NAME

hasp − PWB/UNIX IBM Remote Job Entry

SYNOPSIS

/usr/hasp/haspinit

/usr/hasp/hasphalt

DESCRIPTION

Hasp is the communal name for a collection of programs and a file organization that allow PWB/UNIX, equipped with an appropriate driver for the DQS11-B, to communicate with IBM's Job Entry Subsystems by mimicking an IBM 2770 remote station.

Hasp is initiated by the command *haspinit* and is terminated gracefully by the command *hasphalt.* While active, *hasp* runs in background and requires no human supervision. It quietly transmits to the IBM system jobs that have been queued by the command *send*(I) and messages that have been entered by the command *rjestat*(I). It receives from the IBM system print and punch data sets and message output. It enters the data sets into the proper PWB/UNIX directory and notifies the appropriate user of their arrival. It scans the message output to maintain a record on each of its jobs. It also makes these messages available for public inspection, so that *rjestat*(I), in particular, may extract responses.

Unless otherwise specified, all files and commands described below live in directory */usr/hasp* (first exceptions: *send* and *rjestat*).

There are two sources of data that is to be transmitted by *hasp* from PWB/UNIX to an IBM System/370. In both cases, the data is organized as files in *ebcdic*(V) format. The first is a single file *haspmesg* that is reserved for message input. It is written by the enquiry command *rjestat*(I) and is assigned a priority for transmission. The second source, containing the bulk of the data, consists of jobs that have been entered into the *xmit∗* queue by the program *haspqer.* On completion of processing, *send* invokes *haspqer.* As each file is queued, a subordinate *info/logx∗* file is created to save the name, numeric id, login directory, and tty letter of the user who is doing the queueing. Upon successful transmission of the data to the IBM system, *haspdisp* will move this information into the *jobsout* file and delete the *info/logx∗* file.

Each time *haspinit* is invoked, the *xmit∗* queue is compacted, along with the associated *info/logx∗* files, and its beginning and end are calculated. A three-digit sequence number specifying the first free slot at the end of the queue is written to file *haspstat.* This number is subsequently updated by *haspqer* each time that a new job is entered into the queue. A pointer to the beginning of the queue is maintained by *haspmain.* It is periodically compared to the current end of the queue to determine whether any jobs are waiting to be transmitted. A null lock-file *hasplock* is created with mode zero to prevent simultaneous updating of *haspstat.*

In anticipation of receiving output, *hasp* always maintains a vacant file *tmp∗* in its own directory. Output from the IBM system is initially written into this file and is classified as either a print data set, a punch data set, or message output. Print output is converted to an ASCII text file, with standard tabs. Form feeds are suppressed, but the last line of each page is distinguished by the presence of an extraneous trailing space. Punch output is converted to EBCDIC format. This classification and both conversions occur as the output is received; *tmp∗* files are moved or copied into the appropriate user's directory and assigned the name *prnt∗* or *pnch∗*, respectively, or placed into user directories under user-specified names, or used as input to programs to be automatically executed, as specified by the user. This process is driven by the ‘‘usr=...’’ specification. *Hasp* retains ownership of these files and permits read-only access to them. Files of message output are digested by *hasp* immediately and are not retained.

A record is maintained for each job that passes through *hasp.* Identifying information is extracted contextually from files transmitted to and received from the IBM system. From each file transmitted, *hasp* extracts the job name, the programmer's name, the user name, the destination directory name, and the message level. This information is temporarily stored, in the order of submission of jobs, in file*jobsout.* It is retrie ved, by job name and programmer's name, when the IBM system acknowledges the job and assigns a number to it.

The IBM system automatically returns an acknowledgement message for each job it receives. Other status messages are returned in response to enquiries entered by users and in response to enquiries that *hasp* itself generates every ten minutes. All messages received by *hasp* are appended to the *resp* file. The *resp* file is automatically truncated when it reaches 32,000 bytes. Each sequence of enquiries written to the message file *haspmesg* should be preceded by an identification card image of the form ‘‘/*$UX<process id>’’. The IBM system will echo back the first portion of this card image, as this is an illegal command. The appearance of process ids in the response stream permits responses to be passed on to the proper users. *Hasp* enters process id zero on all enquiries it generates on its own behalf.

While it is active, *hasp* occupies at least the two process slots that are appropriated by *haspinit.* These slots are used to run *haspmain,* that supervises data transfers, as well as *haspdisp,* that performs dispatching functions; these two processes are connected by a pipe. The function of *haspmain* is to cycle repetitively, looking for data to transfer either to or from the IBM system. When it finds some, it spawns a child process, either *haspxmit* or *hasprecv,* to effect the transfer. It waits for its child to complete its task and then passes an event notice to *haspdisp. Haspmain* exits normally as soon as it detects the file *haspstop* (created by *hasphalt*), and exits reluctantly whenever it encounters a run of errors. An attempt is made to manage the null file *haspdead* so that it exists precisely when *haspmain* is not executing. *Haspinit* has the capability of *dialing* any remote IBM system with the proper hardware and software configuration. A file *haspsoff* is created by *hasphalt* to signal that the phone should be hung up by *haspmain.*

Ordinarily, *haspdisp* waits for event completion notices from *haspmain. Haspdisp* follows up the events described by directing output files, updating records, and notifying users. It may spawn the program *haspcopy* to copy output across file systems. *Haspdisp* references the system files */etc/passwd* and */etc/utmp* to correlate user names, numeric ids, and terminals. Normal termination of *haspmain* causes *haspdisp* to exit also. In the case of error termination, *haspdisp* delays about one minute and then reboots RJE by executing *haspinit* again.

Event notices begin with a one-digit code. The code ‘‘0’’ alone signals normal termination. Other event notices consist of a code in the range ‘‘1’’ to ‘‘6,’’ followed by the name of a file in the */usr/hasp* directory. Notices are issued as each file in the *xmit∗* queue is transmitted and as each *tmp∗* file is filled with output. These files are moved to new temporary names before the event notice is composed. Transmitted files (code 1) are renamed *zmit∗* and output files (codes 3-5) are renamed *prt∗, pch∗,* or *msg∗,* depending on their type. When *haspdisp* gets around to following up on the events described, the files will either be deleted or moved to a permanent destination.

Event notices are written to the *log* file at the time they are received by *haspdisp.* A typical section of the log looks as follows:

```
1zmit283
5msg61
1zmit284
5msg62
3prt63
```

Additional lines are written to the log by *haspinit.* Each reboot of *haspinit* is marked by a time stamp. If the previous execution of *haspmain* ended in error, an exception notice precedes the time stamp. Exception notices are formatted by *haspmain* and consist of a sequence of capital letters. The most common is ‘‘AAAAA’’, that indicates five successive failures to acquire the line for a transmission to the host. A sequence of time stamps alternating with ‘‘AAAAA’’ indicates that the host is not responding to RJE. Each time the RJE facility is booted via the *haspinit* program, the *log* file is cleaned out. A copy of its last contents is placed in a file named *slog.*

Several RJE programs, including the *send*(I) command, use the *ustat*(II) system call to determine the remaining capacity of the file systems they use, in an attempt to keep roughly 1,000 blocks and 50 i-nodes free. *Haspinit* issues a warning when fewer than 2,000 blocks or 100 i-nodes are available. *Send* shuts down when only 1,500 blocks remain, and *haspmain* stops accepting output from the IBM system when the capacity falls to 1,200 blocks. In addition, output files are limited in size at all times to a maximum of 512K bytes. Of this, only 64K bytes are guaranteed. How much more output will be accepted depends on the current capacity of the *hasp* file system. Excess data is simply discarded, with no provision for retrieving it.

Most *hasp* files and directories are protected from unauthorized tampering. The exception is the *pool* directory, that is provided so that *send*(I) can create temporary files in the correct file system. *Haspqer* and *rjestat*(I), the user's interfaces to *hasp,* operate in *setuid* mode to contribute the necessary permission modes. *Rjestat*(I), incidentally, extends to anyone who can login as *rje* complete freedom to enter console commands. When invoked with a **+** argument, it suppresses the **d** that begins a display command and allows one to cancel or re-route jobs.

Some minimal oversight of each *hasp* subsystem is required. The *hasp* mailbox should be inspected and cleaned out periodically. The*job* directory should also be check ed. The only files placed there are output files whose destination file systems are out of space. Users should be given a short period of time (say, a day or two), and then these files should be removed. In the source code for the program,*haspdisp* is a *define* statement for the character string **SFILE**. If it is set to 1, all returned jobs for which PWB/UNIX RJE can not find the ''usr=**...**'' specification are placed into the *job* directory. This feature is primarily intended for those PWB/UNIX systems that are connected to the IBM systems whose output format is not known to *haspdisp.*

Usage statistics are recorded in the directory */usr/hasp/usg,* if it exists. Six files will be created and updated. Each will contain data on a per-user-id basis. File *hasp.in.sum* accumulates the number of blocks transmitted by *hasp;* file *hasp.in.cnt* records the number of transmissions; file *hasp.in.max* records the size, in blocks, of the largest job sent. Files *hasp.out.sum, hasp.out.cnt* and *hasp.out.max* contain the same statistics for output received by *hasp.* The program *usage* may be used to print these statistics; ''usage file [user-id1 ...]'' will print out the statistics gathered in *file*. If the optional user-id list is present, only the statistics for these user-ids will be printed.

The configuration table */usr/rje/lines* is accessed by all components of RJE. Its six columns may be labeled ''host'', ''system'', ''directory'', ''prefix'', ''device'', and ''types''. Each line of the table (maximum of six) defines an RJE connection. ''Host'' is the name of a remote computer: **A, B,** or **1110**. ''System'' is a string of capital letters identifying PWB/UNIX systems. The first specifies where the RJE connection is normally terminated; the remainder specify where it may be backed-up to if the primary RJE system goes down. ''Directory'' is the directory name of the servicing RJE subsystem. ''Prefix'' is the string prepended (redundantly) to several crucial files and programs in the directory: *hasp, hasp2, uvac*. ''Device'' is the name of the controlling DQS-11B, with ''*/dev/*'' excised. ''Types'' contains information on the type of connection to make. It contains the logon id and phone number to use when RJE is to automatically*dial* a remote IBM system. An**n** in this field indicates that this entry is not for dial-up. When a dial-up entry is initiated, the phone number found here is automatically dialed, PWB/UNIX RJE logs on, and normal RJE processing occurs. When*hasphalt* is e xecuted, RJE signs off and hangs up the phone automatically. If the first character in the ''types'' field is an **i**, all console status facilities are inhibited (e.g., *rjestat*(I) will not behave like a status terminal, and the ten-minutes automatic status inquiry is inhibited).

The file */usr/rje/sys* contains the single-letter name of the current PWB/UNIX system. An RJE connection will be considered available if this is its primary system or if this is one of its backup systems and the associated directory is mounted. *Send*(I) and *rjestat*(I) select an a vailable connection by indexing on the ''host'' field of the configuration table. *Hasp* programs inde x on the ''prefix'' field. A subordinate directory, *sque,* exists in */usr/rje* for use by *haspdisp* and *shqer* programs. This directory holds those output files that have been designated as standard input to some executable file. This designation is done via the ''usr=...'' specification. *Haspdisp* places the output files here and updates the file *log* to specify the order of execution, arguments to be passed, etc. *Shqer* e xecutes the appropriate files. A program called *compact* compacts the *log* file. It should be executed before *shqer* and RJE have been started.

All HASP programs are reentrant; therefore, if more than one HASP is to be run on a given PWB/UNIX system, simply link, via *ln*(I), HASP2 program names to HASP names in */usr.*

FILES

configuration-dependent and general-purpose RJE files:

/dev/rjei	DQS11-B
/dev/tty?	terminals
/etc/utmp	list of active users
/etc/passwd	user population
/usr/rje/sys	PWB/UNIX system name, e.g., ‘‘A’’
/usr/rje/lines	PWB/UNIX RJE lines configuration table
/usr/rje/sque/log	log information for *shqer*

user files

∗/.mail	a user’s mailbox
∗/prnt∗	a user’s print data set
∗/pnch∗	a user’s punch data set

hasp files (relative to the *directory* entry in the RJE configuration table):

hasp∗	mostly programs
haspdead	inactive flag
haspsoff	dial-up hang-up signal
haspstop	halt signal
haspmesg	message slot
haspstat	queue end record
hasplock	lockout file
xmit∗	jobs queued
info/logx∗	haspqer loginfo
job/∗	output from jobs whose file systems are out of space
jobsout	fifo job store
tmp∗	output files
log	event log
resp	concatenated responses from the IBM system
status	RJE message of the day
pool/stm∗	*send*(I) temporaries
usg/∗	usage statistics

SEE ALSO

send(I), rjestat(I), rje(IV), ebcdic(V), regen(VIII)
Guide to IBM Remote Job Entry for PWB/UNIX Users by A. L. Sabsevitz.
System Components: IBM 2770 Data Communication System, IBM SRL GA27-3013.
OS/VS2 HASP II Version 4 System Programmer’s Guide IBM SRL GC27-6992.

DIAGNOSTICS

Haspinit provides brief error messages describing obstacles to bringing up *hasp.* They can best be understood in the context of the RJE source code. The most frequently occurring one is ‘‘cannot open /dev/rjei’’. This may occur if the DQS-11B status register shows something other than READY (octal 200). It will also occur if another process already has the DQS-11B open, or if the exclusive use flag (_dqsx+3, _dqsx+73, etc.) has remained set after a close of the DQS-11B.

Once *hasp* has been started, users should assist in monitoring its performance, and should notify operations personnel of any perceived need for remedial action. *Rjestat*(I) will aid in diagnosing the current state of RJE. It can detect, with some reliability, when the far end of the communications line has gone dead, and will report in this case that the host computer is not responding to RJE. It will also attempt to reboot *hasp* if it detects a prolonged period of inactivity on the DQS-11B.

BUGS

The name *hasp* is an anachronism. It is used only as a collective name and could represent *hasp, jes2, asp,* etc.