Appendix B. Precision Time Kernel Programming Information

The following sections describe programming interfaces and calling sequences for modified SunOS, Ultrix and OSF/1 kernel software that manages the local clock and timer functions. The software provides improved accuracy and stability through the use of a disciplined local clock and interface for use with the Network Time Protocol (NTP) or similar time-synchronization protocol. The NTP Version 3 distribution supports these features for all three kernels.

There are separate versions of the software that operate with a Sun Microsystems SPARCstation with the SunOS 4.1.x kernel, Digital Equipment DECstation 5000 with the Ultrix 4.x kernel and Digital Equipment 3000 AXP Alpha with the OSF/1 V1.x kernel. The following sections describe the kernel interfaces to the protocol daemon and user application programs. The ideas are based on suggestions from Jeffery Mogul and Philip Gladstone and a similar interface designed by the latter. It is important to point out that the functionality of the original Unix adjtime() system call is preserved, so that the modified kernel will work as the unmodified one should the kernel PLL not be in use. In this case the ntp_adjtime() system call can still be used to read and write kernel variables that might be used by a protocol daemon other than NTP, for example.

B.1. The ntp_gettime() System Call

The syntax and semantics of the ntp_gettime() call are given in the following fragment of the timex.h header file. This file is identical in the SunOS, Ultrix and OSF/1 kernel distributions. Note that the timex.h file calls the syscall.h system header file, which must be modified to define the SYS_ntp_gettime system call specific to each system type. The kernel distributions include directions on how to do this.

```
/*
 * This header file defines the Network Time Protocol (NTP) interfaces
 * for user and daemon application programs. These are implemented using
 * private system calls and data structures and require specific kernel
 * support.
 *
 * NAME
 *      ntp_gettime - NTP user application interface
 *
 * SYNOPSIS
 *      #include <sys/timex.h>
 *
 *      int system call(SYS_ntp_gettime, tptr)
 *
 *      int SYS_ntp_gettime             defined in syscall.h header file
 *      struct ntptimeval *tptr         pointer to ntptimeval structure
 *
 * NTP user interface - used to read kernel clock values
 * Note: maximum error = NTP synch distance = dispersion + delay / 2;
 * estimated error = NTP dispersion.
 */
struct ntptimeval {
        struct timeval time;            /* current time */
```

```
        long maxerror;          /* maximum error (usec) */
        long esterror;          /* estimated error (usec) */
};
```

The ntp_gettime() system call returns three values in the ntptimeval structure: the current time in unix timeval format plus the maximum and estimated errors in microseconds (usec). While the 32-bit long data type limits the error quantities to something more than an hour, in practice this is not significant, since the protocol itself will declare an unsynchronized condition well below that limit. If the protocol computes either of these values in excess of 16 seconds, they are clamped to that value and the local clock declared unsynchronized.

Following is a detailed description of the ntptimeval structure members.

struct timeval time;

> This member is set to the current system time, expressed as a Unix timeval structure. The timeval structure consists of two 32-bit words, one for the number of seconds past 1 January 1970 and the other the number of microseconds past the most recent second's epoch.

long maxerror;

> This member is set to the value of the time_maxerror kernel variable, which establishes the maximum error of the indicated time relative to the primary reference source, in microseconds. This variable can also be set and read by the ntp_adjtime() system call. For NTP, the value is determined as the synchronization distance, which is equal to the root dispersion plus one-half the root delay. It is increased by a small amount (time_tolerance) each second to reflect the clock frequency tolerance. This variable is computed by the time-synchronization daemon and the kernel and returned in a ntp_gettime() system call, but is otherwise not used by the kernel.

long esterror;

> This member is set to the value of the time_esterror kernel variable, which establishes the estimated error of the indicated time relative to the primary reference source, in microseconds. This variable can also be set and read by the ntp_adjtime() system call. For NTP, the value is determined as the root dispersion, which represents the best estimate of the actual error of the local clock based on its past behavior, together with observations of multiple clocks within the peer group. This variable is computed by the time-synchronization daemon and returned in a ntp_gettime() system call, but is otherwise not used by the kernel.

B.2. The ntp_adjtime() System Call

The syntax and semantics of the ntp_adjtime() call is given in the following fragment of the timex.h header file. Note that, as in the ntp_gettime() system call, the syscall.h system header file must be modified to define the SYS_ntp_adjtime system call specific to each system type.

```
/*
 * NAME
 *      ntp_adjtime - NTP daemon application interface
 *
 * SYNOPSIS
 *      #include <sys/timex.h>
```

```
 *
 *      int system call(SYS_ntp_adjtime, mode, tptr)
 *
 *      int SYS_ntp_adjtime             defined in syscall.h header file
 *      struct timex *tptr              pointer to timex structure
 *
 * NTP daemon interface - used to discipline kernel clock oscillator
 */
struct timex {
        int mode;                       /* mode selector */
        long offset;                    /* time offset (usec) */
        long frequency;                 /* frequency offset (scaled ppm) */
        long maxerror;                  /* maximum error (usec) */
        long esterror;                  /* estimated error (usec) */
        int status;                     /* clock command/status */
        long time_constant;             /* pll time constant */
        long precision;                 /* clock precision (usec) (read only) */
        long tolerance;                 /* clock frequency tolerance (ppm)
                                         * (read only) */
};
```

The ntp_adjtime() system call is used to read and write certain time-related kernel variables summarized in this and subsequent sections. Writing these variables can only be done in superuser mode. To write a variable, the mode structure member is set with one or more bits, one of which is assigned each of the following variables in turn. The current values for all variables are returned in any case; therefore, a mode argument of zero means to return these values without changing anything.

Following is a description of the timex structure members.

int mode;

This is a bit-coded variable selecting one or more structure members, with one bit assigned each member. If a bit is set, the value of the associated member variable is copied to the corresponding kernel variable; if not, the member is ignored. The bits are assigned as given in the following fragment of the timex.h header file. Note that the precision and tolerance are intrinsic properties of the kernel configuration and cannot be changed.

```
/*
 * Mode codes (timex.mode)
 */
#define ADJ_OFFSET              0x0001      /* time offset */
#define ADJ_FREQUENCY           0x0002      /* frequency offset */
#define ADJ_MAXERROR            0x0004      /* maximum time error */
#define ADJ_ESTERROR            0x0008      /* estimated time error */
#define ADJ_STATUS              0x0010      /* clock status */
#define ADJ_TIMECONST           0x0020      /* pll time constant */
```

long offset;

If selected, this member (scaled) replaces the value of the time_offset kernel variable, which defines the current time offset of the phase-lock loop. The value must be in the range ±130 ms in the present implementation. If so, the clock status is automatically set to TIME_OK.

long time_constant;

If selected, this member replaces the value of the time_constant kernel variable, which establishes the bandwidth of "stiffness" of the kernel PLL. The value is used as a shift, with the effective PLL time constant equal to a multiple of 1 << time_constant, in seconds. The optimum value for the time_constant variable is log2(update_interval) - 4, where update_interval is the nominal interval between clock updates, in seconds. The default value of zero is appropriate with a radio clock and update intervals of about 16 seconds and corresponds to a PLL time constant of about 15 minutes. Longer update intervals can be used as indicated by host or network overheads, but values larger than seven are not ordinarily useful, unless the local clock timebase is derived from a precision oscillator.

long frequency;

If selected, this member (scaled) replaces the value of the time_frequency kernel variable, which establishes the intrinsic frequency of the local clock oscillator. This variable is scaled by 1 << SHIFT_KF in parts-per-million (ppm), giving it a maximum value of about ±130 ppm and a minimum value (frequency resolution) well down into the noise.

long maxerror;

If selected, this member replaces the value of the time_maxerror kernel variable, which establishes the maximum error of the indicated time relative to the primary reference source, in microseconds. This variable can also be read by the ntp_gettime() system call. For NTP, the value is determined as the synchronization distance, which is equal to the root dispersion plus one-half the root delay. It is increased by a small amount (time_tolerance) each second to reflect the clock frequency tolerance. This variable is computed by the time-synchronization daemon and the kernel and returned in a ntp_gettime() system call, but is otherwise not used by the kernel.

long esterror;

If selected, this member replaces the value of the time_esterror kernel variable, which establishes the estimated error of the indicated time relative to the primary reference source, in microseconds. This variable can also be read by the ntp_gettime() system call. For NTP, the value is determined as the root dispersion, which represents the best estimate of the actual error of the local clock based on its past behavior, together with observations of multiple clocks within the peer group. This variable is computed by the time-synchronization daemon and returned in a ntp_gettime() system call, but is otherwise not used by the kernel.

int status;

If selected, this member replaces the value of the time_status kernel variable, which records whether the clock is synchronized, waiting for a leap second, etc. In order to set this variable explicitly, either (a) the current clock status is TIME_OK or (b) the member value is TIME_BAD; that is, the ntp_adjtime() call can always set the clock to the unsynchronized state or, if the clock is running correctly, can set it to any state. In any case, the ntp_adjtime() call

always returns the current state in this member, so the caller can determine whether or not the request succeeded.

long precision;

This member is set equal to the time_precision kernel variable in microseconds upon return from the system call. This variable, which cannot be written, represents the maximum error in reading the local clock. The value, which is ordinarily equal to the kernel variable *tick*; but, in the modified kernels with microtime() routine, the value is 1 μs. This variable is computed by the kernel for use by the time-synchronization daemon, but is otherwise not used by the kernel.

long tolerance;

This member is set equal to the time_tolerance kernel variable in parts-per-million (ppm) upon return from the system call. This variable, which cannot be written, represents the maximum frequency error or tolerance of the particular platform and is a property of the architecture and manufacturing process.

B.3. Command/Status Codes

The kernel routines use the local clock status variable time_status, which records whether the clock is synchronized, waiting for a leap second, etc. DThe value of this variable is returned as the result code by both the ntp_gettime() and ntp_adjtime() system calls. In addition, it can be explicitly read and written using the ntp_adjtime() system call, but can be written only in superuser mode. Values presently defined in the timex.h header file are as follows:

```
/*
 * Clock command/status codes (timex.status)
 */
#define TIME_OK    0            /* clock synchronized */
#define TIME_INS   1            /* insert leap second */
#define TIME_DEL   2            /* delete leap second */
#define TIME_OOP   3            /* leap second in progress */
#define TIME_BAD   4            /* clock not synchronized */
```

A detailed description of these codes as used by the leap-second state machine is given elsewhere in this document. In case of a negative result code, the kernel has intercepted an invalid address or, for the ntp_adjtime() system call, a superuser violation.

B.4. Kernel Variables

The following kernel variables are defined by the new code:

```
long time_offset = 0;           /* time adjustment (usec) */
```

This variable is used by the PLL to adjust the system time in small increments. It is scaled by 1 << SHIFT_UPDATE in binary microseconds. The maximum value that can be represented in a 32-bit longword is about ±130 ms and the minimum value or precision is one microsecond.

```
long time_constant = 0;         /* pll time constant */
```

This variable determines the bandwidth or “stiffness” of the PLL. It is used as a shift, with the effective value in positive powers of two. The default value (0) corresponds to a PLL time

constant of about 15 minutes. The update interval should not be more than (1 << time_constant) * 16 in seconds; so, with this default, the update interval is 16 s. These values are appropriate when the server and client are both located on a fast LAN and yields the lowest jitter and highest accuracy. For use with remote servers, where the delay is tens of milliseconds or more, the time constant and update interval should be increased by a factor of 4 or more, or an increase in time_constant of 2. Time_constant values larger than 6 are probably not useful, unless the local clock timebase is derived from a precision oscillator.

```
long time_tolerance = MAXFREQ;          /* frequency tolerance (ppm) */
```

This variable represents the maximum frequency error or tolerance of the particular platform and is a property of the architecture. It is expressed as a positive number greater than zero in parts-per-million (ppm). The default MAXFREQ (100) is appropriate for conventional workstations.

```
long time_precision = 1000000 / HZ;     /* clock precision (usec) */
```

This variable represents the maximum error in reading the local clock. It is expressed as a positive number greater than zero in microseconds and is usually based on the number of microseconds between timer interrupts, 3906 μs for the Ultrix kernel, 976 μs for the OSF/1 kernel. However, in cases where the time can be interpolated between timer interrupts with microsecond resolution, such as in the unmodified SunOS kernel and modified Ultrix and OSF/1 kernels, the precision is specified as 1 μs. This variable is computed by the kernel for use by the time-synchronization daemon, but is otherwise not used by the kernel.

```
long time_maxerror;                     /* maximum error */
```

This variable establishes the maximum error of the indicated time relative to the primary reference source, in microseconds. For NTP, the value is determined as the synchronization distance, which is equal to the root dispersion plus one-half the root delay. It is increased by a small amount (time_tolerance) each second to reflect the clock frequency tolerance. This variable is computed by the time-synchronization daemon and the kernel, but is otherwise not used by the kernel.

```
long time_esterror;                     /* estimated error */
```

This variable establishes the estimated error of the indicated time relative to the primary reference source, in microseconds. For NTP, the value is determined as the root dispersion, which represents the best estimate of the actual error of the local clock based on its past behavior, together with observations of multiple clocks within the peer group. This variable is computed by the time-synchronization daemon and returned in system calls, but is otherwise not used by the kernel.

```
long time_phase = 0;                    /* phase offset (scaled us) */
long time_freq = 0;                     /* frequency offset (scaled ppm) */
time_adj = 0;                           /* tick adjust (scaled 1 / HZ) */
```

These variables control the phase increment and the frequency increment of the local clock at each tick. The time_phase variable is scaled by 1 << SHIFT_SCALE in microseconds, giving a maximum adjustment of about ±128μs/tick and a resolution of about 60 femtoseconds/tick. The time_freq variable is scaled by 1 << SHIFT_KF in parts-per-million (ppm), giving it a

maximum value of over ±2000 ppm and a minimum value (frequency resolution) of about .001 ppm. The time_adj variable is the actual phase increment in scaled microseconds to add to time_phase once each tick. It is computed from time_phase and time_freq once per second.

```
long time_reftime = 0;                  /* time at last adjustment (s) */
```

This variable is the second's portion of the system time on the last call to adjtime(). It is used to adjust the time_freq variable as the time since the last update increases.

```
int fixtick = 1000000 % HZ;             /* amortization factor */
```

In some systems such as the Ultrix and OSF/1 kernels, the local clock runs at some frequency that does not divide the number of microseconds in the second. In order that the clock runs at a precise rate, it is necessary to introduce an amortization factor, in microseconds, into the local timescale, in effect a leap-multimicrosecond. This is not a new kernel variable, but a new use of an existing one.

B.5. Architecture Constants

Following is a list of the important architecture constants that establish the response and stability of the PLL and provide maximum bounds on behavior in order to satisfy correctness assertions made in the protocol specification. These are for the Ultrix kernel.

```
#define HZ 256                          /* timer interrupt frequency (Hz) */
#define SHIFT_HZ 8                      /* log2(HZ) */
```

The HZ define (a variable in some kernels) establishes the timer interrupt frequency, 100 Hz for the SunOS kernel, 256 Hz for the Ultrix kernel and 1024 Hz for the OSF/1 kernel. The SHIFT_HZ define expresses the same value as the nearest power of two in order to avoid hardware multiply operations. These are the only parameters that need to be changed for different kernel timer interrupt frequencies.

```
#define SHIFT_KG 8                      /* shift for phase increment */
#define SHIFT_KF 20                     /* shift for frequency increment */
#define MAXTC 6                         /* maximum time constant (shift) */
```

These defines establish the response and stability characteristics of the PLL model. The SHIFT_KG and SHIFT_KF defines establish the damping of the PLL and are chosen by analysis for a slightly underdamped convergence characteristic. The MAXTC define bounds the maximum time constant of the PLL as a sanity check.

```
#define SHIFT_SCALE (SHIFT_KF + SHIFT_HZ) /* shift for scale factor */
#define SHIFT_UPDATE (SHIFT_KG + MAXTC) /* shift for offset scale factor */
#define FINEUSEC (1 << SHIFT_SCALE) /* 1 usec in scaled units */
```

The SHIFT_SCALE define establishes the decimal point on the time_phase variable which serves as a an extension to the low-order bits of the local clock variable. The SHIFT_UPDATE define establishes the decimal point of the phase portion of the ntp_adjtime() update. The FINEUSEC define represents 1 μs in scaled units.

```
#define MAXPHASE 128000                 /* max phase error (usec) */
#define MAXFREQ 100                     /* max frequency error (ppm) */
```

```
#define MINSEC 16                 /* min interval between updates (s) */
#define MAXSEC 1200               /* max interval between updates (s) */
```

These defines establish the performance envelope of the PLL, one to bound the maximum phase error, another to bound the maximum frequency error and the remaining two to bound the minimum and maximum time between updates. The intent of these bounds is to force the PLL to operate within predefined limits in order to conform to the correctness models assumed by time-synchronization protocols like NTP and DTSS. An excursion which exceeds these bounds is clamped to the bound and operation proceeds accordingly. In practice, this can occur only if something has failed or is operating out of tolerance, but otherwise the PLL continues to operate in a stable mode. Note that the MAXPHASE define conforms to the maximum offset allowed in NTP before the system time is reset (by settimeofday(), rather than incrementally adjusted (by ntp_adjtime()).