Redbooks Paper

Erich Amrehn
Ronald Annuss
Arwed Tschoeke

Accounting and monitoring for z/VM Linux guest machines

Introduction

This IBM® Redpaper is intended for clients working with Linux for zSeries® under z/VM® in a proof of concept or pilot phase. We illustrate a basic accounting and system monitoring implementation to collect and correlate data from both z/VM and Linux.

The first part of this paper illustrates how to set up basic accounting. The second part introduces monitoring concepts that allow clients to estimate resources required for production. We monitor single Linux systems as well as the overall system resource usage. Correlation of both z/VM and Linux monitoring data helps locate resource bottlenecks and identify the cause.

This Redpaper is based on z/VM Version 4.4 and SLES Version 8 for the Linux systems. Linux installations use the default settings.The concepts and examples in this paper can be implemented using standard base z/VM and Linux utilities, but we provide links to licensed z/VM and ISV products that may be used as well.

System setup

Although the examples in this Redpaper can be implemented using standard z/VM and Linux tools, we strongly recommend considering supported, licensed products when deploying in a production environment. We provide information about accounting and monitoring products in “Commercial accounting products” on page 10 and “Commercial monitoring products” on page 30.

Within z/VM, we use the IUCV *ACCOUNT and *MONITOR services (as well as the IBM provided user DISKACNT) in conjunction with the user defined ACCSRV and MONSRV virtual machines to process z/VM data. Routines for collecting and transforming the data are written in REXX. z/VM data is exported via NFS and accessed by the Linux-based report generator.

Within Linux, we use the sysstat utilities to continuously log system activity. The Apache HTTP server is used to generate reports. The CGI scripts on the report system are written in Perl. Gnuplot is used to create the graphs.

The system configuration is depicted in Figure 1.

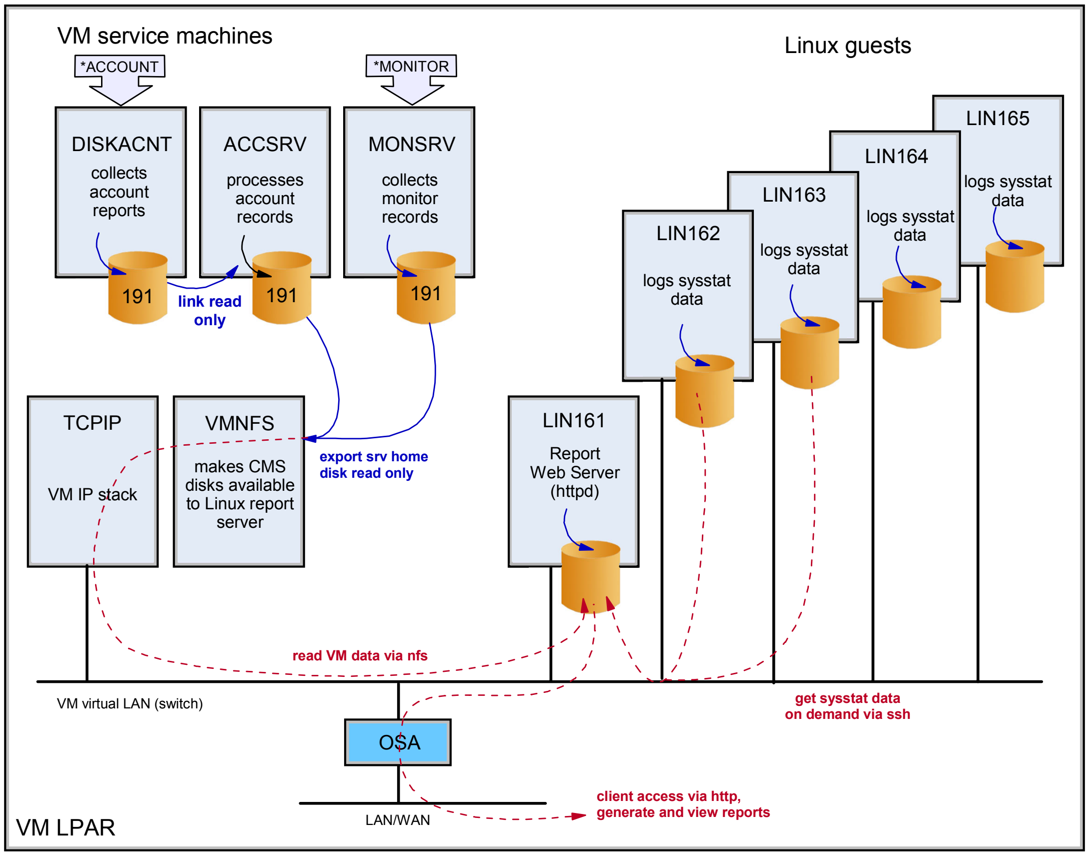


Figure 1 System configuration

The solid (blue) lines indicate permanent connections and dataflow. The dotted (red) lines show the dataflow initiated only for report generation.

Accounting z/VM Linux guests

Nowadays many clients are looking for ways to consolidate their distributed systems onto zSeries using Linux and z/VM. One of the advantages of this consolidation scenario is the ability to establish a precise accounting system. Since IT costs are always a key factor when considering server consolidation, it is also necessary to establish a charge back method for the consolidation platform where resources are shared between multiple users.

We show how to utilize the z/VM accounting facility to build a simple accounting system for Linux guests running under z/VM.

z/VM accounting records

z/VM has an accounting system service to collect accounting information for guest machines. If the accounting is enabled, CP collects resource usage information about guests and stores that data in memory. Service machines connect via IUCV (*ACCOUNT) to the CP accounting facility to retrieve accounting records. Account records are in binary format and can be processed by programs or scripts. We give a short description of available accounting records here. For more detailed information refer to *z/VM V4R4.0 CP Planning and Administration*, SC24-6043.

z/VM 4.4 accounting records for the following types of resource usage are available:

- Virtual machine resource usage – Type 01
- Dedicated devices –Type 02
- Temporary disk space – Type 03
- Journaling – Types 04, 05, 06, 08 and 0I
- SNA/CCS- Type 07
- Inter-System Facility for Communications – Type 09
- Logging changes to a user's privilege – Type 0A
- Virtual disk in storage space – Type 0B
- Network data transmissions – Type 0C

In this Redpaper, we use record types 01 and 0C.

Type 01 records contain CPU time information. Records of this type are created when a user logs on, logs off, and when the CP `ACNT` command is issued. Figure 1 shows the type 01 record fields we use.

Table 1 Used fields from type 01 account records

Column	Description
1–8	User ID
17–28	Date and time of accounting (in mmddyyhhmmss format)
29–32	Number of seconds connected to CP
33–36	Milliseconds of processor time used, including time for supervisor functions
37–40	Milliseconds of virtual CPU time used
80	Accounting record identification code

Type 0C records contain guest network usage information for virtual network adapters. Type 0C records are written when the CP `ACNT` command is issued. Figure 2 shows the type 0C record fields we use.

Table 2 Used fields from type 0C account records

Column	Description
1–8	User ID
51–58	Bytes Sent
59–66	Bytes Received
80	Accounting record identification code

z/VM accounting model

To understand how z/VM accounting operates, we briefly discuss accounting principles. This discussion is based on a simple accounting model using only a subset of the available accounting information.

Within z/VM, all resources are shared among guests. z/VM accounting can determine how much of a shared resource is used by a single guest in a selected time frame. This information can be used to charge the user or organization owning the guest. Shared resources can include:

- CPU
- Memory
- Network devices
- I/O adapters
- DASD space

In our accounting model, we consider only CPU and network usage for Linux guests. For simplicity, we assume all other resources are static and calculate percentages for each guest.

As an example, consider z/VM running in an LPAR with the following configuration:

- Two Integrated Facility for Linux (IFL) processors
- 2 GB of memory
- 100 GB of attached Direct Access Storage Device (DASD) storage
- One shared OSA-Express GbE network adapter

A Linux guest is defined to z/VM with the following configuration:

- One virtual processor
- 512 MB of virtual storage
- 15 GB of DASD storage
- One virtual network interface connected to a guest LAN or virtual switch

Charges to the Linux guest might be calculated as follows:

- **DASD charges**
  The 15 GB of DASD storage can be billed at a fix cost per month to the user.
- **Virtual storage charges**
  If the sum total of virtual storage defined to all Linux guests is assumed to be 6 GB, the user could be charged a percentage based on the virtual machine size. In the example, this translates to $512\ \mathrm{MB} / 6\ \mathrm{GB}\ =\ 8.3\%$ of virtual storage. From the 2 GB of real memory available to z/VM, this translates to 170 MB per month charged to the guest.
- **CPU and network charges**
  CPU and network charges can be calculated based on actual usage as recorded by z/VM accounting.

Although this model may be not accurate enough for production, it may be useful to estimate resource usage for a pilot or proof of concept project. A more sophisticated accounting model could include measured data other resource usage, including:

- **Actual consumed memory**
  For example, these charges could be based on average value of the working set size for a guest and paging activity data.
- **The I/O channel utilization or amount of I/O data transferred**

In addition, costs may be varied based on the time a resource is used (day or night).

Set up a service machine to collect account records

To collect accounting information, we use two service guest machines:

- **DISKACNT**
  This IBM-provided example service machine continuously runs a program to retrieve and store account records to its A-disk when the CP **ACNT** command is issued.
- **ACCSRV**
  We create this service machine to process the accounting records. ACCSRV has read-only access to the DISKACNT A-disk. ACCSRV is automatically started through AUTOLOG and operates in disconnected mode. The service machine periodically executes a REXX script via WAKEUP. This in turn executes the CP **ACNT ALL** command, which then processes account records to extract actual Linux guest resource usage values.

Note: Account records for z/VM guest LANs are created only when both:

- The guest LAN is defined with the option ACCOUNT ON.
- The NETA option is set in the user directory entry.

To ensure that the z/VM accounting is always enabled, we add the following line to the PROFILE EXEC of the AUTOLOG1 user:

```
'CP RECORDING ACCOUNT ON'
```

The ACCSRV user

The ACCSRV user must have class A privileges in order to issue the CP **ACNT** command. We set the secondary console to the operator because ACCSRV always operates disconnected. The A-disk must be large enough to hold accounting data: 50 cylinder in this example. Example 1 shows the directory entry for the ACCSRV user.

Example 1 ACCSRV user directory entry

```
USER ACCSRV   AUTOONLY 32M   32M AG
 INCLUDE IBMDFLT
 AUTOLOG AUTOLOG1 OP1 MAINT
 MACHINE XA
 LINK MAINT 493 493 RR
 LINK DISKACNT 191 192 RR
 MDISK 191 3390 0750 0050 430W01  MRV READ     WRITE    MULTIPLE
```

Note: For security reasons, we prevent manual logon to this virtual machine by specifying AUTOONLY as the password. This service machine is started only via AUTOLOG.

Account record list

Account records received by DISKACNT contain data for all z/VM guests. To process only records for Linux users, we create a ACC LIST file. This contains a list of all guests for which accounting records are to be saved. Other guests such as TCPIP or VMNFS can be included as well. A sample ACC LIST file is shown in Example 2.

Example 2 ACC LIST file contents

```
LIN161
LIN162
LIN163
LIN164
LIN165
```

Account record processing

Account record processing is implemented with three REXX scripts:

- **PROFILE EXEC**
  PROFILE EXEC contains an endless loop executed nightly using the WAKEUP utility (with the wakeup time defined in the file WAKEUP TIMES file). The CP **ACNT ALL** command is issued within the loop and the A-disk DISKACNT user is re-accessed to reflect the changed account record file contents.
- **READACC EXEC**
  Next, the script READACC EXEC is called to process newly created account records. Resource usage values for users listed in ACC LIST file are stored in a daily account file.
- **CLEAN EXEC**
  Finally, the CLEAN EXEC script is called to purge old account files. The loop is then halted via WAKEUP. This creates a daily summary file with accounting information for Linux guests. These summary files are available to the Linux reporting system using NFS.

PROFILE EXEC

Example 3 shows the PROFILE EXEC script of our ACCSRV machine.

Example 3 PROFILE EXEC script

```
/* PROFILE EXEC - forever loop processes account records each night */
'CP SET RUN ON'
'CP SP CONS * START'
'ACC 493 U'

DO FOREVER
  'wakeup (file(wakeup times)'
  'REL 192'
  'ACNT ALL'
  'SLEEP 1 MIN'
  'ACC 192 D'
  'SLEEP 1 MIN'
  'READACC'
  'SLEEP 1 MIN'
  'CLEAN'
  'SLEEP 10 MIN'
END
```

The file WAKEUP TIMES file contains the following line:

```
ALL     23:55:00   01/01/01
```

READACC EXEC

The READACC REXX script reads and processes the newest account file from the A-disk of the DISKACNT user. Processing proceeds by selecting accounting records for selected Linux guests and by translating those records into human readable form. Results are stored in a file with the current day as its extension.

The account file produced by DISKACNT contains binary records for every logged on guest when the CP **ACNT ALL** command is executed. In addition, logon and logoff event records are present. The record file can contain multiple records for each user. READACC EXEC finds the desired records using REXX pipe commands. The major stages and flow of these pipes are shown in Figure 2 on page 7.

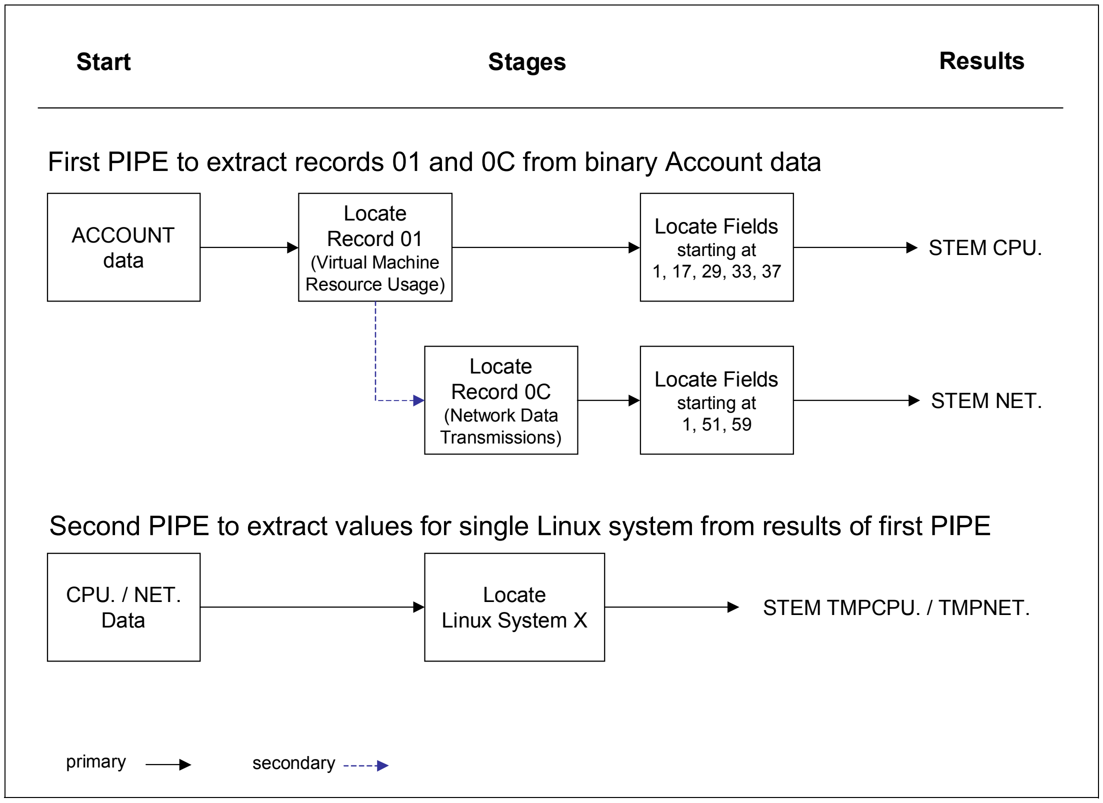

Start
Stages
Results
First PIPE to extract records 01 and 0C from binary Account data
ACCOUNT data
Locate Record 01 (Virtual Machine Resource Usage)
Locate Fields starting at 1, 17, 29, 33, 37
STEM CPU.
Locate Record 0C (Network Data Transmissions)
Locate Fields starting at 1, 51, 59
STEM NET.
Second PIPE to extract values for single Linux system from results of first PIPE
CPU. / NET. Data
Locate Linux System X
STEM TMPCPU. / TMPNET.
primary
secondary

Figure 2 Extraction pipe flow

An extract of the READACC script shown in Example 4 depicts the relevant pipe commands. The complete file with comments is listed in “ACCSRV” on page 32.

Example 4 Account processing pipe commands

```
/* extract the records types 1 and C from account file */
'pipe (end ?) <' input,
'| a: locate 80 /1/',
'| SPECS 1-8 1 17-22 10 29-32 C2X 20 33-36 C2X 30 37-40 C2X 40',
'| stem cpu.',
'? a:',
'| locate 80 /C/',
'| SPECS 1-8 1 51-58 C2X 10 59-66 C2X 30',
'| stem net.'

. . .
/* filter out the values for a single guest from the list (list.i) */
  'pipe stem cpu.',
  '| locate 1-8 /'list.i'/',
  '| stem tmpcpu.'
  'pipe stem net.',
  '| locate 1-8 /'list.i'/',
  '| stem tmpnet.'
```

The values for CPU time and network transmissions are accumulated for each user. then they are written to a file with the current date as file type (extension) such as:

```
ACC $A102803 A
```

Each line in the log file has the format:

```
<UserID>_<Date>_<Connect>_<TCPU>_<VCPU>_<Bytes rec.>_<Bytes send>
```

Where:

UserID	z/VM user ID as listed in ACC LIST
Date	Date the account record was created (in MMDDYY format)
Connect	Time the user logged on (in hhhh:mm:ss format)
TCPU	Accumulated total CPU time (user time + hypervisor overhead)
VCPU	Virtual CPU time (user time only) (in hhhh:mm:ss format)
Bytes rec	Accumulated network data received (expressed as long integer)
Bytes send	Accumulated network data sent (expressed as long integer)

Fields are separated by spaces.

CLEAN EXEC

The CLEAN EXEC script shown in Figure 5 is used for housekeeping purposes. Old account files are purged. The number of retained files is defined in the variable **keep**.

Example 5 CLEAN EXEC script

```
/* CLEAN EXEC – housekeeping routine to purge old account records */

keep = 40

'PIPE (endchar ?)',
'cms listfile acc $a*',
'| sort 10.8',
'| a: take last 'keep,
'| stem dummy.',
'?',
'a: | stem old.'

do i=1 to old.0
  call lineout , 'cleaning old files 'old.i
  'erase 'old.i
end
```

Note: During operation, the DISKACNT user's A-disk files rapidly fills up. In this example, only one cylinder is allocated (enough space for approximately 20-30 days worth of account data). If you plan to run this procedure for an extended period, you should plan to include the DISKACNT user in the housekeeping procedure or to increase the size of its A-disk.

Web page to generate reports online

Next, we explain setup of a simple Web front end to generate reports from the collected accounting data. We configure an Apache HTTP server to act as a report server on one of the Linux systems (LIN161).

Access to the accounting data

The A-disk of the ACCSRV user is exported by VMNFS. To access data from the Linux reporting system, we permanently mount the A-disk of the ACCSRV user over NFS. The mount point and options are specified in /etc/fstab as:

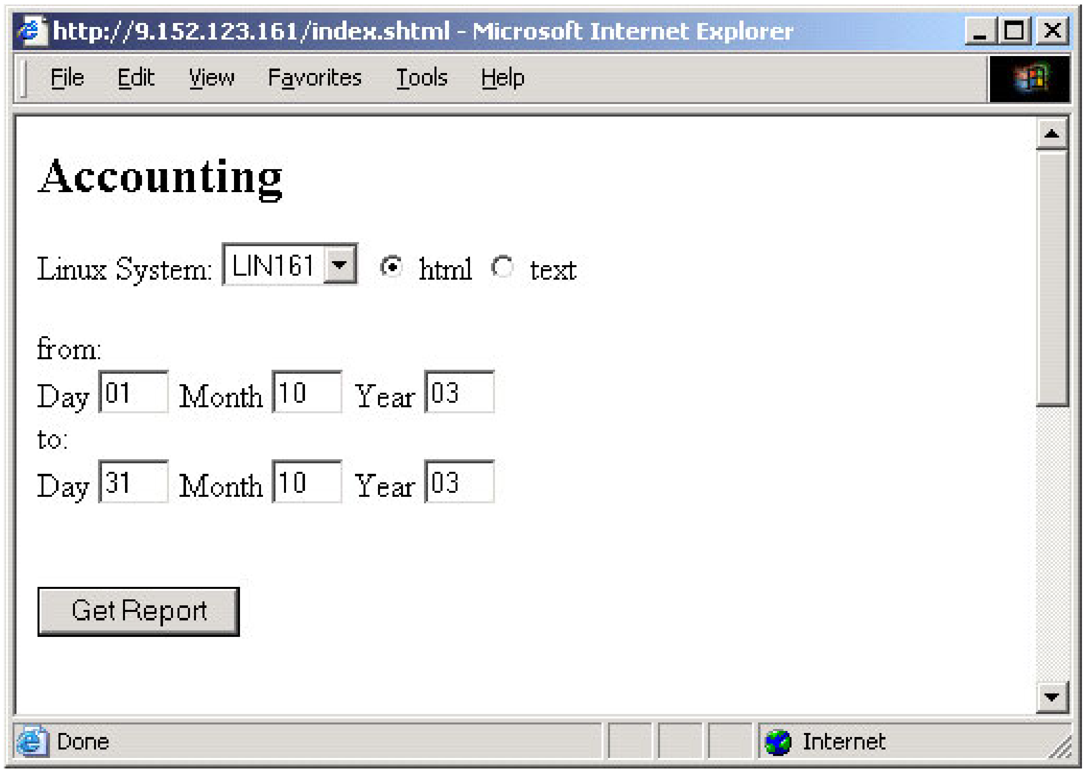
```
10.1.1.1:/ACCSRV,userid=monsrv,password=XXXX /mnt/accsrv nfs ro,auto  0 0
```

See “VMNFS” on page 32 for complete VMNFS setup instructions.

Important: The z/VM user ID and password must be included in the /etc/fstab file. When the **`mount`** command is executed, these are values are displayed on the terminal. In order to secure the user ID and password, user access to the server should be restricted to the system administrator.

Report front end

On the Linux (LIN161) reporting server, we implement a simple HTML interface to the accounting report using the Apache Web server. With the interface, a user may select the guests and time frames for an accounting report. Figure 3 shows a screen shot of the HTML front end.


http://9.152.123.161/index.shtml - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Get Report
Done
Internet

Accounting

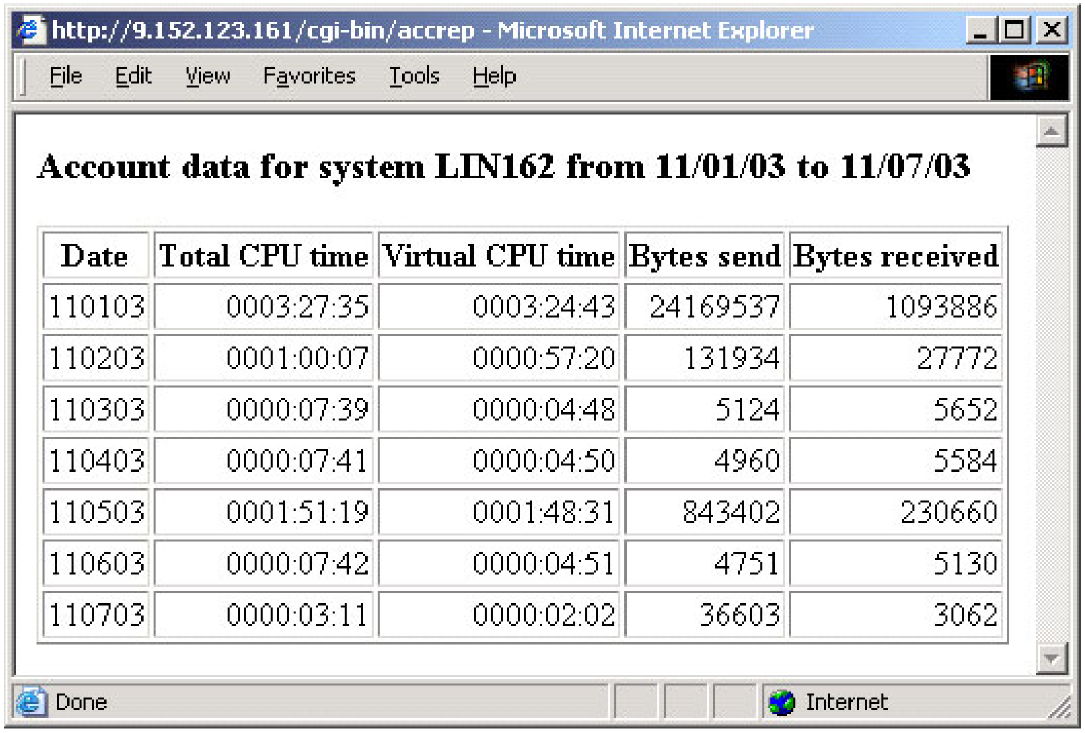
Linux System: LIN161 [✓ html] text

from:
Day 01 Month 10 Year 03
to:
Day 31 Month 10 Year 03

Figure 3 Accounting report front end

Both HTML and text reports may be generated by the CGI report generator (using the **`accrep`** command). The HTML file and the **`accrep`** Perl CGI script are listed in “Linux report system” on page 36.

A sample HTML account report is shown in Figure 4 on page 10.


http://9.152.123.161/cgi-bin/accrep - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
Internet

Account data for system LIN162 from 11/01/03 to 11/07/03

Date	Total CPU time	Virtual CPU time	Bytes send	Bytes received
110103	0003:27:35	0003:24:43	24169537	1093886
110203	0001:00:07	0000:57:20	131934	27772
110303	0000:07:39	0000:04:48	5124	5652
110403	0000:07:41	0000:04:50	4960	5584
110503	0001:51:19	0001:48:31	843402	230660
110603	0000:07:42	0000:04:51	4751	5130
110703	0000:03:11	0000:02:02	36603	3062

Figure 4 HTML account report

A sample text output is shown in Example 6.

Example 6 CPU and memory usage report

```
Account data for system LIN161 from 11/01/03 to 11/07/03

Date     Total CPU time   Virtual CPU time      Bytes send  Bytes received

110103       0000:05:53         0000:04:29          634444         4371053
110203       0000:05:42         0000:04:18          794743         4261515
110303       0000:04:33         0000:03:10            5308            5836
110403       0000:04:34         0000:03:10            4960            5584
110503       0000:06:54         0000:05:31          822395         7045657
110603       0000:04:41         0000:03:18           62243          176833
110703       0000:02:00         0000:01:26           43941          171477
```

Text output can be further processed by other scripts.

Commercial accounting products

This section provides a short overview of some available commercial accounting products that support for z/VM accounting data.

CA Unicenter VM: Account

Unicenter VM: Account is a resource accounting, reporting and capacity management system for the VM environment that provides:

- Project accounting, software-package accounting and minidisk and Shared File System (SFS) accounting
- Online querying

- Reporting and invoicing
- Budget control facilities
- Real-time collecting, validating, processing, and reporting on VM accounting data to allow accurate cost allocation and recovery
- Preservation of data integrity and protection against lost accounting data
- Workload balancing through discounts and surcharges
- An audit trail for all CMS user activity, allowing long-term trend analysis
- Full-screen maintenance of the Unicenter VM: Account project, client, account number, and rate information tables

Unicenter VM: Account continuously collects VM accounting information, performing real-time validation and costing. Queries and reports provide up-to-the-minute information. Details can be found at:

`http://www.ca.com`

CIMS Resource Accounting

CIMS consolidates a wide variety of accounting data from multiple systems. CIMS centralizes processing of accounting data and converts this data to a common format for costing and reporting. Output may be used for static or Web-based reporting or as a feed to existing financial systems. The CIMS Chargeback System supports data from CIMS Data Collectors for mainframes, UNIX®, Linux, Oracle, MS Windows®, and many other sources. Data collectors track usage data associated with:

- Operating systems
- Databases
- E-mail systems
- Storage systems
- Networks
- Any application, system, or monitor that creates usage metering data

Details can be found at:

`http://www.cimslab.com`

SAS - IT Charge Management

Working in concert with SAS IT Resource Management, SAS IT Charge Management provides an enterprise solution for tracking and managing IT costs. Utilizing the latest in software technology — an open architecture, client-server design, portability across multiple platforms, and a graphical user interface — IT Charge Management is flexible, powerful, easy to use, and easy to maintain.

Use SAS IT Charge Management to:

- Allocate IT resources to particular business cost centers
- Apply different charge rates
- Employ your own billing schemes
- Supply users with a quick and easy way to audit and manage IT resource use
- Provide IT with a quick and accurate response to billing and auditing inquiries
- Offer invoice and costs reporting in multiple currencies

For reporting, SAS IT Charge Management offers AccountView, which is a facility that takes the guesswork out of determining IT usage within your company. Your IT organization, executive management, and end users will have access to user-defined and ready-to-run reports. In addition, AccountView offers standard reports including charge item lists, charge by client (summary and detail), charge by item, charge by shift, and top 10 divisions by charge. Details can be found at:

`http://www.sas.com`

Monitoring z/VM Linux guests

The concept of shared resources is central to operating in a zSeries environment. However, when running Linux under z/VM, it can be difficult to determine how many resources are consumed by any single guest. There are many advanced z/VM and Linux monitoring tools; most focus on one part of the equation (either z/VM or Linux).

While a Linux guest may show high CPU utilization (for instance, using 100% of the CPU) inside Linux, this does not imply the guest is a high CPU consumer from a z/VM point of view. It may get only a few CPU-cycles from z/VM. To determine actual resource usage, one must consider the actual physical resources provided to the guest by z/VM.

We show how to correlate resource usage data and how to get an “absolute” view of resource allocation from inside a z/VM LPAR running Linux guests. The examples assume a setup with CPUs or IFLs dedicated to the LPAR.

Overview of z/VM monitor records

As an introduction to z/VM monitoring, we include a small extract from the z/VM performance publications. This is intended to give the reader a better understanding of the methodology and examples used here. For more detailed information (especially on programming interfaces), refer to *z/VM V4R3.0 Performance Reporting Facility Function Level 410*, SC24-6027.

The CP Monitor facility collects system performance data that can be made available to an external data reduction program for analysis. Statistics related to system operation or contention for major system resources can be generated. These resources include processors, storage, I/O devices, and the paging subsystem. You can control the amount and nature of the data collected. In general, monitoring is performed in this order:

1. The user employs the privileged CP **`MONITOR`** command to control monitoring. This includes the type, amount, and nature of data to be collected.
2. The monitor collects performance data and stores monitor records in a saved segment.
3. A CMS application program connects to the CP *MONITOR System Service to establish a data link with CP.
4. The application retrieves and processed monitor records from the saved segment.

Monitor System Service (*MONITOR)

The monitor system service (*MONITOR) notifies connected virtual machines when records are created by the z/VM monitor. The user-selected sets of statistics are collected and stored as monitor records in a user-defined saved segment. Statistics are grouped into sets called *domains*. Domains correspond to specific areas of system operation and are summarized in Table 3 on page 13.

Table 3 Monitor domains

Number	Name	Description
0	System	Contains information about system-wide resource usage
1	Monitor	Contains information about installation configuration (processors, paging, storage, I/O, and so on) and on the type of monitoring enabled
2	Scheduler	Contains information about scheduler queues, the flow of work through the scheduler, and the resource allocation strategies used by the scheduler and the dispatcher
3	Storage	Contains information about use of real, virtual, expanded, and auxiliary storage
4	User	Contains information about virtual machines (such as scheduling status, virtual I/O and vector use, and logon and logoff events)
5	Processor	Contains data on work dispatched to a given processor as well as other data related to processor usage
6	I/O	Contains information about I/O requests, error recovery, interrupts, and other information for real devices
7	Seek	Contains information about seek operations for DASD devices
10	Application Data	Contains application data copied from a virtual machine's storage when this storage has been declared to CP for collecting the data generated by the application program in that virtual machine

Monitor data

The CP `MONITOR SAMPLE` and `MONITOR EVENT` commands control collection of system statistics. *MONITOR provides the location of monitor records to a virtual machine. CP Monitor collects data during CP operation and stores ("reports") it as monitor records in the saved segment. Two types of data are collected:

- **Event data**
  Event data is collected and reported each time a designated system event occurs. The data reported represents the status of the system at the time the event occurred.
- **Sample data**
  Sample data is collected and reported at the end of each designated time interval. The types of sample data are:
  - **Single-sample data**
    This data is collected once during the time interval (at the end of that time interval). Some of this data represents the system status at the time the data was collected. Other data consists of the accumulated counters, states, or elapsed times values collected at the end of each time interval.
  - **High-frequency sample data**
    This data is collected at a higher rate than it is reported on. The data is reported along with single-sample data. At each high-frequency sampling time, the data collected is added to its corresponding counters. The data reported is the accumulated counts or state values collected since the start of high-frequency sampling.

Use the CP `MONITOR` command to select the type of data to collect. You also can control the time interval for single-sampling and the rate for high-frequency sampling.

As soon as at least one virtual machine connects to *MONITOR and the **MONITOR START** command has been issued, a set of data (referred to as *configuration data*) is collected. This data describes the configuration of the system and the monitor profile at the time monitoring started. There are sample configuration as well as event configuration records.

While monitoring is active, new configuration data is generated each time a virtual machine connects to *MONITOR.

Linux sysstat tools and data collection

We use the sysstat package included in the SLES 8 distribution to collect data from inside Linux guests. The sysstat package consists of tools to collect system resource usage data such as:

- I/O transfer rates
- Paging activity
- Process-related activities
- Interrupts and network activity
- Memory and swap space utilization
- CPU utilization
- Kernel activities
- TTY statistics

The two utilities we use are the **sa1** and the **sar** commands.

The sa1 command

We use the cron scheduler to regularly collect data gathered by the **sa1** command. No installation is required to use the package because it is included in the default SuSE installation. We need only adjust the cron interval used to schedule the **sa1** command.

In our example, we changed the interval from the default value of ten minutes to one minute. This introduces little or no noticeable overhead. Our cron files (/etc/cron.d/sysstat) are shown in Example 7.

Example 7 Contents of /etc/cron.d/sysstat file

```
#crontab for sysstat

#activity reports every 1 minutes everyday
*/1 *   * * *     root  /usr/lib/sa/sa1
```

Note: The **sa1** command must be scheduled to run on each Linux guest to be monitored.

Daily activity files in binary format are stored in the /var/log/sa directory:

```
-rw-r--r--    1 root     root      1520869 Sep 12 23:59 sa.2003_09_12
-rw-r--r--    1 root     root      1520869 Sep 13 23:59 sa.2003_09_13
-rw-r--r--    1 root     root      1520869 Sep 14 23:59 sa.2003_09_14
-rw-r--r--    1 root     root       622213 Sep 15 09:48 sa.2003_09_15
```

The sar command

The **sar** command processes files produced by **sa1** to generate readable reports. The **sar** command has various switches to select reported data; these switches and their usage are documented in the **sar** man page.

For our example, we use the **–u** switch to report CPU utilization, and the **–r** switch to report memory utilization. Output is shown in Example 8.

Example 8   Output of the sar command

```
# sar -ur
Linux 2.4.19-4suse-SMP (lin161)        11/13/03

00:00:00          CPU     %user     %nice   %system     %idle
00:01:00          all      0.05      0.00      0.18     99.77
00:02:01          all      0.03      0.00      0.03     99.93
00:03:00          all      0.03      0.00      0.03     99.93
00:04:00          all      0.02      0.00      0.05     99.93
...
00:00:00 kbmemfree kbmemused  %memused kbmemshrd kbbuffers  kbcached kbswpfree kbswpused %swpused
00:01:00      5256    250016     97.94         0     48096    163880         0         0     0.00
00:02:01      5256    250016     97.94         0     48096    163880         0         0     0.00
00:03:00      5256    250016     97.94         0     48096    163880         0         0     0.00
00:04:00      5252    250020     97.94         0     48096    163884         0         0     0.00
```

Refer to the **sar** man page for details on field meanings in the report.

Configure RSA authentication

Data is collected and reports generated at the central Linux server discussed in “Report front end” on page 9. We use the remote command execution facility provided by Secure Shell (SSH) to run the **sar** command on each monitored host from the LIN161 server. To automate the process, we configure SSH to use RSA authentication so as to bypass normal user password prompting. With RSA authentication, remote commands are authenticated using public key exchange.

Perform the following steps to configure RSA authentication between the central reporting system LIN161 and monitored Linux guests:

1. **Enable RSA authentication for SSH on each monitored guest.**
   RSA authentication is enabled by default in SLES 8.
2. **Generate a public/private key pair on the central reporting system.**
   Remote commands are run by CGI scripts executed from the Apache Web server. We create keys for the wwwrun user by using the following command sequence on the LIN161 server:

   ```
   # su - wwwrun
   $ pwd
   $ /var/lib/wwwrun
   $ ssh-keygen -t rsa
   Generating public/private rsa key pair.
   Enter file in which to save the key (/var/lib/wwwrun/.ssh/id_rsa):
   Enter passphrase (empty for no passphrase):
   Enter same passphrase again:
   Your identification has been saved in /var/lib/wwwrun/.ssh/id_rsa.
   Your public key has been saved in /var/lib/wwwrun/.ssh/id_rsa.pub.
   The key fingerprint is:
   17:bb:76:6c:f2:0b:2e:0b:b7:b0:f8:72:25:11:d6:82 wwwrun@lin161
   $ cat .ssh/id_rsa.pub
   ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAq...
   ```

   Keys are generated using the **ssh-keygen** command. A null passphrase is specified. The public key is stored in the /var/lib/wwwrun/.ssh/id_rsa.pub file.

Note: The wwwrun user is the default Apache user for SuSE installations created when Apache is installed. Its home directory is /var/lib/wwwrun.

3. **Create a Linux user on each monitored host.**
   We name the user saruser.
4. **Copy the LIN161 public key to each monitored host.**
   Append the /var/lib/wwwrun/.ssh/id_rsa.pub file from LIN161 to the /home/saruser/.ssh/authorized_keys file on each monitored host.
5. **Log in to each monitored host from the central reporting system.**
   Add the LIN161 system to the list of known hosts for user wwwrun:

```
S ssh -f saruser@lin162 ls
The authenticity of host 'lin162 (9.152.123.162)' can't be established.
RSA key fingerprint is d3:82:77:c0:dc:c5:8f:06:0f:1d:68:42:fc:b8:1c:b6.
Are you sure you want to continue connecting (yes/no)? yes
```

The **sar** command can now be remotely executed from the central reporting server using:

```
$ ssh -f saruser\@lin162 sar -ur -f /var/log/sa/sa.2003_09_23
```

Within the CGI script, file names are constructed from Perl variables.

Configure service machine to collect monitor records

To continuously collect z/VM resource usage data, we create a service guest machine named MONSRV. MONSRV is started automatically from AUTOLOG and operates in disconnected mode. A REXX script loads the standard monitor saved segment MONDCSS and connects MONSRV to the CP *MONITOR service. The script calculates CPU usage values and logs to a daily file. Log files are accessible to the central Linux reporting system using NFS.

The MONSRV user

The user MONSRV must have privilege class A or E to process monitor data. Because MONSRV runs disconnected, we set the secondary console to the operator. We define the IUCV *MONITOR and the saved segment MONDCSS to MONSRV. Its A-disk is defined large enough to hold the data (50 cylinders in our example). Example 9 shows the directory entry for the MONSRV user.

Example 9 The MONSRV user directory entry

```
USER MONSRV AUTOONLY 32M   32M EG
 INCLUDE IBMDFLT
 AUTOLOG AUTOLOG1 OP1 MAINT
 MACHINE XA
 OPTION QUICKDSP
 SHARE ABSOLUTE 3%
 IUCV *MONITOR MSGLIMIT 255
 NAMESAVE MONDCSS
 MDISK 191 3390 0750 0050 430W01  MRV READ     WRITE    MULTIPLE
```

Note: For security reasons, we prevent manual logon to this virtual machine by specifying AUTOONLY as the password. This service machine is started only via AUTOLOG.

The monitor list

CP monitor records data for all z/VM users. We select monitor records for specific guests using the MON LIST file. The format for each entry is:

```
<USERID>_<number of virtual CPUs>_<virtual memory in MB>_<IP/hostname>
```

Fields are space-delimited and have corresponding sizes of:

```
< 8 > 1 < 2 > 1 < 6 > 1 < unlimited >
```

The USERID and the number of virtual CPUs fields are used to calculate the CPU usage for the guest. Virtual memory size field is used to calculate the graphics scale. The IP address (or hostname) is used to connect to the guest over SSH.

Note: For z/VM users, the virtual memory and the hostname fields are optional. For Linux guests, these fields are mandatory.

The first record in the MON LIST file must be the SYSTEM pseudo-user description (specifying the number of physical CPUs defined to the LPAR). SYSTEM acts as a container for all z/VM users not listed in the file. For example, SYSTEM contains activity by users such as OPERATOR, TCPIP, or VMNFS as well as CP overhead reported as z/VM system services. Every VM user may be specified. A sample MON LIST file is shown in Example 10.

Example 10 Contents of the MON LIST file

```
SYSTEM   2
TCPIP    1
MONSRV   1
ACCSRV   1
LIN161   1  256    lin161
LIN162   2  512    lin162
LIN163   1  256    lin163
LIN164   1  256    lin164
LIN165   1  256    lin165
```

Monitor records and formulas

A complete listing and description for z/VM monitor records can be found on:

http://www.vm.ibm.com/pubs/ctlblk.html

For our calculations, we use sample data from the monitor record fields depicted in Table 4 on page 18.

Table 4 Monitor record fields used

Domain	Record	Field	Description
System 0	System data 2	MRHDRTOD	Time at which this record was built, in time-of-day (TOD) clock format: See *IBM System 370 XA Principle of Operation* for explanation of format.
		SYTPRP_PFXPRBTM	Total CPU time spent in emulation mode on this processor for all users
		SYSTPRP_PFXUTIME	Total CPU time spent on this processor that was charged to users: It does not include any time that CP spends on behalf of a user doing such things as instruction simulation or page translation. This is in CPU time, not elapsed (or wall clock) time.
		SYSTPRP_PFXTMSYS	Total CPU time spent on this processor that has been charged to the system instead of the user: It includes time spent by the system on behalf of the user doing such things as instruction simulation. It also includes time doing other types of system work, such as making scheduling decisions.
	User data 8	SYSTUSR_SYSUSRS	Cardinal count of current logged-on users
User 4	User activity data	USEACT_VMDUSER	User ID
		USEACT_VMDTTIME	Total time that has been expended on this VMDBK: This includes time that the VMDBK spent doing its own work and also time that the CP spent doing work on behalf of the VMDBK. It is in CPU timer format. The complement of this field is the elapsed time in TOD clock units.
		USEACT_VMDVTIME	Total time that this user was running and doing productive work: It does *not* include any time that CP spends on behalf of that user doing such things as instruction simulation or page translation. This is in CPU timer format. The complement of this field is the elapsed time in TOD clock units.
		USEACT_VMDWSSPR	This user's projected working set size: This is the scheduler's estimate of the amount of storage that this user will require to be resident as it runs during its next stay in the dispatch list. It is based upon the user's past behavior and, in some cases, on the behavior of similar types of users.

Note: In the formulas that follow:

- **last** refers to values from the preceding record
- **first** refers to values of the current monitor record

As CPU time values are cumulative counts, we calculate differences in successive monitor record values.

Every record is time stamped with a field (MRHDRTOD) indicating when it was created. We use only sample data records created at the same time (or at least nearly the same time). We use the time stamp of Domain 0 Record 2 to calculate interval length for all records.

z/VM system-wide CPU utilization

The physical processor percent busy equals the total user time (all users) plus system overhead divided by the sum of the online time for all processors as expressed by the formula in Figure 5.

$$\text{system percent CPU busy} = \frac{\sum_{CPU=0}^{n} (\text{total user time} + \text{system overhead})}{\sum_{CPU=0}^{n} \text{online time}} \bullet 100$$

Figure 5 Calculating system CPU busy percentage

In Figure 5:

- $\text{total user time} \equiv SYTPRP\_PXFUTIME_{last} - SYTPRP\_PXFUTIME_{first}$
- $\text{system overhead} \equiv SYTPRP\_PXFTMSYS_{last} - SYTPRP\_PXFTMSYS_{first}$
- $\text{online time} \equiv MRHDRTOD_{first} - MRHDRTOD_{last}$

Note: Processor time counters are based on the CPU timer (which counts backwards). To obtain positive deltas, we subtract successive CPU time values in reverse order (the current value is subtracted from the previous value).

Per user CPU usage

To calculate the CPU usage for each guest, we use two similar formulas (extracted monitor data from the user domain record). Figure 6 depicts the calculation used to compute total CPU usage for a guest.

$$\text{total user percent CPU busy} = \frac{\sum_{CPU=0}^{n} \text{total user time}}{\sum_{CPU=0}^{n} \text{online time}} \bullet 100$$

Figure 6 Calculating total user CPU busy percentage

Figure 7 on page 20 depicts the calculation used to compute emulated user CPU usage for a guest.

Note: Emulated user percent CPU busy is a measure of real CPU percentage given to Linux in the interval. CP overhead is the difference between total and emulated CPU percent values. All values are expressed as a percentage of one physical or logical CPU. Values up to $n \bullet 100\%$ on an n-way system are possible.

$$\text{emulated user percent CPU busy} = \frac{\sum_{\text{CPU}=0}^{n} \text{emulated user time}}{\sum_{\text{CPU}=0}^{n} \text{online time}} \bullet 100$$

Figure 7 Calculating emulated user CPU busy percentage

In Figure 6 on page 19 and Figure 7:

- $\text{total user time} \equiv \text{USEACT\_VMDTTIME}_{\text{last}} - \text{USEACT\_VMDTTIME}_{\text{first}}$
- $\text{emulated user time} \equiv \text{USEACT\_VMDVTIME}_{\text{last}} - \text{USEACT\_VMDVTIME}_{\text{first}}$
- $\text{online time} \equiv \text{MRHDRTOD}_{\text{first}} - \text{MRHDRTOD}_{\text{last}}$

Again, we subtract CPU time values in reverse order to obtain positive differences.

Note: Since we only use sample data, we can not calculate usage values for sub-interval periods. Sub-interval periods occur when a user logs on or logs off. We disregard these intervals by eliminating negative or incorrect values. To include these intervals, you must enable and collect monitor event data and modify the logging script accordingly. For the scope of this Redpaper, this was deemed unnecessary.

The logging procedure

The complete logging facility is implemented in three scripts:

- **RUN EXEC**
  Executed at startup from the PROFILE EXEC, this script loads the monitor saved segment, starts the monitor, and opens a STARMON pipe to connect to the saved segment. It then calls the COLLECT REXX script.
- **COLLECT REXX**
  This script runs an endless loop to extract output from the pipe. Usage data is calculated and logged in to a daily file.
- **CLEAN EXEC**
  This script runs daily to purge old log files.

RUN EXEC

This script is shown in Example 11.

Example 11 RUN EXEC script

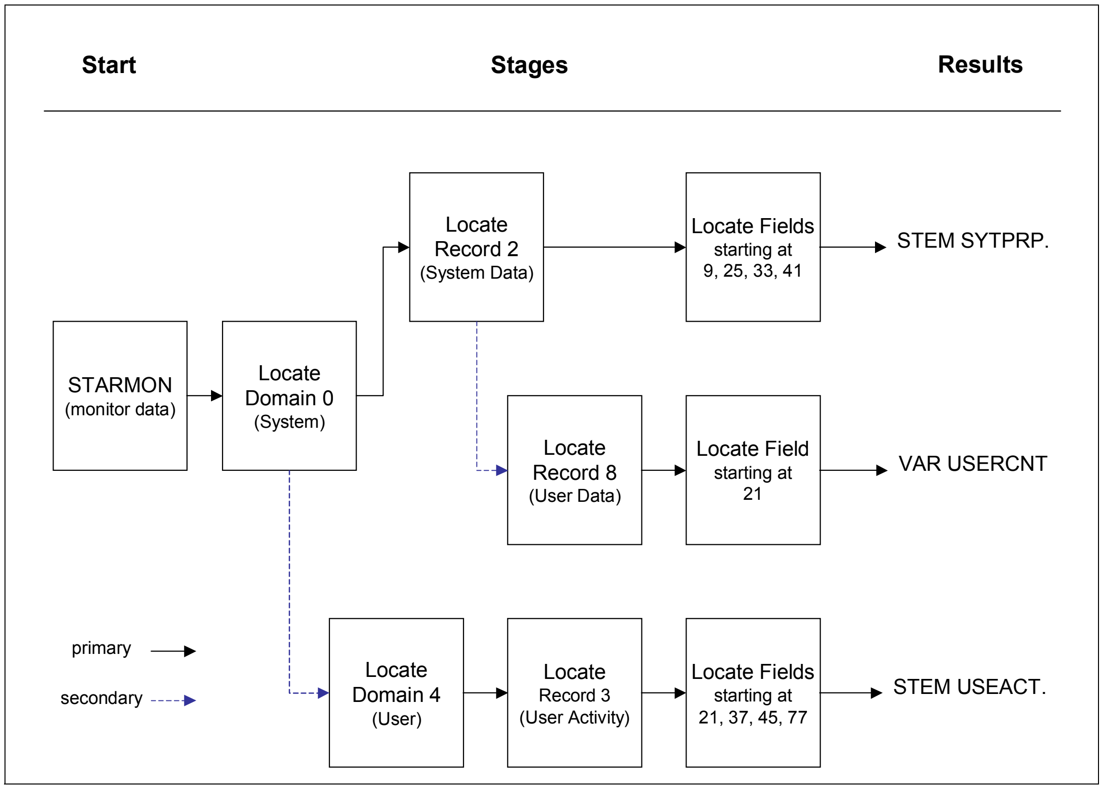
```
/* EXEC to start the collection of monitor data */

n = 60 - substr(time(),7,2)   /* start at a 00 sec */
'CP SLEEP 'n' SEC'            /* interval         */

'SEGMENT LOAD MONDCSS'
'CP MONITOR SAMPLE ENABLE SYSTEM ALL'
'CP MONITOR SAMPLE ENABLE USER ALL'
'CP MONITOR SAMPLE START'
'PIPE STARMON MONDCSS SAMPLE | COLLECT'
```

The sleep statement at the beginning ensures our that records have a full minute timestamp and therefore match the timestamps of the Linux **sar** data.

COLLECT REXX

The COLLECT REXX script executes an endless loop to read data written by the RUN EXEC. Variables used in the script hold values extracted from the pipe. Helper variables calculate the differences between the last and the current monitor value and are organized in stems (arrays) according to the number of processors and users. The formulas used are those explained in “Monitor records and formulas” on page 17. Details can be found in the comments provided with “MONSRV” on page 34.

The major stages and flow of the pipe we use to extract the monitor data is shown in Figure 8.



Start
Stages
Results
STARMON (monitor data)
Locate Domain 0 (System)
Locate Record 2 (System Data)
Locate Fields starting at 9, 25, 33, 41
STEM SYTPRP.
Locate Record 8 (User Data)
Locate Field starting at 21
VAR USERCNT
Locate Domain 4 (User)
Locate Record 3 (User Activity)
Locate Fields starting at 21, 37, 45, 77
STEM USEACT.
primary
secondary

Figure 8 Extraction pipe flow

Stages select desired records and fields. The secondary output of the stages contains unmatched data of the stage. It is either used as the input of a next stage or discarded.

At the end of each stage, data is stored in the SYTPRP and USEACT stems and the USERCNT variable. The complete pipe command is shown in Example 12.

Example 12 The monitor pipe command

```
/*-- read system and user records into variables --------------------*/
  /*-- (monitor records from domain 0 rec 2 and domain 4 rec 3) -------*/
  'callpipe (end ?) *:',
  '| a: locate 5 X00',
  '| b: locate 8 X02',
  '| take first ' cpucnt,
  '| SPECS 9-16 C2X 1 25-32 C2X 21 33-40 C2X 41 41-48 C2X 61',
  '| stem sytprp.',
  '? b:',
  '| locate 8 X08',
  '| SPECS 21-24 C2D 1',
  '| var usrcnt',
  '? a:',
  '| locate 5 X04',
  '| locate 8 X03',
  '| take first ' usrcnt,
  '| SPECS 21-28 1 37-44 C2X 11 45-52 C2X 31 77-80 C2D 51',
  '| stem useact.'
```

Extracted data is used to calculate the CPU usage for the whole system as well as for each individual user. Calculated values are written to a log file with the current date as file type (extension) such as:

```
MON $M092503 A
```

Each log file line has the format:

```
<Time Stamp>_<System Data>_<User1 Data>_<User2 Data>…_<UserN Data>
```

Where:

Time Stamp	Timestamp (in hh:mm:ss format)
System Data	Block of space-delimited fields containing total CPU, user CPU, system CPU, and emulated CPU times. The format of the block is: `<total>_<user>_<system>_<emulated>` This data is written only once per log line.
User Data	A block of space-delimited fields containing user total CPU time, user virtual CPU time, and projected working set size for each user listed in the MON LIST file. The format of the block is: `<UserID>_<total>_<virtual>_<working set size>` A sample extract is shown in Figure 13.

Example 13 Sample user data log entry

```
16:17:00   2.50   1.88    0.61   1.29 SYSTEM   0.50 0 0 TCPIP    0.00 0.00 2478 MONSRV
0.02 0.02 233 ACCSRV   0.00 0.00 216 LIN161   0.35 0.25 36994 LIN162   0.80 0.58 16418
LIN163   0.34 0.25 36601 LIN164   0.21 0.12 36568 LIN165   0.28 0.18 36615
```

As stated before, the entry total CPU for the SYSTEM pseudo-user shows the difference between the system wide total CPU usage and the sum of the total CPU usage of all listed users. We interpret this as the combined overhead for z/VM system services such as TCPIP, VMNFS, OPERATOR activity, etc.

CLEAN EXEC

The clean script runs daily. It utilizes a pipe to list all monitor files and sort them by oldest date. The variable *keep* controls the number of files to retain, and the remainder are deleted. The complete script is shown in Example 14.

Example 14   The CLEAN EXEC script

```
/* CLEAN EXEC - housekeeping routine to purge old monitor records */
keep=7

'PIPE (endchar ?)',
'cms listfile mon $m*',
'| sort 10.8',
'| a: take last 'keep,
'| stem dummy.',
'?',
'a: | stem old.'

do i=1 to old.0
  call lineout , 'cleaning old files 'old.i
  'erase 'old.i
end
```

Correlation of the Linux and z/VM data

With logging scripts in z/VM and Linux (the **sar** reports), CPU and memory usage values from both sides are available. To determine the real application performance, one must know the amount of resources (CPU cycles) available to the application. Because resource sharing is central to z/VM, the CPU cycles available to a Linux guest vary according to overall system load. Therefore, we correlate usage data reported by Linux (virtual or “from the inside view”) to the data derived from z/VM (real or “from the outside view”).

CPU usage

For each Linux guest, z/VM must consume some CPU cycles for hypervisor work (CP overhead). Other cycles are consumed by Linux itself. Typically, more virtual CPUs (Linux systems) are defined than the number of installed physical processors. A Linux guest controls only a portion of the available CPU cycles during periods of high activity. The actual CPU percentage allocated to a Linux guest during an interval is the user virtual CPU usage from z/VM perspective (not the percentage that Linux reports).

For example, consider a Linux guest that reports 100% CPU busy in an interval. In the same interval, z/VM may report only 80% virtual user CPU percentage (and possibly 5% for CP user overhead) for the guest. With dedicated IFL processors, z/VM reported values are actual physical usage values. (Shared processors are beyond the scope of this Redpaper.)

Even though Linux is running at 100% during the interval, its virtual CPU is only running at 80% of the normal (physical) speed. Thus, jobs run longer or application response time is accordingly longer.

Memory usage

Linux is designed to efficiently use random access memory (RAM), allocating portions for the kernel, applications, buffers, and caches. After allocating kernel and application memory, Linux allocates unused memory to buffers and cache. Eventually, most available memory is used. While this policy (buffers and cache preferred over unused memory) may be correct for servers with dedicated memory, this has consequences when running Linux under z/VM.

In z/VM systems, the total sum of virtual memory defined for all guests typically exceeds the available physical memory. This works because z/VM manages virtual memory, allocating real memory pages only to active guest virtual memory pages. Inactive pages are moved to expanded storage or paging space. With Linux guests, most or all virtual memory pages are active because Linux allocates unused memory for buffers and cache.

This can lead to situations where memory over-commitment starts degrading overall system performance. z/VM cannot page out memory pages for long periods (as most Linux guest virtual memory pages are active). Instead, z/VM must steal pages from active users. Therefore, it is best to limit the virtual memory size of a Linux guest to that required by the kernel itself and the running applications, with a minimum size required for buffers and caches.

To determine the optimal virtual memory size for a Linux guest, monitor memory usage by the guest (especially the buffer and cache size). In our example, we show the projected working set size reported by z/VM. This is the amount of memory that z/VM considers active for that guest.

Note: If the sum of the working sets of all guests exceeds available physical memory, paging and/or page stealing will occur.

Web page to generate reports online

In this section, we explain configuration of a simple Web page to generate graphical reports from the collected monitor data. We run an Apache HTTP server on the LIN161 Linux report server guest introduced in “Report front end” on page 9. This Linux guest is dedicated for the task, and user access to the server should be restricted to the system administrator. Reports are generated on-demand (instead of periodically) for minimal monitoring overhead. We use the gnuplot package to create the graphs.

Note: The gnuplot is included in the SLES 8 distribution.

The gnuplot package

Gnuplot is a command-driven function plotting program. It can be driven either interactively or by a configuration file. Output can be generated either on an X-Window display or as an image. Input can read from data files. Details on gnuplot can be found at:

http://www.gnuplot.info

We use Perl CGI scripts to generate temporary configuration files for gnuplot. Then we use the **`gnuplot`** command to produce a graph image based on the chosen settings and data files. The resulting image is displayed in a dynamically created HTML document.

Example 15 shows one configuration file generated for the z/VM system total report.

Example 15 Example gnuplot configuration file

```
set title "VM total CPU usage (09/25/03)"
set terminal png color
set data style lines
set size 1,0.5
set xdata time
set ylabel "CPU %"
set timefmt "%H:%M:%S"
set yrange [0:300]
set xrange ["00:00":"23:59"]
set format x "%H:%M"
plot '/mnt/monsrv/mon.$m092503' using 1:2 t "%CPU total" , '/mnt/monsrv/mon.$m092503'
using 1:4 t "%CP overhead"
```

Access to the data

“Configure RSA authentication” on page 15 explains RSA authentication configuration between the reporting system and monitored Linux guests. We use SSH remote execution facility to generate **sar** reports on monitored Linux guests:

```
ssh -f saruser\@$hostname sar -ur -f /var/log/sa/sa.20$year_$month_$day
```

Where:

$hostname	Specifies the IP address or DNS hostname of the guest specified in the MON LIST file.
$day, $month, $year	Specify values from the report front end HTML form.

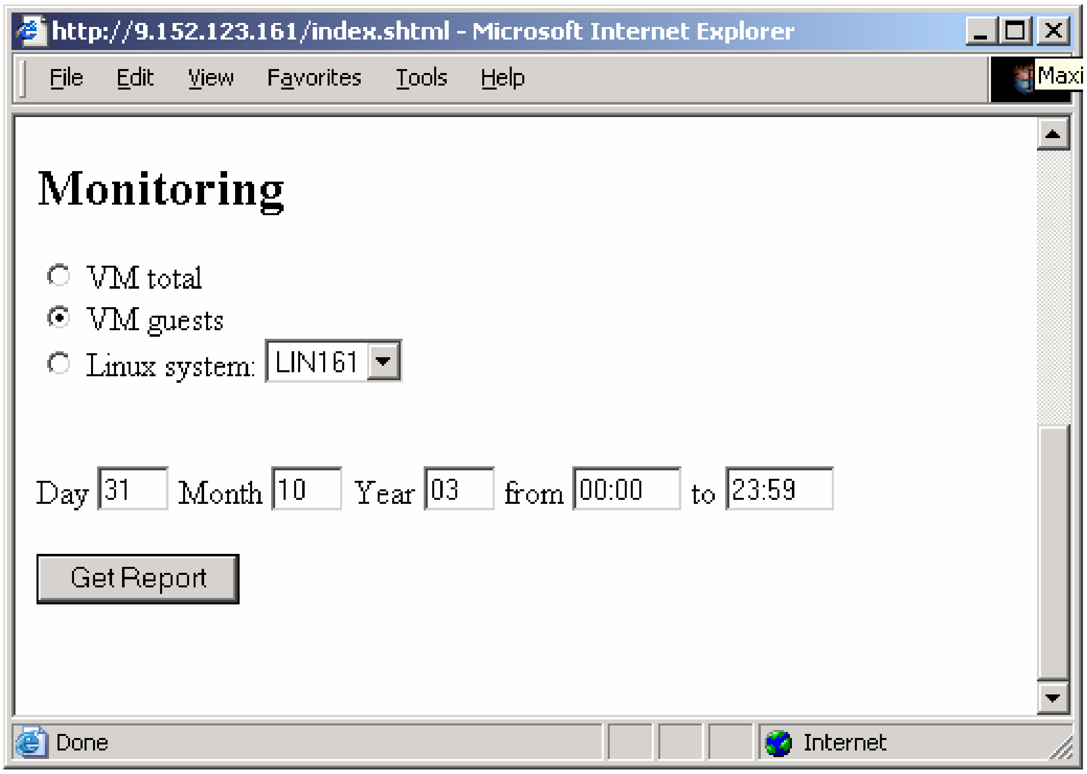
To access the z/VM data, we mount the home disk of user MONSRV via NFS. See “VMNFS” on page 32 for the VMNFS setup. The mount point and options are specified in /etc/fstab are:

```
10.1.1.1:/MONSRV,userid=monsrv,password=XXXX /mnt/monsrv nfs ro,auto  0 0
```

Important: Because the z/VM user ID and password are included in the /etc/fstab file, it is important to secure the guest. User access to this server should be restricted to the system administrator.

Report front end

The report generated from the HTML form is shown in Figure 9 on page 26. Various report types and date/time combinations may be selected.


http://9.152.123.161/index.shtml - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Monitoring

( ) VM total
(●) VM guests
( ) Linux system: LIN161

Day 31 Month 10 Year 03 from 00:00 to 23:59

Get Report

Done | Internet

Figure 9 Front end monitoring report

After pressing the ‘Get Report’ button, the **monrep** CGI script is called with the selected values. The HTML source and **monrep** Perl script are listed in “MONSRV” on page 34.

z/VM system total report

The z/VM system total report in Figure 10 shows overall CPU usage for the LPAR and CP system overhead.

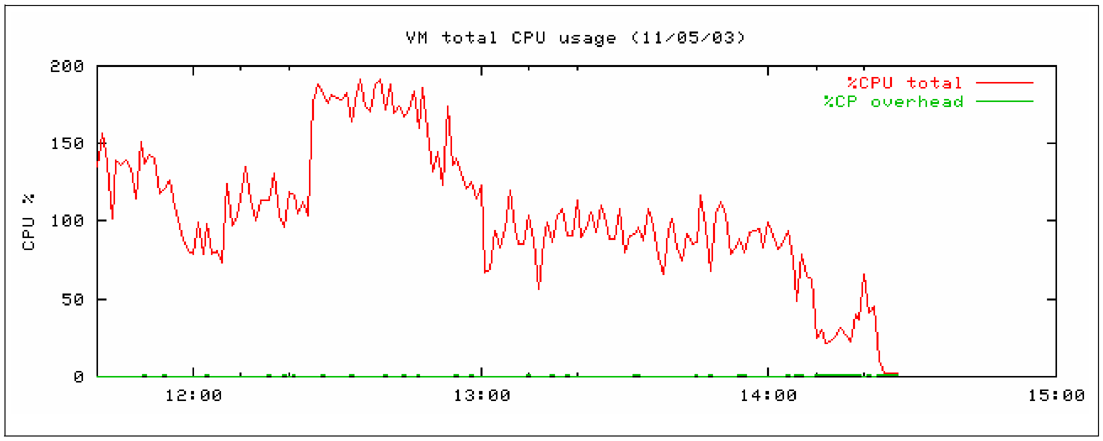


Figure 10 z/VM system total report

z/VM user report

The stacked type diagram z/VM user report in Figure 11 on page 27 shows accumulated CPU usage for all guests listed in the MON LIST file.

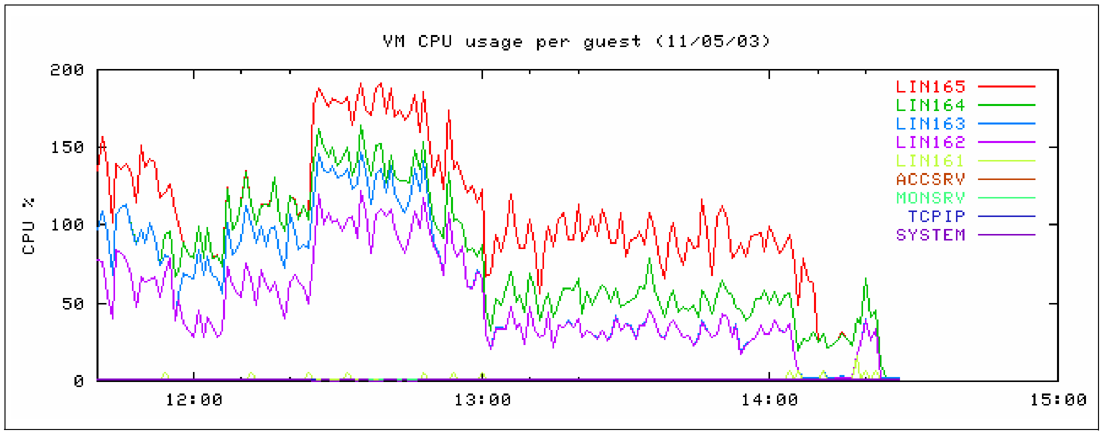


Figure 11 z/VM user report

CPU usage is the total user CPU usage (virtual and CP overhead) for each user. The pseudo-user SYSTEM records all activity caused by the system or monitored users. In this context, SYSTEM is interpreted as combined overhead for z/VM system services.

Single Linux system report

The report for a single Linux system in Figure 12 shows the correlated CPU and memory usage reported from Linux and z/VM.

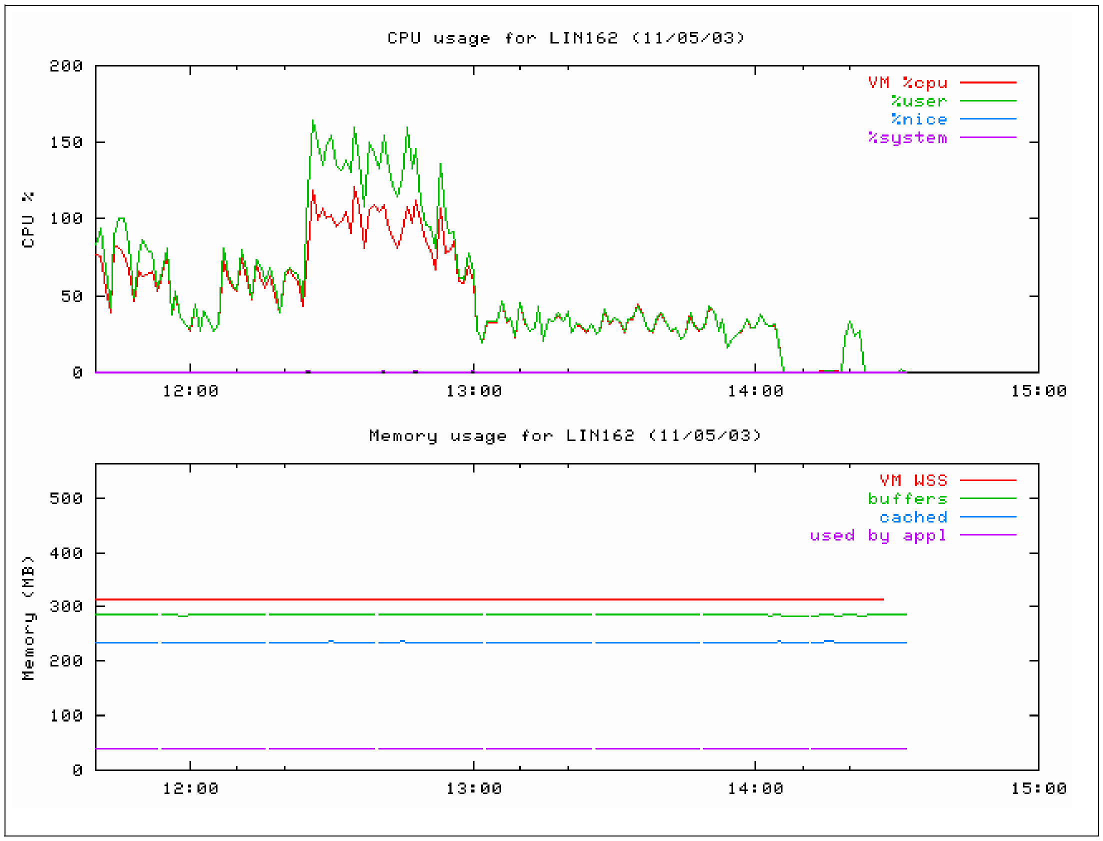


Figure 12 Single Linux system report

z/VM values represent user virtual CPU usage and projected working set size.

System wide heavy CPU usage

To demonstrate a situation with heavy CPU usage, we ran a benchmark (PI calculation) on some Linux guests to show virtual CPU speed declining during periods of CPU contention. This example runs on a z/VM LPAR with two physical processors (IFLs). Five Linux guests are defined; three run the benchmark. The three benchmark guests are defined as:

- **LIN162**
  Defined with two logical processors, assumed to be production system 1
- **LIN163**
  Defined with one logical processor, assumed to be production system 2
- **LIN164**
  Defined with one logical processor, assumed to be a test system

Two benchmark runs are made on the LIN162, 163, and 164 guests; a delay between each guest is introduced when starting the benchmark. For the first run, each guest is assigned a relative share (weight) of 100 (the default for z/VM guests). For the second run, the LIN162 guest is assigned a relative weight of 200, and the LIN164 guests are assigned a relative weight of 50. The z/VM user report in Figure 13 shows that the Linux guests fully utilize CPU resources.

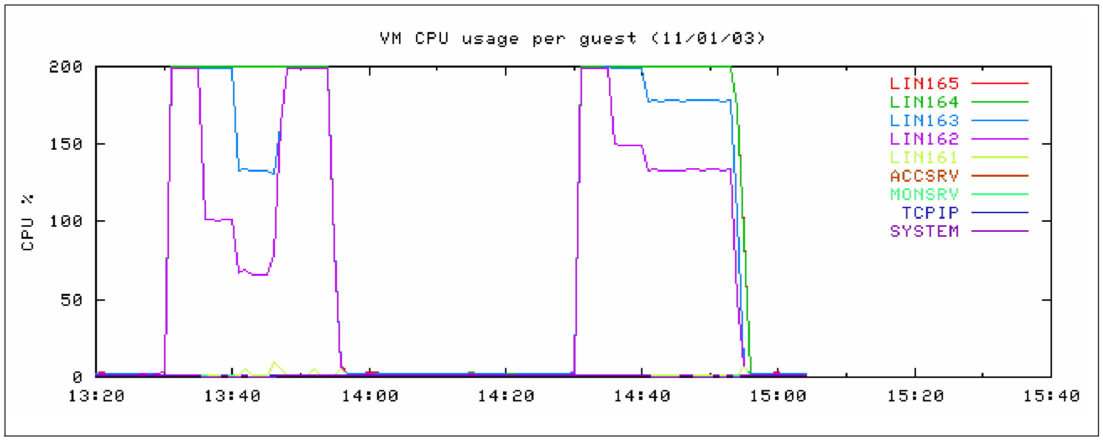


Figure 13 Benchmark z/VM user report

In the first benchmark run (in the interval 13:20-14:00), we note:

- During the first minute, the LIN162 guest consumes 100% of both physical processors.
- When the LIN163 guest starts, the LIN162 and LIN163 guests each consume the equivalent of 100% of a single processor.
- When the LIN164 guest starts, z/VM allocates the equivalent of 66% of a single processor to guest. This is expected because there are three guests with equal weight sharing two processors.

In Figure 14 on page 29, CPU utilization is depicted as reported from the perspective of LIN162 Linux guest. Similarly, Figure 15 on page 29 and Figure 16 on page 29 report CPU utilization from the perspective of the LIN163 and LIN164 Linux guests.

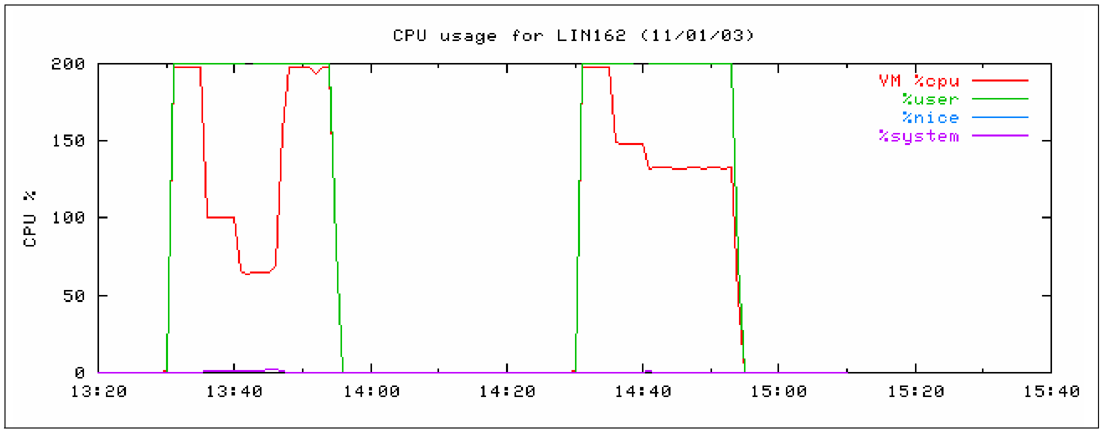


Figure 14 Benchmark LIN162 CPU report

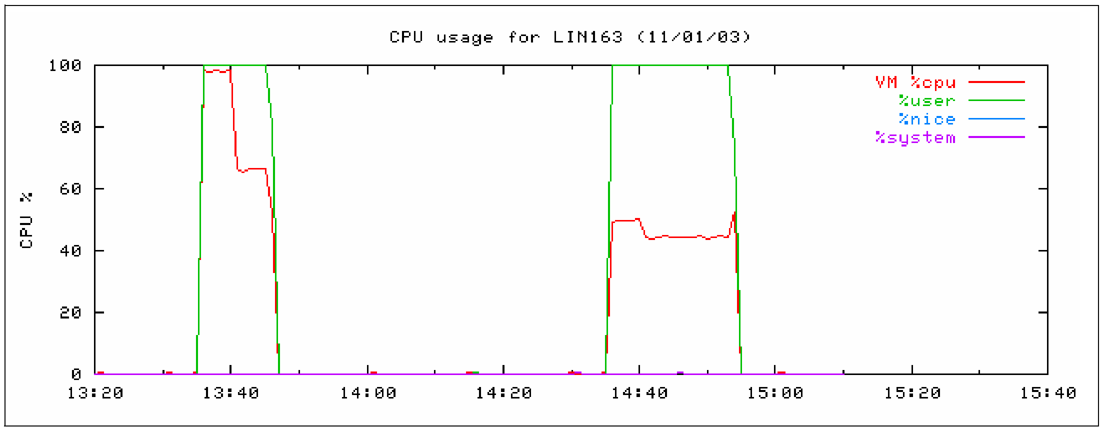


Figure 15 Benchmark LIN163 CPU report

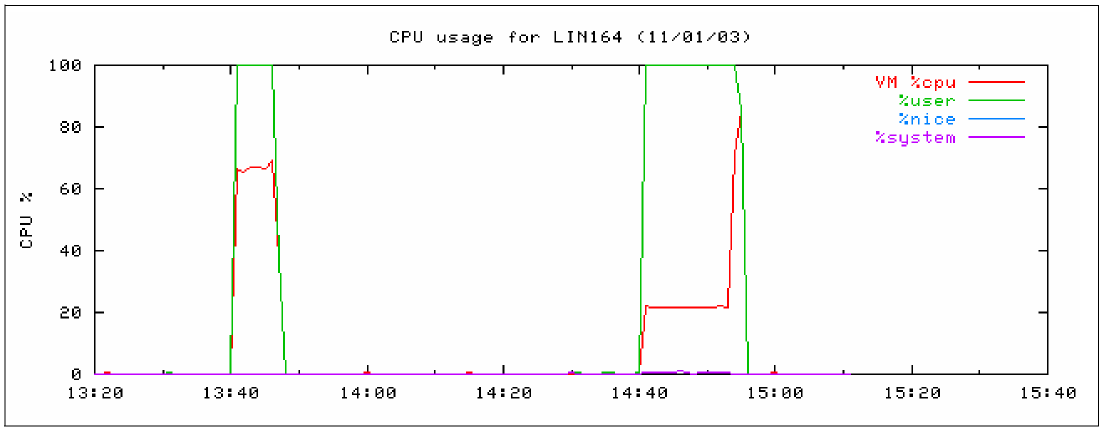


Figure 16 Benchmark LIN164 CPU report

These reports illustrate that Linux always “sees” $n \bullet 100\%$ CPU usage (where n is the amount of virtual processors defined to the system). Correlating the z/VM and Linux data, we see a decline in the speed of the virtual processors during a period of high activity as recorded in the %cpu (red) graph.

In Figure 14, we note the most dramatic decline in virtual CPU speed for the LIN162 (down to 66% from 200%) when CPU resources are constrained. Because LIN162 is a production system, we compensate for the loss of CPU speed by assigning a relative weight of 200 to LIN162 and a relative weight of 50 to LIN164 before the second run. The effect is noted in Figure 14 on page 29 in the second run. In this run, the utilization decline from 200% is reduced to 120%.

Using these reports, you can identify periods of resource constraints and take appropriate action (such as adjusting relative or absolute shares according to workload).

Linux memory usage

After Linux IPL, total memory usage reported by both Linux and the z/VM projected working set size is typically much smaller than the defined virtual memory size. Eventually, Linux fills unused free memory with cache and buffers. The z/VM working set size grows accordingly.

When all memory is used and an application requests more memory, Linux reclaims memory from the cache and buffers. The z/VM working set size remains the same (approximately equal to the guest virtual memory size). The best opportunity to reduce the z/VM working set size comes from reducing the size of buffer and cache memory. This can be accomplished by reducing the size of the guest virtual memory.

Memory usage monitor reports (such as depicted in Figure 17) can help determine how much virtual memory size may be reduced for Linux guests. Look at memory usage over long periods of time.

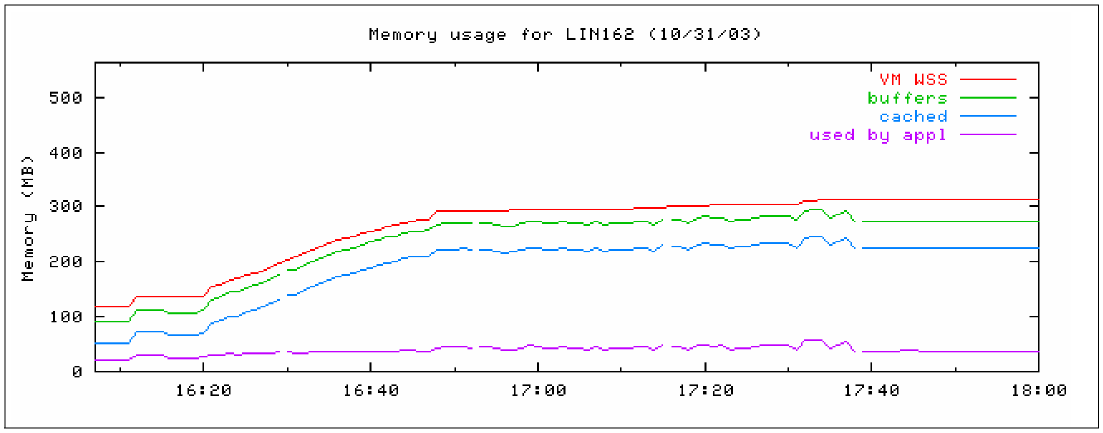


Figure 17 Single Linux system memory report

Commercial monitoring products

This section provides a short overview of some commercial monitoring products available for z/VM and Linux.

IBM Performance Toolkit for VM

The Performance Toolkit for VM provides enhanced monitor and performance report capabilities for z/VM systems programs, system operators, or performance analysts. Offered as a priced optional feature of z/VM, the Performance Toolkit for VM is derived from the FCON/ESA program (5788-LGA). Functions provided by Performance Toolkit for VM include:

- Operation of the system operator console in full-screen mode
- Management of multiple (local or remote) z/VM systems
- Processing of VM history files and VM monitor data captured by the MONWRITE utility
- Graphic performance analysis using a Web browser or PC 3270 emulator graphics
- TCP/IP performance reporting

In addition to analyzing VM performance data, the Performance Toolkit for VM processes Linux performance data obtained from the Resource Management Facility (RMFTM) Linux performance gatherer, rmfpms **`(using the rmfpms command)`**. The rmfpms application is available from the zSeries RMF™ Web site. The Linux performance data from RMF can be viewed and printed similarly to the way VM data is presented. Details can be found at:

`http://www.vm.ibm.com`

Velocity Software ESALPS

ESALPS collects and analyzes network data, Linux servers, and your z/VM systems in a consistent and integrated format. Data is continually collected, presented real-time, automatically analyzed (triggering exception reporting), and stored in a single performance database. Reports are then generated for long-term trend analysis and capacity planning purposes. This information is accessible via your CMS terminal or Web browser.

The ESALPS feature set highlights include:

- z/VM full subsystem measurement
- Standardized reporting
- Exception monitoring and reporting

The ESALPS suite includes ESATCP to acquire network and Linux performance data; ESAMON to display real-time network, Linux, and VM performance; ESAMAP to create reports and detailed analysis after the fact; and ESAWEB to provide an internet graphical interface to your users. Details are available at:

`http://www.velocitysoftware.com`

BMC MAINVIEW

With MAINVIEW for Linux - Servers, you can monitor specific Linux systems or you can monitor all your Linux systems at once. If you wish, you can see an aggregate view of how all of your Linux systems are running and then drilll down to a specific Linux system for detailed analysis. This approach allows the Linux technician to monitor hundreds or even thousands of Linux systems expected to run in a z/VM environment.

In addition, the MAINVIEW for Linux - Servers product offers:

- Real-time, interval, and historical displays of Linux performance and availability data
- Outside-in VM performance information
- CPU load+ MP user statistics (per processor or engine)
- VM Storage utilization statistics
- I/O device statistics (per real device)
- CP-owned device statistics (per real CP-owned)
- Detailed monitoring of system processes

The MAINVIEW for Linux - Servers interface is designed so that you can quickly identify the systems and applications that consume the most resources. At the same time, you also have the ability to quickly drill down to any monitored Linux system to look at specific detail. Details can be found at:

`http://www.bmc.com`

Code listings

This section contains the example scripts and files that are not listed completely in previous sections. Softcopy versions of theses files are available as described in “Additional material” on page 42.

VMNFS

Example 16 shows the VMNFS relevant entries in the different TCPIP configuration files.

Example 16 PROFILE TCPIP file

```
AUTOLOG
  PORTMAP   0           ; PORTMAP SERVER
  VMNFS     0           ; NFS SERVER
ENDAUTOLOG

PORT
  111  TCP PORTMAP             ; Portmap Server
  111  UDP PORTMAP             ; Portmap Server
  2049 UDP VMNFS               ; NFS Server
  2049 TCP VMNFS     NOAUTOLOG ; NFS Server
```

Example 17 shows the entries in the SYSTEM CTCPARMS file relevant to VMNFS.

Example 17 SYSTEM CTCPARMS file

```
:NICK.VMNFS    :ANONYMOUS.YES
               :PARMS.
:NICK.PORTMAP  :PARMS.
```

Example 18 shows the entries in the VMNFS CONFIG file to export the ACCSRV and MONSRV A-disks.

Example 18 VMNFS CONFIG file

```
EXPORT /ACCSRV               ACCSRV.191,RO,LINES=NL,TRANS=YES
EXPORT /MONSRV               MONSRV.191,RO,LINES=NL,TRANS=YES
```

ACCSRV

Example 19 contains the complete listing of the READACC EXEC script discussed in “READACC EXEC” on page 6.

Example 19 READACC EXEC script

```
/*---------------------------------------------------------------------*/
/* Name:     READACC EXEC                                              */
/* Funktion: script to produce human readable records out of VM       */
/*           account records                                           */
/*           collects CPU and NET usage values for users listed in     */
/*           file ACC LIST, writes log to daily file                   */
/*---------------------------------------------------------------------*/

numeric digits 100

/*-- read list of account guests --------------------------------------*/
lstcnt = lines('ACC LIST A')
do i = 1 to lstcnt
  list.i = substr(linein('ACC LIST A'),1,8)
end
call lineout 'ACC LIST A' /* close stream */

/*-- get the latest account file --------------------------------------*/
'pipe cms listfile account * d | sort 10.8 desc | var input'
call lineout , 'processing 'input
```

```
/*-- set the output file ----------------------------------------------*/
accdata = 'ACC 'substr(input,10,8)' A'

/*-- main routine to select and process -------------------------------*/
/*-- account records 01 and 0C ----------------------------------------*/
'pipe (end ?) <' input,
'| a: locate 80 /1/',
'| SPECS 1-8 1 17-22 10 29-32 C2X 20 33-36 C2X 30 37-40 C2X 40',
'| stem cpu.',
'? a:',
'| locate 80 /C/',
'| SPECS 1-8 1 51-58 C2X 10 59-66 C2X 30',
'| stem net.'

/*-- add up values for each guest -------------------------------------*/
do i = 1 to lstcnt
  conn = 0
  tcpu = 0
  vcpu = 0
  bsen = 0
  brec = 0

  'pipe stem cpu.',
  '| locate 1-8 /'list.i'/',
  '| stem tmpcpu.'
  'pipe stem net.',
  '| locate 1-8 /'list.i'/',
  '| stem tmpnet.'

  date = substr(tmpcpu.1,10,6)
  do j = 1 to tmpcpu.0
    conn = conn + x2d(substr(tmpcpu.j,20,8))
    tcpu = tcpu + x2d(substr(tmpcpu.j,30,8)) / 1000
    vcpu = vcpu + x2d(substr(tmpcpu.j,40,8)) / 1000
  end
  line = list.i' 'date' 'tconv(conn)' 'tconv(tcpu)' 'tconv(vcpu)
  do j = 1 to tmpnet.0
    bsen = bsen + x2d(substr(tmpnet.j,10,16))
    brec = brec + x2d(substr(tmpnet.j,30,16))
  end
  line =line' 'bsen' 'brec
  call lineout accdata, line
end
call lineout accdata/* close stream */
exit
/*-- end of main routine ----------------------------------------------*/

/*-- function to convert seconds to hhhh:mm:ss ------------------------*/
tconv: procedure
  arg s
  s = format(s,,0)
  h = s % 3600
  s = s - h * 3600
  m = s % 60
  s = s - m * 60
  hms = right(h,4,0)':'right(m,2,0)':'right(s,2,0)
return hms
```

MONSRV

Example 20 contains the complete listing of the READACC EXEC script discussed in "READACC EXEC" on page 6.

Example 20 COLLECT REXX script

```
/*---------------------------------------------------------------------*/
/* Name:     COLLECT REXX                                              */
/* Funktion: script to produce human readable records out of VM       */
/*           monitor records                                           */
/*           collects CPU usage values for system and users listed in  */
/*           file MON LIST, writes log to daily file                   */
/*---------------------------------------------------------------------*/

numeric digits 100

/*-- initialize some variables ----------------------------------------*/
usrcnt = 0
lasttime='00:00:00'
day = date('U',,,'')
init = 3                /*-- first 3 loops needed for initialisation --*/

/*-- read the list of guests to be monitored --------------------------*/
addvcpu = 0
lstcnt = lines('MON LIST A')
do i = 1 to lstcnt
  list.i = linein('MON LIST A')
  user.i.numcpu = substr(list.i,10,2)
  if i > 1 then do
    addvcpu = addvcpu + user.i.numcpu - 1
  end
  list.i = substr(list.i,1,8)
end
cpucnt = user.1.numcpu

/*-- initialize system fields -----------------------------------------*/
totcpu = 0
usercpu = 0
syscpu = 0
emucpu = 0
do i = 1 to cpucnt
  mrhdrtod_l.i = 0
  pfxprbtm_l.i = 0
  pfxutime_l.i = 0
  pfxtmsys_l.i = 0
end

/*-- initialize user fields -------------------------------------------*/
do i = 2 to lstcnt
  user.i.currcpu = 1
  do j = 1 to user.i.numcpu
    user.i.currcpu.j.vmdttime_l = 0
    user.i.currcpu.j.vmdvtime_l = 0
  end
end
user.1.vcpu = 0
user.1.wss = 0


/*-- the main loop to read and transform monitor records --------------*/
```

```
do forever
  /*-- read system and user records into variables sys and usr --------*/
  /*-- (monitor records from domain 0 rec 2 and domain 4 rec 3) -------*/
  'callpipe (end ?) *:',
  '! a: locate 5 X00',
  '! b: locate 8 X02',
  '! take first ' cpucnt + addvcpu,
  '! SPECS 9-16 C2X 1 25-32 C2X 21 33-40 C2X 41 41-48 C2X 61',
  '! stem sytprp.',
  '? b:',
  '! locate 8 X08',
  '! SPECS 21-24 C2D 1',
  '! var usrcnt',
  '? a:',
  '! locate 5 X04',
  '! locate 8 X03',
  '! take first ' usrcnt,
  '! SPECS 21-28 1 37-44 C2X 11 45-52 C2X 31 77-80 C2D 51',
  '! stem useact.'

  /*-- calculate the system totals and store in variable line ---------*/
  /*-- we store the last values before the new calculation to ---------*/
  /*-- get in sync with user values -----------------------------------*/
  line = lasttime' 'format(totcpu,5)' 'format(usercpu,5),
         ' 'format(syscpu,5)' 'format(emucpu,5)
  lasttime = time()
  user.1.tcpu = totcpu
  totcpu = 0
  usercpu = 0
  syscpu = 0
  emucpu = 0
  do i = 1 to cpucnt
    tod_f = x2d(substr(sytprp.i,1,16))
    prbtm_f = x2d(substr(sytprp.i,22,15))
    utime_f = x2d(substr(sytprp.i,42,15))
    tmsys_f = x2d(substr(sytprp.i,62,15))
    intv = tod_f - mrhdrtod_l.i
    prbtm = pfxprbtm_l.i - prbtm_f
    utime = pfxutime_l.i - utime_f
    tmsys = pfxtmsys_l.i - tmsys_f
    totcpu = totcpu + trunc((utime + tmsys) / intv * 100 ,2)
    usercpu = usercpu + trunc(utime / intv * 100 ,2)
    syscpu = syscpu + trunc(tmsys / intv * 100 ,2)
    emucpu = emucpu + trunc(prbtm / intv * 100 ,2)
    /*-- store last values for next difference-calculation ------------*/
    mrhdrtod_l.i = tod_f
    pfxprbtm_l.i = prbtm_f
    pfxutime_l.i = utime_f
    pfxtmsys_l.i = tmsys_f
  end

  /*-- initialize user fields -----------------------------------------*/
  if init > 0 then init = init - 1
  do i = 2 to lstcnt
    user.i.tcpu = 0
    user.i.vcpu = 0
    user.i.wss = 0
      do j = 1 to user.i.numcpu
        user.i.currcpu = 1
      end
    end

    /*-- calculate user values and store in variable line ---------------*/
    do i = 1 to usrcnt
      name = substr(useact.i,1,8)
      /*-- only look for users in the list ------------------------------*/
      do j = 2 to lstcnt
        if name = list.j then do
          vmdttime_f = x2d(substr(useact.i,12,15))
          vmdvtime_f = x2d(substr(useact.i,32,15))
          k = user.j.currcpu
          user.j.currcpu = user.j.currcpu + 1
          if user.j.currcpu.k.vmdttime_l > 0 then do
            ttime = user.j.currcpu.k.vmdttime_l - vmdttime_f
            vtime = user.j.currcpu.k.vmdvtime_l - vmdvtime_f
            utotcpu = trunc(ttime / intv * 100,2)
            uvirtcpu = trunc(vtime / intv * 100,2)
            user.j.tcpu = trunc(user.j.tcpu + utotcpu,2)
            user.j.vcpu = trunc(user.j.vcpu + uvirtcpu,2)
            user.j.wss = user.j.wss + substr(useact.i,52,15)
            user.1.tcpu = user.1.tcpu - utotcpu
          end
          /*-- store last values for next calculation ------------------*/
          user.j.currcpu.k.vmdttime_l = vmdttime_f
          user.j.currcpu.k.vmdvtime_l = vmdvtime_f
        end
      end
    end
    do i = 1 to lstcnt
      line = line' 'list.i' 'user.i.tcpu' 'user.i.vcpu' 'user.i.wss
    end



    /*-- write the transformed data in daily output file (timestamped) --*/
    if init > 0 then init = init - 1
    if init = 0 then do
      call lineout 'MON $M'day' A' , line
      call lineout 'MON $M'day' A'
      if (substr(time(),1,5) = "23:50") then do
        call lineout , 'calling clean exec'
        call clean
      end
      day = date('U',,'')
    end
  end
  /*-- end of forever(main) loop (exit via #cp ext) ---------------------*/
```

Linux report system

Example 21 contains the HTML source for the index.shtml file discussed in "Report front end" on page 9.

Example 21 HTML front end (index.shtml)

```
<html>
<!-- frontend for VM accounting and monitor reports -->
<h2>Accounting</h2>
<form action="cgi-bin/accrep" method="post">
```

```
Linux System: <select name="sys" size="1">
<option selected>LIN161</option>
<option>LIN162</option>
<option>LIN163</option>
<option>LIN164</option>
<option>LIN165</option>
</select>
<input type="radio" name="type" value="html" checked> html
<input type="radio" name="type" value="text"> text<br><br>
from:<br>
Day <input name="fday" type="text" size="2" maxlength="2" value="<!--#exec cmd="date
--date '1 week ago' +%d" -->">
Month <input name="fmonth" type="text" size="2" maxlength="2" value="<!--#exec cmd="date
--date '1 week ago' +%m" -->">
Year <input name="fyear" type="text" size="2" maxlength="2" value="<!--#exec cmd="date
--date '1 week ago' +%y" -->"><br>
to:<br>
Day <input name="tday" type="text" size="2" maxlength="2" value="<!--#exec cmd="date
--date '1 day ago' +%d" -->">
Month <input name="tmonth" type="text" size="2" maxlength="2" value="<!--#exec cmd="date
--date '1 day ago' +%m" -->">
Year <input name="tyear" type="text" size="2" maxlength="2" value="<!--#exec cmd="date
--date '1 day ago' +%y" -->"><br>
<br>
<br>
<input type=submit value="Get Report"></form>
<br>
<hr>
<br>
<h2>Monitoring</h2>
<form action="cgi-bin/monrep" method="post">
<input type="radio" name="type" value="vm"> VM total<br>
<input type="radio" name="type" value="guests" checked> VM guests<br>
<input type="radio" name="type" value="linux"> Linux system:
<select name="sys" size="1">
<option selected>LIN161</option>
<option>LIN162</option>
<option>LIN163</option>
<option>LIN164</option>
<option>LIN165</option>
</select><br>
<br>
<br>
Day <input name="day" type="text" size="2" maxlength="2" value="<!--#exec cmd="date +%d"
-->">
Month <input name="month" type="text" size="2" maxlength="2" value="<!--#exec cmd="date
+%m" -->">
Year <input name="year" type="text" size="2" maxlength="2" value="<!--#exec cmd="date
+%y" -->">
 from <input name="from" type="text" size="5" maxlength="5" value="00:00">
 to <input name="to" type="text" size="5" maxlength="5" value="23:59"><br>
<br>
<input type=submit value="Get Report"></form><br>
</html>
```

Example 22 contains the complete listing of the **accrep** CGI script discussed in “Access to the data” on page 25.

Example 22 Account report CGI script (accrep)

```
#!/usr/bin/perl

# Perl CGI script to produce VM account reports
# output as HTML table or text file

use CGI;

$query = new CGI;

$fd = $query->param('fday');
$fm = $query->param('fmonth');
$fy = $query->param('fyear');
$td = $query->param('tday');
$tm = $query->param('tmonth');
$ty = $query->param('tyear');
$sys = $query->param('sys');
$type = $query->param('type');

# get files for selected time frame from VM NFS mount and store
# values for selected Linux system in /tmp/accdata
system("grep -h $sys /mnt/accsrv/acc.?a* | gawk
'\$2==\"$fm$fd$fy\",\$2==\"$tm$td$ty\"{print \$2,\$4,\$5,\$6,\$7}' > /tmp/accdata");

print "Content-type: text/html\n\n";

# produce a HTML table out of /tmp/accdata
if ($type eq "html") {
   print "<html>\n";
   print "<h3>Account data for system $sys from $fm/$fd/$fy to $tm/$td/$ty</h3>\n";
   print "<table border=\"1\">\n";
   print "<tr><th>Date</th><th>Total CPU time</th><th>Virtual CPU time</th><th>Bytes
send</th><th>Bytes received</th></tr>\n";
   system("gawk '{printf \"<tr
align=%s><td>%s</td><td>%s</td><td>%s</td><td>%s</td><td>%s</td></tr>\\n\",
\"\\\"right\\\"\", \$1, \$2, \$3, \$4, \$5}' /tmp/accdata");
   print "</table>\n";
   print "</html>";
}

# produce a text report out of /tmp/accdata
if ($type eq "text") {
   print "<pre>\n";
   print "Account data for system $sys from $fm/$fd/$fy to $tm/$td/$ty\n\n";
   print "Date      Total CPU time    Virtual CPU time          Bytes send      Bytes
received\n\n";
   system("gawk '{printf \"%6s %17s %19s %19s %19s\\n\", \$1, \$2, \$3, \$4, \$5}'
/tmp/accdata");
   print "</pre>\n";
}
```

Example 23 contains the complete listing of the **monrep** CGI script used to generated a monitor report.

Example 23 Monitoring report CGI script (monrep)

```
#!/usr/bin/perl

# Perl CGI script to produce VM and Linux CPU/MEM usage graphs
# the graphs are generated with gnuplot in png format
```

```
use CGI;

$query = new CGI;

$type = $query->param('type');
$sys = $query->param('sys');
$day = $query->param('day');
$month = $query->param('month');
$year = $query->param('year');
$from = $query->param('from');
$to = $query->param('to');

# specify input files
$safile = "sa.20$year\_$month\_$day";
$monfile = "/mnt/monsrv/mon.\$m$month$day$year";

# list of guests to be monitored
open(MON,"</mnt/monsrv/mon.list");
@guests = <MON>;
seek(MON,8,0);
read(MON,$cpu,2,0);
close(MON);

# the config file for gnuplot
open(DEM,"> /tmp/gnuplot.dem");

# report for a single Linux system
if ($type eq "linux") {
   $i = 1;
   # find the hostname/IP address
   foreach $guest (@guests) {
       ($guest,$proc,$mem,$hostname) = split(" ",$guest);
       last if ($guest eq uc($sys));
       $i++;
   }
   # get the sa log file and extract CPU and MEM usage values in temporary files
   @sar = `/usr/bin/ssh -f saruser\@$hostname sar -ur -f
/var/log/sa/sa.20$year\_$month\_$day`;
   @sarcpu = grep(/all/,@sar);
   pop(@sarcpu);
   open(SAR, "> /tmp/sarcpu");
   print(SAR @sarcpu);
   close(SAR);
   @sarmem = grep(!/[Lar]/,@sar);
   open(SAR, "> /tmp/sarmem");
   print(SAR @sarmem);
   close(SAR);

   # config file to plot the data from /tmp/sarcpu and /tmp/sarmem
   # and VM log file in the specified timeframe
   # Linux values are stacked type, VM values are single graphs
   print(DEM "set terminal png color\n" ,
       "set multiplot\n" ,
       "set title \"CPU usage for $sys ($month/$day/$year)\"\n" ,
       "set data style lines\n" ,
       "set size 1,0.5\n" ,
       "set origin 0.0,0.5\n" ,
       "set xdata time\n" ,
       "set ylabel \"CPU %\"\n" ,
       "set timefmt \"%H:%M:%S\"\n" ,
            "set yrange [0:" , $proc * 100 , "]\n" ,
            "set xrange [\"$from\":\"$to\"]\n" ,
            "set format x \"%H:%M\"\n" ,
            "plot '$monfile' using 1:(\$" , 4+$i*4 , ") t \"VM %cpu\" , " ,
            "'/tmp/sarcpu' using 1:((\$3+\$4+\$5)*$proc) t \"%user\" , " ,
            "'/tmp/sarcpu' using 1:((\$4+\$5)*$proc) t \"%nice\" , " ,
            "'/tmp/sarcpu' using 1:((\$5)*$proc) t \"%system\"\n" ,
            "set title \"Memory usage for $sys ($month/$day/$year)\"\n" ,
            "set data style lines\n" ,
            "set size 1,0.5\n" ,
            "set origin 0.0,0.0\n" ,
            "set xdata time\n" ,
            "set ylabel \"Memory (MB)\"\n" ,
            "set timefmt \"%H:%M:%S\"\n" ,
            "set yrange [0:" , $mem * 1.1 , "]\n" ,
            "set xrange [\"$from\":\"$to\"]\n" ,
            "set format x \"%H:%M\"\n" ,
            "plot '$monfile' using 1:(\$" , 5+$i*4 , "*4/1024) t \"VM WSS\" , " ,
            "'/tmp/sarmem' using 1:((\$3)/1024) t \"buffers\" , " ,
            "'/tmp/sarmem' using 1:((\$3-\$6)/1024) t \"cached\" , " ,
            "'/tmp/sarmem' using 1:((\$3-\$6-\$7)/1024) t \"used by appl\"\n" ,
            "set nomultiplot");
}

# z/VM system total report
if ($type eq "vm") {
   # config file to plot the system data from VM log data
   print(DEM "set title \"VM total CPU usage ($month/$day/$year)\"\n" ,
            "set terminal png color\n" ,
            "set data style lines\n" ,
            "set size 1,0.5\n" ,
            "set xdata time\n" ,
            "set ylabel \"CPU %\"\n" ,
            "set timefmt \"%H:%M:%S\"\n" ,
            "set yrange [0:", $cpu * 100 , "]\n" ,
            "set xrange [\"$from\":\"$to\"]\n" ,
            "set format x \"%H:%M\"\n" ,
            "plot '$monfile' using 1:2 t \"%CPU total\" , " ,
            "'$monfile' using 1:4 t \"%CP overhead\"\n");
}

# z/VM user report
if ($type eq "guests") {
   # config file to plot guest usage values from VM log data
   # values/graphs are stacked type
   print(DEM "set title \"VM CPU usage per guest ($month/$day/$year)\"\n" ,
            "set terminal png color\n" ,
            "set data style lines\n" ,
            "set size 1,0.5\n" ,
            "set xdata time\n" ,
            "set ylabel \"CPU %\"\n" ,
            "set timefmt \"%H:%M:%S\"\n" ,
            "set yrange [0:", $cpu * 100 , "]\n" ,
            "set xrange [\"$from\":\"$to\"]\n" ,
            "set format x \"%H:%M\"\n" ,
            "plot ");
   # routine to build the stacked graphs
   @guests = reverse @guests;
   $i = scalar(@guests);
   foreach $guest (@guests) {
```

```
            ($guest,$proc) = split(" ",$guest);
            print(DEM "'$monfile' using 1:(");
            for ($j = 1; $j <= $i; $j++) {
                print(DEM "\$",3+$j*4);
                if ($j < $i) {
                   print(DEM "+");
                } else {
                   print(DEM ") t \"$guest\"");
                   if ($i > 1) { print(DEM " , "); }
                }
            }
            $i--;
        }
    }
    close(DEM);

    print "Content-type: image/png\n\n";

    # plot!
    system("/usr/bin/gnuplot /tmp/gnuplot.dem");
```

Additional material

This Redpaper refers to additional material that can be downloaded from the Internet, as described below.

Locating the Web material

The Web material associated with this Redpaper is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

`ftp://www.redbooks.ibm.com/redbooks/REDP3818/`

Alternatively, you can go to the IBM Redbooks Web site at:

`http://www.redbooks.ibm.com/`

Click **Additional materials** and open the directory that corresponds with the redbook form number, REDP3818.

Obtaining the Web material

The additional Web material that accompanies this Redpaper includes the following files:

File name	*Description*
REDP3818.zip	Zipped code samples

How to use the Web material

Create a subdirectory (folder) on your workstation. Then unzip the contents of the Web material zip file into this folder. The extracted content is organized in three directories:

- **ACCSRV**
  This directory contains data and script files used by the ACCSRV z/VM virtual machine described in "The ACCSRV user" on page 5. Files include:
  - ACC.LIST
    The ACC LIST file shown in Example 1 on page 5
  - ACCSRV.DIRECT
    The z/VM user directory entry shown in Example 2 on page 5
  - CLEAN.EXEC
    The CLEAN EXEC script shown in Example 5 on page 8
  - PROFILE.EXEC
    The PROFILE EXEC for ACCSRV shown in Example 3 on page 6
  - READACC.EXEC
    The READACC EXEC script shown in Example 19 on page 32
  - WAKEUP.TIMES
    The WAKEUP TIMES file described in "Account record processing" on page 6
- **MONSRV**
  This directory contains data and script files used by the MONSRV z/VM virtual machine described in "The MONSRV user" on page 16. Files include:
  - CLEAN.EXEC
    The CLEAN EXEC script shown in Example 14 on page 23

  - COLLECT.REXX
    The COLLECT EXEC script shown in Example 20 on page 34
  - MON.LIST
    The MON LIST file shown in Example 10 on page 17
  - MONSRV.DIRECT
    The z/VM user directory entry shown in Example 9 on page 16
  - PROFILE.EXEC
    The PROFILE EXEC for MONSRV
  - RUN.EXEC
    The RUN EXEC script shown in Example 11 on page 21

- **replinux**
  This directory contains data and script files used by the Linux report system guest. Files include:
  - etc/fstab
    The /etc/fstab file needed to NFS-mount the ACCSRV user A-disk as discussed in "Access to the data" on page 25
  - etc/cron.d/sysstat
    The /etc/cron.d/sysstat file shown in Example 7 on page 14
  - srv/www/cgi-bin/accrep
    The Perl accrep CGI script shown in Example 22 on page 37
  - srv/www/cgi-bin/monrep
    The Perl monrep CGI script shown in Example 23 on page 38
  - srv/www/htdocs/index.shtml
    The report HTML front end shown in Example 21 on page 36
- **VMNFS**
  This directory contains configuration files for VMNFS. Files include:
  - PROFILE.TCPIP
    The PROFILE TCPIP file shown in Example 16 on page 32
  - SYSTEM.DTCPARMS
    The SYSTEM DTCPARMS file shown in Example 17 on page 32
  - VMNFS.CONFIG
    The VMNFS CONFIG file shown in Example 18 on page 32

Related publications

ITSO publications

- *Linux on IBM @server zSeries and S/390: Distributions*, SG24-6264

  http://www.ibm.com/redbooks/abstracts/sg246264.html
- *Linux on IBM @server zSeries and S/390: ISP/ASP Solutions*, SG24-6299

  http://www.ibm.com/redbooks/abstracts/sg246299.html
- *Linux on IBM @server zSeries and S/390: Performance Measurement and Tuning*, SG24-6926

  http://www.ibm.com/redbooks/abstracts/sg246926.html

Other resources

- *z/VM 4.4 CP Planning and Administration*, SC24-6043
- *z/VM 4.4 CP Command and Utility Reference*, SC24-608
- *Linux for zSeries and S/390 Device Drivers and Installation Commands*, LNUX-1303
- *SuSE Linux Enterprise Server 8 for IBM S/390 and IBM zSeries - Installation*

Referenced Web sites

- Computer Associates Home page

  `http://www.ca.com`
- CIMS Lab, Inc. Home page

  `http://www.cimslab.com`
- SAS home page

  `http://www.sas.com`
- z/VM CMS and CP Data Areas and Control Blocks

  `http://www.vm.ibm.com/pubs/ctlblk.html`
- Gnuplot Central

  `http://www.gnuplot.info`
- IBM z/VM and VM.ESA Home page

  `http://www.vm.ibm.com`
- Velocity Software, Inc. Home page

  `http://www.velocitysoftware.com`
- BMC Software home page

  `http://www.bmc.com`

The team that wrote this Redpaper

This Redpaper was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Erich Amrehn is a certified Senior IT Specialist at the EMEA Technical Marketing Competence Center (TMCC), Boeblingen, Germany. Before joining the TMCC, he worked as a project leader at the International Technical Support Organization, Poughkeepsie Center. During that time, he wrote Redbooks™ and taught Linux topics worldwide. Before joining the ITSO in 1998, he worked as a technical consultant to the IBM System/390® division for e-commerce on S/390® in Europe, the Middle East, and Africa. He also has 13 years of VM experience in various technical positions in Germany and other areas in Europe and worldwide.

Ronald Annuss is an IT Specialist working for IBM in Germany. He holds a diploma in Geodesy from the Technical University Berlin. He has worked with Linux for S/390 since it became available in early 2000, and has worked on several Linux for S/390 projects with clients in Germany. He is an Red Hat Certified Engineer (RHCE).

Arwed Tschoeke is an IBM zSeries Systems Engineer, located in Hamburg, Germany. He worked for four years in the xSeries® presales support team specializing in Linux and MSCS.

He currently focuses on zOS and cross platform solutions. He holds a degree in Physics from the University of Kaiserslautern, Germany.

Thanks to the following people for their contributions to this project:

Stuart Swain
International Technical Support Organization, Raleigh Center

Ella Buslovich
International Technical Support Organization, Poughkeepsie Center

Notices

This document created or updated on May 7, 2004.

Send us your comments in one of the following ways:

- Use the online **Contact us** review redbook form found at:
  ibm.com/redbooks
- Send your comments in an Internet note to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks