54 AIX

April 2000

update

In this issue

Published by

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

North American office

Xephon/QNA
Post Office Box 350100, Westminster CO 80035-0100, USA
Telephone: (303) 410 9344
Fax: (303) 438 0290

Contributions

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

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

Editor

Harold Lewis

Disclaimer

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

Subscriptions and back-issues

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

AIX Update on-line

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

Printed in England.

Using Go-Joe and GlobalHost for AIX

The AIX Bonus Pack, which is included in the AIX distribution disks, is a collection of IBM and third-party programs that complement and enhance the basic facilities provided by the AIX operating environment. 'Go-Joe' and 'GlobalHost for AIX', both of which were developed by GraphOn, are a couple of recent additions to it.

These programs provide special X server facilities, network protocols, and a super-lightweight Java applet for running X-Windows-based applications on thin clients.

PROBLEMS ADDRESSED BY THIS SOFTWARE

Most existing GUI-based AIX applications rely on the X-Window system, which was developed about 15 years ago. The way X-Windows functions assumes that the underlying system comprises the user's screen and input devices directly connected to a computer that runs both the X server software and the client software that displays output on the user's screen. X applications communicate with X servers by means of a special protocol that's used to transmit requests to manipulate various resources, such as windows, cursors, fonts, 'pixmaps', and graphics contexts. When both the client and the X server are on the same machine, they communicate via a shared memory mechanism that's relatively efficient. However, the widespread availability of desktop computers in recent years has changed this architecture significantly. Nowadays, the client is typically located on a remote server and its output is displayed on a local screen by X server software that is installed on the user's desktop machine. This situation has many disadvantages.

The X server software has to be acquired and installed on the client, usually at a cost of hundreds of dollars. The software puts a heavy demand on the host's resources, such as disk, memory, and CPU. The software has to be customized to connect to the remote host and invoke specific applications, and its operation also results in additional traffic that clogs corporate networks.

GO-JOE AND GLOBALHOST

GraphOn's solution to this problem comprises three components. The first, GlobalHost, is located on the server. GlobalHost replaces the host's X-server device driver layer (*ddx*) with software that communicates directly with the remote client. Instead of directly driving the client's graphics hardware, the software drives a virtual graphics card on the remote client.

The second component is, thus, the virtual graphics card, which is implemented by Go-Joe, a Java applet that's downloaded from the host on-demand by clients running industry-standard Web browsers, applet viewers, or Java-based hardware display devices, such as IBM's Network Station. Go-Joe's thin-client applet viewer handles keyboard and mouse input and displays the output of the X client on the browser's window.

These communicate by means of GraphOn's proprietary RapidX protocol, which is the third component. Using this protocol results in a significant reduction in network bandwidth used compared with the X protocol used by conventional X-Window systems. RapidX uses three techniques to reduce traditional X protocol traffic: the first replaces X11 primitives by more compact RapidX primitives, the second uses two-dimensional data compression for transferred bitmaps, and the third applies a general compression algorithm to reduce transmitted data. The net result is impressive, especially when used on low-bandwidth wide-area networks and dial-up connections.

INSTALLATION AND CUSTOMIZATION

The following file sets are needed to install GraphOn successfully:

- *X11.dt.rte*
- *X11.apps.rte*
- *X11.apps.xterm*
- *X11.base.rte*
- *X11.font.coreX*
- *X11.vfb*.

In addition, file sets containing messages needed by the above file sets should be installed for all locales that are to be used on the system.

For it to be possible to invoke the program from a remote browser, it is necessary for it to interact with any previously installed and configured Web server, such as Lotus Go Web server, Internet Fasttrack Web sever, or IBM HTTP server (Apache).

The program itself is supplied in two file sets on the AIX Bonus Pack CD-ROM:

- *Graphon.client 2.2.0.0*
- *Graphon.server 1.0.8.0*.

The product is installed in directory */usr/G11R6.*

After installing Graphon's file sets using **smit**, you should perform the following customizations to configure GlobalHost:

1 Add the following line to */etc/services*:

```
go-login      491/tcp      #Graphon
```

2 Add the following line to */etc/inetd.conf*:

```
Go-login stream tcp  nowait root /usr/G11R6/bin/go-login
➤  /usr/G11R6/bin/goglobal -- -inetd
```

(Note that the above is a single line; '➤' is the continuation character.)

3 Force the **inetd** daemon to re-read its configuration file by executing following command:

```
refresh -s inetd
```

The next step is to configure your Web server to allow it to connect to GlobalHost. First, execute the script:

```
/usr/G11R6/client/Go-Joe_config
```

to place Go-Joe's Java and HTML files in a directory that is accessible to your Web server. You will be prompted for the name of the directory, and the following assumes a typical installation for common Web servers:

Netscape Fast Track server:

```
/usr/ns-home/docs/Go-Joe
```

Lotus Go Web server:

```
/usr/lpp/internet/server_root/pub/Go-Joe
```

IBM HTTP server:

```
/usr/lpp/HTTPServer/share/htdocs/Go-Joe
```

The last step is to edit the file *xsession.html* that is located in this directory. Find the line that contains the definition of the Java applet parameter *server*:

```
<!--
     Parameter: server
     Specifies the server running the GlobalHost software.
     No default.
-->

<param name=server value=server_running_globalhost/go-login
                         goes here>
```

Replace the string ‘server_running_globalhost/go-login goes here’ with the host name of your server (*zeus*, in my case).

```
<!--
     Parameter: server
     Specifies the server running the GlobalHost software.
     No default.
-->

<param name=server value=zeus>
```

To launch Go-Joe, simply start your Web browser or Java-enabled device and navigate to the HTML file that loads the Java applet (*xsession.html* in the example above). The exact location of this file will vary according to how and where Go-Joe was installed, a typical location being something like:

```
http://server_running_globalhost/Go-Joe/xsession.html
```

This file loads the Go-Joe applet, presenting you with a login banner that allows you to specify your username and password for the session. Click the *Start X Session* button to begin the X session to the Unix host. The Go-Joe applet will then connect to the Unix host and display the normal X session for the username that was entered.

TIPS AND TRICKS

The first thing you should know is that the product ships with a bug. The directory */usr/G11R6/etc/keytables/win95_jdksun_en.kt*, which is created during the installation, should be removed and replaced by an identically named file that can be downloaded from the support pages of the *http://www.graphon.com* Web site.

The Go-Joe applet is displayed in the browser's default applet frame. Its attributes can be changed by modifying the *HEIGHT=* and *WIDTH=* values of the *APPLET* tag of the *xsession.html* file. Most browsers allow the applet frame to be larger than the browser window itself, providing scroll bars for moving around the virtual display.

You can improve your use of the available display area by disabling your browser's toolbars and command menus. This can be achieved by creating an HTML file with embedded JavaScript that removes these display features when you load *xsession.html*.

If you have a two-button mouse and are interested in emulating the middle button, press both buttons simultaneously.

To display the output of an X-Window-based program running on a remote host, define your *DISPLAY* environment variable as *hostname:n.0*, where *hostname* is the name of system that's running your Go-Joe session and *n* is the number of the display for the session in which you're interested.

Go-Joe provides several diagnostic tools and outputs to help diagnose problems that may arise as a result of misconfiguration or other difficulties. These should be used whenever problems are encountered.

JAVA CONSOLE LOG

The output of your browser's Java console can be very useful if the applet terminates prematurely – as long as your browser provides access to the console. When viewing the console, look out for exceptions (the Java term for errors) that may be listed. Exceptions may cause further exceptions as the applet continues to execute, and it is the original exception that needs rectifying and not the one that eventually causes execution to fail. In addition, the Go-Joe applet occasionally prints messages on the status bar (though not all Java

environments show these messages, and the messages may be overwritten by the system's own status messages). These messages are also sent to the Java console, where they should be visible even if they are not visible on the status line.

The GlobalHost software redirects the standard error output of GlobalInit and all the X clients started by the software (the window manager and its children) to a file called */tmp/Xerr:n*, where *n* is the display number of the session. This file contains diagnostic messages from both the GlobalInit program and X clients for the entire session.

The */tmp/xlogfile* provides additional GlobalHost information (server version number, keytable filename, etc).

REFERENCES

1 *http://www.graphon.com*

2 *file://usr/G11R6/html/index.html*.

Alex Polak
System Engineer
APS (Israel)

Debugging malloc in AIX 4.3.3

While mismanaged memory allocation is a frequent source of errors in any substantial program, modern programming languages, such as C and C++, still lack built-in features that could help find such errors. This problem is aggravated by the fact that some of these errors are undetected by the operating system and come to light only in production environments. Another related issue is the tracking of allocated memory and the discovery of so-called memory leaks. Memory areas that are allocated but unused by the program add to the complexity of memory management and adversely affect the performance of the system.

A number of third-party and public-domain tools and libraries are available to combat the memory allocation problem. They include:

- ZeroFault from TKG
- Great Circle from Geodesics
- DBMALLOC, which is public domain
- MALLOCDEBUG, which is also public domain.

Each has its own approach and advantages. IBM recently joined this list with the introduction of AIX 4.3.3's Debug Malloc feature. This article describes this feature, illustrating its abilities using short C programs.

DEBUG MALLOC ACTIVATION

One nice feature of Debug Malloc is that you don't need any special compilation or linkage options to activate it. All you need to do is define the following external variables before you invoke the program to be debugged: *MALLOCTYPE* and *MALLOCDEBUG*. The first one must be set to value 'debug' and the second one should contain a comma-separated list of debugging checks to be invoked. For example, if you are using the Korn shell (**ksh**), you should execute the following commands ('➤', the continuation character, indicates a formatting line break that's not present in the original code):

```
export MALLOCTYPE=debug
export MALLOCDEBUG=align:2,postfree_checking,
➤  override_signal_handling
```

If you are using either the C shell (**csh**) or TC shell (**tcsh**) you should execute the following commands:

```
setenv MALLOCTYPE debug
setenv MALLOCDEBUG align:2,postfree_checking,
➤  override_signal_handling
```

DEBUG MALLOC OPTIONS

The environment variable *MALLOCDEBUG* may contain any of the following options, which can be specified in any order:

```
align:N
postfree_checking
validate_ptrs
override_signal_handling
allow_overreading
report_allocations
record_allocations
```

The options are described below.

- *align:N*

  By default, *malloc* returns a pointer to a two-word boundary for programs running in 32-bit mode or a four-word boundary for programs running in 64-bit mode. The Debug Malloc *align:N* option is used to change the default alignment and set it to any power of 2 between 0 and 4096 inclusive. However, applications using DCE components are restricted to an alignment value of 8.

  The following formula can be used to determine the exact number of bytes allowed for overreads/overwrites when Debug Malloc makes an allocation request for *size* bytes, with *N* being the value of *MALLOCDEBUG*'s *align* option:

  ```
  (((size / N ) + 1) *N) - size) % N
  ```

  For instance, if *N* is set to '2', Malloc Debug allocates the exact number of bytes requested if *size* is even, disallowing all overreads and/or overwrites. If *size* is odd, one byte is allocated for overreads and overwrites.

  If *N* is set to '0', no overreads and/or overwrites are allowed by Debug Malloc.

- *postfree_checking*

  When this option is set, Debug Malloc detects attempts to access memory that has been freed. When such an attempt is made, Debug Malloc reports the error and aborts the program. This option automatically enables the *validate_ptrs* option.

- *validate_ptrs*

  This option causes *free()* to check its parameter to make sure that it actually points to a memory block that was previously allocated

by *malloc()*, *calloc()*, or *realloc()* and that the block has not since been freed. If an illegal value is detected, Debug Malloc prints an error message and aborts the program.

- *override_signal_handling*

  Debug Malloc uses the standard Unix *SIGSEGV* (segmentation violation) and *SIGIOT* signals to stop the running program when it detects memory usage violations. If the program blocks or catches either of these signals, the Debug Malloc system will not function properly. The *override_signal_handling* option instructs Debug Malloc that the application being debugged needs special handling in order to overcome this problem. When this option is specified, Debug Malloc performs the following actions when entering each call to a memory allocation function, such as *malloc()*, *realloc()*, *free()*, or *calloc()*:

  1 Disable any existing signal handlers set by the application.

  2 Set the action for *SIGIOT* and *SIGSEGV* signals to the default values of *SIG_DFL*.

  3 Unblock *SIGIOT* and *SIGSEGV*.

  However, if the application's signal handler modifies the actions of *SIGSEGV* between the memory allocation routine calls and performs an invalid memory access, Debug Malloc will not report the error, the application will not abort, and no core file will be produced.

  Note that this option is ineffective in handling processes that use threads as Debug Malloc uses the *sigprocmask()* system call, while threaded programs tend to use the *pthread_sigmask()* system call.

- *allow_overreading*

  Debug Malloc will report any attempt to read memory that is located past the allocated blocks. This option instructs Debug Malloc to ignore errors of this kind in order to allow the uncovering of other types of problem first.

- *record_allocations*

  When this option is specified, Debug Malloc maintains an allocation record for each *malloc()* request. Each record contains the following information:

  – The original address returned to the caller by *malloc()*

  – A six-function ‘traceback’ starting from the call to *malloc()*.

  Each allocation record is kept until the memory associated with its allocation has been freed.

- *report_allocations*

  This option causes Debug Malloc to report all allocations active when the program exits. All memory areas that have not been freed are reported and tagged with any information kept as a result of specifying the *record_allocations* option.

DEBUG MALLOC OUTPUT

If Debug Malloc is enabled and the application runs to normal completion, then Debug Malloc hasn’t detected any memory usage problems. If a memory access or usage error is detected, the program’s execution is terminated and a core file is produced. If the error is caused by an illegal memory access, such as an attempt to write outside an allocated area or read a freed area, then a segmentation violation occurs and a *SIGSEGV* signal is sent to the program. If a memory routine, such as *malloc(), realloc(), free(),* or *calloc(),* detects an error, then a descriptive error message is reported to the user and the standard C library *abort()* subroutine is called.

PERFORMANCE CONSIDERATIONS AND LIMITATIONS

When Debug Malloc is running, the memory allocation subsystem consumes significantly more memory than usual. The size of each *malloc()* request increases by:

```
4096 + 2 * sizeof(unsigned long)
```

and is then rounded up to the next multiple of *PAGESIZE* (4096). As

Debug Malloc places each individual allocated block on a separate page, programs that make many requests for small amounts of memory experience dramatically increased memory usage. Such programs may fail if memory allocation requests fail as a result of a lack of memory or paging space. The same is true of applications that require a large amount of memory to run successfully.

Because of these performance considerations, it's not advisable to use Debug Malloc on a system-wide basis, for instance, by including the definition:

```
MALLOCTYPE=debug
```

in either */etc/environment* or another system-wide initialization file. The same is true of running X server via the **X** or **xinit** commands under Debug Malloc. Any attempt to do so will result in the X server failing as a result of a lack of available memory. However, individual X client applications can still be executed under Debug Malloc without problem.

USING DEBUG MALLOC

In this section, I present a number of programs that demonstrate various memory-related programming errors and show how Debug Malloc can be used to detect them.

All the examples were compiled using IBM's C and C++ compilers Version 3.6.6.2 under AIX 4.3.3.2. All of them run without error when *MALLOCTYPE* is undefined, and all were compiled using the **-g** option that enables the inclusion of debugging information in the executable created by the compiler.

All the samples were run with the following definitions of the *MALLOCTYPE* and *MALLOCDEBUG* external variables:

```
export MALLOCTYPE=debug
export MALLOCDEBUG=align:0,postfree_checking,validate_ptrs,
➤  override_signal_handling,record_allocations
```

MEMORY OVERREAD EXAMPLE

```
#include <stdlib.h>

main()
{
char *p = malloc(10);
int i;

  bzero(p, 10);
  for (i = 0 ; i <= 10 ; ++i) /* should be "i < 10" */
  /* Memory Overead generated here */
  printf("p[%d] = %d\n", i, p[i]);
}
```

When the **dbx** debugger is invoked, the following line is correctly identified as the location of the error:

```
Segmentation fault in main at line 10
```

MEMORY OVERWRITE EXAMPLE

```
main()
{
    char *src = "hello world\n";
    int len = strlen(src);
    char dst = malloc(len); /* Should allocate len + 1 ! */
    int i;

    for (i = 0; i <= len; ++i)
        /* Memory overwrite Error in the next line */
        dst[i] = src[i];
}
```

This program terminates with a *Segmentation fault(coredump*) message. When the **dbx** debugger is invoked, the following line is correctly identified as the location of the error:

```
Segmentation fault in main at line 11
```

MEMORY REALLOCATION ERROR

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>

main()
{
```

```
    char inbuf[BUFSIZ], outbuf[BUFSIZ];
   /* outbuf should be malloc()ed */
    char *foo = outbuf;
    int foolen = sizeof(outbuf);

    foo[0] = '\0';
    while (fgets(inbuf, sizeof(inbuf), stdin)) {
        if (strlen(inbuf) + strlen(foo) + 1 > foolen)
            /* Memory reallocation error in the following call */
            foo = realloc(foo, foolen += BUFSIZ);
        strcpy(foo + strlen(foo), inbuf);
    }
    fputs(foo, stdout);
}
```

The program terminates with following message:

```
Debug Malloc: free() called with pointer not allocated by malloc.
Malloc arena is 0x20001010 to 0x20101000, pointer passed in is
0x2ff21a70
ksh: 25170 IOT/Abort trap(coredump)
```

When the **dbx** debugger is invoked, the command **where** displays the following traceback, correctly identifying the offending line:

```
raise(??) at 0xd1147d28
abort() at 0xd1141450
do_debug_free(??) at 0xd113b06c
do_debug_realloc(??, ??) at 0xd113af7c
realloc(??, ??) at 0xd113d874
main(), line 16 in "breall.c"
```

‘DOUBLE FREE’ ERROR

```
#include <stdio.h>

main()
{
    FILE *f = fopen("tmp", "w");
    if (f) {
        fprintf(f, "hello world\n");
        fclose(f);
        /* Double Free Error is generated here, the FILE */
        /* struct was freed by fclose() */
        free(f);
    }
}
```

The program terminates with following message:

```
Debug Malloc: free() called with pointer not allocated by malloc.
Malloc arena is 0x20001010 to 0x20101000, pointer passed in is
0xf0533a60
IOT/Abort trap(coredump)
```

When the **dbx** debugger is invoked, the command **where** displays the following traceback, correctly identifying the offending line:

```
raise(??) at 0xd1147d28
abort() at 0xd1141450
do_debug_free(??) at 0xd113b06c
main(), line 10 in "dfree.c"
```

MEMORY LEAK REPORT IN A C PROGRAM

```
main(int argc, char ** argv)
{
    char *t;
    int i;

    for (i = 0; i < 10; i++) {
        t = (char *)malloc(1024);
        if (argc > 1) free(t);
    }
}
```

When this program is invoked without a command line argument, eleven allocations are reported by Debug Malloc. When the program is invoked with at least one command line argument, Debug Malloc reports only one allocation, which is made by the runtime system in *atexit()*.

MEMORY LEAK REPORT IN A C++ PROGRAM

```
class Store {
    struct store {
        void *ptr;
        store *next;
    };

    store *top;
    public:
    Store() : top(0) { }
    void insert(void *ptr);
};
```

```
void Store::insert(void *ptr) {
    store *s = new store;
    s->next = top;
    top = s;
    s->ptr = ptr;
    return;
}

int main(int argc, char ** argv) {
    if (argc>1) {
        Store store;

        for (int i = 0; i < 10; i++) {
            int *f = new int;
            store.insert(f);
        }
    }
    return 0;
}
```

When this program is invoked without a command line argument, Debug Malloc reports 125 allocations made by various components of C++'s run-time system. When the program is invoked with at least one command line argument, Debug Malloc reports 145 allocations, adding to the previous number the allocations performed by the program using C++'s *new* operator. You should also note that allocations made in the *insert* method of the *Store* class are reported by the following traceback:

```
Allocation #5: 0x20122FF8
        Allocation traceback:
        0x2012301C  .__start
        0x20123020  main
        0x20123024  insert__5StoreFPv
        0x20123028
        0x2012302C  malloc
```

The identifier *insert__5StoreFPv* is the name of the *insert* method of *Store* class after it's been mangled by the C++ compiler.

A Polak
System Engineer
APS Israel (Israel)

Workload Manager

INTRODUCTION

Resource management has become a fashionable thing to do. This is largely a result of the current trend towards server consolidation. Consolidation means big servers handling mixed workloads and, in such environments, resource and workload management can bring real benefits by ensuring that system and application response time and throughput meet the performance objectives stipulated by the business (or the Service Level Agreement). At the same time, resource management may help in smoothing out some of the peaks and troughs in demand, thus enabling the same job mix to run on a smaller, more optimized system.

This article reviews Workload Manager (WLM), a tool that's available with AIX 4.3.3, and its use for commercial applications and server consolidation. Before looking in detail at WLM, I introduce some of the issues and principles involved in workload and resource management. Afterwards, I discus other areas and applications where resource management is performed outside the control of the operating system.

WHAT IS RESOURCE MANAGEMENT?

What is understood by the term 'resource management' depends somewhat on the background of the reader. For the purpose of this article, the term is defined as the ability to allocate and/or limit computing resources (CPU, memory, disk, I/O, and network bandwidth) either manually or automatically, applying the restriction to specific computing operations, programs, or groups of programs in order to meet the performance and throughput objectives of the organization.

Resource management presents a number of challenges. Once the performance objectives of an application are specified, the system must then decide how to run each application given that:

1 The resource requirements of individual applications are generally not known in advance

2 Toady's tendency towards consolidation results in larger and more varied workloads, so that applications end up competing for the same resources.

WHY MANAGE RESOURCES?

At first sight, resource management seems like a 'nice-to-have' feature rather than something that offers real value. This is because one of the primary roles of an operating systems is to manage and optimize resource utilization. Thus, the scheduler allocates CPU resources to each process or thread, and the virtual memory manager resolves conflicts for use of real-memory. On closer inspection, however, it can be seen that resource management can offer real benefits for the IT service provider. In particular, optimization of resources means that the existing computing infrastructure can be used to its maximum and peaks and troughs in demand can be absorbed gracefully.

The object of today's drive towards consolidated application and data servers is to run larger and more varied workloads on fewer platforms while, at the same time, adapting to new demands and workloads. Experience with computer systems provides system administrators with a number of heuristic approaches to addressing different performance problems. Unfortunately, these procedures seldom lead to exactly the right solution and, even when do, the solutions apply only to simple configurations. By contrast, the key object of resource management is that the system manages itself in such a way that it meets the stipulated business objectives without additional human intervention.

MIXED WORKLOADS

With server consolidation comes a larger variety of workloads. This means more types of job executing both at any one time and over a period of time. For example, during the day the server may support on-

line transactions, internal communications (e-mail, news, workflow, intranet), an Internet Web server, and resource planning software. At night the overhead from transaction-based applications diminishes and the system may run batch jobs, such as data mining queries, database restructuring, and/or back-up operations. Weekly or monthly, the payroll application is executed and, perhaps, an application is run to produce management reports showing current trends etc.

In such environments, resource management comes into its own by managing workloads and resources, thereby ensuring continued operation and the maintenance of response-time objectives.

Resource management is closely related to workload management. In fact, the boundary between the two is somewhat blurred, as it's impossible to manage resources without having an impact on the relative progress of applications that are running. Resource and workload management simplify performance management by attempting to use all computing resources effectively. The system must decide which resources to allocate to which job, deciding how much of those resources to allocate and for how long.

The goals of resource and workload management are to:

- Optimize computing resource use.
- Provide acceptable response times and throughput for key applications.
- Adapt to the fluctuating loads typical of today's consolidated, multi-application systems.

WHAT CONSUMES RESOURCES?

A number of terms that are in common use are, nevertheless, often ill-defined. These include *process*, *job*, *workload*, *task*, *client*, and *user*. In addition, their meaning often depends on the context in which they are used.

From AIX's point of view, processes and their threads consume CPU time and memory, wait in the run queue, and open files. Processes may be grouped together via a script, which is often run in the background.

This collection of related processes, which may execute in parallel or sequentially, is sometimes referred to as a *job*, a term that comes from the mainframe world.

For databases, which are implemented by AIX processes, threads, and files, it is whatever is at the end of an open database connection that generates load, and hence affects resource consumption. This may be a client workstation in a two-tier architecture or the middleware comprising the second tier in a three-tier architecture. In the former, the database can easily relate clients to resource consumption and may then apply resource management and control. However, in a three-tier architecture, clients are frequently multiplexed over a smaller number of database connections. This hides the end-user from the database, which can no longer distinguish between on-line requests and background or batch jobs.

In three-tier architectures it is only the transaction monitor that is in a position to equate generated load with users. The transaction monitor can see all clients on the first tier and has knowledge of the load generated by each incoming request. However, today's transaction monitors are concerned only with execution time, which relates indirectly to CPU occupation. They do not take into account the consumption of other system resources necessary to service a given request.

MANAGED RESOURCES

There are a number of different types of computing resource that may be managed by system administrators and resource management applications; these are described below.

- *CPU occupation*

  The instruments that may be used to control and limit an application's CPU utilization include:

  – *Percentage CPU occupation*

    This stipulates the minimum and/or maximum CPU occupancy, expressed as a percentage. An example of the

way this is used is to ensure that a non-priority application does not impede the functioning of a high-priority one while ensuring that the low-priority application is not completely blocked.

– *Maximum CPU occupation*

  This is the upper limit to the CPU time that an application may consume before being forcibly stopped or suspended by the operating system. This is used to prevent an application from 'running-away' in an infinite loop, or a batch job from running over into the TP window, etc.

– *Relative priority*

  By assigning different priorities to each application, it is possible to provide rapid response times for those that require it. In very high workload environments, a priority-based schema may starve lower priority applications of resources to the extent that they make little or no forward progress. This can be remedied by also specifying the minimum percentage CPU utilization for low-priority applications.

– *Allowable CPUs*

  In multiprocessor systems, it is possible to limit an application to executing on a subset of the available CPUs. In this way, certain CPUs may be dedicated to specific applications.

• *Memory occupation*

  Limiting the amount of real-memory that an application may use prevents real memory being monopolized by a small number of applications that have large working sets. This enables the system to ensure the response-time maxima of all running applications.

• *Disk space occupation*

  While the cost of on-line storage continues to fall, disk space remains a finite resource. Controlling the amount of disk space each user may consume is one of the easiest resource management

mechanisms that can be implemented. Nearly all of today's computing systems offer some form of control on disk usage.

- *I/O utilization*

  Disk space utilization is not only a crude form of resource management, it also has little effect on system performance. A more subtle, yet more powerful, control mechanism is to limit the number of I/O operations and/or the amount of data transferred between memory and I/O over a given period (typically one second). This mechanism prevents processes being blocked and waiting for I/O to complete as a result of another application consuming all available bandwidth.

- *Network occupation*

  This is an important yet difficult parameter to control. This is because a network connection implies a client-server architecture of some sort. So, if resource management is running on the server, it can limit the amount of traffic originating from the server, but has no effect on the amount being generated by clients. A number of research projects are working on this problem and some prototype implementations have been deployed.

RESOURCE MANAGEMENT IN AIX

The most common approach to workload management is to do nothing and leave resource allocation to the operating system. AIX uses the following mechanisms for resource management:

- The scheduler selects jobs to run when a CPU becomes available. Both users and system administrators may set job priorities so that certain applications receive preferential treatment.
- Threads or processes may be 'bound' to one or more CPUs so that key applications run on reserved processors and are guaranteed response times. The bind policy should be used cautiously – used on its own, it does not reserve CPU time.
- The Virtual Memory Manager (VMM) controls both real and

virtual memory usage – when the swap space starts to become insufficient, it automatically kills a few jobs.

- Disk quotas and user limits may be used to control the maximum resources that a user may consume at a given moment.
- User resource limits may be specified by the **ulimit** command. This command can be used to control file size, data size, physical memory usage, stack size, and CPU time.

As the operating system is the only entity that has a complete and detailed view of the state of the system, this approach has its merits. Even though AIX's external controls are somewhat simplistic, it still implements a level of resource management. This means that, for example, a long-running and CPU-intensive application will see its relative priority decrease, allowing other jobs access to the CPU. This technique is both attractive and simple, as no additional interfaces or controls are needed. This approach may be completely adequate for smaller systems that host a small number of fairly homogeneous jobs. In short, if this model satisfies the business requirements, then it is the recommended solution.

Hence, for light and moderately loaded systems, the resource management provided for by the kernel is usually sufficient. Problems may arise under heavy load, when the operating system must juggle with available resources and conflicting requests. In such circumstances, the work performed by the operating system increases and the kernel itself places an increasing demand on the system resources, which further degrades performance. When real memory is in short supply the kernel may spend time swapping pages into and out of paging space, which is known as thrashing. In such circumstances, additional workload and resource management is required to guarantee continued service and reasonable response times.

So, as the workload and its diversity increases, so AIX's ability to manage its own resources will reach its limits, and performance (response-time and throughput) will tend to become more volatile. This is where WLM comes in.

AIX 4.3.3'S WORKLOAD MANAGER (WLM)

AIX's WLM (Workload Manager) is bundled with AIX 4.3.3. It provides additional resource and workload management, giving system administrators closer control over the amount of CPU and physical memory that may be consumed by different applications. Later releases will also manage paging space and I/O. Though the new tool is called 'workload manager', it expresses priorities and resource limits in terms of the underlying system, and thus is more of a resource manager than a workload manager. The use of WLM is optional and, when WLM is disabled, the system behaves exactly as it did in earlier releases of AIX. When configured, WLM consumes considerably less than 1% of CPU.

WLM manages resources within a single AIX system. It does not manage resources across the nodes of a cluster, though it may run on each individual node. Much of the work on WLM, including its implementation, has been and is being carried out by Bull in Grenoble, France.

WLM CONCEPTS

WLM introduces a number of concepts new to AIX and, hence, some new vocabulary also. WLM manages groups of processes called *classes*. Resource allocation is controlled by *shares* and *limits*. Classes may be grouped and prioritised in collections known as *tiers.*

Classes

When WLM is activated, the system administrator defines a number of named classes. These may be specified at boot time or dynamically when the system is running. As processes are created, they are placed in a particular class. Placement in a given class is based on the user-id (UID), group-id (GID), or application name (the full path name of the executable binary), the choice being based on the first criterion matched. Processes created using the *fork()* system call are placed in the same class as their parent process, though placement is re-evaluated if *exec()* is called.

Each class has a name and an optional description. The class name may have up to 16 characters. There are two pre-defined classes:

- *Default class*

  This is a catch-all class for non-root processes. All classes that are not assigned to a specific class are placed in the *Default* class. Resource allocation for this class can be modified from its initial values, as is the case with all other classes.

- *System class*

  This is the corresponding class for root (system) processes that haven't been allocated to a specific class. By default, the *System* class has a minimum memory limit of 1%.

Despite these catch-all classes, some processes and resources cannot be managed by WLM. Two examples of this are shared memory pages and processes that are already running when WLM is started. These processes and resources are placed in a pseudo-class called *Unclassified*. WLM does not manage processes in the *Unclassified* class.

Tiers

To help prioritize the different classes, each class is placed in a *tier*. A tier can be thought of as a priority level, with available levels ranging from '0' to '9' ('0' having the highest priority). Resources are first allocated to a class in tier '0'. Any remaining resources are then allocated to classes (and hence processes) in tier '1', etc.

Shares, targets, and limits

Each class is allocated a number of *shares* for each managed resource. Resource allocation is expressed as the ratio of the number of shares allocated to a class divided by the total number of shares allocated to all active classes in the same tier. Hence, in a tier that contains three classes (*A*, *B*, and *C*) with 10, 15, and 25 shares respectively, the classes receive 20%, 30% and 50% of a given resource respectively. Should a class *D* with 50 shares become active in the same tier, then the resource allocations for A, B, C, and D would become 10%, 15%, 25%, and 50% respectively.

A class may be allocated between 1 and 65,535 shares. Only active classes (that is, classes with active processes) are used in resource allocation calculations. The resource consumption of a class is the sum of that of all processes in the class. So, if there are ten processes in a class called *database*, and the *database* class has a CPU limit of 20%, then the sum of the CPU usage of all ten processes in the class can't exceed 20%. This means that one process can consume 20% of CPU and the remaining nine 0%, or two processes can consume 10% each with the remaining eight being idle, etc.

Shares are known as *soft limits* or *targets*, as they indicate the desired resource allocation at any given time. When a new class becomes active, the desired resource allocation percentages take a few seconds to be achieved.

WLM also supports *hard limits*. A maximum hard limit is never exceeded and a minimum hard limit is – if possible – always met or exceeded (the reason for the 'if possible' qualifier is that a class may not be able to meet a minimum CPU occupation limit – for example, if a class contains only applications that are memory-bound and need more working memory than is allocated or available, then the applications are unable to run for lack of memory). Limits are expressed as percentage values, the range being '0' to '100'. The default minimum value is '0' and the default maximum is '100'.

Shares and limits may be modified without restarting WLM (at least, in later releases of the product – the initial one does not support the dynamic addition or suppression of classes). Hard limits should be used with caution – inappropriate values can wreak havoc on system performance.

WLM performs a number of sanity checks on values used for shares and limits. For example, if the calculated percentage resource allocation is less than the minimum resource limit, WLM uses the minimum as the target, and a similar check is performed on the maximum value.

HOW WLM WORKS

While WLM focuses on processes and groups of processes, the AIX scheduler operates on threads. So, how is the difference reconciled?

WLM maintains an internal priority for each class. As it monitors the resource consumption of processes in a class, it modifies the class's priority. It then adds the class's priority to the AIX scheduler priority of all the threads in all the processes in the class. Thus, when resource usage is below target, the priorities are increased, and when it's above target, they are decreased.

WLM does not control real-time or fixed priority processes. Only processes with *SCHED_OTHER* threads are managed by WLM. In addition *wait* threads and those that call *plock()* are not managed either.

As far as control of memory allocation is concerned, processes in classes that are at or below their minimum memory limit will not have pages stolen from them (except by other processes in the same class or in another class that is also below its minimum memory limit). Pages pinned in memory are not assigned to a class.

WLM CONFIGURATION

WLM configuration is a key phase that requires careful consideration as it directly impacts system and application response time under the varying loads. WLM, which acts on scheduling priorities, as described above, is thus known as a 'fair-share scheduler'.

Before implementing resource management, you must translate the performance requirements of your business applications into resource usage limits under different loads. This translation process requires a good understanding of the various factors that influence system performance and the characteristics of the applications to be controlled.

Once the analysis phase is complete, WLM allocates a share of the available resources to each application or group of applications. WLM uses this share to modify AIX's priorities dynamically during the execution of each application, so that each group receives its allocated share.

Assuming the necessary privileges, a user or system administrator can move running processes from one class to another. This results in memory usage being assigned to the new class and CPU usage being recalculated.

WLM ADMINISTRATION

There are three complementary ways of administering WLM:

- WebSM – the Web-based system administration tool. To go directly to the WLM page, use the command:

  ```
  wsm wlm
  ```

- **smit** – the **tty** or X-Windows system administration tool. Use the command:

  ```
  smit wlm
  ```

- You can use the Command Line Interface (CLI) and edit the relevant configuration files.

smit and WebSM provide a graphical and less error-prone way of creating and modifying files and executing the commands than is available via manual configuration.

The files that contain WLM configuration information are known as 'WLM property files'. It's possible to have several sets of property files for different circumstances – for example, one set may favour interactive users during office hours, while a second ensures that nightly batch jobs run to completion before the next morning's on-line users arrive at their desks.

Property files are usually placed in directories rooted at */etc/wlm*. */etc/wlm/current*, which is a symbolic link that points to the directory that actually contains the current set of property files (for example, the current class definitions are held in */etc/wlm/current/classes*). This mechanism provides a flexible way of changing resource allocation rules periodically by, say, using a shell or Perl script that is executed by **cron**, changes the symbolic link, and reinitializes WLM.

WLM PROPERTY FILES

The following list of WLM's property files may help you understand their role in WLM's administration. Each property set comprises five flat text files that may be edited using a standard text editor. The files are:

- *description*

  A short description of the WLM configuration. For example ‘nightly batch config’ or ‘prototype office-hours config’.

- *classes*

  A description of each class and (optionally) the tier to which it belongs. If the tier is not specified, the class defaults to ‘tier-0’ (the highest priority tier).

- *shares*

  The memory and CPU shares allocated to each class.

- *limits*

  Each class’s minimum and maximum hard limits for CPU and memory. The default values are ‘0’ and ‘100’ respectively.

- *rules*

  The selection rules for determining the class into which to place newly started applications. The order in which rules are stated is important as selection is based on the first match.

shares, *limits*, and *rules* may be modified while WLM is running. However, if *classes* is changed, WLM must be restarted, and all processes that are currently running are put in the *Unclassified* class. This means that, in the first release of WLM, a reboot is required after changing *classes*. This limitation is lifted in later releases, which allow the assigning of existing processes to classes and the modification of classes without a reboot.

This article concludes in next month’s issue of AIX Update.

Jez Wain
System Architect
Bull SA (France)

Scheduling software distribution scripts

I have written a tool for software distribution to AIX servers that we have successfully used for more than two years. I would like to share this program with other *AIX Update* readers. The version of the program included in this article is the 'automatic' one, which can be used to schedule distribution scripts to run at a pre-defined date and time. We generally schedule our distribution scripts to run at night, so as not to overload the network.

This article includes three scripts: **schedule**, **swd3_oto**, and **ftrvrg**. **shedule.sh** is the script that's used to manage software distribution scripts. **schedule** schedules **swd3_oto** for distribution, and **swd3_oto** calls **ftrvrg** to carry out the actual file transfer. **ftrvrg** processes only one file per invocation, sending it to a remote AIX server. It's up to **swd3_oto** to handle the transfer of more than one file to multiple remote AIX servers.

The script **swd3_oto.sh** copies compressed tar files (*kisim9000.Z*, etc) to remote AIX servers. **swd3_oto.sh** takes three parameters:

- *plan name* – the name of AIX servers
- *cargo list* – the data files to be distributed
- *rcp limit* – the maximum parallel rcp command.

Note the use of the continuation character, '➤', in the code below to indicate a formatting line break that doesn't appear in the original source code.

SCHEDULE.SH

```
#!/bin/ksh
# By Adnan Akbas Dec'97 Ver.1.5
# This script schedules software distribution scripts
# to run on a desired date and time.

rev=$(tput rev)
nor=$(tput sgr0)
bol=$(tput bold)
```

```
# script definition

 scr1=swd3_oto.sh
 scr2=script2.sh
 scr3=script3.sh
 scr4=script4.sh
 scr5=script5.sh

function warnmsg {
  print "\a\n$bol$*$nor"
}

function errormsg {
  print "\a\n$rev $* $nor\a"
}

# Create a uniquely named cargo file.

function file_name_creator {
   'date' | read a b c d e f
   file_name_suffix=${b}${c}${d}
   cargo_list=/u/ftpuser/fileuse/cargo_lists/cargo_${file_name_suffix}
}

# Check the content of the cargo file.

function cargo_contents_controler {
   while read cargo_item ; do
    if [[ ${cargo_item} != "" ]] ; then
       if [[ ! -s /u/ftpuser/${cargo_item} ]] ; then
          print "File name error.!!!"
          print "File: ${cargo_item} does not exist."
          print "Check file names and try again."
          exit
       fi
       chmod 664 /u/ftpuser/${cargo_item}
    else
       print "There are empty rows in the cargo list or the cargo
              ➤  list is empty."
       print "Check file names and try again."
       exit
    fi
   done < $cargo_list
}

clear
################################################################################
# 1. get script name.
################################################################################
echo " AUTOMATIC START "
```

```
echo " ================ "
echo " 1.) Aktarim.....................swd3_oto.sh "
echo " 2.) script2.....................script2.sh "
echo " 3.) script3.....................script3.sh "
echo " 4.) script4.....................script4.sh "
echo " 5.) script5.....................script5.sh "
print; print
print -n  Enter your choice ...
read choice
case $choice in
    1 ) scr=$scr1 ;;
    2 ) scr=$scr2 ;;
    3 ) scr=$scr3 ;;
    4 ) scr=$scr4 ;;
    5 ) scr=$scr5 ;;
    * ) errormsg " Wrong Choice! "
        exit 1 ;;
esac
plan=plan_all
let max_rcp_limit=60
wc -l /u/ftpuser/plan_all | read subelist_max junk
###############################################################################
# 2. get plan name.
###############################################################################
clear
print \\nDefault plan = $plan , enter 1 to change and ENTER to accept.
read chk_plan
if [[ $chk_plan -eq 1 ]] ; then
   print \\n rPlan names that can be used :\\n
   ls plan*
   print \\n
   print Plan Description: \\n
   print plan_all : All Servers                \\n
   read plan?"Enter the distribution plan name:"
   if [[ ! -s $plan ]] ; then
      print \\nYou've entered an invalid plan name. Please check again.
      print \\nAfter checking your plan, restart the script.\\n
      exit 2
   fi # check if plan exists and is not empty.
fi # check whether plan has changed.
###############################################################################
# 3. get maximum rcp number at the same time
###############################################################################
clear
print \\nDefault maximum rcp number= $max_rcp_limit, to change press 1,
      ➤  to accept press ENTER.
read chk_rcp
if [[ $chk_rcp -eq 1 ]] ; then
   read max_rcp_limit?"Enter prallel rcp limit :"
   if [[ $max_rcp_limit -lt 1 || $max_rcp_limit -gt $subelist_max ]] ;
```

```
   ➤  then
      print \\nParallel rcp limit must be between 1 and $subelist_max
      print \\nCheck rcp number and restart script.\\n
      exit 3
   fi # check whether rcp is 'meaningless'.
fi # check whether rcp has changed.
plan_name=$plan
rcp_number=$max_rcp_limit
###############################################################################
# 4. get cargo list from user.
###############################################################################
clear
print -r "Type file names that will be distributed to the cargo list."
print -r "Exit file after you have saved."
print "Press ENTER to continue."
read chk
file_name_creator
vi $cargo_list
cargo_contents_controler
cargo=${cargo_list##/*/}
###############################################################################
# 5. get date from user
###############################################################################
clear
print;
read isim?" Enter your name (for log): "
print;
print " Now: `date` "
print;
print  " Enter the date you want to run your script: ..."
print;
read ay?" Month ( jan - dec ): "
read gun?" Day ( 01 - 31 )    : "
read saat?" Hour ( 00 -23 )    : "
read dakika?" Minute ( 00 - 59 ) : "
print;
###############################################################################
# 6. Checking if the date entered is correct.
###############################################################################
if [[ $gun -le 31 ]] && [[ $gun -ge 1 ]] && [[ $saat -le 23 ]] &&
   ➤  [[ $saat -ge n
   case "$ay" in
   ( jan | feb | mar | apr | may | jun | jul | aug | sep | oct |
     ➤  nov | dec )
        clear
        print -n "Time entered : $gun $ay ${saat}:${dakika}:00
                  ➤  $year ..." ;;
   ( * )
        errormsg " date error ---> $gun $ay ${saat}:${dakika}:00
                   ➤  $year "
```

```
        print; print
        print "After you have checked the date, restart the script."
        exit 4
   esac
else
   errormsg " date error ---> $gun $ay ${saat}:${dakika}:00 $year "
   print; print;
   print "After you have checked the date start the script again."
   exit 5
fi
print; print;
#############################################################################
# 8. Final Check.
#############################################################################
print \\nPlan name= $plan , Maximum rcp= $max_rcp_limit\\n
print Files to be distributed : \\n
cat $cargo_list
print \\nIf you comfirm press ENTER , to exit press 1
read chk_final
if [[ $chk_final -eq 1 ]] ; then
   exit 6
fi
print;
#############################################################################
# 9. schedule the script with its parameters.
#############################################################################
echo /u/ftpuser/bin/$scr $plan_name $rcp_number $cargo_list | at
➤  ${saat}${dakik
#############################################################################
# 10. recording jobs into a file.
#############################################################################
if [[ $? = 0 ]] ; then
   print "OTO  $isim --> $scr  SCHEDULED TO  '$gun $ay
          ➤  ${saat}:${dakika}:00 $yeg
   print "               using  $plan_name  rcp/rexec=$rcp_number
          ➤  $cargo " >> g
print >> /u/ftpuser/fileuse/process.log
fi
print; print; print;
print "Press ENTER ....."
read
```

SWD3_OTO.SH

```
#!/bin/ksh
# SWD versiyon 2.1
# by Adnan Akbas
# auto-start version
#############################################################################
```

```
# parameter initilization
################################################################################
sleep 5
plan_all_curr=1
#
wc -l /u/ftpuser/plan_all | read subelist_max junk
#
done_branch=0
#
plan=${1}
#
max_rcp_limit=${2}
curr_rcp_num=0
#
cargo_list=${3}
#
'date' | read a b c d e f
file_name_suffix=${b}${c}${d}
#
cat $plan > /u/ftpuser/fileuse/tmp/plan_${file_name_suffix}
plan=/u/ftpuser/fileuse/tmp/plan_${file_name_suffix}
#
cat $cargo_list >
➤  /u/ftpuser/fileuse/cargo_lists/cargo_${file_name_suffix}
cargo_list=/u/ftpuser/fileuse/cargo_lists/cargo_${file_name_suffix}
wc -l $cargo_list | read c_l_size junk
#
function dist
{
       let cnt=0
       while read cargo_item ; do  # cargo list control
             print " $sube_kodu / $done_branch --> $cargo_item
                     ➤  tekrar : $don / "
             /home/ftpuser/bin/ftrvrg.sh $sube_kodu $cargo_item >
             ➤  /dev/null &
             wait
             kisim_no=${cargo_item#kisim}
             grep tamamlandixx
             ➤  /u/ftpuser/log/${sube_kodu}_${kisim_no}.log |gre2
             if [[ $t2 -eq 1 ]] ; then
                 let cnt+=1
             fi
       done < $cargo_list  # cargo list control
       if [[ $cnt -eq $c_l_size ]] ; then
          print ${sube_kodu} >>
          ➤  /u/ftpuser/fileuse/tmp/full_ok_${file_name_suff}
       fi
}

function control
```

```
{
 let curr=$plan_all_curr
#
    tail +$curr plan_all | read sube_kodu
    let done_branch+=1
    grep $sube_kodu $plan > /dev/null # plan control start
    if [[ $? = 0 ]] ;then  # plan control
       ps -ef -o ruser,comm | grep ftpuser | grep rcp | wc -l |
                 ➤  read curr_rcp_nt
       while [[ $curr_rcp_num -ge $max_rcp_limit ]] ; do
        sleep  40
        ps -ef -o ruser,comm | grep ftpuser | grep rcp | wc -l |
                  ➤  read curr_rcp_m
       done # rcp control finish
       desicion=do_it
    fi # plan control finish
}

###############################################################################
# MAIN PROGRAM
###############################################################################
tekrar=2
let don=1
while [[ $don -le $tekrar ]] ; do
  while [[ $done_branch -lt $subelist_max ]] ; do
     desicion=do_not_it
     control
     if [[ $desicion = "do_it" ]] ; then
         dist &
     fi # desicion
     let plan_all_curr+=1
  done
  let don+=1
  #
  done_branch=0
  #
  plan_all_curr=1
  #
  ###############################################################################
  # Finished AIX servers subtracted from the plan.
  ###############################################################################
  sort -n -u -o $plan $plan
  if [[ -a /u/ftpuser/fileuse/tmp/full_ok_${file_name_suffix} ]] ; then
     sort -n -u -o  /u/ftpuser/fileuse/tmp/full_ok_${file_name_suffix}
                    ➤  /u/ftpuse
r/fileuse/tmp/full_ok_${file_name_suffix}

comm -23 $plan /u/ftpuser/fileuse/tmp/full_ok_${file_name_suffix} >
➤  /u/ftpuser/fileuse/tmp/temp_${file_name_suffix}
```

```
     cat /u/ftpuser/fileuse/tmp/temp_${file_name_suffix}  > $plan
  fi
  wc -l $plan | read kalan junk
  if [[ $kalan -lt 5 ]] ; then
     exit 0
  fi
  #
done # tekrarlar
```

FTRVRG.SH

```
#!/bin/ksh
# Sends a file to a AIX server.
#
# By Adnan Akbas.
#
usage='ftrvrg.sh server_name file_name'
version='0.09'

function trycmd {
  for trycmdcntr in 1 2 0 ; do
    "$@"
    let ret=$?
    if (( ret != 0 )) ; then
      print "\a'$@' : error code=$ret"
      if [[ $trycmdcntr = 0 ]] ; then
        return 1
      else
        sleep $((trycmdcntr*trycmdcntr*1))
        continue
      fi
    fi
    print "'$@' tamamlandixx."
    break
  done
  return 0
}

function transfer {
for try in 1 2 ; do
  print "\n $bn ${try}. deneme $(date)"
  trycmd rcp -p $fn AN$bn:$fn 2>&1
  let ret=$?
  if (( ret != 0 )) ; then
    continue
  fi
  nfn=$yad${fn#kisim}
  print "\n $bn  Bitis $(date)"
```

```
  exit 0
done
print '\a*** COPY OPERATION FAILED!!!\n\a'
return $ret
}
################################################################################
# main program
################################################################################
if (( $# > 3 )) ; then
  print "Kullanim: $usage" >&2
  exit 2
fi
bn=${1:?"server name not given"}
fn=${2:?"file name not given"}
yad=${3:-part}
if [[ ! -r $fn ]] ; then
  print "filename can not be read"
  exit 3
fi
################################################################################
#control
################################################################################
kisim_no=${fn#kisim}
logfile=/u/ftpuser/log/${bn}_${fn#kisim}
logfile=${logfile}.log

grep tamamlandixx /u/ftpuser/log/${bn}_${kisim_no}.log | grep -c
➤  AN${bn} | read2
if [ $t2 = 0 ] ; then
################################################################################
# checks if there is a rcp working for that server.
################################################################################
   ps -ef | grep ftpuser | grep rcp | grep AN${bn} > /dev/null 2>&1
   ret=$?
   if [[ $ret != 0 ]] ; then
      transfer >$logfile 2>&1 </dev/null
      exit $?
   fi
   exit 5
else
   print $(date) was sent before >>$logfile
   exit 7
fi
```

Adnan Akbas
System programmer
Pamukbank (Turkey)

Memory Usage Report

INTRODUCTION

Memory Usage Report (**mur**) is a shell script that extracts information from the output of the **ps vg** command and uses it to prepare reports on real and virtual memory usage by text and data segments for each process running in the system. The accuracy of these reports thus depend entirely on that of the output produced by **ps vg**. The script can be run from any account, though the option to display the amount of real memory in the system requires *root* privileges, as this depends on the execution of **bootinfo -r** command. The script allows the user to print the report and to select a process id from a list of current processes, where applicable. The script also allows users to monitor the growth of memory usage by a particular process during a given time interval.

The script has options to display the following:

- Real memory in system
- Virtual memory (swap space) in system
- Real memory for text and data segments for one process
- Memory acquisition for a process
- Real memory for text and data segments for all processes
- Swap space for text and data segments for a process
- Swap space for text and data segments for all processes.

MUR.SH

```
################################################################################
# Name       : mur.sh (memory usage report)
#
# Description: The script displays various memory-related statistics.
#
# Notes      : 1. The script contains following functions:
```

```
#
#                  o InitializeVariables
#                  o HandleInterrupt
#                  o PrintSpoolFile
#                  o MoveCursor
#                  o DisplayMessage
#                  o DisplayMenu
#                  o FormatUnderscores
#                  o ProcessOption
#                  o RootUser
#                  o ProcessExit
#                  o main
#                  o GetRealMemory
#                  o GetSwapSpaceDetails
#                  o GetRealMemoryForTextAndDataForSpecificProcess
#                  o MonitorSpecificProcessMemoryAquisition
#                  o GetRealMemoryForTextAndDataForAllProcess
#                  o GetVirtualMemoryForSpecificProcess
#                  o GetVirtualMemoryForAllProcess
#
#               2. The script relies on the output from ps vg command and
#                  therefore, the results displayed here are as accurate as
#                  the output from ps vg command.
################################################################################
################################################################################
# Name       : InitializeVariables
#
# Description: The function initializes all required variables.
################################################################################
InitializeVariables()
{
# define locations
TEMP_FILE="/tmp/mur_$$.tmp"
TEMP_FILE_1="/tmp/mur_$$_1.tmp"
TEMP_FILE_2="/tmp/mur_$$_2.tmp"
REPORT_FILE="/tmp/mur_$$.dat"
#
# escape sequences
ESC="\0033["
RVON=_[7m           # revrese video on
RVOFF=_[27m         # reverse video off
BOLDON=_[1m         # bold on
BOLDOFF=_[22m       # bold off
BON=_[5m            # blinking on
BOFF=_[25m          # blinking off
#
# define Menu title
MUR="${RVON}Memory Usage Report${RVOFF}"
```

```
#
# define exit codes
SEC=0
FEC=1
TRUE=0
FALSE=1
SLEEP_DURATION=4    # no of seconds allowed for sleep command
ERROR="${RVON}${BON}mur.sh:ERROR:${BOFF}"
INFO="${RVON}mur.sh:INFO: "
#
# message
WORKING="Working.........${RVOFF}"
INTERRUPT="Program interrupted ! Quitting early${RVOFF}"
INVALID_OPTION="Invalid entry ${RVOFF}"
INVALID_ENTRY="Invalid entry ${RVOFF}"
PRINT_OK="Successfully submitted the print job${RVOFF}"
PRINT_NOT_OK="Failed to submit the print job${RVOFF}"
NOT_NUMERIC="Value must be numeric${RVOFF}"
OSERROR="\${ERR_MSG}${RVOFF}"
POLLING="Starting to poll the process${RVOFF}"
INVALID_PID="Process id \$PID is invalid${RVOFF}"
#
# define signals
SIGTSTP=18 ; export SIGTSTP  # ctrl-z
SIGHUP=1  ; export SIGHUP   # when session disconnected
SIGINT=2  ; export SIGINT   # ctrl-c
SIGTERM=15 ; export SIGTERM  # kill command
}
################################################################################
# Name       : HandleInterrupt
#
# Description: The function calls ProcessExit.
################################################################################
HandleInterrupt ()
{
DisplayMessage I "${INTERRUPT}"
ProcessExit $FEC
}
################################################################################
# Name       : MoveCursor
#
# Input      : Y and X coordinates
#
# Returns    : None
#
# Description: Moves the cursor to the required location (Y, X).
#
# Notes      : 1. Must be run in ksh for print to work. Also, print
```

```
#                 must be used to move the cursor because echo does
#                 not seem to work.
################################################################################
MoveCursor ( )
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
YCOR=$1
XCOR=$2
 echo   "${ESC}${YCOR};${XCOR}H"
}
################################################################################
# Name       : DisplayMessage
#
# Description: The function displays message
#
# Input      : 1. Message type (E = Error, I = Informative)
#              2. Error Code as defined in DefineMessages ().
################################################################################
DisplayMessage ( )
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
MESSAGE_TYPE=$1
MESSAGE_TEXT=`eval echo $2`
MoveCursor 24 1
if [ "${MESSAGE_TYPE}" = "E" ]
then
  echo "`eval echo ${ERROR}`${MESSAGE_TEXT}\c"
else
  echo "`eval echo ${INFO}`${MESSAGE_TEXT}\c"
fi
sleep ${SLEEP_DURATION}
return ${TRUE}
}
################################################################################
# Name       : FormatUnderscores
#
# Description: The function assigns appropriate number of underscores(=)
#              to the variable UNDERSCORE to be used in conjunction with a
#              header.
#
# Input      : Line containing the header
################################################################################
FormatUnderscores ()
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
#
# assign parameter
LINE="$1"
```

```
#
# initialize UNDERSCORE
UNDERSCORE=
#
# initialize index
IND=1
#
# get no of characters in $LINE
NO_CHARS=`echo "$LINE" | wc -c`
#
# subtract the carriage return
NO_CHARS=`expr $NO_CHARS - 1`
while [ "$IND" -le "$NO_CHARS" ]
do
 UNDERSCORE="${UNDERSCORE}="
 IND=`expr $IND + 1`
done
}
###############################################################################################
# Name       : PrintSpoolFile
#
# Description: Prints the named file
#
# Input      : File name to be printed
###############################################################################################
PrintSpoolFile ( )
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
FILE_TO_BE_PRINTED=$1
#
# print file
while true
do
 clear
 echo "Do you wish to print the output file (Y/N)?:\c"
 read REPLY
 case $REPLY  in
   n|N ) return $TRUE ;;
   y|Y ) break ;;
   *  )  : ;;
 esac
done
#
# get printer name
while true
do
  clear
  echo "Enter printer name for lp command(q to quit):\c"
```

```
  read PRINTER
  case $PRINTER in
   "" )  : ;;
   q|Q) break ;;
    * ) lp -d$PRINTER ${FILE_TO_BE_PRINTED} > /dev/null 2>&1 ;
      if [ $? -eq 0 ]
      then
        DisplayMessage I "${PRINT_OK}" ;
        break ;
      else
        DisplayMessage E "${PRINT_NOT_OK}" ;
      fi ;;
  esac
done
}
###############################################################################
# Name       : DisplayMenu
#
# Description: The function displays the menu.
###############################################################################
DisplayMenu ()
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
clear
echo  "    ##########################################"
echo  "    #    $MUR                               #"
echo  "    #                                       #"
echo  "    #  5. Real Memory in System             #"
echo  "    #                                       #"
echo  "    #  10. Virtual Memory(swap space)       #"
echo  "    #      in System                        #"
echo  "    #  15. Real Memory for Text & Data      #"
echo  "    #      Segment for a Process            #"
echo  "    #  20. Monitor Memory Aquisition        #"
echo  "    #      for a Process                    #"
echo  "    #  25. Real Memory for Text & Data      #"
echo  "    #      Segment for All Processes        #"
echo  "    #  30. Swap Space for Text & Data       #"
echo  "    #      Segment for a Process            #"
echo  "    #  35. Swap Space for Text & Data       #"
echo  "    #      Segment for All Processes        #"
echo  "    #                                       #"
echo  "    #  99. Exit                             #"
echo  "    ##########################################"
echo  "      Enter Option --->\c"
read OPTION
}
###############################################################################
```

```
# Name       : ProcessOption
#
# Description: The function processes menu option.
################################################################################
ProcessOption()
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
case $OPTION  in
    5) GetRealMemory ;;
   10) GetSwapSpaceDetails;;
   15) GetRealMemoryForTextAndDataForSpecificProcess;;
   20) MonitorSpecificProcessMemoryAquisition;;
   25) GetRealMemoryForTextAndDataForAllProcess;;
   30) GetVirtualMemoryForSpecificProcess;;
   35) GetVirtualMemoryForAllProcess;;
   99) clear; ProcessExit $SEC;;
   * ) DisplayMessage E "${INVALID_OPTION}"
esac
}
################################################################################
# Name       : DisplayListOfValues
#
# Description: The function displays list of values for system processes.
#              and extracts the process id from the selected record.
################################################################################
DisplayListOfValues ( )
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
PID=  # value selected by user
echo "      List of values for System Processes " > ${TEMP_FILE}
echo "      ==================================== " >> ${TEMP_FILE}
echo "      Select Value by Deleting Line and Saving File\n" >>\
                          ${TEMP_FILE}
ps -eaf >> ${TEMP_FILE}
cp ${TEMP_FILE} ${TEMP_FILE_1}
view ${TEMP_FILE}
PID=`diff ${TEMP_FILE} ${TEMP_FILE_1} | tail -1 | awk {'print $3'}`
COMMAND=`diff ${TEMP_FILE} ${TEMP_FILE_1} | tail -1 | cut -c 50-80 | \
                  cut -d' ' -f1`
}
################################################################################
# Name       : GetProcessId
#
# Description: The function gets a process id from the user.
################################################################################
GetProcessId ()
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
```

```
while true
do
 clear
 echo "Enter Process Id (l=list of values q=quit):\c"
 read PID
 case $PID in
  l|L ) DisplayListOfValues ;
     if [ "${PID}" = "" ]
     then
          : ;
     else
         break ;
     fi ;;
  "" ) DisplayMessage E "${INVALID_ENTRY}" ;;
   * ) break ;;
 esac
done
}
################################################################################
# Name       : GetCommand
#
# Description: The function gets the command for a specific process.
#
# Input      : Process Id
#
# Returns    : $TRUE if a command is associated with the process id
#              $FALSE if no command is associated with the process id
################################################################################
GetCommand ()
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
P_PID="$1"
COMMAND=`ps -eaf |  grep "${P_PID}" | grep -v "grep" | cut -c 48-80`
if [ "${COMMAND}" = "" ]
then
   return $FALSE
else
   return $TRUE
fi
}
################################################################################
# Name       : GetRealMemory
#
# Description: The function obtains and displays amount of real memory for
#              the system.
#
# Notes      : The function requires root privilege because of invocation
#              of bootinfo -r command.
```

```
################################################################################
GetRealMemory ()
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
DATETIME=`date "+%d/%m/%Y at %H:%M:%S"`
clear
MIKB=`su -c "bootinfo -r" 2> ${TEMP_FILE}`  # memory in kilobytes
if [ $? -ne 0 ]
then
  ERR_MSG=`cat ${TEMP_FILE}`
  DisplayMessage E "${OSERROR}"
  return $FALSE
fi
MIBY=`expr $MIKB \* 1024` # memory in bytes
HEADER="Amount of Real Memory in System on ${DATETIME}"
FormatUnderscores "${HEADER}"
echo "   $HEADER"   > ${REPORT_FILE}
echo "   $UNDERSCORE" >> ${REPORT_FILE}
echo "$MIBY bytes "   >> ${REPORT_FILE}
#
# view the file
#
view ${REPORT_FILE}
#
# print the file
#
PrintSpoolFile ${REPORT_FILE}
}
################################################################################
# Name       : GetSwapSpaceDetails
#
# Description: The function obtains the swap space details.
################################################################################
GetSwapSpaceDetails ()
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
DATETIME=`date "+%d/%m/%Y at %H:%M:%S"`
lsps -a | sed 1d | awk {'printf("%-19s%-12s%-10s",$2,$4,$6)'} > ${TEMP_FILE}
HEADER="Swap Space Details in System on ${DATETIME}"
FormatUnderscores "${HEADER}"
echo "   $HEADER"   > ${REPORT_FILE}
echo "   $UNDERSCORE\n" >> ${REPORT_FILE}
echo "Physical Volume  Size    Active" >> ${REPORT_FILE}
cat ${TEMP_FILE} >> ${REPORT_FILE}
#
# view the file
#
view ${REPORT_FILE}
```

```
#
# print the file
#
PrintSpoolFile ${REPORT_FILE}
}
###############################################################################
# Name       : MonitorSpecificProcessMemoryAquisition
#
# Description: The function is used to monitor real memory size for both
#              text and data segment for a specific process at a given time
#              interval for a period of time.
###############################################################################
MonitorSpecificProcessMemoryAquisition ()
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
TIME_INTERVAL=""  # required time interval
NO_POLLS=     # number of iterations
#
# get a process id
GetProcessId
#
# get polling time interval
while true
do
  clear
  echo "Enter Time Interval, in seconds, for Polling the Process:\c"
  read TIME_INTERVAL
  case $TIME_INTERVAL in
    "" ) DisplayMessage E "${INVALID_ENTRY}" ;;
    *  ) if ( [ `expr $TIME_INTERVAL + 0` -eq $TIME_INTERVAL ]) \
                              > /dev/null 2>&1
       then
          break ;
       else
          DisplayMessage E "${INVALID_ENTRY}" ;
       fi;;
 esac
done
#
# get number of polls
while true
do
  clear
  echo "Enter Number of Polls:\c"
  read NO_POLLS
  case $NO_POLLS in
    "" ) DisplayMessage E "${INVALID_ENTRY}" ;;
    *  ) if ( [ `expr $NO_POLLS + 0` -eq $NO_POLLS ]) > /dev/null 2>&1
```

```
       then
          break ;
       else
          DisplayMessage E "${INVALID_ENTRY}" ;
       fi;;
 esac
done
DisplayMessage I "${POLLING}" ;
DATETIME=`date "+%d/%m/%Y at %H:%M:%S"`
HEADER="Real Memory Size(kbytes) for Text and Data Segment on ${DATETIME}"
FormatUnderscores "${HEADER}"
echo "$HEADER"   > ${REPORT_FILE}
echo "$UNDERSCORE" >> ${REPORT_FILE}
HEADER="Process Id = $PID"
FormatUnderscores "${HEADER}"
echo "       $HEADER"   >> ${REPORT_FILE}
echo "       $UNDERSCORE" >> ${REPORT_FILE}
echo "Command            Text Segment Data Segment">>${REPORT_FILE}
NO_CYCLE=0
while [ $NO_CYCLE -lt $NO_POLLS  ]
do
  #
  # extract value for text and data segment togethers (RSS)
  TEXTDATASEG=`ps vg | grep " $PID " | grep -v "grep" | awk {'print $7'}`
  #
  # extract value for text segment only (TRS)
  TEXTSEG=`ps vg | grep " $PID " | grep -v "grep" | awk {'print $10'}`
  #
  # calculate value for data segment only
  DATASEG=`expr $TEXTDATASEG - $TEXTSEG`
  echo "$COMMAND $TEXTSEG $DATASEG" | \
         awk {'printf("%-30s%-14s%-10s\n",$1,$2,$3)'} > ${TEMP_FILE}
  cat ${TEMP_FILE} >> ${REPORT_FILE}
  #
  # view the file
  view ${REPORT_FILE}
  sleep $TIME_INTERVAL
  NO_CYCLE=`expr $NO_CYCLE + 1`
done
#
# print the file
PrintSpoolFile ${REPORT_FILE}
}
################################################################################
# Name       : GetRealMemoryForTextAndDataForSpecificProcess
#
# Description: The function obtains the real memory size for both text and
#              data segment for a specific process.
```

```
################################################################################
GetRealMemoryForTextAndDataForSpecificProcess()
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
while true
do
  clear
  echo "Enter Process Id (l=list of values q=quit):\c"
  read PID
  case $PID in
    q|Q ) return $FALSE ;;
    l|L ) DisplayListOfValues ;
       if [ "${PID}" = "" ]
       then
          : ;
       else
          break ;
       fi ;;
    "" ) DisplayMessage E "${INVALID_ENTRY}" ;;
    *  ) if ! GetCommand $PID
       then
         DisplayMessage E "${INVALID_PID}" ;
       else
           break ;
       fi ;;
 esac
done
#
# extract value for text and data segment togethers (RSS)
TEXTDATASEG=`ps vg | grep " $PID " | grep -v "grep" | awk {'print $7'}`
#
# extract value for text segment only (TRS)
TEXTSEG=`ps vg | grep " $PID " | grep -v "grep" | awk {'print $10'}`
#
# calculate value for data segment only
DATASEG=`expr $TEXTDATASEG - $TEXTSEG`
echo "$COMMAND $TEXTSEG $DATASEG" | \
         awk {'printf("%-30s%-14s%-10s",$1,$2,$3)'} > ${TEMP_FILE}
DATETIME=`date "+%d/%m/%Y at %H:%M:%S"`
HEADER="Real Memory Size(kbytes) for Text and Data Segment on ${DATETIME}"
FormatUnderscores "${HEADER}"
echo "$HEADER"   > ${REPORT_FILE}
echo "$UNDERSCORE" >> ${REPORT_FILE}
HEADER="Process Id = $PID"
FormatUnderscores "${HEADER}"
echo "       $HEADER"   >> ${REPORT_FILE}
echo "       $UNDERSCORE" >> ${REPORT_FILE}
echo "Command            Text Segment Data Segment">> ${REPORT_FILE}
```

```
cat ${TEMP_FILE} >> ${REPORT_FILE}
#
# view the file
view ${REPORT_FILE}
#
# print the file
PrintSpoolFile ${REPORT_FILE}
}
###############################################################################
# Name       : GetRealMemoryForTextAndDataForAllProcess
#
# Description: The function obtains the real memory size for both text and
#              data segment for all processes in the system.
###############################################################################
GetRealMemoryForTextAndDataForAllProcess ()
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
TOT_TEXTSEG=0   # accumulative total for TEXT
TOT_DATASEG=0   # accumulative total for DATA
TOT_MEM_IN_USE=0 # total memory in use by all processes
DisplayMessage I "${WORKING}"
#
# get process and memory details
ps vg | sed 1D | while read LINE
do
 #
 # extract value for text and data segment together for each process
 TEXTDATASEG=`echo "${LINE}" | awk {'print $7'}`
 if [ "${TEXTDATASEG}" = "" ]
 then
   #
   # daft record
   continue
 fi
 #
 # extract value for text segment only
 TEXTSEG=`echo "${LINE}" | awk {'print $10'}`
 #
 # calculate value for data segment only
 DATASEG=`expr $TEXTDATASEG - $TEXTSEG`
 #
 # accumulate these values
 TOT_TEXTSEG=`expr $TOT_TEXTSEG + $TEXTSEG`
 TOT_DATASEG=`expr $TOT_DATASEG + $DATASEG`
done
TOT_MEM_IN_USE=`expr $TOT_TEXTSEG + $TOT_DATASEG`
echo "${TOT_TEXTSEG} ${TOT_DATASEG}" | \
         awk {'printf("Text=%-14s Data=%-14s",$1,$2)'} > ${TEMP_FILE}
```

```
echo "Total memory in use = ${TOT_MEM_IN_USE}" >> ${TEMP_FILE}
DATETIME=`date "+%d/%m/%Y at %H:%M:%S"`
HEADER="Memory Size(kbytes) for All Text and Data Segments on ${DATETIME}"
FormatUnderscores "${HEADER}"
echo "$HEADER"   > ${REPORT_FILE}
echo "$UNDERSCORE" >> ${REPORT_FILE}
cat ${TEMP_FILE} >> ${REPORT_FILE}
#
# view the file
view ${REPORT_FILE}
#
# print the file
PrintSpoolFile ${REPORT_FILE}
}
##############################################################################################
# Name       : GetVirtualMemoryForSpecificProcess
#
# Description: The function obtains displays the virtual memory size
#              for a specific process.
##############################################################################################
GetVirtualMemoryForSpecificProcess ()
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
while true
do
  clear
  echo "Enter Process Id (l=list of values q=quit):\c"
  read PID
  case $PID in
    q|Q ) return $FALSE ;;
    l|L ) DisplayListOfValues ;
       if [ "${PID}" = "" ]
       then
          : ;
       else
          break ;
       fi ;;
    "" ) DisplayMessage E "${INVALID_ENTRY}" ;;
    *  ) break ;;
 esac
done
#
# extract value for SIZE
SIZE=`ps vg | grep " $PID " | grep -v "grep" | awk {'print $6'}`
echo "$COMMAND $SIZE" | \
         awk {'printf("%-30s%-14s",$1,$2)'} > ${TEMP_FILE}
DATETIME=`date "+%d/%m/%Y at %H:%M:%S"`
HEADER="Virtual Memory Size(kbytes) on ${DATETIME}"
```

```
FormatUnderscores "${HEADER}"
echo "$HEADER"   > ${REPORT_FILE}
echo "$UNDERSCORE" >> ${REPORT_FILE}
HEADER="Process Id = $PID"
FormatUnderscores "${HEADER}"
echo "       $HEADER"     >> ${REPORT_FILE}
echo "       $UNDERSCORE"   >> ${REPORT_FILE}
echo "Command            Size">> ${REPORT_FILE}
cat ${TEMP_FILE} >> ${REPORT_FILE}
#
# view the file
view ${REPORT_FILE}
#
# print the file
PrintSpoolFile ${REPORT_FILE}
}
################################################################################
# Name       : GetVirtualMemoryForAllProcess
#
# Description: The function obtains and displays the total virtual memory
#              size for all processes in the system.
################################################################################
GetVirtualMemoryForAllProcess ()
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
TOTSIZE=0     # total virtual memory size in use
DisplayMessage I "${WORKING}"
#
# extract value for SIZE
ps vg | sed 1D | while read LINE
do
 SIZE=`echo "${LINE}" | awk {'print $6'}`
 if [ "${SIZE}" = "" ]
 then
   #
   # daft record
   #
   continue
 fi
 TOTSIZE=`expr $TOTSIZE + $SIZE`
done
DATETIME=`date "+%d/%m/%Y at %H:%M:%S"`
HEADER="Used Virtual Memory Size(kbytes) on ${DATETIME}"
FormatUnderscores "${HEADER}"
echo "$HEADER"   > ${REPORT_FILE}
echo "$UNDERSCORE" >> ${REPORT_FILE}
echo "${TOTSIZE}" >> ${REPORT_FILE}
#
# view the file
```

```
view ${REPORT_FILE}
#
# print the file
PrintSpoolFile ${REPORT_FILE}
}
###############################################################################
# Name       : ProcessExit
#
# Description: The function removes any temporary files and makes a graceful
#              exit.
#
# Input      : Exit Code
###############################################################################
ProcessExit ()
{
EXIT_CODE="$1"
clear
rm -f ${REPORT_FILE}
rm -f ${TEMP_FILE_1}
exit ${EXIT_CODE}
}
###############################################################################
# Name       : main
#
# Description: The function invokes all other functions.
#
# Notes      : 1. The function invokes following functions:
#                 o InitializeVariables
#                 o DisplayMenu
#                 o ProcessOption
###############################################################################
main ()
{
InitializeVariables
while true
do
  DisplayMenu
  ProcessOption
done
}
#
# invoke main
main
```

Arif Zaman
DBA/Developer
High-Tech Software Ltd (UK)

IBM has announced Commerce Integrator Version 1.1, which simplifies the integration of Net.Commerce or WebSphere Commerce Suite with back-end systems.

Version 1.1 requires Net.Commerce 3.2 or WebSphere Commerce Suite 4.1. There are two versions: Commerce Integrator Pro runs on AIX, NT, and Solaris, while the Start version is available only on NT. The Pro version targets sites that need to access information from multiple back-end applications.

Out now, prices for the Pro version start at US$50,000.

The company also announced a four-port 10/100-BaseTX Ethernet PCI Adapter for the RS/6000, providing four Ethernet ports on a single PCI slot. The adapter is compatible with both 32 and 64 bit PCI slots and supports AIX Version 4.3.3 or later. Out now, it costs US$1,500.

Finally, the company announced Network Client Manager, which provides server-based management for a broad range of client platforms and applications. These include PCs, NCs, and other thin-client hardware, such as point-of-sale devices. While the first release of the product runs on NT only, AIX, OS/2 and Linux versions are expected in the near future. Prices for these versions are not available now.

For further information, contact your local IBM representative.

* * *

Continuus Software has announced Continuus/CM Version 5.0, the latest version of the company's task-based change management software, promising extended scalability and comprehensive migration facilities.

Version 5.0 handles both content and software change management employing a 'second generation' methodology for automating tracking and communication of changes across distributed development teams. It's designed to simplify the change management process with a scalable repository and a team-oriented workflow approach to development.

Out now, versions are available for AIX and also Windows 95, 98, NT, Solaris, HP-UX, Compaq TRU64 Unix on Alpha, Siemens SINIX, and SGI IRIX. Prices start at US$19,500.

For further information contact:
Continuus Software Corporation, 108 Pacifica, 2nd Floor, Irvine, CA 92618, USA
Tel: +1 949 453 2200
Fax: +1 949 453 2276
Web: http://www.continuus.com

Continuus Software, 6 Bracknell Beeches, Old Bracknell Lane, Bracknell, Berkshire RG12 7BW, UK
Tel: +44 1344 788100
Fax: +44 1344 788111