LA-UR- 94-2790

Conf-950233--1

NERD

Network Event Recording Device: An Automated System for Network Anomaly Detection, and Notification (Draft)

David G. Simmons, Network Engineering
Ronald Wilkins, Network Engineering
Los Alamos National Laboratory

1.0 Overview

1.1 The Problem

The goal of the Network Event Recording Device (NERD) is to provide a flexible, reliable, and autonomous system for network logging and notification when significant network anomalies occur. The NERD is also charged with increasing the efficiency and effectiveness of currently implemented network security procedures.

While it has always been possible for network and security managers to review log files for evidence of network irregularities, the NERD provides real-time display of network activity, as well as constant monitoring and notification services for managers. Similarly, real-time display and notification of possible security breaches will provide improved effectiveness in combating resource infiltration from both inside and outside the immediate network environment.

1.2 Background

The Network Event Recording Device, or NERD, was originally begun by Sally Wilkins, Craig Idler, and Ben Crane in May, 1991. The originally defined goals of the NERD were to be "a service on a network for recording network events and affecting some appropriate notification that is determined by the criticality of the events. NERD may be a stand alone host or may be one of several network services on a host dedicated for such tasks." [11] While the basic functional description and purpose of the NERD has not changed substantially since that time, the implementation and features of the NERD have undergone radical revisions in order to make it a flexible, production quality system addressing the ever-widening problems of network management and security. The rewrite of the NERD, including the development of several new and important aspects of the system, was completed at Los Alamos National Laboratory by David G. Simmons under the direction of Ron Wilkins and Dale Land in the Network Engineering Group (CIC-5). Ongoing work to

continuously upgrade the capabilities and services provided by the NERD, as well as to address new and emerging network and security issues, is also being conducted.

1.3 Environment

The target environment for the NERD System is the Integrated Computing Network (ICN) at Los Alamos National Laboratory. The ICN is the central computing network, serving over 9000 users on a variety of supercomputers, mainframe, minicomputers and workstations, as well as file storage devices, communications interfaces, routers, bridges and terminals. The ICN at Los Alamos is one of the largest and most complex computing resources in the world. The systems within the ICN use a variety of operating systems, including UNICOS, AIX, ULTRIX, UNIX System V, BSD UNIX 4.3, and Sun's Solaris operating system, among others. The variety of operating systems present in the ICN was a central driving force in the final design of the NERD System.

In addition to hardware variations, the ICN is divided into several separate partitions based on four defined security levels. At this writing, the ICN is undergoing a major reorganization (the ICN2 Project) to further isolate these partitions and provide greater security against intrusion. In so doing, however, it has also complicated the task of centralized network monitoring and notification by requiring a complete physical separation between networks processing classified data, and those processing unclassified material, and thus a need for simultaneous services on the two networks [2].

1.4 Functional Overview

The NERD is not necessarily, as the name might imply, a single entity, but is rather a suite of programs that, when run together, constitute a sophisticated Network Event Monitoring System, a Notification System for significant Network Events, and a self-monitoring and diagnostic system to ensure reliable operation. The system also provides an intuitive, interactive, window-based interface to the information maintained by the NERD.

In order to minimize system maintenance requirements, and to allow a high degree of portability, the NERD is built on a standard Berkeley System Designs 4.3 UNIX ® syslogd daemon process, making the NERD effectively blind to hardware and operating system differences across a heterogeneous network such as the ICN at Los Alamos. The modified process is not run on all hosts on the network, but only on the NERD data server, allowing the NERD to provide the full range of its services to all hosts, or a specified subset of hosts, without software modifications.

The NERD provides host-authentication routines which allow a high degree of granularity in the control of access to the various logging capabilities allowed to remote hosts, notification abilities granted to remote hosts, and access to the NERD's log files. NERD uses the network log files generated on the various service nodes to monitor network activity. Since the NERD uses existing sources of data, we were able to incorporate them into the NERD with little impact on the client systems.

DISCLAIMER

The Authentication, Logging and Notification system, and the User Interface to those systems will be described here. In addition, the suite of support processes written to ensure reliability of the NERD will also be described.

2.0 General Issues

2.1 Motivating Factors

The NERD project is seen as a significant opportunity to increase efficiency and drive down costs in the areas of network management as well as network resource allocation. However, a significant aspect of the NERD's capabilities lie in increasing the effectiveness of present and future network security procedures through providing a flexible notification interface.

2.1.1 Network Efficiency

Given the dependency of the Los Alamos National Laboratory's Central Computing Facility on networked computers and network technology, a system for effectively managing those networks was required. Loss of a section of the network, or communications problems between segments of the network, have the potential to cause large-scale problems throughout the facility, and a set of tools for overseeing the functioning of the networks, and the machines on those networks, was seen as a critical area for improved network management. In addition to general Network monitoring, it was also important to be able to monitor the status of hardware on the networks in order to be able to avoid system crashes. The economic benefits of improved network security and performance are well recognized. [5*] Improved efficiency in managing networks, and network resources, has the potential to decrease the financial impact of network maintenance and management on network providers and network managers.

The problem of effective network management applies equally to large-scale computing facilities and Local and Wide Area Networks. The management of resources in a large Computing Facility, such as Los Alamos, with its high-speed networks of supercomputers, mainframes, and minicomputers can be significantly impacted by mismanagement of network resources. A simple example of network resource mismanagement is an infinite loop in email forwarding, creating a "cycled user." A user forwards his mail from machine X to machine Y. Forgetting he has done this, he later forwards his mail from machine Y to machine X. Though most mail protocols will *age* messages and discard them, the effect of large numbers of cycling email messages on a network can be substantial. A security profiler may also see such activity as a potential security violation, and take unwarranted action against a user or a series of hosts on the network to curtail the activity. [6*] In this way, resource degradation on even small Local Area Networks of workstations can significantly reduce productivity and drive up the cost of doing business.

By reducing the difficulty of network resource monitoring and management, and therefore increasing the network managers' efficiency, the NERD has the potential to realize significant time and resource savings.

2.1.2 Security Enhancement

An important, and often overlooked, avenue to increased security lies in the simple management of networks and their associated resources. Even using the most advanced network security technology available will not guarantee adequate security unless effective network management procedures are developed and followed. Issues of network security and resource management are of paramount importance in the Los Alamos National Laboratory Central Computing Facility. As with resource management, security management has the potential to directly influence the cost of maintaining and providing network services. Furthermore, even small breaches of security can lead to large-scale financial losses to network providers. [9, 3]

Given the growing presence of open systems, a large majority of which are based on Unix, and the inherent weaknesses in Unix, providing effective means of management in distributed, open systems becomes an increasingly important commitment. While security of the Unix operating system is increasing, the same can not always be said for users and applications on those systems. It is therefore imperative to implement a network-wide system of auditing and control to ensure adequate security and management of users and processes on open systems. [7, 1] While much of this functionality already exists, often the data generated and stored in log files is never adequately examined due to a lack of an effective interface to those files.

2.2 Analysis of and Integration with Existing Systems and Software

In designing a new network monitoring system, we felt it was imperative to design a system that could be rapidly integrated into existing network monitoring technology. Such a design approach would upon implementation, maximize the actual use of the system, and minimize the impact on users. Requiring all systems desiring to use the NERD's monitoring and notification systems to implement new software would, we felt, place an undue burden on system managers, and increase the maintenance requirements to such a level as to make the system impractical. As with any system, the easier it is to integrate the system into existing protocols and practices, the more likely it is to be used, and the more rapidly it will be used network-wide.

In order to make remote system modification unnecessary, the NERD is based on a modified Berkeley UNIX® syslogd (system logging daemon) process. By using a standard, well-known, and widely used network logging system, the NERD is able to be integrated quickly into an existing network structure without large-scale modifications to that structure. By running the modified syslogd process only on one machine, no software modifications are required network-wide. This approach allows for the rapid integration of the NERD Notification capabilities to existing systems.

Additionally, by using a widely available and well-understood mechanism, we were able to provide users with an easy interface to using and customizing their use of the NERD. The integration of the NERD with the existing ICN structure was completed with no measurable impact on existing services or performance. When the ICN2 structure is fully implemented, we expect similarly minimal impact.

The NERD was designed to continuously monitor and log all network events on any machine or set of machines without significant changes to those machines. In fact, any system running UNIX or any of its standard variants can use the NERD's central logging and archival system with a simple change in a configuration file as long as the operating system supports the use of a syslogd process. If the system does not support syslogd, the NERD can still be used, though with more difficulty, as long as the operating system permits transmission of UDP datagrams to a well-known address and port.

2.3 Cross-platform Adaptation

In order to deal effectively with the wide variety of hardware platforms and operating systems present in the ICN, it has at times been necessary to implement different software for handling data from the various service nodes on the network. [4] For the purposes of the NERD, this approach was seen as placing too high a burden on both the administrators of the various nodes, and those maintaining the NERD software in the future. For this reason an approach was pursued which would require no specialized software for differing hardware platforms and operating systems. Instead, we rely on the wide availability of UNIX and UNIX-like processes to handle the differences among hardware and operating system platforms.

This system-independent approach was not a part of the original design of the NERD, but rather was implemented as the design parameters changed. As originally specified, all hosts logging to the NERD would be required to run a client-version of the modified syslogd process. [1] This requirement was seen as too restrictive on the types and number of systems that could effectively use the NERD, and was changed to the more open implementation described above. Requiring that the modified NERD process be run only on one machine, the NERD host itself, enables all other hosts, regardless of hardware configuration or operating systems, to use the NERD while continuing to use any and all software already in use.

2.4 System Autonomy

By designing the NERD's operation to be independent of the presence or absence of any specific service nodes within the ICN, the NERD is able to function regardless of the status of any segment of the network environment, with the exception of the network connection of the NERD itself to the ICN backbone. Should a service node, or set of nodes, either cease operation, or suddenly begin utilizing the NERD, no significant changes need be made to the NERD's configuration. As new systems are brought into the ICN, they can be integrated into NERD without interruptions in existing services to other nodes.

2.5 Data Handling and Authentication

Authenticating and handling the large volume of data generated by the networks currently utilizing the NERD has proven to be a larger issue than at first imagined. The volume of data which the NERD handles has increased steadily as new systems are brought into ser-

vice, and we have been forced to adapt our data handling procedures to accommodate these changes.

2.5.1 Data Transmission

All incoming messages to the NERD are read from a standard set of well-known ports. From remote hosts, the NERD looks for messages on UDP port 514 [8*]. Additionally, the NERD listens for host generated messages on /dev/log, and for messages generated by the kernel itself on /dev/klog. The scheme used by NERD represents no change from the implementation provided by BSD 4.3+ UNIX syslogd, and is used for that reason. These input routes are all opened at start-up, and are continuously polled for incoming messages.

2.5.2 Data Storage

All information sent to the NERD is stored in log files maintained by the NERD. The NERD also stores all of its own events in these log files. The formats of these files will be discussed in section XXX. All NERD log files are backed up both locally and to the Common File System, the Laboratory's main file storage system, to ensure an accurate archive of network data, and the integrity of current network data in the event of a system crash. Should Network or Security managers need to review NERD log files, they are archived for 1 year on the Common File System. Current log files are kept resident on the data server for 7 days. The NERD logging daemon uses the log file /var/adm/NERD_log, while the display-driver process logs to /var/adm/video_log. The storage and backup procedures surrounding the video_log will be described in more detail in the data display section.

2.5.3 Host Authentication

Upon receipt of a message from one of the input sources, the sending host is extracted from the data packet header, and is verified against the NERD's list of allowed hosts. Only hosts listed in the authorization file (/etc/NERDhosts) are permitted to log events to the NERD. This is a deviation from both the standard implementation of syslogd, which allows any incoming message to be logged, and from the original design specifications of the NERD, which would have followed the syslogd implementation. The policy change was made in order to prevent unauthorized access to the NERD as well as to better control the amount of data logged by the NERD.

In the event of an attempted logging from an unauthorized host, the attempt, including the hostname making the attempt, is logged by the NERD, and the message is then discarded. All messages are time- and host-stamped with time of arrival and sending hostname for identification purposes.

The authentication routine implemented by the NERD currently relies on the integrity of the underlying transmission protocol (in this case, UDP) for trustworthy host identification. Authorized hosts are kept in memory, and all incoming messages are validated against this list. For performance reasons, an additional check for allowed Notification functions is also performed during this authentication routine. We felt that an additional

access to determine allowed services later on would unnecessarily burden the system, and so all authentication checks were combined. Allowed notification services are saved for use in the Notification routines described later.

2.5.4 Data Authentication

In order to provide the needed consistency in the data gathered by the NERD, checks are run on all incoming messages to ensure the validity of the message and its contents. In the case of messages that are found to be invalid the NERD attempts to reformat the message into a usable form. for example, messages containing control or other non-printable characters are converted. Control characters are converted to their ASCII equivalent (^D, for control-D) so that managers can be aware of attempts to circumvent security measures by sending control strings across the network. Corrupt or missing time-stamps are also repaired, and hostnames are appended to all messages to make clear the source of all messages. When known, the process id generating the message is also added to the message in order to make network debugging and problem resolution easier.

3.0 Notification

The Network Event Recording Device is designed to notify Network Managers of significant network events via electronic mail, digital pager, public address announcements, and video displays. Once validated for particular services by the NERD, a system manager can customize his or her system to provide varying degrees of notification should significant events occur. Again, this customization can, in all but the most elaborate cases, be performed without specialized software. In the event that an important Network Event occurs, and a System Manager has not specified a notification procedure, the Network Operations Center (CIC-5) on-call operator will be notified of the event.

3.1 Notification Options

Currently the NERD provides four levels of notification to authorized hosts: Announcement through DECTalker Synthesized speech, paging through the Los Alamos CIC Alphanumeric Pager Interface (CAPI), electronic mail, and display through the NERD interface (Nerdint) Graphical User Interface tool. Notifications are generated based on the following mapping. The levels are Facility and Priority levels as defined in <sys/syslog.h>. See appendix A for a complete listing:

Facility	Priority	Action
LOG_LOCAL7	LOG_EMERG	Public Address Announcement
LOG_LOCAL7	LOG_WARN	CAPI Page
LOG_LOCAL7	LOG_NOTICE	Electronic Mail
ANY	ANY	Video Display

TABLE 1. Notification Mappings

3.1.1 Host Authentication

As with host authentication for incoming messages, hosts are also authenticated for allowed Notification services. Hosts that attempt to access a Notification Service for which they are not authorized are denied that service, and the attempt is logged by the NERD. Further processing of the message from that host is then terminated. As described earlier, the authorization level of the sending host is determined during the host authentication procedure. This authorization level is then consulted at the time when Notification is attempted. Most Notification Services have a default action (or the option of implementing a default option) in order to avoid lost Notifications.

3.1.2 General Message Format

In order for a host to initiate Notification services, the incoming message must be formatted in such a way that the NERD can determine both the Notification level requested, and the designated recipient of the Notification service. Since Notification Services are mapped by syslog facility and priority levels, this determination is provided by standard syslogd functions. Designated recipients, if any, are determined through routines specifically designed to perform the Notification Service requested.

In order to pass the first hurdle, all messages requesting a NERD Notification Service must be sent with a facility level of LOG_LOCAL7, as defined in the header file syslog.h. Messages processed with other facility levels are not eligible for Notification services.

In order to determine the desired recipient of the Notification, most Notification requests need to include a recipient address (described in the relevant sections below) followed by the designated delimiter character set '<?>'.

3.1.3 Delimiter Character Set

Originally, the delimiter for accessing the Notification recipient was a single character ('?'). However, problems arose with certain 3rd party software packages, most notably the SunNet Manager package. In addition to software incompatibilities, we encountered problems with messages being mistaken for Notification requests. One of the functions requested of the NERD early on was the ability to log all accesses through the Terminal Internet Gateway (TIG) to the ICN. When users would mistakenly type the character '?' in their username, the NERD would attempt a Notification service, often erroneously. In order to alleviate this problem, a 3-character delimiter set was implemented. We have found that, with the 3-character set, no instances of erroneous Notification attempts have occurred.

The delimiter character set '<?>' was chosen as a least-likely combination of characters to occur in a message body, while still being easily recognizable and easily remembered by users wishing to format Notification requests.

3.1.4 Notification Through Public Address Announcement

The NERD is capable of supporting up to 4 DECTalkers, allowing for announcements in different 'zones' of a facility. Currently the NERD is only utilizing two zones. Announcement zones are configurable through a talker configuration file, allowing for changes in zones based on identified usage patterns. Multiple zones can be used simultaneously by ORing zone identifiers together in the Notification Recipient field of the message. By default, if a message from an authorized host, of the corresponding Facility and Priority levels is received, without a specified zone number, the NERD will announce the message to the defined default-zone.

3.1.5 Notification Through CIC Alphanumeric Pager Interface (CAPI)

Another of the key requirements of the NERD is the ability to quickly notify responsible network managers of significant network events that might require attention. The NERD's interface to the CAPI pager system provides an extremely rapid response time to an urgent network situation. In order to make the system as useful as possible, the NERD provides the full text of the network message to the manager specified in the Notification Recipient field of the message. The NERD does no verification of the validity of the Recipient field, instead relying on the CAPI database to provide that service. Invalid pager identifiers are simply dropped by the CAPI system, while sendmail rejects a badly formed message, logging the error through syslogd. In the event that no recipient is specified, the NERD will attempt to send a page to a default address defined in a configuration file. By allowing the NERD to accept a default pager address from a file, the NERD is able to follow an on-call system where different managers are responsible at different times of the day or week.

3.1.6 Notification Through Electronic Mail

Electronic mail notification can notify a single individual, or a list of responsible parties, when lower level network event occur. Mail recipients are specified in the Notification Recipient field of the incoming message. Again, the Recipient field of the message is not validated, relying instead on sendmail to deal with invalid mail addresses. Through a configuration file, a default mail address can be specified, allowing all messages of Facility LOG_LOCAL7 and Priority LOG_NOTICE to generate an email notification. We have found, however, that due to the volume of traffic in this Facility/Priority range, such a default address is neither advantageous nor useful.

3.1.7 Notification Through Video Display

Through the NERDINT (Network Event Recording Device INTerface) messages received by the NERD can be viewed in real-time by system managers. Any message of any Facility or Priority level from an authorized host will be sent to the video support process and will subsequently be available for examination on a NERDINT. In addition, *all* Notification events, as described above, are logged to the video support process. The video display and associated processes will be described more fully in section 5.0

4.0 Implementation

The Network Event Recording Device is based on a modified Berkeley Systems Design (BSD) syslogd process. Though significant changes were made in the syslogd code, the basic functionality of the original syslogd process was maintained in the interest of providing both consistent service and a consistent interface for users. As with the original process, all modifications and additions were done using the C programming language.

A potential flaw in the original code was also uncovered, and significant effort was directed towards correcting what appeares to be a non-standard approach to handling interprocess and network communications by syslogd.

4.1 Alterations to Basic Functionality

As stated, all basic functions of the original syslogd process have been maintained, though some have been altered. In order to help create a more detailed audit trail of incoming events, changes were made in the way in which syslogd records and formats messages, as well as how messages to be forwarded to other hosts are treated. Also implemented was a change to syslogd's handling of repeated, duplicate messages, again to improve the ability of managers to effectively track potential network problems.

4.1.1 Message Format

The original syslogd process, as provided by BSD, simply follows a configuration file (commonly /etc/syslog.conf) as to the manner in which incoming messages are to be processed. This functionality was maintained by our modifications in order to provide consistency of service and communication between differing versions of syslogd.

The modifications consisted of ensuring that all messages are tagged with the sending hostname, and preserving the originating process identifier for all incoming and outgoing messages. Hostnames as extracted from the datagram, are appended to all messages in the form '{ hostname }' to allow network managers to quickly locate messages from a given set of hosts. (See Figure 1.)

In addition to preserving hostnames, a change was made to the manner in which syslogd formats messages before forwarding them to other designated hosts. In the original implementation, syslogd processes forwarding a message sent by a peer syslogd process first stripped the process identifier from the head of the message before forwarding the message string. This situation resulted in a loss of information on messages forwarded through multiple hosts. By changing this behavior and leaving the originating process identifier intact, a more accurate audit trail of system messages has been preserved. Admittedly this modification will only be seen if multiple hosts are running the modified syslogd process, but in adding this functionality to our process, we hope to begin the propagation of the practice throughout hosts utilizing the NERD.

4.1.2 File Format

The NERD maintains a log file similar to that maintained by a standard syslogd process. The name and location of the log file are different for the NERD process, which requires that log information be kept in the file /var/adm/NERD_log. This file can be kept anywhere, as long as appropriate links are installed.

A typical message written to the NERD's log file is shown in figure 1. Notice that, in the example given, the originating process (in this case adgate.lanl.gov's syslog process) is kept, as is the id of the machine which forwarded the message to that host (in this case, colman.lanl.gov). Finally, the hostname which generated the message (adgate.lanl.gov) is appended to the message, as described earlier.

FIGURE 1.

Aug 9 12:50:00 adgate.lanl.gov syslog: colman.lanl.gov : 109586 AUTHENTICATE successful. { adgate.lanl.gov }

4.1.3 Message Processing

After authentication, messages are processed by the NERD based on facility and priority levels. Duplicate messages are suppressed. This suppression is similar to standard syslogd protocol. However, a significant change has been made to the NERD's handling of duplicate or repeat messages. Unlike standard syslogd, where a message stating "Last message repeated x times" is generated, the NERD preserves the repeated message, logging the message "The message <message text> was repeated x times." This functional enhancement was added to the NERD in order to make senders of repeated messages easier to track, isolate and control. Each message received by the NERD is saved in a buffer after being processed so that the message immediately following it can be compared to it. A call to the C Library routine strcmp() is used for the comparison.

4.1.4 Notification Procedures

After an authenticated incoming message has been logged to the appropriate log files, and any message forwarding has been completed (as defined in the configuration file) the NERD's specialized notification procedures are invoked. Messages are first checked for access privileges by comparing the service requested with the allowed services extracted during the message authentication procedure. Allowed services for each host are stored in a 16 bit integer, and are extracted by OR'ing the requested service with the allowed services for the given host.

After this second round of notification authentication, an allowed request is executed by subroutines for each service. For notification services requiring a Notification Recipient Field (NRF) notices are parsed to extract the required address. In the event that no NRF is found, a default address is attempted. If no default address is defined or available, the message is then discarded and control is returned. After extracting the NRF, the message is formatted correctly and sent to the requested service. Access to CAPI is done via a direct

connection to sendmail, as is completion of email notification. In the case of email notification, a useful Subject: header field is also constructed, as is a valid return address. The Subject: field always references the hostname which generated the Notification. The return address generated by the NERD is a reference to a maintained mailing list for the NERD itself. Direct responses to email messages generated by the NERD will be sent to members of the NERD Development and Maintenance Team.

Access to the DECTalkers is controlled through a direct access to the serial ports on the NERD. DECTalkers connected to the serial ports translate the incoming message and broadcast over an attached Public Address system. In order to send data to the DECTalkers in an understandable format, some data manipulation was required in the DECTalker Notifier routine. As part of the Announcement Notification System, messages are scanned for included zone numbers, and are broadcast based on those zone numbers. In the event that no encoded zone number is found, a default zone is used for broadcast. The NERD supports up to 4 distinct zones, though the number of zones possible is limited only by the number of serial ports available on the host machine. At the present time, only two zones are implemented, comprising the two areas of the Central Computing Facility in which the majority of systems are housed.

Email and CAPI access, done via sendmail, are processed similarly, and hence will be treated together here. Due to the modification noted earlier in which process identifiers are no longer stripped from messages, process identifiers must be removed prior to email or CAPI notification to prevent interference with sendmail. In the code shown in Figure 2 we remove the process identifier and search for an included address. If no address is found, we check the default address, returning if that, too is not present.

FIGURE 2. Email and CAPI message handling

```
/* MUST eliminate process identifier from head of message */
    qmark = index(mess, ':');
    if(qmark) mess = qmark + 2;
    qmark = NULL;
    qmark = strstr (mess,DELIM);        /* search for end of addressee */
    if(qmark == NULL ){
        if(DEFAULT_MAIL[0] != NULL){
            sprintf(qmark, "%s", mess); /* move message text */
            sprintf(mess, "%s", DEFAULT_MAIL); /* use default */
        }
        else return;
    }
    else {
        mess[(qmark - mess)] = '\0'; /* mess now the address */
        qmark += 3;
    }
```

Figure 3 shows the process by which we locate a valid email address, or the previously initialized default email address, the hostname (appended to all messages in the format "{ hostname }") is removed for use in the Subject: field, the return address is set, and the mail message is sent. In order to prevent the transmission of restricted or classified data through the NERD, a SECURE compile flag is provided. If set at compile time, only a predefined (using #define) mail message can be sent by the NERD. Through this mechanism, we eliminate the possibility of transmission of restricted or classified data outside the secure partition of the ICN by the NERD.

FIGURE 3. Email and CAPI message sending

```
    mmark = index(qmark, '{'); /* strip hostname from end */
    if(mmark == NULL) mmark = qmark;
    else mmark += 2;
/* subject always references originating hostname */
    sprintf(sub, "Subject: NERD Notice from: %s", mmark);
    sub[strlen(sub) - 1] = '\n' ;/* add subject */
    sprintf(qmark, "%s\n", qmark);

#ifdef SECURE
    /* secure NERDs can only send predefined email messages! */
    sprintf(qmark, "%s", SECURE_EMAIL);
#endif
/*
 * use raw sendmail so that mail comes 'from' NERD. Also, have to repeat
 * the To: to avoid 'Apparently to:' in the header.
 */
```

Finally, once all parameters have been set (Subject: field, To: field, and the message text itself), the message is sent using sendmail. As seen in Figure 4, the full path to sendmail is specified in-line to reduce the possibility of spoofing. We felt that hard-coding certain aspects of the interface to electronic mail would make infiltrating and compromising the NERD system more difficult for possible intruders. Finally, success or failure of the email (or CAPI) routine is logged to the NERD itself, showing the intended recipient, and the included message. The send_driver() routines will be described in detail in the User Interface section.

FIGURE 4. Access to sendmail

```
    sprintf(command, "/usr/lib/sendmail -fnerd %s\n", mess);
    sprintf(message, "To: %s\n", mess);
    mfd = popen(command,"w");
    if (mfd != NULL) {
        fwrite(message, strlen(message), 1, mfd);
        fwrite(sub, strlen(sub), 1, mfd);
        fwrite("\n", 1, 1, mfd);    /* end headers or sendmail breaks */
        fwrite(qmark, strlen(qmark), 1, mfd); /* write the complete message text */
```

```
        pclose(mfd);
        sprintf(drivertext,"Mail message: %s Sent to: %s",qmark, mess);
        send_driver(drivertext, 'm');
    }                   /* endif mfd != NULL */
    else{
        sprintf(drivertext,"Mail message: %s NOT Sent to: %s (sendmail error)",qmark,
                mess);
        send_driver(drivertext, 'm');
    }
```

Following the completion of the above routines, control is returned to the main event handling loop for processing further messages.

5.0 User Interface

The Network Event Recording Device provides a Graphical User Interface for real-time monitoring of current network events as well as reviewing past network logs. The interface, called Nerdint, provides a live connection to the NERD giving the user an up-to-date view of network activity. This interface is customizable by each user, allowing managers to view almost any redefinable subset of network events as they occur. By running this interface, a Network Manager can immediately see color-coded priority messages relating to her system.

In addition to providing real-time monitoring of Network Events, the Nerdint allows Network Managers to notify others of significant network events directly from within the Nerdint. In order to make notification easier, the Nerdint provides an interface to the Los Alamos Trouble Ticket System, the LANL Digital Paging system (CAPI), and standard electronic mail. Users can forward highlighted messages within the NERD to any of these systems without leaving the Nerdint windowing system.

The User Interface requires, in addition to the GUI tool itself, a set of support processes on the NERD loghost to provide host authentication and real-time support of all Nerdints.

5.1 Video Support processes

In order for the NERD to support the Graphical Interface tool Nerdint, we needed to provide a means of supporting these interfaces in real-time so that incoming messages would be broadcast to all attached displays in a timely manner. To accomplish this task we implemented another daemon process to handle all support of Nerdints, including host authentication, transmission of all log information, and continuous updating of all displays. Due to the high volume of traffic, the originally designed video support process was broken into two separate processes, each providing a distinct set of services to the attached Nerdints.

The other process, mdriver (for minidriver) handles all requests from an attached Nerdint and is the intermediary between Nerdints and the driver process.

5.2 The Driver Process

The main support process, called driver, handles all requests for connections, host authentication, and distribution of new messages from syslogd. This process provides an accounting of all connection attempts by Nerdint hosts as well.

5.2.1 Communication With syslogd

In order for the driver process to perform its tasks, it must receive constant input from the modified syslogd process described earlier. The driver process maintains its own log files of all messages, formatted for proper interpretation by the Nerdints it serves. The log file maintained by the driver process is defined as /var/adm/video_log, though this file can be moved if appropriate links are installed. All messages from valid hosts, validated to log to the video display, are forwarded to the driver process through the routing send_driver() as mentioned in section 4.1.3, Figure 4. All communication between these two processes is done using the User Datagram Protocol (UDP) through a port defined in the services file / etc/services. This port number can be changed for different hosts, running different software packages which may reserve ports which are unused on other systems. Though this approach can lead to possible data loss through packets being dropped, it allows for greater fault tolerance between the two processes.

Previously, we used a connected TCP socket over an assigned port (typically 4003) for data transmission between these two processes. However, as the number of Nerdints increased, we saw a substantial increase in the load on the driver process. This increased load translated into a backlog of messages developing in the socket pipeline between syslogd and driver. Eventually, we saw a complete breakdown of communication between the two processes, leading to data loss by the syslogd process itself. Since the syslogd processes log files are considered crucial to network evaluation, this loss of data was unacceptable. By implementing the UDP communication between processes, we expect to see no loss of data by the syslogd process. While loss of data to the driver process is still a possibility, such a loss is more acceptable because the data contained in the message is previously logged, just not available for viewing through the Nerdint GUI.

5.2.2 Host Authentication

As with messages sent to the NERD for logging, connection attempts by Nerdints on remote hosts are all authenticated. A list of allowed hosts is maintained on the NERD (typically /etc/host.allowed). Only hosts listed in this file are permitted connections to the NERD's data files.

The driver process listens on a registered TCP port for connection attempts by Nerdints. The requesting hostname is extracted from the packet header, and is validated through the allowed hosts file. Attempts by unauthorized Nerdints are all logged through syslogd to alert system managers to possible intrusion attempts. Once authenticated, the driver process logs the successful connection through syslogd, and creates an mdriver process to service that Nerdint. All subsequent messages received by driver are forwarded to that mdriver process, and on to the connected Nerdint.

The driver process also maintains a list of all connected Nerdints, the hosts on which they are running, and the display on which they appear. In the event that the driver process quits unexpectedly, this list is consulted on restart, and reconnection with those hosts is attempted.

When a Nerdint terminates, the driver logs the disconnect, and removes the host and display names from its maintained list.

5.2.3 Communication With mdriver Processes

Once a requesting host has been authenticated, the driver process attempts to open a pair of Unix-to-Unix sockets for communication with the mdriver process it is about to spawn. Since the socketpair() system call is implemented only for the Unix domain, the driver and mdriver support processes must run concurrently on the same host. [10] Once opened, this connected socketpair will remain open and in use for the life of the spawned mdriver process. The driver process maintains a list in memory of all open socketpairs, and, upon receipt of a new message from syslogd, writes that message out to all mdrivers over the outbound socket of the pair. In the event that an error occurs during this transmission process, the socketpair is closed and removed from the active list. The driver process then disassociates itself from the now orphaned mdriver process, effectively killing it. By cycling through, and writing to, all mdrivers, each display is, in turn, updated with the latest messages.

5.3 The mdriver Process

The mdriver process is invoked only by the driver process, and only on behalf of an authenticated host requesting a data connection to serve a Nerdint. The mdriver process, running as a subprocess of the driver process, was designed to remove a significant portion of the burden of serving attached Nerdint from the main driver process. The mdriver process maintains active connections to both the main driver process as well as the Nerdint on the remote machine.

5.3.1 mdriver Communication With Nerdint

Once invoked by the main driver process, the mdriver establishes a connected TCP socket with the remote host running the Nerdint. Initially, the mdriver will send the entire display data file (/var/adm/video_log) to the Nerdint. Subsequently, it will update the Nerdint, over the connected socket, when new messages are received from the main driver process.

In addition, the mdriver process listens for requests from its connected Nerdint for retransmission of the data file. Retransmission of the data file is required when the display parameters of the Nerdint are changed to display a larger set of data.

In order to prevent any other process from attempting to start an mdriver, and thus gain access to the NERD's data files, the mdriver process must have a parent process. In the event that the mdriver is orphaned (leaving its parent process the init process, under Unix)

it immediately exits. this prevents Nerdints from becoming detached from continuous updates as well.

In the event that a Nerdint exits, it closes its end of the connected socket to its mdriver process on the NERD Data Server (NDS). The mdriver, when it detects a closed socket, exits, notifying the parent driver process that a Nerdint has disconnected. Through this chain of notifications, the main driver process is able to track attached Nerdints, connections, and disconnects. All Nerdint activity is logged through syslogd in order to maintain an audit trail of access to the NERD's data files and processes.

5.4 Video Log Format

As stated, the NERD's driver process maintains a separate data file for servicing Nerdints. while this file contains all of the information as that in /var/adm/NERD_log (see Figure 1), the format of the file is very different. Each entry contains the time-stamp which the syslog entry has, only the time is entered as a raw time number (number of seconds since the epoch). Additionally, each message contains a one letter code, referencing its display level (see table 2), and a unique identification number. Following the identification number is the complete text of the message logged by syslogd (See Figure 5).

FIGURE 5. Message format for /var/adm/video_log

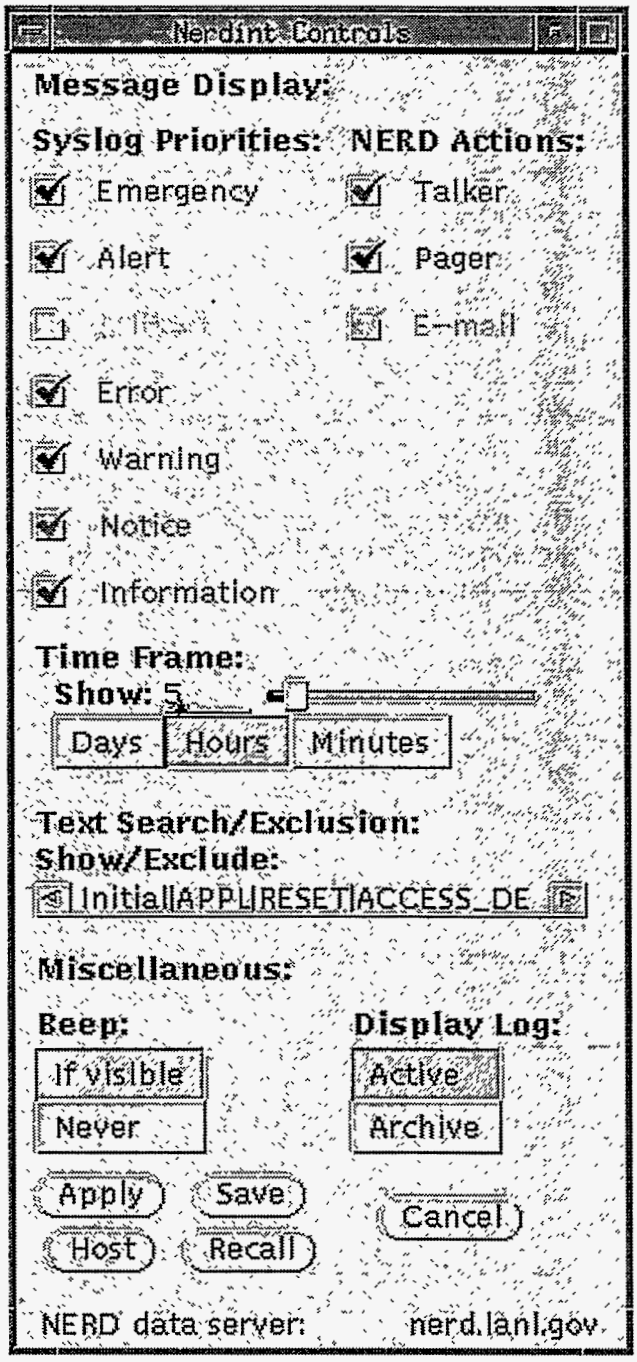
unique_id|level|time_stamp|message_text|
example (from Figure 1): 776461895|l|775391925|syslog: colman.lanl.gov : 109586 AUTHENTICATE successful. { adgate.lanl.gov }|

The character '|' is used as a field separator in the file, enabling rapid parsing of the message by the Nerdint for display and sorting purposes. By including an unique message id number with each message, duplicate messages can be deleted by a receiving Nerdint should they be received due to transmission errors between the NERD and the remote display.

5.5 The NERD Graphical User Interface (Nerdint)

The Nerdint is a window-based application, currently implemented under Sun MIcrosystems OpenWindows which allows the constant monitoring of network events as recorded by the NERD. In addition to viewing incoming messages, the Nerdint also allows users to track NERD Notification Service Requests, and their completion status. Consisting of a number of windows, the Nerdint allows users to filter incoming messages based on priority levels, message content and age of messages. Through a Nerdint Control Panel Window (NCPW) (see figure 6) users can select what messages are displayed, whether or not the display will produce an audible warning when new items are received, and the NERD data server to receive updates from, in an environment where multiple data servers exist. The NCPW also allows users to save current display parameters to a defaults file (.Nerdint-defaults, in the users home directory) for future session, as well as to Recall settings from the defaults file if changes have been made.

FIGURE 6. Nerdint Control Panel Window



5.5.1 Display Priorities

The Nerdint allows a user to show or exclude all messages based on their associated priority level, as determined in syslogd. Each message type is color-coded on the display screen for easy identification of priority levels.

Similarly, NERD actions are differentiated by color, allowing users to easily see the contents and recipient of a NERD-generated email message, CAPI page, or DECTalker announcement. See figure 7 for an example of Syslog priority and NERD Action displays.

FIGURE 7. Nerdint Message Display, showing a message of priority Emergency, with the associated Talker message generated.

5.5.2 Time Frame Display

The Nerdint allows users to set an expiration time for messages, ensuring that only messages within a given time frame will be displayed. The range allowed is from 1 minute to 60 days.

The display list is scanned at regular intervals and expired messages are expunged, keeping the display current at all times.

5.5.3 Text Search and Text Exclusion

In order to deal with the volume of messages logged to the NERD, and to make a manager's use of the Nerdint more productive, a mechanism was needed to allow managers to display only those messages which they needed or wanted to see, filtering out all others. The mechanism employed also allows for rapid message location within an increasingly large amount of network data.

The Nerdint allows users to either show *only* those messages containing a given string, or to *exclude* all messages containing that string. In order to make the interface more familiar to Unix users, a 'piped' approach is also implemented. A user can specify a series of strings to be either shown or excluded (see Figure 8).

FIGURE 8. Example of a Text Search String and a Text Exclude String

NERD.lanl.gov|knsc.lanl.gov|sendmail|popper

!sendmail|popper|duffer.lanl.gov

In the first example in Figure 8, only messages containing one of the given strings will be displayed. A message not containing one or more of the strings will not appear in the Display Window.

In the second example, by beginning the search string with the character '!', only messages *not* containing any one of the given strings will be displayed. Again, the character '|' is used as a field delimiter in both cases.

Other Control Window Settings will not be described here.

5.6 Message Handling and Display

As previously described, messages are forwarded from syslogd to attached Nerdints through the two video support processes. All messages are forwarded to all attached Nerdints, allowing Nerdints themselves to determine whether or not a message is to be displayed. Messages that the Nerdint determines are not to be displayed are simply discarded. Displayed messages are kept in memory for inclusion in email messages and CAPI pages, allowing users to select a message from the Display Screen, and forward it to others.

When display parameters are changed the Nerdint determines whether the new parameters will require the NERD data file to be retransmitted. In the event that more data is requested than is currently displayed, the Nerdint contacts its connected mdriver process and requests the NERD data file, which is then sent.

Though this approach places some increased burden on the network resources if complete data files are continuously being retransmitted, this approach was found to have the least impact on the essential logging functions of both syslogd and driver on the NERD itself.

6.0 Self-Diagnostics and Monitoring

In order to provide highly reliable service, and to minimize the chances of lost data, the NERD has a complete set of self-diagnostic and monitoring processes that continually check the critical portions of the NERD's system. Interruptions in service availability from any of the NERD processes is logged, and an attempt is made to restore service. Should a service restart be unsuccessful, the Self Diagnostic processes begin notification procedures to alert system managers to potential problems on the NERD.

At regular, configurable intervals, the Diagnostic processes monitor critical NERD system parameters. Some of the systems monitored continuously are the disk space availability on partitions crucial to NERD logging, the status of the network interfaces crucial to NERD functioning, and the status of crucial NERD processes such as syslogd and driver.

6.1 Disk Space Availability

Because a lack of available file space would cause significant data loss on the NERD, a small process, called diskckd, is started at system start-up time to monitor disk various disk partition availability. Any number of specific disk partitions can be monitored at set intervals, and notification procedures begun based on set levels of disk availability. As disk space begins to approach a defined level of criticality, Notification Procedures such as email, CAPI pages, and DECTalker announcements are instigated to inform managers of the escalating possibility that disks will reach capacity and data may be lost.

6.2 Network Connections

If the network input queue for the NERD's syslogd process grows too large, the possibilities for lost data or more serious complications increases dramatically. The status of the

NERD's syslogd network connection is crucial to NERD function, and hence is necessary to continuously monitor.

The system process netckd was developed to run regular checks on the network interface to syslogd, or any other network interface, and report problems immediately. As written, the netckd process uses the CAPI pager routine to notify network managers when input queues exceed defined levels.

6.3 System Process Monitoring

One of the stated goals of the NERD was to provide a reliable network service to network managers. Ensuring reliability and availability of the NERD meant making sure that all critical NERD system processes are in continuous operation. Because of defects in hardware or software, systems, even relatively reliable systems, occasionally crash, hang, or become unavailable. To minimize the impact of such unexpected failures on NERD availability, a mechanism was needed to monitor the syslogd, driver, and other NERD system daemons.

To do this, another small system daemon process was written to continuously check on critical NERD functions, restarting them if necessary. At regular intervals, the process, procckd, checks on a defined, configurable set of processes. Should any of them not be found, the process restarts them, and logs the event. Should the process be unable to restart a process, or should the process itself fail, an attempt is made to initiate a Notification Procedure before exiting alert system managers of the fault.

7.0 Current Usage

7.1 LANL Central Computing Facility Users

At the present time, the NERD is being used to log a variety of systems throughout the CCF and to provide specific notification features to each of these systems. More specifically, the NERD is presently providing the announcements of network connectivity problems between the CCF and all Cray Supercomputers in the open (unclassified) partition. According to the managers responsible for those systems, since the system was brought on-line, the NERD has provided automated notification of the CCF operators of problems within the Common File System, allowing for quick resolution. As all Crays have moved from UNICOS 6.1 to UNICOS 7.0, the old system of notification, not described here, has become inaccessible, and the NERD provides the sole source of system status in the new environment.

The Data Storage Group at Los Alamos is currently using the NERD to monitor tape mount requests within the CCF. Their system uses NERD to broadcast a message in the CCF when tape mounts aren't satisfied within 4 minutes of the mount request.

Also within the CCF, the NERD provides audit-trail logging of connections into and out of the Terminal Internet Gateways (TIGs), logging all incoming and outgoing connections,

validations, and failures. This logging provides a security audit trail of all CCF access from dial-ups and remote sites.

7.2 Local Area Network Usage

The Desktop Group, responsible for administering LAN services throughout the Laboratory, is presently using the NERD to continuously monitor the wide variety of Local Area Networks for which it provides network support. System Managers within the group are using the NERD software package to provide central logging and notification for a large and complex system of LANs throughout the Laboratory. In order to allow the Desktop Group to better utilize the NERD's capabilities, we have set up a separate NERD data server for the Desktop Group which logs only Local Area Network events to a separate NERD server.

8.0 Conclusions

In the course of development and implementation of this complex software system, we encountered several obstacles, as well as some interesting possible defects in legacy code. the difficulties we encountered, our solutions and rational for those solutions will be discussed in this section, along with our findings of possible defects in legacy code.

8.1 Implementation Problems

One of the most difficult problems we encountered in the development of the NERD was the ability of the communications protocols to deal with large amounts of traffic, often consisting of large messages. The original design relied on connected TCP sockets for all interprocess communication. While this system worked well in the test situation, its shortcomings were seen immediately upon implementation in a busy production environment. As the number of systems depending on the NERD for event recording and notification increased, the rate of failure increased as well. Since failures resulted in loss of network data, this situation was unacceptable, and a new approach to interprocess communications was pursued.

With the implementation of the User Datagram Protocol for all communications between syslogd and the driver process the degradation in system reliability has disappeared. While an unburdened (logging only local events, or logging for a limited number of hosts) tends to have a network input queue length of 0 bytes, the NERD, when fully loaded under the TCP communication scheme, kept an average input queue length of greater than 2000 bytes. Since the implementation of the UDP Communication Protocol on the NERD, we have seen an average input queue length, when the system is fully loaded, of less than 1000 bytes, indicating that the new communications approach has significantly reduced the risk of data loss

8.2 Code Inconsistencies

In Berkeley Systems Design syslogd, a potential bug was discovered in the way in which that process handles open socket descriptors for incoming messages. I have been able to find no rationale for the way in which BSD code handles this situation. The BSD code, shown in figure 9, uses integers, and the macro FDMASK to set file descriptors, and passes this value to the select() system call, blocking until one or more of the opened file descriptors is ready for I/O. While this approach appears to work, W. Richard Stevens [10] presents a method of I/O multiplexing for reading from multiple socket inputs that is more robust, allowing for reading on multiple socket descriptors while preventing blocking.

FIGURE 9. BSD Syslogd Input/Output Multiplexing

```
/*
 * Copyright (c) 1983, 1988 Regents of the University of California.
 * All rights reserved.
 */
    int funix, inetm, klogm;
    for (;;) {
            int nfds, readfds = FDMASK(funix) | inetm | klogm;
          errno = 0;
          dprintf("readfds = %#x\n", readfds);
          nfds = select(20, (fd_set *) &readfds, (fd_set *) NULL,
             (fd_set *) NULL, (struct timeval *) NULL);
          if (nfds == 0)
                continue;
          if (nfds < 0) {
                if (errno != EINTR)
                      logerror("select");
                continue;
          }
          dprintf("got a message (%d, %#x)\n", nfds, readfds);
    . . .
            if (readfds & inetm) {
                len = sizeof frominet;
                i = recvfrom(finet, line, MAXLINE, 0,
                   (struct sockaddr *) &frominet, &len);
                if (i > 0) {
                      extern char *cvthname();

                      line[i] = '\0';
                      printline(cvthname(&frominet), line);
                } else if (i < 0 && errno != EINTR)
                      logerror("recvfrom inet");
          }
```

While the original code provided by BSD4.3 for multiplexing the I/O for syslogd appears to have no significant, obvious flaws, when the load on the NERD increased dramatically, we saw a rapid rise in the number of unexplained data alignment errors and segmentation violations, traced back to the above segment. We have yet to arrive at a definite reason for why the errors began to occur, and increased in frequency as the load on the process increased.

However, since replacement with the code shown in figure 10, we have seen no alignment or segmentation violations originating from the corresponding section of code. Again, we have not reached a definitive conclusion as to why these errors occurred, or why the change in multiplexing approaches appears to have solved the problem. A complete cessation of alignment and segmentation violations is, we believe, fairly compelling evidence to support our change.

FIGURE 10. I/O Multiplexing As Done by NERD

```
fd_set readfds;
int funix, finet, fklog, max, nfds;
for (;;) {
    FD_ZERO(&readfds); /* zero out then set bits */
    FD_SET(funix, &readfds);
    FD_SET(finet, &readfds);
    FD_SET(fklog, &readfds);
    /*
     * set max descriptor to max of funix, finet, fklog
     */
    max = (funix > finet) ? funix : finet;
    max = (max > fklog) ? max : fklog;


    errno = 0;
    dprintf("readfds = %#x\n", readfds, funix, finet, fklog);
    nfds = select(max + 1, &readfds, (fd_set *) NULL,
      (fd_set *) NULL, (struct timeval *) NULL);
    dprintf("got a message (%d, %#x)\n", nfds, readfds);
...
     if(FD_ISSET(finet, &readfds)) {
        len = sizeof frominet;
        i = recvfrom(finet, line, MAXLINE, 0, &frominet, &len);
        if (i > 0) {
            extern char *cvthname();
            InetHostName = cvthname(&frominet);
            line[i] = '\0';
            printline(InetHostName, line);
        }
```

```
        else if (i < 0 && errno != EINTR)
            logerror("recvfrom inet");
    }   /* end if finet */
```

We have requested comment from the original author of syslogd.c, Mr. Eric Allman, regarding his rationale for the handling of the I/O streams in the original versions of syslogd. While we look forward to his insight and comments, we have not received a response at the time of this writing.

9.0 Miscellaneous

For more information on the NERD, contact David G. Simmons, CIC-5, B-255, davids@lanl.gov, or nerd@nerd.lanl.gov.

Appendix A.

/usr/include/sys/syslog.h

```
/*      @(#)syslog.h 1.5 88/08/19 SMI; from UCB 7.1 6/5/86      */
/*
 * Copyright (c) 1982, 1986 Regents of the University of California.
 * All rights reserved. The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 */
#ifndef _sys_syslog_h
#define _sys_syslog_h
/*
 * Facility codes
 */
#define LOG_KERN        (0<<3)  /* kernel messages */
#define LOG_USER        (1<<3)  /* random user-level messages */
#define LOG_MAIL        (2<<3)  /* mail system */
#define LOG_DAEMON      (3<<3)  /* system daemons */
#define LOG_AUTH        (4<<3)  /* security/authorization messages */
#define LOG_SYSLOG      (5<<3)  /* messages generated internally by syslogd */
#define LOG_LPR         (6<<3)  /* line printer subsystem */
#define LOG_NEWS        (7<<3)  /* netnews subsystem */
#define LOG_UUCP        (8<<3)  /* uucp subsystem */
#define LOG_CRON        (15<<3) /* cron/at subsystem */
        /* other codes through 15 reserved for system use */
#define LOG_LOCAL0      (16<<3) /* reserved for local use */
#define LOG_LOCAL1      (17<<3) /* reserved for local use */
#define LOG_LOCAL2      (18<<3) /* reserved for local use */
#define LOG_LOCAL3      (19<<3) /* reserved for local use */
#define LOG_LOCAL4      (20<<3) /* reserved for local use */
#define LOG_LOCAL5      (21<<3) /* reserved for local use */
#define LOG_LOCAL6      (22<<3) /* reserved for local use */
#define LOG_LOCAL7      (23<<3) /* reserved for local use */

#define LOG_NFACILITIES 24      /* maximum number of facilities */
#define LOG_FACMASK     0x03f8  /* mask to extract facility part */

/*
 * Priorities (these are ordered)
 */
#define LOG_EMERG       0       /* system is unusable */
#define LOG_ALERT       1       /* action must be taken immediately */
#define LOG_CRIT        2       /* critical conditions */
#define LOG_ERR         3       /* error conditions */
#define LOG_WARNING     4       /* warning conditions */
#define LOG_NOTICE      5       /* normal but signification condition */
#define LOG_INFO        6       /* informational */
#define LOG_DEBUG       7       /* debug-level messages */
#define LOG_PRIMASK     0x0007  /* mask to extract priority part (internal) */

/*
 * arguments to setlogmask.
 */
#define LOG_MASK(pri)   (1 << (pri))/* mask for one priority */
#define LOG_UPTO(pri)   ((1 << ((pri)+1)) - 1) /* all priorities through pri */
```

```
/*
 * Option flags for openlog.
 *
 *     LOG_ODELAY no longer does anything; LOG_NDELAY is the
 *     inverse of what it used to be.
 */
#define LOG_PID         0x01    /* log the pid with each message */
#define LOG_CONS        0x02    /* log on the console if errors in sending */
#define LOG_ODELAY      0x04    /* delay open until syslog() is called */
#define LOG_NDELAY      0x08    /* don't delay open */
#define LOG_NOWAIT      0x10    /* if forking to log on console, don't wait() */

#endif /*!_sys_syslog_h*/
```

References.

1) G. Black, **Update on Open Systems Security**, *Computers & Security*, Elsevier Science Publishers Ltd., Vol 11, pg 699-702

2) Final Report of the Design Review Phase of the Independent Verification and Validation of the LANL ICN (Draft), February 15, 1994

3) G. Hardy, **Reviewing Logs Using Knowledge Based Systems**, *Proceedings of The Ninth World Conference on Computer Security, Audit and Control*, Elsevier Science Publishers Ltd., November 1992

4) J. Hochberg et al. **NADIR: An automated System for Detecting Network Intrusion and Misuse**, *Computers & Security*, Elsevier Science Publishers Ltd., Vol 12, No. 3, pg 238

5) D. Lindsay, *An Overview of Leading Security Issues, Information Security An Integrated Approach*, Information Security An Integrated Approach, J.E. Ettinger, Ed., Chapman & Hall, New York.

6) U. Manber, **Chain Reactions in Networks**, *Computer*, IEEE Computer Society Publications, Vol. 23, No. 10

7) K. Morgan and T. Hamer, **Network Security: Aid to Industry Introduction to Communications Audit and Security**, *Proceedings of The Ninth World Conference on Computer Security, Audit and Control*, Elsevier Science Publishers Ltd., November 1992

8) J. Reynolds, J. Postel, **Assigned Numbers**, *Network Working Group Request for Comments: 1060*, March, 1990

9) S. Rowley, *Management: The key to effective distributed systems security*, Information Security An Integrated Approach, J.E. Ettinger, Ed., Chapman & Hall, New York.

10) W. Stevens, UNIX Network Programming, Prentice Hall, New York

11) S. Wilkins, Network Event Recording Device Functional Description (Draft), May 6, 1991