# A Kernel Programming Interface for Precision Time Signals

Technical Memorandum

Sponsored by: ARPA/ARMY contract DABT63-95-C-0046, NSF grant NCR-93-01002 and NSWC/NCEE contract A30327-93

David L. Mills
Electrical Engineering Department
University of Delaware
31 January 1996

## 1. Introduction

In this memorandum a generic programming interface is proposed which provides a hardware and software interface for precision timing signals, such as the pulse-per-second (PPS) signal generated by some radio clocks and cesium oscillators. It argues for a generic capability in the ubiquitous Unix kernel, which could be used for a wide variety of measurement applications, including network time synchronization and experiments involving performance measurement and evaluation of computer networks and transmission systems. The hardware to do this requires only a serial port and a modem control lead, such as the data carrier detect (DCD) lead, which can be driven by an external source via a level converter/pulse generator. Appropriate kernel modifications to support a generic measurement facility using this signal are described in [MIL96b], along with specimen segments of kernel code that has been implemented in Unix kernels for Sun, HP and DEC workstations. A discussion of generic measurement technique is given in [MIL96a].

## 2. Design Considerations

It remains to specify a generic programming interface with which portable programs can make use of this facility independent of specific kernel implementation. This would ordinarily be achieved by lobby of the POSIX apparatus, which is to be pursued. Meanwhile, the several schemes for improving timekeeping precision suggested elsewhere require some degree of craft, if coexistence with current operating system conventions is to be preserved. There are a number of ways, some more suited for product maintenance than others, as described below.

1. The required feature support is included in the kernel sources distribution and controlled by a compiler switch set at kernel build time. If compiled, the feature is always enabled. This is how the precision clock modifications (microtime()) are implemented in the DEC MIPS and Alpha kernels.

2. The required feature support is included in the kernel sources distribution and controlled by a compile switch set at kernel build time. If compiled, the feature must be selectively activated using special system calls ntp_gettime() and ntp_adjtime() at run time. This is how the phase-lock loop modifications are implemented in the present Sun, DEC and HP kernels.

3. The required feature support is provided by an optional module which is dynamically loaded and activated at run time. The feature is enabled only if loaded and requires no change to the stock kernel. This is how the line disciplines tty_clk and chu_clk for timestamp capture are implemented in the Sun kernel.

## 3. Proposed Interface

The present implementation strategy for kernel modifications has been designed for experiment and evaluation; therefore, some care has been taken for a provision to reliably disable the features, should their use cause problems in normal system operation. In addition to this requirement, the serial port driver modifications suggested in this memorandum need to be controlled on a line-by-line basis, since there will very likely be some ports running standard terminal support and some running the modified support.

This requires some means to enable and disable the various features, which is most convenient using special ioctls. While this could be done in a number of ways, the following design may be typical. These ioctls are in addition to the ntp_gettime() and ntp_adjtime() ioctls mentioned above. Each ioctl is issued on an open file descriptor associated with a serial port (tty) device. Following is a specimen description of the calling sequences for these ioctls. The names are for illustration only.

### 3.1. timestamp_intercept() - set intercept character

This ioctl enables and disables the feature which inserts a timestamp in the input buffer following one of a set of specified intercept characters. The argument is a pointer to a zero-terminated list of ASCII intercept characters. If an input character matches one of these characters, a timestamp in Unix timeval format is captured and inserted in the input buffer immediately following the character. An argument string consisting of a single null character disables the feature. A side effect of an intercept character is to capture a timestamp for later retrieval using the fetch_timestamp() ioctl.

### 3.2. control_dcd() - control DCD signal

This ioctl enables and disables selected features associated with the data carrier detect (DCD) signal on a serial port. The argument is a pointer to a 32-bit control word. Bits can be set in this word to enable or disable various options, including:

ENABLE_DCD When reset (default), operation of the serial port is unchanged and the DCD signal of the serial port processed as specified in the terminal interface structure. When set, a DCD signal transition of minimum specified amplitude and duration and selected polarity causes the driver to capture a timestamp for later retrieval using the fetch_timestamp() ioctl (see below). Note that, when this bit is set, serial port modem control is disabled as if the LOCAL bit is set in the terminal interface structure.

SINGLE_DCD When reset (default), DCD timestamp capture is enabled whenever the enable_dcd bit is set. In this case, later timestamps can overwrite earlier ones. When set, capture is automatically disabled following the first event and must be enabled again, either by another control_dcd() ioctl or by a fetch_timestamp() ioctl. In this case, DCD transitions that occur while in the not-enabled state are lost and may or may not be indicated by a subsequent error return.

NEGATIVE_DCD When reset (default), the active DCD transition is set to the positive-going edge. When set, the active transition is set to the negative-going edge. In this connection, "positive" and "negative" refer to the RS-232 electrical signal description.

### 3.3. fetch_timestamp() - fetch DCD timestamp

This ioctl returns a timestamp previously captured at a timestamp event, either as the result of an intercept character specified by the timestamp_intercept() ioctl, or a DCD transition enabled by the control_dcd() ioctl. The argument is a pointer to a structure of two members, the first a Unix timeval structure and the second a 32-bit integer. Upon return, the timeval structure contains the system time at the most recent signal event and the integer contains the sequence number of that event.

### 3.4. Signals

In some applications, it would be useful to provide a signal interrupt in a way similar to other devices. This would be possible only if there were a pre-existing mechanism to present modem control status transitions as signals, in which case the DCD signal would be raised at the same time the timestamp is captured. Whether this feature should be provided as a special option or a standard feature is for further study.

### 3.5. Error processing

The three ioctls defined above return a status code in the fashion typical of other ioctls of this type. In addition to the usual argument and file descriptor checks, it may be useful to do some error checking on the external signal itself. Following are some typical checks and suggested recovery actions.

Noise check In order to avoid possible kernel lockup due to an excessively noisy DCD signal or high interrupt frequency, the serial port chip modem control interrupt-enable line can be disabled immediately following an interrupt. The line can be re-enabled by any of the above three ioctls or automatically after a nominal delay in the order of 10 ms. If there is a missed-transition error bit in the modem control status word, an indication should be provided in the ioctl return status code.

Sequence check The fetch_timestamp() ioctl returns the sequence number of the most recent timestamp event, but otherwise does no checking for lost events. In many applications, lost events do not affect the application processing. Where it is necessary to know if an event is lost, the application can use the sequence numbers to check for gaps.

## 4. References and Bibliography

Note: The following publications are available from the web page http://www.eecis.udel.edu/~mills.

[MIL96b] Mills, D.L. A Kernel Programming Interface for Precision Time Signals. Electrical Engineering Technical Memorandum, January 1996.

[MIL93] Mills, D.L. Precision synchronization of computer network clocks, Electrical Engineering Department Report 93-11-1, University of Delaware, November 1993, 66 pp.

[MIL94b] Mills, D.L. A kernel model for precision timekeeping. Electrical Engineering Department Report 94-10-1, University of Delaware, October, 1994, 34 pp.

[MIL95] Mills, D.L. Improved algorithms for synchronizing computer network clocks. IEEE/ACM Trans. Networks (June 1995), 245-254.

[MIL96a] Mills, D.L. Time and Time Interval Measurement with Application to Computer and Network Performance Evaluation. Electrical Engineering Technical Memorandum, January 1996, 17 pp.

[MIL96b] Mills, D.L. A Kernel Model for Precision Timekeeping. Electrical Engineering Technical Memorandum, January 1996, 28 pp.