UNIX System

Remote Job Entry

This document is part of the ADMINISTRATOR'S GUIDE. Therefore the pagenumbers don't begin with 1.

10. UNIX SYSTEM REMOTE JOB ENTRY

GENERAL

A. Purpose

This section contains information on the design and operation of the UNIX System Remote Job Entry (RJE). In this document, RJE refers to the facilities provided by UNIX operating system and *not* to the Remote Job Entry feature of the HASP and JES2 subsystems produced by International Business Machines (IBM).

The information contained in this section should be used to augment the information contained in the UNIX System Administrator's Manual [rje(8)] and the "Remote Job Entry" section of the UNIX System User's Guide. There will be assumptions made concerning allocation of responsibilities between UNIX system and IBM operations, hardware configuration, etc. Although these assumptions may not fully apply to your location, they should not interfere with the intent of this document.

The major topics discussed in this document are as follows:

- SETTING UP—Hardware requirements and RJE generation on the IBM and UNIX systems.

- DIRECTORY STRUCTURES—The controlling RJE directory structure and a typical RJE subsystem directory structure.

- RJE PROGRAMS—Programs that make up an RJE subsystem.

- UTILITY PROGRAMS—Programs available for debugging or tracing.

- RJE ACCOUNTING—The accounting of jobs done by RJE and some methods for using this accounting data.

- TROUBLESHOOTING—Error recovery and procedures for identifying and fixing RJE problems.

B. Facilities

Discussions will focus on a hypothetical RJE connection between a UNIX system, whose nodename is *pwba*, and an IBM 370/168, referred to as *B*. We also assume that *pwba* is connected to an IBM 370/158, referred to as *C*. The UNIX operating system machine emulates an IBM System/360 remote multileaving work station.

SETTING UP

A. Hardware

In the remainder of this guide, the hardware described below will be referred to as the *physical device;* and its name will be referred to as **device?; where ?** is the device number. To use RJE on a UNIX system, the following hardware is needed:

DEC Hardware

- KMC11-B Microprocessor—Used to drive the RJE line.

- DMC11-DA or DMC11-FA line unit—The DMC11-DA interfaces with Bell 208 and 209 synchronous modems or equivalent. Speeds of up to 19,200 bits per second can be used. The DMC11-FA interfaces with Bell 500 A LI/5 synchronous modems or equivalent. Speeds of up to 250,000 bits per second can be used.

Each KMC/DMC pair supports a single RJE connection. On the DMC11 line unit, the Cyclic Redundancy Check (CRC) switch should be set to inhibit automatic transmission of CRC bytes. The line unit should hold the line at logical zero when inactive.

3B20S Processor Hardware

- TN82—The user interface board equipped with firmware for synchronous communications.

- UN53—Synchronous communications board. This board contains hardware to handle up to four synchronous lines. One line can be high-speed with V.35 or EIA interface.

One TN82/UN53 pair and the RJE software currently support up to four low-speed (9.6 KB or lower) RJE lines or one high-speed (56 KB) RJE line.

B. IBM Generation

The following applies to the host IBM system. The remote line to the UNIX operating system machine should be described as a System/360 remote work station. The following parameters must be initialized and must agree with their counterparts on the UNIX operating system machine:

- Number of printers (NUMPR)—The number of logical printers (up to 7)

- Number of punches (NUMPU)—The number of logical punches (up to 7)

- Number of readers (NUMRD)—The number of logical readers (up to 7).

The JES2 parameters for the hypothetical connection to IBM system *B* are as follows:

```
RMT5 S/360,LINE=5,CONSOLE,MULTI,TRANSP,NUMPR=5,
        NUMPU=1,NUMRD=5,ROUTECDE=5
R5.PR1 PRWIDTH=132
R5.PR2 PRWIDTH=132
R5.PR3 PRWIDTH=132
R5.PR4 PRWIDTH=132
R5.PR5 PRWIDTH=132
R5.PU1 NOSUSPND
R5.RD1 PRIOINC=0,PRIOLIM=14
R5.RD2 PRIOINC=0,PRIOLIM=14
R5.RD3 PRIOINC=0,PRIOLIM=14
R5.RD4 PRIOINC=0,PRIOLIM=14
R5.RD5 PRIOINC=0,PRIOLIM=14
```

System *pwba* is referenced by line 5 (LINE=5), remote 5 (RMT5). It is defined as having a console, for the rjestat(1C) command, five printers, one punch, and five readers. Although you may have up to seven printers or punches, the total number of printers and punches may not exceed eight. The line is described as a transparent (TRANSP), multileaving (MULTI) line. The remaining information describes attributes of the printers, punches, and readers.

Normally, separator pages are transmitted with IBM print files. UNIX system RJE does not remove separator pages. To prevent transmission of separator pages on printer 1 of the previous example, its attributes would be:

```
R5.PR1 PRWIDTH=132,NOSEP
```

NOSEP should be included for all printers when separator pages are not desired. Most IBM systems can also be told via a console command to cancel transmission of separator pages on printers. This can be done from the

IBM system console or from the remote UNIX operating system machine via **rjestat.** For example, the following JES2 command would cancel separator page transmission on printer 1:

$TR5.PR1,S=N

C. UNIX System Generation

If the RJE remote dialing facility is to be used, the administrator must make sure that the definition for RJECU in the file */usr/include/rje.h* is the device to be used for remote dialing. RJECU is defined to be */dev/dn2* when distributed. To compile and install RJE, the normal make(1) procedures are used (see the appropriate " Setting up the UNIX System " section of this guide). Once an RJE subsystem has been installed, the remote line must be described in the configuration file */usr/rje/lines.* This file as it exists on the hypothetical system *pwba* is as follows:

```
B   pwba   /usr/rje1   rje1   vpm0   5:5:1   1200:512:y
C   pwba   /usr/rje2   rje2   vpm1   1:1:1   1200:512
```

The */usr/rje/lines* is accessed by all components of RJE. Each line of the table (maximum of 8) defines an RJE connection. Its seven columns may be labeled *host, system, directory, prefix, device, peripherals,* and *parameters.* These columns are described as follows:

- *host*—The IBM System name, e.g., *A, B, C.* This string can be up to six characters long.

- *system*—The UNIX System nodename [see **uname**(1)].

- *directory*—The directory name of the servicing RJE subsystem (e.g., */usr/rje2*).

- *prefix*—The string prepended to most files and programs in the *directory* (i.e., *rje2*).

- *device*—The name of the controlling Virtual Protocol Machine (VPM) device, with */dev/* excised. In order to specify a VPM device, all VPM software must be installed, and the proper special files must be made [see **vpm**(7) and **mknod**(1M)].

- *peripherals*—Information on the logical devices (readers, printers, punches) used by RJE. There are three subfields. Each subfield is separated by ":" and is described as follows:

  1. Number of logical readers.

  2. Number of logical printers.

  3. Number of logical punches.

Note: The number of peripherals specified for an RJE subsystem *must* agree with the number of peripherals that have been described on the remote machine for that line.

- *parameters*—This field contains information on the type of connection to make. Each subfield is separated by ":". Any or all fields may be omitted; however, the fields are positional. All but trailing delimiters must be present. For example, in:

  1200:512:::9-555-1212

subfields 3 and 4 are missing. Each subfield is defined as follows:

  1. *space*—This subfield specifies the amount of space (*S*) in blocks that RJE tries to maintain on file systems it touches. The default is 0 blocks. **Send**(1C) will not submit jobs and rjeinit issues

a warning when less than 1.5S blocks are available; **rjerecv** stops accepting output from the host when the capacity falls to S blocks; RJE becomes dormant until conditions improve. If the space on the file system specified by the user on the "usr=" card would be depleted to a point below S, the file will be put in the *job* subdirectory of the connection's home directory rather than in the place that the user requested.

2. *size*—This subfield specifies the size in blocks of the largest file that can be accepted from the host without truncation taking place. The default is no truncation. Note the UNIX system has a default 1 megabyte file size limit.

3. *badjobs*—This subfield specifies what to do with undeliverable returning jobs. If an output file is undeliverable for any reason other than file system space limitations (e.g., missing or invalid "usr=" card) and this subfield contains the letter **y**, the output will be retained in the *job* subdirectory of the home directory; and login **rje** is notified via **mail**(1). If this subfield has any other value, undeliverable output will be discarded. The default is "n".

4. *console*—This subfield specifies the status of the interactive status terminal for this line. If the subfield contains an **i**, the status console facilities of **rjestat** will be inhibited. In all cases, the normal noninteractive uses of **rjestat** will continue to function. The default is "y".

5. *dial-up*—This subfield contains a telephone number to be used to call a host machine. The telephone number may contain the digits 0 through 9, and the character "-", which denotes a pause. If the telephone number is not present, no dialing is attempted; and a leased line is assumed.

When multiple readers have been specified, jobs that are submitted for transmission to IBM are assigned to the reader with the fewest cards on it. Each reader gets an equal amount of service. This prevents smaller jobs from having to wait for a previously submitted large job to be transmitted. When multiple printers or punches have been specified, returning jobs get assigned to free printers (or punches) allowing smaller output files to bypass large output files.

Deciding how many peripherals to specify depends on the use of that RJE subsystem. If an RJE subsystem is heavily used for off-line printing (i.e., output does not return to the UNIX operating system machine), the administrator would want to specify multiple readers but would not have a need for multiple printers or punches.

DIRECTORY STRUCTURES

A.   Controlling Directory

The controlling directory used by RJE is */usr/rje*. This directory contains RJE programs for use by separate RJE subsystems (e.g., *rje1, rje2, rje3*) and the shell queuer's directory. Most RJE programs existing here have been compiled such that each RJE subsystem shares the text of these programs. A snapshot of this directory on our hypothetical machine is as follows:

```
-rwxr-xr-x     3 rje    rje     4068 Mar  4 10:42 cvt
-rw-r--r--     1 rje    rje       42 Apr 10 09:52 lines
-rwxr-xr-x     3 rje    rje    15096 Apr 10 13:01 rjedisp
-rwxr-xr-x     3 rje    rje     2328 Mar  4 10:21 rjehalt
-rwxr-xr-x     3 rje    rje    10396 Apr 15 10:07 rjeinit
-r-x------     3 rje    rje      785 Apr  8 09:00 rjeload
```

```
-rwsr-xr-x   3 rje   rje    5040 Mar 27 09:28 rjeqer
-rwxr-xr-x   3 rje   rje    4072 Apr  1 15:40 rjerecv
-rwxr-xr-x   3 rje   rje    3888 Mar 27 09:35 rjexmit
-rwsr-xr-x   1 root  rje    2696 Mar 27 14:42 shqer
-rwxr-xr-x   3 rje   rje    5920 Apr  2 15:47 snoop
drwxr-xr-r   2 rje   rje      80 Mar 25 13:26 sque
```

The RJE subsystems are generated in their own directory by linking the program names in this directory to the appropriate names in the subsystem directory. The programs are described in the part "RJE PROGRAMS". The file *lines* is the configuration file used by all RJE subsystems. The directory *sque* is used by the shell queuer (**shqer**). This directory contains:

```
-rw-r--r--   1 rje   rje       0 Feb 14 14:04 errors
-rw-r--r--   1 rje   rje       0 Feb 14 14:04 log
```

When **shqer** has work to do, the files *log* and *errors* will be of nonzero length; and temporary files (*tmp***) will also appear here.

B. Subsystem Directory

The RJE subsystem described in this section maintains the connection between *pwba* and IBM *B* and will be referred to as *rje1*. The first line of */usr/rje/lines* describes *rje1*. As noted in this file, *rje1* runs in the directory */usr/rje1*. A snapshot of this directory is as follows:

```
-rw-r--r--   1 rje   rje    4990 Apr 15 08:30 acctlog
-rwxr-xr-x   3 rje   rje    4068 Mar  4 10:42 cvt
-rw-r--r--   1 rje   rje       0 Apr 15 04:02 errlog
drwxrwxrwx   2 rje   rje     192 Apr 10 09:51 job
-rw-r--r--   1 rje   rje     194 Apr 15 08:11 joblog
-rw-r--r--   1 rje   rje       0 Apr 15 08:11 resp
-rwxr-xr-x   3 rje   rje   15096 Apr 10 13:01 rjeldisp
-rwxr-xr-x   3 rje   rje    2328 Mar  4 10:21 rjelhalt
-rwxr-xr-x   3 rje   rje   10396 Apr 15 10:07 rjelinit
```

```
-r-x------      3 rje   rje      785 Apr  8 09:00 rjelload
-rwsr-xr-x      3 rje   rje     5040 Mar 27 09:28 rjelqer
-rwxr-xr-x      3 rje   rje     4072 Apr  1 15:40 rjelrecv
-rwxr-xr-x      3 rje   rje     3888 Mar 27 09:35 rjelxmit
drwxr-xr-x      2 rje   rje      144 Apr 15 08:30 rpool
-rwxr-xr-x      3 rje   rje     5920 Apr  2 15:47 snoop0
drwxrwxrwx      2 rje   rje      176 Apr 10 13:03 spool
drwxr-xr-x      2 rje   rje      224 Apr 10 13:56 squeue
-rw-r--r--      1 rje   rje        0 Apr 15 10:30 stop
-rw-r--r--      1 rje   rje      274 Mar  7 20:25 testjob
```

The programs rje1*, cvt, and snoop0 are linked to the corresponding programs in /usr/rje. The remaining files and their uses are as follows:

- *acctlog*—Accounting data is stored in this file if it exists. This file is the responsibility of the RJE administrator.

- *errlog*—Used by *rjel* to log errors. It can be useful for debugging *rjel* problems.

- *joblog*—Used by rje1qer and rjestat to notify rje1xmit that a job (or console request) has been submitted. It also contains the process-group number of the *rjel* processes. The program cvt can be used to convert this file to a readable form.

- *resp*—Contains console messages received from IBM *R* These messages can be responses for rjestat or IBM responses to submitted jobs (i.e., on reader messages). This file is truncated if it grows to a size greater than 70,000 bytes.

- *stop*—Indicates that rje1halt has been executed. The existence of this file indicates to rjestat that *rjel* has been halted by the operator.

- *testjob*—A sample job that can be submitted to test the *rjel* subsystem. Originally, the job control statements may have to be changed to suit your IBM system.

When *rjel* terminates abnormally, the file *dead* should appear in this directory. This file contains a short message indicating why *rjel* is not operating and is used by rjestat to report the problem. The remaining directories and their uses are as follows:

- *job*—Used to save undeliverable jobs if the proper parameter has been specified in /usr/rje/lines. The sample job described above is also delivered to this directory. This directory should be mode 777.

- *rpool*—Contains temporary files used to gather output from the remote machine. These files are named *pr**(for print output files) and *pu**(for punch output files). Once a complete file has been received, the file is dispatched in the proper way by rje1disp.

- *spool*—Used by send to store temporary files to be submitted to the remote machine. This directory must be mode 777.

- *squeue*—Used by *rje1* to store submitted files until they are transmitted. The program **rje1qer** is used by **send** to move the temporary files in the *spool* directory to this directory.

RJE PROGRAMS

All programs described below, with the exception of **rjestat**, exist in */usr/rje*. These programs are "shared text" and are linked (except **shqer**) to the proper names in each subsystem directory. The names described below are generic; the programs in the *rje2* directory would be **rje2qer, rje2init**, etc.

Each available RJE subsystem occupies three process slots. The slots used are **rje?xmit** for the transmitter, **rje?recv** for the receiver, and **rje?disp** for the dispatcher. One additional process slot is used for **shqer** regardless of how many subsystems are available.

Each RJE subsystem tries to be self-sustaining and logs any errors encountered during normal operation in its *errlog* file.

A. Rjeqer

This program is used by **send** to queue files for transmission. When invoked, it performs the following steps:

1. Moves temporary *pnch*(4) format file in *spool* directory to *squeue* directory.

2. Writes an entry at end of file *joblog* containing:

   - name of file to be transmitted

   - submitter's user ID

   - number of card images in file

   - message level for this job.

   The file *joblog* is used to notify **rjexmit** of work to be done.

3. Notifies user that file has been queued.

Send determines which host system is desired and invokes the proper **rje?qer** by getting the *prefix* from the *lines* file (e.g., if sending to IBM *C* from our machine, **rje2qer** would be invoked).

B. Rjeload

This program is used to start an RJE subsystem. Its prefix determines the subsystem to start (e.g., rje2load starts *rje2*). To start the RJE subsystems on UNIX operating system machines, the following commands are executed in */etc/rc* when changing to *init* state 2 (multiuser):

```
rm −f /usr/rje/sque/log
su rje −c  " /usr/rje1/rje1load device0 "
su rje −c  " /usr/rje2/rje2load device1 "
[If the V.35 interface (see vpmstart(1C) for 3B-20 only)
is used for, say, subsystem rje1,
then use command line:
su rje −c  " V_35=1 /usr/rje1/rje1load device0 "  ]
```

The file */usr/rje/sque/log* is removed to ensure the correct operation of **shqer**. When invoked, **rjeload** performs the following steps:

1. Uses VPM device from */usr/rje/lines* to link the proper devices [see **vpmset(1C)**].

2. Loads device given as argument with RJE protocol script.

3. Executes **rje?init** to start **rje?** processes (e.g., **rje2load** executes **rje2init**).

C.   Rjehalt

This program is used to halt an RJE subsystem. To halt *rje2* on UNIX operating system machines, */usr/rje2/rje2halt* is executed. This should be done in the **shutdown** procedure for your machine to ensure graceful termination of RJE. Rjehalt will allow only those users with permission to halt an RJE subsystem. Rjehalt uses the header on the file *joblog* to get the process-group of the RJE subsystem processes. This group is signaled to terminate. When all processes have terminated, **rjehalt** sends a "signoff" record to the host machine. This signoff record is taken from the file *signoff* (ASCII text) if it exists; otherwise, a "/*signoff" record is sent. On completion, **rjehalt** creates the file *stop* in the subsystem directory. The presence of the file *stop* in a subsystem directory causes **rjestat** to report to users that RJE to the corresponding host has been stopped by the operator.

D.   Rjeinit

This program initializes an RJE subsystem. It is used by **rjeload** and can be used to restart a subsystem if the VPM script has previously been started. **Rjeinit** should only be executed by user *rje*. Rjeinit fails if there are less than 100 blocks or 10 inodes free in the file system. It issues a warning if there are less than 1.5X blocks (where X is the first field in the parameters for that line) or 100 inodes free in the file system. If **rjeinit** fails, the reason for the failure is reported; and the file *dead* is created containing "Init failed". This will be reported by **rjestat** until a subsequent **rjeinit** succeeds. The **rjeinit** performs the following functions:

1. Dials a remote host if specified.

2. Truncates console response file *resp*.

3. Sends a signon record to the host. The signon record is taken from file *signon* (ASCII text) if it exists; otherwise, **rjeinit** sends a blank record as a signon.

4. Sets up pipes for process communication.

5. Resets process-group for RJE subsystem and restarts error logging.

6. Rebuilds *joblog* file from jobs queued for transmission.

7. Notifies **rjedisp** (via a pipe) of any returned files still remaining in *rpool* directory.

8. Starts appropriate background processes **rje?xmit, rje?recv, and rje?disp**.

9. Reports started or not started.

If failure occurs in a background process, it is reported by that process (error logging). The failing process will normally attempt to reboot the subsystem by executing **rje?init** with a + as its argument. When **rjeinit** is executed with + as its argument, this indicates an attempted reboot; and **rjeinit** will behave differently.

E.   Rjexmit

This program writes data to the VPM device. The **rjexmit** is started by **rjeinit** and runs in the background. When running, rjexmit performs the following processing:

1. Checks *joblog* file for files to be transmitted. This is done every 5 seconds when not transmitting data. When transmitting data, the *joblog* is checked after transmitting one block from each active *reader* and

console (*Reader* refers to the logical readers used by RJE. *Console* refers to the RJE logical console which is separate from the logical readers.)

2. Queues files from *joblog* according to first two characters of the file name:

  - *rd**—These files are queued on the reader with the fewest cards. Normal use of the **send** command creates these files.

  - *sq**—These files are queued on the last available reader to assure sequential transmission. Using the **−x** option to the **send** command creates these files.

  - *co**—These files are queued on the console. The **rjestat** command creates these files. All files described above contain EBCDIC data.

3. Sends information to **rjedisp** (via a pipe) for use in user notification of job status.

4. Builds blocks for transmission from active readers and the console. These blocks are built according to the multileaving protocol.

5. Performs following peripheral control:

  - Sends requests to open readers when jobs have been assigned to them. These readers are not active until a grant is received from **rjerecv** (via a pipe).

  - Halts and activates readers when waits or starts (respectively) are received from **rjerecv**.

  - Sends printer or punch grants when an open request is received from **rjerecv**.

6. Notifies **rjedisp** that a file has been transmitted and unlinks the file.

If **rjexmit** encounters fatal errors, it creates the *dead* file with an appropriate message and signals the other background processes to exit. If possible, **rjexmit** will attempt to reboot the RJE subsystem by executing **rjeinit**.

F.   Rjerecv

This program reads data from the VPM device. The **rjerecv** is started by **rjeinit** and runs in the background. When running, **rjerecv** performs the following processing:

1. Reads blocks of data received from host system.

2. Handles data received according to its type. The two types of data are:

  - **Control information**—Rjerecv performs the following peripheral device control:

    a.   Notifies **rjexmit** of grants to its requests to open readers.

    b.   Passes wait and start reader information to **rjexmit**.

    c.   Passes open requests (for printers and punches) from the host to **rjexmit**.

  - **User Information**—The three major types of user information received are:

    a.   Console responses and job status messages. This data is appended to the *resp* file for use by **rjestat** and **rjedisp**.

    b.   The printer output from user jobs. This data is collected in temporary files (*pr*) in the *rpool* directory. When a complete print job has been received, **rjerecv** notifies **rjedisp** (via a pipe) that the file is to be dispatched.

    c.   The punch output from user jobs. This data is handled the same as printer output except that the *rpool* files are named *pu*.

3. If console response file *resp* exceeds 70,000 characters, **rjerecv** truncates file.

4. **Rjerecv** stops accepting output from the remote machine if the number of free blocks in the file system falls below *space* blocks.

5. **Rjerecv** truncates files to *size* blocks if a received file exceeds this value.

If **rjerecv** encounters fatal errors, it creates the *dead* file with an appropriate error message, signals the other background processes to exit, and reboots the RJE subsystem.

G.   Rjedisp

This program dispatches user information. **Rjedisp** is started by **rjeinit** and runs in the background. When running, **rjedisp** performs the following processing:

1. Dispatches output. The two types of output are printer and punch output. After receiving notification of output ready from **rjerecv**, **rjedisp** searches for a "usr=" line in the received file. The format of a "usr=" line is as follows:

           usr=(user,place,level)

Rjedisp dispatches output according to the place field.

2. Dispatches messages. The three types of messages are as follows:

- Job transmitted—This message is sent to the submitting user when **rjedisp** reads this event notice from the **rjexmit** pipe.

- Job acknowledgment—**rjedisp** dispatches IBM acknowledgment messages to submitting users.

- Output processing—**rjedisp** dispatches job output messages according to the options specified on the "usr=" card. A normal output message indicates the returned file name is ready. Messages can be masked by using the *level* on the "usr=" card.

3. Whenever output is to be handled by **shqer**, **rjedisp** checks that **shqer** is running. This is done by looking for the **shqer** *log* file. If this file does not exist, **rjedisp** starts **shqer**.

H.   Shqer

This program executes user programs when they appear in the *place* field of the "usr=" line in a returned output file (print or punch). **Shqer** is started by **rjedisp** when the first output file using this feature is returned. Subsequent files using this feature are logged for execution by **rjedisp**. When started, **shqer** performs the following processing:

1. Builds the *log* file from file names in */usr/rje/sque* directory. Each log entry is the name of a file (*tmp?*) that contains the following information:

- name of file to be executed

- name of input file (file returned from IBM)

- name of IBM job

- programmer's name

- IBM job number

- user's name from "usr=" line

- user's login directory

- minimum file system space.

2. **Shqer** uses two parameters. The first is the delay time between *log* file reads. The second is a *nice*(2) factor which is applied to any programs spawned by **shqer**. These values are defined in */usr/include/rje.h* (*QDELAY* and *QNICE*).

3. When each log entry is read, the appropriate program is spawned with the following characteristics:

- The returned RJE file is standard input to the program.

- The standard and diagnostic outputs are */dev/null*

- The LOGNAME, HOME, and TZ variables are set to appropriate values.

- The arguments to the spawned program, in order, are:

  a. a numerical value indicating that file system free space is equal or above (0) or below (1) *space* blocks.

  b. IBM job name.

  c. programmer's name.

  d. IBM job number.

  e. user's login name.

4. After executing each program, the *tmp?* file and returned RJE file are removed.

UTILITY PROGRAMS

A. Snoop

Snoop is the generic name of a program that can be used to trace the state of a VPM device and its associated communications line. **Snoop** depends on the **trace**(7) driver for its information. It reads trace entries from */dev/trace* and converts them into a readable form that is printed on the standard output.

The usable name of **snoop** for a particular RJE subsystem is **snoopN**, where **N** is the minor device number of the VPM device. In our hypothetical system, *vpm0* is used by the *rje1* subsystem; and *vpm1* is used by the *rje2* subsystem. Therefore, */usr/rje1/snoop0* and */usr/rje2/snoop1* are linked to */usr/rje/snoop.*

Each snoop prints trace entries for its associated VPM device. Trace entries are printed in the following form:

> sequence   type   information

where:

- *sequence* specifies the order of trace occurrences. It is a value between 0 and 99.

- *type* specifies the action being traced (e.g., transfers, driver activity).

- *information* describes data being transferred and driver activity.

Refer to Table 10.A for the meaning of the trace *types* and associated *information*.

B.   Rjestat

This program is supplied as a user command. The program's two functions are to describe the status of the RJE subsystems and to provide a remote IBM status console. The remainder of this part describes these two functions.

RJE Status

When invoked, rjestat reports the status of the RJE subsystems. If remote system ("host") names are specified, only those statuses are reported. The rjestat uses the following rules to report the status of a subsystem:

- Rjestat prints the contents of the file *status* if it exists in the subsystem directory. This file can contain any message the administrator wishes to have printed when users use rjestat.

- If the file *dead* exists in the subsystem's directory, the subsystem is not operating and the reason is contained in the file. The rjestat reports that RJE to "host" is down and prints the contents of the *dead* file as the reason.

- If the file *stop* exists in the subsystems directory, the rjehalt program has been used to inhibit that RJE subsystem. Rjestat reports that RJE to "host" has been stopped by the operator.

- If neither the *dead* nor the *stop* file exists, rjestat reports that RJE to "host" is operating normally.

Rjestat is supplied as the user's vehicle for checking the status of RJE. It is not meant to be an administrative tool; however, the reason for failure can be used to track the problem.

Status Console

To use rjestat as a status console, the −s *host* argument is used. Rjestat prints the status of the subsystem, then prompts with *host*: if the subsystem is up. Each console request is submitted to the RJE processes for transmission, and output is handled as specified. Rjestat checks the status prior to submitting each request and will tell the user to try later if the subsystem goes down. Rjestat allows the rje or superuser logins to submit other than display requests. For a complete description of how to use the status console features, see rjestat(1C).

C.   Cvt

This program converts any subsystem's *joblog* file to readable form. The first line printed is the process group number of the subsystem processes. The remaining output consists of entries in the following form:

> file   user-id   records   level

where "file" is the name of the submitted file, "user-id" is the submitters user number, "records" is the number of card images, and "level" is the message level. The "records" and "level" fields are not used if the file name is *co** (console request submitted by rjestat).

RJE ACCOUNTING

Each RJE subsystem will store accounting information in the *acctlog* file if it exists. It is the responsibility of the RJE administrator to create and maintain this file in the subsystem's directory. Entries in this file describe RJE line use and are of the following form:

> day   time   file   **user**   records

Each field is delimited by a tab character. The meanings of each field is as follows:

1. day—The day of occurrence in the form *mm/dd.*

2. time—The time of occurrence in the form *hh:mm:ss.*

3. file—The name of the UNIX system file. The first two characters identify its type as follows:

   - *rd/sq*—The file was transmitted to the remote system.

   - *pr*—The print output file was received from the remote system.

   - *pu*—The punch output file was received from the remote system.

4. user—The user ID of the user responsible for the transfer.

5. records—The number of records (card images) transferred for this file.

Since *acctlog* data is not used by RJE, it should not be allowed to grow too large. This can be accomplished by moving or processing the file during a system reboot (i.e., in */etc/rc* before the RJE subsystems are started).

The following list describes some of the reports that could be generated from the *acctlog* data. Implementation of a program to produce accounting reports is the responsibility of the administrator.

   - Periodic Reports—By using the "day" and "time" fields in the data, periodic usage reports can be produced.

   - By User Reports—By using the "user" field in the data, usage-by-user reports can be produced.

   - By Subsystem Reports—By using the */usr/rje/lines* file information and each *acctlog* file, a usage-by-subsystem (or remote system) report can be produced. Other reports can be produced using the type of file, size of jobs, etc.

TROUBLESHOOTING

This section deals with RJE problems and some methods for resolving them. The topics discussed in this section are as follows:

   - Automatic Error Recovery

   - Manual Error Recovery

   - RJE Problems

   - VPM Problems

   - Trace Interpretation.

A.   Automatic Error Recovery

RJE attempts to be self-sustaining with respect to its availability. In general, if problems occur on the communications line or the remote machine (e.g., a crash), RJE will continually try to restart itself (this action will be referred to as "reboot"). For example, if an RJE subsystem is started using **rjeload** but the IBM system is not available, a fatal error will occur. The process that detects this error (usually **rjexmit** or **rjerecv**) will reboot the subsystem by executing **rjeinit** with a + as its argument. When **rjeinit** detects a + argument, it waits 1 minute before attempting to bring up the subsystem.

The **rjehalt** program can be used to prevent an RJE subsystem from rebooting itself when the remote system is not available for a known period of time. When the remote system is made available, the subsystem may be started in the normal way.

B.   Manual Error Recovery

In order to manually recover from errors, one must know how to start and stop an RJE subsystem. There are two ways to start an RJE subsystem:

- **rje?load**—This program loads and starts the VPM script and executes **rje?init**.

- **rje?init**—This program starts the *rje?* subsystem. In order to use this program, the VPM script must be loaded and started.

To stop the *rje?* subsystem, the **rje?halt** program should be executed. This stops the subsystem gracefully and will prevent a reboot.

The **rjeload** program must be used to start RJE for the first time (after a UNIX system reboot). Subsequently, as long as the script is running, execution sequences of **rjehalt** and **rjeinit** will stop and start RJE.

Manually starting and stopping RJE can be useful in tracking down problems. For example, if user jobs are not being submitted to the host machine, the following sequence can ease identification of the problem:

1. Halt ailing subsystem.

2. Start a **snoop** process in the background with its output redirected to a file.

3. Restart subsystem.

4. Scan **snoop** output to determine where the problem is.

The snoop program is the most useful software tool for identifying RJE problems. Its uses are described in the subpart "Trace Interpretation".

C.   RJE Problems

This section describes problems that can occur in an RJE subsystem. These problems generally occur when the subsystem has not been set up properly. The following is a list of things to check to ensure that an RJE subsystem has been set up properly.

1. IBM description—The description of the remote UNIX machine must be consistent with the description in the subpart "IBM Generation".

2. UNIX system description—The file */usr/rje/lines* must be set up properly. The subpart "UNIX System Generation" describes this file in detail.

3. VPM setup—The VPM software must be installed and the proper VPM and physical devices made. Each VPM device must correspond to the proper physical device; see **vpm(7)**.

4. Free space—As a general rule, all file systems must have a reasonable amount of free space. File systems containing RJE subsystems must have sufficient free space to ensure proper RJE operation.

5. Directories—Each subsystem's directory and the controlling directory should be checked for the following:

- All needed files exist.

- The proper prefix is on each applicable RJE program.

- The link count is correct for files that are linked.

- All file and directory modes are correct.

6. Initialization—Peripherals information must be consistent on both systems. The line must be started on the IBM system, proper hardware connections made, etc.

Problems with a subsystem are indicated by error messages. The rjeinit checks for obstacles in bringing up RJE. If an obstacle is found, an error message indicating the obstacle is printed on the error output. If a problem is encountered during normal operation, the message is logged in the *errlog* file. This file, error messages, output from **snoop**, and the checklist above should be used to determine and fix any subsystem problems. Generally, if a subsystem is set up properly but will not operate, the problem is the way the VPM or KMC has been set up, the remote system, or the hardware.

D. VPM Problems

After installing the hardware and making the appropriate devices, all VPM software and devices must be made [see **vpm(7)**]. The program **rjeload** links the devices to be used by the corresponding RJE subsystem.

The following is a list of items to check when problems occur:

1. Proper hardware—The appropriate hardware must be installed. Be sure the device is properly described to the system and passes diagnostics.

2. Proper devices—The major and minor device numbers for the physical device and VPM devices must be correct. It should also be verified that **rjeload** program is called with the correct device names.

3. Script runs—Verify the VPM script is able to run. This is done by tracing the proper device with the proper **snoop** program. Snoop will print "started" entries for both the physical device and VPM script. If no output appears from **snoop** when **rjeload** is executed, either the hardware is not working properly or the hardware or VPM has not been set up properly. Output of any other type from **snoop** should indicate where the problem is occurring.

E. Trace Interpretation

This part describes how to interpret trace output from the **snoop** program and gives several examples.

Lines with type TR are traces from the VPM script. All others are driver traces and indicate the following:

- CL—Activity occurring when the device has been closed.

- OP—Activity occurring when the device has been opened.

- RD—Read from device occurred.

- WR—Write to device occurred.

- ST—Start or stop activity.

- SC—Script termination type, termination value is given.

Table 10.A enumerates all possible trace lines for each type and describes the event. The remainder of this part consists of example trace output and its interpretation. Comments describing events will appear after the "*" in trace output. If more than one VPM were running, sequence numbers might not appear in order. For clarity, example sequences will be in order.

Normal RJE Startup

The following is an example of trace output when RJE has been started up. In this case the remote machine responds to the enquiry byte (ENQ). The RJE subsystem signs on to the machine then follows the handshaking protocol (exchanging ACKs).

```
Tracing device0
  0  ST    Startack      *  Physical device started
  1  TR    Started       *  Script started
  2  ST    Start         *  VPM Driver start
  3  OP    Opened        *  VPM Device open
  4  WR    84 bytes      *  Signon record written
  5  TR    S-ENQ         *  Enquiry byte sent
  6  TR    R-ACK         *  Received acknowledgment
  7  TR    S-BLK         *  Sent signon block
  8  TR    R-ACK         *  Block acknowledged
  9  TR    S-ACK         *  Handshaking
 10  TR    R-ACK         *       •
 11  TR    S-ACK         *       •
 12  TR    R-ACK         *       •
 13  TR    S-ACK         *       •
 14  TR    R-ACK         *       •
 15  TR    S-ACK         *       •
 16  TR    R-ACK         *       •
 17  TR    S-ACK         *  Handshaking
```

If any jobs had been submitted via the send command or jobs were waiting to be returned, the traces would reflect the transfers rather than handshaking.

RJE Startup—IBM Not Responding

This example shows trace output when RJE has been started but does not receive a response from the remote machine. In general, the RJE script will time-out if a response is not received from the remote machine within 3 seconds of the last transmission. When a time-out is detected while starting up, the ENQ is retransmitted. This is repeated six times before the script gives up. Other time-out responses will be discussed later.

Tracing device0

86	ST	Startack	*	Physical device started
87	TR	Started	*	Script started
88	ST	Start	*	VPM driver start
89	OP	Opened	*	VPM device open
90	WR	84 bytes	*	Signon record written
91	TR	S-ENQ	*	Enquiry byte sent
92	TR	TIME-OUT	*	No response to enquiry
93	TR	S-ENQ	*	Enquiry byte sent
94	TR	TIME-OUT	*	No response
95	TR	S-ENQ	*	Enquiry byte sent
96	TR	TIME-OUT	*	No response
97	TR	S-ENQ	*	Enquiry byte sent
98	TR	TIME-OUT	*	No response
99	TR	S-ENQ	*	Enquiry byte sent
0	TR	TIME-OUT	*	No response
1	TR	S-ENQ	*	Enquiry byte sent
2	TR	TIME-OUT	*	No response
3	RD	1 bytes	*	1 byte read (error)
4	ST	Stopchk	*	Safety check
5	ST	Stopack(0)	*	Script termination normal
6	CL	Clean	*	Cleanup done
7	ST	Stopped	*	VPM script stopped
8	CL	Closed	*	VPM device closed

The above sequence will be repeated approximately every minute until a positive response is received from the host. During that minute, the RJE subsystem is dormant; and the rjestat command will report that IBM is not responding. When this occurs, either the IBM machine is not available, down, line not started, etc.; or there is a communications problem somewhere from where the physical device transmits data to where it receives data. The RJE administrator should first verify that the IBM machine is up, and the communications line has been started. If so, a hardware trace of the communications line should be done to aid in detecting the problem.

Transmitting and Receiving

This example shows trace output from the start of job transmission through its return. For simplicity, only one job is being transmitted and returned.

Tracing device0

94	TR	R-ACK	*	Handshaking
95	TR	S-ACK	*	•
96	TR	R-ACK	*	•
97	TR	S-ACK	*	Handshaking
98	WR	4 bytes	*	Open reader request written
99	TR	R-ACK	*	Handshaking
0	TR	S-BLK	*	Sent open request block
1	TR	R-OKBLK	*	Received block (grant)
2	TR	S-ACK	*	Block acknowledged
3	RD	7 bytes	*	Read seven bytes (grant)
4	TR	R-ACK	*	Handshaking
5	TR	S-ACK	*	Handshaking
6	WR	481 bytes	*	First block written
7	WR	470 bytes	*	Second block written
8	TR	R-ACK	*	Handshaking
9	TR	S-BLK	*	First block sent
10	TR	R-ACK	*	Block acknowledged
11	WR	470 bytes	*	Third block written
12	TR	S-BLK	*	Second block sent
13	TR	R-OKBLK	*	Received block (on reader msg)
14	WR	470 bytes	*	Fourth block written
15	RD	66 bytes	*	Read 66 bytes (on reader msg)
16	TR	S-BLK	*	Third block sent
17	TR	R-ACK	*	Block acknowledged
18	WR	147 bytes	*	Fifth block written
19	TR	S-BLK	*	Fourth block sent
20	TR	R-ACK	*	Block acknowledged
	•		*	
	•		*	More of the same
	•		*	
93	TR	R-ACK	*	Handshaking
94	TR	S-ACK	*	Handshaking
95	TR	R-OKBLK	*	Received block (request)
96	TR	S-ACK	*	Block acknowledged
97	RD	7 bytes	*	Read open printer request
98	TR	R-ACK	*	Handshaking
99	TR	S-ACK	*	•
0	TR	R-ACK	*	•
1	TR	S-ACK	*	•
2	TR	R-ACK	*	•

3	TR	S-ACK	*	Handshaking
4	WR	4 bytes	*	Printer grant written
5	TR	R-ACK	*	Handshaking
6	TR	S-BLK	*	Block sent (grant)
7	TR	R-OKBLK	*	First block received
8	TR	S-ACK	*	Block acknowledged
9	RD	64 bytes	*	Read first block
10	TR	R-OKBLK	*	Second block received
11	TR	S-ACK	*	Block acknowledged
12	RD	505 bytes	*	Read second block
13	TR	R-OKBLK	*	Third block received
14	TR	S-ACK	*	Block acknowledged
15	TR	R-OKBLK	*	Fourth block received
16	TR	S-ACK	*	Block acknowledged
17	TR	R-ACK	*	Handshaking
18	TR	S-ACK	*	●
19	TR	R-ACK	*	●
20	TR	S-ACK	*	Handshaking
21	RD	470 bytes	*	Read third block
22	RD	494 bytes	*	Read fourth block
23	TR	R-ACK	*	Handshaking
24	TR	S-ACK	*	Handshaking
●			*	
●			*	etc.
●			*	

Requests and grants are part of the multileaving protocol. When jobs are being transmitted and received simultaneously, as in a busier RJE subsystem, much less handshaking is involved. Rather than acknowledging blocks with ACKs, the protocol allows a block to be returned (this implies acknowledgment of the received block). The following example shows trace output at a busy time:

tracing device0

45	TR	R-OKBLK	*	Received block
46	TR	S-BLK	*	Sent block
47	WR	493 bytes	*	
48	RD	496 bytes	*	
49	TR	R-OKBLK	*	Received block
50	RD	65 bytes	*	
51	WR	4 bytes	*	
52	TR	S-BLK	*	Sent block
53	TR	R-OKBLK	*	Received block
54	TR	S-BLK	*	Sent block
55	WR	493 bytes	*	
56	RD	7 bytes	*	
57	TR	R-OKBLK	*	Received block
58	WR	493 bytes	*	
59	RD	496 bytes	*	
60	TR	S-BLK	*	Sent block
61	TR	R-OKBLK	*	Received block

Notice that since there is work to be done on both sides acknowledgments are implied.

Time-out Error Recovery

This example shows activity resulting from time-outs occurring during normal operation. These time-outs were caused because the remote JES3 system has performance problems, and occasionally, does not respond in the required 3 seconds.

Time-Out Error Recovery

```
Tracing device1
27    TR    S-ACK          *    Handshaking
28    TR    R-ACK          *         •
29    TR    S-ACK          *         •
30    TR    TIME-OUT       *    No response
31    TR    S-NAK          *    Not acknowledged
32    TR    TIME-OUT       *    No response
33    TR    S-NAK          *    Not acknowledged
34    TR    R-ACK          *    Response
35    TR    S-ACK          *    Handshaking
36    TR    R-ACK          *         •
            •              *         •
            •              *         •
            •              *         •
54    TR    R-ACK          *         •
55    TR    S-ACK          *    Handshaking
56    TR    TIME-OUT       *    No response
57    TR    S-NAK          *    Not acknowledged
58    TR    R-ACK          *    Response
59    TR    S-ACK          *    Handshaking
            •
            •
```

The response to these time-outs are NAKs (not acknowledged). RJE will respond this way up to six times before giving up and attempting a reboot. At this time **rjestat** would report that there are "Line Errors". NAK is a request to retransmit the previous response.

Communication Line Errors

This example shows trace output from an RJE subsystem that uses a dial-up connection. The phone line is noisy and is prone to dropping.

Tracing vpml

63	TR	S-ACK	*	Handshaking
64	TR	R-ACK	*	•
65	TR	S-ACK	*	•
66	TR	R-JUNK	*	Noise on the line
67	TR	S-NAK	*	Not acknowledged
68	TR	R-ACK	*	Recovery
69	TR	S-ACK	*	
70	TR	R-ACK	*	
71	TR	S-ACK	*	
72	TR	TIME-OUT	*	Line has dropped
73	TR	S-NAK	*	Attempting to recover
74	TR	TIME-OUT	*	•
75	TR	S-NAK	*	•
76	TR	TIME-OUT	*	•
77	TR	S-NAK	*	•
78	TR	TIME-OUT	*	•
79	TR	S-NAK	*	•
80	TR	TIME-OUT	*	•
81	TR	S-NAK	*	•
82	TR	TIME-OUT	*	•
83	RD	1 bytes	*	1 byte read (error)
84	ST	Stopack(0)	*	Script termination normal
85	CL	Clean	*	Cleanup
86	ST	Stopped	*	VPM script stopped
87	CL	Closed	*	VPM device closed

The error read in the above sequence causes RJE to reboot and **rjestat** to report line errors. If this type of thing were to occur frequently, a different method of communication should be used.

Error Responses

As seen in the parts above, the response to most errors is to send a NAK. The only exception is when starting up. Whenever a NAK is received on either side, it indicates that the previous transmission was not properly received. This should be followed by retransmission of the previous data. Generally, NAKs should not occur frequently and should be followed by recovery. If errors occur frequently or NAKs do not cause recovery, the line should be checked for problems.

On some IBM systems (e.g., JES2), an I/O error is printed at the system console whenever a NAK is received. These I/O errors can also be helpful in detecting the problem; however, they will not be discussed here as they vary with the system. It is assumed that someone in IBM support can assist if needed.

TABLE 10.A

TYPE	INFORMATION	MEANING
CL	Closed	The VPM device has been closed.
CL	Clean	The VPM driver is cleaning up for this device.
OP	Opened	The VPM has been successfully opened.
OP	Failed (start)	The open failed because the script within the physical device could not be started.
RD	*num* bytes	*Num* bytes were read from the VPM device by *rjerecv.*
SC	Exit (*num*)	The VPM script has terminated abnormally. The VPM termination code is *num.* Termination codes are defined in *vpm*(7).
ST	Startack	The physical device acknowledges that it has been started.
ST	Stopchk	Safety wakeup check on stop procedure.
ST	Stopack (*num*)	The physical device acknowledges that it is being halted by the vpm device. The VPM termination code is *num.* Termination codes are defined in *vpm*(7).
ST	Stopped	The VPM script has been stopped.
TR	Started	The script has started tracing.
TR	R-ACK	A 2-byte acknowledgment (ACK) string has been received from the remote system. This indicates that the previous transmission was properly received.
TR	S-ACK	A 2-byte acknowledgment (ACK) string has been transmitted to the remote system.
TR	R-NAK	A "not-acknowledged" (NAK) character has been received from the remote system. This indicates that the previous transmission was not properly received.
TR	S-NAK	A "not-acknowledged" (NAK) character has been transmitted to the remote system.
TR	R-ENQ	An enquiry (ENQ) character has been received from the remote system.
TR	S-ENQ	An enquiry (ENQ) character has been transmitted to the remote system.
TR	R-WAIT	The remote machine has requested that no data be transmitted to it.
TR	R-OKBLK	A valid data block was received from the remote machine.
TR	R-ERRBLK	An invalid Cyclic Redundancy Check (CRC) was received with a data block.
TR	R-SEQERR	The block sequence count on a received data block was invalid.
TR	R-JUNK	An invalid data block was received from the remote system.
TR	TIME-OUT	The remote machine did not respond within 3 seconds.
TR	S-BLK	A data block has been transmitted to the remote system.
WR	*num* bytes	*Num* bytes were written to the VPM device by *rjexmit.*