PVM List of Frequently Asked Questions v1.6 June 23, 1995

Contents

1 General **1**
1.1 What is PVM? 1
1.2 The history of PVM 1
1.3 What machines does PVM run on? 1
1.4 What languages does it work with? 2
1.5 Is there a book on PVM? 2
1.6 Is there a list of published material related to PVM? 2
1.7 What support exists for PVM? 2
1.8 How do I contact the authors? 2
1.9 Is there a server for contributed software? 3
1.10 Who is using PVM? 3
1.11 Does PVM version 2 still exist? 3
1.12 Are you working on version 4? 3
1.13 What is XPVM? 3
1.14 What are ... (related topics and add-on systems) 4
1.14.1 What is HeNCE? 4
1.14.2 What is pvmfs? 4
1.14.3 What is tpvm? 5
1.14.4 What is PIOUS? 5
1.15 How do I get files from Netlib? 5

2 Installation and the System **6**
2.1 How do I get PVM? 6
2.2 How do I get documentation? 6
2.3 How do I get updates to the software? 6
2.4 What is the most recent version of PVM? 6
2.5 What versions of PVM are interoperable? 7
2.6 What do I have to type to install PVM? 7
2.7 I get errors when building the PVM release 7
2.8 What does `PVM_ROOT` do? 8
2.9 What are "machine architectures" 8
2.10 How do I apply patches 8
2.11 What should I tell my new users about using PVM? 9

2.12 Machines at my site share a single `/tmp` directory 9
2.13 Can I have two different versions of PVM on the same machine 9
2.14 How do I start PVM? 9
2.15 What are these files in `/tmp`? 10
2.16 What do I call the host file? 10
2.17 Hostfile defaults aren't getting set 11
2.18 Hostfile options aren't taking effect 11
2.19 What does "speed" in the machine configuration mean? . . . 11
2.20 How do I read the on-line manual pages? 12
2.21 How do I shut down PVM? 12
2.22 Does PVM use shared memory? 12

3 Common error messages 13
3.1 Error messages printed by libpvm 13
3.2 Error messages in the pvmd log file 13
3.3 `Can't start pvmd` 13
3.4 `Can't contact local daemon` 14
3.5 `Version mismatch` 14
3.6 `No such host` 15
3.7 `Duplicate host` 15
3.8 `libpvm [t40001]: mxfer() EOF on pvmd sock` 15
3.9 `No such file` 15
3.10 `failed to start group server` 15

4 Programming 16
4.1 Writing Programs 16
4.1.1 What is aimk? 16
4.1.2 How can I build my programs for different architectures? 16
4.1.3 Is message order between tasks preserved? 16
4.1.4 How can I pack a struct in a message? 16
4.1.5 The compiler can't find the PVM header files 18
4.1.6 What libraries do I need to link with? 19
4.1.7 Missing symbols when I try to link my application . . 19
4.2 Running and debugging programs 19
4.2.1 The sender doesn't get a copy of multicast messages . 19
4.2.2 My tasks can't read input from the keyboard 20
4.2.3 My tasks can't open input/output files 20
4.2.4 Where does stuff printed in tasks go? 20

4.2.5 I don't see printed output from tasks right away or at all . . . 21
4.2.6 In what directory do tasks spawned by PVM run? . . 21
4.2.7 How can I spawn a process with a nice priority value? 21
4.2.8 What does "mc pkt ... scrapped" mean? . . . 22
4.2.9 What does "message scrapped" mean? . . . 22
4.2.10 Memory used by the pvmd increases when programs run . . . 22
4.2.11 My program is hanging . . . 23

5 Advanced programming **23**

5.1 How can I get input from PVM messages and e.g. X? 23
5.2 I'm not getting notify messages . . . 24
5.3 My application needs to start PVM itself . . . 24
5.4 Can I spawn shell scripts . . . 24
5.5 Can a task fork and the child be a task? . . . 24
5.6 Can I use signals in a PVM program? . . . 25
5.7 What signals are safe to use in a PVM program? . . . 26
5.8 I need to get machine loads to write a load balancer . . . 26
5.9 How can I rewind a message buffer to unpack it again? 26

1 General

1.1 What is PVM?

PVM (Parallel Virtual Machine) is a portable message-passing programming system, designed to link separate host machines to form a "virtual machine" which is a single, manageable computing resource.

The virtual machine can be composed of hosts of varying types, in physically remote locations. PVM applications can be composed of any number of separate processes, or components, written in a mixture of C, C++ and Fortran. The system is portable to a wide variety of architectures, including workstations, multiprocessors, supercomputers and PCs.

PVM is a byproduct of ongoing research at several institutions, and is made available to the public free of charge.

1.2 The history of PVM

The first version of PVM was written during Summer of 1989 at Oak Ridge National Laboratory. It was not released, but was used in applications at the lab.

Version 2 was written from scratch during February 1991 at the University of Tennessee, Knoxville, and released in March of that year. It was intended to clean up and stablize the system so that external users could benefit.

After a year and a half of continuing research we felt that we had reached a limit in being able to add to PVM version 2. Version 3 was redesigned from scratch, and a complete rewrite started in September 1992, with first release of the software in March 1993. While similar in spirit to version 2, version 3 includes features that didn't fit the old framework – most importantly fault tolerance, better portability and scalability.

1.3 What machines does PVM run on?

The most common PVM platform is a Unix machine, and PVM currently runs on more than 30 different ones. Chances are, yours is included. Porting it to other ones is generally a simple matter. A complete list of predefined architectures can be found in the PVM source in file `doc/arches`.

Special ports of PVM for some message-passing processors are included with the Netlib distribution. These include the Intel iPSC/860, iPSC/2 and Paragon and TMC CM-5. Shared memory ports are also included that use

System-5 shared memory for communication and run on Sun, SGI, DEC, Kendall Square, Sequent and IBM machines.

Ports exist aside from the Netlib distribution for the Cray T-3D (a CRI product), Parsytech (♠♡♣♢ I don't have a reference for this one), MEIKO Computing Surface (♠♡♣♢ or this) and other machines. Ports have also been done to VMS (by Dan Clark, `dclark@cse.ogi.edu`) and OS/2 (by Jan Ftacnik, `ftacnikj@fnal05.fnal.gov`).

1.4 What languages does it work with?

PVM is independent of any particular language (it is written in C). Programs can be written in one or more languages. Libraries for C, C++ and Fortran are provided with the distribution. It has been used with other languages, such as Lisp.

1.5 Is there a book on PVM?

The book, *PVM: Parallel Virtual Machine – A Users' Guide and Tutorial for Networked Parallel Computing*, by Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam is available from MIT Press (1994). It is also available in Postscript from netlib and in html at `http://www.netlib.org/pvm3/book/pvm-book.html`.

1.6 Is there a list of published material related to PVM?

There is a pretty good list in the ORNL PVM Mosaic home page:

`http://www.epm.ornl.gov/pvm/pvm_home.html`

1.7 What support exists for PVM?

PVM is free software, generated as a byproduct of research; so, bugs get fixed on an as-time-allows basis. The developers are currently exploring more formal means of providing support. Also, we urge you to read and post to newsgroup `comp.parallel.pvm`.

1.8 How do I contact the authors?

You can send e-mail to `pvm@msr.epm.ornl.gov`. This alias is monitored by the authors.

1.9 Is there a server for contributed software?

Yes, there is a directory on netlib for contributed software related to PVM. Send a note to the PVM mail alias if you have something to contribute.

1.10 Who is using PVM?

There are hundreds or thousands of PVM users worldwide, in academia and industry. Applications range from teaching parallel programming to experimental and production systems.

Surveys of who is using PVM and for what were taken in 1992 and 1993. For a copy of the results (these are quite large), send a request to the PVM alias.

1.11 Does PVM version 2 still exist?

Yes, you can still get the code for PVM version 2.4.2 (December 1992), though we are no longer working on it and support will be very minimal. We are currently working on version 3, which evolved (§1.2) out of version 2.

1.12 Are you working on version 4?

Sort of, but don't hold your breath. There are things we'd like to add to version 3 that make us want to tear it up and start over, but there's still plenty of work to be done on the current version.

1.13 What is XPVM?

XPVM is a graphical console and monitor for PVM. It provides a graphical interface to the PVM console commands and information, along with several animated views to monitor the execution of PVM programs. These views provide information about the interactions among tasks in a parallel PVM program, to assist in debugging and performance tuning.

To analyze a program using XPVM, a user need only compile their program using the PVM library, version 3.3 or later (which has been instrumented to capture tracing information at run-time). Then, any task spawned from XPVM will return trace event information, for analysis in real time, or for post-mortem playback from saved trace files.

1.14 What are ... (related topics and add-on systems)

1.14.1 What is HeNCE?

HeNCE (Heterogeneous Network Computing Environment) is an X-window based software environment designed to assist scientists in developing parallel programs that run on a network of computers. HeNCE provides the programmer with a high level abstraction for specifying parallelism. HeNCE is based on a parallel programming paradigm where an application program can be described by a graph. Nodes of the graph represent subroutines and the arcs represent data dependencies. Individual nodes are executed under PVM. HeNCE is composed of integrated graphical tools for creating, compiling, executing, and analyzing HeNCE programs. HeNCE relies on the PVM system for process initialization and communication. The HeNCE programmer, however, will never explicitly write PVM code. During or after execution, HeNCE displays an event-ordered animation of application execution, enabling the visualization of relative computational speeds, processor utilization, and load imbalances.

Through HeNCE a scientist can easily decompose existing C or Fortran source code into pieces which can be executed in parallel over an existing collection of workstations or supercomputers. This approach allows the programmer to reuse existing programs and extract unused performance out of existing computers.

HeNCE also supports the graphical configuration of PVM hosts, assists in the generation of architecture-dependent object modules, and contains provisions for task scheduling based on user supplied cost matrices.

HeNCE is available from netlib via anonymous FTP: Use host `netlib2.cs.utk.edu`, login as `anonymous`, change directory to `/hence`. For more information about HeNCE, send e-mail to `hence@cs.utk.edu`.

1.14.2 What is pvmfs?

PVMFS is a PVM task process which facilitates I/O for programs utilizing PVM on a cluster of workstations. The file server presents a common file system and support for coordinated I/O to its parent and siblings.

It works by grabbing I/O syscalls such as `read()` and `open()`, and sending the requests to file server via PVM messages. Therefore, it works with unmodified source code, *stdio*, etc. It was written by Christopher G. Phillips (`pefv700@pe.utexas.edu`). He is no longer supporting it, but it is still available on netlib, as a patch to version 3.3.4.

1.14.3 What is tpvm?

TPVM is a threads-based subsystem for PVM. For more information, contact Vaidy Sunderam, `vss@mathcs.emory.edu`. See also `ftp://mathcs.emory.edu/pub/vss`.

1.14.4 What is PIOUS?

PIOUS is a parallel file system architecture that provides cost-effective, scalable bandwidth in a network computing environment. PIOUS employs data declustering, to exploit the combined file I/O and buffer cache capacities of networked computing resources, and transaction-based concurrency control, to guarantee access consistency without explicit synchronization.

For more informatiom, contact Vaidy Sunderam, `vss@mathcs.emory.edu`. See also `ftp://mathcs.emory.edu/pub/vss`.

1.15 How do I get files from Netlib?

You can retrieve files from netlib in many ways, including e-mail, ftp, xnetlib, mosaic and rcp.

If you know the *package* containing the software you want, here's how to get the index file for it:

How	What
e-mail	`echo "send index from` *package*`" | mail netlib@ornl.gov`
ftp	host `netlib2.cs.utk.edu` login as `anonymous` `cd` *package* `get index`
xnetlib	select directory *package*
mosaic	`http://netlib2.cs.utk.edu/`*package*`/index.html`
rcp	`rcp anon@netlib2.cs.utk.edu:`*package*`/index index`

For more information about file retrieval from netlib,
`finger anon@netlib2.cs.utk.edu`.
For more information about Xnetlib,
`echo "send index from xnetlib" | mail netlib@ornl.gov`

2 Installation and the System

2.1 How do I get PVM?

The primary server for PVM is `netlib.ornl.gov`. In the PVM directory (`pvm3`) are sources, patches, documentation, contributed software and other miscellaneous files.

From netlib, you can get PVM files via e-mail, ftp, xnetlib, mosaic or rcp. Other sites may have different facilities.

Via e-mail:

```
echo "send index from pvm3" | mail netlib@ornl.gov
```

Using Xnetlib, select directory `pvm3`.

Via FTP: Use host `netlib2.cs.utk.edu`, login as `anonymous`, change directory to `/pvm3`.

The Mosaic URL is: `http://netlib2.cs.utk.edu/pvm3/index.html`

For more information about netlib, see §1.15.

2.2 How do I get documentation?

I'd reccommend getting the book (§1.5).

A user guide, kept up-to-date with the current software release, is also available on netlib (§2.1).

On-line manual pages (§2.20) are included with the source distribution.

Slides from recent tutorials can also be found on netlib.

2.3 How do I get updates to the software?

There is no automatic update system, or individual notification. A major version change (e.g. 3.2 to 3.3) seems to happen about once or twice a year. Between those times, patches are released as needed, usually every few months.

When a new distribution or patch becomes available, an article is posted to `comp.parallel.pvm`.

2.4 What is the most recent version of PVM?

The most recent version is always on `netlib.ornl.gov`. See §2.1.

2.5 What versions of PVM are interoperable?

The protocols used in building PVM are evolving, with the result that binaries from newer releases are not compatible with older ones. Compatibility is determined by the pvmd-task and task-task protocol revision numbers. These are compared when two PVM entities connect; they will refuse to interoperate if the numbers don't match.

PVM releases with the same second digit in their version numbers (for example 3.2.0 and 3.2.6) will interoperate. Changes that result in incompatibility are held until a major version change (for example, from 3.2 to 3.3).

2.6 What do I have to type to install PVM?

The following commands download, unpack, build and install a release:

start in directory just above PVM root,
for example `$HOME` *or* `/usr/local`

```
% ftp netlib2.cs.utk.edu
Name:  anonymous
Password:  your id, user@host.domain
ftp> cd pvm3
ftp> bin
ftp> get pvm3.3.tar.z.uu  (or the most recent version)
ftp> quit
% uudecode pvm3.3.tar.z.uu
% zcat pvm3.3.tar.Z | tar xf -
% cd pvm3
% setenv PVM_ROOT $cwd
% make
```

2.7 I get errors when building the PVM release

The compiler may print a few warning messages; we suggest you ignore these unless the build doesn't complete or until you have some other reason to think there is a problem. If you can't build the unmodified distribution "out of the box" on a supported architecture, let us know.

2.8 What does PVM_ROOT do?

The environment variable `PVM_ROOT` tells PVM where the system is installed, so one part can find another. It is the only thing that needs to be set in order to be able to run PVM. It must also be set when building the PVM installation from the source distribution.

If you normally use `csh`, add a line like this to your `.cshrc` file:

```
setenv PVM_ROOT $HOME/pvm3
```

If you normally use `sh`, add these lines to your `.profile`:

```
PVM_ROOT=$HOME/pvm3
PVM_DPATH=$HOME/pvm3/lib/pvmd
export PVM_ROOT PVM_DPATH
```

Make sure these are set in your current session too.

Older versions of PVM assumed an installation path of `$HOME/pvm3`. Versions 3.3 and later require that the `PVM_ROOT` variable always be set. Note: For compatibility with older versions of PVM and some command shells that don't execute a startup file when used with `rsh`, `PVM_ROOT` defaults to `$HOME/pvm3` if it's not set.

2.9 What are "machine architectures"

PVM assigns an *architecture name* to each kind of machine on which it runs, to distinguish between machines that run different executables.

This allows PVM to select the correct executable at run time. It also allows builds of the system or applications to proceed in parallel (in separate directories named after the architecture).

PVM also assigns a *data encoding* to each machine; One or more architectures map to each data encoding. Machines with different architectures, but the same data encoding must run different executables, but can pass data back and forth without converting it.

2.10 How do I apply patches

Get the patches from netlib. By convention, they go in `$PVM_ROOT/patches` and are named, e.g., `pvm334to5` (converts version 3.3.4 to 3.3.5). To install them, you'll need to have the `patch` command (free software) installed on your system.

To install a patch, change to the `$PVM_ROOT` directory and type e.g.: `patch -p0 < patches/pvm334to5`. Check the file `patches/README` for any special instructions.

2.11 What should I tell my new users about using PVM?

To use PVM, they must set `PVM_ROOT` (§2.8). They should also set `MANPATH` (§2.20) if it's supported, to be able to read on-line manual pages.

The easiest way to get started is to copy the examples from `$PVM_ROOT/examples` to `$HOME/pvm3/examples`, compile them and try them out, modify them, etc. There are small tutorials in the PVM book and user guide.

2.12 Machines at my site share a single /tmp directory

PVM creates files in `/tmp` for logging and locking. These will collide if different machines share the same tmp.

Get a PVM release version 3.2.3 or later. It includes a compile-time option to change the names of the files in `/tmp` to include the host name, avoiding the conflict.

If you want this option to be enabled for all machine types, edit `src/Makefile.aimk` and add `-DSHAREDTMP` to the (uncommented) `OPTIONS=` line near the top. If you want it set only for certain architectures, edit the config files for those machines (`conf/*.def`) and add `-DSHAREDTMP` to each `ARCHCFLAGS=` line.

You'll need to rebuild PVM and relink any applications that use libpvm.

2.13 Can I have two different versions of PVM on the same machine

No problem – at least with recent versions (3.2 and later). You can install as many versions as you want, in for example `/usr/local/pvmX.Y.Z`. Then `PVM_ROOT` should be set to the path of the version you want to use. Note that the different versions may not *interoperate* (§2.5).

2.14 How do I start PVM?

To get PVM running, you must start either a pvmd or the PVM console by hand. The startup scripts are named `pvmd3` and `pvm`, respectively, and reside in directory `$PVM_ROOT/lib` (add this to your shell path). Just type `pvm`.

2.15 What are these files in /tmp?

The pvmd writes error and warning messages in a log file named `/tmp/pvml.`*uid*. *uid* is your numeric user id (generally the number in the third colon-separated field of your *passwd* entry).

Try the following command to get your uid:

```
(grep `whoami` /etc/passwd || ypmatch `whoami` passwd) \
  | awk -F: '{print $3;exit}'
```

If PVM was built with the `SHAREDTMP` option (used when a cluster of machines shares a `/tmp` directory), the log file will instead be named `/tmp/pvml.`*uid*.*hostname*.

The pvmd publishes the address of the socket to which local tasks connect in a file named `/tmp/pvmd.`*uid*. The pvmd creates the socket address file while starting up, and removes it while shutting down. Another pvmd won't start if the file already exists.

If the pvmd can't create the file because the permissions of `/tmp` are set incorrectly or the filesystem is full, it won't be able to start up. If the pvmd is killed with un uncatchable signal or other catastrophic event such as a (Unix) machine crash, you may need to remove the socket address file before another pvmd will start on that host.

2.16 What do I call the host file?

You can call it anything you like and put it anywhere. It's passed to PVM (the console or pvmd) as a command line parameter. For example:

```
ig 3; cat > H
thud
honk
^D

ig 4; pvm H
pvm> conf
3 hosts, 2 data formats
                    HOST     DTID     ARCH   SPEED
                      ig    40000    ALPHA    1000
                    thud    80000     SUN4    1000
                    honk    c0000    ALPHA    1000
```

```
pvm> halt
libpvm [t40001]: mxfer() EOF on pvmd sock

ig 5;
```

2.17 Hostfile defaults aren't getting set

Note that default lines (beginning with '*') aren't cumulative. New defaults are applied to the system defaults, not the current ones. For example, if the following two lines are entered in the order shown, only the `ep` default will be left changed.

```
* so=pw
* ep=pvm3/bin/$PVM_ARCH:/bin:/usr/bin
```

Combine multiple default lines into a single one.

2.18 Hostfile options aren't taking effect

Make sure the names in the host file and those used to add hosts match exactly. The pvmd uses `strcmp()` to match names in the host file, so local names won't match fully qualified ones, e.g. `thud` is different than `thud.cs.utk.edu`.

Also, the master pvmd gets its name from `gethostbyname()` (like the `hostname` command) unless you change it with the `-n` command-line parameter. If this doesn't match a name in the host file, parameters for the master pvmd won't get set.

2.19 What does "speed" in the machine configuration mean?

It was part of the outline of a load-balancing mechanism, but it's not currently used. The idea was that if you're running more than one task per host in the virtual machine, the faster machines should get more tasks. The scheduler used by `pvm_spawn()` should look at the speeds and current loads when assigning new tasks to hosts. You can actually set the speed of each host (which defaults to 1000) to a value from 1 . . . 1000000 by using the host file sp= option.

It's possible to write a wrapper function for spawn that reads the host configuration and list of running tasks and makes such a decision. If you're really serious about it, think about writing a *resource manager*.

2.20 How do I read the on-line manual pages?

The manual pages reside in $PVM_ROOT/man. If your system supports the `MANPATH` environment variable to supply the `man` command with a list of directories, add this one to it.

If you use `csh` or a derivative, add the following lines near the end of your `.login` or `.cshrc` file:

```
if (! $?MANPATH) setenv MANPATH /usr/man:/usr/local/man
setenv MANPATH "${MANPATH}:$PVM_ROOT/man"
```

If you use `sh`, add the following lines to your `.profile`:

```
if [ "$MANPATH" = "" ]; then
    MANPATH=/usr/man:/usr/local/man
fi
MANPATH="${MANPATH}:$PVM_ROOT/man"
export MANPATH
```

2.21 How do I shut down PVM?

The preferred way to shut down a virtual machine is to type *halt* at the PVM console, or to call libpvm function `pvm_halt()`. When shutting PVM down from the console, you may see an error message such as *EOF on pvmd sock*. This is normal and can be ignored.

You can instead kill the pvmd process. It will shut down, killing any local tasks with `SIGTERM`. If you kill a slave pvmd, it will be deleted from the virtual machine. If you kill the master pvmd, the slaves will all exit too. Always kill the pvmd with a catchable signal, for example `SIGTERM`. If you kill it with `SIGKILL` it won't be able to clean up after itself, and you'll have to do that by hand.

2.22 Does PVM use shared memory?

Recent versions (3.3 and later) of PVM use shared memory as a transport mechanism for communication between tasks (§1.3). PVM does not simulate shared memory (provide shared-memory primitives to the programmer).

3 Common error messages

3.1 Error messages printed by libpvm

Error messages printed by libpvm are in the format: `libpvm[txxxxx]: XXX`, where `xxxxx` is the task id and `XXX` is the error message. The error return codes and messages are listed in the `pvm3.h` header file and described in the libpvm(3) manual page.

3.2 Error messages in the pvmd log file

The pvmd logs all sorts of operational errors in its log file (see §2.15). Some may be helpful (for example when trying to start new hosts), but most probably won't make much sense.

3.3 `Can't start pvmd`

If you're starting PVM from scratch (by typing "pvm"), this means the console is unable to start the pvmd. It may not be able to find the executable because `PVM_ROOT` isn't set or because PVM isn't installed correctly. Or, the pvmd may be having trouble starting up. If the pvmd complains that it can't bind a socket, perhaps the host name set for the machine does not resolve to an IP address of one of its interfaces, or that interface is down. The console/pvmd option `-n`*name* can be used to change the default.

If you see this message when adding a host to the virtual machine, it means that the master pvmd is having trouble starting a slave pvmd on the host. PVM can fail to start up for a number of reasons. Besides those out of your control, such as a host or network being down, the following are the most common.

When a pvmd fails to start up, you may not get the entire error message. Try running PVM on the remote host by hand by typing (to the shell): `rsh` *host* `'$PVM_ROOT/lib/pvmd -s'`. On some machines, you need to use `remsh` rather than `rsh`. You should see only a complaint like `slave_config: bad args`.

If you get `Permission denied`, it probably means that your .rhosts file on the remote machine doesn't include your local host name. You need to add a line containing: /em local-host local-user-name to your `.rhosts` file on the remote machine.

Sometimes a machine has more than one name, and the remote machine may think your local machine has a different name than you do. To find out what the remote machine thinks is the name of your local machine, log in to the remote machine and type:

`who am i`

This will show you what the remote host thinks is the name of the local host (it may be truncated to 16 characters). Make sure you put this name (in full, not truncated) in your .rhosts file on the remote machine. Some older machines will insist that you spell the name exactly the same, including getting the case right; newer machines will let you spell it in either upper or lower case.

If you get `Login incorrect`, it probably means that there isn't an account on the remote machine with the same login name you're using the local machine. You need to add a `lo=`*username* option to your PVM host file.

Make sure `PVM_ROOT` is set on the slave host (§2.8) or that `PVM_DPATH` is set on the master host.

Make sure you're not printing anything in your shell startup file, or trying to set terminal modes (e.g. with `stty`). If you must print something, either move it to `.login`, or put it in a conditional so it only happens when you log in and not when using rsh, for example:

```
if ( { tty -s } && $?prompt ) then
    echo terminal type is $TERM
    stty erase '^?' kill '^u' intr '^c' echo
endif
```

3.4 Can't contact local daemon

This means that a previously running pvmd crashed, leaving behind its *socket address* file. The pvmd should log error message `pvmd already running?`. Find and delete the address file (see §2.15).

3.5 Version mismatch

The console and pvmd (or pvmd and another pvmd) were built from different, incompatible, versions of PVM. Make sure compatible versions are installed everywhere.

3.6 `No such host`

This means a host name in the host file (or the name used with the pvmd `-n` option) couldn't be resolved to an IP address.

IP numbers can't be used in the host file, because `rsh` and `rexec()` don't accept them.

3.7 `Duplicate host`

This means that PVM thinks there is another pvmd (owned by the same user) already running on this host. If there really isn't one running, you'll need to delete the socket address file (§2.15).

3.8 `libpvm [t40001]: mxfer() EOF on pvmd sock`

If you just typed "halt" at the console, don't worry about this message; it doesn't mean that anything is wrong. Otherwise, it means the pvmd has died or been shut down. Check the pvmd log file (§2.15).

3.9 `No such file`

This message is printed when a PVM application is run without starting a pvmd first.

3.10 `failed to start group server`

This means that a function in the group library (`libgpvm3.a`) could not spawn a *group server task*. Tasks using group library functions must be able to communicate with this server. It is started automatically if one is not already running. The group server executable (`pvmgs`) normally resides in `$PVM_ROOT/bin/$PVM_ARCH`, which must be in the pvmd search path. If you change the path using the host file `ep=` option, make sure this directory is still included. The group server may be spawned on any host, so be sure one is installed and your path is set correctly everywhere.

4 Programming

4.1 Writing Programs

4.1.1 What is aimk?

Aimk is a wrapper program for *make* that determines the PVM machine architecture and changes to a subdirectory before running make.

It is intended mainly for building the PVM distribution from source, but may be useful to you in maintaining your own PVM programs. Read the man page.

4.1.2 How can I build my programs for different architectures?

PVM doesn't do it automatically, but the procedure can often be automated easily – if your directories are cross-mounted and you use aimk (§4.1.1).

A script such as the following builds a program for each of several machines in parallel, tagging the make output with the host name:

```
foreach m (ig thud cranberry garlic)
    rsh $m "cd $cwd ; make" < /dev/null \
    |& sed -e "s/^/${m}: /" &
end
```

4.1.3 Is message order between tasks preserved?

Between two given tasks, A and B, message order is preserved. That is, if A sends ten messages to B using `pvm_send()`, B will download them in that same order. Note messages can be picked out of the middle of the receive queue in B, though, by causing `pvm_recv()` or `pvm_nrecv()` to select a particular message.

The order of arrival *between* direct messages (`pvm_send()`) and multicast messages (`pvm_mcast()` or `pvm_bcast()`) is not guaranteed. It stays consistent right now (version 3.3) because the same routing path is taken by each message. Note that the path is not the same (and ordering is not guaranteed) when using `PvmRouteDirect`: Sent messages go from task to task, while multicast messages are routed via the pvmds.

4.1.4 How can I pack a struct in a message?

Unfortunately, that's a hard thing to do in C without some sort of preprocessing before compilation. You'll need to write special pack and unpack routines for any struct you want to pack, for example:

```
struct foo {
    int *values;        /* vector of (length) elements */
    short length;
    double perf;
    int flags;
};

int
pkfoo(foop, cnt, std)
    struct foo *foop;   /* vector of foos */
    int cnt;            /* number of foos */
    int std;            /* stride between foos */
{
    int cc;

    while (cnt-- > 0) {
        if (cc = pvm_packf("%hd %*d %lf %d",
                           foop->length,
                           (int)foop->length,
                           foop->values,
                           foop->perf,
                           foop->flags))
            return cc;
        foop += std;
    }
    return PvmOk;
}

int
upkfoo(foop, cnt, std)
    struct foo *foop;
    int cnt;
    int std;
{
    int cc;

    while (cnt-- > 0) {
```

```
            if ((cc = pvm_unpackf("%hd", &foop->length))
            ||  (!(foop->values = (int*)malloc(foop->length
                                              * sizeof(int)))
                 && (cc = PvmNoMem))
            ||  (cc = pvm_packf("%*d %lf %d",
                                (int)foop->length,
                                foop->values,
                                &foop->perf,
                                &foop->flags)))
                return cc;
            foop += std;
        }
        return PvmOk;
    }
```

Note that if the struct contains only primitive types (or other structs that contain only primitive types), that is, no pointers, and you're passing it to a host of identical type (not just one with the same data representations (because they might not build structs the same way)), you could get away with:

```
struct foo foo1;

pvm_pkbyte((char*)&foo1, n * sizeof(struct foo), 1);
...
pvm_upkbyte((char*)&foo1, n * sizeof(struct foo), 1);
```

4.1.5 The compiler can't find the PVM header files

PVM applications written in C should include header file pvm3.h, as follows:

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

You may need to specify the PVM include directory in the compiler flags as follows:

```
cc ...  -I$PVM_ROOT/include ...
```

A header file for Fortran (fpvm3.h) is also supplied. You can include it using a statement like:

INCLUDE 'fpvm3.h'

You may want to make a copy of `fpvm3.h` in your source directory.

4.1.6 What libraries do I need to link with?

PVM applications written in C must be linked with at least the base PVM library, libpvm3.a. Fortran applications must be linked with both libfpvm3.a and libpvm3.a. Programs that use group functions must also be linked with libgpvm3.a. On some operating systems, PVM programs must be linked with still other libraries (for the socket or XDR functions).

Note that the order of libraries in the link command may be important. You may also need to specify the PVM library directory in the link command. A correct order is shown below (your compiler may be called something other than cc or f77).

```
cc/f77 [ compiler flags ] [ source files ] [ loader flags ]
       -L$PVM_ROOT/lib/$PVM_ARCH -lfpvm3 -lgpvm3 -lpvm3
       [ libraries needed by PVM ] [ other libraries ]
```

`aimk` (§4.1.1) automatically sets environment variable `PVM_ARCH` to the PVM architecture name and `ARCHLIB` to the necessary system libraries. You can use these variables to write a portable, shared makefile, called `Makefile.aimk`.

4.1.7 Missing symbols when I try to link my application

If the undefined symbols are anything like: `builtin_va_arg_incr`, you're probably linking an application built using gcc against a libpvm built using another compiler, or the other way round. If you build any part of a program using gcc, you probably have to link it using gcc.

If you're missing other symbols, from PVM or XDR or socket functions, see also §4.1.6.

4.2 Running and debugging programs

4.2.1 The sender doesn't get a copy of multicast messages

Libpvm automatically removes the sender from the tid list passed to mcast. It seems more often to be the case that the sender doesn't want to get a copy of the message, even if it is in the list of addresses.

If the sender must get the message too, you'll need to send the message explicitly with, f.e.:

```
pvm_send(pvm_mytid(), code);
```

Or to do it more efficiently, after the mcast, call:

```
pvm_setrbuf(pvm_getsbuf());
```

4.2.2 My tasks can't read input from the keyboard

Spawned tasks have no *stdin*, because `/dev/null` is opened as file 0. You can open files as usual for input, however.

4.2.3 My tasks can't open input/output files

This is probably because they're not running in the expected directory. See §4.2.6.

4.2.4 Where does stuff printed in tasks go?

Why does stuff printed by child tasks appear in the master pvmd log file, while output from the parent task appears on the window where you start the parent?

The spawned tasks' output goes to their pvmds; their *stdouts* are not inherited from the parent task. Each pvmd collects output from its tasks through pipes and passes it to the master pvmd, which logs it in `/tmp/pvml.`*uid*(§2.15) and on its stdout. Additionally, *stderr* of spawned tasks is duped to stdout - they're not separate files. Finally, spawned tasks have no *stdin*, because `/dev/null` is opened as file 0.

Tasks started by hand have normal stdin, stdout and stderr connections.

You can redirect output (stdout/err) from spawned tasks to the console, and to a file if you like. Type "help spawn" for more information. If the output is not redirected to the console, it goes to the master pvmd logfile as before.

PVM output redirection is inherited, so if you spawn a task from the console and redirect its output to the console, you'll also get output from all its children.

You can also call `pvm_catchout()` in the master task of your program before spawing child tasks. Output from children will be sent to the master and printed on its stdout. Be sure to call `pvm_exit()` to have to master wait for all output from children before exiting.

4.2.5 I don't see printed output from tasks right away or at all

The output from a spawned task goes through a pipe to the task's local pvmd, which routes it in a message to the output collector (the master pvmd, console or ancestor task that called `pvm_catchout`).

The C *stdio* library (`fgets`, `printf`, etc.) buffers input and output for efficiency whenever possible. That is, it tries to reduce the frequency of actual read or write system calls. It decides whether to buffer by looking at the underlying file descriptor of a stream. If the descriptor is for a tty, it buffers only a line at a time, that is, the buffer is flushed whenever the newline character is encountered. If the descriptor is a file, pipe, or socket, however, stdio buffers up much more, typically 4k bytes.

Thus their stdio libraries buffer more than a line before writing to the socket (because they don't know they're really talking to a human).

There are a few ways to see your output at the correct time. You can put: `fflush(stdout)`

after each `printf()` (or periodically in the program). Or, change the buffering mode of the stdout stream to line-oriented for the entire program by putting: `setlinebuf(stdout)`

near the top of the program. See the *setlinebuf* man page for more information.

Fortran users may encounter similar problems. Some Fortran libraries include a FLUSH subroutine, some don't; you'll have to figure that one out yourself.

Also, if the program is killed or doesn't exit cleanly, (in C, by calling `pvm_exit` and `exit`) any buffered output may be lost.

4.2.6 In what directory do tasks spawned by PVM run?

Tasks inherit working directory `$HOME` from the pvmd, unless you change it with host file option `wd=` (this is a per-host option). If you open files in a task to read or write, they will be in `$HOME` unless you specify an absolute or relative path (f.e. `"src/glarp/input-file"`).

4.2.7 How can I spawn a process with a nice priority value?

Sorry, you can't. You'll have to call `setpriority()` in your code. Or you could replace your program with a shell script, f.e.:

```
% cd ~/pvm3/bin/SUN4
```

```
% mv prog prog-
% echo 'P=$0"-"; shift; exec nice -10 $P $@' > prog
%  chmod 755 prog
```

Then when you spawn `prog` the shell script will exec `prog-` (the real thing) at a new priority level.

4.2.8 What does "mc pkt ... scrapped" mean?

With early releases of version 3, you may sometimes see the following message in the pvmd log file:

`[t80140000] netinpkt() mc pkt from t41829 for t8437d scrapped (no dst)`

This message does not mean that an error has occurred. It is printed when a multicast packet between pvmds that actually arrived is resent – because the sender thinks the packet was dropped, possibly because the ack was lost. The receiver logs an error instead of silently dropping the packet. This doesn't mean messages were lost or corrupted.

4.2.9 What does "message scrapped" mean?

This message in the pvmd log file means that a message arrived at the pvmd addressed to a task that didn't exist. This can be caused by sending a message to a random destination or to a task that has exited or was killed.

4.2.10 Memory used by the pvmd increases when programs run

When you run programs under PVM, a `ps` may show the memory used by the pvmd increasing. After the program stops, the percentage of memory doesn't decrease.

The pvmd data size will grow as it runs. It uses malloc() (which uses sbrk() probably) to get buffers and malloc never returns memory to the system. However, the buffers are freed, so malloc uses the same space the next time it needs buffers. I would expect the data size to grow when your program is first run, then stay more or less constant for subsequent runs.

Of course, there could be a memory leak in the pvmd, though we don't know of any big ones.

4.2.11 My program is hanging

Tracing might help you find out where it's getting stuck. Try starting your program from the console with the spawn command:

`spawn -@` *filename*.

You may need to first use `trace *` to enable tracing on all events. A tip – if you must have a tty to run your program (to type input to it), you can spawn an xterm and start the process there. Use the following alias for convenience:

```
alias tr-xterm spawn -.  -@ /bin/csh -cf 'exec xterm -n PVM'
```

This means "spawn an xterm on this host with tracing enabled". When you start a PVM program in the xterm window, it will inherit the task context held by the xterm, and so have tracing enabled for it. The trace information will go back to the console.

A warning – some people write code that uses `pvm_parent()` to decide if a process is a master or slave. If you start the master as shown, it will have a parent task (the PVM console), and may do the wrong thing. There's no way right now to set a task's parent to null.

Note that the task context held by the xterm is freed when the task exits. To re-run the program, you'll need to start another xterm.

5 Advanced programming

5.1 How can I get input from PVM messages and e.g. X?

It is possible for a PVM task to get input from PVM and other sources at the "same time", and there are at least two common ways to do it.

The simplest is to poll each input in turn, and handle any that having data pending. For example, you might use `pvm_probe()` to check for messages and `XCheckWindowEvent()` to check for X events. The obvious disadvantage is that spin-waiting wastes CPU time. This is less of a problem on dedicated machines than time-shared ones. You can slow down the polling frequency by using `pvm_trecv()` to introduce a small timeout in the loop.

A more efficient but also more complex solution is to call `pvm_getfds()` to get list of file descriptors used by libpvm for communication, and then wait on these (and your own) files with `select()`. This is simple if direct routing is disabled, since only a single socket (to the pvmd) will be in use.

5.2 I'm not getting notify messages

Note that `pvm_notify()` is called by the task requesting the notify messages (to be sent to it). The list of tasks (or hosts) that trigger notify messages is passed to notify. A common programming mistake is to get this backward.

5.3 My application needs to start PVM itself

A function, `pvm_start_pvmd()`, exists for this purpose. There is a man page.

5.4 Can I spawn shell scripts

Yes, as well as other non-PVM programs (e.g. `/bin/date`). Even if a spawned program makes no PVM calls, it will have a task id and context (output redirection, etc.). When it exits, the pvmd will catch the exit signal and clean up the task context.

If a non-PVM program (such as a shell script) starts a PVM program, the PVM program *reconnects* to the context allocated to its parent, claiming the task id and context. When it exits, it takes the task id with it. This means you can spawn a shell script that starts a PVM program, and the PVM program will receive the same task id as was returned by the call to spawn. The debugger script (`$PVM_ROOT/lib/debugger`) does this.

Note that if a second PVM program is started by the shell script, it will *not* reconnect to the same task context, which is freed by the exit of the first PVM program. Take for example the following timeline:

Parent process →		Child	Action
xterm			Exec'd by *pvmd*
	script		Exec'd by *xterm*
		$prog_1$	Claims task context
			Context is freed
		$prog_2$	Allocated new task context

Thus, $prog_2$ gets an anonymous context, as though it was not spawned by PVM. It has a new task id and no output redirection.

5.5 Can a task fork and the child be a task?

Yes, but you must first disconnect the child from PVM manually. Otherwise, both tasks will attempt to talk to the pvmd through the same socket, using the same task context, and things will get scrambled.

The following code fragment can be used to fork and reconnect the child as another PVM task:

```
if (cc = fork()) {      /* parent */
    if (cc == -1)
        perror("fork");

    /*
    *  parent does stuff here
    */

} else {                /* child */
    pvmendtask();
    pvm_mytid();

    /*
    *  child now enrolled as separate task
    */
}
```

Note that the child task gets an anonymous context – as though it was not spawned by PVM. Most importantly, it does not have PVM output redirection. However, if the parent task stays alive at least as long as the child, the child's output will be collected through the pipe to the pvmd as though the *parent* had printed it.

5.6 Can I use signals in a PVM program?

You can send signals using `pvm_sendsig()` and catch them, but you'll probably get into trouble if you call PVM functions (even *pk* and *upk*) from within a signal handler. PVM doesn't block signals anywhere, so if the program is already in a PVM function, catches a signal and enters another one, some of the global state in libpvm might get stomped on.

If you use signals, don't call PVM functions (or *libc* functions such as `malloc`, depending on which machine you're using) in signal handlers. Check out the man pages for signal and the introductions to sections 2 and 3 on your machine.

As an example, you might use a signal to wake up a computing task to receive a message. About the only safe way to do this I can think of is to probe the PVM sockets (which you can get with `pvm_getfds()`) with `select()` and set a flag, then notice that in your main computation loop and call `pvm_nrecv()` to see if anything has arrived.

Also, note that signal numbers are not mapped between different architectures. They just usually are the same.

5.7 What signals are safe to use in a PVM program?

PVM doesn't use signals itself, except `SIGTERM`, which is sent by `pvm_kill` reset from the console, or if the pvmd goes down. I'd suggest using `SIGUSR1` and `SIGUSR2`.

But, see §5.6.

5.8 I need to get machine loads to write a load balancer

If this was easy, we would already have done it.

The simplest way I can think of is to open a pipe to read from a program that gets it for you. This is somewhat portable and doesn't require special permission to run. Likely programs are *uptime, w* and *who*. Other useful programs for getting machine status are *vmstat, pstat* and *ps*.

Using any of these is kind of a hack, but I've never found a better way. You could also write your own code that gets relevant kernel variables directly. The problems with that approach are that the program may need special permission to run (f.e. setgid to group kmem), and it would be even less portable, since kernels from different companies are subtantially different.

Another method to determine cpu availability is to run a program and measure cpu time consumed vs. real time, e.g. using `getrusage()` and `gettimeofday()` (among others). But this may require you to impose a dummy load on the system and would likely be slower, since the program must run for a while to get an accurate reading.

5.9 How can I rewind a message buffer to unpack it again?

If you are willing to trick libpvm, you can rewind a message. Setting it as the current packable message, and then back, will reset the read pointer.

```
m = pvm_setsbuf(pvm_getrbuf());
pvm_setrbuf(pvm_setsbuf(m));
/* rbuf now rewound */
```

(The purpose of m is just to save the current send buffer, if one exists)

But, this is a dirty trick. It should work in any of the unix ports, but I'm not sure about the others. And it's not a real feature, so it could conceivably go away.

We've been asked for a rewind before, but I've held out because it breaks optimizations one might want to do, like transferring the message data during unpack instead of receive, and discarding it as you go to save space.