NAME

aimk - Portable make wrapper script

SYNOPSIS

aimk [-here] [make arguments]

DESCRIPTION

aimk is a wrapper program for make, used to portably select options to build PVM and PVM applications on various machines. Each port of PVM is assigned an architecture name. The name is used both during compilation (to conditionally switch in code) and at runtime (to select an executable or host).

aimk uses the value of environment variable $PVM_ARCH if it is set, otherwise it calls $PVM_ROOT/pvmgetarch to determine the architecture name. pvmgetarch is a script that sniffs at various parts of the system to determine the correct architecture name. It is updated as new PVM ports are defined, and can be augmented locally.

aimk determines the machine architecture and execs make, passing it the architecture and a configuration file along with arguments supplied to aimk. It runs make in a subdirectory to prevent executables from becoming intermixed and to permit overlapping compiles. A different makefile can be placed in each subdirectory or a single makefile, Makefile.aimk, can be shared between architectures. Per-architecture definitions from the $PVM_ROOT/conf directory are appended to the common makefile. aimk calls make is called in one of three ways, depending on what makefiles are present:

i. If $PVM_ARCH/Makefile or $PVM_ARCH/makefile exists, change directory to $PVM_ARCH and exec make there:

```
(cd $PVM_ARCH ; make PVM_ARCH=$PVM_ARCH < aimk args >)
```

ii. Else if Makefile.aimk exists, create $PVM_ARCH directory if it doesn't exist, then:

```
(cd $PVM_ARCH ; \
make -f $PVM_ROOT/conf/$PVM_ARCH.def \
-f ../Makefile.aimk PVM_ARCH=$PVM_ARCH < aimk args >)
```

iii. Else just exec make in current directory:

```
make PVM_ARCH=$PVM_ARCH < aimk args >
```

If aimk succeeds in calling make, the exit status is that of make, otherwise it is 1.

FLAGS

-here Forces aimk to run make in the current directory, e.g. converts case i. to case iii.

EXAMPLES

The following Makefile.aimk file builds and installs hello, creating the PVM binary directory if it doesn't exist. It can be run concurrently on machines of different types, sharing the same source directory.

```
LDIR    =  -L$(PVM_ROOT)/lib/$(PVM_ARCH)
PVMLIB  =  -lpvm3
SDIR    =  ..
BDIR    =  $(HOME)/pvm3/bin
XDIR    =  $(BDIR)/$(PVM_ARCH)
CFLAGS  =  -g -I$(PVM_ROOT)/include
LIBS    =  $(LDIR) $(PVMLIB) $(ARCHLIB)

$(XDIR):
        - mkdir $(BDIR) $(XDIR)

hello: $(SDIR)/hello.c $(XDIR)
        $(CC) $(CFLAGS) -o $@ $(SDIR)/$@.c $(LIBS)
        mv $@ $(XDIR)
```

ENVIRONMENT

$PVM_ROOT	Root path of PVM installation.
$PVM_ARCH	PVM architecture name for machine.

FILES

$PVM_ROOT/lib/aimk	The aimk program
$PVM_ROOT/conf/$PVM_ARCH.def	Arch config file

SEE ALSO

pvm_intro(1PVM)

NAME

pvm - PVM version 3 console

SYNOPSIS

pvm [-options] [hostfile] &

DESCRIPTION

Pvm is a stand alone pvm task which allows the user to interactively query and modify the virtual machine. The console can be started and stopped multiple times on any of the hosts in the virtual machine without affecting PVM or any applications that may be running.

When started pvm determines if PVM is already running and if not automatically executes pvmd3 on this host, passing pvmd3 the command line options and hostfile. Thus PVM need not be running to start the console. Once started the console prints the prompt:

```
pvm>
```

The following console commands are available:

```
add  hostname(s)     - add hosts to virtual machine
alias                - define/list command aliases
conf                 - list virtual machine configuration
delete  hostname(s)  - delete hosts from virtual machine
echo                 - echo arguments
halt                 - stop pvmds
help [command]       - print helpful information about a command
id                   - print console task id
jobs                 - list running jobs
kill  task-tid       - terminate tasks
mstat host-tid       - show status of hosts
ps -a                - list all pvm tasks
pstat task-tid       - show status of tasks
quit                 - exit console
reset                - kill all tasks
setenv               - display/set environment variables
sig  signum task     - send signal to task
spawn [opt] a.out    - spawn task
     opts are: -(count)  number of tasks, default is 1
         -(host)   spawn on host, default is any
         -(ARCH)   spawn on hosts of ARCH
         -?        enable debugging
               ->        redirect task output to console
               -> file   redirect task output to file
               ->>file   redirect task output append to file
trace                - set/display trace event mask
unalias - undefine command alias
version - show libpvm version
```

pvm reads $HOME/.pvmrc before reading commands from the tty, so it can be used to customize the console environment, for example:

```
alias ? help
alias j jobs
setenv PVM_EXPORT DISPLAY
# print my id
echo new pvm shell
id
```

EXAMPLES

pvm

Starts up pvmd3 on the local host or connects to running pvmd3.

pvm hostfile

Starts up console and pvmd3, which inturn reads the hostfile and adds the listed computers to the virtual machine.

SEE ALSO

pvmd3(1PVM)

NAME

PVM - Parallel Virtual Machine System Version 3

DESCRIPTION

PVM is a software system that enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational resource.

The individual computers may be shared- or local-memory multiprocessors, vector supercomputers, specialized graphics engines, or scalar workstations, that may be interconnected by a variety of networks, such as ethernet, FDDI.

User programs written in C, C++ or Fortran access PVM through library routines (libpvm3.a and libfpvm3.a).

Daemon programs (pvmd3) provide communication and process control between computers.

MACHINE ARCHITECTURE

In the PVM system, machines are assigned a short string to identify their architectures (this includes operating system type as well as CPU type). The types currently predefined in the distribution are:

AFX8	Alliant FX/8
ALPHA	DEC Alpha/OSF-1
BAL	Sequent Balance
BFLY	BBN Butterfly TC2000
BSD386	80[34]86 running BSDI or BSD386
CM2	Thinking Machines CM-2 Sun front
CM5	Thinking Machines CM-5
CNVX	Convex using IEEE floating-point
CNVXN	Convex using native f.p.
CRAY	Cray
CRAY2	Cray-2
CRAYSMP	Cray S-MP
DGAV	Data General Aviion
E88K	Encore 88000
HP300	HP 9000 68000 cpu
HPPA	HP 9000 PA-Risc
KSR1	Kendall Square
I860	Intel RX Hypercube
IPSC2	Intel IPSC/2
LINUX	80[34]86 running Linux
MIPS	Mips
NEXT	NeXT
PGON	Intel Paragon
PMAX	DEC/Mips arch (3100, 5000, etc.)
RS6K	IBM/RS6000
RT	IBM/RT
SGI	Silicon Graphics IRIS
SGI5	Silicon Graphics IRIS running OS >= 5.0

SUN3	Sun 3
SUN4	Sun 4, 4c, sparc, etc.
SUN4SOL2	Sun 4 running Solaris
SYMM	Sequent Symmetry
TITN	Stardent Titan
U370	IBM 3090 running AIX
UVAX	DEC/Microvax
VCM2	Thinking Machines CM-2 Vax front

SEE ALSO

pvm(1PVM), pvmd3(1PVM), PVM 3.3 User's Guide and Reference Manual

AUTHORS

A. L. Beguelin [4,5], J. J. Dongarra [1,2], G. A. Geist [2], W. C. Jiang [1], R. J. Manchek [1], B. K. Moore [1], V. S. Sunderam [3]

1. University of Tennessee, Knoxville TN.
2. Oak Ridge National Laboratory, Oak Ridge TN.
3. Emory University, Atlanta GA.
4. Carnegie Mellon University, Pittsburgh PA
5. Pittsburgh Supercomputer Center, Pittsburgh PA

NAME

pvmd3 - PVM version 3 daemon

SYNOPSIS

pvmd [-options] [hostfile]

DESCRIPTION

Pvmd3 is a daemon process which coordinates unix hosts in a virtual machine. One pvmd3 must run on each host in the group. They provide the communication and process control functions needed by the user's PVM processes. The daemon can be started manually with a host file argument that will automatically start the remote pvmds. The local and remote pvmds can also be started from the pvm console program pvm.

The name of the daemon executable is pvmd3. It is usually started by a shell script, $PVM_ROOT/lib/pvmd.

OPTIONS

The following options may be specified on the command line when starting the master pvmd or pvm console:

-dmask Set pvmd debug mask. Used to debug the pvmd or libpvm (not intended to be used to debug application programs). Mask is a hexadecimal number which is the sum of the following bits:

Bit	Information
1	Packet routing
2	Message routing and entry points
4	Task management
8	Slave pvmd startup
10	Host table updates
20	Select loop (below packet layer)
40	IP network
80	Multiprocessor port debugging
100	Resource manager interface
200	Application (messages with no destination, etc.)

-nname Specify an alternate hostname for the master pvmd to use. Useful when gethostname() returns a name not assigned to any network interface.

The following options are used by the master pvmd when starting slaves and are only of interest to someone writing a hoster. Don't just go using them, now.

-s Start pvmd in slave mode. Hostfile cannot be used, five additional parameters must be supplied: master pvmd index, master IP, master MTU, slave pvmd index, and slave IP.

−S Same as −s, but slave pvmd doesn’t wait for its stdin to be closed after printing its parameters. Used for manual startup.

−f Slave doesn’t fork after configuration (useful if the slave is to be controlled or monitored by some process).

HOST FILE FORMAT

Each host in the virtual machine must have an entry in the host file. Lines beginning with a splat (#), optionally preceded by whitespace, are ignored.

A simple host file might look like:

```
# my first host file
thud
fred
wilma
barney
betty
```

This specifies the names of five hosts to be configured in the virtual machine.

The master pvmd for a group is started by hand on the localhost, and it starts slaves on each of the remaining hosts using the rsh or rexec command. The master host may appear on any line of the hostfile, but must have an entry.

The simple format above works fine if you have the same login name on all five machines and the name of the master host in your .rhosts files on the other four.

There are several host file options available:

lo=NAME Specifies an alternate login name (NAME) to use.

so=pw This is necessary when the remote host cannot trust the master. Causes the master pvmd to prompt for a password for the remote host in the tty of the pvmd (note you can’t start the master using the console or background it when using this option) you will see:

    Password (honk.cs.utk.edu:manchek):

you should type your password for the remote host. The startup will then continue as normal.

dx=FILE Specifies the path of the pvmd executable. FILE may be a simple filename, an absolute pathname, or a path relative to the user's home directory on the remote host. This is mainly useful to aid in debugging new versions of PVM, but may have other uses.

ep=PATH Specifies a path for the pvmd to search for executable program components when spawning a new process. The path may have multiple elements, separated by colons (:).

wd=PATH Specifies a working directory in which all spawned tasks on this host will execute.

sp=VALUE Specifies the relative computational speed of this host compared to other hosts in the configuration. VALUE is an integer in the range [1 - 1000000]

bx=PATH Specifies the debugger program path. Note: the environment variable PVM_DEBUGGER can also be set.

so=ms Rarely used. Causes the master pvmd to request user to manually perform the startup of a pvmd on a slave host when rsh and rexec network services are disabled but IP connectivity exists. See section "MANUAL STARTUP".

A dollar sign ($) in an option introduces a variable name, for example $PVM_ARCH. Names are expanded from environment variables by each pvmd.

Each of the flags above has a default value. These are:

```
lo      The loginname on the master host.
so      Nothing
dx      $PVM_ROOT/lib/pvmd
ep      pvm3/bin/$PVM_ARCH:$PVM_ROOT/bin/$PVM_ARCH
wd      $HOME
sp      1000
bx      $PVM_ROOT/lib/debugger
```

You can change these by adding a line with a star (*) in the first field followed by the options, for example:

```
* lo=afriend so=pw
```

This sets new default values for 'lo' and 'so' for the remainder of the host file, or until the next '*' line. Options set on the last '*' line also apply to hosts added dynamically using pvm_addhosts().

Host options can be set without starting the hosts automatically. Information on host file lines beginning with '&' is stored, but the hosts are not started until added using pvm_addhosts().

Example hostfile:

```
# hostfile for testing on various platforms
fonebone
refuge
# installed in /usr/local/here
sigi.cs            dx=/usr/local/pvm3/lib/pvmd
# borrowed accts, "guest", don't trust fonebone
*                  lo=guest  so=pw
sn666.jrandom.com
cubie.misc.edu
# really painful one, must start it by hand and share a homedir
& igor.firewall.com  lo=guest2  so=ms  ep=bob/pvm3/bin/$PVM_ARCH
```

MANUAL STARTUP

When adding a host with this option set you will see on the tty of the pvmd:

```
*** Manual startup ***
Login to "honk" and type:
$PVM_ROOT/lib/pvmd -S -d0 -nhonk 1 80a9ca95:0cb6 4096 2 80a95c43:0000
Type response:
```

after typing the given command on host honk, you should see a line like:

```
ddpro<2312> arch<ALPHA> ip<80a95c43:0a8e> mtu<4096>
```

type this line on the tty of the master pvmd. You should then see:

```
Thanks
```

and the two pvmds should be able to communicate.

Note you can't start the master using the console or background it when using this option.

STOPPING PVMD3

The preferred method of stopping all the pvmds is to give the halt command in the pvm console. This kills all pvm tasks, all the remote daemons, the local daemon, and finally the console itself. If the master pvmd is killed manually it should be sent a SIGTERM signal to allow it to kill the remote pvmds and clean up various files.

The pvmd can be killed in a manner that leaves the file /tmp/pvmd.uid behind on one or more hosts. Uid is the numeric user ID (from /etc/passwd) of the user. This will prevent PVM from restarting on that host. Deletion of this file will fix this problem:

```
rm `( grep $user /etc/passwd || ypmatch $user passwd )
     | awk -F: '{print "/tmp/pvmd."$3; exit}'`
```

SEE ALSO

pvm(1PVM)

NAME

pvm_addhosts - Add hosts to the virtual machine.

SYNOPSIS

```
C    int info = pvm_addhosts( char **hosts, int  nhost,  int
*infos )

Fortran   call pvmfaddhost( host, info )
```

PARAMETERS

hosts   An array of strings naming the hosts to be added. Pvmd must already be installed and the user must have an account on the specified hosts.

nhost   Integer specifying the length of array hosts.

infos   Integer array of length nhost which returns the status for each host. Values less than zero indicate an error, while positive values are TIDs of the new hosts.

host   Character string naming the host to be added.

info   Integer status code returned by the routine. Values less than nhost indicate partial failure, and values less than 1 indicate total failure.

DISCUSSION

The routine pvm_addhosts adds the computers named in hosts to the configuration of computers making up the virtual machine. The names should have the same syntax as lines of a pvmd hostfile (see man page for pvmd3): A hostname followed by options of the form xx=y.

If pvm_addhosts is successful, info will be equal to nhost. Partial success is indicated by 0 < info < nhost, and total failure by info < 1. The array infos can be checked to determine which host caused the error.

The Fortran routine pvmfaddhost adds a single host to the configuration with each call. info will be 1 if successful or < 0 if error.

The status of hosts can be requested by the application using pvm_mstat and pvm_config. If a host fails it will be automatically deleted from the configuration. Using pvm_addhosts a replacement host can be added by the application, however it is the responsibility of the application developer to make his application tolerant of host failure. Another use of this feature would be to add more hosts as they become available, for example on a weekend, or if the application dynamically determines it could use more computational power.

EXAMPLES

C:

```
static char *hosts[] = {
    "sparky",
    "thud.cs.utk.edu",
};
info = pvm_addhosts( hosts, 2, infos );
```

Fortran:

```
CALL PVMFADDHOST( 'azure', INFO )
```

ERRORS

The following error conditions can be returned by pvm_addhosts:

PvmBadParam	giving an invalid argument value.
PvmAlready	another task is currently adding hosts.
PvmSysErr	local pvmd is not responding.

and in the infos vector:

PvmBadParam	bad hostname syntax.
PvmNoHost	no such host.
PvmCantStart	failed to start pvmd on host.
PvmDupHost	host already configured.
PvmBadVersion	pvmd protocol versions don't match.
PvmOutOfRes	PVM has run out of system resources.

SEE ALSO

pvmd3(1PVM), delhost(3PVM), start_pvmd(3PVM), config(3PVM), mstat(3PVM)

NAME

pvm_advise() - (PVM 3.1) Controls use of direct task-to-task routing. Relaced by pvm_setopt().

SYNOPSIS

```
C    int info = pvm_advise( int route )

Fortran   call pvmfadvise( route, info )
```

PARAMETERS

route   Integer advising PVM to set up direct task-to-task links.

route options

PvmDontRoute	1	Don't allow direct links to this task
PvmAllowDirect	2	Allow but don't request direct links
PvmRouteDirect	3	Request direct links

info   Integer returning error status.

DISCUSSION

The routine pvm_advise advises PVM on whether or not to set up direct task-to-task links (using TCP) for all subsequent communication. Once a link is established it remains until the application finishes. If a direct link can not be established because one of the two tasks has requested PvmDontRoute or because no resources are available, then the default route through the PVM daemons is used. pvm_advise can be called multiple times to selectively establish direct links, but is typically set only once near the beginning of each task. PvmAllowDirect is the default advise setting. This setting on task A allows other tasks to set up direct links to A. Once a direct link is established between tasks both tasks will use it for sending messages. pvm_advise returns the error status in info. The performance of direct task-to-task links can be up to a factor of two better than the default route. The draw back is a lack of scalability of the direct links. Some versions of UNIX limit the number of links to no more than 30.

EXAMPLES

C:

```
info = pvm_advise( PvmRouteDirect );
```

Fortran:

```
CALL PVMFADVISE( PVMROUTEDIRECT, INFO )
```

ERRORS

This error condition can be returned by pvm_advise

PvmBadParam giving an invalid route value.

SEE ALSO

pvm_setopt(3PVM)

NAME

pvm_archcode - Returns the data representation code for a PVM architecture name.

SYNOPSIS

```
C       int cod = pvm_archcode( char *arch )

Fortran   call pvmfarchcode( arch, cod )
```

PARAMETERS

arch   Character string containing the architecture name.

cod   Integer returning architecture code.

DISCUSSION

The routine pvm_archcode returns an integer given an architecture name. The code returned identifies machines with compatible binary data formats. For example, SUN4 and RS6K have the same code, while ALPHA has a different one (because a few datatypes have different sizes). This lets you know when you can get away with using PvmDataRaw instead of PvmDataDefault encoding to pass messages between tasks on two machines.

Naturally, you shouldn't assume the values returned by pvm_archcode are etched in stone; the numbers have no intrinsic meaning except that if two different arch names map to the same value then they're compatible.

EXAMPLES

C:

```
struct pvmhostinfo *hip;
int i;

pvm_config((int *)0, (int *)0, &hip);
i = pvm_archcode(hip[0].hi_arch);
```

Fortran:

```
CALL PVMFARCHCODE( 'RS6K', k )
```

ERRORS

No error conditions are be returned by pvm_archcode.

SEE ALSO

pvm_config(3PVM), pvm_initsend(3PVM), pvm_notify(3PVM), pvm_tasks(3PVM), pvm_tidtohost(3PVM)

NAME

pvm_barrier - Blocks the calling process until all processes in a group have called it.

SYNOPSIS

C int info = pvm_barrier(char *group, int count)

Fortran call pvmfbarrier(group, count, info)

PARAMETERS

group Character string group name. The group must exist and the calling process must be a member of the group.

count Integer specifying the number of group members that must call pvm_barrier before they are all released. Though not required, count is expected to be the total number of members of the specified group.

info Integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION

The routine pvm_barrier blocks the calling process until count members of the group have called pvm_barrier. The count argument is required because processes could be joining the given group after other processes have called pvm_barrier. Thus PVM doesn't know how many group members to wait for at any given instant. Although count can be set less, it is typically the total number of members of the group. So the logical function of the pvm_barrier call is to provide a group synchronization. During any given barrier call all participating group members must call barrier with the same count value. Once a given barrier has been successfully passed, pvm_barrier can be called again by the same group using the same group name.

If pvm_barrier is successful, info will be 0. If some error occurs then info will be < 0.

EXAMPLES

C:

```
     inum = pvm_joingroup( "worker" );
             .
             .
     info = pvm_barrier( "worker", 5 );
```

Fortran:

```
          CALL PVMFJOINGROUP( "shakers", INUM )
          COUNT = 10
          CALL PVMFBARRIER( "shakers", COUNT, INFO )
```

ERRORS

These error conditions can be returned by pvm_barrier

PvmSysErr	pvmd was not started or has crashed.
PvmBadParam	giving a count < 1.
PvmNoGroup	giving a non-existent group name.
PvmNotInGroup	calling process is not in specified group.

SEE ALSO

joingroup(3PVM)

NAME

pvm_bcast - Broadcasts the data in the active message buffer to a group of processes.

SYNOPSIS

```
C    int info = pvm_bcast( char *group, int msgtag )

Fortran   call pvmfbcast( group, msgtag, info )
```

PARAMETERS

group
: Character string group name of an existing group.

msgtag
: Integer message tag supplied by the user. msgtag should be >= 0. It allows the user's program to distinguish between different kinds of messages . info integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION

The routine pvm_bcast broadcasts a message stored in the active send buffer to all the members of group. In PVM 3.2 and later the broadcast message is not sent back to the sender. Any PVM task can call pvm_bcast(), it need not be a member of the group. The content of the message can be distinguished by msgtag. If pvm_bcast is successful, info will be 0. If some error occurs then info will be < 0.

pvm_bcast is asynchronous. Computation on the sending processor resumes as soon as the message is safely on its way to the receiving processors. This is in contrast to synchronous communication, during which computation on the sending processor halts until a matching receive is executed by all the receiving processors.

pvm_bcast first determines the tids of the group members by checking a group data base. A multicast is performed to these tids. If the group is changed during a broadcast the change will not be reflected in the broadcast. Multicasting is not supported by most multiprocessor vendors. Typically their native calls only support broadcasting to all the user's processes on a multiprocessor. Because of this omission, pvm_bcast may not be an efficient communication method on some multiprocessors.

EXAMPLES

C:

```
info = pvm_initsend( PvmDataRaw );
info = pvm_pkint( array, 10, 1 );
```

```
          msgtag = 5 ;
          info = pvm_bcast( "worker", msgtag );
```

Fortran:

```
          CALL PVMFINITSEND( PVMDEFAULT )
          CALL PVMFPKFLOAT( DATA, 100, 1, INFO )
          CALL PVMFBCAST( 'worker', 5, INFO )
```

ERRORS

These error conditions can be returned by pvm_bcast

PvmSysErr	pvmd was not started or has crashed.
PvmBadParam	giving a negative msgtag.
PvmNoGroup	giving a non-existent group name.

SEE ALSO

joingroup(3PVM)

NAME

pvm_bufinfo() - Returns information about a message buffer.

SYNOPSIS

C int info = pvm_bufinfo(int bufid, int *bytes, int *msgtag, int *tid)

Fortran call pvmfbufinfo(bufid, bytes, msgtag, tid, info)

PARAMETERS

bufid
: Integer specifying a particular message buffer identifier.

bytes
: Integer returning the length in bytes of the entire message.

msgtag
: Integer returning the message label. Useful when the message was received with a wildcard msgtag.

tid
: Integer returning the source of the message. Useful when the message was received with a wildcard tid.

info
: Integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION

The routine pvm_bufinfo returns information about the requested message buffer. Typically it is used to determine facts about the last received message such as its size or source. pvm_bufinfo is especially useful when an application is able to receive any incoming message, and the action taken depends on the source tid and the msgtag associated with the message that comes in first. If pvm_bufinfo is successful, info will be 0. If some error occurs then info will be < 0.

EXAMPLES

C:

```
bufid = pvm_recv( -1, -1 );
info = pvm_bufinfo( bufid, &bytes, &type, &source );
```

Fortran:

```
CALL PVMFRECV( -1, -1, BUFID )
CALL PVMFBUFINFO( BUFID, BYTES, TYPE, SOURCE, INFO )
```

ERRORS

This error condition can be returned by pvm_bufinfo.

PvmNoSuchBuf	specified buffer does not exist.
PvmBadParam	invalid argument

SEE ALSO

recv(3PVM)

NAME

pvm_catchout - Catch output from child tasks.

SYNOPSIS

C

```
#include <stdio.h>

int bufid = pvm_catchout( FILE *ff )
```

Fortran

```
call pvmfcatchout( onoff )
```

PARAMETERS

ff File descriptor on which to write collected output.

onoff Integer parameter. Turns output collection on or off.

DISCUSSION

The routine pvm_catchout causes the calling task (the parent) to catch output from tasks spawned after the call to pvm_catchout. Characters printed on stdout or stderr in children tasks are collected by the pvmds and sent in control messages to the parent task, which tags each line and appends it to the specified file. Output from grandchildren (spawned by children) tasks is also collected, provided the children don't reset PvmOutputTid.

Each line of output has one of the following forms:

```
[txxxxx] BEGIN
[txxxxx] (text from child task)
[txxxxx] END
```

The output from each task includes one BEGIN line and one END line, with whatever the task prints in between.

In C, the output file descriptor may be specified. Giving a null pointer turns output collection off. In Fortran, output collection can only be turned on or off, and is logged to stdout of the parent task.

If pvm_exit is called while output collection is in effect, it will block until all tasks sending it output have exited, in order to print all their output. To avoid this, output collection can be turned off by calling pvm_catchout(0) before calling pvm_exit.

pvm_catchout() always returns 0.

EXAMPLES

C:

```
#include <stdio.h>

pvm_catchout(stdout);
```

Fortran:

```
CALL PVMFCATCHOUT( 1 )
```

SEE ALSO

pvm_exit(3PVM), pvm_setopt(3PVM), pvm_spawn(3PVM)

NAME

pvm_config - Returns information about the present virtual machine configuration.

SYNOPSIS

```
C    int info = pvm_config( int *nhost, int  *narch,  struct
pvmhostinfo **hostp )
     struct pvmhostinfo {
          int  hi_tid;
          char *hi_name;
          char *hi_arch;
          int  hi_speed;
     };

Fortran   call pvmfconfig( nhost, narch, dtid,  name,  arch,
speed, info )
```

PARAMETERS

nhost Integer returning the number of hosts (pvmds) in the virtual machine.

narch Integer returning the number of different data formats being used.

hostp Pointer to an array of structures which contain information about each host including its pvmd task ID, name, architecture, and relative speed.

dtid Integer returning pvmd task ID for this host

name Character string returning name of this host

arch Character string returning name of host architecture

speed Integer returning relative speed of this host. Default value is 1000.

info Integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION

The routine pvm_config returns information about the present virtual machine. The information returned is similar to that available from the console command conf.

The C function returns information about the entire virtual machine in one call. The Fortran function returns information about one host per call and cycles through all the hosts. Thus, if pvmfconfig is called nhost times, the entire virtual machine will be represented. If pvm_config is successful, info will be 0. If some error occurs then info will be < 0.

EXAMPLES

C:

```
info = pvm_config( &nhost, &narch, &hostp );
```

Fortran:

```
Do i=1, NHOST
  CALL PVMFCONFIG( NHOST,NARCH,DTID(i),HOST(i),ARCH(i),SPEED(i),INFO )
Enddo
```

ERRORS

The following error condition can be returned by pvm_config

PvmSysErr   pvmd not responding.

SEE ALSO

tasks(3PVM)

NAME

pvm_delhosts - Deletes hosts from the virtual machine.

SYNOPSIS

C   int info = pvm_delhosts(char **hosts, int nhost, int *infos)

Fortran   call pvmfdelhost(host, info)

PARAMETERS

hosts An array of pointers to character strings containing the names of the machines to be deleted.

nhost Integer specifying the number of hosts to be deleted.

infos Integer array of length nhost which contains the status code returned by the routine for the individual hosts. Values less than zero indicate an error.

host Character string containing the name of the machine to be deleted.

info Integer status code returned by the routine. Values less than nhost indicate partial failure, values less than 1 indicate total failure.

DISCUSSION

The routine pvm_delhosts deletes the computers pointed to in hosts from the existing configuration of computers making up the virtual machine. All PVM processes and the pvmd running on these computers are killed as the computer is deleted. If pvm_delhosts is successful, info will be nhost. Partial success is indicated by 1<= info < nhost, and total failure by info < 1. The array infos can be checked to determine which host caused the error.

The Fortran routine pvmfdelhost deletes a single host from the configuration with each call.

If a host fails, the PVM system will continue to function and will automatically delete this host from the virtual machine. An application can be notified of a host failure by calling pvm_notify. It is still the responsibility of the application developer to make his application tolerant of host failure.

EXAMPLES

C:

```
static char *hosts[] = {
     "sparky",
     "thud.cs.utk.edu",
};
info = pvm_delhosts( hosts, 2 );
```

Fortran:

```
CALL PVMFDELHOST( 'azure', INFO )
```

ERRORS

These error conditions can be returned by pvm_delhosts

PvmBadParam	giving an invalid argument value.
PvmSysErr	local pvmd not responding.
PvmOutOfRes	PVM has run out of system resources.

SEE ALSO

addhost(3PVM), notify(3PVM)

NAME

pvm_exit - Tells the local pvmd that this process is leaving PVM.

SYNOPSIS

C   int info = pvm_exit(void)

Fortran   call pvmfexit(info)

PARAMETERS

info   Integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION

The routine pvm_exit tells the local pvmd that this process is leaving PVM. This routine does not kill the process, which can continue to perform tasks just like any other serial process.

pvm_exit should be called by all PVM processes before they stop or exit for good. It must be called by processes that were not started with pvm_spawn.

EXAMPLES

C:

```
/* Program done */
pvm_exit();
exit();
```

Fortran:

```
CALL PVMFEXIT(INFO)
STOP
```

ERRORS

PvmSysErr   pvmd not responding

NAME
pvm_freebuf - Disposes of a message buffer.

SYNOPSIS
```
C    int info = pvm_freebuf( int bufid )

Fortran   call pvmffreebuf( bufid, info )
```

PARAMETERS
bufid   Integer message buffer identifier.

info   Integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION
The routine pvm_freebuf frees the memory associated with the message buffer identified by bufid. Message buffers are created by pvm_mkbuf, pvm_initsend, and pvm_recv. If pvm_freebuf is successful, info will be 0. If some error occurs then info will be < 0.

pvm_freebuf can be called for a send buffer created by pvm_mkbuf after the message has been sent and is no longer needed.

Receive buffers typically do not have to be freed unless they have been saved in the course of using multiple buffers. But pvm_freebuf can be used to destroy receive buffers as well. So messages that arrive but are no longer needed can be destroyed so they will not consume buffer space.

Typically multiple send and receive buffers are not needed and the user can simply use the pvm_initsend routine to reset the default send buffer.

There are several cases where multiple buffers are useful. One example where multiple message buffers are needed involves libraries or graphical interfaces that use PVM and interact with a running PVM application but do not want to interfere with the application's own communication.

When multiple buffers are used they generally are made and freed for each message that is packed. In fact, pvm_initsend simply does a pvm_freebuf followed by a pvm_mkbuf for the default buffer.

EXAMPLES

C:

```
bufid = pvm_mkbuf( PvmDataDefault );
        :
info = pvm_freebuf( bufid );
```

Fortran:

```
CALL PVMFMKBUF( PVMDEFAULT, BUFID )
        :
CALL PVMFFREEBUF( BUFID, INFO )
```

ERRORS

These error conditions can be returned by pvm_freebuf

PvmBadParam giving an invalid argument value.

PvmNoSuchBuf giving an invalid bufid value.

SEE ALSO

mkbuf(3PVM), initsend(3PVM), recv(3PVM)

NAME

pvm_gather - A specified member of the group receives messages from each member of the group and gathers these messages into a single array.

SYNOPSIS

```
C    int info = pvm_gather( void *result,  void *data,
                   int count, int datatype, int msgtag,
                   char *group, int rootginst)

Fortran    call pvmfgather(result, data, count, datatype,
                            msgtag, group, rootginst, info)
```

PARAMETERS

result On the root this is a pointer to the starting address of an array datatype of local values which are to be accumulated from the members of the group. If n if the number of members in the group, then this array of datatype should be of length at least n*count. This argument is meaningful only on the root.

data For each group member this is a pointer to the starting address of an array of length count of datatype which will be sent to the specified root member of the group.

count Integer specifying the number of elements of datatype to be sent by each member of the group to the root.

datatype Integer specifying the type of the entries in the result and data arrays.

msgtag Integer message tag supplied by the user. msgtag should be >= 0. It allows the user's program to distinguish between different kinds of messages.

group Character string group name of an existing group.

rootginst Integer instance number of group member who performs the gather of the messages from the members of the group.

info Integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION

pvm_gather() performs a send of messages from each member of the group to the specified root member of the group. All group members must call pvm_gather(), each sends its array data of length count of datatype to the root which accumulates these messages into its result array. It is as if the root receives count elements of datatype from the ith member of the group and places these values in its result array starting with offset i*count from the beginning of the result array. The root task is identified by its instance number in the group.

In using the scatter and gather routines, keep in mind that C stores multidimensional arrays in row order, typically starting with an initial index of 0; whereas, Fortran stores arrays in column order, typically starting with an offset of 1.

Note: pvm_gather() does not block. If a task calls pvm_gather and then leaves the group before the root has called pvm_gather an error may occur.

The current algorithm is very simple and robust. A future implementation may make more efficient use of the architecture to allow greater parallelism.

EXAMPLES

C:
```
   info =  pvm_gather(&getmatrix, &myrow, 10, PVM_INT,
                      msgtag, "workers", rootginst);
```

Fortran:
```
   CALL PVMFGATHER(GETMATRIX, MYCOLUMN, COUNT, INT4,
  &                MTAG, 'workers', ROOT, INFO)
```

ERRORS

These error conditions can be returned by pvm_gather

PvmNoInst	Calling task is not in the group
PvmBadParam	The datatype specified is not appropriate
PvmSysErr	Pvm system error

SEE ALSO

pvm_bcast(3PVM), pvm_barrier(3PVM), pvm_psend(3PVM)

NAME

pvm_getinst - Returns the instance number in a group of a PVM process.

SYNOPSIS

```
C    int inum = pvm_getinst( char *group, int tid )

Fortran   call pvmfgetinst( group, tid, inum )
```

PARAMETERS

group — Character string group name of an existing group.

tid — Integer task identifier of a PVM process.

inum — Integer instance number returned by the routine. Instance numbers start at 0 and count up. Values less than zero indicate an error.

DISCUSSION

The routine pvm_getinst takes a group name group and a PVM task identifier tid and returns the unique instance number that corresponds to the input. It can be called by any task whether in the group or not. If pvm_getinst is successful, inum will be >= 0. If some error occurs then inum will be < 0.

EXAMPLES

C:

```
inum = pvm_getinst( "worker", pvm_mytid() );
--------
inum = pvm_getinst( "worker", tid[i] );
```

Fortran:

```
CALL PVMFGETINST( 'GROUP3', TID, INUM )
```

ERRORS

These error conditions can be returned by pvm_getinst

PvmSysErr	pvmd was not started or has crashed.
PvmBadParam	giving an invalid tid value.
PvmNoGroup	giving a non-existent group name.
PvmNotInGroup	specifying a group in which the tid is not a member.

SEE ALSO

joingroup(3PVM), gettid(3PVM)

NAME

pvm_getopt - Returns the value of libpvm options.

SYNOPSIS

C int val = pvm_getopt(int what)

Fortran call pvmfgetopt(what, val)

PARAMETERS

what Integer defining option to get. One of:

PvmRoute	1	Message routing policy
PvmDebugMask	2	Libpvm debug mask
PvmAutoErr	3	Auto error reporting
PvmOutputTid	4	Stdout destination for children
PvmOutputCode	5	Output message tag
PvmTraceTid	6	Trace data destination for children
PvmTraceCode	7	Trace message tag
PvmFragSize	8	Message fragment size
PvmResvTids	9	Allow messages to reserved tags and TIDs
PvmSelfOutputTid	10	Stdout destination
PvmSelfOutputCode	11	Output message tag
PvmSelfTraceTid	12	Trace data destination
PvmSelfTraceCode	13	Trace message tag

val Integer returning the value of the option.

DISCUSSION

The routine pvm_getopt returns the value of the specified option in PVM. For a discussion of options and values, see pvm_setopt.

EXAMPLES

C:

```
val = pvm_getopt( PvmFragSize );
```

Fortran:

```
CALL PVMFGETOPT( PVMAUTOERR, VAL )
```

ERRORS

This error condition can be returned

PvmBadParam giving an invalid value.

SEE ALSO

setopt(3PVM)

NAME

pvm_getrbuf - Returns the message buffer identifier for the active receive buffer.

SYNOPSIS

```
C    int bufid = pvm_getrbuf( void )

Fortran   call pvmfgetrbuf( bufid )
```

PARAMETERS

bufid   Integer returning message buffer identifier for the active receive buffer.

DISCUSSION

The routine pvm_getrbuf returns the message buffer identifier bufid for the active receive buffer or 0 if there is no current buffer.

EXAMPLES

C:

```
bufid = pvm_getrbuf();
```

Fortran:

```
CALL PVMFGETRBUF( BUFID )
```

ERRORS

No error conditions are returned by pvm_getrbuf

SEE ALSO

getsbuf(3PVM)

NAME

pvm_getsbuf - Returns the message buffer identifier for the active send buffer.

SYNOPSIS

```
C       int bufid = pvm_getsbuf( void )

Fortran   call pvmfgetsbuf( bufid )
```

PARAMETERS

bufid   Integer returning message buffer identifier for the active send buffer.

DISCUSSION

The routine pvm_getsbuf returns the message buffer identifier bufid for the active send buffer or 0 if there is no current buffer.

EXAMPLES

C:
```
     bufid = pvm_getsbuf();
```

Fortran:
```
     CALL PVMFGETSBUF( BUFID )
```

ERRORS

No error conditions are returned by pvm_getsbuf

SEE ALSO

getrbuf(3PVM)

NAME

pvm_gettid - Returns the tid of the process identified by a group name and instance number.

SYNOPSIS

```
C    int tid = pvm_gettid( char *group, int inum )

Fortran   call pvmfgettid( group, inum, tid )
```

PARAMETERS

group — Character string that contains the name of an existing group.

inum — Integer instance number of the process in the group.

tid — Integer task identifier returned.

DISCUSSION

The routine pvm_gettid returns the tid of the PVM process identified by the group name group and the instance number inum. If pvm_gettid is successful, tid will be > 0. If some error occurs then tid will be < 0.

EXAMPLES

C:

```
tid = pvm_gettid("worker",0);
```

Fortran:

```
CALL PVMFGETTID('worker',5,TID)
```

ERRORS

These error conditions can be returned by pvm_gettid.

PvmSysErr	Can not contact the local pvmd most likely it is not running.
PvmBadParam	Bad Parameter most likely a NULL character string.
PvmNoGroup	No group exists by that name.
PvmNoInst	No such instance in the group.

SEE ALSO

joingroup(3PVM), getinst(3PVM)

NAME

pvm_gsize - Returns the number of members presently in the named group.

SYNOPSIS

```
C    int size = pvm_gsize( char *group )

Fortran   call pvmfgsize( group, size )
```

PARAMETERS

group   Character string group name of an existing group.

size   Integer returning the number of members presently in the group. Values less than zero indicate an error.

DISCUSSION

The routine pvm_gsize returns the size of the group named group. If there is an error size will be negative.

Since groups can change dynamically in PVM 3.0, this routine can only guarantee to return the instantaneous size of a given group.

EXAMPLES

C:

```
size = pvm_gsize( "worker" );
```

Fortran:

```
CALL PVMFGSIZE( 'group2', SIZE )
```

ERRORS

These error conditions can be returned by pvm_gsize

PvmSysErr	pvmd was not started or has crashed.
PvmBadParam	giving an invalid group name.

SEE ALSO

joingroup(3PVM)

NAME

pvm_halt - Shuts down the entire PVM system.

SYNOPSIS

```
C    int info = pvm_halt( void )

Fortran   call pvmfhalt( info )
```

PARAMETERS

info   Integer returns the error status.

DISCUSSION

The routine pvm_halt shuts down the entire PVM system including remote tasks, remote pvmds, the local tasks (including the calling task) and the local pvmd.

The task calling pvm_halt() must ignore or catch signal SIGTERM or it will be killed along with all the others.

ERRORS

The following error conditions can be returned

PvmSysErr   local pvmd is not responding.

SEE ALSO

delhosts(3PVM), kill(3PVM), exit(3PVM)

NAME

pvm_hostsync - Get time-of-day clock from PVM host.

SYNOPSIS

```
C    #include <sys/time.h>
     int info = pvm_hostsync( int host, struct timeval *clk,
          struct timeval *delta )

Fortran   call pvmfhostsync( host, clksec, clkusec, deltasec, deltausec, info )
```

PARAMETERS

host — TID of host.

clk — (or clksec and clkusec) Returns time-of-day clock sample from host.

delta — (or deltasec and deltausec) Returns difference between local clock and remote host clock.

DISCUSSION

pvm_hostsync samples the time-of day clock of a host in the virtual machine and returns both the clock value and the difference between local and remote clocks.

To reduce the delta error due to message transit time, local clock samples are taken before and after reading the remote clock. Delta is the difference between the mean local clocks and remote clock.

Note that the delta time can be negative. The microseconds field is always normalized to 0..999999, while the sign of the seconds field gives the sign of the delta. For example:

```
3.25 Sec =  3,250000
0        =  0,0
-1 uSec  = -1,999999
-1 Sec   = -1,000000
-1.1 Sec = -2,999000
```

In C, if clk or delta is a null pointer, that parameter is not returned.

ERRORS

If pvm_hostsync is successful, it returns PvmOk. These error conditions can be returned by pvm_hostsync

PvmSysErr	pvmd not responding.
PvmNoHost	specified host not in virtual machine.
PvmHostFail	host is unreachable (and thus possibly failed)

SEE ALSO

pvm_config(3PVM)

NAME

pvm_initsend - Clear default send buffer and specify message encoding.

SYNOPSIS

```
C    int bufid = pvm_initsend( int encoding )

Fortran   call pvmfinitsend( encoding, bufid )
```

PARAMETERS

encoding
: Integer specifying the next message's encoding scheme.

Options in C are:

Encoding value		MEANING
PvmDataDefault	0	XDR
PvmDataRaw	1	no encoding
PvmDataInPlace	2	data left in place

Option names are shortened in Fortran to:

Encoding value		MEANING
PVMDEFAULT	0	XDR
PVMRAW	1	no encoding
PVMINPLACE	2	data left in place

bufid
: Integer returned containing the message buffer identifier. Values less than zero indicate an error.

DISCUSSION

The routine pvm_initsend clears the send buffer and prepares it for packing a new message. The encoding scheme used for the packing is set by encoding. XDR encoding is used by default because PVM can not know if the user is going to add a heterogeneous machine before this message is sent. If the user knows that the next message will only be sent to a machine that understands the native format, then he can use PvmDataRaw encoding and save on encoding costs.

PvmDataInPlace encoding specifies that data be left in place during packing. The message buffer only contains the sizes and pointers to the items to be sent. When pvm_send is called the items are copied directly out of the user's memory. This option decreases the number of times a message is copied at the expense of requiring the user to not modify the items between the time they are packed and the time they are sent. PvmDataInPlace allows only dense (stride = 1) data in version 3.3.

If pvm_initsend is successful, then bufid will contain the message buffer identifier. If some error occurs then bufid will be < 0.

EXAMPLES

C:

```
bufid = pvm_initsend( PvmDataDefault );
info = pvm_pkint( array, 10, 1 );
msgtag = 3 ;
info = pvm_send( tid, msgtag );
```

Fortran:

```
CALL PVMFINITSEND(PVMRAW, BUFID)
CALL PVMFPACK( REAL4, DATA, 100, 1, INFO )
CALL PVMFSEND( TID, 3, INFO )
```

ERRORS

These error conditions can be returned by pvm_initsend

PvmBadParam	giving an invalid encoding value
PvmNoMem	Malloc has failed. There is not enough memory to create the buffer

SEE ALSO

mkbuf(3PVM)

NAME

pvm_joingroup - Enrolls the calling process in a named group.

SYNOPSIS

```
C       int inum = pvm_joingroup( char *group )

Fortran   call pvmfjoingroup( group, inum )
```

PARAMETERS

group   Character string group name of an existing group.

inum   Integer instance number returned by the routine. Instance numbers start at 0 and count up. Values less than zero indicate an error.

DISCUSSION

The routine pvm_joingroup enrolls the calling task in the group named group and returns the instance number inum of this task in this group. If there is an error inum will be negative.

Instance numbers start at 0 and count up. When using groups a (group, inum) pair uniquely identifies a PVM process. This is consistent with the PVM 2.4 naming schemes. If a task leaves a group by calling pvm_lvgroup and later rejoins the same group, the task is not guaranteed to get the same instance number. PVM attempts to reuse old instance numbers, so when a task joins a group it will get the lowest available instance number. A task can be a member of multiple groups simultaneously.

EXAMPLES

C:
```
inum = pvm_joingroup( "worker" );
```

Fortran:
```
CALL PVMFJOINGROUP( 'group2', INUM )
```

ERRORS

These error conditions can be returned by pvm_joingroup

PvmSysErr	pvmd was not started or has crashed.
PvmBadParam	giving a NULL group name.
PvmDupGroup	trying to join a group you are already in.

SEE ALSO lvgroup(3PVM)

NAME

pvm_kill - Terminates a specified PVM process.

SYNOPSIS

```
C    int info = pvm_kill( int tid )

Fortran   call pvmfkill( tid, info )
```

PARAMETERS

tid Integer task identifier of the PVM process to be killed (not yourself).

info Integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION

The routine pvm_kill sends a terminate (SIGTERM) signal to the PVM process identified by tid. In the case of multiprocessors the terminate signal is replaced with a host dependent method for killing a process. If pvm_kill is successful, info will be 0. If some error occurs then info will be < 0.

pvm_kill is not designed to kill the calling process. To kill yourself in C call pvm_exit() followed by exit(). To kill yourself in Fortran call pvmfexit followed by stop.

EXAMPLES

C:

```
info = pvm_kill( tid );
```

Fortran:

```
CALL PVMFKILL( TID, INFO )
```

ERRORS

These error conditions can be returned by pvm_kill

PvmBadParam	giving an invalid tid value.
PvmSysErr	pvmd not responding.

SEE ALSO

exit(3PVM), halt(3PVM)

NAME

pvm_lvgroup - Unenrolls the calling process from a named group.

SYNOPSIS

```
C    int info = pvm_lvgroup( char *group )

Fortran   call pvmflvgroup( group, info )
```

PARAMETERS

group   Character string group name of an existing group.

info   Integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION

The routine pvm_lvgroup unenrolls the calling process from the group named group. If there is an error info will be negative.

If a process leaves a group by calling either pvm_lvgroup or pvm_exit, and later rejoins the same group, the process may be assigned a new instance number. Old instance numbers are reassigned to processes calling pvm_joingroup.

EXAMPLES

C:

```
info = pvm_lvgroup( "worker" );
```

Fortran:

```
CALL PVMFLVGROUP( 'group2', INFO )
```

ERRORS

These error conditions can be returned by pvm_lvgroup

PvmSysErr	pvmd not responding.
PvmBadParam	giving a NULL group name.
PvmNoGroup	giving a non-existent group name.
PvmNotInGroup	asking to leave a group you are not a member of.

SEE ALSO

joingroup(3PVM)

NAME

pvm_mcast - Multicasts the data in the active message buffer to a set of tasks.

SYNOPSIS

```
C    int info = pvm_mcast( int *tids, int ntask, int  msgtag
)

Fortran   call pvmfmcast( ntask, tids, msgtag, info )
```

PARAMETERS

ntask
: Integer specifying the number of tasks to be sent to.

tids
: Integer array of length ntask containing the task IDs of the tasks to be sent to.

msgtag
: Integer message tag supplied by the user. msgtag should be >= 0. It allows the user's program to distinguish between different kinds of messages .

info
: Integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION

The routine pvm_mcast multicasts a message stored in the active send buffer to ntask tasks specified in the tids array. The message is not sent to the caller even if listed in the array of tids. The content of the message can be distinguished by msgtag. If pvm_mcast is successful, info will be 0. If some error occurs then info will be < 0.

The receiving processes can call either pvm_recv or pvm_nrecv to receive their copy of the multicast. pvm_mcast is asynchronous and based on a minimum spanning tree algorithm between the pvmds. Computation on the sending processor resumes as soon as the message is safely on its way to the receiving processors. This is in contrast to synchronous communication, during which computation on the sending processor halts until the matching receive is executed by the receiving processor.

pvm_mcast first determines which other pvmds contain the specified tasks. Then passes the message to these pvmds which in turn distribute the message to their local tasks without further network traffic.

Multicasting is not supported by most multiprocessor vendors. Typically their native calls only support broadcasting to {\m all the user's processes on a multiprocessor. Because of this omission, pvm_mcast may not be an efficient communication method on some multiprocessors except in the special case of broadcasting to all PVM processes.

EXAMPLES

C:

```
info = pvm_initsend( PvmDataRaw );
info = pvm_pkint( array, 10, 1 );
msgtag = 5 ;
info = pvm_mcast( tids, ntask, msgtag );
```

Fortran:

```
CALL PVMFINITSEND(PVMDEFAULT)
CALL PVMFPACK( REAL4, DATA, 100, 1, INFO )
CALL PVMFMCAST( NPROC, TIDS, 5, INFO )
```

ERRORS

These error conditions can be returned by pvm_mcast

PvmBadParam	giving a msgtag < 0.
PvmSysErr	pvmd not responding.
PvmNoBuf	no send buffer.

NAME

pvm_mkbuf - Creates a new message buffer.

SYNOPSIS

```
C    int bufid = pvm_mkbuf( int encoding )

Fortran   call pvmfmkbuf( encoding, bufid )
```

PARAMETERS

encoding
Integer specifying the buffer's encoding scheme.

Options in C are:

Encoding value		MEANING
PvmDataDefault	0	XDR
PvmDataRaw	1	no encoding
PvmDataInPlace	2	data left in place

Option names are shortened in Fortran to:

Encoding value		MEANING
PVMDEFAULT	0	XDR
PVMRAW	1	no encoding
PVMINPLACE	2	data left in place

bufid
Integer message buffer identifier returned. Values less than zero indicate an error.

DISCUSSION

The routine pvm_mkbuf creates a new message buffer and sets its encoding status to encoding. If pvm_mkbuf is successful, bufid will be the identifier for the new buffer, which can be used as a send buffer. If some error occurs then bufid will be < 0.

With the default setting XDR encoding is used when packing the message because PVM can not know if the user is going to add a heterogeneous machine before this message is sent. The other options to encoding allow the user to take advantage of knowledge about his virtual machine even when it is heterogeneous. For example, if the user knows that the next message will only be sent to a machine that understands the native format, then he can use PvmDataRaw encoding and save on encoding costs.

PvmDataInPlace encoding specifies that data be left in place during packing. The message buffer only contains the sizes and pointers to the items to be sent. When pvm_send is called the items are copied directly out of the user's memory. This option decreases the number of times a message is copied at the expense of requiring the user to not modify the items between the time they are packed and the time they are sent. PvmDataInPlace allows only dense (stride = 1) data in version 3.3.

pvm_mkbuf is required if the user wishes to manage multiple message buffers and should be used in conjunction with pvm_freebuf. pvm_freebuf should be called for a send buffer after a message has been sent and is no longer needed.

Receive buffers are created automatically by the pvm_recv and pvm_nrecv routines and do not have to be freed unless they have been explicitly saved with pvm_setrbuf.

Typically multiple send and receive buffers are not needed and the user can simply use the pvm_initsend routine to reset the default send buffer.

There are several cases where multiple buffers are useful. One example where multiple message buffers are needed involves libraries or graphical interfaces that use PVM and interact with a running PVM application but do not want to interfere with the application's own communication.

When multiple buffers are used they generally are made and freed for each message that is packed.

EXAMPLES

C:

```
bufid = pvm_mkbuf( PvmDataRaw );
  /* send message */
info = pvm_freebuf( bufid );
```

Fortran:

```
     CALL PVMFMKBUF(PVMDEFAULT, MBUF)
*    SEND MESSAGE HERE
     CALL PVMFFREEBUF( MBUF, INFO )
```

ERRORS

These error conditions can be returned by pvm_mkbuf

PvmBadParam	giving an invalid encoding value.
PvmNoMem	Malloc has failed. There is not enough memory to create the buffer

SEE ALSO

initsend(3PVM), freebuf(3PVM)

NAME

pvm_mstat - Returns the status of a host in the virtual machine.

SYNOPSIS

```
C    int mstat = pvm_mstat( char *host )

Fortran   call pvmfmstat( host, mstat )
```

PARAMETERS

host   Character string containing the host name.

mstat   Integer returning machine status:

value	MEANING
PvmOk	host is OK
PvmNoHost	host is not in virtual machine
PvmHostFail	host is unreachable (and thus possibly failed)

DISCUSSION

The routine pvm_mstat returns the status mstat of the computer named host with respect to running PVM processes. This routine can be used to determine if a particular host has failed and if the virtual machine needs to be reconfigured. The function pvm_notify() can also be used to notify the caller that a host has failed.

EXAMPLES

C:
```
mstat = pvm_mstat( "msr.ornl.gov" );
```

Fortran:
```
CALL PVMFMSTAT( 'msr.ornl.gov', MSTAT )
```

ERRORS

These error conditions can be returned by pvm_mstat

PvmSysErr	pvmd not responding.
PvmNoHost	giving a host name not in the virtual machine.
PvmHostFail	host is unreachable (and thus possibly failed).

SEE ALSO

notify(3PVM), config(3PVM)

NAME

pvm_mytid - Returns the tid of the calling process.

SYNOPSIS

```
C    int tid = pvm_mytid( void )

Fortran   call pvmfmytid( tid )
```

PARAMETERS

tid Integer returning the task identifier of the calling PVM process. Values less than zero indicate an error.

DISCUSSION

The routine pvm_mytid enrolls this process into PVM on its first call. It also generates a unique tid if this process was not created by pvm_spawn. pvm_mytid returns the tid of the calling process and can be called multiple times in an application.

Any PVM system call (not just pvm_mytid) will enroll a task in PVM if the task is not enrolled before the call.

The tid is a 32 bit positive integer created by the local pvmd. The 32 bits are divided into fields that encode various information about this process such as its location in the virtual machine (i.e. local pvmd address), the CPU number in the case where the process is on a multiprocessor, and a process ID field. This information is used by PVM and is not expected to be used by applications. Applications should not attempt to predict or interpret the tid with the exception of calling tidtohost()

If PVM has not been started before an application calls pvm_mytid the returned tid will be < 0.

EXAMPLES

C:

```
tid = pvm_mytid( );
```

Fortran:

```
CALL PVMFMYTID( TID )
```

ERRORS

This error condition can be returned by pvm_mytid

PvmSysErr pvmd not responding.

SEE ALSO

tidtohost(3PVM), parent(3PVM)

NAME

pvm_notify - Request notification of PVM event such as host failure.

SYNOPSIS

C int info = pvm_notify(int what, int msgtag, int cnt, int *tids)

Fortran call pvmfnotify(what, msgtag, cnt, tids, info)

PARAMETERS

what Type of event to trigger the notification. Presently one of:

Value	Meaning
PvmTaskExit	Task exits or is killed
PvmHostDelete	Host is deleted or crashes
PvmHostAdd	New host is added

msgtag Message tag to be used in notification.

cnt For PvmTaskExit and PvmHostDelete, specifies the length of the tids array. For PvmHostAdd, specifies the number of times to notify.

tids For PvmTaskExit and PvmHostDelete, an array of length cnt of task or pvmd TIDs to be notified about. The array is not used with the PvmHostAdd option.

info Integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION

The routine pvm_notify requests PVM to notify the caller on detecting certain events. One or more notify messages (see below) are sent by PVM back to the calling task. The messages have tag msgtag supplied to notify.

The notification messages have the following format:

PvmTaskExit
: One notify message for each TID requested. The message body contains a single TID of exited task.

PvmHostDelete
: One notify message for each TID requested. The message body contains a single pvmd-TID of exited pvmd.

PvmHostAdd
cnt notify messages are sent, one each time the local pvmd's host table is updated. The message body contains an integer length followed by a list of pvmd-TIDs of new pvmds. The counter of PvmHostAdd messages yet to be sent is replaced by successive calls to pvm_notify. Specifying a cnt of -1 turns on PvmHostAdd messages until a future notify; a count of zero disables them.

TIDs in the notify messages are packed as integers.

The calling task is responsible for receiving messages with the specified tag and taking appropriate action.

Future versions of PVM may expand the list of available notification events.

EXAMPLES

C:

```
info = pvm_notify( PvmTaskExit, 9999, ntask, tids )
```

Fortran:

```
CALL PVMFNOTIFY( PVMHOSTDELETE, 1111, NUMHOSTS, DTIDS, INFO )
```

ERRORS

PvmSysErr	pvmd not responding.
PvmBadParam	giving an invalid argument value.

SEE ALSO

tasks(3PVM), config(3PVM)

NAME

pvm_nrecv - Non-blocking receive.

SYNOPSIS

```
C    int bufid = pvm_nrecv( int tid, int msgtag )

Fortran   call pvmfnrecv( tid, msgtag, bufid )
```

PARAMETERS

tid Integer task identifier of sending process supplied by the user.

msgtag Integer message tag supplied by the user. msgtag should be >= 0.

bufid Integer returning the value of the new active receive buffer identifier. Values less than zero indicate an error.

DISCUSSION

The routine pvm_nrecv checks to see if a message with label msgtag has arrived from tid. and also clears the current receive buffer if any, If a matching message has arrived pvm_nrecv immediately places the message in a new active receive buffer, and returns the buffer identifier in bufid.

If the requested message has not arrived, then pvm_nrecv immediately returns with a 0 in bufid. If some error occurs bufid will be < 0.

A -1 in msgtag or tid matches anything. This allows the user the following options. If tid = -1 and msgtag is defined by the user, then pvm_nrecv will accept a message from any process which has a matching msgtag. If msgtag = -1 and tid is defined by the user, then pvm_nrecv will accept any message that is sent from process tid. If tid = -1 and msgtag = -1, then pvm_nrecv will accept any message from any process.

The PVM model guarantees the following about message order. If task 1 sends message A to task 2, then task 1 sends message B to task 2, message A will arrive at task 2 before message B. Moreover, if both messages arrive before task 2 does a receive, then a wildcard receive will always return message A.

pvm_nrecv is non-blocking in the sense that the routine always returns immediately either with the message or with the information that the message has not arrived at the local pvmd yet. pvm_nrecv can be called multiple times to check if a given message has arrived yet. In addition the blocking receive pvm_recv can be called for the same message if the application runs out of work it could do before the data arrives.

If pvm_nrecv returns with the message, then the data in the message can be unpacked into the user's memory using the unpack routines.

EXAMPLES

C:

```
tid = pvm_parent();
msgtag = 4 ;
arrived = pvm_nrecv( tid, msgtag );
if ( arrived > 0)
     info = pvm_upkint( tid_array, 10, 1 );
else
     /* go do other computing */
```

Fortran:

```
     CALL PVMFNRECV( -1, 4, ARRIVED )
     IF ( ARRIVED .gt. 0 ) THEN
        CALL PVMFUNPACK( INTEGER4, TIDS, 25, 1, INFO )
        CALL PVMFUNPACK( REAL8, MATRIX, 100, 100, INFO )
     ELSE
*       GO DO USEFUL WORK
     ENDIF
```

ERRORS

These error conditions can be returned by pvm_nrecv.

PvmBadParam	giving an invalid tid value or msgtag.
PvmSysErr	pvmd not responding.

SEE ALSO

recv(3PVM), unpack(3PVM), send(3PVM), mcast(3PVM)

NAME

pvm_pack - Pack the active message buffer with arrays of prescribed data type.

SYNOPSIS

C

```
int info = pvm_packf( const char *fmt, ... )
int info = pvm_pkbyte( char *xp, int nitem, int  stride )
int info = pvm_pkcplx( float *cp, int nitem, int stride )
int info = pvm_pkdcplx(  double  *zp,  int  nitem,  int stride )
int info = pvm_pkdouble( double  *dp,  int  nitem,  int stride )
int info =  pvm_pkfloat(  float  *fp,  int  nitem,  int stride )
int info = pvm_pkint( int *ip, int nitem, int stride )
int info = pvm_pkuint( unsigned int *ip, int nitem, int stride )
int info = pvm_pkushort( unsigned short *ip, int nitem, int stride )
int info = pvm_pkulong( unsigned long *ip,  int  nitem, int stride )
int info = pvm_pklong( long *ip, int nitem, int  stride )
int info =  pvm_pkshort(  short  *jp,  int  nitem,  int stride )
int info = pvm_pkstr( char *sp )
```

Fortran

```
call pvmfpack( what, xp, nitem, stride, info )
```

PARAMETERS

fmt
: Printf-like format expression specifying what to pack. (See discussion).

nitem
: The total number of items to be packed (not the number of bytes).

stride
: The stride to be used when packing the items. For example, if stride = 2 in pvm_pkcplx, then every other complex number will be packed.

xp
: Pointer to the beginning of a block of bytes. Can be any data type, but must match the corresponding unpack data type.

cp
: Complex array at least nitem*stride items long.

zp Double precision complex array at least nitem*stride items long.

dp Double precision real array at least nitem*stride items long.

fp Real array at least nitem*stride items long.

ip Integer array at least nitem*stride items long.

jp Integer*2 array at least nitem*stride items long.

sp Pointer to a null terminated character string.

what Integer specifying the type of data being packed.

what options

STRING	0	REAL4	4
BYTE1	1	COMPLEX8	5
INTEGER2	2	REAL8	6
INTEGER4	3	COMPLEX16	7

info Integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION

Each of the pvm_pk* routines packs an array of the given data type into the active send buffer. The arguments for each of the routines are a pointer to the first item to be packed, nitem which is the total number of items to pack from this array, and stride which is the stride to use when packing.

An exception is pvm_pkstr() which by definition packs a NULL terminated character string and thus does not need nitem or stride arguments. The Fortran routine pvmfpack(STRING, ...) expects nitem to be the number of characters in the string and stride to be 1.

If the packing is successful, info will be 0. If some error occurs then info will be < 0.

A single variable (not an array) can be packed by setting nitem = 1 and stride = 1.

The routine pvm_packf() uses a printf-like format expression to specify what and how to pack data into the send buffer. All variables are passed as addresses if count and stride are specified otherwise, variables are assumed to be values. A BNF-like description of the format syntax is:

```
format : null | init | format fmt
init : null | '%' '+'
```

```
fmt : '%' count stride modifiers fchar
fchar : 'c' | 'd' | 'f' | 'x' | 's'
count : null | [0-9]+ | '*'
stride : null | '.' ( [0-9]+ | '*' )
modifiers : null | modifiers mchar
mchar : 'h' | 'l' | 'u'
```

```
Formats:
  +  means initsend - must match an int (how) in the param list.
  c  pack/unpack bytes
  d  integers
  f  float
  x  complex float
  s  string

Modifiers:
  h  short (int)
  l  long  (int, float, complex float)
  u  unsigned (int)
```

Future extensions to the what argument in pvmfpack will include 64 bit types when XDR encoding of these types is available. Meanwhile users should be aware that precision can be lost when passing data from a 64 bit machine like a Cray to a 32 bit machine like a SPARCstation. As a mnemonic the what argument name includes the number of bytes of precision to expect. By setting encoding to PVMRAW (see pvmfinitsend) data can be transferred between two 64 bit machines with full precision even if the PVM configuration is heterogeneous.

Messages should be unpacked exactly like they were packed to insure data integrity. Packing integers and unpacking them as floats will often fail because a type encoding will have occurred transferring the data between heterogeneous hosts. Packing 10 integers and 100 floats then trying to unpack only 3 integers and the 100 floats will also fail.

EXAMPLES

C:

```
     info = pvm_initsend( PvmDataDefault );
     info = pvm_pkstr( "initial data" );
     info = pvm_pkint( &size, 1, 1 );
     info = pvm_pkint( array, size, 1 );
     info = pvm_pkdouble( matrix, size*size, 1 );
     msgtag = 3 ;
     info = pvm_send( tid, msgtag );

    int count, *iarry;
```

```
double darry[4];
pvm_packf("%+ %d %*d %4lf", PvmDataRaw, count, count, iarry, darry);
```

Fortran:

```
CALL PVMFINITSEND(PVMRAW, INFO)
CALL PVMFPACK( INTEGER4, NSIZE, 1, 1, INFO )
CALL PVMFPACK( STRING, 'row 5 of NXN matrix', 19, 1, INFO )
CALL PVMFPACK( REAL8, A(5,1), NSIZE, NSIZE , INFO )
CALL PVMFSEND( TID, MSGTAG, INFO )
```

ERRORS

PvmNoMem
: Malloc has failed. Message buffer size has exceeded the available memory on this host.

PvmNoBuf
: There is no active send buffer to pack into. Try calling pvm_initsend before packing message.

SEE ALSO

unpack(3PVM), initsend(3PVM)

NAME

pvm_parent - Returns the tid of the process that spawned the calling process.

SYNOPSIS

C int tid = pvm_parent(void)

Fortran call pvmfparent(tid)

PARAMETERS

tid Integer returns the task identifier of the parent of the calling process. If the calling process was not created with pvm_spawn, then tid = PvmNoParent.

DISCUSSION

The routine pvm_parent returns the tid of the process that spawned the calling process. If the calling process was not created with pvm_spawn, then tid is set to PvmNoParent.

EXAMPLES

C:

```
tid = pvm_parent();
```

Fortran:

```
CALL PVMFPARENT( TID )
```

ERRORS

This error condition can be returned by pvm_parent.

PvmNoParent	The calling process was not created with pvm_spawn.
PvmSysErr	Can't contact local pvmd.

NAME

pvm_perror - Prints message describing the last error returned by a PVM call.

SYNOPSIS

C int info = pvm_perror(char *msg)

Fortran call pvmfperror(msg, info)

PARAMETERS

msg Character string supplied by the user which will be prepended to the error message of the last PVM call.

info Integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION

The routine pvm_perror returns the error message of the last PVM call. The user can use msg to add additional information to the error message, for example, its location.

Unless redirected, all stdout and stderr messages are placed in the file /tmp/pvml.<uid> on the master host.

EXAMPLES

C:
```
if ( pvm_send( tid, msgtag ) )
     pvm_perror();
```

Fortran:
```
CALL PVMFSEND( TID, MSGTAG )
IF( INFO .LT. 0 ) CALL PVMFPERROR( 'Step 6', INFO )
```

ERRORS

No error condition is returned by pvm_perror.

NAME

pvm_precv - Receive a message directly into a buffer.

SYNOPSIS

```
C    int info = pvm_precv( int tid, int msgtag, char *buf,
                           int len, int datatype, int *atid,
                           int *atag, int *alen )

Fortran   call pvmfprecv( tid, msgtag, buf, len, datatype,
                        atid, atag, alen, info )
```

PARAMETERS

tid
: Integer task identifier of sending process (to match).

msgtag
: Integer message tag (to match). msgtag should be >= 0.

buf
: Pointer to a buffer to receive into.

len
: Length of buffer (in multiple of data type size).

datatype
: Type of data to which buf points (see below).

atid
: Returns actual TID of sender.

atag
: Returns actual message tag.

alen
: Returns actual message length.

info
: Returns PvmOk on success. Values less than zero indicate an error.

DISCUSSION

The routine pvm_precv blocks the process until a message with label msgtag has arrived from tid. pvm_precv then places the contents of the message in the supplied buffer, buf, up to a maximum length of len * (size of data type).

pvm_precv can receive messages sent by pvm_psend, pvm_send, pvm_mcast, or pvm_bcast.

A -1 in msgtag or tid matches anything. This allows the user the following options. If tid = -1 and msgtag is defined by the user, then pvm_precv will accept a message from any process which has a matching msgtag. If msgtag = -1 and tid is defined by the user, then pvm_precv will accept any message that is sent from process tid. If tid =

-1 and msgtag = -1, then pvm_precv will accept any message from any process.

In C the datatype parameter must be one of the following, depending on the type of data to be unpacked: [Version 3.3.0 - This parameter only determines message length, not data conversion. It only unpacks raw bytes]

datatype	Data Type
PVM_STR	string
PVM_BYTE	byte
PVM_SHORT	short
PVM_INT	int
PVM_FLOAT	real
PVM_CPLX	complex
PVM_DOUBLE	double
PVM_DCPLX	double complex
PVM_LONG	long integer
PVM_USHORT	unsigned short int
PVM_UINT	unsigned int
PVM_ULONG	unsigned long int

In Fortran the same data types specified for unpack should be used.

The PVM model guarantees the following about message order. If task 1 sends message A to task 2, then task 1 sends message B to task 2, message A will arrive at task 2 before message B. Moreover, if both messages arrive before task 2 does a receive, then a wildcard receive will always return message A.

If pvm_precv is successful, info will be = 0. If some error occurs then info will be < 0.

pvm_precv is blocking which means the routine waits until a message matching the user specified tid and msgtag arrives at the local pvmd. If the message has already arrived then pvm_precv returns immediately with the message.

pvm_precv does not affect the state of the current receive message buffer (created by the other receive functions).

EXAMPLES

C:

```
    info = pvm_precv( tid, msgtag, array, cnt, PVM_FLOAT,
                      &src, &rtag, &rlen );
```

Fortran:

```
    CALL PVMFPRECV( -1, 4, BUF, CNT, REAL4,
   >                SRC, RTAG, RCNT, INFO )
```

ERRORS

These error conditions can be returned by pvm_precv

PvmBadParam	giving an invalid tid, msgtag, or datatype.
PvmSysErr	pvmd not responding.

SEE ALSO

pvm_nrecv(3PVM), pvm_unpack(3PVM), pvm_probe(3PVM), pvm_recv(3PVM), pvm_send(3PVM), pvm_trecv(3PVM), pvm_mcast(3PVM)

NAME

pvm_probe - Check if message has arrived.

SYNOPSIS

```
C    int bufid = pvm_probe( int tid, int msgtag )

Fortran   call pvmfprobe( tid, msgtag, bufid )
```

PARAMETERS

tid Integer task identifier of sending process supplied by the user.

msgtag Integer message tag supplied by the user. msgtag should be >= 0.

bufid Integer returning the value of the new active receive buffer identifier. Values less than zero indicate an error.

DISCUSSION

The routine pvm_probe checks to see if a message with label msgtag has arrived from tid. If a matching message has arrived pvm_probe returns a buffer identifier in bufid. This bufid can be used in a pvm_bufinfo call to determine information about the message such as its source and length.

If the requested message has not arrived, then pvm_probe returns with a 0 in bufid. If some error occurs bufid will be < 0.

A -1 in msgtag or tid matches anything. This allows the user the following options. If tid = -1 and msgtag is defined by the user, then pvm_probe will accept a message from any process which has a matching msgtag. If msgtag = -1 and tid is defined by the user, then pvm_probe will accept any message that is sent from process tid. If tid = -1 and msgtag = -1, then pvm_probe will accept any message from any process.

pvm_probe can be called multiple times to check if a given message has arrived yet. After the message has arrived, pvm_recv must be called before the message can be unpacked into the user's memory using the unpack routines.

EXAMPLES

C:

```
     tid = pvm_parent();
     msgtag = 4 ;
```

```
          arrived = pvm_probe( tid, msgtag );
          if ( arrived )
               info = pvm_bufinfo( arrived, &len, &tag, &tid );
          else
               /* go do other computing */
```

Fortran:

```
          CALL PVMFPROBE( -1, 4, ARRIVED )
          IF ( ARRIVED .GT. 0 ) THEN
               CALL PVMFBUFINFO( ARRIVED, LEN, TAG, TID, INFO )
          ELSE
     *         GO DO USEFUL WORK
          ENDIF
```

ERRORS

These error conditions can be returned by pvm_probe.

PvmBadParam	giving an invalid tid value or msgtag.
PvmSysErr	pvmd not responding.

SEE ALSO

nrecv(3PVM), recv(3PVM), unpack(3PVM)

NAME

pvm_psend - Pack and send data in one call.

SYNOPSIS

```
C    int info = pvm_psend( int tid, int msgtag, char *buf,
                           int len, int datatype )

Fortran   call pvmfpsend( tid, msgtag, buf, len, datatype,
                          info )
```

PARAMETERS

tid
: Integer task identifier of destination process.

msgtag
: Integer message tag supplied by the user. msgtag should be >= 0.

buf
: Pointer to a buffer to send.

len
: Length of buffer (in multiple of data type size).

datatype
: Type of data to which buf points (see below).

info
: Integer status code returned by the routine.

DISCUSSION

The routine pvm_psend takes a pointer to a buffer buf its length len and its data type datatype and sends this data directly to the PVM task identified by tid. msgtag is used to label the content of the message. If pvm_psend is successful, info will be 0. If some error occurs then info will be < 0.

pvm_psend data can be received by pvm_precv, pvm_recv, pvm_trecv, or pvm_nrecv.

The pvm_psend routine is asynchronous. Computation on the sending processor resumes as soon as the buffer is safe for reuse. This is in contrast to synchronous communication, during which computation on the sending processor halts until the matching receive is executed by the receiving processor.

In C the datatype parameter must be one of the following, depending on the type of data to be sent: [Version 3.3.0 - This parameter only determines message length, not data conversion. It only sends raw bytes]

datatype	Data Type
PVM_STR	string

PVM_BYTE	byte
PVM_SHORT	short
PVM_INT	int
PVM_FLOAT	real
PVM_CPLX	complex
PVM_DOUBLE	double
PVM_DCPLX	double complex
PVM_LONG	long integer
PVM_USHORT	unsigned short int
PVM_UINT	unsigned int
PVM_ULONG	unsigned long int

In Fortran the same data types specified for pack should be used.

The PVM model guarantees the following about message order. If task 1 sends message A to task 2, then task 1 sends message B to task 2, message A will arrive at task 2 before message B. Moreover, if both messages arrive before task 2 does a receive, then a wildcard receive will always return message A.

pvm_psend does not affect the state of the current outgoing message buffer (created by pvm_initsend and used by pvm_send).

EXAMPLES

C:

```
info = pvm_psend( tid, msgtag, array, 1000, PVM_FLOAT );
```

Fortran:

```
CALL PVMFPSEND( TID, MSGTAG, BUF, CNT, REAL4, INFO )
```

ERRORS

These error conditions can be returned by pvm_psend

PvmBadParam	giving an invalid tid, msgtag, or datatype.
PvmSysErr	pvmd not responding.

SEE ALSO

initsend(3PVM), pack(3PVM), recv(3PVM), send(3PVM)

NAME

pvm_pstat - Returns the status of the specified PVM process.

SYNOPSIS

```
C    int status = pvm_pstat( tid )

Fortran   call pvmfpstat( tid, status )
```

PARAMETERS

tid Integer task identifier of the PVM process in question.

status Integer returns the status of the PVM process identified by tid. Status is PvmOk if the task is running, PvmNoTask if not, and PvmBadParam if the tid is bad.

DISCUSSION

The routine pvm_pstat returns the status of the process identified by tid. Also note that pvm_notify() can be used to notify the caller that a task has failed.

EXAMPLES

C:

```
tid = pvm_parent();
status = pvm_pstat( tid );
```

Fortran:

```
CALL PVMFPARENT( TID )
CALL PVMFPSTAT( TID, STATUS )
```

ERRORS

The following error conditions can be returned by pvm_pstat.

PvmBadParam	Bad Parameter most likely an invalid tid value.
PvmSysErr	pvmd not responding.
PvmNoTask	Task not running.

SEE ALSO

notify(3PVM)

NAME

pvm_recv - Receive a message.

SYNOPSIS

```
C    int bufid = pvm_recv( int tid, int msgtag )

Fortran   call pvmfrecv( tid, msgtag, bufid )
```

PARAMETERS

tid Integer task identifier of sending process supplied by the user.

msgtag Integer message tag supplied by the user. msgtag should be >= 0.

bufid Integer returns the value of the new active receive buffer identifier. Values less than zero indicate an error.

DISCUSSION

The routine pvm_recv blocks the process until a message with label msgtag has arrived from tid. pvm_recv then places the message in a new active receive buffer, which also clears the current receive buffer.

A -1 in msgtag or tid matches anything. This allows the user the following options. If tid = -1 and msgtag is defined by the user, then pvm_recv will accept a message from any process which has a matching msgtag. If msgtag = -1 and tid is defined by the user, then pvm_recv will accept any message that is sent from process tid. If tid = -1 and msgtag = -1, then pvm_recv will accept any message from any process.

The PVM model guarantees the following about message order. If task 1 sends message A to task 2, then task 1 sends message B to task 2, message A will arrive at task 2 before message B. Moreover, if both messages arrive before task 2 does a receive, then a wildcard receive will always return message A.

If pvm_recv is successful, bufid will be the value of the new active receive buffer identifier. If some error occurs then bufid will be < 0.

pvm_recv is blocking which means the routine waits until a message matching the user specified tid and msgtag values arrives at the local pvmd. If the message has already arrived then pvm_recv returns immediately with the message.

Once pvm_recv returns, the data in the message can be unpacked into the user's memory using the unpack routines.

EXAMPLES

C:

```
tid = pvm_parent();
msgtag = 4 ;
bufid = pvm_recv( tid, msgtag );
info = pvm_upkint( tid_array, 10, 1 );
info = pvm_upkint( problem_size, 1, 1 );
info = pvm_upkfloat( input_array, 100, 1 );
```

Fortran:

```
CALL PVMFRECV( -1, 4, BUFID )
CALL PVMFUNPACK( INTEGER4, TIDS, 25, 1, INFO )
CALL PVMFUNPACK( REAL8, MATRIX, 100, 100, INFO )
```

ERRORS

These error conditions can be returned by pvm_recv

PvmBadParam	giving an invalid tid value, or msgtag < -1.
PvmSysErr	pvmd not responding.

SEE ALSO

nrecv(3PVM), unpack(3PVM), probe(3PVM), send(3PVM), mcast(3PVM)

NAME

pvm_recvf - Redefines the comparison function used to accept messages.

SYNOPSIS

```
C    int (*old)() = pvm_recvf( int (*new)(  int  bufid,  int
tid, int tag ))

Fortran   NOT AVAILABLE
```

DISCUSSION

The routine pvm_recvf defines the comparison function to be used by the pvm_recv, pvm_nrecv, and pvm_probe functions. It is available as a means to customize PVM message passing. pvm_recvf sets a user supplied comparison function to evaluate messages for receiving.

recvf returns the old value of the matching function, or 0 if the old function was the default matcher

pvm_recvf is intended for sophisticated C programmers who understand the function of such routines (like signal) and who require a receive routine that can match on more complex message contexts than the default provides.

MATCHING FUNCTION

The default comparison function evaluates the source and message tag associated with all incoming messages.

PARAMETERS

tid   Integer task identifier of sending process supplied by the user.

tag   Integer message tag supplied by the user.

bufid   Integer message buffer identifier.

The matching function should return:

Value	Action taken
< 0	return immediately with this error code
0	do not pick this message
1	pick this message and do not scan the rest
> 1	pick this highest ranked message after scanning them all

Example: Implementing probe with recvf

```
#include <pvm3.h>

static int foundit = 0;

static int
foo_match(mid, tid, code)
     int mid;
     int tid;
     int code;
{
     int t, c, cc;

     if ((cc = pvm_bufinfo(mid, (int*)0, &c, &t)) < 0)
          return cc;

     if ((tid == -1 || tid == t) && (code == -1 || code == c))
          foundit = 1;

     return 0;
}

int
probe(src, code)
{
     int (*omatch)();
     int cc;

     omatch = pvm_recvf(foo_match);

     foundit = 0;

     if ((cc = pvm_nrecv(src, code)) < 0)
          return cc;

     pvm_recvf(omatch);

     return foundit;
}
```

ERRORS

No error conditions are returned by pvm_recvf

SEE ALSO

recv(3PVM), nrecv(3PVM), probe(3PVM)

NAME

pvm_reduce - Performs a reduce operation over members of the specified group.

SYNOPSIS

```
C    int info = pvm_reduce( void (*func)(),
                  void *data, int count, int datatype,
                  int msgtag, char *group, int rootginst)

Fortran    call pvmfreduce(func, data, count, datatype,
                           msgtag, group, rootginst, info)
```

PARAMETERS

func Function which defines the operation performed on the global data. Predefined are PvmMax, PvmMin, PvmSum and PvmProduct. Users can define their own function.

```
SYNOPSIS for func
C   void func(int *datatype, void *x, void *y,
              int *num, int *info)
Fortran    call func(datatype, x, y, num, info)
```

data Pointer to the starting address of an array of local values. On return, the data array on the root will be overwritten with the result of the reduce operation over the group.

count Integer specifying the number of elements of datatype in the data array.

datatype Integer specifying the type of the entries in the data array. (See below for defined types)

msgtag Integer message tag supplied by the user. msgtag should be >= 0. It allows the user's program to distinguish between different kinds of messages.

group Character string group name of an existing group.

rootginst Integer instance number of group member who gets the result.

info Integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION

pvm_reduce() performs global operations such as max, min, and sum over all the tasks in a group. All group members call pvm_reduce() with their local data, and the result of the reduction operation appears on the user specified root task root. The root task is identified by its instance number in the group.

PVM supplies the following predefined functions that can be specified in func.

```
PvmMin
PvmMax
PvmSum
PvmProduct
```

PvmMax and PvmMin are implemented for datatypes byte, short, integer, long, float, double, complex, and double complex. For complex values the minimum [maximum] is that complex pair with the minimum [maximum] modulus. PvmSum and PvmProduct are implemented for datatypes short, integer, long, float, double, complex, and double complex.

C and Fortran defined datatypes are:

C datatypes	FORTRAN datatypes
PVM_BYTE	BYTE1
PVM_SHORT	INT2
PVM_INT	INT4
PVM_FLOAT	REAL4
PVM_CPLX	COMPLEX8
PVM_DOUBLE	REAL8
PVM_DCPLX	COMPLEX16
PVM_LONG	

A user defined function may be used used in func. The argument func is a function with four arguments. It is the base function used for the reduction operation. Both x and y are arrays of type specified by datatype with num entries. The arguments datatype and info are as specified above. The arguments x and num correspond to data and count above. The argument y contains received values.

Note: pvm_reduce() does not block. If a task calls pvm_reduce and then leaves the group before the root has called pvm_reduce an error may occur.

The current algorithm is very simple and robust. A future implementation may make more efficient use of the architecture to allow greater parallelism.

EXAMPLES

C:

```
info =  pvm_reduce(PvmMax, &myvals, 10, PVM_INT,
                   msgtag, "workers", rootgisnt);
```

Fortran:

```
   CALL PVMFREDUCE(PvmMax, MYVALS, COUNT, INT4,
  &                MTAG, 'workers', ROOT, INFO)
```

ERRORS

These error conditions can be returned by pvm_reduce

PvmNoInst	Calling task is not in the group
PvmBadParam	The datatype specified is not appropriate for the specified reduction function.
PvmSysErr	Pvm system error

SEE ALSO

pvm_bcast(3PVM), pvm_barrier(3PVM), pvm_psend(3PVM)

NAME

pvm_reg_hoster - Register task as PVM slave starter.

SYNOPSIS

C
```
#include <pvmsdpro.h>

int cc = pvm_reg_hoster()
```

Fortran Not Available

DISCUSSION

Registers the calling task as a PVM slave pvmd starter. When the master pvmd receives a DM_ADD message, instead of starting the new slave pvmd processes itself, it passes a message to the hoster, which does the dirty work and sends a message back to the pvmd.

Note: This function isn't for beginners. If you don't grok what it does, you probably don't need it. For a more complete explanation of what's going on here, you should refer to the PVM source code and/or user guide section on implementation; this is just a man page. That said...

When the master pvmd receives a DM_ADD message (request to add hosts to the virtual machine), it looks up the new host IP addresses, gets parameters from the host file if it was started with one, and sets default parameters. It then either attempts to start the processes (using rsh or rexec()) or, if a hoster has registered, sends it a SM_STHOST message.

The format of the SM_STHOST message is:

```
int nhosts                // number of hosts
{
    int tid               // of host
    string options        // from hostfile so= field
    string login          // in form ``[username@]hostname.domain''
    string command        // to run on remote host
} [nhosts]
```

The hoster should attempt to run each command on each host and record the result. A command usually looks like:

```
$PVM_ROOT/lib/pvmd -s -d8 -nhonk 1 80a9ca95:0f5a 4096 3
80a95c43:0000
```

and a reply from a slave pvmd like:

```
ddpro<2312> arch<ALPHA> ip<80a95c43:0b3f> mtu<4096>
```

When finished, the hoster should send a SM_STHOSTACK message back to the address of the sender (the master pvmd). The format of the reply message is:

```
{
    int tid                // of host, must match request
    string status          // result line from slave or error code
} []                       // implied count
```

The TIDs in the reply must match those in the request. They may be in a different order, however.

The result string should contain the entire reply (a single line) from each new slave pvmd, or an error code if something went wrong. Legal error codes are the literal names of the pvm_errno codes, for example ``PvmCantStart''. The default PVM hoster can return PvmDSysErr or PvmCantStart, and the slave pvmd itself can return PvmDupHost.

The hoster must use pvm_setmwid() to set the wait ID in the reply message to the same value as in the request. The wait ID in the request is obtained by calling pvm_getmwid().

The hoster task must use pvm_setopt(PvmResvTids, 1) to allow sending reserved messages. Messages must be packed using data format PvmDataFoo.

pvm_reg_hoster() returns PvmOk when successful.

SEE ALSO

pvm_addhosts(3PVM), pvm_config(3PVM), pvm_setmwid(3PVM)

NAME

pvm_reg_rm - Register task as PVM resource manager.

SYNOPSIS

C

```
#include <pvmsdpro.h>

int cc = pvm_reg_rm( struct pvmhostinfo **hip )
struct pvmhostinfo {
     int  hi_tid;
     char *hi_name;
     char *hi_arch;
     int  hi_speed;
};
```

Fortran Not Available

PARAMETERS

hostp

Returns pointer to a pvmhostinfo structure which contains information about the master host.

DISCUSSION

Registers the calling task as a PVM task and slave host manager. This means it intercepts certain libpvm calls from other tasks in order to have a say in scheduling policy. The resource manager will asynchronously receive messages from tasks containing requests for service, as well as messages from pvmds notifying it of system failures.

Before you start using this function, be warned that it's not a trivial thing, i.e. you can't just call it to turn off the default round-robin task assignment. Rather, it allows you to write your own resource manager and hook it to PVM.

To understand what the following messages mean, you should refer to the PVM source code and/or user guide section on implementation; There's just too much to say about them.

When one of the following libpvm functions is called in a task with resource manager set, the given message tag is sent to to resource manager.

Libpvm call	RM message	Normal message
pvm_addhosts()	SM_ADDHOST	TM_ADDHOST
pvm_config()	SM_CONFIG	TM_CONFIG
pvm_delhosts()	SM_DELHOST	TM_DELHOST
pvm_notify()	SM_NOTIFY	TM_NOTIFY
pvm_spawn()	SM_SPAWN	TM_SPAWN
pvm_tasks()	SM_TASK	TM_TASK

pvm_reg_rm()	SM_SCHED	TM_SCHED

The resource manager must in turn compose the following messages and send them to the pvmds:

RM message	Normal message
SM_EXEC	DM_EXEC
SM_EXECACK	DM_EXECACK
SM_ADD	DM_ADD
SM_ADDACK	DM_ADDACK
SM_HANDOFF	(none - change of resource manager)

The following messages are sent asynchronously to the resource manager by the system:

RM message	Meaning
SM_TASKX	notify of task exit/fail
SM_HOSTX	notify of host delete/fail

The resource manager task must use pvm_setopt(PvmResvTids, 1) to allow sending reserved messages. Messages must be packed using data format PvmDataFoo.

pvm_reg_rm() returns PvmOk when successful.

SEE ALSO

pvm_addhosts(3PVM), pvm_config(3PVM), pvm_delhosts(3PVM), pvm_notify(3PVM), pvm_spawn(3PVM), pvm_tasks(3PVM)

NAME

pvm_reg_tasker - Register task as PVM task starter.

SYNOPSIS

```
C    #include <pvmsdpro.h>

     int cc = pvm_reg_tasker()

Fortran   Not Available
```

DISCUSSION

Registers the calling task as a PVM task starter. When a tasker is registered with a pvmd, and the pvmd receives a DM_EXEC message, instead of fork()ing and exec()ing the task itself, it passes a message to the tasker, which does the dirty work and sends a message back to the pvmd.

Note: If this doesn't make sense, don't worry about it. This function is for folks who are writing stuff like debugger servers and so on. For a more complete explanation of what's going on here, you should refer to the PVM source code and/or user guide section on implementation; this is only a man page. That said...

When the pvmd receives a DM_EXEC message (request to exec new tasks), it searches epath (the PVM executable search path) for the file name. If it finds the file, it then either attempts to start the processes (using fork() and exec()) or, if a tasker has registered, sends it a SM_STTASK message.

The format of the SM_STTASK message is:

```
int tid              // of task
int flags            // as passed to spawn()
string path          // absolute path of the executable
int argc             // number of args to process
string argv[argc]    // args
int nenv             // number of envars to pass to task
string env[nenv]     // environment strings
```

The tasker must attempt to start the process when it gets one of these messages. The tasker doesn't reply to the pvmd if the task is successfully started; the task will reconnect to the pvmd on its own (using the identifier in envar PVMEPID).

The tasker must send a SM_TASKX message to the pvmd when any task that it owns (has started) exits, or if it can't start a particular task. The format of the SM_TASKX message is:

```
int tid               // of task
int status            // the Unix exit status (from wait())
int u_sec             // user time used by the task, seconds
int u_usec            // microseconds
int s_sec             // system time used by the task, seconds
int s_usec            // microseconds
```

The tasker task must use pvm_setopt(PvmResvTids, 1) to allow sending reserved messages. Messages must be packed using data format PvmDataFoo.

pvm_reg_tasker() returns PvmOk when successful.

SEE ALSO

pvm_spawn(3PVM), pvm_tasks(3PVM),

NAME

pvm_scatter - Sends to each member of a group a section of an array from a specified member of the group.

SYNOPSIS

```
C    int info = pvm_scatter( void *result, void *data,
                    int count, int datatype, int msgtag,
                    char *group, int rootginst)

Fortran    call pvmfscatter(result, data, count, datatype,
                         msgtag, group, rootginst, info)
```

PARAMETERS

result Pointer to the starting address of an array of length count of datatype which will be overwritten by the message from the specified root member of the group.

data On the root this is a pointer to the starting address of an array datatype of local values which are to be distributed to the members of the group. If n is the number of members in the group, then this array of datatype should be of length at least n*count. This argument is meaningful only on the root.

count Integer specifying the number of elements of datatype to be sent to each member of the group from the root.

datatype Integer specifying the type of the entries in the result and data arrays.

msgtag Integer message tag supplied by the user. msgtag should be >= 0. It allows the user's program to distinguish between different kinds of messages.

group Character string group name of an existing group.

rootginst Integer instance number of group member who performs the scatter of its array to the members of the group.

info Integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION

pvm_scatter() performs a scatter of data from the specified root member of the group to each of the members of the group, including itself. All group members must call pvm_scatter(), each receives a portion of the data array from the root in their local result array. It is as if the root node sends to the ith member of the group count elements from its array data starting at offset i*count from the beginning of the data array. And, it is as if, each member of the group performs a corresponding receive of count values of datatype into its result array. The root task is identified by its instance number in the group.

In using the scatter and gather routines, keep in mind that C stores multidimensional arrays in row order, typically starting with an initial index of 0; whereas, Fortran stores arrays in column order, typically starting with an offset of 1.

The current algorithm is very simple and robust. A future implementation may make more efficient use of the architecture to allow greater parallelism.

EXAMPLES

C:

```
info =  pvm_scatter(&getmyrow, &matrix, 10, PVM_INT,
                    msgtag, "workers", rootginst);
```

Fortran:

```
   CALL PVMFSCATTER(GETMYCOLUMN, MATRIX, COUNT, INT4,
  &                MTAG, 'workers', ROOT, INFO)
```

ERRORS

These error conditions can be returned by pvm_scatter

PvmNoInst	Calling task is not in the group
PvmBadParam	The datatype specified is not appropriate
PvmSysErr	Pvm system error

SEE ALSO

pvm_bcast(3PVM), pvm_barrier(3PVM), pvm_psend(3PVM)

NAME

pvm_send - Immediately sends the data in the active message buffer.

SYNOPSIS

C int info = pvm_send(int tid, int msgtag)

Fortran call pvmfsend(tid, msgtag, info)

PARAMETERS

tid Integer task identifier of destination process.

msgtag Integer message tag supplied by the user. msgtag should be >= 0.

info Integer status code returned by the routine.

DISCUSSION

The routine pvm_send sends a message stored in the active send buffer to the PVM process identified by tid. msgtag is used to label the content of the message. If pvm_send is successful, info will be 0. If some error occurs then info will be < 0.

The pvm_send routine is asynchronous. Computation on the sending processor resumes as soon as the message is safely on its way to the receiving processor. This is in contrast to synchronous communication, during which computation on the sending processor halts until the matching receive is executed by the receiving processor.

The PVM model guarantees the following about message order. If task 1 sends message A to task 2, then task 1 sends message B to task 2, message A will arrive at task 2 before message B. Moreover, if both messages arrive before task 2 does a receive, then a wildcard receive will always return message A.

EXAMPLES

C:

```
info = pvm_initsend( PvmDataDefault );
info = pvm_pkint( array, 10, 1 );
msgtag = 3 ;
info = pvm_send( tid, msgtag );
```

Fortran:

```
CALL PVMFINITSEND(PVMRAW, INFO)
CALL PVMFPACK( REAL8, DATA, 100, 1, INFO )
```

```
CALL PVMFSEND( TID, 3, INFO )
```

ERRORS

These error conditions can be returned by pvm_send

PvmBadParam	giving an invalid tid or a msgtag.
PvmSysErr	pvmd not responding.
PvmNoBuf	no active send buffer. Try pvm_initsend() before send.

SEE ALSO

initsend(3PVM), pack(3PVM), recv(3PVM)

NAME

pvm_sendsig - Sends a signal to another PVM process.

SYNOPSIS

```
C    int info = pvm_sendsig( int tid, int signum )

Fortran   call pvmfsendsig( tid, signum, info )
```

PARAMETERS

tid — Integer task identifier of PVM process to receive the signal.

signum — Integer signal number.

info — Integer status code returned by the routine.

DISCUSSION

The routine pvm_sendsig sends the signal number signum to the PVM process identified by tid. If pvm_sendsig is successful, info will be 0. If some error occurs then info will be < 0.

pvm_sendsig should only be used by programmers with Unix signal handling experience. Many library functions (and in fact the PVM library functions) cannot be called in a signal handler context because they do not mask signals or lock internal data structures.

Further caveat: the signal numbers passed between systems are not mapped - PVM assumes that signal 9 on one system is the same as on another.

EXAMPLES

C:

```
tid = pvm_parent();
info = pvm_sendsig( tid, SIGKILL);
```

Fortran:

```
CALL PVMFBUFINFO( BUFID, BYTES, TYPE, TID, INFO );
CALL PVMFSENDSIG( TID, SIGNUM, INFO )
```

ERRORS

These error conditions can be returned by pvm_sendsig

PvmSysErr	pvmd not responding.
PvmBadParam	giving an invalid tid value.

NAME

pvm_serror - (PVM 3.1) Sets automatic error message printing on or off. Replaced by pvm_setopt().

SYNOPSIS

```
C    int oldset = pvm_serror( int set )

Fortran   call pvmfserror( set, oldset )
```

PARAMETERS

set — Integer defining whether detection is to be turned on (1) or off (0).

oldset — Integer defining the previous setting of pvm_serror.

DISCUSSION

The routine pvm_serror sets automatic error message printing for all subsequent PVM calls by this process. Any PVM routines that return an error condition will automatically print the associated error message. The argument set defines whether this detection is to be turned on (1) or turned off (0) for subsequent calls. In the future a value of (2) will cause the program to exit after printing the error message. pvm_serror returns the previous value of set in oldset.

All stdout and stderr messages are placed in the file /tmp/pvml.<uid> on the master pvmd's host.

EXAMPLES

C:

```
info = pvm_serror( 1 );
```

Fortran:

```
CALL PVMFSERROR( 0, INFO )
```

ERRORS

This error condition can be returned by pvm_serror

PvmBadParam   giving an invalid set value.

SEE ALSO

setopt(3PVM)

NAME

pvm_getmwid, pvm_setmwid - Get or set wait ID of a message.

SYNOPSIS

```
C    int info = pvm_getmwid( int bufid )
     int info = pvm_setmwid( int bufid, int waitid )

Fortran   Not Available
```

PARAMETERS

bufid   Message buffer identifier.

waitid Wait identifier number.

DISCUSSION

A wait identifier is part of a message (like the source, destination, tag and body). It is used to match a reply to the corresponding request.

The default wait ID for a message is zero (none).

pvm_getmwid returns the wait ID associated with a message buffer, or a negative error code.

pvm_setmwid assigns a new wait ID to a message buffer. It returns PvmOk unless an error occurs.

EXAMPLES

```
int src, tag;

pvm_recv(-1, -1);
/* ... process, compose reply message */
pvm_setmwid(pvm_getsbuf(), pvm_getmwid(pvm_getrbuf()));
pvm_bufinfo(pvm_getrbuf(), (int *)0, &tag, &src);
pvm_send(src, tag);
```

ERRORS

The following error conditions can be returned by pvm_getmwid or pvm_setmwid:

PvmBadParam   invalid value for bufid argument.
PvmNoSuchBuf message buffer bufid doesn't exist.

SEE ALSO

pvm_bufinfo(3PVM)

NAME

pvm_setopt - Sets libpvm options.

SYNOPSIS

```
C    int oldval = pvm_setopt( int what, int val )

Fortran   call pvmfsetopt( what, val, oldval )
```

PARAMETERS

what Integer defining what to set. One of:

PvmRoute	1	Message routing policy
PvmDebugMask	2	Libpvm debug mask
PvmAutoErr	3	Auto error reporting
PvmOutputTid	4	Stdout destination for children
PvmOutputCode	5	Output message tag
PvmTraceTid	6	Trace data destination for children
PvmTraceCode	7	Trace message tag
PvmFragSize	8	Message fragment size
PvmResvTids	9	Allow messages to reserved tags and TIDs
PvmSelfOutputTid	10	Stdout destination
PvmSelfOutputCode	11	Output message tag
PvmSelfTraceTid	12	Trace data destination
PvmSelfTraceCode	13	Trace message tag

val Integer specifying new setting of option. Predefined values for PvmRoute are:

PvmDontRoute	1	Don't request or grant connections
PvmAllowDirect	2	(Default) Don't request but allow
PvmRouteDirect	3	Request and allow connections

oldval Integer returning the previous setting of the option.

DISCUSSION

The routine pvm_setopt is a general purpose function used to set miscellaneous options in the PVM library. What is one of:

PvmRoute

Advises PVM on whether to set up direct task-to-task links PvmRouteDirect (using TCP) for all subsequent communication. Once a link is established it persists until the application finishes. If a direct link can not be established because one of the two tasks has requested PvmDontRoute or because adequate resources aren't available, then the default route through the PVM daemons is used. On multiprocessors such as Intel Paragon this option is ignored because the communication between tasks on these machines always uses the native protocol for direct communication. pvm_setopt can be called multiple times to selectively establish direct links, but is typically set only once near the beginning of each task. PvmAllowDirect is the default route setting. This setting on task A allows other tasks to set up direct links to A. Once a direct link is established between tasks both tasks will use it for sending messages.

PvmDebugMask
: When debugging is turned on, PVM will log detailed information about its operations and progress on its stderr stream. val is the debugging level. Default is not to print any debug information.

PvmAutoErr
: When an error results from a libpvm function call and PvmAutoErr is set to 1 (the default), an error message is automatically printed on stderr. Setting it to 0 disables this, while setting it to 2 causes the library to terminate the task after printing the an error message.

PvmOutputTid
: Sets the stdout destination for children tasks (spawned after the call to pvm_setopt). Everything printed on the standard output of tasks spawned by the calling task is packed into messages and sent to the destination. val is the TID of a PVM task. Setting PvmOutputTid to 0 redirects stdout to the master pvmd, which writes to the log file /tmp/pvml.<uid> The default setting is inherited from the parent task, else is 0.

PvmOutputCode
: Sets the message tag for standard output messages. Should only be set when a task has PvmOutputTid set to itself.

PvmTraceTid
: Sets the trace data message destination for children tasks (spawned after the call to pvm_setopt). Libpvm trace data is sent as messages to the destination. val is the TID of a PVM task. Setting PvmTraceTid to 0 discards trace data. The default setting is inherited from the parent task, else is 0.

PvmTraceCode
: Sets the message tag for trace data messages. Should only be set when a task has PvmTraceTid set to itself.

PvmFragSize
: Val specifies the message fragment size in bytes. Default value varies with host architecture.

PvmResvTids
: A val of 1 enables the task to send messages with reserved tags and to non-task destinations. The default (0) causes libpvm to generate a PvmBadParam error when a reserved identifier is specified.

PvmSelfOutputTid
: Sets the stdout destination for the task. Everything printed on stdout is packed into messages and sent to the destination. Note: this only works for spawned tasks, because the pvmd doesn't get the output from tasks started by other means. val is the TID of a PVM task. Setting PvmSelfOutputTid to 0 redirects stdout to the master pvmd, which writes to the log file /tmp/pvml.<uid>. The default setting is inherited from the parent task, else is 0. Setting either PvmSelfOutputTid or PvmSelfOutputCode also causes both PvmOutputTid and PvmOutputCode to take on the values of PvmSelfOutputTid and PvmSelfOutputCode, respectively.

PvmSelfOutputCode
: Sets the message tag for standard output messages.

PvmSelfTraceTid
: Sets the trace data message destination for the task. Libpvm trace data is sent as messages to the destination. val is the TID of a PVM task. Setting PvmSelfTraceTid to 0 discards trace data. The default setting is inherited from the parent task, else is 0. Setting either PvmSelfTraceTid or PvmSelfTraceCode also causes both PvmTraceTid and PvmTraceCode to take on the values of PvmSelfTraceTid and PvmSelfTraceCode, respectively.

PvmSelfTraceCode
: Sets the message tag for trace data messages.

pvm_setopt returns the previous value of the option.

EXAMPLES

C:

```
oldval = pvm_setopt( PvmRoute, PvmRouteDirect );
```

Fortran:

```
CALL PVMFSETOPT( PVMAUTOERR, 1, OLDVAL )
```

ERRORS

This error condition can be returned

PvmBadParam giving an invalid value.

SEE ALSO

getopt(3PVM)

NAME

pvm_setrbuf - Switches the active receive buffer and saves the previous buffer.

SYNOPSIS

C int oldbuf = pvm_setrbuf(int bufid)

Fortran call pvmfsetrbuf(bufid, oldbuf)

PARAMETERS

bufid Integer specifying the message buffer identifier for the new active receive buffer.

oldbuf Integer returning the message buffer identifier for the previous active receive buffer.

DISCUSSION

The routine pvm_setrbuf switches the active receive buffer to bufid and saves the previous active receive buffer oldbuf. If bufid is set to 0 then the present active receive buffer is saved and no active receive buffer exists.

A successful receive automatically creates a new active receive buffer. If a previous receive has not been unpacked and needs to be saved for later, then the previous bufid can be saved and reset later to the active buffer for unpacking.

The routine is required when managing multiple message buffers. For example switching back and forth between two buffers. One buffer could be used to send information to a graphical interface while a second buffer could be used send data to other tasks in the application.

EXAMPLES

C:

```
rbuf1 = pvm_setrbuf( rbuf2 );
```

Fortran:

```
CALL PVMFSETRBUF( NEWBUF, OLDBUF )
```

ERRORS

These error conditions can be returned by pvm_setrbuf

PvmBadParam giving an invalid bufid.
PvmNoSuchBuf switching to a non-existent message buffer.

SEE ALSO

setsbuf(3PVM)

NAME

pvm_setsbuf - Switches the active send buffer.

SYNOPSIS

C int oldbuf = pvm_setsbuf(int bufid)

Fortran call pvmfsetsbuf(bufid, oldbuf)

PARAMETERS

bufid Integer the message buffer identifier for the new active send buffer. A value of 0 indicates the default receive buffer.

oldbuf Integer returning the message buffer identifier for the previous active send buffer.

DISCUSSION

The routine pvm_setsbuf switches the active send buffer to bufid and saves the previous active send buffer oldbuf. If bufid is set to 0 then the present active send buffer is saved and no active send buffer exists.

The routine is required when managing multiple message buffers. For example switching back and forth between two buffers. One buffer could be used to send information to a graphical interface while a second buffer could be used send data to other tasks in the application.

EXAMPLES

C:
```
     sbuf1 = pvm_setsbuf( sbuf2 );
```
Fortran:
```
     CALL PVMFSETSBUF( NEWBUF, OLDBUF )
```

ERRORS

These error conditions can be returned by pvm_setsbuf

PvmBadParam giving an invalid bufid.
PvmNoSuchBuf switching to a non-existent message buffer.

SEE ALSO

setrbuf(3PVM)

NAME

pvm_gettmask, pvm_settmask - Get or set trace mask of a task or its children.

SYNOPSIS

```
C    #include <pvm3.h>      #include <pvmtev.h>

     int info = pvm_gettmask( int who, Pvmtmask mask )
     int info = pvm_settmask( int who, Pvmtmask mask )

Fortran   Not Available
```

PARAMETERS

who Specifies which trace mask to get or set:

PvmTaskSelf	The current task
PvmTaskChild	Children tasks to be spawned

mask Trace mask (owned by caller), which is filled in with the current trace vector on calling pvm_gettmask or used to set the current trace vector on calling pvm_settmask.

info Status code returned. Values less than zero indicate an error.

DISCUSSION

Each task has a trace mask, which can be used to individually enable tracing of each libpvm function. The mask is inherited from its parent task (or initially all cleared if the task has no parent). When calls are made to functions in libpvm, the parameters and results are sent in messages to the trace sink of the task (variable PvmTraceTid set by calling pvm_setopt - see man page).

The PVM console and XPVM use this system to get trace data from application programs.

A task can get or set either its own trace mask, or the mask passed to children it spawns. Note the mask must be set before the spawn call; setting the child trace mask has no effect on already-running children.

The trace mask can be kept in a temporary variable, declared as:

```
Pvmtmask mymask;
```

Right now, this is a character array with approximately 80 bits.

There are four macros defined to manipulate trace masks. Along with the functions, these allow a program to get the current trace mask, modify or save it, and put it back:

```
TEV_INIT_MASK(mask)
TEV_SET_MASK(mask, kind)
TEV_UNSET_MASK(mask, kind)
TEV_CHECK_MASK(mask, kind)
```

TEV_INIT_MASK initializes a mask to all bits cleared and must be used to initialize a trace mask (unless that is done by setting it from pvm_gettmask). TEV_SET_MASK sets a bit in a mask. Bit values are defined in <pvmtev.h>. Likewise, TEV_UNSET_MASK clears a single bit and TEV_CHECK_MASK checks to see if a bit is set, returning nonzero if true.

Note: Most events in the trace mask come in pairs, the first (ending in ...0) is generated on entry to a function and the second (...1) on exit. The trace mask has only half as many bits as there are events; events are enabled in pairs. In other words, if you select TEV_SEND0, you'll also get TEV_SEND1.

The trace mask is deliberately kept as a printable (and null-terminated) string, for two reasons. First, it is passed through the environment between tasks, in the form:

```
PVMTMASK=@@AAOG@@NO@L@@@@@@@@
```

so the pvmd doesn't have to explicitly manage it. Second, the mask variables can be passed between tasks using pvm_pkstr() or otherwise manipulated as strings.

pvm_gettmask and pvm_settmask return PvmOk on success, else a negative value.

EXAMPLES

C:

```
#include <pvm3.h>
#include <pvmtev.h>

Pvmtmask m;

pvm_gettmask( PvmTaskChild, m );
TEV_SET_MASK(m, TEV_ADDHOSTS0 );
pvm_settmask( PvmTaskChild, m );
pvm_spawn( "worker", (char **)0, PvmTaskDefault, "", 1, (int *)0);
```

ERRORS

The following error condition can be returned by pvm_gettmask or pvm_settmask:

PvmBadParam invalid value for who argument.

SEE ALSO

pvm(1PVM), pvm_getopt(3PVM), pvm_setopt(3PVM)

NAME

pvm_spawn - Starts new PVM processes.

SYNOPSIS

```
C    int numt = pvm_spawn( char *task, char **argv, int
flag, char *where, int ntask, int *tids )

Fortran   call pvmfspawn( task, flag, where, ntask, tids,
numt )
```

PARAMETERS

task Character string which is the executable file name of the PVM process to be started. The executable must already reside on the host on which it is to be started. The name may be a file in the PVM search path or an absolute path. The default PVM search path is $HOME/pvm3/bin/$PVM_ARCH/ .

argv Pointer to an array of arguments to the executable (if supported on the target machine), not including the executable name, with the end of the array specified by NULL. argv[0] of the spawned task is set to the executable path relative to the PVM working directory (or absolute if an absolute filename was specified). If the executable needs no arguments, then the second argument to pvm_spawn is NULL.

flag Integer specifying spawn options.

In C, flag should be the sum of:

Option	value	MEANING
PvmTaskDefault	0	PVM can choose any machine to start task
PvmTaskHost	1	where specifies a particular host
PvmTaskArch	2	where specifies a type of architecture
PvmTaskDebug	4	Start up processes under debugger
PvmTaskTrace	8	Processes will generate PVM trace data. *
PvmMppFront	16	Start process on MPP front-end.
PvmHostCompl	32	Use complement host set

In Fortran, flag should be the sum of:

Option	value	MEANING
PVMDEFAULT	0	PVM can choose any machine to start task
PVMHOST	1	where specifies a particular host
PVMARCH	2	where specifies a type of architecture
PVMDEBUG	4	Start up processes under debugger
PVMTRACE	8	Processes will generate PVM trace data. *

* future extension

where — Character string specifying where to start the PVM process. Depending on the value of flag, where can be a host name such as ``ibm1.epm.ornl.gov'' or a PVM architecture class such as ``SUN4''. If flag is 0, then where is ignored and PVM will select the most appropriate host.

ntask — Integer specifying the number of copies of the executable to start.

tids — Integer array of length ntask returning the tids of the PVM processes started by this pvm_spawn call.

numt — Integer returning the actual number of tasks started. Values less than zero indicate a system error. A positive value less than ntask indicates a partial failure. In this case the user should check the tids array for the error code(s).

DISCUSSION

The routine pvm_spawn starts ntask copies of the executable named task. On systems that support environment, spawn passes selected variables from parent environment to children tasks. If set, the envar PVM_EXPORT is passed. If PVM_EXPORT contains other names (separated by ':') they will be passed too. This is useful for e.g.:

```
setenv DISPLAY myworkstation:0.0
setenv MYSTERYVAR 13
setenv PVM_EXPORT DISPLAY:MYSTERYVAR
```

The hosts on which the PVM processes are started are determined by the flag and where arguments. On return the array tids contains the PVM task identifiers for each process started.

If pvm_spawn starts one or more tasks, numt will be the actual number of tasks started. If a system error occurs then numt will be < 0. If numt is less than ntask then some executables have failed to start and the user should check the last ntask - numt locations in the tids array which will contain error codes (see below for meaning). The first numt tids in the array are always valid.

When flag is set to 0 and where is set to NULL (or ``*'' in Fortran) a heuristic (round-robin assignment) is used to distribute the ntask processes across the virtual machine.

If the PvmHostCompl flag is set, the resulting host set gets complemented. Also, the TaskHost hostname "." is taken as localhost. These can be used, for example, to spawn n - 1 tasks on host "." with flags TaskHost|HostCompl to get all but the localhost.

In the special case where a multiprocessor is specified by where, pvm_spawn will start all ntask copies on this single machine using the vendor's underlying routines.

The spawned task will have argv[0] set to the the executable path relative to its inherited working directory (or possibly an absolute path), so the base filename can be got by using:

```
char *p;
p = (p = rindex(argv[0], '/')) ? p + 1 : argv[0];
```

If PvmTaskDebug is set, then the pvmd will start the task(s) under debugger(s). In this case, instead of executing pvm3/bin/ARCH/task args it executes pvm3/lib/debugger pvm3/bin/ARCH/task args. debugger is a shell script that the users can modify to their individual tastes. Presently the script starts an xterm with dbx or comparable debugger in it.

EXAMPLES

C:

```
numt = pvm_spawn( "host", 0, PvmTaskHost, "sparky", 1, &tid[0] );
numt = pvm_spawn( "host", 0, (PvmTaskHost+PvmTaskDebug),
               "sparky", 1, &tid[0] );
numt = pvm_spawn( "node", 0, PvmTaskArch, "RIOS", 1, &tid[i] );

char *args[] = { "12", "60" , (char*)0 };
numt = pvm_spawn( "FEM1", args, 0, 0, 16, tids );

numt = pvm_spawn( "pde", 0, PvmTaskHost, "paragon.ornl", 512, tids );
```

Fortran:

```
FLAG = PVMARCH + PVMDEBUG
CALL PVMFSPAWN( 'node', FLAG, 'SUN4', 1, TID(3), NUMT )
CALL PVMFSPAWN( 'FEM1', PVMDEFAULT, '*', 16, TIDS, NUMT )
CALL PVMFSPAWN( 'TBMD', PVMHOST, 'cm5.utk.edu', 32, TIDS, NUMT )
```

ERRORS

These error conditions can be returned by pvm_spawn either in numt or in the tids array.

PvmBadParam	giving an invalid argument value.
PvmNoHost	Specified host is not in the virtual machine.
PvmNoFile	Specified executable cannot be found. The default location PVM looks in is ~/pvm3/bin/ARCH, where ARCH is a PVM architecture name.
PvmNoMem	Malloc failed. Not enough memory on host.

PvmSysErr	pvmd not responding.
PvmOutOfRes	out of resources.

NAME
     pvm_start_pvmd - Starts new PVM daemon.

SYNOPSIS
     C   int info = pvm_start_pvmd(int argc, char **argv, int
     block)

     Fortran   NOT AVAILABLE

PARAMETERS
     argc   Number of arguments in argv.

     argv   An array of arguments to the executable.

     block   Integer specifying whether to block until startup of
             all hosts complete or return immediately.

     info   Integer returning the error code.

DISCUSSION
     The routine pvm_start_pvmd starts up a pvmd3 process, the
     master of a new virtual machine. The routine can be set to
     block until startup is complete or to return immediately.
     pvm_start_pvmd returns zero on success.

ERRORS
     The following error conditions can be returned
           PvmDupHost   a pvmd is already running.
           PvmSysErr   local pvmd is not responding.

SEE ALSO
     addhost(3PVM)

NAME

pvm_tasks - Returns information about the tasks running on the virtual machine.

SYNOPSIS

```
C    int info = pvm_tasks( int  where,  int  *ntask,  struct
taskinfo **taskp )

     struct taskinfo {      int  ti_tid;       int  ti_ptid;
     int  ti_host;       int  ti_flag;       char *ti_a_out;
     } taskp;


Fortran   call pvmftasks( where,  ntask,  tid,  ptid,  dtid,
flag, aout, info )
```

PARAMETERS

where   Integer specifying what tasks to return information about. The options are:

0	for all the tasks on the virtual machine
pvmd tid	for all tasks on a given host
tid	for a specific task

ntask   Integer returning the number of tasks being reported on.

taskp   Pointer to an array of structures which contain information about each task including its task ID, parent tid, pvmd task ID, status flag, and the name of this task's executable file. The status flag values are: waiting for a message, waiting for the pvmd, and running.

tid   Integer returning task ID of one task

ptid   Integer returning parent task ID

dtid   Integer returning pvmd task ID of host task is on.

flag   Integer returning status of task

aout   Character string returning the name of spawned task. Manually started tasks return blank.

info   Integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION

The routine pvm_tasks returns information about tasks presently running on the virtual machine. The information returned is the same as that available from the console command ps. The C function returns information about the entire virtual machine in one call. The Fortran function returns information about one task per call and cycles through all the tasks. Thus, if where = 0, and pvmftasks is called ntask times, all tasks will be represented. If pvm_tasks is successful, info will be 0. If some error occurs then info will be < 0.

EXAMPLES

C:

```
info = pvm_tasks( 0, &ntask, &taskp );
```

Fortran:

```
Do i=1, NTASK
    CALL PVMFTASKS( DTID, NTASK, TID(i), PTID(i), DTID(i),
&                  FLAG(i), AOUT(i), INFO )
EndDo
```

ERRORS

The following error condition can be returned by pvm_tasks

PvmBadParam	invalid value for where argument.
PvmSysErr	pvmd not responding.
PvmNoHost	specified host not in virtual machine.

SEE ALSO

config(3PVM) and tidtohost(3PVM)

NAME

pvm_tidtohost - Returns the host of the specified PVM process.

SYNOPSIS

C int dtid = pvm_tidtohost(tid)

Fortran NOT AVAILABLE

PARAMETERS

tid Integer task identifier of the PVM process in question.

dtid Integer returns the tid of the host's pvmd3 or a negative value if an error.

DISCUSSION

The routine pvm_tidtohost returns the host id on which the process identified by tid is located.

ERRORS

The following error conditions can be returned

PvmBadParam	Bad Parameter most likely an invalid tid value.
PvmSysErr	pvmd not responding.

SEE ALSO

config(3PVM) and tasks(3PVM)

NAME

pvm_trecv - Receive with timeout.

SYNOPSIS

```
C    #include <sys/time.h>
     int bufid = pvm_trecv( int tid, int msgtag, struct timeval *tmout )

Fortran   call pvmftrecv( tid, msgtag, sec, usec, bufid )
```

PARAMETERS

tid Integer to match task identifier of sending process.

msgtag Integer to match message tag; should be >= 0.

tmout (or sec and usec) Time to wait before returning without a message.

bufid Integer returns the value of the new active receive buffer identifier. Values less than zero indicate an error.

DISCUSSION

The routine pvm_trecv blocks the process until a message with label msgtag has arrived from tid. pvm_trecv then places the message in a new active receive buffer, also clearing the current receive buffer. If no matching message arrives within the specified waiting time, pvm_trecv returns without a message.

A -1 in msgtag or tid matches anything. This allows the user the following options. If tid = -1 then pvm_trecv will accept a message from any process which has a matching msgtag. If msgtag = -1 then pvm_trecv will accept any message that is sent from process tid. If tid and msgtag are both -1, then pvm_trecv will accept any message from any process.

In C, the tmout fields tv_sec and tv_usec specify how long pvm_trecv will wait without returning a matching message. In Fortran, two separate parameters, sec and usec are passed. With both set to zero, pvm_trecv behaves the same as pvm_nrecv, which is to probe for messages and return immediately even if none are matched. In C, passing a null pointer in tmout makes pvm_trecv act like pvm_recv, that is, it will wait indefinitely. In Fortran, setting sec to -1 has the same effect.

The PVM model guarantees the following about message order. If task 1 sends message A to task 2, then task 1 sends message B to task 2, message A will arrive at task 2 before message B. Moreover, if both messages arrive before task 2 does a receive, then a wildcard receive will always return message A.

If pvm_trecv is successful, bufid will be the new active receive buffer identifier. If no message is received, pvm_trecv returns 0. If some error occurs then bufid will be < 0.

Once pvm_trecv returns, the data in the message can be unpacked into the user's memory using the unpack routines.

EXAMPLES

C:

```
struct timeval tmout;

tid = pvm_parent();
msgtag = 4 ;
tmout.tv_sec = 60;
tmout.tv_usec = 0;
if ((bufid = pvm_trecv( tid, msgtag, &tmout )) > 0) {
     pvm_upkint( tid_array, 10, 1 );
     pvm_upkint( problem_size, 1, 1 );
     pvm_upkfloat( input_array, 100, 1 );
}
```

Fortran:

```
     CALL PVMFTRECV( -1, 4, 60, 0, BUFID )
     IF (BUFID .EQ. 0) GO TO 666
     CALL PVMFUNPACK( INTEGER4, TIDS, 25, 1, INFO )
     CALL PVMFUNPACK( REAL8, MATRIX, 100, 100, INFO )
666  CONTINUE
```

ERRORS

These error conditions can be returned by pvm_trecv

PvmBadParam	giving an invalid tid value, or msgtag < -1.
PvmSysErr	pvmd not responding.

SEE ALSO

pvm_nrecv(3PVM), pvm_recv(3PVM), pvm_unpack(3PVM), pvm_probe(3PVM), pvm_send(3PVM), pvm_mcast(3PVM)

NAME

pvm_unpack - Unpack the active message buffer into arrays of prescribed data type.

SYNOPSIS

C

```
int info = pvm_unpackf( const char *fmt, ... )
int info = pvm_upkbyte( char *xp, int nitem, int stride)
int info = pvm_upkcplx( float *cp, int nitem, int stride)
int info = pvm_upkdcplx( double *zp, int nitem, int stride)
int info = pvm_upkdouble( double *dp, int nitem, int stride)
int info = pvm_upkfloat( float *fp, int nitem, int stride)
int info = pvm_upkint( int *ip, int nitem, int stride)
int info = pvm_upkuint( unsigned int *ip, int nitem, int stride )
int info = pvm_upkushort( unsigned short *ip, int nitem, int stride )
int info = pvm_upkulong( unsigned long *ip, int nitem, int stride )
int info = pvm_upklong( long *ip, int nitem, int stride)
int info = pvm_upkshort( short *jp, int nitem, int stride)
int info = pvm_upkstr( char *sp )
```

Fortran

```
call pvmfunpack( what, xp, nitem, stride, info )
```

PARAMETERS

fmt Printf-like format expression specifying what to pack. (See discussion)

nitem The total number of items to be unpacked (not the number of bytes).

stride The stride to be used when packing the items. For example, if stride = 2 in pvm_upkcplx, then every other complex number will be unpacked.

xp Pointer to the beginning of a block of bytes. Can be any data type, but must match the corresponding pack data type.

cp Complex array at least nitem*stride items long.

zp Double precision complex array at least nitem*stride items long.

dp Double precision real array at least nitem*stride items long.

fp Real array at least nitem*stride items long.

ip Integer array at least nitem*stride items long.

jp Integer*2 array at least nitem*stride items long.

sp Pointer to a null terminated character string.

what Integer specifying the type of data being unpacked.

	what	options	
STRING	0	REAL4	4
BYTE1	1	COMPLEX8	5
INTEGER2	2	REAL8	6
INTEGER4	3	COMPLEX16	7

info Integer status code returned by the routine. Values less than zero indicate an error.

DISCUSSION

Each of the pvm_upk* routines unpacks an array of the given data type from the active receive buffer. The arguments for each of the routines are a pointer to the array to be unpacked into, nitem which is the total number of items to unpack, and stride which is the stride to use when unpacking.

An exception is pvm_upkstr() which by definition unpacks a NULL terminated character string and thus does not need nitem or stride arguments. The Fortran routine pvmfunpack(STRING, ...) expects nitem to be the number of characters in the string and stride to be 1.

If the unpacking is successful, info will be 0. If some error occurs then info will be < 0.

A single variable (not an array) can be unpacked by setting nitem = 1 and stride = 1.

The routine pvm_unpackf() uses a printf-like format expression to specify what and how to unpack data from the receive buffer. All variables are passed as addresses. A BNF-like description of the format syntax is:

```
    format : null | init | format fmt
```

```
init : null | '%' '+'
fmt : '%' count stride modifiers fchar
fchar : 'c' | 'd' | 'f' | 'x' | 's'
count : null | [0-9]+ | '*'
stride : null | '.' ( [0-9]+ | '*' )
modifiers : null | modifiers mchar
mchar : 'h' | 'l' | 'u'
```

```
Formats:
  +  means initsend - must match an int (how) in the param list.
  c  pack/unpack bytes
  d  integer
  f  float
  x  complex float
  s  string

Modifiers:
  h  short (int)
  l  long  (int, float, complex float)
  u  unsigned (int)
```

Future extensions to the what argument in pvmfunpack will include 64 bit types when XDR encoding of these types is available. Meanwhile users should be aware that precision can be lost when passing data from a 64 bit machine like a Cray to a 32 bit machine like a SPARCstation. As a mnemonic the what argument name includes the number of bytes of precision to expect. By setting encoding to PVMRAW (see pvmfinitsend) data can be transferred between two 64 bit machines with full precision even if the PVM configuration is heterogeneous.

Messages should be unpacked exactly like they were packed to insure data integrity. Packing integers and unpacking them as floats will often fail because a type encoding will have occurred transferring the data between heterogeneous hosts. Packing 10 integers and 100 floats then trying to unpack only 3 integers and the 100 floats will also fail.

EXAMPLES

C:

```
info = pvm_recv( tid, msgtag );
info = pvm_upkstr( string );
info = pvm_upkint( &size, 1, 1 );
info = pvm_upkint( array, size, 1 );
info = pvm_upkdouble( matrix, size*size, 1 );

int count, *iarry;
double darry[4];
```

```
pvm_unpackf("%d", &count);
pvm_unpackf("%*d %4lf", count, iarry, darry);
```

Fortran:

```
CALL PVMFRECV( TID, MSGTAG );
CALL PVMFUNPACK( INTEGER4, NSIZE, 1, 1, INFO )
CALL PVMFUNPACK( STRING, STEPNAME, 8, 1, INFO )
CALL PVMFUNPACK( REAL4, A(5,1), NSIZE, NSIZE , INFO )
```

ERRORS

PvmNoData
: Reading beyond the end of the receive buffer. Most likely cause is trying to unpack more items than were originally packed into the buffer.

PvmBadMsg
: The received message can not be decoded. Most likely because the hosts are heterogeneous and the user specified an incompatible encoding. Try setting the encoding to PvmDataDefault (see pvm_mkbuf).

PvmNoBuf
: There is no active receive buffer to unpack.

SEE ALSO

pack(3PVM)