NAME

syd – periodically updates local timing data using a remote NTP server

SYNOPSIS

syd [**-m** *time* **-r** *number* **-e** *time* **-d** *time* **-w** *time* **-k** *number* **-f** *filename*] *host name*

DESCRIPTION (basic features)

syd (a SYncronization Daemon) creates a daemon that keeps updated a file (the **interface file**) containing the timing data needed to obtain an accurate approximation of the UTC time value starting from the system clock value. The user can have access to the data encoded in the interface file by including the appropriate header and linking the library to his the program, or by using the shell commands provided. See below for a description of **toutatis** (TOolset for a Universal Time Accurate TimeStamping).

Since the daemon does not touch at sensible data (namely, the system clock), it does not necessarily need special user privileges to be run, and any user or application may have his own time reference.

The daemon uses a probabilistic protocol in order to obtain the timing from a NTP server indicated by the user in the mandatory *host-name* parameter.

During its execution, syd outputs on the stderr a complete log of its activity. This data are used for testing and tuning purposes, and should be re-directed in a log file.

Syd may terminate as an effect of four kinds of event:

- if it receives a SIGINT signal;
- if an unrecoverable networking error occurs;
- if the network quality of service does not allow to maintain the requested accuracy.
- if a timing inconsistency is detected;

The first event causes the normal termination of the daemon. The others are regarded as anomalous (see below the **DIAGNOSTICS** section).

The operation of the daemon is controlled by a number of options, whose configuration is both critical and non-trivial. Below we list the options, and in a separate section we indicate a simple protocol to obtain a reasonable (though non optimal) configuration. The default values hardwired in the command provide a foolproof but coarse grain (1 second accuracy) synchronization that can be helpful for testing and for the first phase of the tuning protocol (see **TUNING PROTOCOL** below).

OPTIONS

-m time
indicates the minimum transmission delay between the local host and the remote NTP server (in seconds) or vice-versa; default is 0;

-r number
indicates the expected drift of the local clock: it is a ratio expressed in seconds per seconds; default is 0.0002, more than 15 seconds per day;

-e time indicates the lower threshold for the precision (in seconds); when this threshold is reached, the program has reached contact with the server; default is 0.5 seconds;

-d time indicates the upper threshold for the precision (in seconds); when this precision is reached the program has lost contact with the server; default is 1 second;

-w time
indicates the interval between two successive retries (in seconds), when the first one does not reach the lower threshold precision; the default is 60 seconds;

-k number
the maximum number of retries that the program is allowed to make before the lower threshold precision is reached; the default is 10.

-f filename

the name of the interface file that will contain the timing data maintained by the daemon; default "lc.syd" in the directory where syd in invoked;

EXAMPLES

The following is the simplest invocation:

syd NTP-server

where the only parameter indicates the server that provides the NTP service to the local-domain network. Note that the time provided by the server may or may not reflect the UTC: the only requirement is that its drift with respect to UTC is constant.

This is a sample line with all options in use:

syd -m0.02 -r0.0001 -e0.05 -d0.1 -w60 -k5 -f log x.y.z

that means that the program will try to synchronize the local clock with the one of host x.y.z, that the minimum time to communicate with that node is estimated in 20 milli-seconds, that the drift of the local clock is estimated as 100 parts per million (which is a quite pessimistic estimate in most cases), that the clock accuracy must remain below one 100 milli-seconds (this is adequate for most multimedia applications), and that the protocol can rest when the accuracy is below 50 milli-seconds. In case Syd does not reach this precision using 5 retries separated by a 60 seconds lapse, the accuracy is set to the best one obtained during the preceding retries. If this is below the worst case accuracy of 100 msecs, the program will schedule another run after an appropriate lapse of time. If the precision is not acceptable, the program will terminate with a failure.

DIAGNOSTICS

The command returns the following return codes:

0 The program is interrupted by an INT signal. This corresponds to the normal termination.

-1 Abnormal termination. The last lines in the stderr output indicate the reason of the trouble.

Here we concentrate on the basic diagnostics, that can be used to understand the reasons of a failure. The next section is dedicated to the analysis of the log to obtain an acceptable tuning.

There are 3 classes of abnormal termination:

- if an unrecoverable networking error occurs: network unreachable, inexistent host etc. A short diagnostic message is printed on the log.
- if the network quality of service does not allow to maintain the requested accuracy. This occurs when the communication speed degrades for a long period of time, so that the protocol cannot resume in the following retries. This is the case when a network reconfiguration alters the expected behavior. The content of the log should indicate what happened, and whether to modify the parameters.
- if a timing inconsistency is detected. This is the case when some of the indicated parameters are not consistent with the observation. For instance, if the communication takes less than 40 milliseconds in the example above (remember that the minimum communication delay was indicated as 20 msecs) we will probably have a timing inconsistency. The program makes a number of consistency checks, so that when the parameters do not respond to the reality, it is likely that an "inconsistent timing" notification will appear. The content of the log should indicate where is the fault.

As far as possible, we have eliminated unsafe failure modes: the administrator is always informed about the reason of the failure, and obtains enough information to correct the configuration.

TUNING PROTOCOL

The following protocol may help to find appropriate values to use for the option parameters.

- the only information that must be necessarily provided is the server host-name. Browse the web at

  http://www.eecis.udel.edu/~mills/ntp/clock2.html

or

http://www.eecis.udel.edu/~mills/ntp/clock1.html

and select a number of NTP servers near to you, and that have suitable access policies;

- run simultaneously a different instance of **syd** on each server for a couple of hours using the default values, except for the host, and for the data file (that must be different for each instance of **syd).** Collect the output of each activation in a separate file, by redirecting the output of each **syd** command on a different file;
- concerning the daemons that spontaneously terminate before you kill them, you can conclude that either the line to the NTP server is too slow, or the NTP server is not sufficiently accurate, or both. In any case, they are useless for the minimal need of having a timing accuracy of 1 second. You can discard them from the following analysis.
- using the **analyse.sh** script you analyze the output file obtained by redirecting the output of **syd.** Just passing the name of the file as parameter to the script you obtain a harvest of important facts about the operation of syd.

  You can remove from your selection those servers for which **syd** times out frequently: compare the line "retries that timed out" with the previous lines to discover this.

  The indicated **drift** is an approximation of the real drift of your system clock in parts per million. You can safely double this figure, convert it in seconds per seconds (i.e. divide it by 10E6), and use it as the option parameter following **-r.** Then use the median of the **round-trip** and the minimum **accuracy of the server** to compute the values for the **-e** and **-d** options. The former one can take the value of the **half of the median** of the round-trip time incremented with the **minimum** precision of the server (converted in seconds) and the latter can be set to two or three times the value used for -e. You can also select those servers that exhibit a better (lower) value for the "precision after an attempt".
- you run a syd instance for each of the NTP servers selected in the previous step, without redirecting the output on a file. Take the value indicated in the first line "Delay to next attempt (dna)" and interrupt the program (ctrl-C). Divide the value (in seconds) by 20, and use the computed value for the command line parameter following -w;
- you can use the values computed in the above steps to run one or more instances of Syd, collecting the standard outputs on files. This time the run can span several days, and you can safely use the timing information provided by syd for your purposes. At the end of the experiments, check that the value of the median of the computed drifts did not exceed the drift indicated in the -r command line parameter, and that most attempts are successful at the first attempt. Also double-check that the value of the -w command line parameter satisfied the thumb rule indicated above. Now syd is configured to keep the timing information with an accuracy corresponding to the command line parameter indicated after -d, with a probability of success, for each attempt, of 99.9%. This should correspond to a Mean Time Between Failures of some months, and is quite appropriate for most applications. In the case that you are not satisfied with that accuracy, you can proceed with the following refinement steps;
- analyzing the result of the script on the output file of the selected server, consider the lines "Precision of the server". You cannot obtain an accuracy better than that indicated as the minimum accuracy of the server. Chose the servers that have an accuracy lower than your target accuracy, and observe the values of the "minimum round-trip time". Divide this figure by four, convert it in seconds, and use it for the command line parameter following "-m". Decrement the value formerly computed for the -e parameter, and recompute the value of the -d parameter as indicated above. You should run a further long experiment after this step, to ensure that it never occurs that the round-trip is lower than four times the figure indicated after the "-m".
- if the previous check is ok, and you are not yet satisfied with your accuracy, you can exploit the probabilistic nature of the algorithm: instead of the median of the round-trip time, you can use one of the previous lines (75%, 90%, etc.). Use the new value as indicated in the preceding step to

compute a new value for the -e and -d command line parameters. For each line you scale up, add one unit to the value of of the -k command line parameter. After this step, the synchronization attempts will not be successful at the first try.

- further refinements are still possible, but I have not found any easy recipe for them. You should first understand the formal foundations of the algorithms used in the program reading the papers indicated below (see FURTHER READINGS), and then help yourself...

ENVIRONMENT

none

THE TOUTATIS PACKAGE

In this section we give an overview of the components of the tool set: a more precise reference can be found in the specific man pages.

Syd is the main component of a tool-set. The role of **syd** is that of maintaining a small piece of data that can be used in order to compute the reference time (i.e. the time at the NTP server) using the local system clock. This information is contained in a *interface file* that can be read by other applications. In this way, the same **syd** process can serve several users.

However, the operation of accessing a **interface file** is time consuming, and the accuracy of the timestamp obtained reading the file first, and then using the data will drop to unacceptable levels. Therefore the data in read in advance from the file, and is used when needed: we call the data structure used to store the information from the *interface file* a *logical clock,* represented by a data type **struct log_clock.** The constructor for this data type is invoked with the function **new_log_clock.**

The are two main functions that retrieve and store an interface file, called respectively **read_log_clock** and **write_log_clock.**

Once the interface file is read, its content can be used repeatedly, and for a relatively long time, in order to produce accurate timestamps. The function that implements this operation id the **get_log_clock.**

Since no operation involved in the production of the timestamp uses external or shared resources, this operation is extremely fast, and does not affect the original accuracy. However, the accuracy gradually degrades, as an effect of the uncompensated drift of the system clock. When a new timestamp is obtained from a logical clock, the user application can check whether the accuracy is appropriate, since the **get_log_clock** returns an estimate of the accuracy of the timestamp.

Other functions exist that inspect the state of the logical clock structure:

is_accurate
: checks whether the logical clock is usable;

get_drift
: returns the computed drift of the system clock with respect to the NTP server clock;

get_offset
: returns the computed offset from the NTP server clock to the system clock;

get_owner
: returns the pid of the **syd** process that maintains that logical clock (on behalf of the **interface file);**

get_stamp
: returns the number of times the interface file has been updated;

get_leap_second
: checks the presence of a leap second;

log_clock2c
: returns a human readable string summarizing the content of a logical clock.

A few functions alter the data contained in the logical clock:

set_log_clock
: updates a logical clock using a time value provided as parameter;

reset_leap_second
: resets the leap second indication

set_leap_second
: sets the leap second indication;

SAMPLE APPLICATIONS

We have included some sample applications, whose code exemplifies the use of the tool-set; see the man pages for further details:

syd_kill
: kills the syd process that maintains a given interface file;

syd_stat
: prints a verbose description of the content a given interface file;

syd_date
: prints the current date, according with the data in a given interface file;

syd_setup
: corrects the system clock using the content of a given interface file. Read carefully the man page before use.

HISTORY

The very basic idea comes from paper [1]. G. Alari worked out an effective improvement of it, during his master thesis: shortly after his graduation in 1992, he performed several experiments and proved its effectiveness. We gathered the results, and I added a probabilistic analysis of the behavior of the protocol. The final results of this joint work are in [2].

In order to make the idea practically applicable, we decided to write a client for one of the existing protocols: NTP (see [3]) proved to be excellent for this purpose. The work of adapting and implementing the algorithm was appointed as master thesis job to Luigi Piunti, who wrote the first prototype [4].

I have been the supervisor of this thesis too. After the graduation of L.Piunti, I debugged and completely re-engineered the prototype, and prepared a public domain release. The first release is dated February 1997.

The second release, with additional servers, is dated May 1997.

In the third release I abandon the approach of synchronizing the system clock, and of providing built-in fault tolerance. Instead, syd maintains the timing information in an interface file, and other commands use this information to provide a time reference to other applications (and possibly to synchronize the system clock. The whole clock synchronization activity is therefore decomposed into several modules, and syd comes as a fundamental one. This version is strongly influenced by the experience with JAVA_syd. It is dated January 1998.

A further release dated August 1999 improves the diagnostics and the documentation. An extensive testing in a real scale environment is also planned.

SEE ALSO

syd_date(1), syd_kill(1), syd_setup(1)

[1] Cristian,F. "Probabilistic Clock Synchronization", Distributed Computing, 3(1989):146-158.

[2] Alari,G. and Ciuffoletti,A. "Implementing a Probabilistic Clock Synchronization", Real Times Systems, 13(1) (to appear).

[3] Mills,D.L. "Internet Time Synchronization", IEEE Transactions on Communications, COM-39(10):1482-1493.

[4] Piunti,L. "Un cliente NTP per la sincronizzazione probabilistica del clock", Tesi di Laurea, Universita' di Pisa.

BUGS

The purpose of this tool has been primarily the test of some abstract conclusions: as a consequence the user may find that some behaviors would be unacceptable for a commercial product. We apologize for this, and invite anybody which is interested in turning this into a real product to get in touch with the author (see below). Bugs list:

- Does not diagnose correctly some network problems (like an inexistent host).
- Little diagnostics when input data are inconsistent.
- Remember that Unix is **NOT** a real-time operating system...
- Syd can't cope with leap second days (one is at the end of June 1997).

The operation of syd is without notice if the system clock is manually or automatically adjusted, causing deviations incompatible with the maximum drift indicated as command line parameter after -r option.

IDENTIFICATION

Author: Augusto Ciuffoletti (augusto@di.unipi.it).
Contributors: Gianluigi Alari, Luigi Piunti.