The IBM AS/400
A technical introduction

by Tom Van Looy <tom@ctors.net>
January 2009

This paper was written for the AS/400's 20th anniversary,
and was presented at HAR2009.

Index

History

The Application System 400 was developed at IBM Rochester, Minnesota as a so called midrange system. This basically means that it's a small mainframe system. Rochester started building business computers with the System/3.

Know to have some kind of “fortress” mentality, Rochester systems didn't had large influences from East Coast universities like MULTICS, Unix, VMS, Windows NT and IBM's mainframe operating systems had. This resulted in Rochester's systems being separatist and different. The Rochester system code is not shared with anyone, not even inside IBM. No one except the system code developers in Rochester can generate system code.

The System/38 was built on the S/3 (1969) but being very different from it. Designed by Dr. Frank Soltis, it grew from the S/3 to the S/32 (1975) and S/34 (1977), and was finally announced to the public in 1978. In 1983 the S/36 was released as the latest variant of the S/3. This release was done to satisfy S/32 and S/34 users who found the S/38 to large and expensive for their needs.

Then IBM started an attempt to combine all of it's (incompatible) midrange computers into one. This was project Fort Knox. Fort Knox would replace the S/36 and S/38, Series/1, RS/6000 and a bunch of small IBM mainframe systems. The project failed disastrous. At some point IBM even declared the S/38 as “non-strategic”, and discouraged customers from buying the system. This made it difficult for the S/38 to remain competitive, and left Rochester with nothing.

Rochester's backup plan was project Siverlake. This project took some stuff from S/36, S/38, some leftovers from Fort Knox and turned this into the AS/400. In many ways the AS/400 was actually just a new implementation of the S/38. It was announced to the public in 1988.

Note that the System/38 was not built on the System/370. Many people declared this in the past and IBM even funded projects to make the S/38 software run on the S/370. All these projects have failed. Note that the S/370 was preceded by the S/360 (1964). This range of systems led to today's System z.

To make a long story short, in the past the AS/400 was renamed several times:

- AS/400
- AS/400e
- eServer iSeries
- eServer i5
- System i5
- System i
- POWER systems

I probably forget a few, but anyway. Since 2008 (20 years later), we know the AS/400 as the IBM i. The i standing for integrated. At the time of this writing, the latest version of IBM i is 6.1. IBM i runs on POWER (Performance Optimized With Enhanced RISC) hardware.

What is IBM i?

In the late 1960's, to settle some lawsuits, IBM agreed to not bundle software with it's computers anymore. Bundling was a practice that require customers to buy the hardware to get the system software. This was done to avoid that other companies cloned the system's hardware and sold it at lower prices.

With the S/38 they wanted to make a system where the MI was the only external interface. As a result, every part of the system that was hardware dependent was packaged as part of the hardware. Because microcode in those days was considered hardware, the operating system kernel of the S/38 was packaged as part of the microcode. At the time of shipping, the packaging laws were changed again but the S/38 kept this design.

If you loosely compare the AS/400 system software to GNU/Linux, the microcode would be the Linux part and IBM i would be the GNU part. Do mind that IBM i is more sophisticated and extended than only operating system software.

Versions:

- System Support Program (SSP) – System/36
- Control Program Facility (CPF) – System/38
- OS/400 – AS/400
- i5/OS – iSeries
- IBM i – POWER

With the release of IBM POWER 6 systems in 2008, IBM finally finished the merge of the System i and System p hardware. Today it is possible to run IBM i, AIX and Linux on the IBM POWER systems hardware line. Mainly because of finishing this merge, Dr. Frank Soltis will retire from IBM at the end of this year.

IBM i is defined by the five fundamental architectural principles, which will be explained in the following paragraphs.

Technology independence

Unlike other computer systems, the IBM i is not defined by it's hardware. Programs don't speak directly to the hardware but instead speak to the Technology Independent Machine Interface (TIMI). The MI is a virtual machine interface to the system, consisting of two parts; computational and object-oriented instructions, and operands upon which these instructions act. Objects are the only supported data structure at the MI level. Between the MI and the hardware is the microcode or System Licensed Internal Code (SLIC).

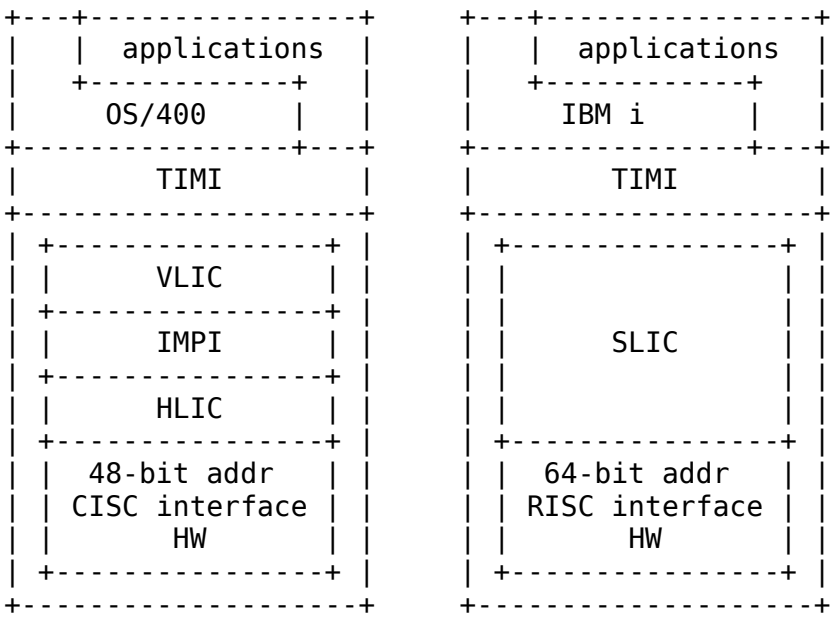
The advantage of this technology independence was demonstrated in 1995 when the AS/400 moved from the 48-bit CISC (IMPI) to the 64-bit RISC (PowerPC) processor architecture. Customers could just save/restore their programs on the new machine, and they would run as truly 64-bit applications. Without even the need to recompile anything. IBM could just rewrite the LIC components that were aware of the changes and thus preserve the MI's integrity.

Compare this to how we are today moving from 32-bit to 64-bit on the x86 platform. Even though most 32-bit applications will run on 64-bit hardware, the applications will still have to be rewritten to fully employ the 64-bit design. The same story happened when Intel moved from 16-bit to 32-bit.

To quote an other example of the past. When people would migrate from Digital's 32-bit VAX to 64-bit Alpha, Digital estimated that 15 to 20 percent of every application needed to be rewritten.

When the AS/400 was still a 48-bit CISC system, the microcode was implemented in a vertical and horizontal version. The VLIC translated instructions passed to the MI, and handed the translated instructions to the HLIC that dealt directly with the hardware. HLIC was the microcoded emulator of the Internal Microprogrammed Interface (IMPI) processor architecture.

When the AS/400 moved to 64-bit RISC, the LIC was rewritten. Because a RISC processor has no microcoded emulator, there is no HLIC part. They ended up having only one level of microcode, and they renamed it to System Licensed Internal Code (SLIC). The original VLIC was programmed in PL/MP (Programming Language / Machine Product). Since PL/MP was not usable for OO programming, the SLIC was written in C++ and PowerPC assembler. The SLIC, nor PPC AS are accessible to the end users.



Since the S/38, the system's instruction set defines all pointers as 128-bit. Today, 128-bit processors are already used for special purpose gaming and graphics applications. When at a point in the future 128-bit general purpose processors would appear, IBM i will already be fully 128-bit enabled.

Object-based design

IBM i has a fully object-based system design. Everything within the system is stored as an object, consisting out of two inseparable parts. A descriptive part, which defines valid ways of using the object; and a data part, which covers the functional aspect of the object.

An object-based design has security implications. If an object enters the system as data, it must remain data forever. Entity's are encapsulated. The descriptive part of the object will not let you, for example, treat the object as an executable. Objects are atomic and can't be manipulated except as an entire object.

Objects communicate with each other by the process of message passing. Every addressable object in IBM i has a unique message queue attached.

IBM i doesn't have a fully object-oriented design, because it lacks some of the basic characteristics of a OO model. For example, it doesn't support subclass creation or inheritance and polymorphism is minimally defined.

Hardware integration

Because of it's business-computing nature, the AS/400 was optimized for information-intensive environments instead of compute-intensive environments. Meaning that users would do simple operations on a large amount of data, rather than perform complex operations on a relatively small amount of data.

With the arrival of Java based business-intelligence and e-business applications, the general business environment became increasingly compute-intensive. The move to PowerPC processors was intended to enhance the compute-intensive processing capabilities of the system. With this integration they tried to create a balanced design between fast processing, high-bandwidth memory and high-performance I/O for all business applications.

An IBM POWER processor's capacity is expressed in a Commercial Processing Workload (CPW) value. The CPW disregards the CPU percentage. Instead it is based on the TPC-C (Transaction Processing Performance Council) benchmark and simulates the database server of an on-line transaction processing (OLTP) environment . It counts the number of transactions a system can complete while delivering 90% of those transaction at less than five seconds response time.

The POWER processors actually support five architectures: 64-bit Amazon, 64-bit PowerPC, 32-bit PowerPC, 64-bit POWER and 32-bit POWER.

Software integration

Instead of having to get additional fundamental software components from other vendors, software for security, communications, backup/recovery and the database (DB2/400) are fully integrated into the standard operating system.

These components are separately installed as licensed program's. IBM creates updates and patches for the system called Program Temporary Fixes (PTF). These can be installed temporary or permanent. You have the ability to start the system with or without loading the temporary applied fixes. And, you can make the temporary fixes permanent when you tested them and know they solved your problem and don't cause new ones.

When you add storage devices to the machine, the system will just recognize and use it. The system also does stuff like automatically extending files that are full etc. DB2/400 is a modern database and supports journaling, commitment control, triggers, referential integrity, stored procedures, it also has a predictive query governor, an EXPLAIN command to analyze SQL statements, support for distributed databases, replication, etc. Because technical database administration tasks are subsumed into overall system management tasks, most IBM i installations don't need a traditional database administrator.

Single-level store

Virtual memory is organized by the SLIC as a single-level store. IBM i has a 64-bit address space which can address 18 quintillion bytes of data. All programs and data reside in this massive space and are addressable from a single permanent address, which is never reused. Users just have to reference programs by name or by pointer, and don't need to worry where it resides on the system.

The single level store is about sharing. There is no need to create a separate address space when executing a new task. Program's don't copy objects into a user's address space. Also, task switching is very fast because it's simple as saving the processor's registers and performing a branch instruction to the location where the task resides. The virtual addressing mechanism is responsible for moving the object into real memory.

Programmers or system administrators don't have the ability to access real memory. These kind of functions are managed by the SLIC, invisible to the MI or to IBM i.

Each 8 bytes of memory in IBM i has a specific bit referred to as a tag bit. If the tag bit is set, the memory location is treated as a MI address pointer, used to access data and MI objects. If a user program modifies any data in the MI address pointer, the tag bit is turned off by hardware, disabling it’s further use as an object pointer.

Work management

Jobs

There are four major types of jobs inside a system:

- Interactive
- Batch
- Communication
- System

Jobs on the system are identified by a combination of job-number/user-name/job-name. The job-numbers are not reused.

A subsystem has workstation entry's, these entry's describe one or more work stations that are controlled by the subsystem. Interactive jobs will use a terminal session. They run in the foreground and tie up a workstation screen until they finish. Interactive programs are called. When a user is singed on, that user is running an interactive job, when the user calls a program or issues a command from the command line, the resulting program runs interactively.

Batch jobs run in the background and don't tie up a workstation screen. Batch jobs are submitted. Before a batch job becomes active it must pass through a job queue. A subsystem can be fed by several job queues, but a job queue can only belong to one subsystem. Jobs have a priority on the jobq, jobs with equal priority are processed in FIFO order.

Communication jobs are similar to batch jobs, only they are started by a remote IBM i system.

System jobs are not critical to master because you can do nothing with them for performance. But they do take resources. So, it's handy if you know which jobs are system backup jobs, etc.

Pools

The IBM i's memory is divided into storage pools. By default, the system is divided in a pool for system tasks (*MACHINE) and a pool where everything else runs (*BASE). Most people will create additional pools for specific types of jobs. For example, a pool can be configured to dynamically adjust the paging options of the memory inside the pool to increase system performance. The pool sizes and job activity level of shared pools can be automatically adjusted by the system (QPFRADJ system value). You can also make sure that

some jobs will not use up all memory by placing them in a fixed size private pool.

Jobs don't run directly in a storage pool, they run inside subsystems instead. A job has a job description, this object contains the attributes the job will have. These attributes make sure the job can be recognized by the IBM i and are different from the job's runtime attributes. The job description contains stuff like the output queue and the job's initial library list. The job description also contains routing data, this data contains information that's later used by the job class. A job class is an object that provides runtime attributes like runtime priority and CPU time slice.

Programming

OPM

The traditional languages like RPG and COBOL used to write AS/400 applications are first translated into MI programs and then morphed into PowerPC programs by the internal optimizing compiler, before the program is executed.

In the early days of the AS/400, the program model was called the Original Program Model (OPM). MI instructions were generated in a fairly direct manner. The OPM source was generated into an Intermediate Representation of a Program (IRP). Next, the IRP had to be converted into MI instructions. This was done by the Program Resolution Monitor (PRM) and resulted in an OPM program template.

As new languages like C/400, Pascal, etc. were implemented on the AS/400, some extensions had to be added to the OPM. This resulted in an Extended Program Model (EPM). The compilers for these new languages were implemented with separate front- and back ends. The new compiler back end was called Common Use Back End (CUBE-1). The only type of call instruction was call external. This is a type of late binding. All references are resolved at runtime, which comes with a performance cost.

ILE

To improve the performance of modular programming, a new programming model was introduced. The Integrated Language Environment (ILE) introduced new language compilers, a new optimizing translator and a binder facility. The output of an ILE translator is not a program object like with OPM, but instead is a module. The ILE binder packages these modules into programs and introduced the ability to do early binding. ILE uses the 3th generation of compiler back end technology (CUBE-3).

A module contains procedures which can be called at an entry point, with optional parameters. A module is not executable, instead modules are contained in programs. A program has a single entry point and is dynamically called. Programs can contain multiple modules which can be generated by different language compilers.

Service programs are also executable units. They have multiple entry points, one for each procedure. Each procedure can be statically called. The purpose of service programs is to support multiple types of static calls. Programs use bind by copy (CALLPGM), where procedures are resolved at compile time. Multiple copies of the same module may exist in memory if the module is bound into multiple programs. Service programs use bind by reference (CALLB). Symbolic links to modules inside the service programs are stored into the programs. When a program is activated, those link are resolved inside the service program and replaced by the calls to the address of the procedures.

This is all similar to how Linux and Windows work. Below is an overview:

	ILE	Windows	Linux (ELF)
After compilation	*.MODULE	*.obj	*.o
Bind by copy	*.PGM	*.exe	*
Bind by reference	*.SRVPGM	*.dll	*.so

Java programs are also first compiled into byte codes that are part of the MI. These byte codes are then used by the Java Virtual Machine (JVM), which is implemented in SLIC.

The IBM i uses a three level hierarchy to manage the flow of work through a system. A job on the IBM i is a unit of work represented as an object that contains, among other things, a process structure that's used to manage the system resources required to complete the unit of work. The third level in the flow of work are kernel threads. Threads are a portion of the process and are scheduled for execution by the process management component of SLIC. The IBM i also supports two user thread management interfaces, namely pthreads (POSIX) and the Java thread class.

A process is a system object at the MI, called a process control space (PCS). There is no equivalent OS/400 object. A process object's responsibility is to tie together the resources needed to execute a program. A process object contains an activation group. When a program is called the CALLPGM activates the program. It completes the inter-program binding and implicitly creates an activation group. Activation groups are always named, if not explicitly then implicitly. An activation group is the working storage allocated to run one or more programs. Each activation group contains a:

- program static storage area (PSSA): section used for global static data
- program automatic storage area (PASA): the call/return stack, used to allocate local variables
- heap storage area: dynamic data that does not adhere to a stack structure, variable length data etc.

A system may use these storage areas to use a combination of registers, memory and disk storage. That's why these areas are referred to as storage areas and not memory areas. The static and heap storage are accessible by any thread with a program or procedure running in the activation group. The automatic storage is associated with the thread. Activation groups can be shared among multiple threads, and a thread may have code actively running in multiple activation groups.

Exploring the system

Command interface

An immediately noticeable distinctive feature of the IBM i is its command syntax and menu-based command interface.

Each command consists of a concatenation of simple abbreviations. For example, the command “work with object” is WRKOBJ, while the command “work with output queue” would be WRKOUTQ. Every command can be prompted with F4, and every command's parameters can be prompted as well. Every command / parameter has a help, you can just put your cursor at it and press F1.

The command entry prompt is always available, but the system also provides a series of topic-oriented menus to facilitate command use. For example, the MAIN menu looks like this:

```
Select one of the following:
     1. User tasks
     2. Office tasks
     3. General system tasks
     4. Files, libraries, and folders
     5. Programming
     6. Communications
     7. Define or change the system
     8. Problem handling
     9. Display a menu
    10. Information Assistant options
    11. iSeries Access tasks
    90. Sign off
```

When you call the MAJOR menu with GO MAJOR, you get the following:

```
     1. Select Command by Name                                  SLTCMD
     2. Verb Commands                                             VERB
     3. Subject Commands                                       SUBJECT
     4. Object Management Commands                          CMDOBJMGT
     5. File Commands                                         CMDFILE
     6. Save and Restore Commands                           CMDSAVRST
     7. Work Management Commands                            CMDWRKMGT
     8. Data Management Commands                            CMDDTAMGT
     9. Security Commands                                      CMDSEC
    10. Print Commands                                         CMDPRT
    11. Spooling Commands                                      CMDSPL
    12. System Control Commands                            CMDSYSCTL
    13. Program Commands                                       CMDPGM
```

File system structure

An object name is maximum 10 characters long and is contained in a library (*LIB object). A library is similar to a directory on Unix. The root of the IBM i file system is the special library QSYS. This is the only *LIB object that can contain other *LIB objects. All IBM-supplied system library names begin with the letter Q or #, it's considered good practice to never let your own library names start with one of these two letters.

The object name consists of the given name concatenated with the object type. It is perfectly possible to have two objects named OBJ1, one being a file (*FILE) and one being a program (*PGM). Related objects are grouped into libraries (object type *LIB). Every object is implicitly related to a library, and always explicitly accessible like <library>/<object>. Objects are stored in EBCDIC format, so when transferring data to the PC, translation to ASCII is needed.

A job on the system uses a library list, comparable to $PATH on Unix. A library list is composed of a system and user part. Library QSYS will be in the system part of the library list. And, for example, a library TVL will be in the user part of the library list.

The IBM i has an Integrated Filesystem (IFS). In the early years, OS/400 only had the QSYS single level file system hierarchy. Over time, additional file systems were supported by adding a Virtual File System (VFS) architecture. The VFS provides a common interface to all file systems in the IFS. One remarkable file system is the QOpenSys file system. This file system contains Unix-based files and directories, and is fully

compatible with the Portable Operating-System Interface for Unix (POSIX) and the X/Open Portability Guide (XPG). The IFS also supports things like optical file systems and servers like NFS and SMB.

Users

Users are assigned a user class that holds some special authority levels. The IBM i superuser is called QSECOFR (security officer).

Authority levels:

- all object (*ALLOBJ)
- auditing (*AUDIT)
- I/O system configuration (*IOSYSCFG)
- job control (*JOBCTL)
- save system (*SAVSYS)
- security administration (*SECADM)
- service functions (*SERVICE)
- spool control (*SPLCTL)

Originally, users would use an IBM 5250 terminal device to connect to the system. Few 5250 terminals still exist, the term 5250 now refers to the content of the data stream itself. IBM provides IBM i users with a 5250 emulator instead.

When you connect to the system you get a “sing on” screen, which is configurable but by default, looks something like this:

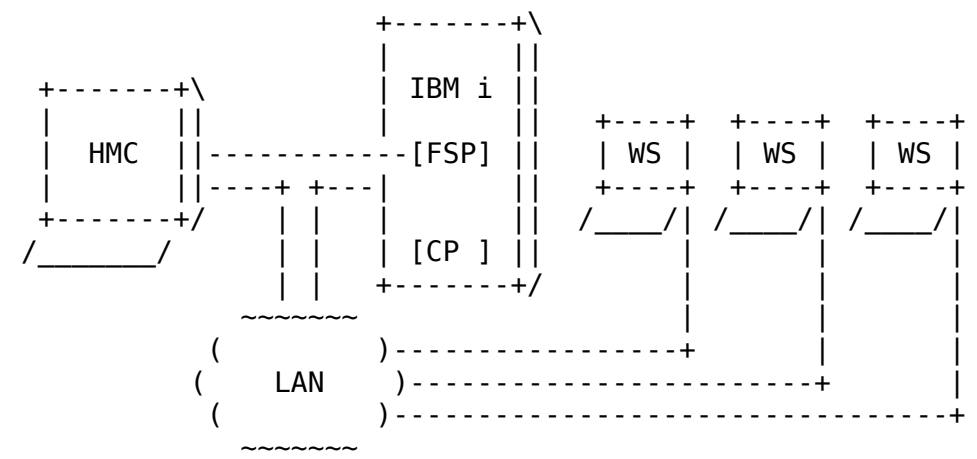
```
                Sign On
                           System  . . . . . :   SOMESYS
                           Subsystem . . . . :   QINTER
                           Display . . . . . :   QPADEV0001
    User  . . . . . . . . . . . . .
    Password  . . . . . . . . . . .
    Program/procedure . . . . . . .
    Menu  . . . . . . . . . . . . .
    Current library . . . . . . . .
```

At sign on, the user's initial library list gets set and it's default menu is shown. Because a user, like everything else on the system, is an object, a message queue is also attached. All these parameters are configurable in the user profile. When signed on, IBM i supports group jobs on your terminal. This can be compared to the GNU screen utility on Unix, without the detach functionality.

Starting the system

The IBM i is managed from a console. There are two types of consoles. The obsolete twinax console, and the Hardware Management Console (HMC). The second being required since POWER 6. The HMC is basically an IBM System x running Linux with some base utility's, X with Fluxbox to provide graphical login's, and a Java based application for system administration.

The HMC connects to the IBM i's Flexible Service Processor (FSP) over a network interface. The FSP is a firmware that provides access to the IBM i's System Service Tools (SST) and Dedicated Service Tools (DST) for diagnostics, initialization, configuration, run-time error detection and correction. SST can also be accessed from a 5250 connection, DST is only alternatively accessible from the system control panel (CP).



The HMC can be used to initiate the system's Initial Program Load (IPL). Or, simply stated, boot the system. The FSP sends the IPL values to the HMC. For example, code C6004056 means that “Journal recovery is running”. You can look up the IPL values of the last boot by using the command “CALL QWCCRTEC”. This generates a QPSRVDMP spoolfile.

IPL attributes can be configured with the CHGIPLA command, for example you can specify to not clear the system's output queue's etc. There are also some system values important to IPL. For example, system value QCTLSBSD holds the name of the controlling subsystem. By default IBM ships two subsystems: QBASE and QCTL. These subsystems have autostart jobs to start a few required subsystems. The QCTL controlling subsystem will start the following subsystems (QBASE starts just a subset of those):

- QINTER (interactive jobs)
- QBATCH (batch jobs)
- QCMN (communication jobs)
- QUSRWRK (user work)
- QSERVER (server jobs)
- QSPL (spooling)

People can bring the system in a restricted state by stopping all subsystems except the controlling subsystem (ENDSBS SBS(*ALL)). For example, this will make sure that object locks are removed so they don't get in the way. Things like the console display and backup jobs will typically run in the controlling subsystem.

The QSTRUPPGM system value holds the name of the first program that is started when the IPL is complete. Typically this program will start all additional subsystems.

A few other interesting system values are:

- QDATETIME, keeps the current date and time
- QMODEL, the system model number
- QSRLNBR, the system serial number
- QSYSLIBL, the system part of the library list
- QSECURITY, the security level

Other environments

Unix

In addition to the QOpenSys Unix-based filesystem, the IBM i also incorporates an entire 64-bit AIX runtime environment. The problem with this environment was that Unix only supports process-local storage. Unix doesn't have the concept of a large shared storage space like in the IBM i.

Process local storage was implemented by providing teraspaces. These temporary storage area's provide private storage for a process. The AIX runtime was initially called Private Address Space Environment (PASE), but later this acronym was renamed to Portable Application Solution Environment.

PASE gives you a Unix shell and a bunch of tools. It's even possible to run an Apache webserver, X, ... You can call a PASE shell script from IBM i with: QSH CMD('ls /home/tvl'). It's also possible to call a IBM i command from the PASE shell with: system ”WRKOBJLCK OBJ(MYFILE) OBJTYPE(*FILE)”. You can even run DB2 query's from PASE and pipe the output to some other script.

In my opinion, the PASE environment makes IBM i more flexible, especially when you want to get a small job done. For example, some command's don't have the option to output to a file. IBM does provide API's so you can implement this functionality yourself. But in PASE you can just call the command and redirect to a file.

A disadvantage of PASE is that it's not technology independent. Instead of talking to the TIMI, it has a syscall interface to the SLIC. You will need to recompile your PASE programs when you move to an other hardware architecture.

Hypervisor

The IBM i can do Logical Partitioning (LPAR). This means you can run multiple instances and even different version of IBM i on the same hardware. A hypervisor provides the isolation between user- and system state programs. The hypervisor on the IBM i has hardware support in the PowerPC processor. LPAR's are also created and managed from the HMC, it is possible to allocate resources to a partition at runtime, in a dynamic way.

IBM ported the Linux kernel to the IBM i platform, so Linux can run inside an LPAR. Both Redhat and Novell provide a Linux distribution for the POWER platform.

Also, AIX can run in an LPAR. This was also possible on the System i. But, back then i5/OS didn't run on AIX's System p hardware. Today System i and System p are merged into the POWER platform, making the hardware identical.

Windows

Windows does not run on the POWER platform. Because Windows isn't open-source software, it's not so easy to port the kernel to an other platform. But IBM did put some effort in making it possible to consolidate Windows servers onto the IBM i. They created an Integrated xSeries Server (IXS). The IXS is physically packaged on a card with an Intel processor and PC memory that fits inside the IBM i. Linux can also run on an IXS.

Bibliography

- Fortress Rochester, the inside story of the IBM iSeries; ISBN 1-58304-083-8
- Inside the AS/400; ISBN 1-882419-13-8
- Goodbye, AS/400, Old Friend; http://www.itjungle.com/tfh/tfh040708-story05.html
- What is an iSeries; http://systeminetwork.com/article/what-iseries
- Understanding AS/400 system operations; ISBN 1-58347-015-8
- Architecture of the IBM AS/400; Vincent LeVeque, James Madison University, CS511
- Wikipedia, the free encyclopedia; http://en.wikipedia.org/ (S/3, S/32, S/34, S/36, S/38, AS/400)
- i5/OS information center, http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp
- When is PowerPC not PowerPC, http://systeminetwork.com/article/when-powerpc-not-powerpc

About the author

Tom Van Looy started out his IT career as a software developer in 2005 and moved over to the position of network / system engineer in 2007. Since he changed jobs in 2009, Tom doesn't use the AS/400 in production anymore. But the respect to this remarkable system has remained. Tom can be reached by email at tom@ctors.net.

This paper was written for the AS/400's 20th anniversary, and was presented at HAR2009.

Legal notice