48

AIX

October 1999

update

In this issue

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 550955
From USA: 01144 1635 33823
E-mail: harryl@xephon.com

North American office

Xephon/QNA
1301 West Highway 407, Suite 201-405
Lewisville, TX 75077-2150
USA
Telephone: 940 455 7050

Contributions

If you have anything original to say about AIX, or any interesting experience to recount, why not spend an hour or two putting it on paper? The article need not be very long – two or three paragraphs could be sufficient. Not only will you actively be helping the free exchange of information, which benefits all AIX users, but you will also gain professional recognition for your expertise and that of your colleagues, as well as being paid a publication fee – Xephon pays at the rate of £170 ($250) per 1000 words for original material published in AIX Update.

To find out more about contributing an article, see *Notes for contributors* on Xephon's Web site, where you can download *Notes for contributors* in either text form or as an Adobe Acrobat file.

Editor

Harold Lewis

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, scripts, and other contents of this journal before making any use of it.

Subscriptions and back-issues

A year's subscription to *AIX Update*, comprising twelve monthly issues, costs £180.00 in the UK; $275.00 in the USA and Canada; £186.00 in Europe; £192.00 in Australasia and Japan; and £190.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the November 1995 issue, are available separately to subscribers for £16.00 ($23.00) each including postage.

AIX Update on-line

Code from *AIX Update* is available from Xephon's Web page at *www.xephon.com/aixupdate* (you'll need the user-id shown on your address label to access it).

Printed in England.

Analysing core files remotely

The article *Who is to blame for the core file?* (Werner Klauser, *AIX Update 41*, March 1999) explained the ins and outs of locating and analysing core files. For larger systems, the manual processing of core files is tedious, and so a script that performs this task is a useful tool. An example of when such a script is particularly useful is during the development and debugging of distributed systems, where several workstations have intercommunicating processes.

In this article we provide a script that remotely and automatically locates and analyses cores on several machines.

As the previous article in *AIX Update* stated, a core file is normally generated when the operating system detects certain error conditions and sends a signal to the process. These signals are defined in */usr/include/signal.h*, and include, for example, *SIGSEG* (signal '11'), which typically occurs when an array access is 'out of bounds'. Alternatively, you could trigger a core dump manually using the command **kill -11 <pid>**, since the handling of this signal triggers a core dump by default. The signal may also result in an entry in the error log, which can be viewed using the command **errpt -a | more**. AIX provides an optional 'findcore' package in */usr/samples* that makes use of the error log, and AIX support staff can usually analyse cores when provided with the output file */tmp/ibmsupt* that is generated by running the command **snap -g**. You can locate cores manually using the **find** command, determine the originating program using **lquerypv -n core 6b0 16**, and analyse them using **dbx**. For simplicity, our script is similar to this manual procedure, though the methods used in the **findcore** package would also have advantages.

AUTOMATED REMOTE ANALYSIS

Now suppose we are interested in monitoring several machines for cores. The operations we would like to be able to perform on the cores are: **locate**, **dbx**, and **remove**. We've created a script to accomplish this, where the script can be invoked with the following command:

```
probe_site.csh [-x] [-r] machines
```

where:

- *x*
  Extracts the cores through **dbx**
- *r*
  Erases the core
- *machines*
  Is a list of hostnames of machines to be searched.

The default action is simply to list the locations of core files on the specified machines.

As the script depends on remotely executing commands via the **rsh** command, the remote hosts must enable password-less login through the */.rhosts* file.

When the filesystems are large, locating core files can be time consuming. In our case, cores are located only beneath certain directories, this standard being enforced by programs running the **chdir** subroutine before a core dump. In order to extract the program stack using **dbx**, the location of the executable must be known. In order to save time, our script introduces the variables *coreSearchDirs* and *exeSearchDirs* as a list of directories to be searched. Of course, you could just set both variables to '/' if execution time is not important.

Note that we used the output of the **dbx** core instead of **lquerypv** to determine the executable name, as we found this easier to parse.

Below are listings of the scripts and a sample output when the scripts are invoked by the following command:

```
probe_site.csh -x cam094 > probe_site.out
```

By examining the output with some knowledge of the distributed system, we might conclude that a possible race condition exists between producers and consumers of an initialization state – the processes dumped cores because they tried to use data before another process had initialized it.

Note the use of the continuation character, '➤', in the code that follows to indicate that one line of code maps to several lines of text.

PROBE_SITE.CSH

```
################## probe_site.csh ####################
#!/bin/csh -f
#
# Utility to list and optionally dbx and remove cores
#
# Name: probe_site.csh
#
# Usage:
#   probe_site.csh [-r] [-x] nodes ...
#     -r option causes the script to remove all found core files.
#     -x option to obtain a dbx stack trace from the core.
#     nodes is a list of nodes to check
#
# March 1999 Andre Scheunemann and Jeff Towers
#      Initial Version
#

set thisNode=`hostname`
set removeCores=""
set dbxCores=""
set coreSearchDirs=("/user_fs/dae/err" "/user_fs/user_fs1/log")
set exeSearchDirs=("/user_fs/dae/bin" "/user_fs/clmscnf/exercises"
➤  "/user_fs/simex_plus/conf/utilities" "/user_fs1/ship/excon")

#
# parse command line for dbx and remove options
#
while ( 1 )
  switch ( "$1" )
  case "-r":
    set removeCores="YES"
    breaksw

  case "-x":
    set dbxCores="YES"
    breaksw

  default:
    break
  endsw

  shift
end

set nodes="$*"
if ( "$nodes" == "" ) then
  echo "Syntax: $0 [-x] [-r] nodes
  exit
```

```
endif

foreach node ( $nodes )
  echo "================================== $node
  ➤  =================================="

  set found=`ping -c 1 $node |& grep "1 packets received"`
  if ( "$found" == "") then
    echo "Node $node not responding to ping."
    continue
  endif

  set cores=""
  set i=1

  while ( $i <= $#coreSearchDirs )
    set d="$coreSearchDirs[$i]"

    if ( "$node" == "$thisNode" ) then
      set cores=($cores `find $d -name core -ls`)
    else
      set cores=($cores `rsh $node -l root find $d -name core -ls`)
    endif

    @ i = $i + 1
  end    # while ( $i < $#coreSearchDirs )

  if ( "$cores" != "" ) then
    echo "Core Files:"

    set i=1
    while ( $i <= $#cores )
      @ a = $i + 10
      @ b = $i + 7
      @ c = $i + 8
      @ d = $i + 9
      set f=$cores[$a]
      set prog=""
      set ex=""

      #
      # Try to obtain the executable name from the core file.
      #
      set t=`rsh $node -l root "(cd $f:h; echo quit | dbx core) |& grep
      ➤  'Core file program'"`
      set prog=`echo $t | awk '{print $4}' | sed -e 's/(//;s/)//'`
      set coreDate="$cores[$b] $cores[$c] $cores[$d]"

      echo "==>  $f [$prog] ($coreDate)"
```

```
      if ( "$dbxCores" == "YES" ) then
        #
        # Search for a file that matches the program name under one of
        # the directories that may contain the executables.
        # We then invoke dbx using the found executable and extract the
        # stack trace.
        #
        if ( "$prog" != "" ) then
          set j=1
          while ( $j <= $#exeSearchDirs && "$ex" == "" )
            set ex=`rsh $node -l root find "$exeSearchDirs[$j]" -name
            ➤  $prog -print`
            @ j = $j + 1
          end    # while ( $j < $#exeSearchDirs )

          if ( "$ex" != "" ) then
            #
            # If we find the matching executable we get the stack
            # trace. Keep only the first match found.
            #
            if ( $#ex > 1 ) set ex=$ex[1]

            echo "dbx $ex core"
            rsh $node -l root "echo 'where; quit' | dbx $ex $f |&
            ➤  fgrep -v 'warning: Unable to access'"
            echo "-------------------------------------------------
            ➤  -----------"
            echo ""
          else
            echo "Unable to locate corresponding executable: $prog"
          endif
        else
          echo "Unable to determine executable from core file.":
        endif
      endif    # if ( $dbxCores == YES )

      #
      # Now handle removal of core files if requested.
      #
      if ( "$removeCores" == "YES" ) then
        rsh $node -l root "rm -f $f"
      endif

      @ i = $i + 11
    end    # while ( $i <= $#cores )

    echo ""
  endif    # if ( $cores != empty )
end    # foreach ( $nodes )
################ probe_site.csh ###################
```

PROBE_SITE.OUT

```
################ probe_site.out ##################
============================== cam094 ==============================
Core Files:
==>  /user_fs/user_fs1/log/coreAP2/core [ap2c0_excon.exe.287]
➤  (Apr 13 15:17)
dbx /user_fs/clmscnf/exercises/126/ap2c0_excon.exe.287 core
Type 'help' for help.
warning: The core file is truncated.  You may need to increase the
ulimit for file and coredump, or free some space on the filesystem.
reading symbolic information ...
[using memory image in /user_fs/user_fs1/log/coreAP2/core]

IOT/Abort trap in raise at 0xd007fa14
0xd007fa14 (raise+0x20) 80410014          l   r2,0x14(r1)
raise(??) at 0xd007fa14
abort() at 0xd001b8a8
abort_logger_(??, ??, ??, ??), line 1179 in "cel_routines.c"
Init_Role(), line 198 in "init_role.c"
AInit(), line 193 in "ainit12_deb.c"
aevent_usleep(0xa), line 144 in "aevent_usleep.c"
aevent_usleep_front(0xa), line 36 in "aevent_front.c"
adisp_usleep(0xa), line 421 in "displib.c"
ainitdisp(), line 327 in "ainitdisp.c"
main(argc = 4, argv = 0x2ff216dc, 0x2ff216f0, 0x2, 0x2ff22ff8, 0x0,
➤  0x42, 0x80000000), line 97 in "adisp.c"
---------------------------------------------------------------

==>  /user_fs/user_fs1/log/coreAP1/core [ap1c0_geo_pws.exe.225]
➤  (Apr 13 15:17)
dbx /user_fs/clmscnf/exercises/126/ap1c0_geo_pws.exe.225 core
Type 'help' for help.
warning: The core file is truncated.  You may need to increase the
ulimitfor file and coredump, or free some space on the filesystem.
reading symbolic information ...
[using memory image in /user_fs/user_fs1/log/coreAP1/core]

IOT/Abort trap in raise at 0xd007fa14
0xd007fa14 (raise+0x20) 80410014          l   r2,0x14(r1)
raise(??) at 0xd007fa14
abort() at 0xd001b8a8
abort_logger_(??, ??, ??, ??), line 1179 in "cel_routines.c"
Init_Role(), line 198 in "init_role.c"
AInit(), line 185 in "ainit7.c"
aevent_usleep(0xa), line 144 in "aevent_usleep.c"
aevent_usleep_front(0xa), line 36 in "aevent_front.c"
adisp_usleep(0xa), line 421 in "displib.c"
ainitdisp(), line 327 in "ainitdisp.c"
main(argc = 4, argv = 0x2ff21888, 0x2ff2189c, 0x2, 0x2ff22ff8, 0x0,
```

```
➤  0x42, 0x80000000), line 97 in "adisp.c"
------------------------------------------------------------

################ probe_site.out ###################
```

Andre Scheunemann and Jeff Towers
Software Analysts
CAE Electronics Ltd (Canada)

Message of the day

INTRODUCTION

This article presents two scripts for managing the 'message-of-the-day': **mmotd.sh**, which manages the message-of-the-day, and **rmotd.sh**, which replaces the message-of-the-day. In AIX, the contents of the */etc/motd* file are automatically displayed at logon. The scripts allows this message file to be maintained, allowing users to manipulate its entries. The script **mmotd.sh** provides the following options:

- Add a message-of-the-day
- Modify the message-of-the-day
- Delete the message-of-the-day
- View a specific message-of-the-day
- View the message file
- Add a default message
- Modify the default message
- Initialize the message file
- Initialize **rmotd**'s log file
- View **rmotd**'s log file
- Replace the current *motd* file.

rmotd.sh is a shell script that runs as a **cron** job to replace the contents of */etc/motd* file just after midnight every night. It replaces the contents of the file with the day's message from the message file. If no message for the day is found in the message file, the default message is used, so this should be maintained in the message file at all times. The script also maintains a log file.

The **mmotd.sh** script allows users to add messages only for future dates and/or the current day. As well as allowing these messages to be displayed using the **rmotd.sh** script, **mmotd.sh** has an option for replacing the current message-of-the-day file.

MAKING IT WORK

1 Set appropriate values for the following variables in the **mmotd.sh** script:

   – $MESSAGE_DIR

   – $MESSAGE_FILE

   – $MOTD_PATH (where the scripts reside)

   – $RMOTD_LOG (the log file kept by **rmotd.sh**).

2 Set appropriate values for the following variables in the **rmotd.sh** script:

   – $MESSAGE_DIR

   – $MESSAGE_FILE

   – $RMOTD_LOG (the log file kept by **rmotd.sh**).

3 Run **mmotd.sh** from *root* account to initialize a new message file and to add new messages.

4 Make an entry in *corncob* file under *root* account as follows:

```
#
# replace message-of-the-day file at 12.01 am
#
1  0  *  *  *  /home/ecatmgr/sh/rmotd.sh > /dev/null 2>&1
```

Note the continuation character, '➤', in the code that follows to indicate that one line of code corresponds to several lines of print.

MMOTD.SH (MANAGE MESSAGE-OF-THE-DAY)

```
#! /bin/ksh
###############################################################################
#                                                                             #
# mmotd.sh - manage the 'message-of-the-day'                                  #
#                                                                             #
# The script manages all aspects of message-of-the-day.                       #
#                                                                             #
# Notes   1   The script contains following functions:                        #
#                                                                             #
#             - InitialiseVariables                                           #
#             - HandleInterrupt                                               #
#             - MoveCursor                                                    #
#             - DisplayMessage                                                #
#             - DisplayMenu                                                   #
#             - ProcessOption                                                 #
#             - GetMessage                                                    #
#             - AddMessage                                                    #
#             - ModifyMessage                                                 #
#             - DeleteMessage                                                 #
#             - ViewSpecificMessage                                           #
#             - ViewMessageFile                                               #
#             - AddDefaultMessage                                             #
#             - ModifyDefaultMessage                                          #
#             - InitializeMessageFile                                         #
#             - InitialiseRMOTDLogFile                                        #
#             - ViewRMOTDLogFile                                              #
#             - IsNumeric                                                     #
#             - LeapYear                                                      #
#             - IsValidDayOfMonth                                             #
#             - IsValidMonth                                                  #
#             - IsValidYear                                                   #
#             - ValidateInput                                                 #
#             - GetMessageDate                                                #
#             - RootUser                                                      #
#             - ProcessExit                                                   #
#             - main                                                          #
#                                                                             #
# History                                                                     #
# Date      Author    Description                                             #
# ------------------------------------------------------------------------- #
# 09/02/99  A Zaman   Initial build                                           #
#                                                                             #
###############################################################################

###############################################################################
#                                                                             #
# InitialiseVariables                                                         #
#                                                                             #
```

```
# This function initializes all required variables.                #
#                                                                  #
####################################################################
InitialiseVariables()
{

# define locations
MESSAGE_DIR="/home/ecatmgr/motd"
MESSAGE_FILE="${MESSAGE_DIR}/motd.dat"
RMOTD_LOG="${MESSAGE_DIR}/rmotd.log" # log file written by rmotd.sh
TEMP_FILE="/tmp/mmotd_$$.tmp"
TEMP_FILE_1="/tmp/mmotd_$$_1.tmp"
MOTD_PATH="/home/ecatmgr/sh/temp" ; export MOTD_PATH

# define date-related variables
DAY=           # day of the message
MON=           # month of the message
YEAR=          # year of the message

# define message-related variables
MSG=           # message-of-the-day

# escape sequences
ESC="\0033["
RVON= [7m                  # reverse video on
RVOFF= [27m                # reverse video off
BOLDON= [1m                # bold on
BOLDOFF= [22m              # bold off
BON= [5m                   # blinking on
BOFF= [25m                 # blinking off

# define menu title
MMOTD="${RVON}Manage message-of-the-day${RVOFF}"

# define exit codes
SEC=0
FEC=1

TRUE=0
FALSE=1

SLEEP_DURATION=4           # number of seconds for sleep command
ERROR="${RVON}${BON}mmotd.sh:ERROR:${BOFF}"
INFO="${RVON}mmotd.sh:INFO: "

# messages
INTERRUPT="Program interrupted! Quitting...${RVOFF}"
INVALID_OPTION="Invalid entry ${RVOFF}"
MOTD_EXISTS="A message exist for day \${DOTM}${RVOFF}"
DEFAULT_EXISTS="A default message already exists${RVOFF}"
```

```
MOTD_ADDED="Successfully added message for day \${DOTM}${RVOFF}"
MOTD_EMPTY="Message not modified\; empty message string${RVOFF}"
MOTD_NOT_ADDED="Failed to add message for day \${DOTM}${RVOFF}"
DMOTD_ADDED="Successfully added default message-of-the-day${RVOFF}"
DMOTD_NOT_ADDED="Failed to add default message-of-the-day${RVOFF}"
MSG_NOT_FOUND="No message exists for day \${DOTM}${RVOFF}"
DMSG_NOT_FOUND="No default message-of-the-day exist${RVOFF}"
MSG_FILE_EMPTY="Message file, \${MESSAGE_FILE}, is empty${RVOFF}"
MOTD_MODIFIED="Successfully modified message for day \${DOTM}${RVOFF}"
MOTD_NOT_MODIFIED="Failed to modify message for day \${DOTM}${RVOFF}"
DMOTD_MODIFIED="Successfully modified default message-of-the-day
➤  ${RVOFF}"
DDMOTD_MODIFIED="Failed to modify default message-of-the-day${RVOFF}"
EDIT_MSG="Modify the message and save the file${RVOFF}"
INVALID_ENTRY="Invalid entry ${RVOFF}"
INVALID_MONTH="\${MON}, is an invalid month ${RVOFF}"
INVALID_MONTH_YEAR="\${MON}, is an invalid month for year
➤  \$YEAR${RVOFF}"
INVALID_DAY="\${DAY}, is an invalid day for month, \${MON} ${RVOFF}"
INVALID_YEAR="\${YEAR}, is an invalid year ${RVOFF}"
NOT_NUMERIC="Value must be numeric${RVOFF}"
FILE_NOT_INITIALIZED="Failed to initialize the file
➤  \${FILE_NAME}${RVOFF}"
FILE_INITIALIZED="Successfully initialized the file
➤  \${FILE_NAME}${RVOFF}"
NO_MSG_FILE="Message file, \${MESSAGE_FILE}, does not exist${RVOFF}"
NO_RMOTD_LOG="Log file ,\${RMOTD_LOG} does not exist${RVOFF}"
OSERROR="\${OSEM}${RVOFF}"
ROOT_USER="The script must be executed from the root account${RVOFF}"

# define signals
SIGNEXIT=0  ;  export SIGNEXIT   # normal exit
SIGHUP=1    ;  export SIGHUP     # when session disconnected
SIGINT=2    ;  export SIGINT     # ctrl-c
SIGTERM=15  ;  export SIGTERM    # kill command
}

##############################################################################
#                                                                            #
# HandleInterrupt                                                            #
#                                                                            #
# This function calls ProcessExit.                                          #
#                                                                            #
##############################################################################
HandleInterrupt ()
{
DisplayMessage I "${INTERRUPT}"
ProcessExit  $FEC
}
```

```
################################################################################
#                                                                              #
# MoveCursor                                                                   #
#                                                                              #
# This function moves the cursor to location (Y, X).                           #
#                                                                              #
# Input   :   X and Y coordinates                                              #
#                                                                              #
# Notes   1   This must be run in ksh for print to work. Also,                 #
#             print must be used to move the cursor as echo                    #
#             does not seem to work.                                           #
#                                                                              #
################################################################################
MoveCursor ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

YCOR=$1
XCOR=$2

echo "${ESC}${YCOR};${XCOR}H"
}

################################################################################
#                                                                              #
# DisplayMessage                                                               #
#                                                                              #
# This function displays a message.                                            #
#                                                                              #
# Input   1   Message type (E = Error, I = Information)                        #
#         2   Error code, as defined in DefineMessages ( ).                    #
#                                                                              #
################################################################################
DisplayMessage ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

MESSAGE_TYPE=$1
MESSAGE_TEXT=`eval echo $2`

MoveCursor 24 1
if [ "${MESSAGE_TYPE}" = "E" ]
then
    echo "`eval echo ${ERROR}`${MESSAGE_TEXT}\c"
else
    echo "`eval echo ${INFO}`${MESSAGE_TEXT}\c"
fi
sleep ${SLEEP_DURATION}
return ${TRUE}
}
```

```
##############################################################################
#                                                                            #
# DisplayMenu                                                                #
#                                                                            #
# This function displays the menu.                                          #
#                                                                            #
##############################################################################
DisplayMenu ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

clear
echo   "       ########################################"
echo   "       #                                      #"
echo   "       #      $MMOTD                          #"
echo   "       #                                      #"
echo   "       #   5  Add a message                   #"
echo   "       #  10  Modify a message                #"
echo   "       #  15  Delete a message                #"
echo   "       #  20  View a specific message         #"
echo   "       #  25  View a message file             #"
echo   "       #  30  Add a default message           #"
echo   "       #  35  Modify the default message      #"
echo   "       #  40  Initialize the message file     #"
echo   "       #  45  Initialize the RMOTD log file   #"
echo   "       #  50  View the RMOTD log file         #"
echo   "       #  55  Replace the current MOTD file   #"
echo   "       #  99  Exit                            #"
echo   "       #                                      #"
echo   "       ########################################"
echo   "       Enter your option --->\c"

read OPTION
}

##############################################################################
#                                                                            #
# ProcessOption                                                              #
#                                                                            #
# This function processes a menu option.                                     #
#                                                                            #
##############################################################################
ProcessOption ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

case $OPTION in
     5) AddMessage ;;
    10) ModifyMessage ;;
    15) DeleteMessage ;;
```

```
    20) ViewSpecificMessage ;;
    25) ViewMessageFile ;;
    30) AddDefaultMessage ;;
    35) ModifyDefaultMessage ;;
    40) InitializeMessageFile ;;
    45) InitialiseRMOTDLogFile ;;
    50) ViewRMOTDLogFile ;;
    55) ReplaceCurrentMotdFile ;;
    99) clear; ProcessExit $SEC ;;
    * ) DisplayMessage E "${INVALID_OPTION}"
esac
}

#############################################################################
#                                                                           #
# GetMessage                                                                #
#                                                                           #
# This function gets the required message.                                  #
#                                                                           #
#############################################################################
GetMessage ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

while true
do
    clear
    echo "Enter the message:\c"
    read MSG
    case "${MSG}" in
        "" ) DisplayMessage E "${INVALID_ENTRY}" ;;
         * ) break ;;
    esac
done
}

#############################################################################
#                                                                           #
# AddMessage                                                                #
#                                                                           #
# This function adds a message to the message file.                         #
#                                                                           #
# Returns :   $TRUE or $FALSE                                               #
#                                                                           #
# Notes   1   The function checks for an existing message before            #
#             writing the current message.                                  #
#                                                                           #
#############################################################################
AddMessage ( )
{
```

```
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

# check for the existence of the file
if [ ! -f ${MESSAGE_FILE} ]
then
    DisplayMessage E "${NO_MSG_FILE}"
    return $FALSE
fi

# get the required message's date
GetMessageDate

#  get the message
GetMessage

# check the existing entry
DOTM="${DAY}${MON}${YEAR}"
if grep "${DOTM}" ${MESSAGE_FILE} > /dev/null 2>&1
then
     # a message exist for this day
     DisplayMessage E "${MOTD_EXISTS}"
     return $FALSE
fi

# write away the message
( echo "${DOTM}:${MSG}" 1>> ${MESSAGE_FILE} ) 2> ${TEMP_FILE}
if [ $? -ne 0 ]
then
    DisplayMessage E "${MOTD_NOT_ADDED}"
    OSEM=`cat ${TEMP_FILE}`
    DisplayMessage E "${OSERROR}"
    return $FALSE
else
    DisplayMessage I "${MOTD_ADDED}"
    return $TRUE
fi
}

##############################################################################
#                                                                            #
# ModifyMessage                                                              #
#                                                                            #
# This function modifies a specific message.                                 #
#                                                                            #
# Returns :   $TRUE or $FALSE                                                #
#                                                                            #
##############################################################################
ModifyMessage ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP
```

```
# check for the existence of the file
if [ ! -f ${MESSAGE_FILE} ]
then
    # no message file found
    DisplayMessage E "${NO_MSG_FILE}"
    return $FALSE
fi

# get the required message's date
GetMessageDate
DOTM="${DAY}${MON}${YEAR}"

# get the required message
if ! grep "${DOTM}" ${MESSAGE_FILE} > ${TEMP_FILE} 2>&1
then
    DisplayMessage E "${MSG_NOT_FOUND}"
    return $FALSE
fi

# pick the message from temporary file
LINE="`cat ${TEMP_FILE}`"
MSG=`echo "${LINE}" | cut -d':' -f2`
echo "${MSG}" > ${TEMP_FILE}
DisplayMessage I "${EDIT_MSG}"
vi ${TEMP_FILE}
MSG=`cat ${TEMP_FILE}`
if [ "${MSG}" = "" ]
then
     DisplayMessage E "${MOTD_EMPTY}"
     return $FALSE
fi

#  write the modified message
sed /^${DOTM}/d ${MESSAGE_FILE} > ${TEMP_FILE}
cp ${TEMP_FILE} ${MESSAGE_FILE}
( echo "${DOTM}:${MSG}" 1>> ${MESSAGE_FILE}) 2> ${TEMP_FILE}
if [ $? -ne 0 ]
then
    DisplayMessage E "${MOTD_NOT_MODIFIED}"
    OSEM=`cat ${TEMP_FILE}`
    DisplayMessage E "${OSERROR}"
    return $FALSE
else
    DisplayMessage I "${MOTD_MODIFIED}"
    return $TRUE
fi
}

##############################################################################
#                                                                            #
```

```
# DeleteMessage                                                          #
#                                                                        #
# This function deletes a specific message from message file.            #
#                                                                        #
# Returns :   $TRUE or $FALSE                                            #
#                                                                        #
##########################################################################
DeleteMessage ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

# check for the existence of the file
if [ ! -f ${MESSAGE_FILE} ]
then
    # no message file found
    DisplayMessage E "${NO_MSG_FILE}"
    return $FALSE
fi

# get the required message's date
GetMessageDate
DOTM="${DAY}${MON}${YEAR}"

# get the required message
if ! grep "${DOTM}" ${MESSAGE_FILE} > ${TEMP_FILE} 2>&1
then
    DisplayMessage E "${MSG_NOT_FOUND}"
    return $FALSE
fi

# delete the message
sed /^${DOTM}/d ${MESSAGE_FILE} > ${TEMP_FILE}
cp ${TEMP_FILE} ${MESSAGE_FILE}
return $TRUE
}

##########################################################################
#                                                                        #
# ViewSpecificMessage                                                    #
#                                                                        #
# This function is used to view a specific message from the              #
# message file.                                                          #
#                                                                        #
# Returns :   $TRUE or $FALSE                                            #
#                                                                        #
##########################################################################
ViewSpecificMessage ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP
```

```
# check for the existence of the file
if [ ! -f  ${MESSAGE_FILE} ]
then
    # no message file found
    DisplayMessage E "${NO_MSG_FILE}"
    return $FALSE
fi

# get the required message's date
GetMessageDate
DOTM="${DAY}${MON}${YEAR}"

# get the required message
if ! grep "${DOTM}" ${MESSAGE_FILE} > ${TEMP_FILE} 2>&1
then
    DisplayMessage E "${MSG_NOT_FOUND}"
    return $FALSE
fi

# view the message
view ${TEMP_FILE}

return $TRUE
}

##############################################################################
#                                                                            #
# ViewMessageFile                                                            #
#                                                                            #
# This function views all messages from message file.                        #
#                                                                            #
# Returns :   $TRUE or $FALSE                                                #
#                                                                            #
# Notes   1   Messages are sorted in ascending order by date.                #
#                                                                            #
##############################################################################
ViewMessageFile ( )
{

trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

# check for the existence of the file
if [  ! -f  ${MESSAGE_FILE}  ]
then
    # no message file found
    DisplayMessage E "${NO_MSG_FILE}"
    return $FALSE
fi

# check the message file for existing messages
```

```
if [ ! -s "${MESSAGE_FILE}" ]
then
    DisplayMessage E "${MSG_FILE_EMPTY}"
    return $FALSE
fi

# sort the message file, first removing the header
sed /^' '/d ${MESSAGE_FILE} > ${TEMP_FILE}

# add header to file for sorted messages
echo "           Message-of-the-day  File   " >  ${TEMP_FILE_1}
echo "           ========================   " >> ${TEMP_FILE_1}

# sort message file without the header
cat ${TEMP_FILE} |sort -t:  -k 1.5,1.8  -k 1.3,1.4  -k 1.1,1.2 >>
➤  ${TEMP_FILE_1}

# view the message file
c
view ${TEMP_FILE}
return $TRUE
}

##############################################################################
#                                                                            #
# AddDefaultMessage                                                          #
#                                                                            #
# This function adds the default message to the message file.                #
#                                                                            #
# Returns :   $TRUE or $FALSE                                               #
#                                                                            #
# Notes   1   The function checks for existing message before                #
#             writing the current one.                                       #
#                                                                            #
##############################################################################
AddDefaultMessage ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

# check for the existence of the file
if [ ! -f ${MESSAGE_FILE} ]
then
    DisplayMessage E "${NO_MSG_FILE}"
    return $FALSE
fi

# check the existing entry
if grep "DEFAULT:" ${MESSAGE_FILE} > /dev/null 2>&1
then
     # a default message already exist for this day
```

```
     DisplayMessage E "${DEFAULT_EXISTS}"
     return $FALSE
fi

# get the required message
GetMessage

# write the message
( echo "DEFAULT:${MSG}" 1>> ${MESSAGE_FILE} ) 2> ${TEMP_FILE}
if  [ $?  -ne 0 ]
then
    DisplayMessage E "${DMOTD_NOT_ADDED}"
    OSEM=`cat ${TEMP_FILE}`
    DisplayMessage E "${OSERROR}"
    return $FALSE
else
    DisplayMessage I "${DMOTD_ADDED}"
    return $TRUE
fi
}

##############################################################################
#                                                                            #
# ModifyDefaultMessage                                                       #
#                                                                            #
# This function modifes the default message in the message file.    #
#                                                                            #
# Returns :   $TRUE or $FALSE                                               #
#                                                                            #
##############################################################################
ModifyDefaultMessage ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

# check for file existence
if [ ! -f ${MESSAGE_FILE} ]
then
    # no message file found
    DisplayMessage E "${NO_MSG_FILE}"
    return $FALSE
fi

# get the required message
if ! grep "DEFAULT:" ${MESSAGE_FILE} > ${TEMP_FILE} 2>&1
then
    DisplayMessage E "${DMSG_NOT_FOUND}"
    return $FALSE
fi
LINE="`cat ${TEMP_FILE}`"
echo "${LINE}" ; read dummy
```

```
MSG=`echo "${LINE}" | cut -d':' -f2`
echo "${MSG}" > ${TEMP_FILE}
DisplayMessage I "${EDIT_MSG}"
MSG=`cat ${TEMP_FILE}`
if [ "${MSG}" = "" ]
then
    DisplayMessage E "${MOTD_EMPTY}"
    return $FALSE
fi

# write the modified message
sed /^DEFAULT/d ${MESSAGE_FILE} > ${TEMP_FILE}
( cp ${TEMP_FILE} ${MESSAGE_FILE} ) 2> ${TEMP_FILE_1}
if [ $? -ne 0 ]
then
    DisplayMessage E "${DMOTD_NOT_MODIFIED}"
    DisplayMessage E "${OSERROR}"
    return $FALSE
    OSEM=`cat ${TEMP_FILE_1}`
fi
( echo "DEFAULT:${MSG}" 1>> ${MESSAGE_FILE}) 2> ${TEMP_FILE}
if [ $? -ne 0 ]
then
    DisplayMessage E "${DMOTD_NOT_MODIFIED}"
    DisplayMessage E "${OSERROR}"
    return $FALSE
else
    DisplayMessage I "${DMOTD_MODIFIED}"
    return $TRUE
fi
}

##############################################################################
#                                                                            #
# InitializeMessageFile                                                      #
#                                                                            #
# This function initializes the message file.                                #
#                                                                            #
# Notes   1   A default message-of-the-day is written as part                #
#             of this process.                                               #
#                                                                            #
##############################################################################
InitializeMessageFile ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

FILE_NAME="${MESSAGE_FILE}"

# check for the existence of the file
if [ ! -f ${MESSAGE_FILE} ]
```

```
then
    # no message file exists, so initialize a new file
    (> ${MESSAGE_FILE} ) > ${TEMP_FILE} 2>&1
    if [ $? -ne 0 ]
    then
        DisplayMessage E "${FILE_NOT_INITIALIZED}"
        OSEM=`cat ${TEMP_FILE}`
        DisplayMessage E "${OSERROR}"
        return $FALSE
    else
        DisplayMessage I "${FILE_INITIALIZED}"

        # write header
        echo "                  Message-of-the-day  File   " >
        ➤  ${MESSAGE_FILE}
        echo "                  ========================   " >>
        ➤  ${MESSAGE_FILE}

        # write a default message
        echo "DEFAULT:Welcome to AIX" >> ${MESSAGE_FILE}
        return $TRUE
    fi
fi

# get confirmation for initializing the existing file
while true
do
    clear
    echo "Message file, ${MESSAGE_FILE} exists"
    echo "Confirm initialization (Y/N):\c"
    read REPLY
    case $REPLY in
        Y) break ;;
        N) return $FALSE ;;
        *) DisplayMessage E "${INVALID_OPTION}" ;;
    esac
done

# initialize existing message file
(> ${MESSAGE_FILE}) > ${TEMP_FILE} 2>&1
if [ $? -ne 0 ]
then
    DisplayMessage E "${FILE_NOT_INITIALIZED}"
    OSEM=`cat ${TEMP_FILE}`
    DisplayMessage E "${OSERROR}"
    return $FALSE
else
    DisplayMessage I "${FILE_INITIALIZED}"
    #
    # write header
```

```
    #
    echo "                  Message-of-the-day  File   " >
    ➤  ${MESSAGE_FILE}
    echo "                  ========================   " >>
    ➤  ${MESSAGE_FILE}

    # write a default message
    echo "DEFAULT:Welcome to AIX" >> ${MESSAGE_FILE}
    return $TRUE
fi
}

##############################################################################
#                                                                            #
# InitialiseRMOTDLogFile                                                     #
#                                                                            #
# This function initializes the log file.                                    #
#                                                                            #
##############################################################################
InitialiseRMOTDLogFile ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP
FILE_NAME="${RMOTD_LOG}"

# check for the existence of the file
if [ ! -f ${RMOTD_LOG} ]
then
    # no log file exists, so initialize a new file
    (> ${RMOTD_LOG}) > ${TEMP_FILE} 2>&1
    if [ $? -ne 0 ]
    then
        DisplayMessage E "${FILE_NOT_INITIALIZED}"
        OSEM=`cat ${TEMP_FILE}`
        DisplayMessage E "${OSERROR}"
        return $FALSE
    else
        DisplayMessage I "${FILE_INITIALIZED}"

        # write header
        echo "        Log File for Message-of-the-day Replacement" >
        ➤  ${RMOTD_LOG}
        echo "        ==========================================" >>
        ➤  ${RMOTD_LOG}
        return $TRUE
    fi
fi

# get confirmation for initializing existing file
while true
do
```

```
    clear
    echo "Log file, ${RMOTD_LOG} exists"
    echo "Confirm initialization (Y/N):\c"
    read REPLY
    case $REPLY  in
        Y) break ;;
        N) return $FALSE ;;
        *) DisplayMessage E "${INVALID_OPTION}" ;;
    esac
done

# initialize the log file
(> ${RMOTD_LOG}) > ${TEMP_FILE} 2>&1
if [ $? -ne 0 ]
then
    DisplayMessage E "${FILE_NOT_INITIALIZED}"
    OSEM=`cat ${TEMP_FILE}`
    DisplayMessage E "${OSERROR}"
    return $FALSE
else
    DisplayMessage I "${FILE_INITIALIZED}"

    # write header
    echo "        Log file for Message-of-the-day replacement" >
    ➤  ${RMOTD_LOG}
    echo "        ==========================================" >>
    ➤  ${RMOTD_LOG}
    return $TRUE
fi
}

##############################################################################
#                                                                            #
# ViewRMOTDLogFile                                                           #
#                                                                            #
# This function views the log file, written by rmotd.sh script.             #
#                                                                            #
##############################################################################
ViewRMOTDLogFile ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

# check for the existence of the file
if [ ! -f ${RMOTD_LOG} ]
then
    # no log file exists
    DisplayMessage E "${NO_RMOTD_LOG}"
    return $FALSE
fi
```

```
# view the file
cp ${RMOTD_LOG} ${TEMP_FILE}
view ${TEMP_FILE}
}

##############################################################################
#                                                                            #
# ReplaceCurrentMotdFile                                                     #
#                                                                            #
# This function replaces current motd file with new one by                   #
# invoking the script rmotd.sh.                                              #
#                                                                            #
##############################################################################
ReplaceCurrentMotdFile ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP
${MOTD_PATH}/rmotd.sh
return $TRUE
}

##############################################################################
#                                                                            #
# IsNumeric                                                                  #
#                                                                            #
# This function checks a given value for to ensure it is numeric.            #
#                                                                            #
# Returns :   $TRUE or $FALSE                                               #
#                                                                            #
##############################################################################
IsNumeric ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP
PARAM="$1"
ARGLEN=`expr "$PARAM" : '.*'`
INDEX=1
while [ $INDEX -le $ARGLEN ]
do
    CHAR=`echo "$PARAM" | cut -c$INDEX-$INDEX`
    if [ "${CHAR}" != "0" -a "${CHAR}" != "1" -a "${CHAR}" != "2" -a \
         "${CHAR}" != "3" -a "${CHAR}" != "4" -a "${CHAR}" != "5" -a \
         "${CHAR}" != "6" -a "${CHAR}" != "7" -a "${CHAR}" != "8" -a \
         "${CHAR}" != "9" ]
    then
        DisplayMessage E "${NOT_NUMERIC}"
        return  $FALSE
    fi
    INDEX=`expr $INDEX + 1`
done
}
```

```
##############################################################################
#                                                                            #
# LeapYear                                                                   #
#                                                                            #
# This function checks whether a given year is a leap year.                  #
#                                                                            #
# Input   :   $YEAR                                                          #
#                                                                            #
# Returns :   $TRUE or $FALSE                                                #
#                                                                            #
##############################################################################
LeapYear ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

RESULT=`bc <<!
scale=2
$YEAR/4
!`
if [ "`echo $RESULT | cut -d'.' -f2`" = "00" ]
then
    # year is a leap year
    return $TRUE
else
    # year is not a leap year
    return $FALSE
fi
}

##############################################################################
#                                                                            #
# IsValidDayOfMonth                                                          #
#                                                                            #
# This function validates a day for a given month.                           #
#                                                                            #
# Input   :   a specific day                                                 #
#                                                                            #
# Returns :   $TRUE or $FALSE                                                #
#                                                                            #
##############################################################################
IsValidDayOfMonth ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP
P_DAY="$1"       # numeric day
CURMON=`date +%m`
CURDAY=`date +%d`
if [ $CURMON -eq  $MON ]
then
    if [ $P_DAY -eq $CURDAY -o $P_DAY -gt $CURDAY ]
    then
```

```
            :
        else
            DisplayMessage E "${INVALID_DAY}"
            return $FALSE
        fi
    fi

    # special processing for February
    if [ "${MON}" = "02" ]
    then
        if LeapYear
        then
            if [ P_DAY -gt 0 -a P_DAY -lt 30 ]
            then
                # day is valid
                return $TRUE
            else
                DisplayMessage E "${INVALID_DAY}"
                return $FALSE
            fi
        else
            if [ P_DAY -gt 0 -a P_DAY -lt 29 ]
            then
                # day is valid
                return $TRUE
            else
                DisplayMessage E "${INVALID_DAY}"
                return $FALSE
            fi
        fi

    # process months with 31 days
    elif [ "${MON}" = "01" -o "${MON}" = "03" -o "${MON}" = "05" -o \
           "${MON}" = "07" -o "${MON}" = "08" -o "${MON}" = "10" -o \
           "${MON}" = "12" ]
    then
        if [ P_DAY -gt 0 -a P_DAY -lt 32 ]
        then
            # day is valid
            return $TRUE
        else
            DisplayMessage E "${INVALID_DAY}"
            return $FALSE
        fi
    else
        if [ P_DAY -gt 0 -a P_DAY -lt 31 ]
        then
            # day is valid
            return $TRUE
        else
```

```
        DisplayMessage E "${INVALID_DAY}"
        return $FALSE
    fi
fi
}

##############################################################################
#                                                                            #
# IsValidMonth                                                               #
#                                                                            #
# This function validates a given month.                                     #
#                                                                            #
# Input   :   Variable $MON                                                  #
#                                                                            #
# Returns :   $TRUE or $FALSE                                                #
#                                                                            #
##############################################################################
IsValidMonth ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP
PARAM="$1"
CURMON=`date +%m`
CURYEAR=`date +%Y`
if [  "${PARAM}" != "01" -a "${PARAM}" != "02" -a "${PARAM}" != "03" \
   -a "${PARAM}" != "04" -a "${PARAM}" != "05" -a "${PARAM}" != "06" \
   -a "${PARAM}" != "07" -a "${PARAM}" != "08" -a "${PARAM}" != "09" \
   -a "${PARAM}" != "10" -a "${PARAM}" != "11" -a "${PARAM}" != "12" ]
then
    DisplayMessage E "${INVALID_MONTH}"
    return $FALSE
fi

# month must be equal to or greater than current month if the year
if [ $YEAR -eq $CURYEAR ]
then
    if [ $PARAM -eq $CURMON -o $PARAM -gt $CURMON ]
    then
        return $TRUE
    else
        DisplayMessage E "${INVALID_MONTH_YEAR}"
        return $FALSE
    fi
else
   return $TRUE
fi
}

##############################################################################
#                                                                            #
# IsValidYear                                                                #
```

```
#                                                                              #
# This function validates a given year.                                        #
#                                                                              #
# Input   :   Variable $YEAR                                                   #
#                                                                              #
# Returns :   $TRUE or $FALSE                                                  #
#                                                                              #
################################################################################
IsValidYear ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

PARAM="$1"
CURYEAR=`date +%Y`

# check the year is equal to or greater than the current one
if [ ${PARAM} -eq $CURYEAR -o ${PARAM} -gt $CURYEAR ]
then
     return $TRUE
else
    DisplayMessage E "${INVALID_YEAR}"
    return $FALSE
fi
}

################################################################################
#                                                                              #
# ValidateInput                                                                #
#                                                                              #
# This function validates all date related values.                             #
#                                                                              #
# Input   :   1 String that specifies the type of validation                   #
#             2 Value to be validated.                                         #
#                                                                              #
################################################################################
ValidateInput ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

VAL_TYPE="$1"
VALUE="$2"

# validate day
if [ "${VAL_TYPE}" = "DAY" ]
then
    # check for number
    if ! IsNumeric "${VALUE}"
    then
        return ${FALSE}
    fi
```

```
    # check for two digits
    ARGLEN=`expr "$VALUE" : '.*'`
    if [ $ARGLEN -ne 2 ]
    then
        DisplayMessage E "${INVALID_LENGTH}"
        return ${FALSE}
    fi
    # validate day of the month
    if ! IsValidDayOfMonth "${VALUE}"
    then
        return ${FALSE}
    fi
    # input is valid
    return ${TRUE}
fi
# validate month
if [ "${VAL_TYPE}" = "MON" ]
then
    # check for number
    if ! IsNumeric "${VALUE}"
    then
        return ${FALSE}
    fi
    # validate month
    if ! IsValidMonth "${VALUE}"
    then
        return $FALSE
    fi
    # input is valid
    return ${TRUE}
fi
# validate year
if [ "${VAL_TYPE}" = "YEAR" ]
then
    # check for number
    if ! IsNumeric "${VALUE}"
    then
        return ${FALSE}
    fi
    # validate for appropriate four digits year
    if ! IsValidYear "${VALUE}"
    then
        return ${FALSE}
    fi
    # input is valid
    return ${TRUE}
fi
}

################################################################################
```

```
#                                                                              #
# GetMessageDate                                                               #
#                                                                              #
# This function gets the date for the message.                                 #
#                                                                              #
################################################################################
GetMessageDate ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

MON=
DAY=
YEAR=

# get year for the message
while true
do
    clear
    echo "Enter the year (using four digits, eg 1999, 2000) for the
    ➤  message:\c"
    read YEAR
    case $YEAR in
        "" ) DisplayMessage E "${INVALID_ENTRY}" ;;
         * ) if ! ValidateInput "YEAR" "${YEAR}"
             then
                 :
             else
                  break ;
             fi ;;
    esac
done

# get month for the message
while true
do
    clear
    echo "Enter the month (using two digits eg 01, 10) for the
    ➤  message:\c"
    read MON
    case $MON in
        "" ) DisplayMessage E "${INVALID_ENTRY}" ;;
         * ) if ! ValidateInput "MON" "${MON}"
             then
                 :
             else
                 break ;
             fi ;;
    esac
done
```

```
# get day for the message
while true
do
    clear
    echo "Enter the day (using two digits eg 01, 10) for the
    ➤  message:\c"
    read DAY
    case $DAY in
        "" ) DisplayMessage E "${INVALID_ENTRY}" ;;
         * ) if ! ValidateInput "DAY" "${DAY}"
             then
                 :
             else
                 break ;
             fi ;;
    esac
done
}

#############################################################################
#                                                                           #
# RootUser                                                                  #
#                                                                           #
# This function checks whether the user is root.                            #
#                                                                           #
# Returns :   TRUE if the user is root                                      #
#             FALSE otherwise                                               #
#                                                                           #
#############################################################################
RootUser ( )
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP

USER=`id | cut -d'(' -f2 | cut -d')' -f1`
if [ "${USER}" = "root" ]
then
    return $TRUE
else
    return $FALSE
fi
}

#############################################################################
#                                                                           #
# ProcessExit                                                               #
#                                                                           #
# This function removes temporary files and makes a graceful exit.  #
#                                                                           #
# Input   :   Exit code                                                     #
#                                                                           #
```

```
################################################################################
ProcessExit ( )
{
EXIT_CODE="$1"
clear
rm -f ${TEMP_FILE}
rm -f ${TEMP_FILE_1}
exit ${EXIT_CODE}
}

################################################################################
#                                                                              #
# main                                                                         #
#                                                                              #
# This function invokes all other functions.                                   #
#                                                                              #
################################################################################
main ( )
{
InitialiseVariables
if ! RootUser
then
    DisplayMessage E "${ROOT_USER}"
    ProcessExit $FEC
fi
while true
do
    DisplayMenu
    ProcessOption
done
}

# invoke main
main
```

This article concludes in next month's issue with the script RMOTD.SH (Replace Message-of-the-Day).

Arif Zaman (UK)

A public domain network monitor

ntop is a public domain utility that gathers and displays statistics about network traffic on the subnet to which the AIX host is connected. **ntop**'s two main characteristics are that it's able to gather and display meaningful data in ways beyond the reach of an average system administrator, and that it has a very advanced Web-based user interface.

NTOP INSTALLATION

The source code for **ntop** is available by FTP from the following URL: *http://www-serra.unipi.it/~ntop*. At the time of writing, the latest version of the program was *1.1*, the source code being delivered in a file called *ntop-1.1.tar.gz*. The program uses the **lsof** utility, which is described in 'Using **lsof** and **lslk**' in *AIX Update* Issue 48 (September 1999). A pre-requisite of installing **ntop** is that the *libpcap* library must be installed. Note that you should install the version of *libpcap* that's referred to by a link at **ntop**'s Web site. All my attempts to use more recent versions of this library available from *ftp://fpt.ee.lbl.gov/* resulted in non-operational versions of **ntop**. Also note that the *libpcap* installation bundle is supplied as an executable file – this should be executed first, and then the resulting directory used as the input to the **installp** command (this procedure can also be done via **smit**). The library and its support files are installed in */usr/local/lib*.

After retrieval and unpacking of the archive file, you must configure the software to match your version of Unix (AIX in our case). To do so, you must change directory to the root directory of **ntop** source code tree and issue the following command:

```
# ./configure
```

When the **configure** script terminates, the top-level directory of the source code tree contains the source files and the makefile needed for generation of **ntop** for your version of AIX.

The next step is to invoke **make** command to compile and link the program:

```
# make.
```

Note that the executable generated should be run only on machines that have the same level of the operating system as the system on which the program was built. This is because the program is closely tied to the structure of operating system internals that are frequently altered by the operating system supplier. I was able to build and use the program on AIX 4.1, 4.2.1, and 4.3.2.

ntop doesn't need to be installed at a particular location in the filesystem. However, the program requires read access to special device files and, therefore, if it's going to be used by a non-root user, its ownership and permissions should be set using following commands:

```
# chown root.system  /usr/local/bin/ntop
# chmod 6111 ntop
```

The manual page that's distributed with the program should be installed in a directory that contains manual pages of other public domain tools, for instance: */usr/local/man.*

NTOP INVOCATION

There are two ways to invoke **ntop** – in 'terminal' mode and 'Web' mode. In the former, the command uses the 'curses' library (the *curses* library is in */usr/lib/libcurses.a* and is a library of functions that enable the programmer to create applications that work in full-screen mode on ASCII terminals) and directs its output to a terminal window. The user is able to customize program display options interactively. In the latter, the program executes as a daemon and reports its output using the HTTP protocol, allowing multiple users to view the output using just a standard Web browser.

The **ntop** command has following options:

- **-r <refresh_delay>**

  Specifies the number of seconds between screen updates, the default being three seconds in terminal mode and 120 seconds in Web mode. Note that, during screen updates, the program is unable to process the network traffic fully.

- **-f <traffic_dump_file>**

Directs the program to process a file containing network traffic data collected by the **tcpdump** utility.

- **-n**

  Directs the program to display numeric IP addresses instead of symbolic host names. This is useful if DNS name resolution is slow or problematic.

- **-p <protocols>**

  This flag is used to select specific IP protocols for display. The format is:

  ```
  <label>=<protocol_list>[,<label>=<protocol_list>]
  ```

  where <*label*> is a symbolic identifier for the protocol list. The format of <*protocol_list*> is:

  ```
  <protocol_name>[|<protocol_name>]
  ```

  where <*protocol_name*> is a valid protocol name specified in the */etc/services* file or a numeric port range, such as '80' or '7000-7500'. The default value, which is used when the **–p** argument is omitted, is shown below (the whole value is one line, as indicated by the continuation character, '➤').

  ```
  FTP=ftp|ftp-data,HTTP=http|www|https,DNS=name,Telnet=telnet,
  ➤ Nbios-IP=net-bios-ns,netbios-dgm|netbios-ssn,POP=pop-2|pop-3|
  ➤ kpop,SNMP=snmp|snmp-trap,NFS=mount|pcnfs|bwnfs|nfs|
  ➤ nfsd-status,X11=6000-6010
  ```

- **-i <interface>**

  Specifies the network interface to be used by **ntop**. This is useful when the computer has more than one network interface and the user needs to capture network traffic from a non-default interface.

- **-w <port_number>**

  Directs **ntop** to operate in Web mode. The program utilizes the HTTP protocol, communicating through the *port_number* specified by the **-w** flag. If *port_number* is '8888', then users should point their Web browser to the following URL: *http://ntophost:8888*, where *ntophost* is the name of the host on which **ntop** is running.

To control access to information broadcast by **ntop**, use authentication based on user names and passwords. These are based on the contents of the file *~/.ntop*, which has the following format:

```
#
# ~/.ntop file format
# name<tab_or_space>password
#
operator  #@ntopaccess@#
```

- **-m <local_subnets>**
  This flag is used to specify subnets whose traffic is to be treated as local. The format of *local_subnets* is:

  ```
  <network_address>/<subnet_mask>[,<network_address>/<subnet_mask>]
  ```

  For instance:

  ```
  "134.54.0.0/255.255.255.0,167.12.0.0./255.255.192.0"
  ```

- **-l <loginterval>**
  Tells **ntop** to log collected statistics in the file *ntop.log* each <*loginterval*> seconds. The format of this file is fairly self-explanatory, and its contents are analysed using third-party tools, such as **gnuplot**.

- **-F <flow_definitions>**
  This is used to define 'flows' or 'streams' of captured network packets that match certain criteria. The format of <*flow_definitions*> is:

  ```
  <flow_label>='<matching_expression>'[,<flow_label>=
  ➤  '<matching_expression>']
  ```

  where <*flow_label*> is a symbolic name for the flow. The syntax of <*matching_expression*> is explained in the appendix to the *NTP User's Guide*, which can be downloaded from *http://www-serra.unipi.it/~ntop/*.

- **[optional filter expression] -**
  This allows the user to limit data collected by the program to include only traffic that matches criteria specified by the expression. The syntax of the optional filter expression is identical to that used to define flows.

Figure 1 shows the information displayed when **ntop** is invoked in interactive mode. The top line on the screen displays the version of the command, the version of the host's operating system, and the interface being monitored. The second line displays the total and current network throughput in packets and kilobytes.

The first column of the table contains the names of hosts that have used the network to send or receive data. Hosts are identified by name, if possible, or IP address. By pressing the 'n' key, the user can toggle the display between symbolic name, IP address, MAC address, and network interface manufacturer. The second column displays the type of network by activity in the last period of measurement, possible values being 'I' (Idle), 'S' (Send), 'R' (Receive), and 'B' (Both send and receive). By default, only active hosts on the local subnet are shown, though the 'd' key can be used to toggle between displaying only active hosts or all hosts, and the 'l' key can be used to toggle between displaying all communicating hosts or just hosts located on the local subnet.

The third and fourth columns display the volume of data received and sent by each host. By default, network traffic is displayed in kilobytes, though pressing the 'p' key changes the display to percentages. The column used to sort rows of data is highlighted by two dashes (the user may select different columns by pressing the 't' key). The last three columns display traffic generated by three specific protocols, the default ones being TCP, UDP, and ICMP. The user can select different

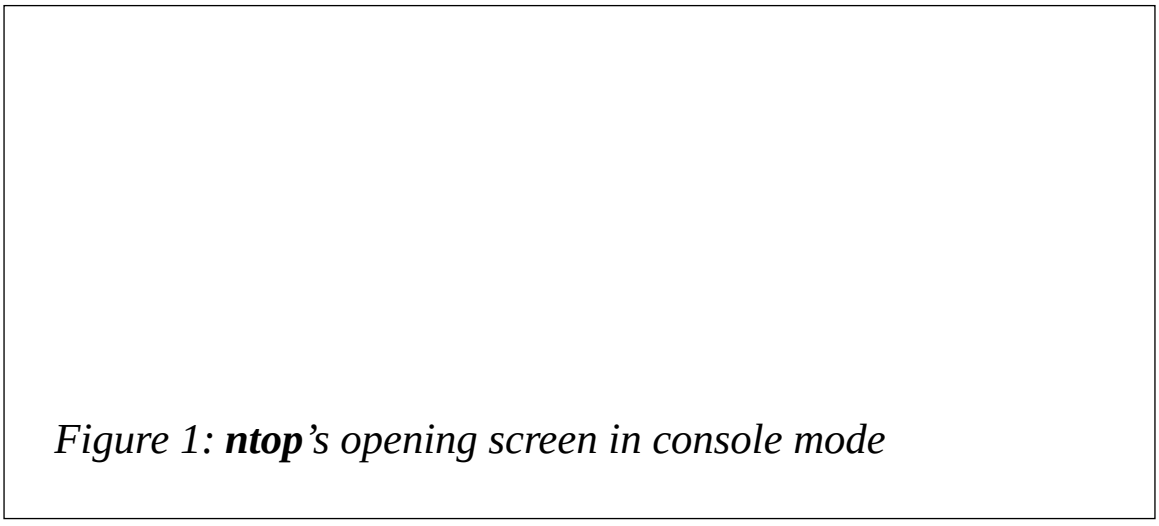

*Figure 1: **ntop**'s opening screen in console mode*

protocols by pressing the space bar. The ‘y’ key toggles the column used for sorting to one of the last three.

NTOP IN WEB MODE

When the **ntop** is invoked in Web mode, users may view network statistics collected by the program by pointing their browser to *http://ntop_host_name:ntop_port_number*, where *ntop_host_name* is the name of the host that executes **ntop** as a daemon and *ntop_port_number* is the number of the port that was passed as an argument using **ntop**’s **-w** flag (see Figure 2).

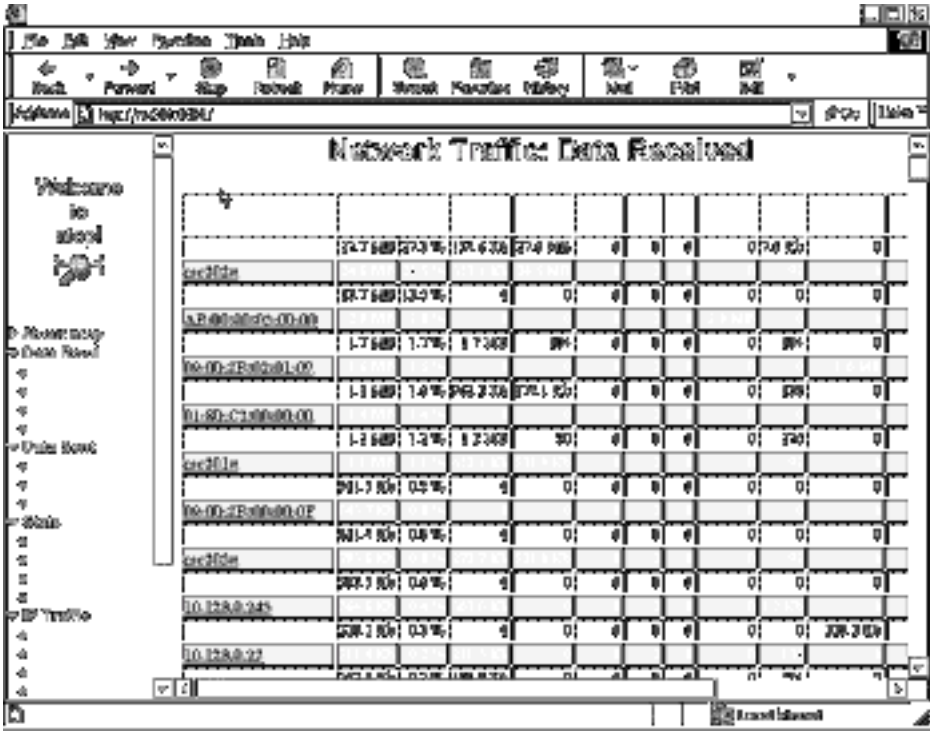

*Figure 2: **ntop**’s Web interface*

The left frame is the menu and the right one displays the actual information. The user can sort table rows by clicking on the column name. Additional information about various items displayed can be obtained by clicking on the various hyperlinks in the tables. Clicking on the hostname or IP address results in host-related statistics being displayed (Figure 3 overleaf).

Figure 3: Additional host information available

The information is displayed in two tables. The first one contains the following items:

- Host’s IP address
- MAC address
- Network card vendor
- Host’s location (local or remote subnet)
- Total data sent/received in bytes and packets
- Broadcast and multicast packets sent by the host
- Distribution of sent and received data by local or remote destination/source.

The second table contains information about the distribution of send and receive data by protocol.

The host information table is sorted according to the bandwidth generated by the host (Figure 4).

Figure 4: Host information table

The ‘IP subnet traffic matrix’ displays amounts of data exchanged between different hosts, as shown in Figure 5.

Summaries are available about both sent and received data, and cells are colour-coded according to level of traffic – cells with high values have a red background, those with intermediate values have a green background, and those with low values have a pink background.

Figure 5: The IP subnet traffic matrix

The network throughput screen (Figure 6) displays histograms of average network throughput during both the last 60 minutes and the last 24 hours.

Figure 6: Network throughput screen

Figure 7: Network traffic distribution

The network traffic distribution screen displays tables that detail the distribution of network traffic by IP protocol type and remote or local destination (Figure 7).

In addition to these charts, many other types of report are produced by **ntop** in Web mode to display network traffic according to various useful criteria.

SUMMARY

In my view, **ntop** is an outstanding program. Its usefulness goes well beyond that of network utilities such as **ping**, **traceroute**, and **tcpdump** that are supplied by the operating system vendors. While these utilities allow system administrators to verify and diagnose basic network connectivity, and the purchase of more expensive and complex software, such as TME10 NetView, is necessary to collect meaningful network statistics, **ntop** provides system and network administrators with the means to answer difficult questions such as:

- Why is local network performance so poor?
- Who is using most of the available network bandwidth?
- Which hosts are currently putting a heavy load on the NFS server?
- What percentage of network bandwidth does a certain network host use?
- Which hosts are connected to processes running on a certain machine and what kind of traffic are they producing?

Additionally the program excels in the way the information is displayed, featuring a clear and detailed Web interface.

I am sure that this program has the potential to be an indispensable tool for every system and network administrator that supervises networks of any size.

Alex Polak
System Engineer
APS (Israel)

A herald generator

If an organization has many RS/6000s and uses **telnet** to access them, it's a good idea to ensure that users know exactly which machine they are logging in to. A good way to do this is to provide a different login prompt for each machine. This can easily be achieved by changing a variable called *herald* in the */etc/security/login.cfg* file.

To make this task even easier, I've written a small script that converts a full-screen design to a herald variable string.

The AIX file reference defines a 'herald' thus:

> "*herald*: Defines the login message printed when the **getty** process opens the port. The default herald is the login prompt. The value is a character string."

If the herald is not specified for a port in *login.cfg*, then a default herald is got from the message catalogue associated with the language set in */etc/environment*.

A possible format for *login.cfg* file is as follows (note the use of the continuation character, '➤', to indicate that one line of code maps to more than one line of print):

```
default:
herald="\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\rIBM AIX
➤  Version 4 for RISC System/6000\n\r(C) Copyrights by IBM
➤  and by others 1982, 1991.\n\rlogin: "

/dev/console:
herald="\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\rIBM AIX
➤  Version 4 for RISC System/6000\n\r(C) Copyrights by IBM
➤  and by others 1982, 1991.\n\rCONSOLE LOGIN: "

/dev/tty12:
herald="\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\rIBM AIX
➤  Version 4 for RISC System/6000\n\r(C) Copyrights by IBM
➤  and by others 1982, 1991.\n\rTTY12 login: "
```

This shows that the herald variable can be used as default for all port's stanzas. It is possible also to have a different login prompt for each 'tty' port.

Customers usually want to make some kind of design in their login prompts. To avoid the trial/error approach, one could create an ASCII file with the desired result. To fill the initial file, you could use the **banner** command. For example:

```
#banner RS/6000 >> heraldxx
#banner COMPANY >> heraldxx
```

Opening the file *heraldxx* using **vi** allows you to make the desired changes using your artistic talents. Remember to use the same number of lines you have available on the TTY on which you want to display the login prompt (24 lines is the most common). An example of a *heraldxx* file is shown below.

```
  #####   #####  #      # ######      #     #      # #      #
 #     # #     # ##    ## #     #    # #   ##     #  #    #
 #       #     # # #  # # #     #   #   #  # #    #   #  #
 #       #     # #  ##  # ######   #     # #  #   #    ##
 #       #     # #      # #        ######## #   #  #    #
 #     # #     # #      # #        #     # #    ##    #
  #####   #####  #      # #        #     # #     #    #

    ######   #####      # #####    ###     ###     ###
    #     # #     #    # #     #   #  #   #   #   #   #
    #     # #          # #         #     # #     # #     #
    ######   #####    #  ######   #     # #     # #     #
    #   #          #   #  #     # #     # #     # #     #
    #    #  #     # #   #     #  #  #   #   #   #   #
    #     #  #####  #    #####    ###     ###     ###

YOU ARE LOGGING MACHINE
IP ADDRESS 202.36.192.4
IBM AIX Version 4.3.x for RISC System/6000
(C) Copyrights by IBM and by others 1982, 1994



LOGIN:
```

The following shell script converts the *heraldxx* ASCII file to a character string:

HERALD.SC

```
#!/bin/ksh
# Herald maker
# This shell script produces a herald variable
```

```
# to be pasted on /etc/login.cfg file

#remove the herstring file if it already exists
rm herstring 2>/dev/null

#usage test
if (( $# < 1 )); then
    echo "type hermaker <herald file>"
    exit
fi

#read the herald file and append to herstring
cat -n $1 | while read x1
do
    x2=`echo "$x1" | cut -f2`
    if [[ $x2 = +([1-9]) ]]; then
        x3=$x3\\\\n
    else
        x3=$x3\\\\n\\\\r$x2
    fi
done
echo "herald = \"$x3 \"" > herstring
```

Now you're ready to paste the herald variable into *login.cfg*. It is advisable to make a copy of *login.cfg* before changing it. Then open *login.cfg* with **vi**, put the cursor below the desired point, and paste 'herstring'. This is done from **vi** command line using the following command:

```
:r herstring
```

Make sure that there are no blank lines between the herald variable and the desired stanza. The login prompt will change when the next **getty** process is created for that TTY.

Andre Barczak
Technical Consultant (New Zealand)

November 1995 – October 1999 index

Below is an index of all topics covered in *AIX Update* since Issue 1, November 1995. The numbers in **bold** are issue numbers, and the numbers in brackets are page numbers. If you'd like to order back-issues of *AIX Update*, please contact Xephon. You'll find contact information listed on page 2 – back-issues are available from Issue 1, and can be obtained from any of Xephon's offices.

Contributing to *AIX Update*

AIX Update is primarily written by practising AIX specialists in user organizations – not journalists, consultants, or marketing people. In our view, such information is far more valuable to AIX professionals than that from other sources.

If you're interested in contributing to *AIX Update*, please download a copy of *Notes for contributors* from Xephon's Web site at *www.xephon.com*. Articles to be considered for publication can be sent the editor at *HarryL@xephon.com*.

AIX news

Merant has announced that Micro Focus Server Express, the company's product for deploying COBOL applications, now runs on AIX 4.3. Server Express helps extend applications to the Web, Unix, and other distributed platforms. Merant also supports AIX 4.3 through the company's PVCS software configuration manager.

For further information contact:
Merant, 2465 E Bayshore Road, Palo Alto, CA 94303, USA
Tel: +1 650 856 4161
Fax: +1 650 856 3724
Web: http://www.microfocus.com

Merant, Speen Court, 7 Oxford Road, Newbury, Berks, RG14 1PB, UK
Tel: +44 1635 32646
Fax: +44 1635 33966

* * *

IBM has announced Version 3.0 of both WebSphere Enterprise Edition and Advanced Edition. The Enterprise Edition includes TXSeries Version 4.3, Component Broker V3.0, CICS Transaction Gateway V3.0.2, DE-Light Gateway, and Component Broker Application Adapters for CICS/IMS, DB2, Oracle, and MQSeries. Also in the package are CICS Universal Client V3.0.2 and the Encina Client V4.3.

Other tools include DB2 UDB V5.2, DCE V2, and MQSeries V5.1, plus a range of application development tools including an early release of VisualAge for Java Enterprise Edition V3, C++ Compiler and Libraries, DB2 Software Developer's Kit V5.2, and Component Broker Toolkit V3.0. Out now on AIX, NT, and Solaris, prices start at US$35,000.

For further information contact your local IBM representative.

* * *

Iona has announced OrbixHome, an IDE for EJB and CORBA that now includes facilities for deployment. Previously dubbed Orbix BeansTalk, the product supports the EJB 1.1 specification and brings together Iona's HomeBase EJB container with the Orbix middleware family, adding a graphical environment for building and deploying middleware components. Initial availability is on AIX, Solaris, HP-UX, Compaq/DEC Unix, and Windows NT. It's due to Ship later this year (prices weren't announced).

For further information contact:
Iona Technologies, 60 Aberdeen Avenue, Cambridge, MA 02138, USA
Tel: +1 617 949 9000
Fax: +1 617 949 9001
http://www.iona.com

Iona Technologies UK Ltd, The Atrium Court, Apex Plaza, Reading RG1 1AX, UK
Tel: +44 118 925 4241
Fax: +44 118 958 7724