105

AIX update

July 2004

In this issue

AIX Update

Published by
Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

Phone: 214-340-5690
Fax: 214-341-7081

Editor
Trevor Eddolls
E-mail: trevore@xephon.com

Publisher
Nicole Thomas
E-mail: nicole@xephon.com

Subscriptions and back-issues

A year's subscription to *AIX Update*, comprising twelve monthly issues, costs $275.00 in the USA and Canada; £180.00 in the UK; £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 2000 issue, are available separately to subscribers for $24.00 (£16.00) each including postage.

AIX Update on-line

Code from *AIX Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at http://www.xephon.com/aix; you will need to supply a word from the printed issue.

Disclaimer

Contributions

When Xephon is given copyright, articles published in *AIX Update* are paid for at the rate of $160 (£100 outside North America) per 1000 words and $80 (£50) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of $32 (£20) per 100 lines. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

Securing Apache with SSL

To secure Web server communications, ideally one would want to encrypt communications between the Web server and the calling browser for information that is regarded as sensitive or personal. This can be accomplished using Apache and Secure Socket Layer (SSL). To determine whether a browser has a secure connection to a Web site, look at the toolbar. If you see the padlock icon in a closed position, this indicates that the connection is to a secure server where the traffic is encrypted. In a normal connection (using HTTP) the padlock would be in an open position.

SSL is the open standard security protocol for the secure transfer of sensitive information over the Internet. As a protocol, it runs above TCP/IP and below HTTP. When using SSL, the server and client can exchange secure information after being authenticated through the use of keys. When a connection to an SSL-aware site is established, a little padlock icon in the locked position appears on your browser. This informs you that the page is now encrypted; also, the URL starts with https: rather than http:.

The packages for SSL to function with are Apache, of course, and Open_SSL. The SSL Apache packages can be found on the AIX ToolBox Linux Application CD. They are in either source or binary RPMs. A better alternative is to download the source from the Apache Web site and compile it. I installed Apache into the */opt/freeware/apache* directory; your directory structure may be different.

The Mod_SSL module provides cryptography via the SSL and the Transport Layer Security (TLS). The OpenSSL contains the OpenSSL toolkit. OpenSSL enables the creation of private keys and certificates. Once the Apache packages have been installed, Apache needs to be restarted:

```
# /opt/freeware/apache/sbin/apachectl  stop
# /opt/freeware/apache/sbin/apachectl  start
```

Be sure to check the Apache error log for errors or any inconsistencies. To have a tidy directory to hold the keys and certificates, I created two directories: one for the server's keys and one for the server's certificates.

THE KEYS

Using traditional methods, a symmetrical encryption is used to encrypt/decrypt information – in other words, the same key is used to both encrypt and decrypt. When sending a key to another party, there is no way of knowing, while sending, whether it has arrived safely with the person you intended to send it to. This is a big problem, especially when dealing with data communication over the Internet. Fortunately, a solution is at hand, that is to use two keys instead – a private key and a public key. The server generates both of these keys. As its name suggests, the public key is available to anyone who wants to read it. This key is used to encode the data that is sent back from the client to the server. The private key is kept on the server in a secure location. This key is used to decode incoming communication. The use of these keys allows the server and client to communicate in a private manner. To reiterate, only the server that holds the private key can read messages encrypted by the public key. Many clients can hold the public key, but there is only one private key holder that can unlock the information from those public keys. A server can have many private keys, but all of them will be random and unique. An ISP will undoubtedly have many private keys for the different domains it hosts.

DIGITAL CERTIFICATES

A Digital Certificate is an electronic document used to identify the provider of a public key. This stops impersonation or mis-use of a public key. Certification Authorities (CA) issue certificates. They are trusted companies that verify the identity of the site that a client connects to. This type of process is akin to when you apply for a passport – you need to provide proof of who you are, like a birth certificate and a proposal form from an outstanding citizen. This type of service is the same as that provided by a CA. When applying for a Certificate request, the CA will check who you are by various means. At the least this will involve proof of who you are, generally a copy of your company/organization registration certificate and a hardcopy from the WHOIS database. Once a certificate is issued, which can take up to a week, it will bind the public key to the name of the requester on the certificate, usually an FQDN, though not necessarily so. Only a public key certified by the certificate will work with the corresponding private key. In this article we will generate our own temporary certificate. The process is the same as for a CA, but of course a CA will not have signed our certificate. But proof of concept will prevail. When you have a CA-signed certificate, the clients will automatically acknowledge the certificate and thus secure communication begins. They are also guaranteeing the identity of the organization or company to whom the certificate has been issued. With a self-signed certificate, some browsers will prompt you that an unsigned certificate is about to be loaded by your browser, but of course you still get the secure connection.

THE PROCESS OF SSL

The SSL server authentication allows a browser to confirm a server's identity. SSL-aware software on the (client) browser can use standard public-key encryption to check that a server's certificate is valid and has been issued by a Certificate Authority. This can be found in the browser's list of trusted CAs.

The SSL client authentication allows a server to confirm a user's identity, using the same techniques. Here's what happens when an SSL connection occurs:

- The client sends a request to connect.
- The server sends a signed certificate (this includes the public key).

- The client verifies the certificate signer, by checking its acceptable CA list.
- The client generates a session key to be used for encryption and sends it to the server encrypted with the server's public key.
- The server uses the private key to decrypt the client-generated session key.
- Client and server exchange information (encrypted, of course).

CREATING THE PRIVATE KEY

A cipher key needs to be at least 1024 bits – anything below this is not considered good for secure commercial communication. Using the OpenSSL utility, an RSA-based key using the triple DES cipher can be generated:

```
# openssl genrsa -des3 1024 > /opt/freeware/apache/ssl.key/server.key
```

If you own more than one domain name that runs from your server, it makes sense to prefix the key with the domain name, like so:

```
# openssl genrsa -des3 1024 >/opt/freeware/apache/ssl.key/
www.somecompany.key
```

When generating a key, it will prompt for a password/pass phase to be entered; openssl will prompt for this password every time Apache starts up. Do remember to document it! Alternatively it may not be appropriate within your business operating environment to have Apache prompt for a password every time it starts up; in that case create a non-encrypted key, by not specifying the *-des3* option, like so:

```
# openssl genrsa 1024 > /opt/freeware/apache/ssl.key/server.key
```

The key needs to be read and written by root only. If this task is not carried out, a user or some third party could get hold of the key and, heaven forbid, impersonate your secure server:

```
# chmod 600 /opt/freeware/apache/ssl.key/server.key
```

GENERATING A CERTIFICATE REQUEST

As discussed previously, we will look at creating a self-signed certificate. However, to create a certificate request to send to a CA, enter the following command:

```
# openssl req -new -key /opt/freeware/apache/ssl.key/server.key -out /
opt/freeware/apache/ssl.csr/server.csr
```

The above command will use the private key just created (server.key) and use it to generate a certificate request file (server.csr). You will be prompted for the password/phase you gave when the private key was generated. A series of questions will follow. When completed, the next step is to send the file to a CA. Three common CAs are:

- www.thawte.com
- www.verisign.com
- www.instanstssl.com.

GENERATING A SELF-SIGNED CERTIFICATE

A self-signed certificate should always be considered a temporary certificate until you receive a valid one from a CA. However, having said that, it is quite common to see self-signed certificates used internally within company LANs/ WANs.

To properly validate a certificate requires a private key, which has already been generated. The next step is to create the self-signed certificate.

As with the Certificate Request, you will be prompted for the password or phase you entered when creating the private key. If you decided not to enter a password for your key then you will not be asked for a password when generating your certificate:

```
# openssl  req -new -key /opt/freeware/apache/ssl.key/server.key -x5Ø9 -
days 365 -out /opt/freeware/apache/ssl.crt/server.crt
```

This uses the configuration from */opt/freeware/apache/ssl/ openssl.cnf*.

You are about to be asked to enter information that will be incorporated into your certificate request.

What you are about to enter is what is called a Distinguished Name, or a DN.

There are quite a few fields but you can leave some blank. For some fields there will be a default value. If you enter '.', the field will be left blank.

```
Country Name (2 letter code) [AU]:GB
State or Province Name (full name) [Some-State]:Essex
Locality Name (eg, city) []:Grays
Organization Name (eg, company) [Internet Widgits Pty Ltd]:A Company Ltd
Organizational Unit Name (eg, section) []:Payroll
Common Name (eg, your name or your server's hostname) []:bumper
Email Address []:dtansley@techemail.com
```

The above command will generate a file called server.crt, containing the self-signed certificate, located in */opt/freeware/apache/ssl.crt/ssl.crt*, using the private key (server.key) created earlier. The **–days** parameter is for how long the certificate will be valid before being rejected by a browser. In the above example the certificate will be valid for a year (365 days); alternatively you can have the temporary certificate valid for 28 days if you so wish, using the parameter:

```
-days 28
```

The *x509* is the standard used when creating certificates; it defines the data and signature parts of the certificate, where data will consist of the serial number, public key, and issuer of the certificate.

A series of questions is then presented as in the above output. Please input the answers to the best of your ability. Some of the questions already have a default answer and you can hit *Enter* on these if you wish.

Please note, the Common Name should match up with the URL of your Web server. If you are on a private network, input the server name here instead. For example if users use, say, bumper as the URL to point to your internal Web server, then use bumper as the Common Name.

After all the information has been entered a self-signed certificate (server.crt) file will be created.

INFORMING APACHE ABOUT SSL

If you compiled Apache from source, you may find you have a separate *ssl.conf* file, where all the SSL options must go. Alternatively, edit the *httpd.conf* file. Although Apache comes pretty much SSL aware, it will need checking, especially on the secure port and the location of the certificates and keys. Most entries go inside the <VirtualHost> block:

```
Listen 80
Listen 443
<VirtualHost   _default_:443>
# virtual servername  - same as your web server
 ServerName www.bumper.co.uk
# virtual document root - change to suit
DocumentRoot "/opt/freeware/www/html"
 #   SSL Engine Switch:
SSLEngine on
# server certificate
SSLCertificateFile /opt/freeware/apache/ssl.key/server.key
#   Server Private Key:
SSLCertificateFile /opt/freeware/apache/ssl.crt/server.crt
</VirtualHost>
```

RUNNING SSL

The next step is to restart Apache:

```
# /opt/freeware/apache/sbin/apachectl  stop
# /opt/freeware/apache/sbin/apachectl  startssl
```

If you have installed the binary version then start Apache with the DSSL flag:

```
# /opt/freeware/apache/sbin/httpd -DSSL
```

When SSL is running HTTPS it will listen on port 443, with normal HTTP listening on port 80. Use **netstat –a** to determine that both http and https are listening.

OpenSSL can also be used to test the handshake of OpenSSL, using an SSL/TLS connection. The following command will produce a long list of connection details, using the key and certificate information that is currently installed. It is a good source to further diagnose any problems if HTTPS is running, but the browser cannot connect to the secure server. It is also gives a good grounding in understanding the process of how OpenSSL interacts:

```
# openssl s_client -connect localhost:443 -state
```

To access a Web page over a secure communication, point a Web browser to one of your Web pages. The URL should start with https, and not http, the 's' indicating a secure HTTP site. So to access a Web server called bumper you would use:

```
https://bumper
```

For the local host use:

```
https://localhost
```

Please note: there is no need to specify the port number for a secure connection.

Depending on the type of browser being used, it may prompt for a series of confirmations before the certificate is loaded by the browser that it is connecting to a secure site. When the connection is established, the padlock icon on the browser will be in a locked position. The certificate details can be displayed by clicking on the padlock. Notice that the 'Certificate State' entry states it is a self-signed certificate.

David Tansley
Global Production Support (IBM p-series) (UK)

Filesystem check

AIX system administration is always a time-consuming job. One major issue is that of looking after filesystems. Often applications write data to a filesystem until the filesystem is filled up and the application stops running. Although AIX error reporting detects that a filesystem is full, it would be nice to have someone alert you *before* a filesystem becomes full.

The following script does exactly this. The script checks the specified filesystems and sends an e-mail to specified users. In my environment there is one host, which has one filesystem that is mounted over NFS on all the other AIX hosts. So I placed the scripts and all the necessary files in that filesystem. I have created an entry in the crontab on every host. The script is run every hour.

The script reads two filetypes:

1 File sgr_fs_'hostname'

The first filetype contains the name of the filesystem and the limit. The limit represents the percentage limit when an alert is fired. The scripts checks the *%Used* value of the **df** command (see example below):

```
df /tmp

Filesystem    512-blocks      Free %Used    Iused %Iused Mounted on
/dev/hd3          720896    361424   50%     2955     4% /tmp
```

The name of the file contains the hostname. For every host that needs to be checked, there is one file in the above mentioned filesystem. Thus the script can be run on different hosts without changing the script. The layout of the file is shown below. Lines that contain # are ignored.

```
# filesystems that need to be checked
# file layout:
#
# <filesystem> <limit>
#
# <filesystem> = file system (mount point) that must be checked
# <limit> = if limit in % is exceeded action is taken
#
/tmp 80
/usr 90
/ 80
# end of file
```

2 File sgr_fs_mailrec and sgr_fs_mailrec_'hostname'

The second file contains the e-mail addresses of every user who needs to be notified. In my environment I had a problem that on different hosts different users needed to be notified. So I created two similar files. The first file contains the users who always need to be notified, the second file contains the hostname in its name and all the users that need to be notified on that specific host. When the script finds both files, they are concatenated. Lines that contain # are ignored.

File sgr_fs_mailrec:

```
# mail recipients for e-mail notification
# to exclude a person, just comment out the line using #
#
# format: name@domain
#
#
admin@mailhost.com
# end of file
```

File sgr_fs_mailrec:

```
# mail recipients for e-mail notification on a specific host
# to exclude a person, just comment out the line using #
#
# format: name@domain
#
#
application.owner@mailhost.com
# end of file
```

WHAT DOES THE SCRIPT DO?

The first function, *check_email_rec*, checks the availability of the file with the e-mail recipients. If the host-specific file is found, it is concatenated with the normal e-mail recipient's file.

The second function, *check_inputfile*, checks the availability of the file that contains the names of the filesystems to be checked. If it is not found, it just exits with a message written to the log file.

The third function, *check_filesystem*, reads the input file. If the filesystem to be checked is not available, a message is written to the logfile and it exits. If the filesystem exists, it is checked using the **df** command.

Using the **set** command splits the output of **df** into $-Variables. Then the fifth parameter, which contains the usage value, is compared with the limit value in the input file. If the usage value exceeds the limit value an e-mail is sent to all the specified users.

And here is the script:

```
#!/usr/bin/sh
#
#
# define variables
export  FS=/usr/rz/proclib/sgr_fs_'hostname'
export  EMAIL_FILE=/tmp/email.$$
export  EMAIL=/tmp/email_cat.$$
export  EMAIL_REC_HOST=/usr/rz/proclib/sgr_fs_mailrec.'hostname'
export  LOG=/tmp/sgr_fs_logfile.$$
export  EMAIL_REC=/usr/rz/proclib/sgr_fs_mailrec

######################################################################
#
#  FUNCTION check_email_rec
#
#
# if there is a host specific e-mail recipient file available then
# set input file for email-recipients to that file name
check_email_rec()
{
if [ ! -a $EMAIL_REC_HOST -o ! -s $EMAIL_REC_HOST ]
then
    cat  $EMAIL_REC   > $EMAILelse
#      EMAIL=$EMAIL_REC_HOST
      cat $EMAIL_REC_HOST $EMAIL_REC > $EMAIL
fi
}
#   End of funtion
######################################################################
######################################################################
#
#  FUNCTION check_inputfile
#
#
# is file with filesystems to be checked?
# if no then do nothing and exit
check_inputfile()
{
if [ ! -a $FS -o ! -s $FS ]
then
        echo ,date, $Ø „File $FS cannot be found or is empty" >> $LOG
        echo 'date' $Ø "********** END ********************" >> $LOG
        exit Ø
fi
}
#   End of funtion
#####################################################
#####################################################
#
#  FUNCTION  check_filesystem
#
#
# read file with filesystem to check
check_filesystem()
{
while read F L
  do
     if [ "$F" != "#" ]; then
                            #ignore line if first character is comment
        if [ ! -d $F ]
        then
              echo 'date' $Ø "Filesystem $F to be checked" >> $LOG
              echo 'date' $Ø "cannot be found " >> $LOG
              echo 'date' $Ø "********** END ************" >> $LOG
              exit Ø
          else
              set 'df -P $F | tail +2' # get last line of df commend
              set 'echo $5 | tr -d %'  # get percentage field
              if [ $1 -gt $L ]; then   # compare percentage field with
                                       # limit in file
              generate_email $1
              fi
         fi
     fi
done <$FS
}
#   End of funtion
#####################################################
#####################################################
#
#  FUNCTION generate_email
#
#
generate_email()
{
```

```
for k in 'cat $EMAIL |  grep "^[a-zA-Z]"'
do
      ls -l $F >> $EMAIL_FILE  # file to be sent to email recipients
      mail -s "Filesystem $F on `hostname` is $1 % full" $k <
$EMAIL_FILE
      rm $EMAIL_FILE
done
rm $EMAIL
}
#   End of funtion
#######################################################################
#######################################################################
#
#  Main function
#
check_email_rec;
check_inputfile;
check_filesystem;
```

Robert Schuster
Systems Administrator (Germany)

E-mail alerts

Our e-mail alert service will notify you when new issues of *AIX Update* have been placed on our Web site. If you'd like to sign up, go to http://www.xephon.com/aix and click the 'Receive an e-mail alert' link.

Reporting information about space on shark storage (ESS)

Unfortunately, there is no AIX command that displays information about the total space and available space on shark storage that is connected to a server. In other words, we needed a **df –k**-like command to see how much space is allocated and how much space is free on our shark storage disks. That's why I have written a script, shark_space.sh, which gets the following information.

If you do not specify a flag:

- Which sharks are connected
- How much space is allocated
- How much space is free.

An example of the output:

```
shark 17152  -- 23808 MB total ,  22208 MB free (93%)
shark 18142  -- 199984 MB total ,  5504 MB free (2%)
shark 14654  -- 95232 MB total , 10608 MB free (11%)
```

In the example above, 171152, 18142, and 14654 are the shark ids, which indicates that three sharks are connected to the server and they have the space statistics shown.

If the **–m flag** is specified, then the script mails the output in a certain format to a dedicated AIX server, which collects all the shark space output from all the servers using shark storage in our company. Additionally, it logs this information to a history file for later use.

An example of the history log file (shark_space_history.log):

```
..............................................................
..............................................................
--------------------------------------------------------------
Shark Space at: 06/02/2004
```

```
shark 17152  -- 23808 MB total ,  22208 MB free (93%)
shark 18142  -- 199984 MB total ,  5504 MB free (2%)
shark 14654  -- 95232 MB total , 10608 MB free (11%)
-------------------------------------------------------------
-------------------------------------------------------------
Shark Space at: 13/02/2004
shark 17152  -- 23808 MB total ,  20608 MB free (86%)
shark 18142  -- 199984 MB total ,  5504 MB free (2%)
shark 14654  -- 95232 MB total , 10208 MB free (10%)
-------------------------------------------------------------
-------------------------------------------------------------
Shark Space at: 20/02/2004
shark 17152  -- 23808 MB total ,  18208 MB free (76%)
shark 18142  -- 199984 MB total ,  5504 MB free (2%)
shark 14654  -- 95232 MB total , 10208 MB free (10%)
-------------------------------------------------------------
-------------------------------------------------------------
Shark Space at: 27/02/2004
shark 17152  -- 23808 MB total ,  18208 MB free (76%)
shark 18142  -- 199984 MB total ,  5504 MB free (2%)
shark 14654  -- 95232 MB total , 8208 MB free (8%)
-------------------------------------------------------------
-------------------------------------------------------------
Shark Space at: 05/03/2004
shark 17152  -- 23808 MB total ,  16608 MB free (69%)
shark 18142  -- 199984 MB total ,  5504 MB free (2%)
shark 14654  -- 95232 MB total , 8208 MB free (8%)
-------------------------------------------------------------
.............................................................
.............................................................
```

To have an overall weekly report, all servers using shark storage must have a crontab entry as follows (which works every Friday at 05:00 am.):

```
0 5 * * 5 /usr/local/sbin/shark_space.sh -m  > /dev/null 2>&1
```

Let's say the server that collects output is myserver1 and **shark_space.sh –m** works on myserver2, myserver3, … , myserver*n*. Then the format of the output that is sent to myserver1 would be as follows:

```
<hostname>:<shark id>:<total space>:<free space>
```

An example (with three sharks):

```
myserver2:17152:23808:22208
myserver2:18142:199984:5504
```

```
myserver2:14654:95232:10608
```

I can give the content of the mail I am sending to myserver1 as an input to a script named write2file.sh on myserver1. To be able to do that, add the following entry to the */etc/aliases* file on myserver1:

```
ess: "| /home/user555/write2file.sh"
```

Note: the command **sendmail -bi** must be run after */etc/aliases* file is updated for any changes to affect sendmail operation.

Whenever I send a mail to ess@myserver1.domain.com from myserver2, … , myserver*n*, the write2file.sh on myserver1 will be executed. Basically, write2file.sh gets the content of the mails and appends them to a file.

And the last step is to execute a third script, shark_summery.sh, from crontab on myserver1 to add all the allocated and available space for each shark and prepare a weekly report. After that, this report is sent to the appropriate people/managers as an Outlook mail.

Crontab entry (must be after **shark_space.sh –m**):

```
0 6 * * 5 /usr/local/sbin/shark_summery.sh  > /dev/null 2>&1
```

(It works every Friday at 06:00 am.)

A weekly report example:

```
----------------------------------------------------------------

  WEEKLY SHARK SPACE REPORT:

  Hostname:  myserver2
  Shark 17115 -- 857120 MB used  19520 MB free
  Shark 18142 -- 309504 MB used  53888 MB free
  Shark 14654 -- 1166624 MB used  49600 MB free

  Hostname:  myserver6
  Shark 17115 -- 9520 MB used  3952 MB free

  Hostname:  myserver11
  Shark 18142 -- 95232 MB used  6336 MB free
```

```
Hostname:  myserver12
Shark 17115 -- 80944 MB used  16848 MB free

Hostname:  myserver16
Shark 17115 -- 633920 MB used  81520 MB free

Hostname:  myserver18
Shark 17115 -- 71424 MB used  2336 MB free

Hostname:  myserver21
Shark 17115 -- 718992 MB used  33616 MB free
Shark 14654 -- 666624 MB used  666624 MB free

Hostname:  myserver27
Shark 17115 -- 9520 MB used  3840 MB free
Shark 18142 -- 37888 MB used  10880 MB free

Hostname:  myserver29
Shark 17115 -- 119024 MB used  8016 MB free
Shark 18142 -- 23808 MB used  17792 MB free

Hostname:  myserver33
Shark 18142 -- 47616 MB used  6240 MB free

Hostname:  myserver44
Shark 18142 -- 14272 MB used  10208 MB free

Hostname:  myserver45
Shark 17115 -- 199984 MB used  5504 MB free
Shark 18142 -- 23808 MB used  22208 MB free

Hostname:  myserver48
Shark 17115 -- 190192 MB used  40432 MB free
Shark 18142 -- 4736 MB used  2496 MB free
Shark 14654 -- 95232 MB used  25088 MB free

Hostname:  myserver55
Shark 18142 -- 95232 MB used  4736 MB free

Hostname:  myserver57
Shark 17115 -- 152560 MB used  17952 MB free
Shark 14654 -- 152560 MB used  17952 MB free

Hostname:  myserver61
Shark 17115 -- 18944 MB used  4032 MB free

 ...............................................
 ...............................................
```

```
  ######################################
 #  TOTAL SPACE SUMMARY           #
  ######################################

  Shark 17115: 2990 GB
  Shark 18142: 636 GB
  Shark 14654: 2032 GB

----------------------------------------------------------------
```

SHARK_SPACE.SH

```
#!/bin/ksh
#
######################################################################
#
# Adnan Akbas, Turkcell
#
# Version 1.0   22.02.2003
# Version 1.1   05.04.2003   mailing output to myserver1
#
# This script displays information about total space and
# available space on sharks connected to this server.
#
# If the -m flag is specified, then the script mails the
# output in a certain format to myserver1 which collects
# all shark space output ...
#
#########################################################################

# This function displays usage of this script

function usage {

[[ -n ${DEBUG} ]] && set -x

cat << EOF

Usage: ${script} [-m] [-v] [-h]

        -m: Output mail to "$mailto"
        -v: Version
        -d: Debug
        -h: Help

This script displays information about total space and
available space on sharks connected to this server, if
no flag is specified. If the -m flag is specified, then
the script mails the output in a certain format to the
dedicated server ...

EOF
exit

}

# This function displays the version info.

function version {

[[ -n ${DEBUG} ]] && set -x

print
echo "$ver  - Author: Adnan Akbas - Turkcell"
exit

}

# This function checks whether a vpath exists on this server

function check {

[[ -n ${DEBUG} ]] && set -x

# Get all vpaths
vpath_list='lsdev -C -c disk -s dpo -t vpath -F name'

# Is there a shark connected?
if [[ "${vpath_list}" = "" ]]
then
  print
  print "NO Shark Storage Device is connected !!!"
  print
  exit
fi

}

# This function checks whether a volume group is active

function check_if_active {

[[ -n ${DEBUG} ]] && set -x

lsvg -o | grep -q $vg
[[ $? = Ø ]] && vg_active=true

}
```

```
# This function determines the shark ids connected to this server

function get_shark_ids {

[[ -n ${DEBUG} ]] && set -x

for vpath in $vpath_list
do

  # get device unit serial number
  odmget -q "name = $vpath" CuDv|egrep connwhere|awk '{print $3}'|tr
'\"' ' ' | cut -c5-9

done

}

# This function adds spaces total/free in sharks

function calculate {

[[ -n ${DEBUG} ]] && set -x

# Getting all sharks that are connected to an array named shark
set -A shark 'get_shark_ids | sort | uniq'
let num_of_sharks=${#shark[*]}-1

# initializing the counters
let i=0
while (( i <= $num_of_sharks ))
do
  let shark_total[$i]=0
  let shark_free[$i]=0
  let i=${i}+1
done

for vpath in $vpath_list
do

  let pp_size='lspv $vpath | grep "PP SIZE" | awk '{print $3}''
  vg='lspv $vpath | grep "VOLUME GROUP" | awk '{print $6}''
  let pp_total='lsvg -p $vg | grep $vpath | awk '{print $3}''
  let pp_free='lsvg -p $vg | grep $vpath | awk '{print $4}''
  shark_id='odmget -q "name = $vpath" CuDv|egrep connwhere|awk '{print
$3}'|tr '\"' ' ' | cut -c5-9'
  vg_active=false
  check_if_active
  if $vg_active
  then
```

```
    # Finding total disk size in MB for each vpath
    let disk_size=${pp_size}*${pp_total}

    # Finding free disk size in MB for each vpath
    let disk_free_size=${pp_size}*${pp_free}

    shark_id='lsvpcfg $vpath | awk '{print $5}' | cut -c4-8'
    let i=0
    while (( i <= $num_of_sharks ))
    do

      # Adding the size allocated/free in each shark storage
      if [[ $shark_id = ${shark[$i]} ]]
      then
        let shark_total[$i]=${shark_total[$i]}+${disk_size}
        let shark_free[$i]=${shark_free[$i]}+${disk_free_size}
      fi
      let i=${i}+1

      # This calculation takes a while and produces no output, so
      # this command puts . on the screen while the loop turns
      echo ".\c"
    done
    echo ".\c"
  fi
done
print
print

}

# This function sends the shark space usage output via mail in a
# certain format to the appropriate server and logs the output
# also in a history file if the script is executed with "-m" option.
# If no flag is specified, it displays shark space usage output only
# on the screen ...

function result {

[[ -n ${DEBUG} ]] && set -x

let i=0

if $mail
then

  echo "-------------------------------------------------------" >>
$logfile
```

```
  print "Shark Space at: $today" >> $logfile

  while (( i <= $num_of_sharks ))
  do

    # Sends mail to the server (alias) with a certain format
    echo "`hostname`:${shark[$i]}:${shark_total[$i]}:${shark_free[$i]}"
| mailx $mailto

    # And logs to the output to a history file for later use
    print "shark ${shark[$i]}  -- ${shark_total[$i]} MB total ,
${shark_free[$i]} MB free ($(( (${shark_free[$i]} * 100) /
${shark_total[$i]} ))%)" >> $logfile

    let i=${i}+1
  done
  echo "-----------------------------------------------------" >>
$logfile

else

  # Prints result on the screen
  while (( i <= $num_of_sharks ))
  do
    print "shark ${shark[$i]}  -- ${shark_total[$i]} MB total ,
${shark_free[$i]} MB free ($(( (${shark_free[$i]} * 100) /
${shark_total[$i]} ))%)"
    let i=${i}+1
  done

fi

}

# Variables #############################

script='basename ${0}'
logfile=/var/adm/shark_space_history.log
today='date +"%d/%m/%Y"'
ver="Version 1.1    05.04.2003"
mail=false

# Here you can determine which server (alias) you can send your output
mailto=ess@rsc003e0.eil.risnet.de

# Main ##################################

# Getting the options
while getopts :mvdh opt
   do
```

```
      case $opt in
            v) version ;;
            m) mail=true ;;
            d) export DEBUG=yes ;;
            h) usage ;;
            *) usage  ;;
      esac
   done

# checking ifwhether we are in debug mode?
[[ -n ${DEBUG} ]] && set -x

# Running the appropriate functions
check
calculate
result

# If we are in debug mode, turn off
[[ -n ${DEBUG} ]] && unset DEBUG
```

WRITE2FILE.SH

```
#!/bin/ksh
##########################################################
#
# Adnan Akbas, Turkcell
#
# 08.04.2003
#
# This script works when a mail comes to the ess alias and
# writes the content of the mail (shark space statistics)
# to a file for later calculation ....
#
##########################################################

# Variables ###########################

# Variables ###########################


# Main ################################

while read line
do

  # getting rid of the mail headers
  echo $line | grep -q "^*"

  if [ "$?" = "0" ]
  then
    echo $line >> $logfile
  fi
done

# Deleting log files older than a month

 find ${local_dir} -name "shark_*" -mtime +30 -exec rm {} \;

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

SHARK_SUMMARY.SH

```
#!/bin/ksh
#########################################################
#
# Adnan Akbas, Turkcell
#
# 09.04.2003
#
# This script prepares the report about shark space and
# sends it to the appropriate people via Outlook mail.
#
#########################################################

# Variables ##################################

today='date +"%d%m%Y"'
local_dir=/var/adm/tmp
logfile=${local_dir}/shark_${today}.out
outfile=/tmp/sharks.out
mailto=adnan.akbas@turkcell.com.tr,trevore@xephon.com

# MAIN ########################################

# Getting all servers into an array

let i=0
cat $logfile | awk -F: '{print $1}' | sort | uniq |
   while read host
   do

     # getting rid of the "*"
     server[$i]=${host#*\*}
     let i=${i}+1

   done
```

```
# start to prepare the report

print > $outfile
print "WEEKLY SHARK SPACE REPORT:" >> $outfile
print >> $outfile

for srv in 'echo ${server[*]}'
do
  echo "Hostname:   ${srv#*\*}" >> $outfile
  cat $logfile | grep $srv |
    while read line
    do
      shark_id='echo $line |  awk -F: '{print $2}''
      shark_total='echo $line |  awk -F: '{print $3}''
      shark_free='echo $line |  awk -F: '{print $4}''
      print "Shark $shark_id -- $shark_total MB total , $shark_free MB
free" >> $outfile
    done
    print >> $outfile
done

# Getting all sharks into an array

let i=0
cat $logfile | awk -F: '{print $2}' | sort | uniq |
   while read ids
   do

     id[$i]=$ids
     let i=${i}+1

   done

echo "##################################" >> $outfile
echo "#  TOTAL SPACE SUMMARY           #" >> $outfile
echo "##################################" >> $outfile
print >> $outfile

let total=0
for shark in 'echo ${id[*]}'
do
  cat $logfile | grep $shark | awk -F: '{print $3}' |
  while read size
  do
    let total=${total}+${size}
  done
  echo "Shark ${shark}:  $(( $total / 1024 )) GB" >> $outfile
  total=0
done
```

```
# Finally, send the report via mail
mail -s "Weekly shark space report" $mailto < $outfile
```

Adnan Akbas
Senior System Administrator
TURKCELL (Germany)

Tuning memory performance on AIX

VIRTUAL MEMORY CONCEPTS

The Virtual Memory Manager (VMM) is the part of the AIX operating system that is responsible for servicing memory requests from the operating system and users' applications. Virtual memory segments are divided into units called pages; each page is either located in real physical memory (RAM) or stored on disk until it is needed. Similarly to other modern operating systems, AIX utilizes virtual memory to address more memory than is physically available in the system. The management of memory pages in RAM or on disk is handled by the VMM.

REAL MEMORY MANAGEMENT

In AIX, virtual memory segments are partitioned into 4096-byte units called pages. Some systems also support a larger page size, typically accessed only through the **shmat** system call. Similarly, real memory is divided into 4096-byte page frames. The VMM has two major functions:

- Manage the allocation of page frames.
- Resolve references to virtual memory pages that are not currently in RAM (stored in paging space) or do not yet exist.

In order to fulfil these functions, the VMM maintains a free list of available page frames. The VMM also uses a page-replacement algorithm to determine which virtual memory pages currently in RAM will have their page frames reassigned to the free list. The page-replacement algorithm takes into account the existence of persistent versus working segments, repaging, and VMM thresholds, which can be set by the system administrator.

FREE LIST

The VMM maintains a list of free (unallocated) page frames that it uses to satisfy page faults. Generally AIX tries to use all of RAM all of the time, except for a small amount that it maintains on the free list. To maintain this small amount of unallocated pages, the VMM uses page outs and page steals to free up space and reassign those page frames to the free list. The virtual memory pages whose page frames are to be reassigned are selected using the VMM's page-replacement algorithm.

PERSISTENT OR WORKING MEMORY SEGMENTS

AIX distinguishes between different types of memory segment. To understand the VMM, it is important to understand the difference between working and persistent segments. A persistent segment has a permanent storage location on disk. Files containing data or executable programs are mapped to persistent segments. When a JFS or JFS2 file is opened and accessed, the file data is copied into RAM. VMM parameters control when physical memory frames allocated to persistent pages should be overwritten and used to store other data.

Working segments are transitory and exist only during their use by a process. Working segments have no permanent disk storage location. Process stack and data regions are mapped to working segments and shared library text segments. Pages of working segments must also occupy disk storage locations when they cannot be kept in real memory. The disk paging space is used for this purpose. When a program exits, all its working pages are immediately placed back on the free list.

COMPUTATIONAL VERSUS FILE MEMORY

Computational memory, also known as computational pages, consists of the pages that belong to working storage segments or program text (executable files) segments. File memory (or file pages) consists of the remaining pages. These are usually pages from permanent data files in persistent storage.

WORKING SEGMENT AND PAGING SPACE

Working pages in RAM that can be modified and paged out are assigned a corresponding slot in paging space. The allocated paging space is used only if the page needs to be paged out. However, an allocated page in paging space cannot be used by another page. It remains reserved for a particular page for as long as that page exists in virtual memory. Because persistent pages are paged out to the same location on disk from which they came, paging space does not need to be allocated for persistent pages residing in RAM.

The VMM has three page space allocation policies:

- Late Page Space Allocation (LPSA)
- Early Page Space Allocation (EPSA)
- Deferred Page Space Allocation (DPSA)

We will discuss the differences between them later in this article.

VMM MEMORY LOAD CONTROL FACILITY

When a process references a virtual memory page that is on disk, because it either has been paged out or has never been read in, the referenced page must be paged in, and this might cause one or more pages to be paged out if the number of available (free) page frames is low. The VMM attempts to steal page frames that have not been referenced recently and, therefore, are not likely to be referenced in the near future, using a page-replacement algorithm.

A successful page replacement keeps the memory pages of all currently-active processes in RAM, while the memory pages of inactive processes are paged out. However, when RAM is over-committed, it becomes difficult to choose pages for page out because they will probably be referenced in the near future by currently running processes. The result is pages that are likely to be referenced soon might still get paged out and then paged in again when actually referenced. When RAM is over-committed, continuous paging in and paging out, called thrashing, can occur. When a system is thrashing, the system spends most of its time paging in and paging out instead of executing useful instructions, and none of the active processes makes any significant progress. The VMM has a memory load control algorithm that detects when the system is thrashing and then attempts to correct the condition.

VIRTUAL MEMORY MONITORING TOOLS

The primary memory monitoring tools are **vmstat**, **ps**, and **svmon**.

The vmstat command

The **vmstat** command summarizes the total active virtual memory used by all the processes in the system, as well as the number of real memory page frames on the free list. Active virtual memory (the *avm* column of the **vmstat** report) is defined as the number of virtual memory working segment pages that have actually been touched. To determine the amount of virtual memory accessed by the system, multiply the value in the *avm* column by 4,096. This number can be larger than the number of real page frames in the machine because some of the active virtual memory pages may have been written out to paging space.

When determining whether a system might be short on memory or if some memory tuning needs to be done, run the **vmstat** command over a set interval and examine the *pi* and *po* columns on the resulting report. These columns indicate the number of paging space page-ins per second and the number of paging space page-outs per second, respectively. If the values are constantly non-zero, there might be a memory bottleneck. Having occasional non-zero values is not a concern, because paging is the main activity of virtual memory. Run the **vmstat** command with the **–s** flag to observe summary statistics of paging and system events since system initialization. If the recently added **–I** flag is specified, physical I/O queue information (*p*) as well as *fi* (file page-ins per second) and *fo* (file page-outs per second) statistics are listed.

The following are a few abridged samples of **vmstat** usage:

```
# vmstat 5 5
kthr     memory             page              faults       cpu
---- ---------- ----------------------- ----------- -----------
r  b   avm   fre      re  pi  po  fr   sr  cy  in  sy   cs   us  sy  id  wa
1  1 1444137 6743241  0   0   0   0    0   0   997 6089 1156  1   1 98  1
0  0 1445119 6739821  0   0   0   0    0   0 2445 10792 6410 1   3 96  0
3  0 1445140 6730397  0   0   0   0    0   0 6561 33460 24121 4 17 78 0
2  1 1445150 6730347  0   0   0   0    0   0 1275 84809 1403  3   4 88 5
0  0 1445492 6729994  0   0   0   0    0   0 988    4747 1022  1   2 97 0
# vmstat -I 5 5
  kthr      memory             page              faults       cpu
-------- ---------- ----------------------- ----------- ----------
r  b  p   avm   fre      fi  fo  pi   po   fr sr  in  sy     cs us sy id wa
1  1  0 1446320 6632273 6  30   0    0    0  0 1024 6225 1261  1  1 98  1
1  0  0 1446329 6620125 0 2452  0    0    0  0 8031 33061 28065 1 5 95  0
2  0  0 1446364 6607257 3 2550  0    0    0  0 8419 33573 29503 1 5 94  0
9  1  0 1446267 6597284 3 2080  0    0    0  0 6830 34832 24290 5 13 80 3
1  0  0 1445354 6590870 0 1512  0    0    0  0 5302 102199 17389 3  7 90 0
# vmstat -s
  kthr      memory             page              faults       cpu
-------- --------- ----------------------- ----------- ----------
r  b  p   avm   fre       fi  fo   pi  po   fr sr in   sy  cs    us  sy id wa
1  1  0 1446320 6632273   6  30   0  0    0  0 1024 6225 1261    1  1 98  1
1  0  0 1446329 6620125   0 2452  0  0    0  0 8031 33061 28065 1  5 95  0
2  0  0 1446364 6607257   3 2550  0  0    0  0 8419 33573 29503 1  5 94  0
9  1  0 1446267 6597284   3 2080  0  0    0  0 6830 34832 24290 5 13 80  3
1  0  0 1445354 6590870   0 1512  0  0    0  0 5302 102199 17389 3 7 90  0
```

The ps command

The **ps** command gives useful information on memory usage. The most useful fields to watch are:

- SIZE – the virtual size of the data section of the process in 1KB units.
- RSS – the real-memory size of the process in 1KB units.
- %MEM – the percentage of real memory used by this process.

The SIZE column

The **v** flag, with the **ps** command, generates the *SIZE* column. This is the virtual size (in paging space) in kilobytes of the data section of the process (displayed as SZ by other flags). This number is equal to the number of working segment pages of the processes that have been touched multiplied by 4. If some working segment pages are currently paged out, this number is larger than the amount of real memory being used. SIZE includes pages in the private segment and the shared library data segment of the process, as shown in the following example:

```
# ps avw |grep -v PID| sort +5 -r |head -n 5
 1155158  pts/0 A     2:10  596 179316 179248    xx    38    52  1.8
2.0 /usr/j
ava14/jre/bin/java -Xms128m -Xmx256m -DAMC=AMC -DAMC
  757956  pts/1 A     0:00   10  1320  1272    xx   188   228  0.0  0.0
-ksh
  356528   lft0 A     0:00   16  1068   852    xx    40    60  0.0  0.0
/usr/sbi
n/getty /dev/console
 1568960  pts/1 A     0:00    0   636   700    xx    52    64  0.0  0.0
ps avw
 1093706  pts/0 A     0:00    9   616   844    xx   188   228  0.0  0.0
-ksh
```

The RSS column

The **v** flag also produces the *RSS* column, as seen in the previous example. This is the real memory (resident set) size, in kilobytes, of the process. This number is equal to the sum of the number of working segment and code segment pages in memory multiplied by 4. Remember that code segment pages are shared among all the currently running instances of the program.

If you want to sort to the sixth column, you will get the output ordered using the RSS column, as shown in the following example:

```
# ps avw|grep -v PID| sort +6 -r |head -n 5
827538  pts/Ø A     Ø:ØØ    1  2596  3884    xx  9Ø26  1288  Ø.Ø  Ø.Ø
perl -d .
/testperl abp v6ØØ
 946416  pts/1 A     Ø:Ø1    Ø  2644  3148    xx   416   5Ø4  Ø.Ø  Ø.Ø -
tcsh
 868482  pts/Ø A     Ø:Ø1    Ø  2288  2792    xx   416   5Ø4  Ø.Ø  Ø.Ø -
tcsh
 8Ø7154  pts/2 A     Ø:ØØ    8  132Ø  126Ø    xx   189   216  Ø.Ø  Ø.Ø -
ksh
 2ØØ858      Ø A     Ø:ØØ    9  1Ø88   872    xx    4Ø    6Ø  Ø.Ø  Ø.Ø /
usr/sbin/getty /dev/console
```

The %MEM column

The *%MEM* column is generated by the **u** and **v** flags. This is calculated as the sum of the number of working segment and code segment pages in memory multiplied by 4 (that is, the RSS value), divided by the size of the real memory of the machine in KB, multiplied by 100, rounded to the nearest full percentage point. This value attempts to convey the percentage of real memory being used by the process. Unfortunately, like RSS, it tends to exaggerate the cost of a process that is sharing program text with other processes. Further, the rounding to the nearest percentage point causes all of the processes in the system that have RSS values under 0.005 times real memory size to have a %MEM of 0.0. For example:

```
# ps auw |head -n 1; ps au |egrep -v "RSS"|sort +3 -r |head -n 5
  942178      - A     Ø:42  393 52764 528ØØ    xx    39    56  Ø.3  1.Ø
/usr/ja
  528386      - A     Ø:ØØ  112  9232 2818Ø    xx 486Ø2 18948  Ø.Ø  1.Ø
ora_sØØ
  5Ø4Ø56      - A     Ø:ØØ  125  9Ø52 28ØØØ    xx 486Ø2 18948  Ø.Ø  1.Ø
ora_pmo
  262284      - A     Ø:ØØ  234  9232 27836    xx 45224 186Ø4  Ø.Ø  1.Ø
```

```
ora_sØØ
  532484      - A     Ø:ØØ    5  8832 2778Ø    xx 486Ø2 18948  Ø.Ø  1.Ø
ora_sØØ
  536582      - A     Ø:ØØ    Ø  882Ø 27768    xx 486Ø2 18948  Ø.Ø  1.Ø
ora_sØØ
```

You can combine all these columns into one output by using the **gv** flags. For example:

```
# ps gvw|head -n 1; ps gv|egrep -v "RSS" | sort +6b -7 -n -r |head -n 5
     PID     TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM
COMMAND
  942178      - A     Ø:44  393 52764 528ØØ    xx    39    56  Ø.3  1.Ø
/usr/ja
  528386      - A     Ø:ØØ  112  9232 2818Ø    xx 486Ø2 18948  Ø.Ø  1.Ø
ora_sØØ
  5Ø4Ø56      - A     Ø:ØØ  125  9Ø52 28ØØØ    xx 486Ø2 18948  Ø.Ø  1.Ø
ora_pmo
  262284      - A     Ø:ØØ  234  9232 27836    xx 45224 186Ø4  Ø.Ø  1.Ø
ora_sØØ
  532484      - A     Ø:ØØ    5  8832 2778Ø    xx 486Ø2 18948  Ø.Ø  1.Ø
ora_sØØ
```

The columns from the previous output that are described in the following sections are also of interest.

The PGIN column

The *PGIN* column shows the number of page-ins caused by page faults. Since all I/O is classified as page faults, this is basically a measure of I/O volume.

The TSIZ column

The *TSIZ* column shows the size of text (shared program) image. This is the size of the text section of the executable file. Pages of the text section of the executable program are brought into memory only when they are touched, that is, branched to or loaded from.

This number represents only an upper bound on the amount of text that could be loaded. The TSIZ value does not reflect actual memory usage.

The TRS column

The *TRS* column shows the size of the resident set (real memory) of text. This is the number of code segment pages multiplied by 4. This number exaggerates the memory usage of programs that have multiple instances running.

The svmon command

The **svmon** command provides a more in-depth analysis of memory usage. It is more informative, but also more intrusive, than the **vmstat** and **ps** commands.

The **svmon** command captures a snapshot of the current state of memory. There are some significant changes in the flags and in the output from the **svmon** command between AIX Version 4.3.2 and AIX Version 4.3.3.

You can use four different reports to analyse the displayed information:

- Global (-G) – displays statistics describing the real memory and paging space in use for the whole system.
- Process (-P) – displays memory usage statistics for active processes.
- Segment (-S) – displays memory usage for a specified number of segments, or the top 10 highest memory usage processes, in descending order.
- Detailed segment (-D) – displays detailed information on specified segments.

Additional reports are available in AIX Version 4.3.3 and later, as follows:

- User (-U) – displays memory usage statistics for the specified login names. If no list of login names is supplied, memory usage statistics display all defined login names.
- Command (-C) – displays memory usage statistics for the processes specified by the command name.

- Workload management class (-W) – displays memory usage statistics for the specified workload management classes. If no classes are supplied, memory usage statistics display all defined classes.

To support 64-bit applications, the output format of the **svmon** command was modified in AIX Version 4.3.3 and later. Additional reports are available in operating system versions later than AIX Version 4.3.3, as follows:

- Frame (-F) – displays information about frames. When no frame number is specified, the percentage of used memory is reported. When a frame number is specified, information about that frame is reported.
- Tier (-T) – displays information about tiers, such as the tier number, the superclass name when the **-a** flag is used, and the total number of pages in real memory from segments belonging to the tier.

VIRTUAL MEMORY TUNING TOOLS

The command **vmtune**, which was the primary tool for system tuning tasks under AIX 4.3 and AIX 5.1, has been replaced under AIX 5.2 by the **vmo** command.

The main parameters to be controlled by these commands in order to tune the AIX VMM system are minperm, maxperm, and maxclient.

VMM parameters minperm and maxperm specify memory thresholds for the page-replacement algorithm in order to control JFS file memory cacheing:

- If the percentage of RAM occupied by file pages rises above maxperm, page replacement steals only file pages.
- If the percentage of RAM occupied by file pages falls below minperm, page replacement steals both file and computational pages.
- If the percentage of RAM occupied by file pages is between

minperm and maxperm, page replacement will steal only file pages unless the file repaging rate is higher than the computational repaging rate.

The parameter strict_maxperm is used to place a hard limit on how much RAM is used as a persistent file cache.

The VMM parameter maxclient specifies memory thresholds for the page-replacement algorithm in order to control JFS2 file memory cacheing. JFS2 pages are allocated from the client segment. The upper limit for client page allocation is based on the value of the maxclient parameter. Once the number of client pages in memory reaches maxclient, page replacement is started on the client pages. JFS2 will compete for client pages with other client segment users such as NFS or compressed file systems. To reduce the likelihood of replacing working storage pages because of a large amount of JFS pages in RAM, reduce the value of the maxclient parameter. For instance, to limit maxclient pages to 50% of cache, execute:

```
vmo -p -o maxclient%=50
```

TUNING A PAGE SPACE ALLOCATION METHOD

Late page space allocation

The late paging space allocation policy means that paging space disk blocks are not allocated until the corresponding pages in RAM are touched. This policy was a default prior to AIX 4.3.2. Using this policy provides better system performance while preventing processes from unnecessarily using too much paging space. However, there is no guarantee that a process will always have sufficient paging space available if it needs to page out because some other process can start later and consume all the paging space.

Early page space allocation

Early page space allocation means that paging space disk blocks are reserved or assigned to a process as soon as memory is requested. The early policy can be turned on by setting the PSALLOC environmental variable to the value *early*. This can be done from within the process or at the command line (PSALLOC=early command). When the process uses the malloc() subroutine to allocate memory, this memory will now have paging-space disk blocks reserved for this process, ie they are reserved for this process so that there is a guarantee, should the process need to page out, that there will always be paging space slots available for it. If you are using an early policy and if CPU saving is a concern, you may want to set another environment variable called NODISCLAIM=true so that each free() subroutine call does not also result in a disclaim() system call.

Deferred page space allocation

On some systems, page space may not ever be needed – even if all the pages accessed have been touched. This is most common on systems with very large amounts of RAM. The deferred page space allocation policy, introduced with AIX 4.3.2.6, will further delay the allocation of paging space until it is necessary to page out the page – which results in no wasted paging space allocation. This can save huge amounts of paging space. This policy, however, can cause over commitment of paging space in cases where more virtual memory than available RAM is accessed.

Choosing between LPSA and DPSA

Running the **vmtune** command with the **-d** option enables the turning on or off of deferred page space allocation in order to preserve the late page space allocation policy. A value of 1 indicates that DPSA should be on, and a value of 0 indicates that DPSA should be off. If you choose to turn off DPSA, make sure that the kernel level is AIX 4.3.2.6 or higher. On AIX 5.2 the following command will turn off/on DPSA:

```
vmo -p -o defps=<0 or 1>
```

Watching paging space and virtual memory

The **vmstat** command (*avm* column), **ps** command (*SIZE*, *SZ*), and other utilities report the amount of virtual memory actually accessed because, with DPSA, the paging space may not get touched. The **svmon** command (up to AIX 4.3.2) shows the amount of paging space being used, so this value may be much smaller than the *avm* value of the **vmstat** command.

It is safer to use the **lsps -s** command than the **lsps -a** command to look at available paging space because the command **lsps -a** shows only paging space that is actually being used. But the command **lsps -s** will include paging space being used along with paging space that was reserved using the EPSA policy.

Utilization of shared memory to lower paging space usage

Processes can explicitly map files directly into memory, which avoids buffering and the system call overhead caused by it. System calls shmat(0) and mmap() can be used to perform this. Alternatively, shared memory segments can be used by multiple processes/threads to share data. Typical use is by database applications to implement a large database buffer cache. Each shared memory segment consumes 256MB of RAM. Before AIX 4.2.1, only 10 regions could be attached. With 4.2.1 and later releases, 11 regions can be attached (many more with 64-bit applications starting with AIX 4.3.0).

Extended shared memory provides for more granular shared memory regions. Its usage is turned on by setting the environmental variable EXTSHM=ON. The size of a shared memory region can be from 1 byte to 256MB, but the address space consumption will be rounded up to the next page (4096) boundary. This method essentially removes the limitation of 11 shared memory regions.

REFERENCES

1 *IBM Certification Study Guide – pSeries AIX System Support*, IBM Corporation, SG24-6185.

2 *AIX 5L Performance Tools Handbook*, SG24-6039-01.

Alex Polyak
System Engineer
APS (Israel)

Up and running with NTP

NTP, the Network Time Protocol, allows different hosts to synchronize their (internal) time clocks across networks, to an internal or external time server. The service xntpd uses NTP to synchronize the computers' time. In a typical NTP set-up, an internal server will synchronize against a stratum 1 (primary) or stratum 2 (secondary) time server; other internal hosts will then synchronize off this host. Where there is a large network of hosts, this workload would be spread to other internal time servers. This will usually be based on geographical location or subnets, creating a pyramid hierarchy set-up.

To poll a stratum server, which is usually free, all that is required is that you e-mail the host administrator with a courtesy e-mail informing them that you wish to use their resources. Make sure that your firewall allows the NTP protocol through – which is UDP at port 123. To locate a stratum time server near to you, check out: http://www.boulder.nist.gov/timefreq.

It may be the case that because of internal policies, accessing external time servers is not an option. This is only an inconvenience, not a show stopper. You can have an internal server as the designated time server, and the rest of the internal hosts can poll the time from that one server. Remember, however, that if you have many hosts, it is best to split them up. A good rule-of-thumb is to have no more than 15 servers polling an internal time server.

SETTING UP NTP ON THE SERVER

First make sure the xntpd server is not running:

```
# stopsrc -s xntpd
```

The file */etc/ntp.conf* holds the key information on xntpd. If the server will be polling from an external time server, you need to put the DNS entry for this machine in here. A typical file could look like:

```
server time.nist.gov   # external
server swisstime.ethz.ch   # external

driftfile /etc/ntp.drift
tracefile /etc/ntp.trace
```

The entry server denotes what follows – it will be a time server that this host is to time synchronize against; in this case it is the time server host (time.nist.gov). Notice that there is another server entry for another time server. Although not required, it allows for redundancy in case the first time server goes off-line for a long period. The xntpd will then synchronize against this server. When xntpd is initially started, it will poll the time difference between the local host and the time server as denoted by the server entry. How much difference there is in the time frequency or drift is stored in the file */etc/ntp.drift*. Xntpd uses this stored value to gradually reduce the drift in time ticks on the internal host. The trace file is used by xntpd when tracing utilities are invoked, like the utility **ntptrace**.

The next task is to start the xntpd service:

```
# startsrc -s xntpd
```

Using **ntpq** or **xntpdc**, which can be invoked interactively or via the command line, one can see whether a server is synchronized, though be aware it may take a few hours to fully synchronize against a stratum time server. Eventually you will see a ‘*’ or a ‘+’ against the time server. The ‘*’ indicates that the server is synchronized, a ‘+’ denotes that it is actively seeking to synchronize:

```
# ntpq -p
 remote      refid          st t  when poll reach delay  offset  disp
===================================================================
* time.nist  192.43.244.18   1 u  130 400   377   35.24  -1.625  7.16
+ swisstime  129.132.2.21    1 u   80 1024 377   39.52  -6.794  3.43
```

Looking at the above output from **ntpq**, we can see that the local host is polling the server time.nist every 400 seconds, but for swisstime, it is every 1024 seconds. The local host is 1.625 seconds behind the clock on the server time.nist. Also notice that both of the servers are (*st*) stratum 1 servers.

Be sure to check out */var/adm/messages*; all loss and re-synchronized events of the time server will be logged there.

If for some reason you are not allowed to use an external time server, then the next best option is to set up an internal master time server. The configuration for this is as follows:

```
server 127.127.1.0  #local stratum host clock
fudge 127.127.1.0 stratum 14
```

The server address of 127.127.1.0, indicates to xntpd that this is on the local host and it will treat this as a reference clock. Because the internal clock will probably not be very accurate, the stratum level is reduced to 14. The smaller the stratum’s number, the higher and more accurate it is in the xntpd hierarchy; the larger the number, the less accurate it is for time synchronization.

One can also use a router to poll the time, because most have xntpd built in. Generally these routers get the time externally. Be sure to check with the network administrator that the router is set up for xntpd. The router will undoubtedly be broadcasting this UDP information. Be sure to have the following additional entry after the router’s server address/FQDN has been added:

```
Broadcastclient
```

SETTING UP NTP ON THE CLIENTS

Once the internal server is synchronized, the next task is to set up the remaining clients within the network to poll the internal time server.

Be sure to first make sure the xntpd service is not running:

```
# stopsrc -s xntpd
```

On each client host put an entry in its *ntpd.conf* that points to the internal time server. For example, assume the local (master) time server is 192.168.1.10, then the following entries will need to be in each *ntp.conf* file on each client:

```
Server 192.168.1.10 # internal ukaix08
driftfile /etc/ntp.drift
tracefile /etc/ntp.trace
```

Before starting the xntpd service you will need to get the current time synchronized using the **ntpdate** command. First check that **ntpd** can communicate with the host, by using **ntpdate** in debug mode; no changes will take effect.

```
# ntpdate -d 192.168.1.10
25 Mar 11:48:55 ntpdate[1576960]: 3.4y
receive(192.168.1.10)
transmit(192.168.1.10)
< more output …… . . >

stratum 14, precision -17, leap 00, trust 000
```

Next, do the actual date synchronization from the client:

```
# ntpdate 192.168.1.10
25 Mar 12:00:39 ntpdate[88388]: adjust time server 192.168.1.10 offset
0.002747sec
```

If you get a *port in use* message, then you are most probably still running xntpd. Stop it and re-run the **ntpdate** command.

When **ntpdate** has completed, restart the service xntpd:

```
# startsrc -s xntpd
```

Using the **ntpq** or **xntpdc** utility check that the client is polling from the internal timeserver. The synchronization process should not take more than a few minutes because the **ntpdate** command has just been used:

```
# ntpq -p
remote              refid        st t when poll reach    delay    offset     disp
==============================================================================
*ukaix08            LOCAL(0)     14 u   29   64   377     0.34     0.000     0.05
```

The utility **ntptrace** can be used to trace the source of your timeserver. Simply enter **ntptrace** on the command line and it will display information about the source of your time server and the distance. It is a very good tool if you find that the local host's time drift is getting greater.

A good port of call for any problems is the */var/adm/messages* file. The following segment shows that the service was stopped and restarted, with the client able to re-synchronize successfully with the host 192.168.1.10:

```
Mar 25 12:00:23 ukaix12 xntpd[9916]: SRC stop issued.
Mar 25 12:00:23 ukaix12 xntpd[9916]: exiting.
Mar 25 12:01:14 ukaix12 xntpd[87984]: 3.4y
Mar 25 12:01:14 ukaix12 xntpd[87984]: tickadj = 1000, tick = 10000,
tvu_maxslew = 99000
Mar 25 12:01:14 ukaix12 xntpd[87984]: precision = 10 usec
Mar 25 12:05:31 ukaix12 xntpd[87984]: synchronized to 192.168.1.10,
stratum=14
```

Xntpd is very easy to configure, and is very low maintenance once up and running with a basic configuration. Xntpd can also be used with authentication and access control options, to properly secure the service.

David Tansley
Global Production Support (IBM p-series) (UK)

Contributing to *AIX Update*

Why not share your expertise and earn money at the same time? *AIX Update* is looking for shell scripts, program code, JavaScript, etc, that experienced users of AIX have written to make their life, or the lives of their users, easier. We are also looking for explanatory articles, and hints and tips, from experienced users.

We will publish your article (after vetting by our expert panel) and send you a cheque, as payment, and two copies of the issue containing the article. Articles can be of any length and should be e-mailed to the editor, Trevor Eddolls, at trevore@xephon.com.

A free copy of our *Notes for Contributors*, which includes information about payment rates, is available from our Web site at www.xephon.com/nfc.

IBM has launched the eServer i5, which is powered by the Power5 microprocessor. An i5 was known as iSeries, and before that, as an AS/400. The Power5 chip offers simultaneous multithreading (SMT), which effectively transforms a single processor core into two logical processors. The Power5 has two processor cores, and therefore is able run four application threads simultaneously.

So, the IBM eServer i5 can integrate and run multiple operating systems simultaneously. And the reason it is relevant is because one of those operating systems is AIX 5L, the others are i5/OS, Linux, and Windows.

For further information contact your local IBM representative.

* * *

Citadel Security Software has announced Hercules 3.0, its vulnerability management solution.

Running on AIX as well as HP-UX and Mac OS X, the product offers a host-based quarantine remediation solution, ConnectGuard. ConnectGuard quarantines all traffic from remote and local machines reconnecting to the network, checks for security policy compliance, and performs remediations on out-of-compliance machines before they are allowed to connect.

Laptops that spend any time outside the corporate firewall pose a threat when they connect to the network behind the corporate firewall. By scanning a system for vulnerabilities as it is connecting to the network and blocking the connection while vulnerability mitigation is facilitated, Citadel helps overcome the problem.

For further information contact:
Citadel Security Software, 8750 N Central Expy, Suite 100, Dallas, TX 75231, USA.
Tel: (214) 520 9292.
URL: http://www.mendocinosoft.com/pages/products.htm.

* * *

Triversity has announced that Transactionware Enterprise, a J2EE POS solution, now supports core products of Microsoft's .NET platform. Transactionware Enterprise runs on AIX, Windows, Linux, and other operating systems, and Transactionware Enterprise extends its versatility with the ability to support IBM 4690 clients and other non-Java environments.

Transactionware Enterprise's J2EE POS is designed to work in harmony with and complement .NET applications and services running on the Windows XP and Windows Server family of operating systems. Employing core .NET products such as SQL Server 2000 and Active Directory, and leveraging technology standards like XML and Web services, Transactionware Enterprise applications such as POS and mobile POS may be deployed as part of a retail organization's larger .NET initiative.

For further information contact:
Triversity, 3550 Victoria Park Avenue, Suite 400, Toronto, Ontario, M2H 2N5 Canada.
Tel: (416) 791 7100.
URL: http://www.triversity.com/newsandevents/release_040112a.html.