research, the problem of hardware of the year 2000 and the possible solutions are discussed .

The synchronisation problem between the PCs in a network and the algorithms to solve these one are also questionned .

Table des matières

Table des figures

Liste des tableaux

Remerciements

La réalisation de ce mémoire n'aurait pas été possible sans le concours d'un certain nombre de personnes auxquelles je voudrais, ici, exprimer ma gratitude :

- Monsieur Jean RAMAEKERS, *professeur de systèmes d'exploitation à l'Institue d'Informatique des Facultés Universitaires Notre-Dame de la Paix, à Namur.*

- Monsieur Bob QUINN, *informaticien, membre de Winsock Group, Stardust Technologies Inc.*

- Monsieur Aimé KASSA, informaticien, *assistant à l'Institue d'Informatique des Facultés Universitaires Notre-Dame de la Paix, à Namur.*

- Nicolas RONDEAUX, étudiant , *2e licence en Informatique à l'Institue d'Informatique des Facultés Universitaires Notre-Dame de la Paix, à Namur.*

- Monsieurs Jean HENRARD, informaticien, *assistant à l'Institue d'Informatique des Facultés Universitaires Notre-Dame de la Paix, à Namur.*

- Les amis à *alt.winsock.programming Newsgroup*

- Jochen Van HOOFSTADT, *étudiant en marketing à Institut d'Enseignement Supérieur de Namur.*

Introduction

Le temps joue un rôle indispensable dans la vie quotidienne : grâce au temps, on organise son travail, on marque les événements ...

Dans le monde d'informatique, plusieurs applications se basent sur le temps d'ordinateur pour son fonctionnement. Par exemple, les applications de la banque, les applications du temps réel...

Ce mémoire est dans le but d'étudier la gestion du temps dans un système non-distribué et distribué.

Dans le *chapitre 1*, nous allons discuter de la gestion du temps dans un PC isolé. A partir de cette étude, nous allons aborder le problème hardware de l'an 2000 et les solutions possibles pour celui-ci.

Dans le *chapitre 2* nous allons discuter de la gestion du temps dans un système distribué. La gestion du temps dans un sytème distribué est non trivial, car pour se synchroniser les ordinateurs doivent échanger les messages contenant le timestamp. Mais le temps de transfert des messages n'est pas stable à cause du charge du réseau et celui de chaque ordinateur . Les algorithmes pour résoudre ce problème sont envisagés.

Nous avons écrit les programmes pour illustrer chaque chapitre.

Chapitre I : Gestion du temps dans un PC

Introduction

Avant d'analyser le problème de la synchronisation du temps dans un système distribué. Toutefois, il nous semble important de comprendre d'abord la façon dont est géré le temps dans un PC (Personal Computer).

Dans un PC, il existe deux horloges qui fonctionnent d'une manière parallèle :

- l'horloge du temps réel - RTC (Real Time Clock) - qui permet de conserver le temps même si le PC est mis hors tension ;
- l'horloge du système – SC (System Clock) - qui est utilisée par le système d'exploitation pour ses tâches, par exemple pour dater des répertoires et des fichiers (date de création ou de modification).

I.L'horloge du temps réel - RTC.

Dans les vieux PC/XTs, la configuration des composants matériels tels qu'un lecteur de disquettes, une carte graphique, le type de disque dur (le nombre des têtes, des cylindres, des secteurs) s'effectuait via des "DIP switches". Comme chaque PC avait sa configuration propre, la configuration à l'aide des "DIP switches" n'était pas très pratique. En outre, un PC/XT n'était pas capable de conserver le temps dès qu'il était mis hors tension. Chaque fois que l'ordinateur était redémarré, la date et le temps étaient remis à leurs valeurs initiales, 04/01/1980[1] et 00h:00m:00s.00. C'était à l'utilisateur de mettre à jour le temps et la date courants par la commande **date** et **time** de DOS. Le figure 1.1 illustre la commande **date** et **time** de DOS.

```
C:\>date ↵
La date actuelle est Ven 04/01/1980
Entrez la nouvelle date (jj.mm.aa) :

C:\>time ↵
L'heure actuelle est  0:01:10,82
Entrez la nouvelle heure :
```

Figure 1. 1: La commande date et time de DOS

Pour résoudre ces problèmes, l'IBM a ajouté dans les PC/ATs, les successeurs des PC/XTs, une mémoire spéciale, le CMOS (Complementary Metal Oxide Semiconductor), et un circuit spécialisé sur la carte mère, le RTC (Real Time Clock).

[1] il y a une raison historique pour ces valeurs : les premiers PCs sont sortis en 1980. Ils ne connaissaient donc pas les dates d'avant 01/01/1980.

I.1. Chip Motorola MC146818

Le CMOS est une petite mémoire RAM (Random Access Memory) qui stocke les informations de configuration du PC et le temps de l'horloge du temps réel, RTC.

Le CMOS et le RTC sont intégrés dans un seul chip - le *Motorola MC146818* ou compatible. Ce chip est alimenté par une pile permettant la conservation des informations contenues dans le CMOS et le fonctionnement du RTC même si la machine est mise hors tension. La vie de la pile dure environ 3 ans et 10 ans si c'est une pile lithium. La figure 1.2 illustre le chip MC146818.

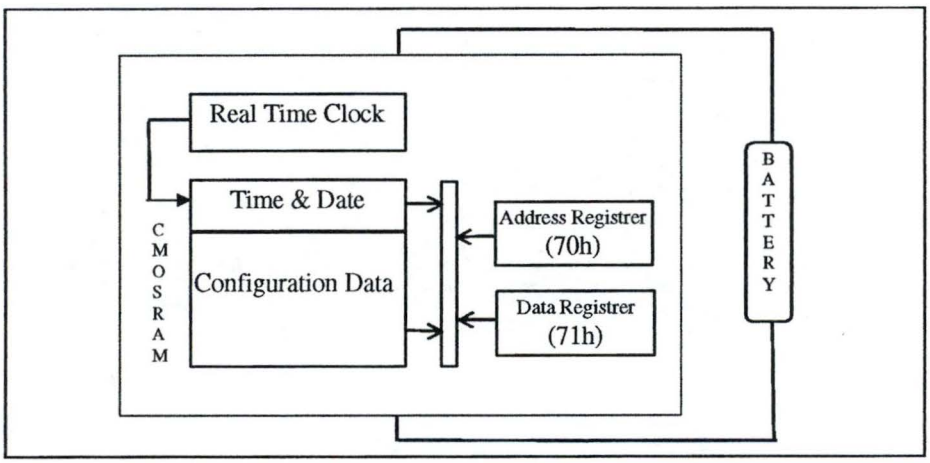


Figure 1. 2 : Chip Motorola MC146818

Le CMOS de *MC146818* contient normalement 64 octets d'information. Comme c'est le temps qui nous intéresse, le tableau 1.1 ne reprend que les significations des octets se rapportant au temps du RTC.

Adresse (Décimale)	Adresse (Hexadécimale)	Signification
0	00h	Seconde
1	01h	Seconde du réveil
2	02h	Minute
3	03h	Minute du réveil
4	04h	Heure
5	05h	Heure du réveil
6	06h	Jour dans la semaine
7	07h	Jour dans un mois
8	08h	Mois
9	09h	An
...	...	...
50	32h	Siècle
...	...	...

Tableau 1. 1 : Les octets de CMOS associés au RTC

Les octets 0,1,2 et 3 prennent une valeur de 00 à 59. Les octets 4 et 5 prennent une valeur de 00 à 23. L'octet 6 prend une valeur de 0 (Dimanche) à 6 (Samedi). L'octet 7 prend une valeur de 1 à 31. L'octet 8 prend une valeur de 1 (Janvier) à 12 (Décembre).

Les informations du RTC sont stockées dans ces octets sous le format BCD (Binary Coded Decimal). Le figure 1.3 illustre le format BCD du nombre décimal 95.

Décimal	9				5			
BCD	1	0	0	1	0	1	0	1

Figure 1. 3 : Binary Coded Decimal

Comme dans l'encodage BCD, chaque octet (8 bits) ne peut que contenir un nombre décimal de deux chiffres, l'octet 9 prend une valeur de 00 à 99. Si l'année est 1995 le contenu de l'octet 9 est 95 (10010101 en BCD), l'information concernant le siècle, c.-à-d. 19 (00011001 en BCD), est stockée dans l'octet 50.

Notons que pour certaines catégories des PC autres que la famille PC/AT, par exemple les PS/2, le siècle est mis dans l'octet 55 (37h) au lieu de l'octet 50.

Exemple

Supposons qu'on soit le 12-09-1995, 23:58:45. Les contenus des octets de CMOS associés au temps du RTC seront (le réveil étant réglé à 00 :00 :00) :

Adresse (Décimale)	Adresse (Hexadécimale)	Signification
0	00h	45 secondes
1	01h	00 seconde (réveil)
2	02h	58 minutes
3	03h	00 minute (réveil)
4	04h	23 heures
5	05h	00 heure (réveil)
6	06h	2 (Mardi)
7	07h	12 (jour)
8	08h	09 (mois)
9	09h	95 (année)
...	...	...
50	32h	19 (siècle)
...	...	...

Tableau 1. 2 : Le contenu des octets de CMOS associés au RTC pour le 12/09/1995, 23:58:45

Remarque
Les octets 1, 3 et 5 contiennent respectivement la seconde, la minute et l'heure du réveil (*alarm*). Lorsque le temps du réveil est atteint, le chip *MC146818* émet une interruption *4ah*. Ainsi, une application peut profiter de ce mécanisme pour effectuer périodiquement des tâches, en utilisant les fonctions *06h* et *07h* de l'interrupteur *1Ah* de BIOS. (voir tableau 1.3)

I.2. Mise à jour de l'horloge RTC

Après chaque seconde, le chip *MC146818* enregistre le temps du RTC en mettant à jour les octets 0, 2, 4, 6, 7, 8, 9. Pendant cette mise à jour , ces octets sont inaccessibles. Notons que le 50ème octet (32h) n'est pas mis à jour par le chip.

Il y a deux façons d'accéder à RTC :

a. directement par le registre d'adressage et le registre de donnée (voir figure 1.2) ; Ces deux registres sont appelés respectivement porte *70h* et porte *71h*.
b. indirectement par l'interrupteur (le service) *1Ah* de BIOS (voir tableau 1.3).

a. Accès direct

La figure 1.4 présente les instructions en assembleur permettant de lire directement le contenu du 50ème octet (32h) du CMOS :

```
mov AL, 32h     ; (1)
out 70h, AL     ; (2)
in AL, 71h      ; (3)
```

Figure 1. 4 : Lecture du contenu du 50ème octet (32h) du CMOS

L'instruction (1) assigne la valeur 32h (50, en décimal) au registre AL. Par l'instruction (2), le contenu du registre AL est assigné au registre d'adresse (la porte 70h) du chip, c.-à-d. l'adresse de l'octet du CMOS que nous voulons lire la valeur. Et, finalement, par l'instruction (3), le contenu de l'octet 50 est transféré vers le registre AL par la porte 71h.

D'une façon similaire, comme illustré sur la figure 1.5, nous pouvons affecter directement au 50ème octet la valeur 20.

```
mov AL, 32h
out 70h, AL
mov AL, 20h
out 71h, AL
```

Figure 1. 5 :Ecriture de la valeur 20 dans le 50ème octet (32h) du CMOS

b. Accès indirect

Nous pouvons aussi lire ou écrire indirectement dans le CMOS en appelant les fonctions 02h, 03h, 04h, 05h, et régler le réveil par les fonctions 06h et 07h de l'interrupteur 1Ah du BIOS (voir tableau 1.3). Le figure 1.6 nous montre comment mettre la date de RTC au 12-09-1995 en utilisant la fonction 05h de l'interrupteur 1Ah du BIOS.

```
mov AH, 05h    ; le nom de fonction est assigné au registre AH
mov CL, 95h    ; la valeur de l'an est assignée au registre CL
mov CH, 19h    ; la valeur du siècle est assignée au registre CH
mov DL, 12h    ; la valeur du jour de mois est assigné au registre DL
mov DH, 09h    ; la valeur du mois est assignée au registre DH
int 1Ah        ; appel de l'interrupteur 1Ah
```

Figure 1. 6 : Exemple de mise à jour de la date du RTC

Le tableau 1.3 décrit ces fonctions en détail.

Fonction	Registre	En entré	En sortie
02h – lire le temps du RTC.	AH CL CH DH Carry	02h	00h Minute Heure Seconde Erreur si <> 0
03h – mettre le temps du RTC.	AH CL CH DH Carry	03h Minute Heure Seconde	00h Erreur si <> 0
04h – lire la date du RTC	AH CL CH DL DH Carry	04h	00h An Siècle Jour Mois Erreur si <> 0
05h – mettre la date du RTC	AH CL CH DL DH Carry	05h An Siècle Jour Mois	00h Erreur si <> 0
06h – régler le réveil	AH CL CH DH Carry	06h Minute Heure Seconde	00h Erreur si <> 0
07h – reset le réveil avant de le régler	AH Carry	07h	 Erreur si <>0

Tableau 1. 3 : Les courantes de l'interrupteur 1Ah du BIOS.

Nous ne devons pas nécessairement écrire un programme pour appeler ces fonctions de BIOS (sauf pour 06h et 07h). En effet, nous pouvons appeler ces fonctions en utilisant le programme *setup* (ce programme est activé en pressant une touche du clavier, souvent la touche DEL, lors du démarrage de la machine) ou, plus facilement, en tapant le commande **date** ou **time** de DOS.

Le RTC a généralement une fréquence de 32768 Hz. Cette fréquence est générée par un "quartz cristal", souvent calibré pour travailler à la température de la chambre (typiquement 25° C). Si cette température est changée fortement, le RTC risque de ne pas fonctionner correctement.

II. L'horloge du système - SC

Comme signalé plus haut, pendant le démarrage de la machine, le BIOS envoie la date et l'heure du RTC au système d'exploitation. Ce dernier utilise cette valeur pour initialiser l'horloge du système. Le système d'exploitation se base sur cette horloge pour, par exemple, dater les fichiers et les répertoires.

II.1. Principe général

Le SC est réalisé sur base du compteur 0 du chip *8253/8254 PIT* (Programmable Interval Timer). Le compteur 1 est utilisé pour rafraîchir la mémoire dynamique et le compteur 2 est utilisé pour générer la tonalité du haut-parleur du PC (figure 1.7). Le compteur 0 reçoit la pulsation CLK et génère à sa sortie OUT0 18,206 "ticks" par seconde. A chaque tick, le chip PIC 8259A génère une interruption 08h ; Lorsque l'interruption 08h est notifiée au CPU, ce dernier met à jour un compteur de 32 bits se trouvant dans la mémoire RAM (16 bits dans *low timer count* et 16 bit dans *high timer count*). Ce compteur contient le nombre de ticks générés depuis minuit. Lorsque ce nombre dépasse une valeur équivalente à 24h (≈ 1573040 ticks), le *timer overflow flag* est mis à 1. Le tableau 1.4 présente le BIOS DATA AREA

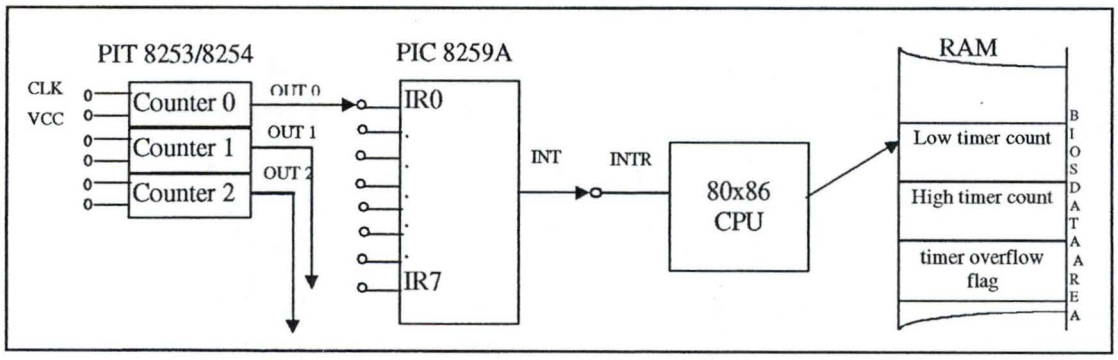


Figure 1. 7 : Principe de l'horloge du système

Adresse	Taille	Compteurs	Signification
40 :6C	Word	Low timer count	Le nombre de ticks depuis Minuit
40 :6E	Word	High timer count	
40 :70	Byte	Timer overflow flag	1 si plus de 24h sont passées depuis minuit

Tableau 1. 4 : BIOS DATA AREA

II.2. Mise à jour de l'horloge SC

Les langages de programmation fournissent souvent des fonctions pour lire/écrire la valeur de l'horloge du système. Par exemple, en BASIC, on utilise la fonction **timer**, en C, les fonctions **_dos_gettime** et **_dos_settime,** ou plus simplement en appelant les commandes **time** et **date** de DOS. Pour rappel, notons que les commandes **time** et **date** peuvent être utilisées pour mettre à jour le RTC.

Dans tous les cas, lorsqu'un programme veut accéder au SC, le DOS appelle la fonction *0h* de l'interrupteur *1Ah* du BIOS. Cette fonction retourne le nombre de ticks depuis minuit sous la forme de 32 bits, les 16 bits de poids fort dans le registre CX et les 16 bits de poids faible dans le registre DX, et la valeur de *timer overflow flag* dans le registre AL. Si la valeur du registre AL vaut 1, c.-à-d. que 24h sont passées depuis minuit, alors le DOS augmente la date d'une unité.

Note : Comme la valeur du registre AL après l'appel de la fonction *0h* vaut soit 0 soit 1, dans le cas où l'ordinateur est allumé pendant plusieurs jours mais sans être utilisé, le DOS est incapable de déterminer correctement la date. Prenons l'exemple d'un ordinateur resté allumé pendant tout un week-end. Le lundi matin, en tapant la commande DATE, le système retourne la date du samedi au lieu de celle de lundi. En fait, quand le DOS appelle la fonction *00h* de l'interrupteur *1Ah* du BIOS, il obtient un nombre de ticks égal plus ou moins à 3146080 (du vendredi soir au lundi matin), et, par conséquent, le registre AL est mis à 1. Comme ce registre contient 1, le DOS augmente la date d'une unité (on passe du vendredi au samedi). Cette erreur a été corrigée dans Windows 3.1.

Conversion de ticks en heures

A partir du nombre de ticks stockés dans les registres CX et DX, le DOS calcule le temps selon la procédure illustrée sur la figure 1.8.

```
Heure := Nombre de ticks DIV 65543   /* division entière */
R := Nombre de ticks MOD 65543       /* reste de la division entière */
Minute := R DIV 1092
R := R MOD 1092
Seconde := R DIV 18.21
R := R MOD 18.21
centième  := R DIV 0.182
```

Figure 1. 8 : La conversion de nombre de ticks en nombre d'heures

Le résultat est affiché sous le format (dans le cas où la commande **time** est utilisée) : hh:mm:ss.pp (heures, minutes, secondes et centièmes de seconde).

Précision

Peut-on dire que la précision du SC est d'1 centième de seconde ? Non ! En effet, comme le chip *8253/8254 PIT* émet 18,206 "ticks" par seconde, chaque "tick" dure 1/18,206 s, c.-à-d. 0,05 seconde. Ainsi, la précision du SC est approximativement de 5 centièmes de seconde plutôt que de 1 centième de seconde.

Conversion de nombre d'heures du RTC en ticks

Pendant le démarrage, quand le BIOS envoie le temps du RTC au DOS, ce dernier le convertit en nombre de ticks en utilisant la formule de la figure 1.9.

Nombre de ticks := Heure * 65543 + Minute * 1092 + seconde * 18.21;

Figure 1. 9 : La conversion de nombre d'heures en nombre de ticks

Dans la formule illustrée sur la figure 1.9, le centième de seconde n'est pas utilisé car la précision du RTC est de 1 seconde. Concrètement, nous ne savons pas à quelle valeur le système initialise la partie "centième de seconde". Quand on utilise la commande **time** pour entrer le temps courant, les horloges SC et RTC sont mises à jour en même temps. La partie centième de seconde étant ignorée par le RTC, il peut y avoir une perte de temps pouvant aller jusqu'à 1 seconde. Prenons un exemple : après la saisie de la valeur 21:05:35.85, le SC contient exactement cette valeur alors que le RTC contient 21:05:35. On perd donc 85 centièmes de seconde.

Notes

1. Notons que le chip 8253/8254 PIT peut être programmé. Par exemple, les programmeurs des jeux le modifie dans le but d'accélérer l'animation graphique. Dès lors, le SC risque de ne plus fonctionner correctement si ces programmes ne remettent pas le chip à son état initial.

2. Dans un système trop chargé, il peut y avoir des ticks qui ne soient pas comptés; l'accumulation de ces ticks non comptés peut considérablement retarder le SC.

III. Le problème du hardware de l'an 2000

III.1. L'identification du problème

Dans quatre mois, nous allons entrer dans le 21e siècle. C'est peut-être un peu tard pour la discussion à propos du bogue de l'an **2000,** mais mieux vaut tard que jamais.

Supposons qu'on soit le 31/12/1999, 23:58:45. Le tableau 1.5 donne l'état des octets du CMOS associés au temps du RTC. Le tableau 1.6 illustre l'état de ces mêmes octets lorsque le système passe au 01/01/2000.

Adresse (Décimale)	Adresse (Hexadécimale)	Signification
0	00h	45
2	02h	58
4	04h	23
6	06h	5 (vendredi)
7	07h	31
8	08h	12
9	09h	99
50	32h	19

Tableau 1. 5 : Les octets du CMOS associés au RTC au 31/12/1999

Adresse (Décimale)	Adresse (Hexadécimale)	Signification
0	00h	00
2	02h	00
4	04h	00
6	06h	6 (Samedi)
7	07h	01
8	08h	01
9	09h	00
50	**32h**	**19**

Tableau 1. 6 : Les octets du CMOS associés au RTC lors du passage au 01/01/2000

Comme signalé plus haut, le 50ème octet qui concerne le siècle n'est pas mis à jour par le RTC. A ce moment, si les applications consultent le RTC, ils vont obtenir une valeur invalide c.-à-d. *00 :00 :00 01-01-**1900***. Par contre, si les applications interrogent l'horloge du système SC, ils vont obtenir une valeur correcte donc 00 :00 :00 01-01-**2000** parce que le SC fonctionne en parallèle avec le RTC. Mais le problème va apparaître après le prochain redémarrage de l'ordinateur.

En effet, lorsque l'ordinateur est redémarré, le BIOS passe le temps de RTC au système d'exploitation. Ce dernier va remarquer que ce temps n'est pas valide (car il n'accepte que le temps à partir de minuit 01-01-1980) et va donc initialiser le SC avec une valeur par défaut, soit souvent 00:00 :00,04-01-1980. Par conséquent, les applications qui utilisent le temps seront perturbées. C'est le bogue de l'an **2000.**

III.2. Les solutions

1. Une solution évidente est le remplacement du RTC actuel par un autre RTC qui supporte l'an 2000, c.-à-d. capable de mettre à jour le 50ème octet du CMOS. Pour l'instant, ce type de RTC est déjà lancé sur le marché par la société américaine *Dallas Semiconductor*.

2. La deuxième solution est la modification du BIOS. Cette modification exige que le BIOS soit de type *flashed BIOS*, c.-à-d., un BIOS modifiable. Après la modification, lorsque le BIOS est interrogé pour le temps, il exécute un algorithme simple pour corriger le temps si c'est nécessaire avant de l'envoyer au demandeur (voir figure 1.10). Toutefois, cet algorithme ne nous permet pas d'éviter le problème de l'an **3000.** En outre, cette solution n'est pas applicable pour les applications consultant directement le RTC. Cependant, cette solution est adoptée par la plupart de vendeurs des PCs pour deux raisons majeures :

a. le nombre des applications qui consultent directement le RTC est très très faible (voir figure 1.11) ;
b. le budget pour cette solution est beaucoup moins important par rapport à celui du remplacement du RTC.

```
Read RTC's date
if (year < 80)
   if (century < 20)
         {
                  century = 20;
                  Read RTC's date;
         }
return RTC's date
```

Figure 1. 10 : L'algorithme pour corriger la date invalide retourné par le RTC

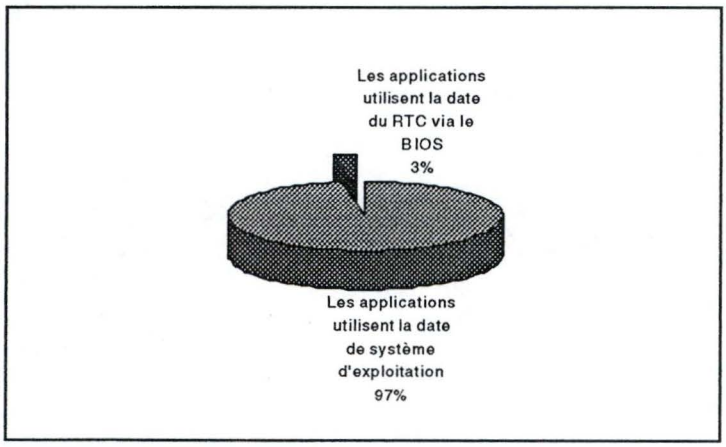


Figure 1. 11 : La répartition en pourcentage des applications en fonction de la manière d'accès à la date (source : Dell, échantillon : 1100 logiciels).

3. Pour les utilisateurs dont le BIOS de la machine est de type ROM BIOS, c.-à-d. que le BIOS n'est pas modifiable, un programme utilisant l'algorithme de figure 1.10 peut être installé. Mais, il faut s'assurer que ce programme soit exécuté avant toutes les applications qui ont besoin du temps.

IV. L'implémentation

Pour illustrer cette étude, nous avons écrit un programme en assembleur permettant de lire les temps de RTC et de SC. Pour le RTC, le temps et la date sont extraits soit directement via les portes *70h* et *71h* soit à l'aide des fonctions de l'interrupteur 1Ah du BIOS (voir tableau 1.3). Pour le SC, la date et le temps sont obtenus en appelant respectivement les fonctions *2Ah* et *2Ch* de l'interrupteur *21h* de DOS. Pour un ordinateur de type Cyrix P200+ allumé depuis 24 heures, le programme affiche un décalage de 6 à 7 secondes entre le RTC et le SC. Nous avons synchronisé deux montres (une Casio et une Seiko) à ces deux horloges du PC. Après 24 heures, les deux montres (Casio et Seiko) possèdent pratiquement le même temps que le RTC alors que le SC possède 6 à 7 secondes d'avance. Ce résultat peut-être expliqué par le changement de la fréquence du chip *8253/8254 PIT* (Dans le ordinateur il y a nombreux des jeux installés).

Nous avons utilisé le même programme sur un ordinateur de type Pentium 75 du pool MERCATOR de l'Institut d'Informatique. L'ordinateur est resté allumé pendant 24h. Après 24h, nous avons constaté que le SC est retardé de 9 à 10 secondes par rapport au RTC. Le résultat peut être expliqué par le charge de travail de l'ordinateur.

Un autre petit programme écrit en assembleur permet de modifier le BIOS DATA AREA en utilisant la fonction *00h* de l'interrupteur *1Ah* du BIOS. L'entrée de programme est le nombre de ticks depuis minuit. Après son exécution (au POOL MERCATOR), l'horloge du système prend une valeur qui correspond au nombre de ticks entré. Nous avons essayé d'exécuter le même programme au POOL MAGELLAN où Windows NT est installé, mis après exécution, la valeur de l'horloge du système reste inchangée : Windows NT ne permet pas à l'utilisateur de changer le temps de cette façon.

V. Conclusion

Dans les PCs (à partir des PC/ATs), il existe deux horloges qui marchent en parallèle : l'horloge du temps réel - RTC (Real Time Clock) - permettant de conserver le temps même si le PC est éteint, et l'horloge du système utilisée par le système d'exploitation pour ses tâches. Le RTC a une précision de l'ordre de 1 seconde, sa stabilité étant influencé par la température. Le SC a une précision d'ordre d'environ 0,05 seconde, sa stabilité pouvant être influencée par les logiciels exécutés ou la charge du système.

Chapitre 2 : La synchronisation des horloges dans un système distribué

Introduction

Dans un système non distribué, les problèmes de synchronisation entre processus peuvent être résolus en utilisant, par exemple, le sémaphore ou la technique des moniteurs. Si deux processus se synchronisent à l'aide d'un sémaphore, ils doivent être capables d'accéder à ce sémaphore (le sémaphore est stocké dans le noyau du système d'exploitation). Cette solution n'est pas convenable si les deux processus se trouvent sur deux machines différentes.

Nous pouvons aussi utiliser l'horloge pour synchroniser les processus. Dans un système non distribué, si le processus **P1** demande l'heure du système au temps **t1** et puis, le processus **P2** demande l'heure du système au temps **t2** alors, on peut dire que $t2 \geq t1$. C'est évident car P1 et P2 ont demandé l'heure d'une même horloge, et nous supposons que l'horloge du système fonctionne correctement. Dans le cas où P1 et P2 se trouvent sur deux machines différentes, nous ne pouvons pas dire que $t2 \geq t1$, car chaque machine a sa propre horloge et chaque horloge pouvant être influencée par des paramètres tels que le matériel utilisé lors de sa fabrication et la température ambiante. Si nous voulons utiliser les horloges pour synchroniser les processus dans un système distribué, ces horloges doivent être synchronisées entre elles-mêmes.

L'exemple suivant nous montre la nécessité de synchroniser les horloges dans un système distribué :

Exemple:

Dans la programmation, chaque fois le code source est modifié, nous devons le recompiler afin de produire un nouveau fichier objet reflétant ce changement. Si nous avons un code source volumineux (10.000 lignes, par exemple) et nous voulons ajouter une instruction, le compilateur doit recompiler le code source juste pour ce petit changement; ceci peut prendre un temps assez long. C'est pourquoi, nous divisons le code source volumineux en des petits modules. Si il y a un changement, le compilateur ne recompile que les modules qui sont plus récents que leurs fichiers d'objet correspondants.

Nous supposons que le éditeur et le compilateur se trouvent sur deux machines différentes (voir Figure 2.1).

Le fichier time.obj est crée à 14h32, plus tard le fichier time.c est modifié et mis à jour à 14h30 car l'horloge d'ordinateur B est retardée par rapport à celle de A. Conséquence : le compilateur ne recompile pas time.c car, pour lui, time.obj est plus récent que time.c, et le programmeur pourrait penser qu'il y a un problème dans le code source.

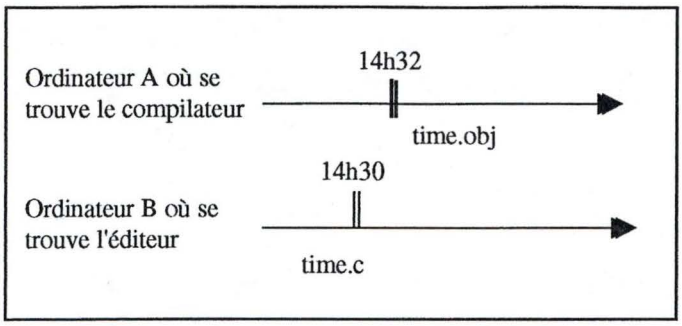


Figure 2. 1 : L'exemple

I.L'horloge logique[2]

I.1 La relation *happen – before*

Dans l'exemple précédent, les deux machines A et B ne doivent pas se synchroniser absolument : il n'est pas nécessaire que A et B indiquent exactement la même heure au temps t. Il suffit que A et B soient d'accord que time.c est mis à jour avant ou après la création de time.obj. Autrement dit, A et B doivent être d'accord sur l'ordre des événements. Cette synchronisation relative entre A et B est appelée la synchronisation entre horloges logiques.

La notion d'horloge logique a été proposée par Lamport en 1978. Pour synchroniser les horloges logiques, Lamport a défini une relation *happen - before*. L'expression *a -> b* se lit *a happens before b.*

Cette expression est valide :

- si a et b sont deux événements dans un même processus et a a lieu avant b (c'est trivial) ;
- si a est un événement indiquant qu'un message est envoyé par le processus P1 au processus P2, et b, un événement indiquant que ce même message est reçu par le processus P2. Ceci est vrai car il faut un certain temps pour que le message de P1 soit reçu par P2.

La relation *happen – before* est transitive :

Si a -> b et b -> c
alors a -> c

Si deux événements a et b ont lieu dans deux processus différents n'échangeant pas des messages entre eux, nous ne pouvons pas dire que *a -> b* est valide, ni b -> a est valide. Dans ce cas, Lamport dit que a et b sont en concurrence.

Nous appelons C(a) la valeur de l'horloge logique quand a lieu l'événement *a*.

[2] Ce texte est une interprétation du *Modern Operating System,* AndrewS.Tanenbaum, [20] et du *Cours du système d'exploitation,* Jean Ramaekers, [16].

Si a -> b
alors C(a) < C(b)

Nous ne pouvons pas reculer l'horloge logique pour ne pas perturber l'ordre des événements.

I.2. L'algorithme de synchronisation des horloges logiques

La Figure 2.2 illustre trois processus qui tournent sur trois machines distinctes(*alpha.be*, *beta.be* et *teta.be*). Chaque machine a sa propre horloge qui marche avec une fréquence constante. Les fréquences de trois horloge sont différentes.

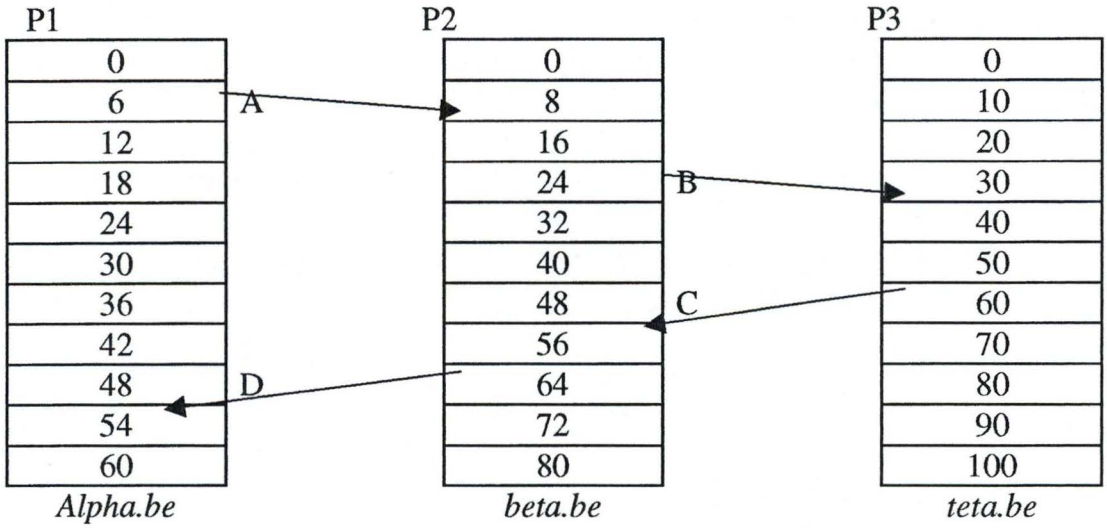


Figure 2. 2 : Horloges logiques non synchronisées

Au temps t=6 de l'horloge logique de la machine *alpha.be*, le processus P1 envoie un message A à P2. A est reçu par P2 au temps t=16 de la machine *beta.be*. Si A contient un timestamp qui indique qu'il a été envoyé au temps t=6 de la machine *alpha.be*, P2 conclut qu'il faut à A 10 ticks pour aller de P1 à P2. De même, P3 conclut qu'il faut à B 16 ticks pour aller de P2 à P3.

Au temps t=60 de la machine *teta.be*, P3 envoie un message C à P2. C arrive à P2 au temps t=56 de la machine *beta.be*. Au temps t=64 de la machine *beta.be*, P2 envoie un message D à P1. D arrive P1 au temps t=54 de la machine *alpha.be*. Nous ne pouvons pas accepter ces valeurs car si a -> b, C(a) doit être plus petite que C(b).

La figure 2.3 nous montre comment Lamport synchronise les horloges logiques.

Grâce à la relation *happen - before*, le message C est envoyé au temps t=60 de la machine *teta.be*, et peut être reçu par P2 au temps t>60. Quand P2 trouve que le temps fournie par l'horloge de *beta.be* est inférieur au timestamp dans C, il met à jour son horloge à 61. De même, P1 met à jour son horloge à 70 au moment où il reçoit le message D.

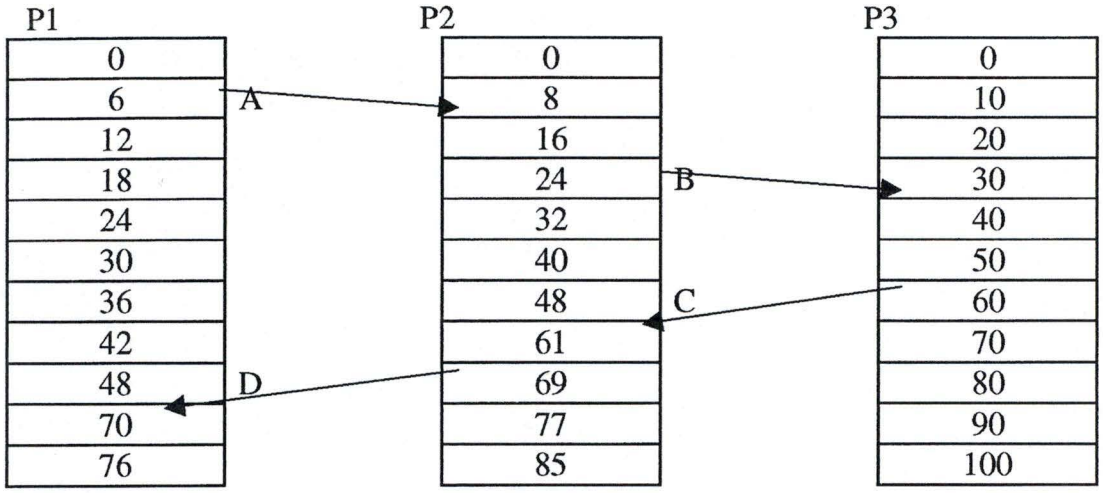


Figure 2. 3 : L'horloges logiques synchronisées

Si un processus envoie ou reçoit plus d'un message dans un même tick, il doit avancer son horloge d'un tick pour chaque message afin de satisfaire l'expression

$$a \rightarrow b \Leftrightarrow C(a) < C(b)$$

II. L'horloge physique

L'algorithme de Lamport permet de distinguer l'ordre des événements dans un système distribué. Le temps assigné à un événement n'est pas nécessairement égal au temps absolu où l'événement a eu lieu. Toutefois, dans un système à temps réel, la connaissance du temps absolu où a lieu un événement est très importante.

Exemple

Comment peut-on lancer 10 missiles qui sont situés à dix endroits différents le 01-09-2000 à 0:05:30 ? Il y a une seule solution : les horloges doivent être synchronisées absolument, c.-à-d., elles doivent indiquer le même temps à un moment donné. Cette synchronisation est appelée la synchronisation entre horloges physiques.

Avant d'aborder les algorithmes qui permettent de synchroniser les horloges physiques, nous allons introduire comment le temps est mesuré.

II.1 Universal Coordinated Time (UTC)

Tous les jours, on a l'impression que le soleil monte de l'Est, atteint la hauteur maximum, puis descend vers l'Ouest. Quand le soleil atteint la hauteur maximum, on parle de *zénith solaire*. Cet événement a lieu chaque jour, à midi plus ou moins. L'intervalle entre deux *zéniths solaires* successifs est appelé la *journée solaire*. Comme 24h équivaut à un jour et 3600 secondes à

une heure, une *seconde solaire* peut être définie comme 1/86400 de *journée solaire.*

En 1940, on a établi que la rotation de la terre au tour d'elle-même n'était pas constante. En effet, la terre tourne de plus en plus lentement au tour d'elle-même à cause de la friction de la marée. Les géologues ont prouvé qu'il y a 300 millions d'années, nous avons eu 400 jours par an. En plus de cette variation se déroulant sur le long terme, la longueur de la *journée solaire* est également influencée quotidiennement par les mouvements à l'intérieur de la terre. Pour diminuer ce dernier, les astrophysiciens mesurent plusieurs *journées solaires* puis calculent la moyenne avant de diviser par 86400. Le résultat est la *seconde moyenne solaire.*

Grâce à l'invention de l'horloge atomique en 1948, nous pouvons mesurer le temps d'une façon plus précise et indépendamment des mouvements de la terre. Les physiciens ont défini une seconde comme le temps nécessaire pour un atome de Césium 133 d'effectuer 9192631770 transitions, ce qui correspond à 1 *seconde solaire moyenne* en 1948.

Actuellement, il y a environ 50 laboratoires qui possèdent une horloge atomique. Chaque laboratoire envoie périodiquement le nombre de transitions de son horloge atomique depuis minuit 01-01-1958 au BIH (Bureau International de l'Heure, Paris). Le BIH calcule la moyenne de ces valeurs et la divise par 9192631770. Le résultat obtenu est appelé *TAI* (International Atomic Time).

Toutefois, malgré sa stabilité, *TAI* pose un problème sérieux : car comme la durée de la journée solaire devient de plus en plus longue, la journée TAI accumule une avance par rapport à celle-ci. De cette façon, la journée composée de 86400 secondes *TAI* est maintenant plus courte de 3 msec par rapport à la *journée solaire moyenne.*

Pour résoudre ce problème, BIH a introduit le *Universal Coordinated Time (UTC).* L'*UTC* est basé sur *TAI* mais lorsque un décalage de 800 msec existe entre ce temps UTC et le temps solaire, on va ajouter ou enlever d'*UTC* une seconde (*leap second*). Depuis 1958, 30 *leap secondes ont été* introduites dans *UTC.*

Aujourd'hui des ordinateurs (les serveurs du temps) équipés d'antennes spéciales peuvent capturer l'*UTC* pour synchroniser leur horloge. L'*UTC* est émis périodiquement à partir des stations de radio ou à partir des satellite. Il faut un certain temps pour que le signal UTC voyage de stations de radio ou de satellites aux serveurs du temps. Si on connaît la vitesse de signal et la distance entre la station radio, le satellite et le serveur du temps on peut calculer ce délai. Malheureusement, la vitesse du signal est influencée par la condition atmosphérique, et la distance entre le satellite et le serveur du temps est variable.

La tableau 2.1 indique la précision de l'*UTC* reçu des sources différentes.

L'UTC reçu de	La Précision
Station de Radio	0,1 – 10 ms
Satellite GEOS	0.1 ms
Satellite GPS	1ms

Tableau 2. 1 : la précision de l'*UTC* reçu de sources différentes

Comme le prix de l'antenne est relativement cher par rapport à celui d'un ordinateur, ceux qui ne sont pas équipés d'une antenne peuvent se connecter à un serveur du temps pour synchroniser indirectement leur horloge avec l'*UTC*.

II.2. L'algorithme de synchronisation des horloges physiques

Dans cette section, nous discutons des algorithmes qui permettent à un ordinateur de synchroniser son horloge avec celle du serveur du temps.

II.2.1 L'algorithme de Cristian

Si un processus P demande le temps à un serveur S en envoyant le message m_r et reçoit la valeur du temps t dans un message m_t, alors en principe il doit mettre à jour son horloge à la valeur :

$$t + T_{trans}$$

où T_{trans} est un intervalle de temps nécessaire pour m_t voyager de S à P. Malheureusement T_{trans} est variable car en général il y a des processus qui sont concurrents avec P et S et des messages qui sont concurrents avec m_t. (voir Figure 2.4)

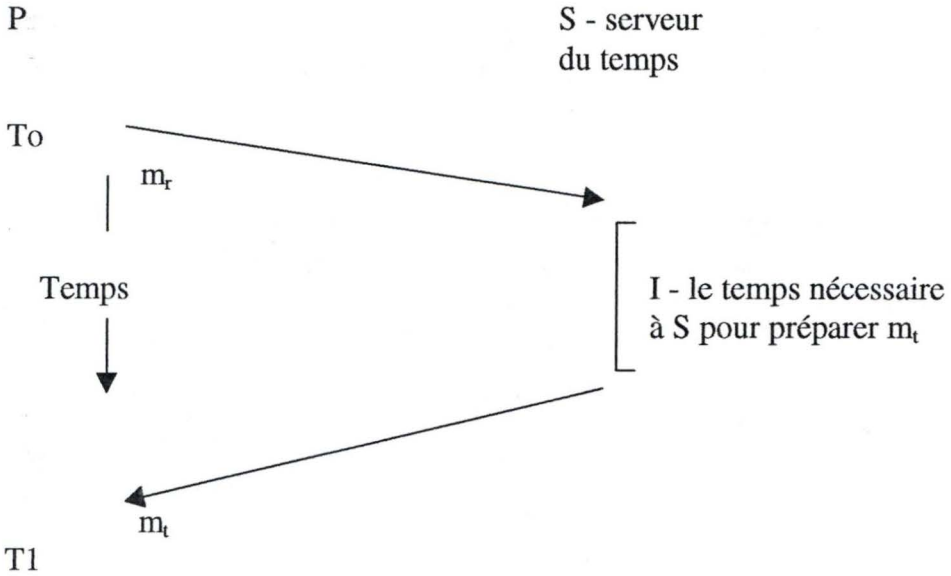


Figure 2. 4: L'algorithme de Critian

Une façon simple pour estimer le temps auquel P met à jour son horloge est $t + T_{round}/2$ où $T_{round} = T_1 - To$ est un intervalle de temps pour envoyer m_r et recevoir m_t (avec une hypothèse : le temps est divisé également avant et après que S mette t dans m_t).

Cette estimation est améliorée si I (le temps nécessaire à S pour préparer m_t) est connu. Dans ce cas :

$$T_{round} = T_1 - To - I$$

Soit min+x est le temps nécessaire à m_r pour aller de P à S et min+y est celui nécessaire à m_t pour aller de S à P . x et y sont nul ou positives.

avec min étant la valeur obtenue quand il n'y a que S et P qui sont exécutés et il n'y a que m_r et m_t qui circulent dans le réseau.

Le moment le plus tôt où S peut mettre t dans m_t est min après que P ait envoyé m_r et le moment le plus tard où S peut le faire est min avant que m_t arrive à P. Donc t est dans un intervalle T_{round}-2min alors l'exactitude de t est $\pm$ ($T_{round}/2$ -min).

Si S est en panne c'est impossible de faire la synchronisation. Pour résoudre ce problème, il faut plusieurs serveurs du temps qui communiquent avec la source UTC. Quand un client veut le temps il envoie les messages à tous les serveurs et prend la réponse qui arrive le plus tôt .

II.2.2 L'algorithme de Berkeley

L'algorithme est décrit par Gusella et Zatti [1989], il convient bien pour synchroniser le temps entre les machines quand il n'existe pas une source de temps extérieur.

Dans cet algorithme le serveur est actif. Périodiquement, il interroge les autre machines. A partir des réponses reçues, en tenant compte du temps de transfert, il calcule un temps moyen avec sa propre horloge y comprise. Plutôt que de renvoyer ce temps moyen, le serveur calcule la différence éventuelle de chaque machine par rapport au temps moyen et renvoie ces valeurs. De cette façon, chaque machine peut ajuster son horloge sans se préoccuper des temps de propagation toujours aléatoires.

Pour que le temps moyen calculé ne soit pas influencé par des temps aberrants venus de certaines machines, il faut éliminer les valeurs suspectes.

Après un délai raisonnable, si le serveur ne reçoit pas toutes les réponses il procédera à l'évaluation sur bases des réponses reçues.

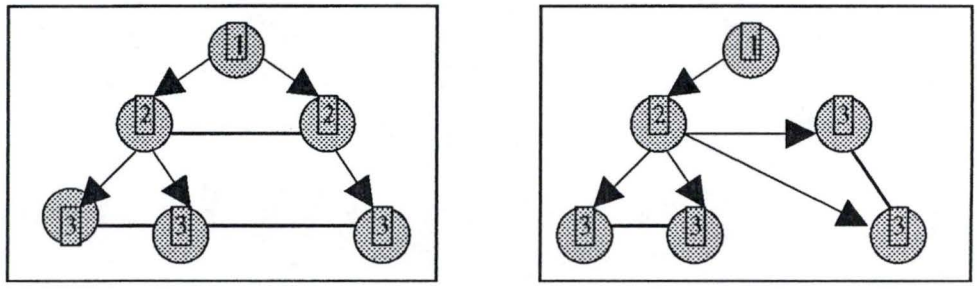
II.2.3 NTP (Network Time Protocol – RFC 1305)

NTP est un protocole très complexe, pour l'implémenter on a écrit plus ou moins 80,000 lignes de code. Aujourd'hui NTP est adopté comme le standard pour la synchronisation d'horloges dans les systèmes distribués. Dans ce travail nous ne nous intéresserons qu'aux mécanismes utilisés par NTP pour estimer le timestamp. Pour plus de détails veuillez vous référer à RFC 1305 [6].

Les fonctionnalités principales de NTP sont :

- Permettre à tout client sur l'Internet d'être synchronisé précisément avec UTC.
- Fournir un service fiable au client.
- NTP utilise les techniques d'authentification pour vérifier si le timestamp provient d'un serveur auquel on peut faire confiance.

Dans NTP l'ensemble des serveurs qui se trouvent sur l'Internet forment un subnet de synchronisation. Les serveurs primaires se connectent directement avec une source de temps (par exemple : horloge d'une station de radio). Les serveurs secondaires sont synchronisés avec ces serveurs primaires ...etc. . Le figure 2.5a illustre ce réseau ;



a b

Figure 2. 5 : subnet de synchronisation

Le serveur primaire est numéroté 1, il occupe stratum 1. Les serveurs secondaires se trouvent à stratum 2 ... etc. L'horloge des serveurs avec un numéro de stratum élevé est moins précis que celle des serveurs avec petit numéro de stratum car les erreurs sont introduites à chaque niveau de stratum.

Les serveurs communiquent avec les serveurs de stratum moins élevé pour définir leur horloge (représenté par une flèche) , et avec des serveurs de stratum de même niveau (représenté par une ligne).

Quand un serveur est en panne les autres peuvent reconfigurer leurs liaisons automatiquement, ce qui permet de fournir un service fiable au client. (voir figure 2.5b)

Les serveurs du temps peuvent fonctionner en 3 modes :

- Le mode *Multicast* est utilisé pour LANs à grande vitesse et où la meilleure exactitude n'est pas demandée. En ce mode , un ou plusieurs serveurs diffusent le timestamp; les machines fonctionnant en mode de client captent le timestamp avec une supposition que le délai est d'ordre de quelques ms.
- Le mode *Procedure-call* est similaire à algorithme de Cristian. En ce mode, un serveur accepte les demandes venant des autres ordinateurs et puis leur réponds avec le timestamp. Ce mode est convenable pour fournir une meilleure précision (que le cas précédent) ou quand le matériel hardware ne permettent pas le mode multicast.
- Le mode *symétrie* est utilisé entre les différents serveurs du subnet de synchronisation pour échanger leur timestamp.

Dans tous les 3 modes, NTP utilise UDP (User Datagram Protocol) pour diminuer le délai, et simplifier l'implémentation.

En 2 modes Procedure-call et symétrie, un couple de messages sont échangés. Quatre timestamps T_{i-3}, T_{i-2}, T_{i-1}, T_i indiqués dans le figure 2.6 pour les messages m, m' échangés entre 2 serveurs A et B.

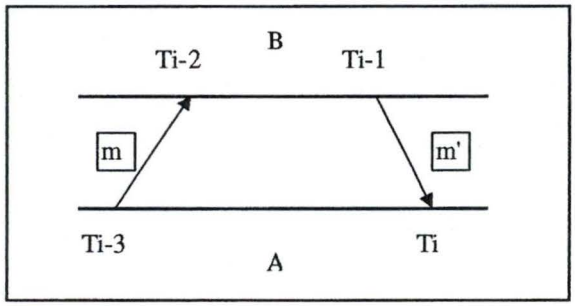


Figure 2. 6 : L'Echange de messages en Procedure-call mode

Pour chaque couple de messages, NTP calcule un offset O_i qui est une valeur estimée de l'écart entre deux horloges. Il calcule aussi le délai D_i qui est le temps total pour transmettre deux messages.

Soit O est l'écart réel entre les deux horloges; t et t' sont le temps nécessaires pour transmettre m et m', respectivement. Alors :

$T_{i-2} = T_{i-3} + t + O$ et $Ti = T_{i-1} + t' - O$ (1)
soit $a = T_{i-2} - T_{i-3}$ et $b = T_{i-1} - T_i$
$D_i = t + t' = a-b$
A partir de équ (1) $\Rightarrow t = a - O \geq 0 \Rightarrow a \geq O$
et $t' = O - b \geq 0 \Rightarrow O \geq b$
$b \leq O \leq a \Rightarrow (a+b)/2 - (a-b)/2 \leq O \leq (a+b)/2 + (a-b)/2$
$\Rightarrow O_i - D_i/2 \leq O \leq O_i + D_i/2$
avec $O_i = (a+b)/2$

Le timestamp contient T_{i-3}, T_{i-2}, T_{i-1} tandis que T_i est calculé à l'arrivée du m'. O_i est appelé estimation de O.

Etant donné un échantillon (O_i, D_i), i = 0..n; Comment peut-on choisir le meilleur O_i ? Le meilleur O_i est celui correspondant au Di le plus petit. L'auteur a introduit un algorithme de Data-filtering qui garde 8 couples récents de (O_i, D_i) ensuite ils sont mis dans une liste qui est classé en ordre croissant de D_i. Le premier élément de la liste , c.-à-d.,(O_o,D_o) est choisi comme l'estimateur.

Le *filter dispersion* ε est défini comme $\varepsilon = \sum_i | O_i - O_o| v$ où i = 0 .. n-1 (dans notre cas n = 8), v est un facteur expérimental.

L'algorithme Peer-Selection a également été introduit pour déterminer la qualité de données échangées entre les serveurs : Grace aux expériences, quand la meilleure fiabilité est demandée, les couples (O_i, D_i) venant d'un serveur de stratum bas et petit *filter dispersion* sont favorisés. Quand la meilleure stabilité est demandée les couples venant d'un serveur de stratum bas et peu éloignés sont favorisés.

III. Implémentation de *STPC* – Synchroniseur du temps pour PC

Pour illustrer la partie précédente nous avons développé STPC (Synchroniseur de Temps pour PC) ; il s'agit d'une application qui permet aux PC de synchroniser son horloge avec les serveurs du temps ou avec les autres PCs.

STPC est développée en language C pour l'environnement Window9x en utilisant Windows Sockets Application programming interface (WSA) et TCP/IP suites. Elle peut jouer le rôle de client et de serveur en même temps.

Dans le cadre d'un réseau Intranet, STPC peut être installé sur les FireWalls et permet aux ordinateurs de synchroniser leur horloge avec les sources du temps à l'extérieur. Cependant, pour accomplir parfaitement cette opération il faut ajouter encore dans STPC les fonctions sécurisantes qui lui permettent de vérifier l'identité de la source du temps. En adaptant cette application à WindowsNT, elle pourrait être utilisée pour fixer un problème de sécurité de WindowsNT que nous avons remarqué en effectuant ce mémoire. En effet, l'administrateur de WindowsNT peut décider si un utilisateur a le droit de changer le temps du système ; mais même si l'utilisateur n'a pas ce privilège il peut toujours le faire en changeant la RTC (Real Time Clock) et redémarrant l'ordinateur. Car comme précisé dans le Chapitre 1, l'horloge du système prend la valeur de la RTC comme sa valeur initiale. Pour résoudre le problème nous pouvons activer STPC au démarrage de WindowsNT, ainsi STPC va synchroniser l'horloge du système de l'ordinateur avec WindowsNTServer ou avec d'autres sources du temps.

Avant de décrire le fonctionnement de STPC, nous abordons brièvement le WSA et le modèle de client-serveur.

III.1 Le WSA - Windows Sockets Application programming interface

En 1991 Microsoft Windows a commencé à jouer un rôle important dans le domaine de système d'exploitation pour PC. Cependant il y a eu un problème pour l'interface entre Microsoft Windows et TCP/IP : chaque fournisseur de TCP/IP pour PC a créé son propre interface de programmation pour TCP/IP; par conséquent c'était très difficile de développer une application de réseaux qui puisse fonctionner avec plusieurs implémentations de TCP/IP. C'est pourquoi, le 20 janvier 1993, la spécification de la version 1.1 de Windows Sockets ou WinSock a été réalisée. Elle établissait un standard pour l'interface de la programmation de réseaux pour Microsoft Windows.

Les applications développées en utilisant WinSock sont compatibles : il ne faut donc aucun changement quand l'application est exécutée ou recompilée sur les implémentations différentes de TCP/IP pour Microsoft Windows.

Le Windows Sockets API est dérivé de Berkeley Sockets API qui a été implémenté dans la version 4.3 de Berkeley Software Distribution (BSD4.3). BSD Sockets est considéré largement comme un standard de l'interface de programmation pour TCP/IP, par conséquent il y a déjà beaucoup de programmes qui ont été écrit pour Berkeley Sockets. La plus part de source code pour Berkeley Sockets peut être utilisée pour Windows Sockets (car le Windows Sockets est très proche le Berkeley Sockets,) en ajoutant seulement la partie de l'interface graphique pour utilisateur de Microsoft Windows. (voir Figure 3.1)

Le modèle de Windows Sockets est composé de trois parties :l' application de Windows Sockets, Windows Sockets API (WSA) et le système de réseaux (voir Figure 3.2). L'application de Windows Sockets utilise les services de système de réseaux en appelant le WSA pour envoyer et recevoir l'information.

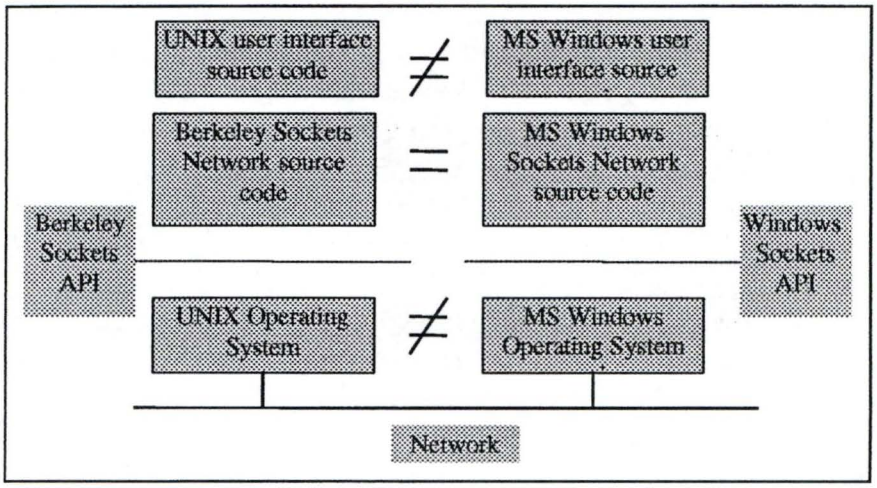


Figure 3. 1 : La portabilité de Windows Sockets

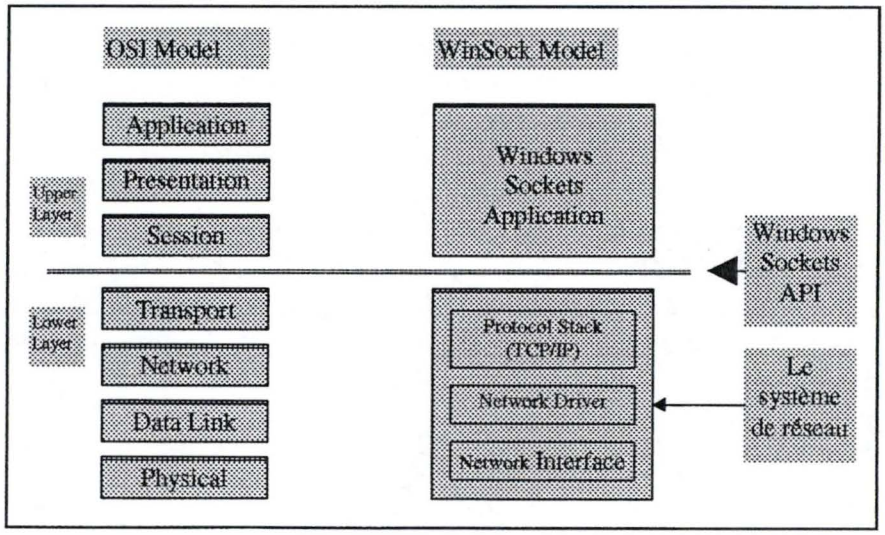


Figure 3. 2 : Model de Windows Sockets par rapport à Model d'OSI

III.1.1 Le modèle de Client - Serveur

L'application réseau peut jouer le rôle de client et/ou de serveur. D'abord l'application serveur est exécutée et attend de recevoir une demande, ensuite une application client est exécuté et envoie une demande au serveur. Après que le contact soit établi, le client et le serveur sont capables d'envoyer et recevoir les données. Chaque connexion entre client et serveur correspond à un couple de socket (un socket est identifié par : adresse IP et port de communication).

Pour que deux sockets communiquent comme client et serveur, tous les deux doivent avoir le même type de socket soit stream(TCP) soit datagram(UDP). Le socket du client doit connaître l'identité de socket du serveur.

Lorsque le socket du client réussit à se connecter à celui du serveur, une association est créée. Cette association contient cinq éléments :

- Le type de socket : TCP/UDP (doit être le même pour le client et le serveur)
- Le IP addresse de client.
- Le numéro de port de client.
- Le IP addresse de serveur.
- Le numéro de port de serveur.

Les tableaux 3.1 et 3.2 illustrent cinq étapes à faire pour toutes les applications de client–serveur.

Etape	Client	Serveur
Ouvrir un socket	Socket()	Socket()
Nommer le socket	Initier le structure sockaddr_in avec l'identité de serveur	Initier le structure sockaddr_in avec l'identité de serveur Bind() Listen()
Créer l'association entre deux sockets	connect()	Accept()
Envoyer et recevoir les données entre sockets	send() recv()	Recv() Send()
Fermer le socket	Closesocket()	Closesocket()

Tableau 3. 1 : l'application de réseaux TCP

Etape	Client	Serveur
Ouvrir un socket	Socket()	Socket()
Nommer le socket	Initier le structure sockaddr_in avec l'identité de serveur	Initier le structure sockaddr_in avec l'identité de serveur Bind()
Créer l'association entre deux sockets et Envoyer et recevoir les donnés entre sockets	sendto() recvfrom()	Recvfrom() Sendto()
Fermer le socket	Closesocket()	Closesocket()

Tableau 3. 2 : l'application de réseaux UDP

III.1.2 Les modes d'opération

Windows Sockets a trois modes d'opération distingués : le mode *bloqué*, le mode *non bloqué* et le mode *asynchrone*.

- Dans le mode *bloqué*, une fonction de Windows Sockets ne se termine que quand l'opération est complétée; la valeur retournée soit succès soit échec. Dans ce mode l'application ne peut pas effectuer plus qu'une opération de réseaux en même temps.
- Dans le mode *non bloqué*, une fonction de Windows Sockets se termine immédiatement; la valeur retournée soit succès soit échec, mais dans ce cas la valeur échec n'est pas nécessairement mauvaise. L'erreur indique peut-être que le WinSock DLL a commencé l'opération mais elle n'est pas complétée à ce moment. Le problème avec le mode *non bloqué* est que l'application doit appeler une fonction à maintes reprises pour compléter une opération ou détecter sa complétion, par conséquent la performance du système est diminuée.
- Le mode *asynchrone* est *non bloqué* car une fonction de Windows Socket se termine avant que l'opération ne soit complétée. Mais Winsock DLL envoie à l'application un message qui indique la complétion ou l'échec d'une opération. Ce mode compense le désavantage des deux modes précédents mais le code source *asynchrone* n'est pas compatible avec Berkeley Sockets.

Les trois modes d'opération de Windows Sockets correspondent exactement à l'analogie suivante. On téléphone à une personne et découvre qu'elle n'est pas disponible immédiatement, on a trois possibilité :

- attendre sans raccrocher jusqu'à ce que la personne vienne répondre. (mode *bloqué*)
- Raccrocher et appeler plus tard. (mode *non bloqué*)
- Laisser un message et la personne nous téléphone plus tard. (mode *asynchrone*)

Dans l'impémentation du STPC, nous avons utilisé le mode asynchrone.

Le Figure 3.3 illustre trois modes de opérations de Windows Sockets.

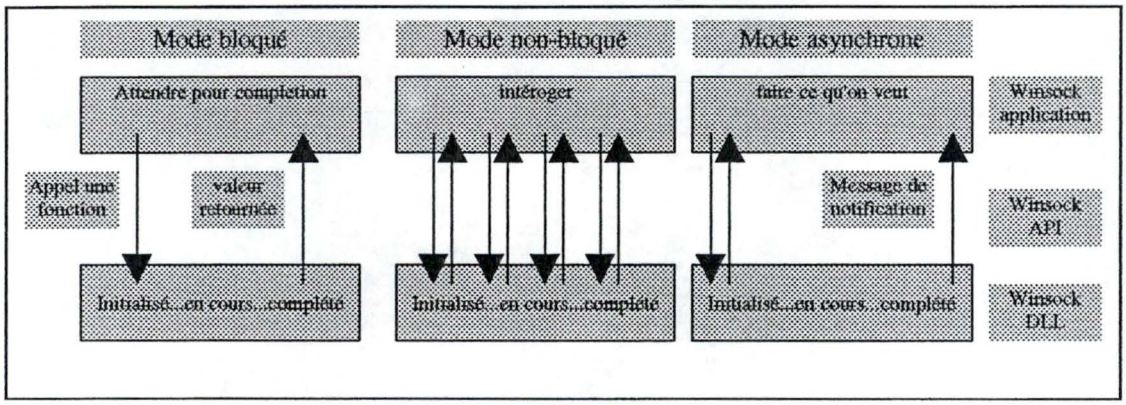


Figure 3. 3 : Les modes d'opérations de Windows Sockets.

III.3 STPC

III.3.1. STPC dans le rôle de client

Dans le rôle de client STPC supporte Daytime protocol, Time protocol et VSNTP (Very Simple Network Time Protocol) ; elle utilise le TCP ou le UDP pour communiquer avec le serveur(voir Figure 3.4). L'utilisateur peut configurer STPC pour qu'elle travaille en mode manuel ou en mode automatique. (voir Figure 3.5)

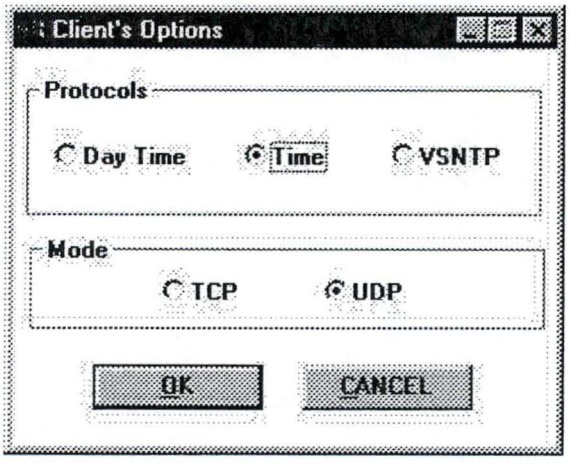


Figure 3. 4 : STPC dans le rôle du client

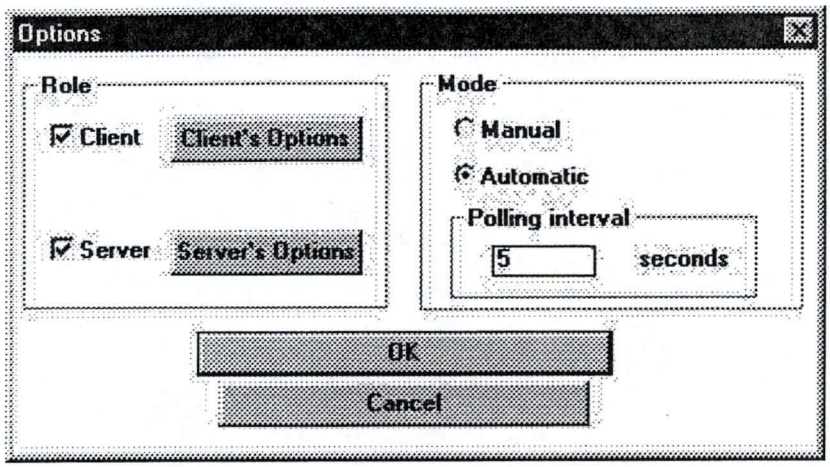


Figure 3. 5 : La fenêtre Options de STPC

- En mode manuel, l'utilisateur doit fournir à STPC le nom du serveur ou son adresse IP, le protocole préféré (Daytime,Time, ou VSNTP) et le type de communication (TCP ou UDP). Lorsque la réponse du serveur arrive, l'utilisateur a alors la possibilité de mettre à jour son horloge grâce au timestamp contenu dans le message, STPC tient compte du délai de transfert du timestamp entre le serveur et le client.
- En mode automatique, STPC communique périodiquement avec un serveur par défaut et automatiquement met à jour son horloge. La fréquence de communication dépend du 'clock drif'. Par exemple si l'horloge a un 'clock drif' en avance de 500 ms à chaque 60s et l'utilisateur désire que l'écart entre son horloge et celle du serveur de temps ne dépasse pas une seconde, alors il doit resynchroniser son horloge toutes les 2 minutes.

Si l'horloge est retardée par rapport au timestamp reçu, STPC avance simplement l'horloge à DELTA, avec DELTA= Ts + Es.

Où Ts est le timestamp.
Es est l'estimation du temp nécessaire au transfert du message entre le serveur et le client.

Si l'horloge est avancée par rapport au timestamp reçu, le problème devient plus compliqué : STPC ne peut pas reculer tout à coup l'horloge à DELTA car cela peut mener à une confusion de l'ordre de événement dans le système. Prenons un exemple.

<u>Exemple:</u>

Supposons que nous avons un fichier time.c qui est mis à jour à 19h puis compilé pour former le fichier time.obj à 19h01. À 19h02 nous synchronisons notre horloge avec un serveur du temps et supposons que notre horloge est avancée de 5 minutes par rapport à celle du serveur, lorsque le message de serveur arrive notre horloge est mise tout à coup à 18h57 (avec une hypothèse : le roundtrip est négligé). De nouveau time.c est mis à jour à 18h59

mais cette fois le compilateur ne le compile pas car time.obj est plus « récent » que time.c .

Le problème peut être résolu en ralentissant l'horloge jusqu'au moment où elle porte la même valeur que celle de l'horloge du serveur. En effet l'UNIX et WindowsNT nous fournissent les fonctions qui permettent de changer la vitesse de l'horloge du système ; mais comme STPC est développée pour Windows 9x nous ne bénéficions pas de ces services, notre solution est de bloquer l'horloge du système jusqu'au moment où elle est synchronisée avec celle du serveur.

III.3.2. STPC dans le rôle de serveur

Dans le rôle de serveur STPC supporte aussi Daytime,Time protocols et VSNTP en même temps, elle peut utiliser TCP et/ou UDP pour communiquer avec les clients. STPC peut s'occuper simultanément de plusieurs connections : chaque fois qu'elle accepte une connexion, elle insère dans une liste chaînée l'enregistrement qui contient l'information concernant cette connexion. Lorsque la connexion est terminée, l'enregistrement qui la concerne est supprimé de la liste chaînée.

Quand le serveur reçoit une requête d'un client, le serveur interroge sa propre horloge, et met la valeur reçue (timestamp) dans le message et l'envoie immédiatement au client.

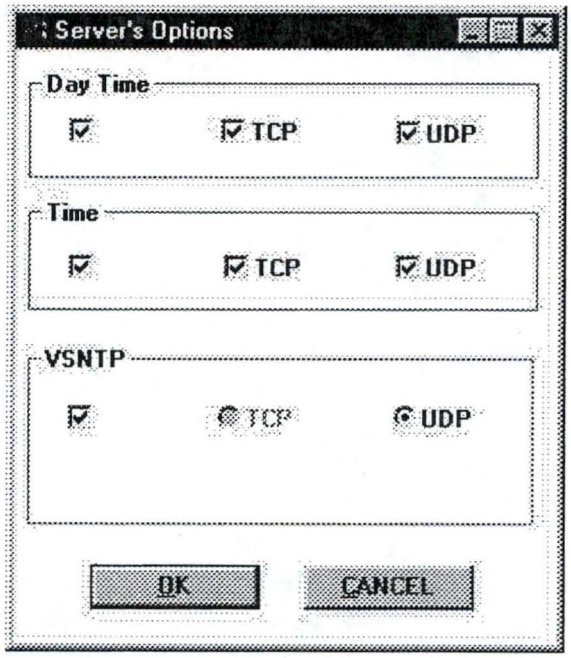


Figure 3. 6 : STPC joue le rôle du serveur

Maintenant nous discutons en détail comment STPC supporte Daytime, Time et VSNTP protocols.

III.3.3. Les protocols Time, Daytime et VSNTP

III.3.3.1 Daytime protocol (RFC 867)

C'est un protocole simple pour échanger le temps dans un système distribué. Il est installé sur la plupart des machines UNIX. Le Daytime serveur renvoie au Daytime client sa date et l'heure courante sous la forme d'une chaîne de caractères.

Quand il est utilisé via TCP, le STPC serveur écoute sur le port 13, le STPC client se connecte à ce port. Si le serveur est incapable de déterminer le temps de son site, il doit refuser la connexion ou la fermer et ne renvoie rien. Si la connexion est établie le serveur renvoie sa date et l'heure courante et ferme la connexion. Le client reçoit la réponse et ferme la connexion.

Quand il est utilisé via UDP, le STPC serveur écoute sur le port 13, le STPC client envoie un datagram vide à la ce port. Quand le datagram arrive si le serveur est incapable de déterminer le temps de son site, il ne répond pas. Si non le serveur renvoie un datagram qui contient sa date et l'heure courante .

Il est possible que les messages envoyés par le STPC client ou serveur soient perdus, ou que le serveur ne soit pas disponible. C'est pourquoi le STPC client démarre un timer avant l'envoi de la demande . Si il reçoit une réponse avant l'expiration du timer il va détruire le timer. Par contre, s'il ne reçoit rien, il redémarre le timer et envoie de nouveau la demande. Si STPC client ne reçoit pas la réponse après la 3e expiration du timer il va arrêter ou tenter de se connecter à un autre serveur.

Dans la spécification de Daytime protocol il n'existe pas de mécanisme pour compenser le temps nécessaire au transfert des messages entre le serveur et le client. Nous l'estimons par Tround/2 avec T_{round} étant l'intervalle du temps entre le moment où le client envoie sa demande et le moment où il reçoit la réponse.

La précision du timestamp de Daytime protocol est de l'ordre de la seconde. Il est acceptable pour la plupart des applications. Il est par contre conseillé d'utiliser VSNTP pour une précision de timestamp de l'ordre des centièmes de seconde.

La spécification de Daytime protocol dit simplement que le serveur doit renvoyer sa date et l'heure courante sous forme d'une chaîne de caractères. Donc dans les implémentations différentes, les timestamps sont souvent différents. Par exemple :

Monday, August 02, 1999 12:33:50-PST
02 AUG 99 12:33:50 PST
50419 99-08-02 12:33:50 00 0 0 50.0 UTC(NIST)

C'est pourquoi quand ce protocole est utilisé, le STPC client permet seulement de consulter le temps de serveur mais il ne permet pas de synchroniser automatiquement son horloge avec celle du serveur.

III.3.3.2.Time protocol (RFC 868)

C'est un protocole simple pour échanger le temps dans un système distribué. Il est installé sur la plupart des machines UNIX. Le Time serveur renvoie au Time client le nombre de secondes de son horloge depuis minuit 01-01-1900 sous la forme d'une valeur codée sur 32 bits. C'est le client qui convertit le nombre de secondes reçu en forme lisible pour l'utilisateur.

Quand le STPC supporte ce protocole son fonctionnement est le même que pour le Daytime protocole à la différence que STPC Time serveur écoute sur le port 37 et le STPC Time client peut être synchronisé indépendamment de l'implémentation du Time serveur car le format du timestamp est unique.

La précision du timestamp de Time protocol est de l'ordre de la seconde. Il est acceptable pour la plupart des applications. Il est par contre conseillé d'utiliser VSNTP pour une précision de timestamp de l'ordre des centièmes de seconde.

III.3.3.3. VSNTP (Very Simple Network Time Protocol)

Dans les deux protocoles précédents, la précision du timestamp est de l'ordre de la seconde, et un mécanisme simple est utilisé pour estimer le temps de transfert du message entre le serveur et le client.

Dans VSNTP nous augmentons la précision du timestamp de l'ordre des centièmes de seconde, et appliquons le mécanisme décrit dans la section II.2.3 pour estimer l'écart entre l'horloge du serveur et du client.

Quand le protocole est utilisé, STPC ne fonctionne que sous le mode UDP. Le STPC serveur écoute sur le port 2123. Le client envoie sa requête au serveur, en y introduisant un timestamp (T_{i-3}) correspondant au moment juste avant l'envoi de celui-ci (voir figure 2.6). Le serveur quand il reçoit ce message va y ajouter immédiatement la valeur de son horloge à ce moment (T_{i-2}). Lorsque le serveur sera prêt à renvoyer le message, il va encore y ajouter la valeur de son horloge à ce moment (T_{i-1}), et l'envoie immédiatement. Lorsque le message est de retour au client, celui-ci interroge son horloge et obtient la valeur de T_i, grâce à T_i, T_{i-1}, T_{i-2}, T_{i-3}, le client est capable (selon la méthode présentée à la section II.2.3) d'estimer l'écart de son horloge avec celle du serveur. Dans ce protocole, le timestamp est le nombre de secondes depuis minuit 01-01-1900 sous la forme d'une valeur codée sur 64 bits.

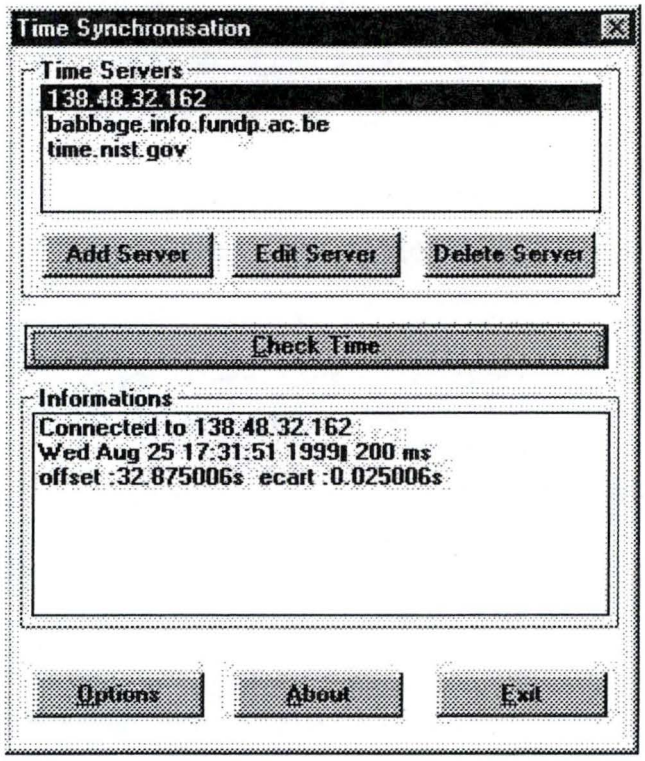


Figure 3. 7 : STPC client supporte VSNTP

Le figure 3.7 présente un STPC client synchronize son horloge avec un STPC serveur qui se trouve sur la machine 138.48.32.162. Le temps (T_{i-1}) du serveur est 17h :31m :51s et 200ms. Dans ce cas l'horloge du client est en retard par rapport à celle du serveur de 32.875006s ± 0.025006s.

III.3.4 Le résultat.

Pendant l'implémentation du STPC, nous avons remarqué que le temps nécessaire au message pour aller de client au serveur et retourner au client (le roundtrip time) en utilisant UDP est aux environs de moitié de celui en utilisant TCP. Par exemple, le temps d'aller et retour de 138.48.32.162 (un PC de l'institut d'informatique) à *time.nist.gov* est environ 160 millisecondes pour UDP et environ 340 millisecondes pour TCP. Cette différence peut être expliquée par le temps nécessaire à l'application pour établir une connexion dans le cas de TCP.C'est pourquoi, nous avons choisi UDP pour VSNTP.

Quand STPC supporte le Time protocol, il permet que les ordinateurs (12 ordinateurs dans le POOL MECATOR de l'institut d'informatique) se synchronisent d'ordre de seconde. Avec VSNTP l'écart entre horloges des ordinateurs est environ 0.05 seconde. Notons que la précision de l'horloge du système est approximativement de 0.05 seconde (voir **II.2**).

IV. Conclusion

Dans ce chapitre nous avons étudié le problème de la synchronisation du temps dans un système distribué. Ensuite, les algorithmes pour la synchronisation ont été envisagés. Il y a deux types de l'algorithme :

1. L'algorithme permet l'application distribuée d'être d'accord avec les autres sur l'ordre des événements. Dans ce cas on dit que l'algorithme permet de synchroniser les *horloges logiques.*
2. L'algorithme permet l'application distribuée d'être d'accord avec les autres sur l'heure exacte des événements. Dans ce cas on dit que l'algorithme permet de synchroniser les *horloge physiques.*

Pour illustrer ce chapitre nous avons développé une application, c.-à-d., STPC (Synchroniseur du Temp pour PC) qui permet de synchroniser les ordinateurs en appliquant les algorithmes de type 2 (*l'horloge physique*). Avec STPC, l'ecart entre horloges des ordinateurs (12 ordinateurs dans le POOL MECATOR de l'institut d'informatique) est approximativement de 0.05 seconde.

Conclusion

Ce mémoire est dans le but d'étudier la gestion du temps dans les ordinateurs et les systèmes distribués.

I. Gestion du temps dans les ordinateurs

Dans le *chapitre 1*, nous avons discuté de la gestion du temps dans un PC isolé. Dans les PCs (à partir des PC/ATs), il existe deux horloges qui marchent en parallèle : l'horloge du temps réel - RTC (Real Time Clock) - permettant de conserver le temps même si le PC est éteint, et l'horloge du système – SC (System Clock) utilisée par le système d'exploitation pour ses tâches. Le RTC a une précision de l'ordre de 1 seconde, sa stabilité étant influencé par la température. Le SC a une précision d'ordre d'environ 0,05 seconde (dans le cas de MS-DOS), sa stabilité pouvant être influencée par les logiciels ou la charge du système.

Grâce à cette étude, nous pouvons comprendre de manière approfondissement le problème hardware de l'an 2000.

Pour illustrer ce chapitre, nous avons écrit un programme en assembleur permettant de lire les temps de RTC et de SC. Avec ce programme nous avons remarqué un écart de 6 à 10 secondes entre RTC et SC d'un PC allumé pendant 24h.

Cependant, cette étude est limitée dans le cas du PC et le système d'exploitation MS–DOS. Nous allons continuer la recherche dans les autres architectures d'ordinateur et système d'exploitation.

II. Gestion du temps dans les systèmes distribués

La synchronisation du temps dans un sytème distribué est non trivial, car pour se synchroniser les ordinateurs doivent échanger les messages contenant le timestamp. Mais le temps de transfert des messages n'est pas stable à cause du charge du réseau et celui de chaque ordinateur . Les algorithmes pour résoudre ce problème sont envisagés. Il y a deux types de l'algorithme :

1. L'algorithme permet l'application distribuée d'être d'accord avec les autres sur l'ordre des événements. Dans ce cas on dit que l'algorithme permet de synchroniser l'*horloges logiques.*
2. L'algorithme permet l'application distribuée d'être d'accord avec les autres sur l'heure exacte des événements. Dans ce cas on dit que l'algorithme permet de synchroniser l'*horloges physiques.*

Pour illustrer ce chapitre nous avons développé une application, c.-à-d., STPC (Synchroniseur du Temp pour PC) qui permet de synchroniser les ordinateurs en appliquant les algorithmes de type 2 (*l'horloge physique*). Avec STPC, l'ecart entre horloges des ordinateurs (12 ordinateurs dans le POOL MECATOR de l'institut d'informatique) est approximativement de 0.05 seconde.

Annexe A. L'implémentation du chapitre 1

I. Le code source *RtcSc.asm* suivant permet de lire les temps de RTC et de SC. Pour le RTC, le temps et la date sont extraits soit directement via les portes *70h* et *71h* soit à l'aide des fonctions de l'interrupteur 1Ah du BIOS (voir tableau 1.3). Pour le SC, la date et le temps sont obtenus en appelant respectivement les fonctions *2Ah* et *2Ch* de l'interrupteur *21h* de DOS.

```
main    segment byte
        assume cs:main,ds:main,ss:nothing
        org     100h
start:
;-----------------------------------------------------------
; PrintHex2 imprime un nombre hex de 2 chiffres.

PrintHex2 MACRO HexDigit
        mov bl,HexDigit
        mov ah,02h
        mov dl,bl       ;imprime
        mov cl,04h      ;premier
        shr dl,cl       ;chiffre
        add dl,30h      ;à
        int 21h         ;l'ecran

        mov dl,bl       ;imprime
        and dl,0fh      ;deuxième
        add dl,30h      ;chiffre
        int 21h         ;à l'écran

        ENDM
;-----------------------------------------------------------
;PrintCmos imprime la valeur du octet de CMOS RAM.

PrintCmos MACRO adress

      mov al,adress
        out 70h, al
        in al,71h
        PrintHex2 al

        ENDM
;-----------------------------------------------------------
;PrintChar imprime un symbole correspondant à un code ascci

PrintChar MACRO ASCIICode

        mov ah,02h
        mov dl,ASCIICode
        int 21h

        ENDM
;-----------------------------------------------------------
```

```
; Imprimer directement la date de RTC en utilisant les portes 70h,71h

        PrintCmos 07h           ;Day of month
        PrintChar 2dh ;Imprimer le symbole '-'
        PrintCmos 08h           ;Month
        PrintChar 2dh ;
        PrintCmos 32h           ;Century
        PrintCmos 09h           ;Year
        PrintChar 20h ;Imprimer un blanc
        PrintChar 20h
; Imprimer directement le temps de RTC en utilisant les portes 70h,71h
        PrintCmos 04h           ;Hour
        PrintChar 3ah ;Imprimer le symbole ':'
        PrintCmos 02h           ;Minute
        PrintChar 3ah
        PrintCmos 00h           ;Second
        PrintChar 0ah
        PrintChar 0dh ;Enter
;-----------------------------------------------------------------
; Imprimer la date de RTC en utilisant la fonction 04h de l'interupteur
; 1Ah de BIOS
        mov ah,04h
        int 1ah
        push cx
        push dx
        mov bh,dl
        PrintHex2 bh  ;Imprimer le jour du mois
        PrintChar 2dh ;Imprimer le symbole '-'
        pop dx
        mov bh,dh
        PrintHex2 bh  ;Imprimer le mois
        PrintChar 2dh
        pop cx
        mov bh,ch
        push cx
        PrintHex2 bh  ;Imprimer la sciècle
        pop cx
        mov bh,cl
        PrintHex2 bh  ;Imprimer l'an
        PrintChar 20h
        PrintChar 20h
; Imprimer le temps de RTC en utilisant la fonction 02h de l'interupteur
; 1Ah de BIOS
        mov ah,02h
        int 1ah
        push dx
        push cx
        mov bh,ch
        PrintHex2 bh  ;Imprimer l'heure
        PrintChar 3ah ;Imprimer le symbole ':'
        pop cx
        mov bh,cl
        PrintHex2 bh  ;Imprimer la minute
        PrintChar 3ah
        pop dx
```

```
        mov bh,dh
        PrintHex2 bh  ;Imprimer la seconde
        PrintChar 0ah ;Feed line
        PrintChar 0dh ;Enter
;-----------------------------------------------------------------
;Imprimer la date de l'horloge du système

        mov ah,2ah
        int 21h
        push dx
        mov dh,0h
        call Printdec   ;Imprimer le jour du mois
        PrintChar 2dh
        pop dx
        mov dl,dh
        mov dh,0h
        call Printdec   ;Imprimer le mois
        Printchar 2dh
        mov dx,cx
        call Printdec   ;Imprimer l'an
        PrintChar 20h

;Imprimer le temps de l'horloge du système

        mov ah,2ch
        int 21h
        push dx
        mov dl,ch
        mov dh,0
        call Printdec   ;imprimer l'heure
        printChar 3ah
        mov dl,cl
        mov dh,0
        call Printdec   ;imprimer la minute
        printChar 3ah
        pop dx
        mov bx,dx
        mov dl,bh
        mov dh,0h
        call Printdec   ;imprimer la seconde
        printChar 2ch
        mov dl,bl
        mov dh,0
        call Printdec   ;imprimer le centième de seconde

        int 20h         ;quitter le programme
;-----------------------------------------------------------------
;Car les fonctions 2Ah et 2Ch de l'interupteur de DOS envoient la
;date et le temps sous la forme Hexadécimale, PrintDec convertit
;la date et le temps en forme Décimale.

PrintDec proc near
        push ax
        push cx
        push dx
```

```
        push si
        mov ax,dx
        mov si,10
        xor cx,cx
nonzero:
        xor dx,dx
        div si
        push dx
        inc cx
        or ax,ax
        jne nonzero
boucle:
        pop dx
        add dl,30h
        mov ah,02h
        int 21h
        dec cx
        or cx,cx
        jne boucle
        pop si
        pop dx
        pop cx
        pop ax
        ret
PrintDec endp

main    ends
        end     start
```

II. Le code source *BDA.asm* suivant permet de modifier le BIOS DATA AREA en utilisant la fonction *00h* de l'interrupteur *1Ah* du BIOS. L'entrée de programme est le nombre de ticks depuis minuit. Après son exécution (au POOL MERCATOR), l'horloge du système prend une valeur qui correspond au nombre de ticks entré.

```
main    segment byte
        assume cs:main,ds:main,ss:nothing
        org     100h
start:
;-----------------------------------------------------------------------------
        mov     ah,01h
        mov     cx,0018h        ; cx et dx contient le nombre des ticks dépuis minuit
        mov     dx,00B0h        ; soit 1800B0h ≈ 1573040 ticks ≈ 24h
        int     1ah
        int     20h
;-----------------------------------------------------------------------------
main    ends
        end     start
```

<u>Note :</u> *RtcSc.asm* et *BDA.asm* sont compilé par le compilateur du Borland :

Tasm RtcSc↵
Tlink/t RtcSc↵

Annexe B. L'implémentation du chapitre 2

Ci-dessous est le code source du STPC (Synchroniseur de Temps pour PC) ; il s'agit d'une application qui permet aux PC de synchroniser son horloge avec les serveurs du temps ou avec les autres PCs. STPC est développée en language C pour l'environnement Window9x en utilisant Windows Sockets Application programming interface (WSA) et TCP/IP suites.

```
#define STRICT
#include <windows.h>
#include <windowsx.h>
#include <winbase.h>
#include <winsock.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <sys\timeb.h>
#include <stdio.h>
#include <dos.h>
#include <mem.h>
#include <math.h>
#include <direct.h>
#include "u:\memoire\wsa_xtra.h"
#include "u:\memoire\winsockx.h"
#include "u:\memoire\timesyn\resource.h"

#define NTP_SCALE  4294967296.0        /* 2^32 */
#define NTP_PACKET_MIN       24
#define NTP_ORIGINATE        0         /* Offset of originate timestamp */
#define NTP_RECEIVE          8         /* Offset of receive timestamp */
#define NTP_TRANSMIT         16        /* Offset of transmit timestamp */

#define IPPORT_VSNTP         2123      /* Port's number for VSNTP*/
#define BUF_SIZE             1024
#define INBUF_SIZE           8
#define TIMEOUT_ID           1
#define TIMEOUT_PERIOD       15000     /* in millisecond*/
#define TIMEOUT_POLLING      2
#define TIMEOUT_IDLE         3
#define TIMEOUT_DIFF         4

/*-------------- global data -------------*/
WSADATA stWSAData;                     /* Winsock DLL info*/

typedef struct stConnData
   {
      SOCKET hSock;
      SOCKADDR_IN stRmtName;
      char achIOBuf[BUF_SIZE];
      struct stConnData FAR * lpstNext;
   } CONNDATA, *PCONNDATA, FAR *LPCONNDATA;

/* The head of the list of daytime's connection*/
LPCONNDATA lpstConnHeadDaytime = 0;
/* The head of the list of Time's connection*/
LPCONNDATA lpstConnHeadTime = 0;
LPCONNDATA lpstConnDaytime,lpstConnTime;

SOCKET hDaytimeTCPSock = INVALID_SOCKET;
SOCKET hDaytimeUDPSock = INVALID_SOCKET;
SOCKET hNewDaytimeSock = INVALID_SOCKET;

SOCKET hTimeTCPSock = INVALID_SOCKET;
SOCKET hTimeUDPSock = INVALID_SOCKET;
SOCKET hNewTimeSock = INVALID_SOCKET;

SOCKET hSNTPUDPSock = INVALID_SOCKET;
SOCKET hSock, hSockLocal;
/* local's and Destination's  address and port number*/
SOCKADDR_IN stLclName,stRmtName;
```

```
char szRmtName[MAXHOSTNAME]={0};
char achInBuf[BUF_SIZE]; //input buffer
char achOutBuf[BUF_SIZE]; //Output buffer

double dInBuf, dOutBuf;
HINSTANCE hInst;
HWND hWinMain;

BOOL bCancel;
BOOL bHead=FALSE;
BOOL bTest=FALSE;
BOOL bClientRole = TRUE; // application act as client
BOOL bServerRole = TRUE; // application act as server
BOOL bOldClientRole;     // before : bOldClientRole = bClientRole
BOOL bOldServerRole;     // after : if CANCEL button is pushed,
                         //         bClientRole = bOldclientRole
                         // The same for bOldServerRole;

BOOL bClientTCP = TRUE;
BOOL bClientUDP = FALSE;
BOOL bClientDaytime = FALSE;
BOOL bClientTime = TRUE;
BOOL bClientSNTP = FALSE;


BOOL bOldClientDaytime;
BOOL bOldClientTime;
BOOL bOldClientSNTP;
BOOL bOldClientTCP,bOldClientUDP;
BOOL bTCPIsDisable,bOldTCPIsDisable; // Checks if TCP mode is disable

BOOL bServerDaytime=TRUE;
BOOL bServerDaytimeTCP=TRUE;
BOOL bServerDaytimeUDP=TRUE;

BOOL bServerTime=TRUE;
BOOL bServerTimeTCP=TRUE;
BOOL bServerTimeUDP=TRUE;

BOOL bServerSNTP=TRUE;
BOOL bServerSNTPUDP=TRUE;

BOOL bOldServerDaytime;
BOOL bOldServerTime;
BOOL bOldServerSNTP;
BOOL bOldServerDaytimeTCP;
BOOL bOldServerDaytimeUDP;
BOOL bOldServerTimeTCP;
BOOL bOldServerTimeUDP;
BOOL bOldServerSNTPUCP;

BOOL bManual = TRUE;
BOOL bAuto = FALSE;
BOOL bOldManual;
BOOL bOldAuto;
BOOL bStopIdle = FALSE;

char szServer[20];
long start,stop,duration; /*for mesurement of round trip*/
struct timeb t;
char szRoundTrip[20];
double originate,receive,transmit,current;
int nPeriod, nOldPeriod; /* Polling's period*/
char pollvalue[10];
time_t stTimeLocal;
long stTimeDiff;

/*------------ function prototypes -----------*/
int WINAPI WinMain (HINSTANCE, HINSTANCE, LPSTR, int);
BOOL CALLBACK Dlg_Main (HWND, UINT, UINT, LPARAM);
```

```
BOOL CALLBACK Dlg_Check (HWND, UINT, UINT, LPARAM);
BOOL CALLBACK Dlg_Options (HWND, UINT, UINT, LPARAM);
BOOL CALLBACK Dlg_Options_Client(HWND, UINT, UINT, LPARAM);
BOOL CALLBACK Dlg_Options_Server(HWND, UINT, UINT, LPARAM);
BOOL CALLBACK Dlg_About (HWND, UINT, UINT, LPARAM);
BOOL InitLstnSock(int,int, PSOCKADDR_IN, HWND, u_int);
BOOL InitClntSock(int,int, PSOCKADDR_IN, HWND, u_int);
void WSAperror(char *,HWND);
u_long GetAddr( LPSTR szHost);
void RemoveConn (LPCONNDATA ,LPCONNDATA);
LPCONNDATA NewConn (SOCKET, PSOCKADDR_IN,LPCONNDATA);
/*-----------------------------------------------------------------
 *  Function: WinMain()
 *
 *  Description:
 *     Initialize WinSock and display main dialog box
 */
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                   LPSTR lpszCmdLine,int nCmdShow)
{
   MSG msg;
   int nRet;

   hPrevInstance = hPrevInstance;
   nCmdShow      = nCmdShow;

   hInst = hInstance;    /* save instance handle */

   /*-------------initialize WinSock DLL------------*/
   nRet = WSAStartup(WSA_VERSION, &stWSAData);
   /* WSAStartup() returns error value if failed (0 on success) */
   if (nRet != 0)
      WSAperror("WSAStartup()", NULL);
      /* No sense continuing if we can't use WinSock */
   else
      {
         DialogBox (hInst,MAKEINTRESOURCE(1), NULL, Dlg_Main);
         /*---------------release WinSock DLL--------------*/
         nRet = WSACleanup();
         if (nRet == SOCKET_ERROR)
            WSAperror("WSACleanup()", NULL);
      }
   return msg.wParam;
} /* end WinMain() */

/*-----------------------------------------------------------------
 * Function: Dlg_Main()
 *
 * Description:
 *  Process dialog messages and asynchronous WinSock messages.
 */
BOOL CALLBACK Dlg_Main(HWND hDlg,UINT msg,UINT wParam,LPARAM lParam)
{
   BOOL bRet = FALSE;
   HWND hwndList, hwndListInfo;
   UINT wTimerID;
   int nAddrSize = sizeof (SOCKADDR);
   int nPort,sign,nOffsetMS;
   int nRet,nLen = SOCKADDR_LEN;
   time_t stTime,stOffsetS;
   struct tm * thoigian;
   struct time TIME;
   char theTime1,theTime2,theTime3,theTime4;
   SOCKET hSockTemp;
   WORD WSAEvent, WSAErr;
   double second,millisecond,OffsetMS,OffsetS;
   int i,k;
   double d,a,b,offset,ecart;
   unsigned char packetOut[NTP_PACKET_MIN],packetIn[NTP_PACKET_MIN];

   hWinMain=hDlg;
```

```
switch (msg)
   {
      case WM_TIMER:
            wTimerID = wParam;

            if (wTimerID == TIMEOUT_DIFF)
               {
                  bStopIdle = TRUE;
                  KillTimer(hWinMain, TIMEOUT_DIFF);
                  KillTimer(hWinMain, TIMEOUT_IDLE);
                  if (bAuto)
                     {
                        nPeriod = nOldPeriod;
                        if (!SetTimer(hWinMain,TIMEOUT_POLLING,nPeriod,NULL))
                           WSAperror("SetTimer() failed", hWinMain);
                        nRet = IDRETRY;
                     }
               }

            if (wTimerID == TIMEOUT_IDLE && !bStopIdle )
               {
                stime(&stTimeLocal);
                if (!SetTimer(hWinMain,TIMEOUT_IDLE,300,NULL))
                     WSAperror("SetTimer() failed", hWinMain);
                nRet = IDRETRY +1;
               }

            if (wTimerID == TIMEOUT_ID)
               {
                  KillTimer(hWinMain,TIMEOUT_ID);
                  nRet = MessageBox(hWinMain,
                                    "No reponse from server ! Try again ?",
                                    "Time Synchronization - Client&Server",
                                    MB_RETRYCANCEL);
               }
            if (wTimerID == TIMEOUT_POLLING)
               nRet = IDRETRY;

            if (nRet == IDRETRY)
               {
                  if (bClientTime)
                     nPort = IPPORT_TIMESERVER;
                  if (bClientDaytime)
                     nPort = IPPORT_DAYTIME;
                  if (bClientSNTP)
                     nPort = IPPORT_VSNTP;

                  hwndList = GetDlgItem(hDlg,IDC_LISTSERVER);
                  nRet=SendMessage(hwndList, LB_GETCURSEL, 0,0);
                  SendMessage(hwndList,LB_GETTEXT,nRet,(LPARAM)szRmtName);
                  stRmtName.sin_addr.s_addr = GetAddr((LPSTR) szRmtName);
                  if (hSock != SOCKET_ERROR)
                     closesocket(hSock);

                  if (bClientTCP)
                     {
                        hSock=InitClntSock(SOCK_STREAM,nPort,&stRmtName
                                           ,hWinMain,WSA_ASYNC);
                        start = GetTickCount();
                        nRet = connect(hSock,(LPSOCKADDR)&stRmtName,
                                                   sizeof (SOCKADDR));
                        if (nRet == SOCKET_ERROR)
                           {
                              int WSAErr = WSAGetLastError();
                              /* Anything but "would block" error is bad */
                              if (WSAErr != WSAEWOULDBLOCK)
                                 {
                                    /* Report error and clean up */
                                    WSAperror("Connect failed", hWinMain);
                                    closesocket(hSock);
```

```
                                hSock = INVALID_SOCKET;
                            }
                        }
                    }
                if (bClientUDP)
                    {
                        hSock=InitClntSock(SOCK_DGRAM,nPort,&stRmtName,
                                           hWinMain,WSA_ASYNC);
                        start = GetTickCount();
                        if (bClientSNTP)
                            {
                                ftime(&t);
                                millisecond=t.millitm;
                                originate = t.time + (millisecond/1000);
                            }
                        nRet= sendto(hSock,(char FAR *) achOutBuf,1,0,
                                     (LPSOCKADDR)&stRmtName,sizeof (SOCKADDR));

                        if (nRet==SOCKET_ERROR)
                            WSAperror("sendto()", hWinMain);
                    }

                    if (!bAuto)
                        {
                            hWinMain = hDlg;
                            if (!SetTimer(hWinMain,TIMEOUT_ID,
                                          TIMEOUT_PERIOD,NULL))
                                WSAperror("SetTimer() failed", hWinMain);
                        }
            }
        break ;
    case WM_INITDIALOG:
        if (bServerRole)
            {
                /* The server support DayTime protocol */
                if (bServerDaytime)
                    {
                        if (bServerDaytimeTCP)
                            hDaytimeTCPSock = InitLstnSock(SOCK_STREAM,
                                              IPPORT_DAYTIME,
                                              &stLclName,hWinMain,WSA_ASYNC);

                        if (bServerDaytimeUDP)
                            hDaytimeUDPSock = InitLstnSock(SOCK_DGRAM,
                                              IPPORT_DAYTIME,
                                              &stLclName,hWinMain,WSA_ASYNC);
                    }
                /* The server support Time protocol */
                if (bServerTime)
                    {
                        if (bServerTimeTCP)
                            hTimeTCPSock = InitLstnSock(SOCK_STREAM,
                                           IPPORT_TIMESERVER,
                                           &stLclName,hWinMain,WSA_ASYNC);

                        if (bServerTimeUDP)
                            hTimeUDPSock = InitLstnSock(SOCK_DGRAM,
                                           IPPORT_TIMESERVER,
                                           &stLclName,hWinMain,WSA_ASYNC);
                    }
                /* The server support very simple NTP */
                if (bServerSNTP)
                        hSNTPUDPSock = InitLstnSock(SOCK_DGRAM,IPPORT_VSNTP,
                                                    &stLclName,hWinMain,WSA_ASYNC);
            }
        break ;
    case WSA_ASYNC:
        hSockTemp = (SOCKET) wParam;
        WSAEvent = WSAGETSELECTEVENT(lParam); /*Extract Event*/
        WSAErr = WSAGETSELECTERROR(lParam);   /* Extract Error*/
```

```
/*Error in asynch notification message*/
if (WSAErr) WSAperror("WSAErr",hWinMain);
switch (WSAEvent)
   {
      case FD_READ:
         ftime(&t);
         stop = GetTickCount();
         millisecond=t.millitm;
         duration=stop-start;

         if (bClientTime)
            nRet = recvfrom (hSockTemp, (char FAR *)&Buf,BUF_SIZE,0,
                  (struct sockaddr *) &stRmtName, &nAddrSize);
         if(bClientDaytime)
            nRet = recvfrom (hSockTemp, (char FAR *)&achInBuf,
                     BUF_SIZE, 0,
                     (struct sockaddr *) &stRmtName, &nAddrSize);
         if (bClientSNTP)
            nRet = recvfrom (hSockTemp,(char FAR *)& packetIn,
                     NTP_PACKET_MIN, 0,
                     (struct sockaddr *) &stRmtName, &nAddrSize);

         /* Display error if receive failed (but don't repeat error
        *  if input message contained an error) */
         if ((nRet == SOCKET_ERROR) && (!WSAErr))
            {
               WSAperror("recieve from error",hWinMain);
               break;
            }
         else
            {
               /*-------------------------------------------
              * CLIENT:
              * If we sent a request, display the response */
               /* Display the data received, and who it came from */
               if (hSockTemp == hSock)
                  {
                     /* if we get the reply then reset the timer*/
                     KillTimer(hWinMain,TIMEOUT_ID);

                     /* if STPC acts as Daytime client*/
                     if (bClientDaytime)
                        {
                           wsprintf (szServer, "Connected to %s",
                                    inet_ntoa(stRmtName.sin_addr));
                           wsprintf (achOutBuf, "Server's time:%s",
                                                      achInBuf);
                           wsprintf (szRoundTrip,
                                    "RoundTrip's time:%ld ms"
                                    ,duration);
                           hwndListInfo = GetDlgItem(hDlg,
                                                      IDC_LISTINFO);
                           SendMessage(hwndListInfo, LB_RESETCONTENT
                                       ,0,0L);
                           SendMessage(hwndListInfo, LB_ADDSTRING,0,
                                                 (LPARAM)szServer);
                           SendMessage(hwndListInfo, LB_ADDSTRING,0,
                                              (LPARAM) achOutBuf);
                           SendMessage(hwndListInfo, LB_ADDSTRING,0,
                                           (LPARAM) szRoundTrip);
                           closesocket(hSock);
                        }

                     /* if STPC acts as Time client*/
                     if (bClientTime)
                        {
                           stTime=ntohl(Buf)-2208988800L;
                           time(&stTimeLocal);
                           stTimeDiff = stTime - stTimeLocal;
                           if (stTimeDiff > 0)
                              {
```

```
                time(&stTimeLocal);
                stTimeLocal += stTimeDiff;
                stime(&stTimeLocal);
            }
        else if (stTimeDiff < 0)
            {
                if (bAuto)
                    KillTimer(hWinMain,
                                    TIMEOUT_POLLING);
                if (!SetTimer(hWinMain,TIMEOUT_DIFF,
                        labs(stTimeDiff*1000),NULL))
                    WSAperror("SetTimer() failed",
                                        hWinMain);
                if (!SetTimer(hWinMain,TIMEOUT_IDLE
                                        ,300,NULL))
                    WSAperror("SetTimer() failed",
                                        hWinMain);
                bStopIdle = FALSE;
                nOldPeriod = nPeriod;
            }
        wsprintf (szServer, "Connected to %s",
                inet_ntoa(stRmtName.sin_addr));
        wsprintf (achOutBuf, "Server's time:%s",
                                ctime(&stTime));
        wsprintf (szRoundTrip,
                "RoundTrip's time:%ld ms",
                                duration);
        hwndListInfo = GetDlgItem(hDlg,
                                IDC_LISTINFO);
        SendMessage(hwndListInfo, LB_RESETCONTENT
                    ,0,0L);
        SendMessage(hwndListInfo, LB_ADDSTRING,0,
                                (LPARAM)szServer);
        SendMessage(hwndListInfo, LB_ADDSTRING,0,
                            (LPARAM) achOutBuf);
        SendMessage(hwndListInfo, LB_ADDSTRING,0,
                        (LPARAM) szRoundTrip);
        wsprintf (achOutBuf,"Offset : %ld seconds"
                                ,stTimeDiff);
        SendMessage(hwndListInfo, LB_ADDSTRING,0,
                            (LPARAM) achOutBuf);
        closesocket(hSock);
    }

if (bClientSNTP)
    {
        current = t.time + (millisecond/1000);
        d = 0.0;
        for (i = 0; i < 8; ++i)
            d = 256.0*d+packetIn[NTP_RECEIVE+i];
        receive = d/NTP_SCALE;
        d = 0.0;
        for (i = 0; i < 8; ++i)
            d = 256.0*d+packetIn[NTP_TRANSMIT+i];
        transmit = d/NTP_SCALE;

        a = receive-originate; b= transmit-current;
        offset = (a+b)/2; ecart=(a-b)/2;
        OffsetMS=modf(offset,&OffsetS);
        stOffsetS = OffsetS;
        nOffsetMS =(OffsetMS*100);
        if (offset > -0.300 && offset != 0.0)
            {
                time(&stTime); stTime += stOffsetS;
                thoigian = localtime(&stTime);
                gettime(&TIME);
                TIME.ti_sec = thoigian->tm_sec;
                TIME.ti_min = thoigian->tm_min;
                TIME.ti_hour = thoigian->tm_hour;
                TIME.ti_hund += nOffsetMS;
                settime(&TIME);
```

```
                }
            else if (offset < -0.300 )
                {
                    if (bAuto)
                        KillTimer(hWinMain,
                                  TIMEOUT_POLLING);
                    if (!SetTimer(hWinMain,TIMEOUT_DIFF,
                                  fabs(offset*1000),NULL))
                        WSAperror("SetTimer() failed",
                                                hWinMain);
                    time(&stTimeLocal);
                    if (!SetTimer(hWinMain,TIMEOUT_IDLE,
                                              300,NULL))
                        WSAperror("SetTimer() failed",
                                              hWinMain);

                    bStopIdle = FALSE;
                    nOldPeriod = nPeriod;
                }

            millisecond=modf(transmit,&second);
            stTime = second; sign=(millisecond*1000);
            wsprintf (szServer, "Connected to %s",
                      inet_ntoa(stRmtName.sin_addr));
            sprintf (achOutBuf,
                      "Server's Time: %s %d ms ",
                                ctime(&stTime),sign);
            hwndListInfo = GetDlgItem(hDlg,
                                          IDC_LISTINFO);
            SendMessage(hwndListInfo, LB_RESETCONTENT
                                              ,0,0L);
            SendMessage(hwndListInfo, LB_ADDSTRING,0,
                                    (LPARAM)szServer);
            SendMessage(hwndListInfo, LB_ADDSTRING,0,
                                  (LPARAM) achOutBuf);
            sprintf (achOutBuf,
                      "offset :%fs  ecart :%fs  ",
                                          offset,ecart);
            SendMessage(hwndListInfo, LB_ADDSTRING,0,
                                  (LPARAM) achOutBuf);
            closesocket(hSock);
        }
    }

if (hSockTemp == hDaytimeUDPSock)
    {
        time(&stTime);
        wsprintf (achOutBuf, "%s",ctime(&stTime));
        nRet = sendto(hDaytimeUDPSock, achOutBuf,
                      strlen(achOutBuf), 0,
                      (LPSOCKADDR)&stRmtName,
                      sizeof (SOCKADDR));
        if (nRet == SOCKET_ERROR)
            WSAperror("FD_READ sendto error",hWinMain);
    }

if (hSockTemp == hTimeUDPSock)
    {
        time(&stTime);
        /* Diff between time() [1-1-1970 UCT]
       and time server [1-1-1900 UCT] */
        stTime += 2208988800;
        theTime4 = (char) stTime & 255;
        stTime = stTime >> 8;
        theTime3 = (char) stTime & 255;
        stTime = stTime >> 8;
        theTime2 = (char) stTime & 255;
        stTime = stTime >> 8;
        theTime1 = (char) stTime & 255;
        stTime = stTime >> 8;
        sprintf(achOutBuf, "%c%c%c%c", theTime1, theTime2,
```

```
                                        theTime3, theTime4);
                nRet = sendto(hTimeUDPSock, achOutBuf,
                      strlen(achOutBuf), 0,
                      (LPSOCKADDR)&stRmtName, sizeof (SOCKADDR));
                if (nRet == SOCKET_ERROR)
                   WSAperror("FD_READ sendto error",hWinMain);
             }

          if (hSockTemp == hSNTPUDPSock)
             {
                receive = t.time + (millisecond/1000);
                d = receive/NTP_SCALE;
                for (i = 0; i < 8; ++i)
                   {
                      if ((k = (int)(d *= 256.0)) >= 256) k = 255;
                      packetOut[NTP_RECEIVE+i] = k;
                      d -= k;
                   }

                ftime(&t);
                millisecond=t.millitm;
                transmit = t.time + (millisecond/1000);
                d = transmit/NTP_SCALE;
                for (i = 0; i < 8; ++i)
                   {
                      if ((k = (int)(d *= 256.0)) >= 256) k = 255;
                      packetOut[NTP_TRANSMIT+i] = k;
                      d -= k;
                   }
                nRet = sendto(hSNTPUDPSock,(char FAR *)& packetOut,
                      NTP_PACKET_MIN, 0,(LPSOCKADDR)&stRmtName,
                                        sizeof (SOCKADDR));
                if (nRet == SOCKET_ERROR)
                   WSAperror("FD_READ sendto error",hWinMain);
             }
       }
    break;
 case FD_ACCEPT:
    if (hSockTemp == hDaytimeTCPSock)
       {
          hNewTimeSock = accept(hDaytimeTCPSock,
                                (LPSOCKADDR)&stRmtName,
                                (LPINT)&nLen );
          /* put the new socket structe in the list */
          lpstConnDaytime=NewConn(hNewTimeSock, &stRmtName,
                                        lpstConnHeadDaytime);
          time(&stTime);
          lstrcpy(lpstConnDaytime->achIOBuf,ctime(&stTime));

          nRet = sendto (lpstConnDaytime->hSock,
                         lpstConnDaytime->achIOBuf,
                          strlen(lpstConnDaytime->achIOBuf), 0,
                         (LPSOCKADDR)&(lpstConnDaytime->stRmtName),
                          sizeof (SOCKADDR));

          if (nRet == SOCKET_ERROR)
             WSAperror("sendto error",hWinMain);
          closesocket(lpstConnDaytime->hSock);
          RemoveConn(lpstConnDaytime,lpstConnHeadDaytime);

       }

    if (hSockTemp == hTimeTCPSock)
       {
          hNewTimeSock = accept(hTimeTCPSock,
                         (LPSOCKADDR)&stRmtName,(LPINT)&nLen );
          lpstConnTime=NewConn(hNewTimeSock, &stRmtName,
                                        lpstConnHeadTime);
          time(&stTime);
          /* Diff between time() [1-1-1970 UCT]
         and time server [1-1-1900 UCT] */
```

```
                    stTime += 2208988800;
                    theTime4 = (char) stTime & 255;
                    stTime = stTime >> 8;
                    theTime3 = (char) stTime & 255;
                    stTime = stTime >> 8;
                    theTime2 = (char) stTime & 255;
                    stTime = stTime >> 8;
                    theTime1 = (char) stTime & 255;
                    stTime = stTime >> 8;

                    sprintf(lpstConnTime->achIOBuf, "%c%c%c%c", theTime1,
                              theTime2, theTime3, theTime4);
                    nRet = sendto (lpstConnTime->hSock,lpstConnTime->achIOBuf,
                                    strlen(lpstConnTime->achIOBuf), 0,
                                   (LPSOCKADDR)&(lpstConnTime->stRmtName),
                                    sizeof (SOCKADDR));
                    if (nRet == SOCKET_ERROR)
                       WSAperror("sendto error",hWinMain);
                    closesocket(lpstConnTime->hSock);
                    RemoveConn(lpstConnTime,lpstConnHeadTime);
                 }
              break;
           case FD_CLOSE:
              break;
           default :
              break;
        } /* End switch(WSAEvent) */
     break;

  case WM_COMMAND:
     switch (wParam)
        {
           case IDC_ADDSERVER:
              DialogBox(hInst,MAKEINTRESOURCE(6),hDlg,Dlg_Check);
              hwndList = GetDlgItem(hDlg,IDC_LISTSERVER);
              if (!bCancel && lstrlen(szRmtName))
                 {
                    SendMessage(hwndList, LB_ADDSTRING, 0,(LPARAM)szRmtName);
                 }
              SetFocus(hwndList);
              break;
           case IDC_EDITSERVER:
              hwndList = GetDlgItem(hDlg,IDC_LISTSERVER);
              nRet=SendMessage(hwndList, LB_GETCURSEL, 0,0);
              SendMessage(hwndList,LB_GETTEXT,nRet,(LPARAM)szRmtName);
              DialogBox(hInst,MAKEINTRESOURCE(6),hDlg,Dlg_Check);
              if (!bCancel && lstrlen(szRmtName))
                 {
                    hwndList = GetDlgItem(hDlg,IDC_LISTSERVER);
                    nRet=SendMessage(hwndList, LB_GETCURSEL, 0,0);
                    SendMessage(hwndList, LB_DELETESTRING, nRet,0);
                    SendMessage(hwndList, LB_ADDSTRING, 0,(LPARAM)szRmtName);
                 }
              SetFocus(hwndList);
              break;
           case IDC_DELSERVER:
              hwndList = GetDlgItem(hDlg,IDC_LISTSERVER);
              nRet=SendMessage(hwndList, LB_GETCURSEL, 0,0);
              SendMessage(hwndList, LB_DELETESTRING, nRet,0);
              SetFocus(hwndList);
              break;
           case IDC_CHECK:
              if (bClientRole)
                 {
                    if (bClientTime)
                       nPort = IPPORT_TIMESERVER;
                    if (bClientDaytime)
                       nPort = IPPORT_DAYTIME;
                    if (bClientSNTP)
                       nPort = IPPORT_VSNTP;
```

```
                hwndList = GetDlgItem(hDlg,IDC_LISTSERVER);
                nRet=SendMessage(hwndList, LB_GETCURSEL, 0,0);
                SendMessage(hwndList,LB_GETTEXT,nRet,(LPARAM)szRmtName);
                stRmtName.sin_addr.s_addr = GetAddr((LPSTR) szRmtName);
                if (bClientTCP)
                   {
                      hSock=InitClntSock(SOCK_STREAM,nPort,&stRmtName
                                         ,hWinMain,WSA_ASYNC);
                      start = GetTickCount();
                      nRet = connect(hSock,(LPSOCKADDR)&stRmtName,
                                                   sizeof (SOCKADDR));
                      if (nRet == SOCKET_ERROR)
                         {
                            int WSAErr = WSAGetLastError();
                            /* Anything but "would block" error is bad */
                            if (WSAErr != WSAEWOULDBLOCK)
                               {
                                  /* Report error and clean up */
                                  WSAperror("Connect failed", hWinMain);
                                  closesocket(hSock);
                                  hSock = INVALID_SOCKET;
                               }
                         }
                   }
                if (bClientUDP)
                   {
                      hSock=InitClntSock(SOCK_DGRAM,nPort,&stRmtName,
                                         hWinMain,WSA_ASYNC);
                      start = GetTickCount();
                      if (bClientSNTP)
                         {
                            ftime(&t);
                            millisecond=t.millitm;
                            originate = t.time + (millisecond/1000);
                         }
                      nRet= sendto(hSock,(char FAR *) achOutBuf,1,0,
                         (LPSOCKADDR)&stRmtName,sizeof (SOCKADDR));

                      if (nRet==SOCKET_ERROR)
                         WSAperror("sendto()", hWinMain);
                   }
                if (bAuto)
                   {
                      if (!SetTimer(hWinMain,TIMEOUT_POLLING,
                                                   nPeriod,NULL))
                         WSAperror("SetTimer() failed", hWinMain);
                   }
                else
                   if (!SetTimer(hWinMain,TIMEOUT_ID,TIMEOUT_PERIOD,
                                                         NULL))
                      WSAperror("SetTimer() failed", hWinMain);
             }
          break;
       case IDC_ABOUT:
          DialogBox(hInst,MAKEINTRESOURCE(2),hDlg,Dlg_About);
          break;
       case IDC_OPTIONS:
          bOldClientRole = bClientRole;
          bOldServerRole = bServerRole;
          bOldManual = bManual;
          bOldAuto = bAuto;
          DialogBox(hInst,MAKEINTRESOURCE(3),hDlg,Dlg_Options);
          break;
       case IDC_EXIT:
          EndDialog(hDlg,msg);
          bRet=TRUE;
          break;
       default :
          break;
    } /* end switch(wParam) */
 break;
```

```
        default :
            break ;
    } /* end switch (msg) */

    return bRet;
} /* end Dlg_Main() */

/*-----------------------------------------------------------------
 * Function: Dlg_Check()
 *
 * Description:
 *  Display dialog CheckTime
 */
BOOL CALLBACK Dlg_Check(HWND hDlg,UINT msg,UINT wParam,LPARAM lParam)
{
    BOOL bRet = FALSE;
    int i,nRet;
    switch (msg)
      {
        case WM_INITDIALOG:
            SetDlgItemText(hDlg,IDC_DEST,szRmtName);
            break ;
        case WM_COMMAND:
            switch (wParam)
              {
                case IDOK:
                    nRet=GetDlgItemText(hDlg,IDC_DEST,szRmtName,MAXHOSTNAME);
                    for (i=0;i<nRet;i++)
                        if (szRmtName[i] == ' ')
                          {
                              WSAperror("Spaces is not allowed", hDlg);
                              bCancel=TRUE;
                          }
                        else
                            bCancel=FALSE;
                    EndDialog(hDlg,msg);
                    bRet=TRUE;
                    break ;
                case IDCANCEL:
                    EndDialog(hDlg,msg);
                    bRet=FALSE;
                    bCancel=TRUE;
                    break ;
                default :
                    break ;
              } /* end switch(wParam) */
            break ;

        default :
            break ;
      } /* end switch (msg) */
    return bRet;
}
/*-----------------------------------------------------------------
 * Function: Dlg_About()
 *
 * Description:
 *  Display dialog About
 */
BOOL CALLBACK Dlg_About(HWND hDlg,UINT msg,UINT wParam,LPARAM lParam)
{
    BOOL bRet = FALSE;

    switch (msg)
      {
        case WM_COMMAND:
            switch (wParam)
              {
                case IDC_OK_ABOUT:
                    EndDialog(hDlg,msg);
                    bRet=TRUE;
```

```
                    break ;

                 default :
                    break ;
              } /* end switch(wParam) */
           break ;

        default :
           break ;
     } /* end switch (msg) */
   return bRet;
}
/*-----------------------------------------------------------------------
 * Function: Dlg_Options()
 *
 * Description:
 *  Display dialog Options
 */
BOOL CALLBACK Dlg_Options(HWND hDlg,UINT msg,UINT wParam,LPARAM lParam)
{
   BOOL bRet = FALSE;

   switch (msg)
      {
         case WM_INITDIALOG:
            CheckDlgButton(hDlg,IDC_CHECKBOX_C,bClientRole);
            CheckDlgButton(hDlg,IDC_CHECKBOX_S,bServerRole);
            CheckDlgButton(hDlg,IDC_MANUAL,bManual);
            CheckDlgButton(hDlg,IDC_AUTO,bAuto);
            EnableWindow(GetDlgItem(hDlg,IDC_OPTIONS_C),bClientRole);
            EnableWindow(GetDlgItem(hDlg,IDC_OPTIONS_S),bServerRole);
            EnableWindow(GetDlgItem(hDlg,IDC_POLLVALUE),bAuto);
            if (!bAuto)
               SetDlgItemText(hDlg,IDC_POLLVALUE,"not available");
            else
               {
                  itoa(nPeriod,pollvalue,10);
                  SetDlgItemText(hDlg,IDC_POLLVALUE,pollvalue);
               }
            break ;
         case WM_COMMAND:
            switch (wParam)
               {
                  case IDC_CHECKBOX_C:
                     bClientRole=!bClientRole;
                     EnableWindow(GetDlgItem(hDlg,IDC_OPTIONS_C),bClientRole);
                     break ;
                  case IDC_CHECKBOX_S:
                     bServerRole=!bServerRole;
                     EnableWindow(GetDlgItem(hDlg,IDC_OPTIONS_S),bServerRole);
                     break ;
                  case IDC_MANUAL:
                     bManual=TRUE;
                     bAuto = FALSE;
                     EnableWindow(GetDlgItem(hDlg,IDC_POLLVALUE),bAuto);
                     break ;
                  case IDC_AUTO:
                     bAuto= TRUE; bManual=FALSE;
                     EnableWindow(GetDlgItem(hDlg,IDC_POLLVALUE),bAuto);
                     break ;
                  case IDC_OPTIONS_C:
                     bOldClientDaytime=bClientDaytime;
                     bOldClientTime=bClientTime;
                     bOldClientSNTP=bClientSNTP;
                     bOldClientTCP=bClientTCP;
                     bOldClientUDP=bClientUDP;
                     bOldTCPIsDisable=bTCPIsDisable;
                     DialogBox(hInst,MAKEINTRESOURCE(4),hDlg,Dlg_Options_Client);
                     break ;
                  case IDC_OPTIONS_S:
                     bOldServerDaytime=bServerDaytime;
```

```
                    bOldServerTime=bServerTime;
                    bOldServerSNTP=bServerSNTP;
                    bOldServerDaytimeTCP=bServerDaytimeTCP;
                    bOldServerDaytimeUDP=bServerDaytimeUDP;
                    bOldServerTimeTCP=bServerTimeTCP;
                    bOldServerTimeUDP=bServerTimeUDP;
                    bOldServerSNTPUCP=bServerSNTPUDP;
                    DialogBox(hInst,MAKEINTRESOURCE(5),hDlg,Dlg_Options_Server);
                    break;
                 case IDOK:
                    if (bManual)
                       KillTimer(hWinMain,TIMEOUT_POLLING);
                    if (bAuto)
                       {
                          GetDlgItemText(hDlg,IDC_POLLVALUE,pollvalue,6);
                          nPeriod = atoi(pollvalue);
                       }
                    EndDialog(hDlg,msg);
                    bRet=TRUE;
                    break;
                 case IDCANCEL:
                    bClientRole=bOldClientRole;
                    bServerRole=bOldServerRole;
                    bManual = bOldManual;
                    bAuto = bOldAuto;
                    EndDialog(hDlg,msg);
                    bRet=TRUE;
                    break;

                 default :
                    break;
              } /* end switch(wParam) */
           break;

        default :
           break;
     } /* end switch (msg) */
   return bRet;
}

/*-----------------------------------------------------------------------
 * Function: Dlg_Options_Client()
 *
 * Description:
 *  Display dialog Options_Client
 */
BOOL CALLBACK Dlg_Options_Client(HWND hDlg,UINT msg,UINT wParam,LPARAM lParam)
{
   BOOL bRet = FALSE;

   switch (msg)
      {
         case WM_INITDIALOG:
            CheckDlgButton(hDlg,IDC_RADIO_C_DAYTIME,bClientDaytime);
            CheckDlgButton(hDlg,IDC_RADIO_C_TIME,bClientTime);
            CheckDlgButton(hDlg,IDC_RADIO_C_SNTP,bClientSNTP);
            CheckDlgButton(hDlg,IDC_RADIO_C_TCP,bClientTCP);
            CheckDlgButton(hDlg,IDC_RADIO_C_UDP,bClientUDP);
            EnableWindow(GetDlgItem(hDlg,IDC_RADIO_C_TCP),(!bTCPIsDisable));
            break;
         case WM_COMMAND:
            switch (wParam)
               {
                  case IDC_RADIO_C_DAYTIME:
                     bClientDaytime=TRUE;
                     bClientTime=FALSE;
                     bClientSNTP=FALSE;
                     if (bTCPIsDisable)
                        {
                           EnableWindow(GetDlgItem(hDlg,IDC_RADIO_C_TCP),
                                                                bTCPIsDisable);
```

```
                        bTCPIsDisable = !bTCPIsDisable;
                    }
                break;
            case IDC_RADIO_C_TIME:
                bClientDaytime=FALSE;
                bClientTime=TRUE;
                bClientSNTP=FALSE;
                if (bTCPIsDisable)
                    {
                        EnableWindow(GetDlgItem(hDlg,IDC_RADIO_C_TCP),
                                                                bTCPIsDisable);
                        bTCPIsDisable = !bTCPIsDisable;
                    }
                break;
            case IDC_RADIO_C_SNTP:
                bClientDaytime=FALSE;
                bClientTime=FALSE;
                bClientSNTP=TRUE;
                bClientTCP=FALSE;
                bClientUDP=TRUE;
                bTCPIsDisable=TRUE;
                CheckDlgButton(hDlg,IDC_RADIO_C_UDP,bClientUDP);
                CheckDlgButton(hDlg,IDC_RADIO_C_TCP,bClientTCP);
                EnableWindow(GetDlgItem(hDlg,IDC_RADIO_C_TCP),bClientTCP);
                break;
            case IDC_RADIO_C_TCP:
                bClientTCP=TRUE;
                bClientUDP=FALSE;
                break;
            case IDC_RADIO_C_UDP:
                bClientTCP=FALSE;
                bClientUDP=TRUE;
                break;
            case IDOK:
                EndDialog(hDlg,msg);
                bRet=TRUE;
                break;
            case IDCANCEL:
                bClientDaytime=bOldClientDaytime;
                bClientTime=bOldClientTime;
                bClientSNTP=bOldClientSNTP;
                bClientTCP=bOldClientTCP;
                bClientUDP=bOldClientUDP;
                bTCPIsDisable=bOldTCPIsDisable;
                EndDialog(hDlg,msg);
                bRet=TRUE;
                break;
            default :
                break;
        } /* end switch(wParam) */
        break;
    default :
        break;
    } /* end switch (msg) */
  return bRet;
}
/*-----------------------------------------------------------------------
 * Function: Dlg_Options_Server()
 *
 * Description:
 *  Display dialog Options_Server
 */
BOOL CALLBACK Dlg_Options_Server(HWND hDlg,UINT msg,UINT wParam,LPARAM lParam)
{
  BOOL bRet = FALSE;

  switch (msg)
    {
      case WM_INITDIALOG:
        CheckDlgButton(hDlg,IDC_CHECK_S_DAYTIME,bServerDaytime);
        EnableWindow(GetDlgItem(hDlg,IDC_RADIO_S_DT_TCP),bServerDaytime);
```

```
    EnableWindow(GetDlgItem(hDlg,IDC_RADIO_S_DT_UDP),bServerDaytime);
    CheckDlgButton(hDlg,IDC_RADIO_S_DT_TCP,bServerDaytimeTCP);
    CheckDlgButton(hDlg,IDC_RADIO_S_DT_UDP,bServerDaytimeUDP);

    CheckDlgButton(hDlg,IDC_CHECK_S_TIME,bServerTime);
    EnableWindow(GetDlgItem(hDlg,IDC_RADIO_S_T_TCP),bServerTime);
    EnableWindow(GetDlgItem(hDlg,IDC_RADIO_S_T_UDP),bServerTime);
    CheckDlgButton(hDlg,IDC_RADIO_S_T_TCP,bServerTimeTCP);
    CheckDlgButton(hDlg,IDC_RADIO_S_T_UDP,bServerTimeUDP);

    CheckDlgButton(hDlg,IDC_CHECK_S_SNTP,bServerSNTP);
    EnableWindow(GetDlgItem(hDlg,IDC_RADIO_S_S_UDP),bServerSNTP);
    CheckDlgButton(hDlg,IDC_RADIO_S_S_UDP,bServerSNTPUDP);
    break ;
 case WM_COMMAND:
    switch (wParam)
       {
          case IDC_CHECK_S_DAYTIME:
             bServerDaytime=!bServerDaytime;
             EnableWindow(GetDlgItem(hDlg,IDC_RADIO_S_DT_TCP),
                                                    bServerDaytime);
             EnableWindow(GetDlgItem(hDlg,IDC_RADIO_S_DT_UDP),
                                                    bServerDaytime);
             break ;
          case IDC_RADIO_S_DT_TCP:
             bServerDaytimeTCP=TRUE;
             bServerDaytimeUDP=FALSE;
             break ;
          case IDC_RADIO_S_DT_UDP:
             bServerDaytimeUDP=TRUE;
             bServerDaytimeTCP=FALSE;
             break ;
          case IDC_CHECK_S_TIME:
             bServerTime=!bServerTime;
             EnableWindow(GetDlgItem(hDlg,IDC_RADIO_S_T_TCP),bServerTime);
             EnableWindow(GetDlgItem(hDlg,IDC_RADIO_S_T_UDP),bServerTime);
             break ;
          case IDC_RADIO_S_T_TCP:
             bServerTimeTCP=TRUE;
             bServerTimeUDP=FALSE;
             break ;
          case IDC_RADIO_S_T_UDP:
             bServerTimeUDP=TRUE;
             bServerTimeTCP=FALSE;
             break ;
          case IDC_CHECK_S_SNTP:
             bServerSNTP=!bServerSNTP;
             EnableWindow(GetDlgItem(hDlg,IDC_RADIO_S_S_UDP),bServerSNTP);
             break ;
          case IDOK:
             EndDialog(hDlg,msg);
             bRet=TRUE;
             break ;
          case IDCANCEL:
             bServerDaytime=bOldServerDaytime;
             bServerTime=bOldServerTime;
             bServerSNTP=bOldServerSNTP;
             bServerDaytimeTCP=bOldServerDaytimeTCP;
             bServerDaytimeUDP=bOldServerDaytimeUDP;
             bServerTimeTCP=bOldServerTimeTCP;
             bServerTimeUDP=bOldServerTimeUDP;
             bServerSNTPUDP=bOldServerSNTPUCP;
             EndDialog(hDlg,msg);
             bRet=TRUE;
             break ;
          default :
             break ;
       } /* end switch(wParam) */
    break ;

 default :
```

```
            break ;
      } /* end switch (msg) */
   return bRet;
}
/*-----------------------------------------------------------------
 * Function: InitLstnSock()
 *
 * Description: Get a TCP/UDP socket, and start listening for
 *  incoming connection requests.
 */
BOOL InitLstnSock(int iType, int iLstnPort, PSOCKADDR_IN pstSockName,HWND hWnd,
                  u_int nAsyncMsg)
{
   int nRet;
   SOCKET hLstnSock;

   /* Get a TCP socket to use for data connection listen */
   hLstnSock = socket (AF_INET, iType, 0);
   if (hLstnSock == INVALID_SOCKET)
      WSAperror("socket()", hWnd);
   else
      {
         /* Request async notification for most events */
         nRet = WSAAsyncSelect(hLstnSock, hWnd, nAsyncMsg,
                               (FD_ACCEPT | FD_READ | FD_WRITE | FD_CLOSE));
         if (nRet == SOCKET_ERROR)
            WSAperror("WSAAsyncSelect()", hWnd);
         else
            {
               /* Name the local socket with bind() */
               pstSockName->sin_family = PF_INET;
               pstSockName->sin_port   = (u_short) htons((u_short)iLstnPort);
               pstSockName->sin_addr.s_addr = INADDR_ANY;
               nRet = bind(hLstnSock,(LPSOCKADDR)pstSockName,SOCKADDR_LEN);
               if (nRet == SOCKET_ERROR)
                  WSAperror("bind()", hWnd);
               else
                  /* if TCP socket then listen for incoming connection requests */
                  if (iType==SOCK_STREAM)
                     {
                        nRet = listen(hLstnSock, 5);
                        if (nRet == SOCKET_ERROR)
                           WSAperror("listen()", hWnd);
                     }
            }
         /* If we had an error then we have a problem.  Clean up */
         if (nRet == SOCKET_ERROR)
            {
               closesocket(hLstnSock);
               hLstnSock = INVALID_SOCKET;
            }
      }
   return (hLstnSock);
} /* end InitLstnSock() */
/*-----------------------------------------------------------------
 * Function: InitClientSock()
 *
 * Description: Get a client's TCP/UDP  socket
 *
 */
BOOL InitClntSock(int iType, int iLstnPort, PSOCKADDR_IN pstSockName,HWND hWnd,
                  u_int nAsyncMsg)
{
   int nRet;
   SOCKET hClntSock;

   hClntSock = socket (AF_INET,iType, 0);
   if (hClntSock == INVALID_SOCKET)
      WSAperror("socket()", hWnd);
   else
      {
```

```
        /* Request async notification for most events */
        nRet = WSAAsyncSelect(hClntSock, hWnd, nAsyncMsg,
                              (FD_ACCEPT | FD_READ | FD_WRITE | FD_CLOSE));
        if (nRet == SOCKET_ERROR)
           {
              closesocket(hClntSock);
              hClntSock = INVALID_SOCKET;
              WSAperror("WSAAsyncSelect()", hWnd);
           }
        else
           {
              pstSockName->sin_family = PF_INET;
              pstSockName->sin_port = (u_short) htons((u_short)iLstnPort);
           }
     }
   return (hClntSock);
}
/*-----------------------------------------------------------------
 * Function: WSAperror()
 *
 * Description: Display an error message
 *
 */
void WSAperror(char szMessage[],HWND hWnd)
   {
      MessageBox (hWnd,szMessage ,"Time Synchronisation - Client&Server",
                  MB_OK | MB_ICONASTERISK);
   }
/*-----------------------------------------------------------------
 *
 */
u_long GetAddr( LPSTR szHost)
   {
      LPHOSTENT lpstHost;
      u_long lAddr = INADDR_ANY;

      /*Check if we have a string */
      if (*szHost)
         {
            // check for a dotted ip address string
            lAddr = inet_addr(szHost);
            if ((lAddr == INADDR_NONE) && (_fstrcmp(szHost,"255.255.255.255")))
               {
                  lpstHost = gethostbyname(szHost);
                  if (lpstHost) //success
                     lAddr = *((u_long FAR *) (lpstHost->h_addr));
                  else
                     lAddr = INADDR_ANY; //failure
               }
         }
      return (lAddr);
   }
/*-----------------------------------------------------------------
 * Function:NewConn()
 *
 * Description: Create a new socket structure and put in list
 */
LPCONNDATA NewConn (SOCKET hSock, PSOCKADDR_IN pstRmtName,LPCONNDATA lpstSockHead)
   {
      int nAddrSize = sizeof (SOCKADDR);
      LPCONNDATA lpstSockTmp, lpstSock = (LPCONNDATA)0;
      HLOCAL hConnData;

      /* Allocate memory for the new socket structure */
      /* LMEM_ZEROINIT : Initializes memory contents to zero.*/
      hConnData = LocalAlloc(LMEM_ZEROINIT, sizeof (CONNDATA));

      if (hConnData)
         {
            /* Lock it down and link it into the list */
            lpstSock = LocalLock(hConnData);
```

```
            /* if the List of connection is empty*/
            if (!lpstSockHead)
               {
                  lpstSockHead = lpstSock;
                  bHead=TRUE;
               }
            else
               {  /*go to the end of the list*/
                  for (lpstSockTmp = lpstSockHead;
                        lpstSockTmp && lpstSockTmp->lpstNext;
                        lpstSockTmp = lpstSockTmp->lpstNext);
                  /* add the new ellement in the list*/
                     lpstSockTmp->lpstNext = lpstSock;
               }

            /* Initialize socket structure */
            lpstSock->hSock = hSock;
            _fmemcpy ((LPSTR)&(lpstSock->stRmtName),
                     (LPSTR)pstRmtName, sizeof (SOCKADDR));
            bTest=FALSE;
         }
      else
         bTest=TRUE;

      return (lpstSockHead);
   } /* end NewConn() */

/*-----------------------------------------------------------------
 * Function: RemoveConn()
 *
 * Description: Free the memory for socket structure
 */
void RemoveConn (LPCONNDATA lpstSock,LPCONNDATA lpstSockHead)
   {
      LPCONNDATA lpstSockTmp;
      HLOCAL hSock;

      if (lpstSock == lpstSockHead)
         lpstSockHead = lpstSock->lpstNext;
      else
         for (lpstSockTmp = lpstSockHead;
               lpstSockTmp;
               lpstSockTmp = lpstSockTmp->lpstNext)
            if (lpstSockTmp->lpstNext == lpstSock)
               lpstSockTmp->lpstNext = lpstSock->lpstNext;

      hSock = LocalHandle(lpstSock);
      LocalUnlock (hSock);
      LocalFree (hSock);

   } /* end RemoveConn() */
```

Liste des acronymes

API	Application Programming Interface
BCD	Binary Coded Decimal
BSD	Berkeley Software Distribution
BIOS	Basic Input Output System
CMOS	Complementary Metal Oxide Semiconductor
DOS	Disk Operating System
MS-DOS	Microsoft Disk Operating System
NTP	Network Time Protocol
PIC	Programmable Interrupt Controller
PIT	Programmable Interval Timer
RAM	Random Access Memory
RFC	Request For Comments
ROM	Read Only Memory
RTC	Real Time Clock
UDP	User Datagram Protocol
UTC	Universal Coordinated Time
TAI	International Atomic Time
TCP/IP	Transmission Control Protocol / Internet Protocol
WSA	Windows Sockets API

Bibliography

1) Charles Calvert ,*Teach Yourself Windows 95 programming in 21 days, 2nd edition,* SAM'S, 1995. ISBN : 0-672-30762-6.

2) Randy Chow, Theodore Johnson, *Distributed Operating Systems & Algorithms,* ADDISON-WESLEY,1997. ISBN : 0-201-49838-3.

3) George Coulouris, Jean Dollimore, Tim Kindberg, *Distributed Systems Concepts and Design, 2nd,* ADDISON-WESLEY,1994. ISBN : 0-201-62433-8.

4) Frank van Gilluwe, *The Undocumented PC, 2nd*, ADDISON-WESLEY, 1997. ISBN : 0-201-47950-8.

5) Samuel P.Harbison, Guy L.Steele Jr, *C A Reference Manual,3nd,* Prentice Hall.

6) Randall Hyde, The Art Of Assembly Language Programming , Ebook, 1996.

7) David L. Mills, *NTP – Network Time Protocol,* DARPA Network Working Group Report RFC-1305, 1992.

8) David L. Mills, *SNTP – Simple Network Time Protocol,* DARPA Network Working Group Report RFC-2030, 1996.

9) David L. Mills, *Precision Synchronization Computer Network Clocks,* ACM computer Communication Review.

10) David L. Mills, *On the Accuracy and Stability of Clocks Synchronized by the Network Time protocol in the Internet System,* ACM Computer Communication Review 21, 5 (October 1991), 8-17.

11) David L. Mills, *On the Chronometry and Metrology of Computer Network Timescales and their Application to the Network Time protocol,* ACM Computer Communication Review 20, 1(January 1990),65-75.

12) David L. Mills, *Modelling and Analysis of Computer Network Clocks* , University of Delaware, Electrical Engineering Department, Technical Report 92-5-2, 1992.

13) David L. Mills, *Internet Time Synchronization : the Network Time protocol,* IEEE Trans. Communication 39, 10(October 1991), 1482-1493.

14) J. Postel, *Daytime Protocol,* DARPA Network Working Group Report RFC 867, 1983.

15) J. Postel, K. Harrenstien, *Time Protocol,* DARPA Network Working Group Report RFC 868, 1983.

16) Jean Ramaekers, *Cours de Système d'Exploitation,*Institut d'Informatique, Facultés Universitaires Notre-Dame de la Paix ,1995.

17) Jean Ramaekers, *Cours de Système d'Exploitation – matières approfondies* Institut d'Informatique, Facultés Universitaires Notre-Dame de la Paix, 1995.

18) Jean Ramaekers, *Cours de Sécurité,* Institut d'Informatique, Facultés Universitaires Notre-Dame de la Paix, 1998.

19) W. Rechard Stevens, *Unix Network Programming,* Prentice – Hall , 1990. ISBN : 0-13-949876-1.

20) W. Rechard Stevens, *Advanced Programming in Unix Environnement,* ADDISON-WESLEY 1992. ISBN : 0-201-56317-7.

21) Bob Quinn , Dave Shute, *Windows Sockets Network Programming,* ADDISON-WESLEY 1996. ISBN : 0-201-63372-8.

22) Andrew S. Tanenbaum, *Modern Operating System,* Prentice Hall, 1992.

23) Andrew S. Tanenbaum, *Computer Networks, 3nd edition,* Prentice Hall.

24) Michael Tischer, Bruno Jennrich, *PC Programmation System, 6 nd,* Micro Application, 1997. ISBN : 2-7429-0544-8.

25) Nguyen Anh Tuan, *Y2K Problem,* PCWORLD Vietnam, (72) October 1998.

26) Pete Woytovetch, *The Century Rollover and the PC System Date,* Dell BIOS Development, 1996.

27) *Year 2000 Compliance and the Industry Standard Personal Computer,* NSTL/National Sofware Testing Laboratories, White Paper.

28) Man Pages de adjtime, ctime, gettimeofday, ntp_adjtime, ntp_gettime, settimeofday, TIMEZONE. SunOS 5.6. Last change : 28 Jan 1997.