99

AIX update

January 2004

In this issue

AIX Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38342
From USA: 01144 1635 38342
E-mail: trevore@xephon.com

North American office

Xephon
PO Box 350100
Westminster, CO 80035-0100
USA
Telephone: 303 410 9344

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 1999 issue, are available separately to subscribers for £16.00 ($24.00) 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.

Editor

Trevor Eddolls

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.

Contributions

When Xephon is given copyright, articles published in *AIX Update* are paid for at the rate of £100 ($160) per 1000 words and £50 ($80) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £20 ($32) 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.

Printed in England.

Automatic expanding filesystems (jfs and jfs2)

INTRODUCTION

Sites tend to buy gigabytes of storage when disks are filled with data. Bytes are cheap to buy, and users don't mind if they use 10KB or 1GB. They won't clean it up and they need it all online. Oracle, for example, copes with this by having databases with self-expanding files.

Administrators are the people who maintain disk space and they have to identify any problems. This can take a lot of time, and it is always too late to ease the pain of users who can't work any more because of the problem.

The administrator can do a few things to make space available in the filesystem, including:

- Removing or zeroing files – this is the easiest way to create free space. Sometimes, however, the freed space in the filesystem is still not enough.
- Expanding the filesystem, if possible. Sometimes the volumegroup has to expand to give the filesystems space to grow.
- Create another filesystem, move files into it, and/or add/move datafile(s) to a database.

We have agents running in the system, which notice whether a filesystem is (eg) 85% full. We can get an e-mail saying, 'filesystem A is 85% full'. Now we don't have to inspect each filesystem every day, but we do have to expand them by ourself. If the users are complaining at 23:00 or batches are experiencing errors because the filesystems are completely filled, I may not be happy, but I can solve the problem.

I wrote a script that will expand the filesystem (jfs and jfs2) in batch. This script can be executed every time a filesystem reaches the maximum percentage full.

The way to make it as fast as possible is to create volumegroups with a pp-size corresponding to the filesystem size. This works because a large filesystem of 50GB with a pp-size of 4MB will need a lot of space to expand into. It would be better to place it in a volumegroup with a pp-size of 128MB or more.

The agent does not work on AIX 5. To make it work, there is a script called check_fsvg. This checks the filesystems that have to be checked. If a filesystem is ‘overloaded’ and needs expanding, it calls bigger_jfs, which releases any available free space in the volumegroup(s) where the filesystem resides. We need to check the free space to ensure that the filesystems can grow as much as they need to. Once a day the administrator gets a single message per volumegroup if the free space in the volumegroup is too small. In the script the amount of free space, eg 10%, can be set. This is for all volumegroups.

This script runs every 30 minutes. This is done by the crontab of a user who is allowed to expand filesystems.

BIGGER_JFS SCRIPT

The script bigger_jfs is called by check_fsvg to expand the filesystem. The script will check the named filesystem(s) and the volumegroup(s) where the filesystem(s) resides. If there is too little free space in a volumegroup, the administrator named at the beginning of the script will be mailed.

```
# Name        : bigger_jfs
# Last change : 27_Ø8-Ø3 Teun Post creation script
# Description : expand a filesystem
# Example     : bigger_jfs /usr 85
#-----------------------------------------------------------------------
# Begin script bigger_jfs
if [ $# -lt 1 ]
then echo "Missing parameter. "
     echo "Example call script using parameters: bigger_jfs fs 9Ø"
     echo "        fs – filesystem, needed to be expanded. (mandatory )"
     echo "        9Ø – filesystem max. % full, after expanding "
     echo "             optional, default = 8Ø%, minimum = 6%"
     exit 1
     fi
```

```
# Variables
unix_administrator="AIX_administrators@yourcompany.com"

# Get machine name, messages can appear on any machine
machine='uname -n'

# Check existence of the filesystem, if not: leave
df -I $1 1>/dev/null 2>/dev/null
if [ $? -ne 0 ]
then echo "Filesystem $1 doesn't exist, script exiting"
     echo "Exit 2 : bigger_jfs"
     exit 2
     fi

# Default perc. max filled = 80, when given as parameter: use that one
max_perc=80
if [ "$2" ]
then max_perc=$2
     fi

# Max perc. smaller than 6? Don't bother – leave with a message.
if [ $max_perc -lt 6 ]
then echo "Maximum perc filled of ${2}% less than 6%, script exiting."
     echo "Exit 3 : bigger_jfs"
     exit 3
     fi

# Get real size and perc filled, if perc less than max: exit
size='df -I $1| grep $1'
perc='echo $size | cut -f5 -d " "| tr -d "%"'
if [ $perc -lt $max_perc ]
then echo "Filesystem $1 has been filled for $perc%, max=$max_perc%,
filesystem will not been changed."
     echo "Exit 4 : bigger_jfs"
     exit 4
     fi

# Redirect output to mail
exec >/tmp/bigger_jfs.$$

# Take a snapshot of the situation before expanding filesystem for
# administration purposes
size='echo $size | cut -f2 -d " "'
echo "Filesystem $1 in $machine has got "
echo "  filled for \t\t: $perc%"
echo "  maximum filled allowed: $max_perc%, "
echo "  action \t\t\t\t\t: filesystem will be expanded."
echo "\nSize filesystem $1: size=$size blocks of 512 bytes before
expanding."
```

```
# Expand the filesystem just 1 extra lp.
# Check the filesystem is below maximum given.
while [ $max_perc -lt $perc ]
do   let size=$size+1
     # Actually change the filesystem
     chfs -a size=$size $1 >/dev/null 2>/tmp/bigger_jfs.tmp
     retc=$?
     if [ $retc -ne 0 ]
then echo "\n\nError in expanding filesystem $1 info: Returncode $retc "
          # There has been an error: full disk or max LP's of logical
          # volume reached. Mail all messages ( including error! )
          cat /tmp/bigger_jfs.tmp
          size='df -I $1| grep $1'
          size='echo $size | cut -f2 -d " "'
     echo "\nInformation of filesystem $1 after error in expanding it: "
          echo "    - filled\t: $perc%, "
          echo "    - size \t: $size blocks of 512 bytes."
          rm /tmp/bigger_jfs.tmp

          # Mail all messages
          mail -s "Error in expanding filesystem $1 in $machine"
$unix_administrator </tmp/bigger_jfs.$$
          rm /tmp/bigger_jfs.$$
          echo "Exit 5 bigger_jfs"
          exit 5
          fi

     # Refresh variables for the next loop
     size='df -I $1| grep $1'
     perc='echo $size | cut -f5 -d " "| tr -d "%"'
     size='echo $size | cut -f2 -d " "'
     done

# Filesystem is succesfully expanded: mail it!
echo "\nInformation of filesystem $1 after expanding: "
echo "    - % filled\t: $perc%, "
echo "    - size \t: $size blocks of 512 bytes."

mail -s "Filesystem $1 expanded on $machine" $unix_administrator </tmp/
bigger_jfs.$$

# Get rid of temp file and leave the script
rm /tmp/bigger_jfs.$$

# End script bigger_jfs
```

CHECK_FSVG SCRIPT

```
# Name          : check_fsvg
# Purpose       : - check if filesystems had to be expanded
#                 - check if volumegroups had to be expanded
#                 - generate mail if a volumegroup is too small
# Author        : T.W. Post
# Creation date : 31-1Ø-2ØØ3
# ============================================================
#
# Begin script check_fsvg
# Define variable volumegroup minimum free percentage (for all vgs)
vg_min_free_perc=1Ø
#
# Define variable administrator mail to get the mail generated by
# this script
administrator_mail="unix_administrator@yourcompany.com"

# in filesystem_files resides the file which contains the
#              filesystems that had to be expanded + their max % filling
filesystems_file="/user_data/trigger_file_systems"

# Use date vars to make sure volumegroup mail
#                                         is just one time send in aday
DATE='date +%d%m'
date_file=/user_data/trigger_file_systems.date

# Part one: check filesystems
# Get all filesystems in a file
df -Ik | grep -v "Mounted on" >/tmp/trig_fs.tmp

# read filesystem name and perc. filled
cat /tmp/trig_fs.tmp | while read one two three four perc fs
     do
     check='fgrep $fs $filesystems_file'
     if [ "$check" ]
     then # filesystem has to be checked
          max_perc_used='echo $check | cut -f1 -d" "'
          per='echo $perc| tr "%" " "'
          if [ "$max_perc_used" -lt "$per" ]
          then # expand filesystem
               bigger_jfs $fs $max_perc_used
               fi
          fi
     done

#
# Part Two : check volumegroups in which the checked filesystems resides
#

# Check whether the script is running the first time this day
#      yes: clean it
#      no : ok.
```

```
DATE_SEND='fgrep "$DATE" $date_file 2>/dev/null'
if [ ! "$DATE_SEND" ]
then >$date_file
     fi

# List all online volumegroups
lsvg -o | while read volume_group
     do
     # List filesystems in the volumegroup
     lsvg -l $volume_group >/tmp/trig_fs.tmp 2>/dev/null

     # Check only a volumegroup if a checked filesystem resides in it
     check=""
     cat /home/data/trigger_file_systems | while read p fs
          do
          check='fgrep $fs /tmp/trig_fs.tmp'
          if [ "$check" ]
          then break
               fi
          done
     if [ "$check" ]
     then # volumegroup has to be checked: get data
          lsvg $volume_group 2>/dev/null | grep -E "TOTAL PPs|FREE PPs"
>/tmp/trig_fs.tmp2
    cat /tmp/trig_fs.tmp2 | while read one two tree four five got_it six
               do
               if [ "$vg_size" ]
               then vg_free=$got_it
               else vg_size=$got_it
                    fi
               done
          # get the minimum free pp's in the volumegroup
          let vg_min=$vg_size*$vg_min_free_perc/100
          if [ $vg_free -lt $vg_min ]
          then # Volumegroup has to few free space
               # Mail it to the group administrator

        # Check whether mail for this volumegroup has been sent this day
          DATE_SEND='fgrep "$DATE $volume_group" $date_file 2>/dev/null'
               if [ "$DATE_SEND" ]
               then  continue
               else  echo "$DATE $volume_group" >>$date_file
                     fi

               # get formatted data
     lsvg $volume_group 2>/dev/null | grep "PP SIZE" >/tmp/trig_fs.tmp2
               read one two tree four five got_it six</tmp/trig_fs.tmp2
               exec >/tmp/trig_fs.tmp3
               echo "Volumegroup $volume_group has to be expanded (
hostname : `hostname`)"
```

```
               echo "      actual data : pp-size $got_it $six"
               echo "                    total pp's: $vg_size "
               echo "                    free  pp's: $vg_free "
               let vg_free=$vg_free*$got_it
               echo "                              = $vg_free $six"
               echo " "
               echo "      minimal free % : $vg_min_free_perc "
               echo "                     = $vg_min PP's "
               let vg_min_mb=$vg_min*$got_it
               echo "                     = $vg_min_mb $six"
               exec >/dev/tty

               # mail the message
               mail -s "Volume group $volume_group in `hostname` has to
be expanded" $administrator_mail</tmp/trig_fs.tmp3
               fi
          fi
     done

# remove work-files
rm /tmp/trig_fs.tmp
rm /tmp/trig_fs.tmp2
# >/dev/null because there isn't always mail! ( i presume )
rm /tmp/trig_fs.tmp3 2>/dev/null
# End script check_fsvg
```

EXAMPLE FILES

Example data in the file */user_data/trigger_file_systems*:

```
80 /oracle_database_fs1
90 /oracle_archive
85 /user_data
```

This means that if the filesystem */oracle_database_fs1* grows to more than 80%, the filesystem will be expanded by bigger_jfs.

/user_data/trigger_file_systems.date:

```
0611
0611 datavg01
0611 datavg03
06 November volumegroup_mail_mailed_to_administrator
```

It will be filled by the script.

Crontab entry:

```
30 * * * * /usr/local/bin/check_fsvg
```

This means that every 30 minutes every day the script */usr/local/bin/check_fsvg* will be run.

EXAMPLE OUTPUT

Example 1: output bigger_jfs

The filesystem has to be expanded, but the disk is full before the filesystem reaches the maximum percentage to be used.

```
bigger_jfs /testme 60
Mail Header:
           Error in expanding filesystem /testme in AIX_machine1
Mail body:
Filesystem /testme in AIX_machine1 has got
  filled for       : 88%
  maximum filled allowed: 60%,
  action           : filesystem will be expanded.

Size filesystem /testme: size=524288 blocks of 512 bytes before
expanding.
Error in expanding filesystem /testme info: Returncode 1
0516-404 allocp: This system cannot fulfill the allocation request.
     There are not enough free partitions or not enough physical volumes
     to keep strictness and satisfy allocation requests.  The command
     should be retried with different allocation characteristics.

Information of filesystem /testme after error in expanding it:
    - filled  : 83%,
    - size    : 557056 blocks of 512 bytes.
```

This means that you must expand the volumegroup because there is no free space available.

Example 2: output bigger_jfs

Here the filesystem has been successfully expanded:

```
bigger_jfs /testme

Mail header:
Filesystem /testme expanded on AIX_machine1

Mail body:
Filesystem /testme in AIX_machine1 has got
  filled for       : 100%
  maximum filled allowed: 80%,
  action           : filesystem will be expanded.

Size filesystem /testme: size=229376 blocks of 512 bytes before
expanding

Information of filesystem /testme after expanding:
    - % filled: 79%,
    - size    : 294912 blocks of 512 bytes.
```

Example 3 : output check_fsvg

Here there is a 2,296MB filesystem that can still expand but the volumegroup size must be increased to ensure that there will be enough free space in the future.

```
Mail header:
         Volume group datavgØ1 in AIX_machine1 has to be expanded
Mail body:
Volumegroup datavgØ1 has to be expanded ( hostname : AIX_machine1 )
      actual data : pp-size 4 megabyte(s)
                    total pp's: 4341
                    free  pp's: 574
                              = 2296 megabyte(s)

      minimal free % : 1Ø
                     = 434 PP's
                     = 1736 megabyte(s)
```

Teun Post
AIX Administrator/Oracle DBA (The Netherlands)

Manage FTP process – revisited

Last month we published the code for the mfp utility, which automates the whole process of performing FTP. Here are an example configuration record and a sample log file.

SAMPLE CONFIGURATION RECORD

```
# FTP  Configuration file
# Do not Edit by Hand
# Use mfp.ksh to add/remove Entries
# Apply Appropriate File Protection to Safeguard Password Entries
# #C# Records are commented out
# #D# Records are logically deleted
# #D##C# Records are commented out and then logically deleted
#
#
# Entry History
# -------------
# 26/02/2003 at 17:01:12 Creation
#
Remote_Host=mastst:Remote_Dir=/
tmp:Remote_File=f1.remote:Remote_Uid=zamana:Remote_Pwd=power01:Local_Dir=/
tmp:Local_File=f1.local:Action=put:Comment=Test
#
# Entry History
# -------------
# 26/02/2003 at 17:03:01 Creation
#
Remote_Host=host1:Remote_Dir=/users/export/
zamana:Remote_File=test.dat:Remote_Uid=zamana:Remote_Pwd=power01:Local_Dir=/
tmp:Local_File=f1.local:Action=put:Comment=Test
```

SAMPLE LOG FILE

```
               FTP Log File On 26/02/2003 at 17:00:01
               ======================================
mfp.sh:INFO:26/02/2003 18:12:11:Successfully started daemon script /
export/home/zamana/ftp/script/pfari.ksh

pfari.sh:INFO:26/02/2003 18:12:22:Performing FTP for configuration
record Remote_Host=mastst:Remote_Dir=/
tmp:Remote_File=f1.remote:Remote_Uid=zamana:Remote_Pwd=power01:Local_Dir=/
tmp:Local_File=f1.local:Action=put:Comment=Test
pfari.sh:INFO:26/02/2003 18:12:22:Preparing script file
pfari.sh:INFO:26/02/2003 18:12:22:Executing ftp command
pfari.sh:INFO:26/02/2003 18:12:22:Successfully performed FTP

pfari.sh:INFO:26/02/2003 18:12:22:Performing FTP for configuration
record Remote_Host=host1:Remote_Dir=/users/export/
zamana:Remote_File=test.dat:Remote_Uid=zamana:Remote_Pwd=power01:Local_Dir=/
tmp:Local_File=f1.local:Action=put:Comment=Test
pfari.sh:INFO:26/02/2003 18:12:22:Preparing script file
pfari.sh:INFO:26/02/2003 18:12:22:Executing ftp command
pfari.sh:ERROR:26/02/2003 18:12:23:Failed to perform FTP
host1: unknown host
Verbose mode on.
Not connected.
Not connected.
Local directory now /tmp
Not connected.

pfari.sh:INFO:26/02/2003 18:12:23:Sleeping for 4
```

Arif Zaman
DBA/Developer (UK)

Logical Partitioning Facility (LPAR) planning and implementation

PARTITIONING OVERVIEW

Logical partitioning enables selected IBM pSeries systems additional operation and configuration flexibility. Available for free, LPAR allows system administrators to configure a single computer into a number of independent systems. Each of these systems, also referred to as partitions, can manage private hardware resources such as processors, memory, I/O adapters, and devices.

Logical partitioning (LPAR) does not limit the number of hardware resources that are contained in a partition. A partition could have any number of the available processors assigned to it, limited only by the total number of processors. Similarly, a partition could have any amount of memory, limited only by the total amount of memory available. An I/O adapter is physically installed in one of many slots in the system. However, with LPAR, any I/O adapter in any I/O drawer can be assigned to any partition. Each partition is an independent system running, possibly, different versions or types of operating system.

Partitions, while sharing the same physical hardware, after completion of the configuration, are booted and used as independent systems.

Partition management is performed using the hardware management console (HMC). Each system that is running partitions and managed by the HMC is referred to as a managed system. A managed system is capable of being configured to use logical partitions (LPARs) or a full system partition. Each partition on a server is defined by a profile. Profiles for logical partitions are created and managed using the HMC.

If your computing needs are considered to be technical, real-time, or high-performance computing, a special type of partitioning called affinity logical partitioning is recommended – if supported by your hardware (presently models p670 and p690).

Dynamic Logical Partitioning (DLPAR) allows you to implement changes to your partitions at any time without affecting a partition’s operation.

Dynamically changing a partition enables a partition’s resources to be changed while the partition is up and running. The operating system that is running in the partition can configure and use additional hardware without being rebooted. In a DLPAR environment, the processors, memory, or input/output adapters can be added, moved, or removed after the partition is up and running.

Systems that are capable of performing dynamic logical partitioning can support the following tasks:

- Processor tasks:
  - adding processors to a partition

  - moving processors from one partition to another
  - removing processors from a partition.
- Memory tasks:
  - adding memory to a partition
  - moving memory from one partition to another
  - removing memory from a partition.
- Input/output tasks:
  - adding a PCI adapter
  - moving a PCI adapter
  - removing a PCI adapter.

An affinity logical partition is a special type of logical partition that has its processors and memory resources located physically close to one another in order to increase performance for computing-intensive workloads. Processors needed for a partition can be grouped to use the closest physical memory available. Hardware resources for affinity partitioning are defined using the HMC. When creating an affinity partition, the HMC automatically determines which processors and memory are grouped and allows you to choose which type of grouping you want. The HMC then creates a profile for each affinity partition and a system profile that contains the affinity partitions for the managed system.

Affinity partitioning is best suited for use in technical computing, real-time computing, and high-performance computing. A system that is set up to use affinity logical partitions can dynamically move I/O devices. However, in order to change the quantity of processors or memory assigned to an affinity logical partition, the partition must be rebooted

A special partition called the full system partition assigns all your managed system’s resources to one large partition. The full system partition is similar to the traditional, non-partitioned method of operating a system. Because all resources are assigned to this partition, no other partitions can be started when the full system partition is running. Likewise, the full system partition cannot be started while other partitions are running.

The HMC allows you to switch from the full system partition to logical partitions.

PARTITIONING BENEFITS

Partitioning enables more flexible computer systems deployment and operation. The ability to confine workloads to dedicated operating system instances, while sharing hardware and environmental resources such as floor space, power supply, and air conditioning, provides many opportunities that have not been available in previous generations of servers. The following are some examples of optimizations of computer operations achievable with partitioning.

Server consolidation

Disparate computer workloads running on a number of smaller servers can be consolidated to a single, larger machine sharing environmental and machine resources. Additionally, it is possible to adjust the configuration of partitions to reflect changes in the demands of the workloads.

Merge development, test and production environments

The typical life-cycle of software project development can be reflected in the partitioning of the server used. At the beginning, most of the resources can be allocated to a partition used for development purposes. When alpha and beta versions of a project have been released, part of the resources can be taken to form a test partition. With the release of the production version, most of the resources can be allocated to the partition running production, while development and testing partitions can be reduced to a minimal set of resources needed for software maintenance.

Consolidating multiple versions of the same operating system

It is possible to configure separate partitions containing different maintenance levels, and even different versions of operating systems, to enable the thorough testing of mission-critical applications before committing to adoption of the latest fixes released by the operating system vendor.

Consolidating different operating systems

At present it is possible to configure separate partitions containing installations of AIX and Linux on the same hardware, enabling the use of different operating systems on a single server.

Scalability balancing

Partitioning enables systems managers to shift computer resources according to the actual demand of the workloads without oversizing system capabilities to fit peak demands. Such adaptations can be performed automatically by the execution of scripts according to predefined schedules or dynamic load changes.

Consolidate applications requiring different time zone settings

Partitioning enables multiple regional workloads to be consolidated onto a single server. The different workloads can run in different partitions, with different operating systems, as well as with different time and date settings.

PARTITIONING PLANNING

In the following sections I will present a case study describing the implementation of a single partitioned system.

Requirements

The following are the requirements given to the system administrator.

Design a system to provide a development project with two servers. The first is to function as a development server with the following capacity – two 1.45GHz Power4 processors and 4GB of RAM. The second has to function as a test server with six 1.45GHz Power4 processors and 12GB of RAM. Both servers are to run the latest maintenance level of AIX 5.2 operating system. The system should have the ability to shift more processor and memory capacity to the test server at the expense of development server capacity.

According to the site's server configuration standards, each server has to fulfil the following additional demands:

- It must contain two system disks – one on-line, the other used for alt_disk_install back-up.
- It must contain two fibre channel adapters for redundant connection to a mass storage subsystem.
- It must contain two network adapters – one Gigabit Ethernet, the other Fast Ethernet.
- Each system must have access (not necessarily concurrently) to the following peripheral devices: CD-ROM or DVD-ROM drive used for system and software installations; 4mm DAT tape used for operating system back-ups and restores; and an ASCII or graphical console.

Proposed hardware configuration

After consultation with IBM technical personnel, the following system hardware was specified:

- One eSeries pServer Model p650.
- Eight 1.45GHz CPUs.
- 16GB of RAM.
- Four 36GB internal disks. The system has been explicitly ordered with split internal SCSI connections enabling each pair of disks to connect to separate SCSI adapters, with each adapter assigned to a different partition. One of the adapters

is a built-in, on-board adapter while the other is a separate, external adapter.

- Four fibre channel adapters.
- Two Gigabit Ethernet adapters.
- One Fast Ethernet adapter (one built-in Fast Ethernet adapter to be used as well).
- One GXT135 graphical adapter.
- One internal DVD-ROM.
- One external 4mm DAT tape.
- One external DVD-ROM.
- Two dual-ported SCSI adapters.

The total number of non-built-in adapters is ten. The basic system drawer CEC has only seven I/O slots, one of which is occupied by a special SCSI bus splitter card. Therefore an additional D10 I/O drawer had to be ordered to accommodate all adapters.

The external DVD-ROM and DAT tape are installed in a separate external I/O device drawer.

HMC controller, which is Model 365 Intel server running RedHat Linux 7.2, is another essential addition to the system.

We have chosen a compact drawer containing a keyboard and a flat screen to double as the system console and the HMC console. In order to achieve this we had to order a special KVM switch from IBM.

All the equipment has been installed in a single T00 Rack.

Actual partition set-up

This section will describe how the initial requirements have been fulfilled by allocation of particular system resources to different partitions.

Processor allocation

No special issues here. Each LPAR needs at least one processor. The development partition has been defined with one as the minimum number of processors, and two as the desired and maximal number. The testing partition has been defined with six as the minimum and desired number of processors and seven as the maximum number.

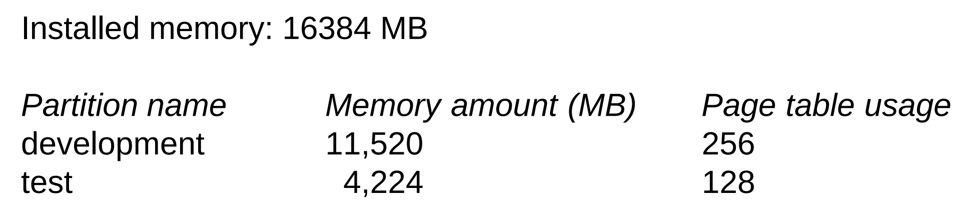
Memory allocation

When a machine is in full system partition mode (no LPARs), all the memory is dedicated to AIX 5L. When a machine is in LPAR mode, some of the memory used by AIX is relocated outside the AIX-defined memory range. For instance in the case of a single small partition (256MB), the first 256MB of memory will be allocated to the hypervisor, part of the system firmware implementing partitioning support, 256MB is allocated to Translation Control Entries (TCEs) and to hypervisor per partition page tables, and 256MB for the first page table for the first partition. TCE memory is used to translate the I/O addresses to system memory addresses. Additional small page tables for additional small partitions will fit in the page table block. Therefore, the memory allocated independently of AIX to create a single 256MB partition is 768MB (0.75GB).

With the previous memory statements in mind, the LPAR requires at least 2GB of memory for two or more LPARs.

The following rules apply only for partitions with AIX 5L or Linux (if available):

- The minimum memory for an LPAR is 256MB. Additional memory can be configured in increments of 256MB.
- The memory consumed outside AIX is from 0.75GB up to 2GB, depending on the amount of memory and the number of LPARs.
- For AIX 5L Version 5.1, the number of LPARs larger than 16GB is limited to two in a system with 64GB of installed memory, because of the memory alignment in AIX 5L Version 5.1.



Installed memory: 16384 MB

Partition name	*Memory amount (MB)*	*Page table usage*
development	11,520	256
test	4,224	128

Figure 1: Installed memory

LPARs that are larger than 16GB are aligned on a 16GB boundary. Because the hypervisor memory resides at the lower end of the memory and TCE resides at the upper end, only two 16GB boundaries are available.

- With AIX 5L Version 5.2, there are no predefined limits concerning partitions larger than 16GB, but the total amount of memory and hypervisor overhead remains a practical limit.

Note: to create LPARs running AIX 5L Version 5.2 or Linux larger than 16GB, the checkbox *Small Real Mode Address Region* must be checked (on the HMC, LPAR Profile, and Memory Options dialog). Do not select this box if you are running AIX 5L Version 5.1.

Page 16 of *Planning for Partitioned-System Operations* contains a table detailing the approximate memory overhead and approximate memory available for partitions for different combinations of total memory size, number of partitions, and versions of the operating system running in the partition.

In our two-partition 16GB RAM system, the actual size of the memory available for allocation to partitions in a 16GB RAM system with all LPARs running AIX 5.2 is between 15GB and 15.25GB.

The development partition has been defined with 2GB as the minimum size of memory and 4GB as the desired and maximum size. The testing partition has been defined with 8GB as minimum, 1GB as the desired size of memory, and 15GB as the maximum size.

An examination of the memory tab of the system properties of our system displays the table shown in Figure 1.

Our preference has been to allocate the smaller partition with all the required memory at the expense of the partition containing the greater amount of the resource.

I/O slots allocation and peripheral devices mapping

This section requires careful planning and knowledge of particular computer resources.

The I/O devices are assigned at the slot level to the LPARs, meaning an adapter installed in a specific slot can be assigned to only one LPAR. If an adapter has multiple devices, such as the 4-port Ethernet adapter or the Dual Ultra3 SCSI adapter, all devices are automatically assigned to one LPAR and cannot be shared.

The internal devices can also be assigned to LPARs, but in this case the internal connections must be taken into account. Devices connected to an internal SCSI controller must be treated as a group, as must devices containing an IDE device that share the same PCI bridge.

The internal disks, the media bays, and the external SCSI port of systems with internal disks are all driven by one SCSI chip on the I/O backplane. This chip is connected to one of the PCI-X-to-PCI-X bridges, which in terms of an LPAR is equal to a slot. Therefore, in a standard configuration, all SCSI resources in the disk and media bays, including external disks connected to the external SCSI port, must be assigned together to the same LPAR. There is no requirement to assign them to a particular LPAR – in fact, they can remain unassigned if the LPAR minimum requirements are obtained using devices attached to a SCSI adapter in the CEC or I/O drawer.

This can result in complications when an LPAR with the internal SCSI resources is active and a second LPAR needs to be installed using the internal media devices. In a standard configuration, this is not possible without shutting down the active LPAR containing all the internal SCSI devices. In this scenario, when the second LPAR is installed using all the internal SCSI devices, you must be careful not to override the disks of the first LPAR.

To avoid this problem, the best solution for providing access to CD-ROMs and DVD-RAMs for different LPARs is probably to use an externally-attached DVD-RAM (FC 7210 Model 025) with a storage device enclosure (FC 7212 Model 102). This external DVD-RAM could be connected to a PCI SCSI adapter (FC 6203), which makes it easy to move the DVD-RAM between different LPARs. This solution also provides the advantages of sharing this DVD-RAM between several servers by attaching it to SCSI adapters in different servers.

Every LPAR needs its disks for the operating system. Systems with internal disks are connected to the internal SCSI port. As described previously, all SCSI devices, including all internal disks, could be assigned only to the same LPAR unless the

Physical location	*Adapter*	*Purpose*
U0.1-P2-I1	10/100/1000 Base-TX PCI-X adapter	Network connectivity
U0.1-P2-I2	FC adapter	SAN connectivity
U0.1-P2-I3	GXT135P graphics adapter	Graphical console
U0.1-P2/Z1	Wide/Ultra-3 SCSI I/O controller internal port 1	Built-in DVD-ROM connectivity
U0.1-P2/Z2	Wide/Ultra-3 SCSI I/O controller internal port 2	Internal disks connectivity
U0.1-P2/E1	10/100 Mbps Ethernet PCI adapter II – internal	Network connectivity
U0.1-P2-I6	FC adapter	SAN connectivity
I/O Group/Group _128/U0.1-P2	ISA bus	Keyboard, mouse and serial ports connectivity

Figure 2: I/O adapters for the development partition

Physical location	*Adapter*	*Purpose*
U0.2-P1-I1	FC adapter	SAN connectivity
U0.2-P1-I2	10/100/1000 Base-TX PCI-X adapter	Network connectivity
U0.2-P1-I3	10/100 Mbps Ethernet PCI Adapter II	Network connectivity
U0.2-P1-I4/Z1	Wide/Ultra-3 SCSI I/O Controller port 1	External DVD-ROM connectivity
U0.2-P1-I4/Z2	Wide/Ultra-3 SCSI I/O Controller port 2	External tape connectivity
U0.2-P1-I6	FC Adapter	SAN connectivity
U0.1-P2-I5 (The only exception to our I/O slot assignment preference rule)	Wide/Ultra-3 SCSI I/O Controller port 2	Internal disks connectivity

Figure 3: I/O adapters for the test partition

system has the capacity to install SCSI splitters, enabling the connection of groups of internal disks to different SCSI adapters. Without such a solution being available to a particular server, external disk space is necessary in order to configure multiple LPARs. Space has to be allocated for external disk subsystems or additional I/O drawers containing disks. The external disk space must be attached with a separate adapter for each LPAR by using SCSI, SSA, or fibre-channel adapters, depending on the subsystem.

The internal serial ports, diskette drive, keyboard, and mouse are connected to an ISA bus that is finally connected to the RIO to PCI-X host bridge. Therefore, these ports and the diskette drive could only be assigned together to one LPAR, but these resources are independent of the SCSI resources.

The number of RIO cards installed has no effect on the number of LPARs supported other than the limitations related to the total number of I/O drawers supported, and the ability to meet the

LPAR minimum requirements in a particular configuration.

In addition to these limitations, the ISA I/O resources can neither be added nor removed using dynamic LPAR, including any devices sharing the same PCI-X bridge, such as serial ports, native keyboard and mouse ports, and the diskette drive.

For consistency, we have decided to stick to the following rule as much as possible: we prefer to allocate I/O adapters from a CEC drawer (U0.1) to a development partition, and I/O adapters from an external drawer (U0.2) to a test partition.

Figure 2 shows the list of I/O adapters required for the development partition.

In Figure 3 is the list of I/O adapters required for the test partition.

We have met all permanent I/O adapter assignment needs. We can transfer the slot U0.2-P1-I4/Z2 from the test partition to the development partition (and back) in order to provide access to a tape device. We can transfer the slots I/O Group/Group_128/U0.1-P2 and U0.1-P2-I3 from the development partition to the test partition in order to provide access to the console display and keyboard/mouse.

REFERENCES

1 *AIX 5L Version 5.2 AIX Installation in Partitioned Environment*, SC23-4382. IBM Corporation

2 *Effective System Management Using the IBM Hardware Management Console for pSeries*, SC24-7038, IBM Corporation

3 *Planning for Partitioned-System Operations*, SA38-0626, IBM Corporation

4 *Site and Hardware Planning Information*, SA38-0508, IBM Corporation

5 *IBM Hardware Management Console for pSeries Installation and Operations Guide*, SA38-0590, IBM Corporation

6 *Electronic Service Agent for eServer pSeries User's Guide*, LCD4-1060, IBM Corporation

7 *PCI Adapter Placement* reference, SA38-0538, IBM Corporation

Alex Polak
System Engineer
APS (Israel)

Curses programming

What is curses? Curses is a package of functions in the Unix library for use in C programs that need to manipulate screens in a non-line-oriented way. Before the advent of GUIs (Graphical User Interfaces), curses packages were used widely to create user interfaces for any interactive programs. For example, SQL*Forms, Oracle's flagship product for creating database application, used the curses package to create all its user interfaces.

Curses programming is all about creating interactive applications (reading input from a box on the terminal or outputting a piece of data in a box drawn at a certain location on the terminal, etc) using special curses functions.

The word 'curses' is also used to refer to this concept of interactive programming under Unix.

EXAMPLE SOURCE CODE LISTING

```
#include  <curses.h>
#include  <time.h>
   #include  <signal.h>

/*************************************************************
*  Name     :  uaccess.c
*
*  Overview : The program draws a window on the screen and
*             accepts two separate inputs, userid  and
*               password, from two separate fields.
*  Notes    : 1. To compile this program  do as follows:
*               cc -o password  password.c /usr/lib/libcurses.a
*
*************************************************************/

/*
#
#  Function Prototypes
#
*/
int  main ( void );
void MakeField  ( WINDOW *ptr,  short ycor, short xcor, short field_length ) ;
void ClearField ( WINDOW *ptr,  short ycor, short xcor, short field_length );
void ClearMessage   ( WINDOW *ptr ) ;
void DisplayMessage ( WINDOW *ptr, char  *msg ) ;
void WriteText   ( WINDOW  *ptr, short ycor, short xcor, char *text ) ;
void GetUserid   (void) ;
void GetPassword (void);
void DisplayTime (WINDOW *ptr);
void HandleInterrupt (int);
/*
#
# declare global  variables
#
*/
WINDOW   *wptr;         /*  pointer to the window structure */
WINDOW   *subwptr;      /*  pointer to the window structure */
char  dummy[2];
char  uid[20];
char  pwd[20];

int  WINWIDTH       = 80;  /* width for main window         */
int  WINHEIGHT      = 23;  /* heigh for main window         */
int  WINXCOR        = 0;   /* x coordinate for location     */
int  WINYCOR        = 0;   /* y coordinate for location     */

int  SUBWINWIDTH    = 40;  /* width for subwindow          */
int  SUBWINHEIGHT   = 10;  /* heigh for subwindow           */
int  SUBWINXCOR     = 15;  /* x coordinate for location     */
int  SUBWINYCOR     = 5;   /* y coordinate for location     */



int  UIDFIELDXCOR   = 12;  /* x coordinate for userid field */
int  UIDFIELDYCOR   = 4;   /* y coordinate for userid field */
int  UIDFIELDLEN    = 20;  /* field length for userdid field*/
int  PWDFIELDXCOR   = 12;  /*x coordinate for password field*/
int  PWDFIELDYCOR   = 6;   /*y coordinate for password field*/
int  PWDFIELDLEN    = 20;  /*field lenght for password field*/

int  UIDTEXTXCOR    = 2;   /*x coordinate for userid boiler plate text   */
int  UIDTEXTYCOR    = 4;   /* y coordinate for userid boiler plate text   */
int  PWDTEXTXCOR    = 2;   /* x coordinate for password boiler plate text */
int  PWDTEXTYCOR    = 6;   /* y coordinate for password boiler plate text */

int  WHTEXTXCOR      = 10;  /* x coordinate for text in window heading */
int  WHTEXTYCOR      = 2;   /* y coordinate for text in window heading */

int  MSGFIELDXCOR   = 1;   /* x coordinate for message field*/
int  MSGFIELDYCOR   = 20;  /* y coordinate for message field*/
int  MSGFIELDLEN    = 77;  /* field length for message field*/
int  MSGTEXTXCOR   = 1;     /* x coordinate for message     */
int  MSGTEXTYCOR   = 20;    /* y coordinate for message     */

int  MSGACKXCOR   = 1;     /* x coordinate for message acknowledge field   */
int  MSGACKYCOR   = 21;    /* y coordinate for message acknowledge field   */
int  MSGACKLEN    = 60;    /* field length for message acknowledge field   */

int  TIMEFIELDXCOR   = 59;   /* x coordinate for time  field*/
int  TIMEFIELDYCOR   = 1;    /* y coordinate for time  field*/
int  TIMEFIELDLEN    = 19;   /* field length for time  field*/

/*************************************************************
*
*  Name     : main ()
*
*  Overview : The functions implements processing structure.
*
*  Notes    :
*************************************************************/
int  main ( void  )
{

signal (SIGINT, ( void * )HandleInterrupt  );

/*
 *  initialise the  screen
 */
initscr( );   /*  mandatory   function  call  */

/*
 *  create main  window
 */
wptr = newwin(WINHEIGHT,WINWIDTH,WINYCOR,WINXCOR);

if ( wptr == (WINDOW * ) NULL )
{
   printf("ERROR:Failed to create the window\n");
   exit (1);
}
/*
 * draw a box around the main window
 */
box(wptr,0,0 );   /*  X and Y cordinates relative to window */
/*
 * refresh screen
 */
wrefresh(wptr);
/*
 * make sub-window
 */
subwptr =
subwin(wptr,SUBWINHEIGHT,SUBWINWIDTH,SUBWINYCOR,SUBWINXCOR);
if ( subwptr == (WINDOW * ) NULL )
{
   printf("Failed to create the sub-window\n");
   exit (1);
}
/*
 * draw a box around the sub window
 */
box(subwptr,0,0 );
wrefresh(subwptr);
/*
 * write heading in the sub-window window
 */
WriteText(subwptr,WHTEXTYCOR,WHTEXTXCOR, "User Access Form" );
/*
 * make field for accepting userid
 */
MakeField(subwptr,UIDFIELDYCOR,UIDFIELDXCOR, UIDFIELDLEN);
/*
 * write boiler plate text for userid field
 */
WriteText(subwptr,UIDTEXTYCOR,UIDTEXTXCOR,"Userid");
/*
 * make field for accepting  pwd
 */
MakeField(subwptr,PWDFIELDYCOR,PWDFIELDXCOR, PWDFIELDLEN);
/*
 * write boiler plate text for password
 */
WriteText(subwptr,PWDTEXTYCOR,PWDTEXTXCOR,"Password");
/*
 * make the field for message
 */
MakeField(wptr,MSGFIELDYCOR,MSGFIELDXCOR, MSGFIELDLEN);
/*
 * make the field for date and time
 */
MakeField(wptr,TIMEFIELDYCOR,TIMEFIELDXCOR, TIMEFIELDLEN);
DisplayTime ( wptr);
/*
 * get inputs
 */
while (1)
{
GetUserid () ;
GetPassword ();

/*
 *  password must be the same as userid
 */
if  ( strcmp(uid, pwd) )
         DisplayMessage(wptr, "ERROR:Invalid password");
else
         break ;

}
/*
 *  remove  all  the window resources
 */
endwin ( );

}

/**************************************************************
*
*  Name     : WriteText ()
*
*  Overview : The functions writes text on the screen
*
*  Notes    :
**************************************************************/
void WriteText ( WINDOW  *ptr, short ycor, short xcor, char *text )
{
```

```
/*
 *  move the cursor to right location
 */
wmove(ptr,ycor,xcor);
waddstr(ptr,text);
wrefresh(ptr);


}

/*************************************************************
*
*  Name     : DisplayMessage ()
*
*  Overview : The functions displays an informatiove or error
*             message on message line
*
*  Notes    :
*************************************************************/
void DisplayMessage ( WINDOW *ptr, char  *msg )
{

 /*
  * display error/informative message
  */
 wattron ( ptr, A_REVERSE);
 WriteText(ptr,MSGTEXTYCOR,MSGTEXTXCOR,msg);
 wattroff ( ptr, A_REVERSE);
 /*
  * display acknowledgement message
  */
 WriteText(ptr,MSGACKYCOR,MSGACKXCOR,"Please acknowledge
message...");
 wgetstr(ptr,dummy);

}

/*************************************************************
*
*  Name     : ClearMessage ()
*
*  Overview : The functions clears the  message from from the
*             message line and acknowledgement message from
*             the acknowledgement message line
*  Notes    :
*************************************************************/
void ClearMessage ( WINDOW *ptr )
{

/*
```

```
 * clear message line
 */
short  i ;
short  xcor = MSGFIELDXCOR ;

wattron ( ptr, A_REVERSE);
for  ( i=Ø ;  i<MSGFIELDLEN ;  i++ )
{
  wmove(ptr,MSGFIELDYCOR, xcor);
  waddstr(ptr," ");
  xcor++ ;
  i++ ;
wrefresh(ptr);

}

/*
 * clear message  acknowledge line
 */
wattroff ( ptr, A_REVERSE);

xcor = MSGACKXCOR ;

for  ( i=Ø ;  i<MSGACKLEN ;  i++ )
{
  wmove(ptr,MSGACKYCOR, xcor);
  waddstr(ptr," ");
  xcor++ ;
  i++ ;

}

wrefresh(ptr);


}

/**************************************************************
*
*  Name     : ClearField ()
*
*  Overview : The functions clears the field
*
*  Notes    :
**************************************************************/
void ClearField ( WINDOW *ptr,  short ycor, short xcor, short field_length )
{

static short  i ;
```

```

wattron ( ptr, A_REVERSE);
for  ( i=0 ;  i<field_length ;  i++ )
{
  echo();
  wmove(ptr,ycor,xcor);
  waddstr(ptr," ");
  xcor++ ;
  wrefresh(ptr);

}


}



/*************************************************************
*
*  Name     : MakeField ()
*
*  Overview : The functions draws a field on the screen within
*              the main window
*  Notes    :
*************************************************************/
void MakeField ( WINDOW *ptr,  short ycor, short xcor, short field_length )
{
/*
 * make field
 */
short i ;
wattron ( ptr, A_REVERSE);

wmove(ptr,ycor,xcor);

for  ( i = 0; i < field_length; i ++ )
       waddstr(ptr," ");
wattroff ( ptr, A_REVERSE);
wrefresh(ptr);

}
/*************************************************************
*
*  Name     : GetUserid ()
*
*  Overview : The functions accepts input  from userid  field.
*
*  Notes    :
*************************************************************/
```

```
void GetUserid ( void )
{


int  xcor_var, xcor, ycor ;     /* absolute xcoordinate within
field  */
int last_pos = Ø ; /* last positon indicator  for field  */
int  ch ;
int i = Ø ;

while ( 1)
{
  keypad(subwptr, TRUE );
  /*
  raw();
   */
  ClearField(subwptr,UIDFIELDYCOR,UIDFIELDXCOR, UIDFIELDLEN);
  ClearMessage(wptr);
  strcpy(uid,"");
  wattron ( subwptr, A_REVERSE);
  /*
   * place the cursor on the first character position in field
   */
  wmove(subwptr,UIDFIELDYCOR,UIDFIELDXCOR);
  xcor_var =Ø;
  noecho();
  i=Ø ;
  while  (1)
   {
    /*
     * accept a character but it wouldn't be displayed
     * we need to explicitly add this character to the field
     */
     ch=wgetch(subwptr);
     if ( ch == 1Ø )
       {
        /*
         * newline entered ; assume input completed
         */
         uid[i]='\Ø';
         break ;
       }

     else if ( ch == KEY_LEFT )
      {
        /*
         * only move cursor to the left if a character has
         * been typed in the  field
         */
        if ( xcor_var > Ø )
```

```
            {
              /*
               * adjust x-coordinate
               */
              xcor_var--;
      wmove(subwptr,UIDFIELDYCOR,(UIDFIELDXCOR + xcor_var ));
              /*
               * set last position indicator to false
               */
              last_pos = 0;
             }
     }
    else if ( ch == KEY_BACKSPACE )
     {
       if ( xcor_var > 0 )
            {
              xcor_var--;
     wmove(subwptr,UIDFIELDYCOR,(UIDFIELDXCOR + xcor_var ) );
              waddch(subwptr,' ');
     wmove(subwptr,UIDFIELDYCOR,(UIDFIELDXCOR + xcor_var ) );
              wrefresh(subwptr);
              last_pos = 0;
              i-- ;
             }
     }
    else if  ( xcor_var  ==  ( UIDFIELDLEN - 1 ) )
     {
        /*
         * cursor is on last character position in the field
         */
          if ( last_pos )
            {
              /*
               * a character has already been accepted for
               * the last position in the field
               */
               beep();
            }
          else
            {
              /*
* accept this character in the last position and  display it
               */
               if ( isalnum (ch)  || isspace (ch) )
                 {
                     waddch(subwptr, ch);
                     wrefresh(subwptr);
                     uid[i]=ch ;
                     i++;
                     /*
                     * set the last position indicator to true
                       */
                      last_pos = 1;
                  }
             }
      }
     else
       {
          /*
           * check for acceptable  characters
           */
          if ( isalnum (ch)  || isspace (ch) )
           {
             waddch(subwptr, ch);
             xcorØ ;
             getsyx(ycor,xcor);
             /* uid[i]=ch ; */
             uid[xcor - 27 ]=ch ;
             xcor_var++;
             i++;
           }
       }
   }
   /*
    * check the input string
    */
    if (  strcmp(uid,"") == Ø  )
             DisplayMessage(wptr, "ERROR:Must enter user id");

    else
          break ;
}

}

/**************************************************************
*
*  Name     : GetPassword ()
*
*  Overview : The functions accepts input from password
*                field.
*  Notes    :
*************************************************************/
void GetPassword ( void )
{

int  xcor_var, xcor, ycor  ;
int last_pos = Ø ;
int   ch ;
int i = Ø ;
```

```

while ( 1)
{
  keypad(subwptr, TRUE );
  raw();
  ClearField(subwptr,PWDFIELDYCOR,PWDFIELDXCOR, PWDFIELDLEN);
  ClearMessage(wptr);
  strcpy(pwd,"");
  wattron ( subwptr, A_REVERSE);
  /*
   * place the cursor on the first character position in field
   */
  wmove(subwptr,PWDFIELDYCOR,PWDFIELDXCOR);
  xcor_var =0;
  noecho();
  i=0 ;
  while  (1)
   {
    /*
     * accept a character but it wouldn't be displayed
     */
     ch=wgetch(subwptr);
     if ( ch == 10 )
       {
        /*
         * newline entered ; assume input completed
         */
         pwd[i]='\0';
         break ;
       }

     else if ( ch == KEY_LEFT )
      {
        if ( xcor_var > 0 )
             {
               xcor_var--;
       wmove(subwptr,PWDFIELDYCOR,(PWDFIELDXCOR + xcor_var ));
               last_pos = 0;
              }
      }
     else if ( ch == KEY_BACKSPACE )
      {
        if ( xcor_var > 0 )
             {
               xcor_var--;
      wmove(subwptr,PWDFIELDYCOR,(PWDFIELDXCOR + xcor_var ) );
               waddch(subwptr,' ');
      wmove(subwptr,PWDFIELDYCOR,(PWDFIELDXCOR + xcor_var ) );
               wrefresh(subwptr);
               last_pos = 0;
```

```
            i-- ;
           }
   }
  else if  ( xcor_var  ==  ( PWDFIELDLEN - 1 ) )
   {
      /*
       * cursor is on last character position in the  field
       */
        if ( last_pos )
          {
            /*
             * a character has already been accepted for
             * the last position in the field
             */
             beep();
          }
        else
          {
            /*
* accept this character in the last position and display it
             */
             if ( isalnum (ch)  || isspace (ch) )
               {
   wmove(subwptr,PWDFIELDYCOR,(PWDFIELDXCOR + xcor_var ) );
                   wrefresh(subwptr);
                   pwd[i]=ch ;
                   i++;
                   /*
                 * set the last position indicator to true
                    */
                   last_pos = 1;
               }
          }
   }
  else
    {
       /*
        * check for acceptable  characters
        */
       if ( isalnum (ch)  || isspace (ch) )
        {
          xcor=0 ;
          getsyx(ycor,xcor);
          pwd[xcor - 27 ]=ch ;
          xcor_var++;
          i++;
   wmove(subwptr,PWDFIELDYCOR,(PWDFIELDXCOR + xcor_var ) );
        }
    }
}
```

```
   /*
    * check the input string
    */
    if (  strcmp(pwd,"") == Ø  )
            DisplayMessage(wptr, "ERROR:Must enter password");

    else
          break ;
}

}



/*************************************************************
*
*  Name     : DisplayTime ()
*
*  Overview : The function displays the data and time.
*
*  Notes    : 1. tm_s->tm_year will  return year  from 19ØØ.
*
*************************************************************/
void DisplayTime ( WINDOW *ptr )
{

time_t          time_int ;
struct tm       *tm_s ;

char            today[3Ø];

time(&time_int);
tm_s = localtime(&time_int);


sprintf (today,"%Ø2d/%Ø2d/%Ø2d  %Ø2d:%Ø2d:%Ø2d", tm_s->tm_mday,tm_s->tm_mon +1,
                                     tm_s->tm_year + 19ØØ,
                    tm_s->tm_hour,tm_s->tm_min, tm_s->tm_sec);
wmove(ptr,TIMEFIELDYCOR, TIMEFIELDXCOR);
wattron ( ptr, A_REVERSE);
waddstr(ptr,today);
wrefresh(ptr);


}
/*************************************************************
*
*  Name     : HandleInterrupt ()
*
```

```
   641    *  Overview : The function displays a message and exits.
   642    *
   643    *  Notes    :
   644    *
   645    **************************************************************/
   646    void HandleInterrupt ( int signo )
   647    {
   648
   649    DisplayMessage(wptr, "ERROR:Program interrupted; quitting
early");
   65Ø    /*
   651     *  remove  all  the window resources
   652     */
   653    endwin ( );
   654
   655    exit (1);
   656
   657    }
   658
   659
```

SOURCE CODE ANALYSIS

1-3 Header files. Curses.h is relevant to all curses programming.

21-30 Function prototypes.

36-37 Two pointer declarations. These pointers point to a pre-defined structure called a WINDOW. Curses.h includes the definition for WINDOW.

38-81 Variable declarations, used for dimensions of specific windows.

91 Main ().

94 Function call for the interrupt signal.

99 Function call to **initscr ()**. It's a mandatory function call for all curses programming. It must be made at the beginning of any curses activity.

104-110 Creation of the main window. The function used is **newwin** ( ) with the desired dimensions. The returned pointer is assigned to wptr.

114 Draw a box around the main window. The function used is **box** ( ).

118 Refresh the screen using **wrefresh** ( ). Having defined a window, any subsequent activity in that window must follow a call to **wrefresh** ().

122-127 Draw a sub-window. The function used is **subwin** ( ) with the desired arguments.

131 Draw a box around the sub-window.

132 Refresh the screen.

136 Write heading for the sub-window by invoking **WriteText** ( ), which is defined in the program.

140-144 Draw a field on-creen to capture the user-id by calling **MakeField** () and **WriteText** ().

148-152 Draw a field on-screen to capture a password by calling **MakeField** ( ) and **WriteText** ( ).

156 Draw a field on-screen for displaying the message.

160-161 Draw a field on-screen for display date and time. Display data and tile by calling **DisplayTime**( ).

165-178 Get user-id from the screen. Get password from the screen. If they are the same

```
    Break
Else
    Continue
```

182 Curses function call to **endwin** ( ). This is required in order to return the terminal to normal working mode.

CONCEPT OF A WINDOW

A window is an internal data representation of an image of what a particular rectangular section of the terminal display may look like. The terminal display as a whole could be said to be a window, in which case its dimensions will be defined as 25 x 80. A window with dimensions of one character in length and one character in height is, in fact, a window of the size of a character. This is the smallest window that curses could possibly handle, but a window could also have dimensions of 128 characters in length and 50 characters in height. This would be bigger than most terminal screens, but, nonetheless, it is still a window.

Data structure called window

To master the various functions used in the example above, you need to understand the curses data structure known as a window, since almost all of the curses routines manipulate this structure in some way.

It is defined in curses.h as follows:

```
struct _win_st
{
     short         _cury, _curx; /* current coordinates */
     short         _maxy, _maxx; /* max coordinates     */
     short         _begy, _begx; /* (Ø,Ø) screen coordinates */
     char     _flags;
     short         _yoffset;      /* actual begy is _begy+_yoffset */
     bool     _clear,       /* clearok() info */
              _leave,       /* leaveok() info */
              _immed,       /* window in immediate mode */
              _sync;        /* auto syncup of parent */
     WINDOW        *_padwin;/* "pad" area for current window */
#ifdef   _VR3_COMPAT_CODE
     _ochtype **_y16;       /* MUST stay at this offset in WINDOW */
#endif
     short         *_firstch;          /* first change in line */
     short         *_lastch;      /* last change in line */
     short         _tmarg, _bmarg;    /* scrolling region bounds */
                              /* MUST stay at this offset in WINDOW */
     unsigned _scroll       : 1; /* scrollok() info */
     unsigned _use_idl : 1;
     unsigned _use_keypad   : 1;
     unsigned _notimeout    : 1;
     unsigned _use_idc : 1;
     chtype        _attrs;  /* current window attributes */
     chtype        _bkgd;   /* background, normally blank */
     int      _delay;       /* delay period on wgetch */
                              /* Ø:  for nodelay */
                              /* <Ø: for infinite delay */
                              /* >Ø: delay time in millisec */
     short         _ndescs;       /* number of descendants */
     short         _parx, _pary; /* coords relative to parent (0,0) */
     WINDOW        *_parent;/* the parent if this is a subwin */
     chtype        **_y;         /* lines of data */
     short         _nbyte;       /* number of bytes to come */
     short         _index;       /* index to hold coming char */
     char     _waitc[CSMAX];/* array to hold partial m-width char */
     bool     _insmode;      /* TRUE for inserting, */
                             /* FALSE for adding */
};

 typedef  struct  _win_st   WINDOW ;
 extern   WINDOW   *stdscr  ,  *curscr ;
```

This structure is in fact curses' internal representation of a window. It contains all the necessary data and information which curses needs to manage the window on the terminal screen. For curses, anything inside or belonging to a window is modifiable.

Physical terminal screen and default window

Before curses can manage the terminal screen it needs to know what it looks like. Therefore, when curses starts up (after the invocation of initscr ( )), the first thing it does is clear the screen. It then places the cursor in the home position, which is the top left-hand corner of the screen. Curses then knows exactly what the physical screen looks like and where the cursor is situated.

The external pointer variables **stdscr** and **curscr** defined in <curses.h> are two virtual-window pointers. These windows are initially the size of the physical screen and created by the curses start-up function **initscr** (). The **stdscr** window is provided for developers, while the **curscr** window is generally reserved for internal curses use.

Physical terminal dimensions and windows

A window can be the same size as or smaller or bigger than the physical terminal screen. A physical terminal can display more than one window. The home coordinates (0,0) for any window is the top left-hand corner of the screen.

SCREENS, WINDOWS, AND TERMINALS

The following list defines each of these terms, which are used widely in the discussion of curses programming.

Screen

A screen is a terminal’s physical output device.

Window

Window objects are two-dimensional arrays of characters. Curses provide **stdscr**, a default window which is the size of the terminal screen. You can use the **newwin** function to create others.

To refer to a window, use a variable declared as WINDOW *.

There are three sub-types of window:

- Subwindow – a window that has been created within another window (the parent window) and whose position has been specified with absolute screen coordinates.
- Derived window – a subwindow whose position is defined relative to the parent window’s coordinates rather than in absolute terms.
- Pad – a special type of window that can be larger than the screen.

Terminal

A terminal is the input and output device that character-based applications use to interact with the user.

CREATING NEW WINDOWS, SUBWINDOWS, AND DERIVED WINDOWS

New windows

If you want to use a window other than the default windows supplied by curses (**stdscr** and **curscr**), you need to create it before it can be accessed. Curses provides the following function:

```
WINDOW    *neww_ptr
 neww_ptr =   newwin (  lines,  cols,   begy,  begx) .
```

where:

- *lines* is the maximum vertical dimension of the new window, specified in units of lines.
- *cols* is the maximum horizontal dimension of the new window, specified in units of columns.
- *begy* is the Y coordinate for the new window in relation to the **stdscr** (0, 0).
- *begx* is the X coordinate for the new window in relation to the **stdscr** (0, 0).

Note: the X and Y coordinates are relative to the home coordinates of the default window, pointed to by **stdscr**, and are located at 0, 0.

Subwindows

```
WINDOW *subw_ptr
subw_ptr = newwin (neww_ptr, lines, cols, begy, begx) .
```

The X and Y coordinates are relative to the home coordinates of the default window, pointed to by **stdscr**, and are located at 0,0.

Derived windows

```
WINDOW *derw_ptr
derw_ptr = newwin (neww_ptr, lines, cols, begy, begx) .
```

The X and Y coordinates are relative to the home coordinates for the parent window.

CURSES FUNCTIONS

There are a lot of curses functions for screen manipulation. In fact, for each standard input and output function there exists a corresponding curses function.

In general, functions that take a pointer to a window as one of the parameters are prefixed with w (eg, **wrefresh**, **wmove**, etc); otherwise, functions act on the default window pointed to by **stdscr** (eg refresh, getch etc).

Curses library

The library is libcurses.a and its usual location is */usr/include/lib*.

INTERACTIVE SHELL

The program creates the effect of two independent windows on the terminal screen with a shell running in each.

```
/***********************************************************************
*  Name     : twinmish (two window multiplexing interactive shell)
*  Overview : A two window multiplexing interactive shells.
*  Notes    : 1.  Compile the program as follows:
*               cc  -o twinmish  twinmish.c  /usr/lib/libcurses.a
***********************************************************************/
/***********************************************************************
*                  INCLUDE FILES
***********************************************************************/
#include  <curses.h>
#include  <signal.h>
#include  <fcntl.h>
#include  <unistd.h>
/***********************************************************************
*                  FUNCTION PROTOTYPES
***********************************************************************/
int  main          (void );
int  CreateWindows (void );
int  CreateShells  (void );
void DisplayError ( char *msg, int line_no );
void DisplayWindowTitle ( char *msg ) ;
void HandleSignal ( int signo );
/***********************************************************************
*                  MODULE CONSTANTS
***********************************************************************/
/*
 * function return codes
 */
#define  SUCCESS  1
#define  FAILURE  0
#define  BUFSIZE  128
/* maximum number of multiplexed windows  */
#define  MAX_WIN    2
```

```
/*  template  for  multiplexed  channels  */
typedef   struct  {
    WINDOW   *win  ;  /* window for this channel     */
    int       out[2]; /* file descriptor for output  */
    int       err[2]; /* file descriptor for err     */
    int       in[2];  /* file descriptor for input    */
    int       pid  ;  /* pid of process controlling this window  */
                  }  WIN ;
/* multiplexed channel  array */
WIN   w[MAX_WIN] ;
/*  template  of  window  positions */
typedef   struct  {
    int       lines; /* no of lines in window    */
    int       cols;  /* no of columns in window  */
    int       begy;  /* y co-ordinate for window */
    int       begx;  /* x co-ordinate for window */
                  }  WINPOS ;
/* define window positions */
static WINPOS  pos[MAX_WIN] = {
                                { 11,79, Ø,Ø },
                                     /*  specification of ist window */
                                { 1Ø,79,14,Ø }
                                      /* specification of 2nd window */
                               } ;
int  pid  ;
/***********************************************************************
*  Name     : main
*  Overview : The function implements processing structure.
*  Notes    :
***********************************************************************/
int main (void)
{
int  nc  ;   /*  no of characters read */
char    buffer[BUFSIZE] ;
int  i, c ;
int  cwin = Ø ;  /* current  window */
/* set the signal */
signal ( SIGCLD, (void * )HandleSignal );
/* create two windows */
if  (  CreateWindows () !=  SUCCESS  )
  return ( FAILURE ) ;
/* create two shells */
if (  CreateShells ()  !=  SUCCESS )
  return ( FAILURE ) ;
nodelay ( stdscr, TRUE );
noecho ();
raw();
DisplayWindowTitle ("Top Window" );
for ( i= MAX_WIN -1 ;  i >= Ø  ;  i-- )
    wrefresh ( w[i].win ) ;
```

```
/* start polling following devices  for input
 * input file descriptor
 * keyboard */
while (1)
 {
 /*  read  from  input pipe in each window  */
  for ( i=Ø ; i < MAX_WIN ; i++ )
    {
      /* stdout  channel in[Ø]
       * shell would have written to  in[1]       */
       nc =  read ( w[i].in[Ø],buffer,BUFSIZE -1 ) ;
       buffer[nc]='\Ø' ;
       waddstr ( w[i].win, buffer );
      /* stderr  channel       */
```

Editor's note: this article will be concluded next month.

Arif Zaman
DBA/Developer (UK)

Recovering deleted files

Here's the situation: a newish AIX programmer deleted their latest program file in error and wanted to know what to do next. First, they wanted to know what the AIX equivalent to the DOS **UNDELETE** command was. Sad to say, there is no **UNDELETE** command in AIX.

I suggested that they restore from the most recent back-up (previous night) and they would have to re-enter all the changes made since then. Unfortunately there was a problem with the most recent back-ups. The file could be restored, but not straightaway.

What to do next? Well, because of the way AIX stores data, each file has an inode (index-node) structure associated with it containing information about the file owner, permissions, size, the physical disk blocks the file is stored on, etc. If a file is deleted, it remains on the disk; it's just the inodes that are marked as being free for use. If the disk is heavily used, the inode will be reused and the disk space will be written over. If it is not used, the file will still be sitting there without any pointers to it.

All I/O activity on the relevant partition must be stopped and the partition unmounted. We placed the system in single-user mode to prevent other processes from overwriting the disk blocks or inodes previously used by the ‘erased’ file.

A raw copy of the partition containing the missing file had to be made and the system brought back into multi-user mode, allowing normal system operation to continue. Parts of a file could have been scattered in a non-contiguous manner over the entire partition.

The next stage was to recover the data.

We did consider writing a shell script that would copy all files being deleted to a different folder before deleting them in the original folder. We could then write a second script (called UNDELETE.SH, say), which would copy the required file back from this second folder to the original folder.

Editor’s note: we would be interested to hear how other users have got on with the problem of recovering accidentally deleted files.

Susan Alnutt
DBA (UK)

AIX news

Softek, part of the Fujitsu group, has announced Softek Replicator, which runs on AIX, z/OS, Windows, HP-UX, Linux and Solaris, and supports any storage array, such as those from EMC, HDS, H-P, IBM, or StorageTek.

The product replicates data writes to one drive array or disk to another across an IP link. The replication is done at the host level, not by the drive array controller. EMC's SRDF is a controller-based replicator only working with EMC arrays. Softek Replicator, because it is host-based, is independent of the drive arrays and controller firmware.

AIX's JFS file system is supported, which, Softek claims, is unique.

For further information contact:
Softek, 1250 East Arques Avenue, M/S 317, Sunnyvale, CA 94085, USA.
Tel: (408) 746 7638.
URL: http://www.softek.fujitsu.com/en/products/replicator.

* * *

UniPress Software has announced Version 6.0 of its FootPrints software specifically designed to optimize the performance, reliability, and scalability of DB2. This new version, FootPrints for DB2, provides organizations running DB2 in AIX environments with a seamless, integrated, Web-based service desk to centralize and automate internal Help Desk operations and external customer service activities.

FootPrints provides a Web-based system that includes centralized customer-request tracking, self-service online capabilities, two-way e-mail management, knowledge management, and reporting.

For further information contact:
UniPress Software, 2025 Lincoln Highway, Edison, New Jersey 08817, USA.
Tel (732) 287 2100.
URL: http://www.unipress.com/footprints/whatsnew.html.

* * *

Nuance has announced Version 3.0 of its Vocalizer software. The product enables automated access to everything from account balances to flight information, e-mail reading to voice-activated dialling.

In addition to the Windows and SPARC Solaris operating systems, Nuance Vocalizer 3.0 now also supports AIX to enable companies with IBM DirectTalk platforms to take advantage of its advanced TTS features.

Nuance has also introduced a new Canadian French text-to-speech (TTS) voice, updated North American English voices, and made a number of other enhancements to its Nuance Vocalizer 3.0 text-to-speech software including: improved name pronunciation, reduced memory utilization, expanded handling of terms from key vertical markets, and broader operating system support.

For further information contact:
Nuance, 1005 Hamilton Court, Menlo Park, CA 94025, USA.
Tel: (650) 847 0000.
http://www.nuance.com/corp/index.html.