PIOUS for PVM
Version 1.2
User's Guide and Reference Manual

Steven A. Moyer
V. S. Sunderam

Department of Mathematics and Computer Science
Emory University, Atlanta, GA 30322, U.S.A.
pious@mathcs.emory.edu

January 1995
(Revised May 1995)

Research supported in part by the National Science Foundation, under awards CCR-9118787 and ASC-9214149, the U. S. Department of Energy, under grant DE-FG05-91ER25105, the Office of Naval Research, under grant N00014-93-1-0278, the Army Research Office, under grant ARO-93G0045, and the National Aeronautics and Space Administration, under grant NAG2-828.

Contents

1 Introduction

PIOUS, the Parallel Input/OUtput System, implements a virtual parallel file system for applications executing in a metacomputing environment. PIOUS supports parallel applications by providing coordinated access to file objects with guaranteed consistency semantics. For performance, PIOUS declusters file data to exploit the combined file I/O and buffer cache capacities of networked computer systems.

This user's guide describes PIOUS Version 1 as implemented for the PVM [4] metacomputing environment. PVM is well established and widely used and thus represents a logical choice of platform for the PIOUS package. However, PIOUS is easily ported to any platform providing similar functionality.

Presented below is a brief overview of the PIOUS project and software. Note that throughout this manual, familiarity with PVM is assumed.

1.1 Project Overview

The PIOUS project started in the Summer of 1993 at Emory University as part of a larger research initiative aimed at understanding and developing flexible, reliable, and scalable network parallel file systems. At that time, no freely available software capable of supporting basic parallel file system research existed. Thus it was decided that the first phase of the initiative would be the development of a software platform to serve as a vehicle for further research. The PIOUS software is the result of that effort.

PIOUS is being made available to the user community at large and is intended to serve both as a platform for supporting high-performance parallel applications, and as a tool for parallel file system research. The PIOUS software is distributed under the terms of the GNU Library General Public License Version 2 (LGPL), as published by the Free Software Foundation, and is provided "as is" without any warranty. A copy of the LGPL is included with the software distribution and details the terms of the license. Note that the LGPL *does* allow proprietary applications to make use of the PIOUS software.

1.2 Software Overview

The PIOUS software architecture is depicted in Figure 1. PIOUS implements a virtual parallel file system within the PVM environment that consists of a service coordinator (PSC), a set of data servers (PDS), and library routines linked with client processes (PVM tasks). PVM provides both the process management facilities required to initiate the components of the PIOUS architecture, and the transport facilities required to carry messages between client processes and the PIOUS virtual file system.

A single PIOUS Service Coordinator initiates major activities within the virtual file system. For example, when a process opens a file the PSC is contacted

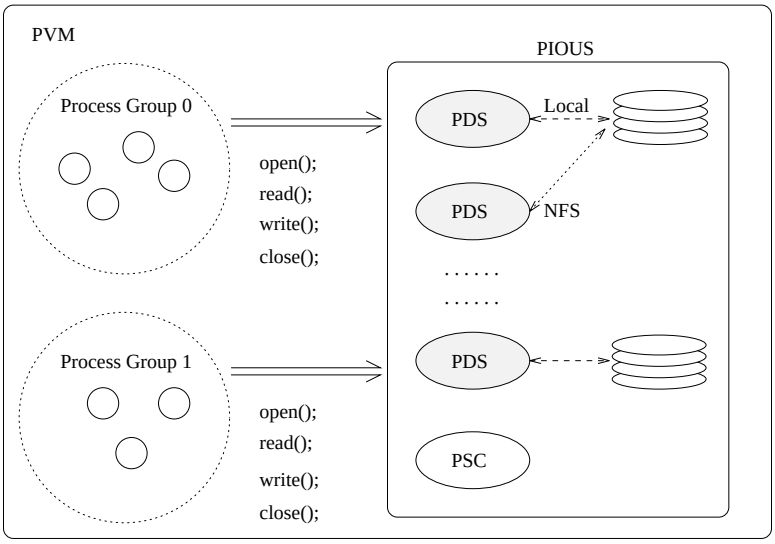

Figure 1: PIOUS software architecture

to obtain file meta-data and to ensure that the requested file access semantics are consistent with those of other processes in a logical process group.

A PIOUS Data Server resides on each machine over which a file is declustered. Data servers access disk storage in response to client requests via the native file system interface. Though ideally each PDS accesses a file system local to the machine on which it resides, two or more PDS may share a file system via a network file service.

Finally, each client process is linked with library functions that define the PIOUS file model and interface (API). Essentially, library functions translate file operations into PSC/PDS service requests.

The PIOUS architecture emphasizes asynchronous operation and scalable performance by employing parallel independent data servers for declustering file data. Coordinated file access is provided via a data server protocol based on volatile transactions [3]. Further design and implementation details can be found in [1, 2].

2 Installing PIOUS for PVM

This section details how to obtain and install PIOUS for use with PVM. It is presumed that PVM is already installed for each machine that is to host the PIOUS virtual file system.

Throughout this document, PVM_ROOT represents the value of the PVM environment variable of the same name that defines the pathname of the PVM system root directory. Similarly, PVM_ARCH represents the value of the PVM environment variable of the same name that indicates the host architecture type.

2.1 Obtaining PIOUS

The PIOUS software and documentation is available from the WWW page located at `http://www.mathcs.emory.edu/pious.html`, or via anonymous ftp from `ftp.mathcs.emory.edu` in the directory `pub/pious1`. Source code is stored as a compressed/uuencoded tar file, with a version prefix label of the form **pious1.x.y**. The corresponding user's guide and reference manual is stored as a compressed/uuencoded postscript file, with a version prefix label of the form **piousUG1.x**.

2.2 Unpacking

The PIOUS source file directory that results from unpacking the tar file may be placed anywhere. However, it is particularly convenient to unpack PIOUS in the same directory as the PVM directory `pvm3`; i.e. such that PVM resides in the directory `PVM_ROOT/../pvm3` and PIOUS resides in the directory `PVM_ROOT/../pious1`.

To unpack the PIOUS source, execute the following commands where `x` and `y` represent version numbers:

```
% uudecode pious1.x.y.tar.z.uu
% uncompress pious1.x.y.tar.Z
% tar xvf pious1.x.y.tar
```

The result is the PIOUS source directory `pious1`.

Similarly, the user's guide can be restored to postscript format by executing the commands:

```
% uudecode piousUG1.x.ps.z.uu
% uncompress piousUG1.x.ps.Z
```

2.3 PIOUS Directory Structure

The PIOUS source directory is structured as follows:

- `pious1/Makefile` is a regular file that guides the make utility during the build process.
- `pious1/doc` is a directory that contains any special release notes and the text to the GNU Library General Public License.

- `pious1/examples` is a directory that contains example PIOUS applications.
- `pious1/fsrc` is a directory that contains PIOUS Fortran library source code files.
- `pious1/lib` is a directory that contains build, start-up, and utility shell scripts.
- `pious1/man` is a directory that contains manual pages for all PIOUS functions; this directory should be added to the MANPATH environment variable.
- `pious1/src` is a directory that contains all PIOUS system source code files.

2.4 Installation Options

Once the PIOUS source files have been unpacked, the next step is to choose among the installation options defined in the file `pious1/Makefile`. The installation options of interest are:

USEPVM33FNS to utilize functions only available in PVM 3.3 or later,

USEPVMDIRECT to employ PvmRouteDirect message routing policy, and

USEPVMRAW to employ PvmDataRaw message encoding.

By default, install uses *all three* of the above options. To change the install options, simply edit the file `pious1/Makefile` as the documentation there describes.

For performance, it is highly recommended that PIOUS be installed with USEPVMDIRECT defined. Due to a known bug in all versions of PVM up through PVM 3.3.3, the latest tested by the authors, PIOUS *must* be installed with USEPVMRAW; as a result, PIOUS can only be executed in a homogeneous data environment.

2.5 Compilation Issues

PIOUS can normally be installed effortlessly on any machine that PVM has been built on, provided that the system has a Unix[1] style interface.

However, if the default C-language compiler is not ANSI compliant, then function and macro declarations in the header file `pious1/src/misc/nonansi.h` may require updating. Declarations in this file are typically not part of a standard header file in non-ANSI C. If the declarations are not correct for the host machine, then edit the file as indicated in the documentation included there.

[1] Unix is a registered trademark of X/Open Company Ltd.

Likewise, if the system will be operating in a network environment that includes machines that do not support 32-bit (or larger) integers, then limit declarations in the header file `pious1/src/include/pious_types.h` may require updating as indicated in the file documentation.

2.6 Building

After unpacking the PIOUS source files and choosing the appropriate installation options, the system is ready to be built. Before attempting to build PIOUS confirm that:

- PVM is already built for the current architecture, *including* the dynamic group library, and
- PVM_ROOT is defined in the build environment.

The architecture type PVM_ARCH will be determined automatically during the build procedure.

PIOUS is built by executing `make` in the `pious1` directory. As a result, the following files are placed in the PVM directory:

- the data server executable `pious1DS` and service coordinator executable `pious1SC` are placed in `PVM_ROOT/bin/PVM_ARCH`,
- the library files `libpious1.a` (C) and `libfpious1.a` (Fortran) are placed in `PVM_ROOT/lib/PVM_ARCH`,
- the header files `pious1.h` (C) and `fpious1.h` (Fortran) are placed in `PVM_ROOT/include`, and
- the start up script `pious` is placed in `PVM_ROOT/lib`.

PIOUS is now ready to be used by any PVM application.

3 Configuring and Starting PIOUS

This section details the steps necessary to prepare the PIOUS execution environment, and to configure and start the PIOUS virtual file system.

3.1 Preparing the Execution Environment

Each data server in PIOUS accesses disk storage via the native file system interface of the host machine, as discussed in section 1.2. Functionally, it does not matter if the files accessed are local or remote to the data server host. Though best performance is usually obtained via local file access, PIOUS can often provide performance benefits even in the case where all data servers share storage by aggregating the CPU and buffer cache capacities of the hosting machines.

It is common in network environments for hosts to have transparent access to both local and remotely mounted (shared) disk storage. In this case the PIOUS data servers can easily access local files. However, if a host has been configured such that all local storage is allocated to the `/tmp` space, then it will be necessary to have the system administrator create a local file system for that host if local file access is desired.

PIOUS applications benefit from the functionality provided by the system regardless of how data storage access is configured.

3.2 Configuring PIOUS

PIOUS library functions allow the user to dynamically specify a set of hosts for declustering file data. Each PIOUS file can be declustered on any subset of the machines in the PVM environment. PIOUS will automatically spawn a data server on each host that contains file data on a demand basis.

Optionally, a set of default data server hosts within the PVM environment can be designated when PIOUS is started. An application can then operate on PIOUS files without specifying location information in the form of hosts for declustering file data. The PIOUS system assumes that if no location information is provided, file data is declustered on the default data servers.

The simplest method for using the PIOUS virtual file system is to predetermine a set of hosts to be used for declustering all PIOUS file data. Each of these hosts is then configured with a local file system if necessary and desired, as discussed above, and added to the appropriate PVM host files. Finally, a PIOUS default data server host file, termed a *dsfile*, is created specifying the chosen hosts. The dsfile is used each time PIOUS is started, thereby relieving applications of ever specifying any location information.

The PIOUS dsfile is similar in syntax to the PVM hostfile. Each dsfile entry specifies information concerning a particular data server host in the form:

hostname [sp= *path*] [lp= *path*]

A data server host entry consists of the name of the host and the optional arguments:

sp= *path* specifies the data server search root path. The *path* argument is an absolute pathname that refers to a directory file to be used as the root directory for all PIOUS pathname resolution at the data server host. That is, the *path* argument is used by PIOUS as a path prefix for all file access at the named host.

Specifying a search root path for each data server host accommodates hosts with different directory structures. For example, it allows declustering a PIOUS file on two data server hosts where a home directory on one is `/home/moyer`, and the corresponding home directory on the other is `/users/people/moyer`.

lp= *path* specifies the data server log directory path. Each PIOUS data server maintains several log files, including an error log. The *path* argument is an absolute pathname that refers to a directory file to be used by the data server on the named host for maintaining log files.

Two or more PIOUS data servers may share the same log directory file simultaneously, as all log files are distinguished by unique host and user identifiers.

Default values for both data server search root and log directory paths are specified via a host entry with a *hostname* argument value of *. To be valid, each dsfile host entry must specify both a search root and log directory path, either directly or by default.

For clarity, white space in a dsfile is ignored except to distinguish tokens. Comments may be inserted and begin with the occurrence of the # character and proceed to the end-of-line.

Figure 2 depicts a data server host file. The example is intentionally more complex than necessary to fully illustrate all features.

3.3 Starting PIOUS

To start PIOUS, it is first necessary to start an instance of PVM that includes the set of machines that will host PIOUS data servers. Refer to the PVM user's guide included with the PVM software distribution for details.

Starting the PIOUS virtual file system in the PVM environment involves executing the start up script:

`PVM_ROOT/lib/pious [` *dsfile* `]`

The start up script begins execution of the PIOUS service coordinator on the local host, which in turn parses the dsfile, if specified, and starts a PIOUS data server on each of the default hosts. A message is then written to the standard output indicating that PIOUS is operational.

PVM applications that utilize the PIOUS virtual file system can now be executed.

4 Creating PIOUS/PVM Applications

PVM applications utilize the PIOUS virtual file system by simply calling PIOUS library functions. PIOUS functions will not interfere with PVM functions in any way, provided that a PVM application *does not* utilize any of the small set of consecutive message tags reserved for PIOUS functions. By default, this set of tags is at the upper bound of the tag space; see section 8 to change the reserved tag range. The range of message tags reserved for PIOUS can be queried via the pious_sysinfo() function.

```
# Data server host file example

# Set default search root and log directory paths

* sp= /users/moyer/piousfiles    lp= /users/moyer/piouslogs

# Host         : snoopy
# Search Root  : /users/moyer/piousfiles
# Log Directory: /users/moyer/piouslogs

snoopy

# Host         : linus
# Search Root  : /users/moyer
# Log Directory: /users/moyer/piouslogs

linus sp= /users/moyer

# Change the search root path default

* sp= /users/moyer

# Host         : lucy
# Search Root  : /users/moyer
# Log Directory: /users/moyer/piouslogs

lucy

# Host         : charliebrown
# Search Root  : /people/moyer/files
# Log Directory: /people/moyer/logs

charliebrown

sp= /people/moyer/files
lp= /people/moyer/logs
```

Figure 2: PIOUS data server host file

To compile a C-language PVM application containing PIOUS library function calls, simply include the header file `pious1.h` and link with the libraries `libpious1.a`, `libgpvm3.a`, and `libpvm3.a`, in that order. Similarly, to compile a Fortran-language application, include the header file `fpious1.h` and link with the libraries `libfpious1.a`, `libpious1.a`, `libgpvm3.a`, `libfpvm3.a`, and `libpvm3.a`. Because the PIOUS header files and libraries are placed in the same directories as the PVM header files and libraries, for a given architecture, no additional search paths are required beyond those necessary to compile a standard PVM application.

4.1 General Usage

PIOUS implements a Unix-style interface, as discussed in section 5. A typical process will open a file, perform read and write operations, and then close the file.

Figures 3 (C) and 4 (Fortran) depict this typical PIOUS library usage for a program that accesses two files. The first file accessed is presumed to reside on a default set of data server hosts, while the second file accessed resides on a set of hosts directly specified in the program text.

4.2 Example and Test Applications

The PIOUS distribution comes with an example application written in both the C and Fortran programming languages. The program `rdwr.c` (`frdwr.f`) is a simple application that creates a file, writes data to the file, reads data from the file, and then closes and unlinks the file. The file name is requested when `rdwr` (`frdwr`) is executed. The file is declustered on a default set of data server hosts that must be specified when PIOUS is started.

The PIOUS distribution also comes with two applications that test the system to insure that it operates correctly in the local environment. The program `qtest.c` performs a quick test of PIOUS functionality for a wide range of application behavior. The program `flibtest.f` performs a simple test of all PIOUS Fortran library functions. Both programs require that a default set of data server hosts be specified when PIOUS is started. `qtest` takes as its first and only argument the dsfile describing the PIOUS configuration; `flibtest` requests the dsfile path name interactively. For either test, specifying an incorrect dsfile will result in erroneous results. It is strongly recommended that both tests be run when PIOUS is installed.

To compile the example and test applications, execute `make examples` in the `pious1` directory; the executables will be placed in `PVM_ROOT/bin/PVM_ARCH`. Note that because Fortran compilers have a somewhat less standard interface than C compilers, it may be necessary to change the `F77` definition in the file `pious1/examples/Makefile.archmk`; see the documentation there for a list of machines that are known to require a different definition.

```
/* PIOUS example -
 *   Access two files, one on the default set of data server hosts
 *   and the other on a set of hosts specified in the program text.
 */
#include <pious1.h>

main()
{ int fd;
  char buf[4096];
  struct pious_dsvec dsv[2];

  /* access file on default set of data server hosts */

  fd = pious_open("file1.dat",
                  PIOUS_RDWR | PIOUS_CREAT | PIOUS_TRUNC,
                  PIOUS_IRUSR | PIOUS_IWUSR);
  pious_write(fd, buf, (pious_sizet)4096);
  pious_lseek(fd, (pious_offt)0, PIOUS_SEEK_SET);
  pious_read(fd, buf, (pious_sizet)4096);
  pious_close(fd);

  /* access file on hosts marcie and patty */

  dsv[0].hname = "marcie";
  dsv[0].spath = "/users/moyer/piousfiles";
  dsv[0].lpath = "/users/moyer/piouslogs";

  dsv[1].hname = "patty";
  dsv[1].spath = "/users/moyer/piousfiles";
  dsv[1].lpath = "/users/moyer/piouslogs";

  fd = pious_sopen(dsv, 2,
                   "file2.dat",
                   PIOUS_RDWR | PIOUS_CREAT | PIOUS_TRUNC,
                   PIOUS_IRUSR | PIOUS_IWUSR);
  pious_write(fd, buf, (pious_sizet)4096);
  pious_lseek(fd, (pious_offt)0, PIOUS_SEEK_SET);
  pious_read(fd, buf, (pious_sizet)4096);
  pious_close(fd);
}
```

Figure 3: PIOUS usage example - C

```
c PIOUS example -
c Access two files, one on the default set of data server hosts
c and the other on a set of hosts specified in the program text.
c
      include 'fpious1.h'

      integer fd, rc
      character*4096 buf
      character*(PIOUS_NAME_MAX * 2) hname
      character*(PIOUS_PATH_MAX * 2) spath, lpath
      character*(PIOUS_NAME_MAX) hname_vct(2)
      character*(PIOUS_PATH_MAX) spath_vct(2), lpath_vct(2)
      equivalence (hname_vct, hname)
      equivalence (spath_vct, spath), (lpath_vct, lpath)

c access file on default set of data server hosts
      call piousfopen('file1.dat',
     >                (PIOUS_RDWR + PIOUS_CREAT + PIOUS_TRUNC),
     >                (PIOUS_IRUSR + PIOUS_IWUSR), fd)
      call piousfwrite(fd, buf, 4096, rc)
      call piousflseek(fd, 0, PIOUS_SEEK_SET, rc)
      call piousfread(fd, buf, 4096, rc)
      call piousfclose(fd, rc)

c access file on hosts marcie and patty
      hname_vct(1) = 'marcie'
      spath_vct(1) = '/users/moyer/piousfiles'
      lpath_vct(1) = '/users/moyer/piouslogs'
      hname_vct(2) = 'patty'
      spath_vct(2) = '/users/moyer/piousfiles'
      lpath_vct(2) = '/users/moyer/piouslogs'

      call piousfsopen(hname, spath, lpath, 2, 'file2.dat',
     >                 (PIOUS_RDWR + PIOUS_CREAT + PIOUS_TRUNC),
     >                 (PIOUS_IRUSR + PIOUS_IWUSR), fd)
      call piousfwrite(fd, buf, 4096, rc)
      call piousflseek(fd, 0, PIOUS_SEEK_SET, rc)
      call piousfread(fd, buf, 4096, rc)
      call piousfclose(fd, rc)
      end
```

Figure 4: PIOUS usage example - Fortran

5 PIOUS File Model and Interface

The PIOUS interface is very similar in many respects to a traditional Unix-style interface, implementing open, close, read, write, and seek file operations. Whenever possible, PIOUS function arguments and behavior are the same as, or a logical extension of, the corresponding Unix functions. Thus anyone familiar with a Unix-style file system interface can easily develop PIOUS applications.

PIOUS diverges from the traditional Unix model by implementing two-dimensional file objects and sophisticated access coordination mechanisms designed to simplify parallel applications development. These features of the PIOUS file system are discussed in detail below. A complete description of all PIOUS functions is presented in Appendix A.

5.1 File Objects

A PIOUS file is a single object composed of one or more physically disjoint data segments. Each data segment is composed of a linear sequence of zero or more bytes. The number of segments in a file is set at the time of creation and does not change for the life of the file. Figure 5 depicts a PIOUS file with four data segments.

PIOUS declusters file data on a set of specified data servers, as previously discussed. Though the method employed for declustering data is not important from a functional point of view, understanding declustering is important for developing high-performance applications.

A PIOUS file is declustered by placing each data segment on a single data server. If the number of data segments exceeds the maximum number of data servers for declustering the file, then segments are mapped to data servers in a round-robin fashion.

For example, the file in Figure 5 is shown declustered on two data servers. To reduce name space pollution at each data server host, the logical file pathname actually refers to a directory file that contains entries representing file data segments. Note that the logical PIOUS file structure directly reflects the physical structure that naturally results from declustering file data.

Because PIOUS files are two-dimensional, array and tabular data map naturally. Thus developing parallel scientific, database, and sorting applications, among others, should be greatly simplified.

5.2 File Views

The PIOUS interface provides a process group[2] with three logical *views* of a file object: global, independent, and segmented. Each file view provides a process group with a distinct set of access semantics, as defined below:

[2]Group membership is a convenient abstraction expressed at the time a file is opened. There is no relationship between logical groups in PIOUS and the PVM process group mechanism.

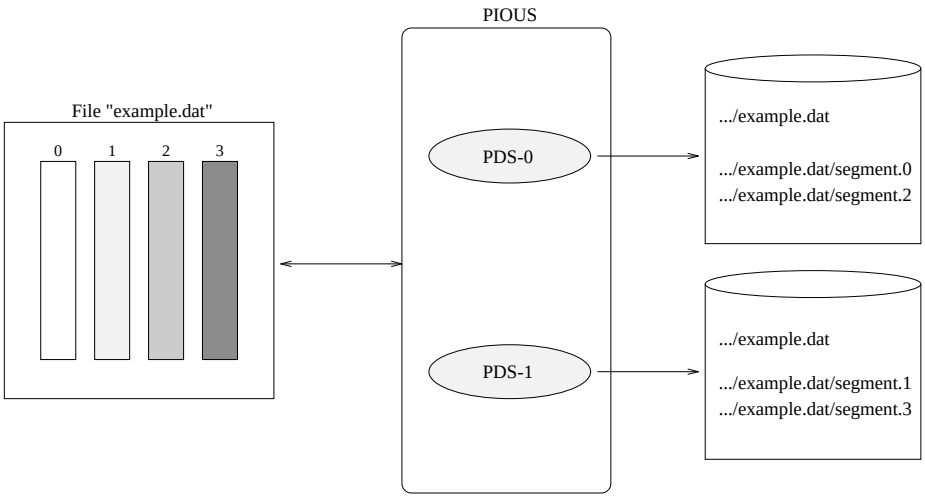

Figure 5: PIOUS file object

global A file appears as a linear sequence of data bytes; all processes in a group share a single file pointer.

independent A file appears as a linear sequence of data bytes; each process in a group maintains a local file pointer.

segmented A file appears in the natural segmented form; each process accesses a specified segment via a local file pointer.

For the global and independent file views, a linear byte sequence is defined by ordering the bytes in a file by fixed size blocks taken round-robin from each data segment; i.e. data is striped across segments. The stripe unit size is specified at the time the file is opened. Note that if the number of file data segments is a multiple of the number of data servers on which the file is declustered, then a linear file view results in an access pattern that is equivalent to traditional disk striping.

A view only defines the way a file is accessed and does not alter the physical representation. Thus a given file can always be opened with any view. All processes in a group must open a file with the same view.

5.3 EOF and Other Issues

PIOUS segmented files can greatly simplify many parallel applications. However, it is important to fully understand the impact segmentation has on file access semantics versus a traditional Unix linear file.

The most obvious difference is the notion of EOF (end-of-file). When accessing a PIOUS file in the segmented view, each process accesses a particular segment with a well defined EOF. However, when accessing a multi-segment file as a linear byte sequence, in either the global or independent view, the notion of EOF is not clear. A read access at a particular file offset can hit the EOF at one segment, yet a read access at a greater file offset can return data from a different segment.

A related issue concerns "holes" in a file. Unix-style file systems specify that a read from an unwritten portion of a file prior to the EOF returns bytes with value zero. When accessing a PIOUS file in the segmented view, each segment accessed by a process will exhibit this behavior. However, when accessing a multi-segment file as a linear byte sequence, this behavior will only be exhibited in the case where each write access that creates a hole results in data being written to all data segments.

Fortunately, working with PIOUS files in the global and independent views is relatively straightforward if the underlying file structure is kept in mind. Applications that consistently access a file as a linear byte sequence and explicitly write all data that is to be read will not encounter any of the issues discussed above.

5.4 Consistency and Fault Tolerance Semantics

PIOUS provides all processes with linearizable (and hence sequentially consistent) access to files opened under any view. Both data access and file pointer update are guaranteed to be atomic with respect to concurrency.

PIOUS also supports file access in two user-selectable fault tolerance modes: *volatile* and *stable*. Stable mode guarantees file consistency in the event of a system failure; volatile mode does not. Volatile mode is intended for non-critical applications that require high-performance. Applications requiring fault tolerance can access files in the lower-performance, but guaranteed consistent, stable mode. Currently, only volatile mode access is enabled.

Important: To provide linearizable file access, PIOUS implements a data server protocol based on volatile transactions [3]. As a result, file read and write operations can time-out if the system is unable to obtain the required locks. A read or write operation that times-out returns an error code indicating that the access has been aborted. Under normal conditions, access time-out is rare.

5.5 User Transaction Facilities

PIOUS files are normally accessed via traditional read and write functions that are guaranteed to be linearizable. To further enhance coordinated file access, PIOUS also supports the grouping of read and write operations within the

context of a single *user-transaction*[3].

User-transactions are an atomic unit with respect to concurrency, guaranteeing linearizability with respect to all other user-transactions and individual access operations. As with single access operations, user-transactions can be performed in either volatile or stable mode, though only volatile mode is currently enabled.

Utilizing user-transactions in a PIOUS application is straightforward. The functions pious_tbegin() and pious_tend() mark the beginning and end of a user-transaction, respectively. All PIOUS access operations that occur between a transaction begin and end point are part of that user-transaction. At any time after calling pious_tbegin(), but prior to calling pious_tend(), a user-transaction can be aborted via pious_tabort().

Any number of files can be accessed within the context of a user-transaction, and any PIOUS function can be called, though certain actions may be denied.

6 Performance Issues

The PIOUS virtual parallel file system is capable of providing high-performance coordinated data access. Below are a few tips for maximizing application performance.

- Access data in large blocks whenever possible. PIOUS does not currently cache data at the client. Thus each read and write operation generates at least one data server request; in the global and independent file views, a single access can generate up to one request per file data segment. As a result, the performance of PIOUS is closely tied to the message passing performance of PVM, which is a function of message size.

- Do not create files with large numbers of segments. In the global and independent file views, the number of data server requests per file access is bounded by the number of file data segments, as discussed above. For a given access size, increasing the number of data segments can result in an increase in the number of data server requests, with a concomitant decrease in request message size.

- Place data servers on hosts with local disk storage. If a data server has to access file data from a remotely mounted disk, then the total amount of network communication is increased. PIOUS data servers perform caching to reduce disk access; this will in turn help to reduce network demand if local storage is not available.

[3] To avoid confusion with the transaction mechanism that PIOUS employs transparently for all read and write operations, a transaction performed at the application level is termed a user-transaction.

- Avoid shared file pointers if not required. The global file view shares a single file pointer among all processes in a group. Because the file pointer can not be updated until each read or write operation has completed, the result is to serialize file access[4].

By understanding the high-level operation of the PIOUS file system, and developing applications accordingly, good performance can easily be achieved.

7 Limitations, Quirks, and No-no's

There are a number of limitations and implementation details that must be kept in mind when using PIOUS in the PVM environment.

- PIOUS is a virtual file system. Though PIOUS looks like the real thing from an application perspective, it takes little effort to sabotage PIOUS via the actual underlying file systems. Such things as manually writing or deleting portions of PIOUS files is highly discouraged.
- PIOUS shares the application message tag space. Because PVM has no notion of message contexts, the PIOUS library functions must share the message tag space with all client processes. Thus applications must *never* use the message tags reserved for PIOUS, as discussed in section 4.
- PIOUS does not maintain a global directory. The system depends on being supplied with location information either directly in open function calls or via a set of default data server hosts, as discussed in section 3.2. If the location information supplied is wrong, a file can not be opened.
- PIOUS can not close open files on exit. If a process opens a file and then exits before closing the file, it is possible, though extremely unlikely, that file updates will be lost. This "feature" can be avoided by installing the software *without* the option VTCOMMITNOACK defined in the file `pious1/Makefile`, though system performance will be degraded significantly in most cases.
- PIOUS does not need to be shutdown. If all files opened by all processes have been closed, then it is not necessary to shutdown PIOUS before halting PVM. However, the function pious_shutdown() has been provided for completeness.
- PIOUS instances can not coexist. Coordinated file access can not be provided by multiple instances of PIOUS executing on different PVM. For example, if one PVM is executing on **hostA** and **hostB**, and another

[4]Shared file pointers can in fact be implemented without access serialization at the cost of violating expected semantics under certain conditions. This feature will probably be added to PIOUS as a dynamically selectable option in a later release.

is executing on **hostB** and **hostC**, each with an instance of PIOUS, then coordinated file access can not be provided for PIOUS files on **hostB**.

Adhering to the above rules poses little burden to the user and will insure that PIOUS performs as expected.

8 Custom Configuration

The PIOUS file system has a number of compile-time system configuration parameters that can be tuned to meet the needs of a particular application domain.

Declarations in `pious1/src/config/pious_sysconfig.h` control all important performance parameters, including the data server cache block size and block count. See the documentation there for further details.

Declarations in `pious1/src/pdce/pdce_msgtagt.h` determine the message tag space reserved for PIOUS library functions. See the documentation there for information on adjusting the reserved tag range.

9 Porting

PIOUS is implemented in a modular fashion that allows it to be easily ported to other metacomputing environments and operating systems. The following source directories contain code relevant to porting:

- `pious1/src/pdce` contains the PIOUS interface layer that provides point to point message transport, service registration and location, and task spawning. This module will require modification when porting PIOUS to other metacomputing environments.
- `pious1/src/pfs` contains the PIOUS native file system interface layer. This module will require modification when porting PIOUS to a native file system that does not provide a standard Unix interface.
- `pious1/src/psys` contains the PIOUS operating system interface layer. This module will require modification when porting PIOUS to an operating system that does not provide a standard Unix interface.

PIOUS was developed with the intent of providing a parallel file system for network computing environments; in particular, networks of workstations. Though it has not been attempted, it is likely that PIOUS will execute as is on a true parallel processor. The pros and cons of doing so will not be discussed here. However, if PIOUS is to be run on a parallel machine, it is recommended that the PIOUS native file system interface be modified to take advantage of the parallel file system that is likely to be provided on the host machine.

10 Support

The developers of PIOUS will provide support as time permits. Question and problems should be directed to `pious@mathcs.emory.edu`. Please be as specific as possible in any bug reports, and be sure to include machine type as well as operating system, PVM, and PIOUS version numbers.

For now it is recommended that the news group `comp.parallel.pvm` be used as an avenue for discussion about PIOUS. If PIOUS gains a sufficient user base or is ported to other metacomputing environments, then the developers will initiate a vote for the creation of a PIOUS news group.

References

[1] Steven A. Moyer and V. S. Sunderam. A parallel I/O system for high-performance distributed computing. In *Proceedings of the IFIP WG10.3 Working Conference on Programming Environments for Massively Parallel Distributed Systems*, pages 245–255, April 1994.

[2] Steven A. Moyer and V. S. Sunderam. PIOUS: A scalable parallel I/O system for distributed computing environments. In *Proceedings of the Scalable High-Performance Computing Conference*, pages 71–78, May 1994.

[3] Steven A. Moyer and V. S. Sunderam. Scalable concurrency control for parallel file systems. Technical Report CSTR-950202, Emory University, February 1995. Revised April 1995.

[4] V. S. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: Practice and Experience*, 2(4):315–339, December 1990.

A PIOUS Library Functions

The following is a complete description of all PIOUS library functions, presented in alphabetical order.

NAME

pious_chmod – set the file permission bits

SYNOPSIS C

int pious_chmod(char *path, pious_modet mode);

int pious_schmod(struct pious_dsvec *dsv, int dsvcnt, char *path, pious_modet mode);

SYNOPSIS FORTRAN

subroutine piousfchmod(path, mode, rc)

subroutine piousfschmod(hname, spath, lpath, dsvcnt, path, mode, rc)

integer dsvcnt, mode, rc

character*(*) hname, spath, lpath, path

DESCRIPTION

pious_chmod() sets the permission bits of regular file *path,* declustered on the default data servers, to the value specified by *mode.*

pious_schmod() performs the same action as pious_chmod(), except that the data servers on which the file is declustered are specified by the *dsvcnt* element vector *dsv.*

Corresponding Fortran functions accept argument values of type integer and character, as required, and return the result code in *rc.* In specifying a set of data servers, *hname, spath,* and *lpath* take the place of the C structure *dsv.*

See pious_open() for a discussion on valid values for arguments *mode* and *dsv (hname, spath, lpath).*

Note that pious_chmod() operates on regular PIOUS files only. To set the permission bits of a directory file use pious_chmoddir().

RETURN VALUES

Upon successful completion, a value of PIOUS_OK (0) is returned. Otherwise, a negative value is returned indicating an error condition.

ERRORS

The following error code values can be returned.

PIOUS_EINVAL
: invalid argument value, or no data server executable on host

PIOUS_EACCES
: path prefix search permission denied

PIOUS_ENOENT
: file does not exist on data servers or *path* is the empty string

PIOUS_EBUSY
: file in use; operation denied

PIOUS_ENAMETOOLONG
: path or path component name too long

PIOUS_ENOTDIR
: a component of the path prefix is not a directory

PIOUS_EPERM
: insufficient privileges to complete operation

PIOUS_EINSUF
: insufficient system resources to complete operation

PIOUS_ETPORT
: error condition in underlying transport system

PIOUS_EUNXP
: unexpected error condition encountered

PIOUS_EFATAL
: fatal error; check data server error logs

SEE ALSO

pious_open(3PIOUS), pious_chmoddir(3PIOUS)

NAME

pious_chmoddir – set directory file permission bits

SYNOPSIS C

int pious_chmoddir(char *path, pious_modet mode);

int pious_schmoddir(struct pious_dsvec *dsv, int dsvcnt, char *path, pious_modet mode);

SYNOPSIS FORTRAN

subroutine piousfchmoddir(path, mode, rc)

subroutine piousfschmoddir(hname, spath, lpath, dsvcnt, path, mode, rc)

integer dsvcnt, mode, rc

character*(*) hname, spath, lpath, path

DESCRIPTION

pious_chmoddir() sets the permission bits of directory file *path* to *mode* on all default data servers. If directory *path* does not exist on a subset of the data servers, then the directory file permission bits are set at the remaining servers.

pious_schmoddir() performs the same action as pious_chmoddir(), except that the directory file permission bits are set on the data servers specified by the *dsvcnt* element vector *dsv.*

Corresponding Fortran functions accept argument values of type integer and character, as required, and return the result code in *rc.* In specifying a set of data servers, *hname, spath,* and *lpath* take the place of the C structure *dsv.*

See pious_open() for a discussion on valid values for arguments *mode* and *dsv (hname, spath, lpath).*

RETURN VALUES

Upon successful completion, a value of PIOUS_OK (0) is returned. Otherwise, a negative value is returned indicating an error condition.

Because a single error code is returned from potentially multiple operations, the value returned is the first error encountered if any.

ERRORS

The following error code values can be returned.

PIOUS_EINVAL
: invalid argument value, or no data server executable on host

PIOUS_EACCES
: path prefix search permission denied

PIOUS_ENOENT
: file does not exist or *path* is the empty string

PIOUS_ENAMETOOLONG
: path or path component name too long

PIOUS_ENOTDIR
: a component of the path prefix is not a directory

PIOUS_EPERM
: insufficient privileges to complete operation

PIOUS_EINSUF
: insufficient system resources to complete operation

PIOUS_ETPORT
: error condition in underlying transport system

PIOUS_EUNXP
: unexpected error condition encountered

PIOUS_EFATAL
: fatal error; check data server error logs

SEE ALSO

pious_open(3PIOUS)

NAME

pious_close – close a file

SYNOPSIS C

int pious_close(int fd);

SYNOPSIS FORTRAN

subroutine piousfclose(fd, rc)

integer fd, rc

DESCRIPTION

pious_close() closes the file associated with open file descriptor *fd.*

The corresponding Fortran function accepts argument values of type integer, and returns the result code in *rc.*

RETURN VALUES

Upon successful completion, a value of PIOUS_OK (0) is returned. Otherwise, a negative value is returned indicating an error condition.

ERRORS

The following error code values can be returned.

PIOUS_EBADF
: invalid file descriptor

PIOUS_EPERM
: file accessed in active user-transaction; close not permitted

PIOUS_EINSUF
: insufficient system resources to complete operation

PIOUS_ETPORT
: error condition in underlying transport system

PIOUS_EUNXP
: unexpected error condition encountered

SEE ALSO

pious_open(3PIOUS)

NAME

pious_fstat – query file status

SYNOPSIS C

int pious_fstat(int fd, struct pious_stat *buf);

SYNOPSIS FORTRAN

subroutine piousffstat(fd, dscnt, segcnt, rc)

integer fd, dscnt, segcnt, rc

DESCRIPTION

pious_fstat() returns in buffer *buf* status information regarding the file associated with the open file descriptor *fd.*

The contents of the status buffer includes the following members:

int dscnt
the number of data servers on which the file is declustered

int segcnt
the number of file data segments

The corresponding Fortran function accepts argument values of type integer, and returns the result code in *rc.*

RETURN VALUES

Upon successful completion, a value of PIOUS_OK (0) is returned. Otherwise, a negative value is returned indicating an error condition.

ERRORS

The following error code values can be returned.

PIOUS_EBADF
invalid file descriptor

PIOUS_EINVAL
invalid *buf* argument

PIOUS_EUNXP
unexpected error condition encountered

SEE ALSO

pious_open(3PIOUS)

NAME

pious_getcwd – get the current working directory path

SYNOPSIS C

```
int pious_getcwd(char *buf, pious_sizet size);
```

SYNOPSIS FORTRAN

```
subroutine piousfgetcwd(buf, rc)
character*(*) buf
integer rc
```

DESCRIPTION

pious_getcwd() copies the current working directory path string into the buffer *buf* of size *size.*

The corresponding Fortran function accepts argument values of type integer and character, as required, and returns the result code in *rc.*

RETURN VALUES

Upon successful completion, a value of PIOUS_OK (0) is returned. Otherwise, a negative value is returned indicating an error condition.

ERRORS

The following error code values can be returned.

PIOUS_EINVAL
: invalid argument value

PIOUS_EINSUF
: buffer size is insufficient to hold cwd string

PIOUS_EUNXP
: unexpected error condition encountered

SEE ALSO

pious_setcwd(3PIOUS)

NAME

pious_lseek – seek to a specified file position

SYNOPSIS C

pious_offt pious_lseek(int fd, pious_offt offset, int whence);

SYNOPSIS FORTRAN

subroutine piousflseek(fd, offset, whence, rc)

integer fd, offset, whence, rc

DESCRIPTION

pious_lseek() sets the file pointer associated with the open file descriptor *fd* as follows:

If *whence* is PIOUS_SEEK_SET then the file pointer is set to byte *offset.*

If *whence* is PIOUS_SEEK_CUR then the file pointer is set to its current value plus *offset.*

The corresponding Fortran function accepts argument values of type integer, and returns the result code in *rc.*

Any error in updating the file pointer implies that the seek, and associated user-transaction if any, is aborted or that the PIOUS system state is inconsistent.

RETURN VALUES

Upon successful completion, a non-negative value indicating the resulting file pointer value is returned. Otherwise, a negative value is returned indicating an error condition.

ERRORS

The following error code values can be returned.

PIOUS_EBADF

fd is not a valid open descriptor

PIOUS_EINVAL

whence argument not a proper value, or resulting pointer offset would be invalid

PIOUS_EABORT

seek/user-transaction aborted normally

PIOUS_EINSUF
: insufficient system resources to complete operation

PIOUS_ETPORT
: error condition in underlying transport system

PIOUS_EUNXP
: unexpected error condition encountered

PIOUS_EFATAL
: fatal error; check data server error logs

SEE ALSO

pious_open(3PIOUS), pious_read(3PIOUS), pious_write(3PIOUS), pious_tbegin(3PIOUS), pious_tabort(3PIOUS), pious_sysinfo(3PIOUS)

NAME

pious_mkdir – create a directory file

SYNOPSIS C

int pious_mkdir(char *path, pious_modet mode);

int pious_smkdir(struct pious_dsvec *dsv, int dsvcnt, char *path, pious_modet mode);

SYNOPSIS FORTRAN

subroutine piousfmkdir(path, mode, rc)

subroutine piousfsmkdir(hname, spath, lpath, dsvcnt, path, mode, rc)

integer dsvcnt, mode, rc

character*(*) hname, spath, lpath, path

DESCRIPTION

pious_mkdir() creates directory file *path,* with permission bits set according to *mode,* on all default data servers. If directory *path* exists on a subset of the data servers, then the directory file is created at the remaining servers.

pious_smkdir() performs the same action as pious_mkdir(), except that the directory files are created on the data servers specified by the *dsvcnt* element vector *dsv.*

Corresponding Fortran functions accept argument values of type integer and character, as required, and return the result code in *rc.* In specifying a set of data servers, *hname, spath,* and *lpath* take the place of the C structure *dsv.*

See pious_open() for a discussion on valid values for arguments *mode* and *dsv (hname, spath, lpath).*

RETURN VALUES

Upon successful completion, a value of PIOUS_OK (0) is returned. Otherwise, a negative value is returned indicating an error condition.

Because a single error code is returned from potentially multiple operations, the value returned is the first error encountered other than PIOUS_EEXIST, if any.

ERRORS

The following error code values can be returned.

PIOUS_EINVAL
: invalid argument value, or no data server executable on host

PIOUS_EACCES
: path prefix search permission or write permission denied

PIOUS_EEXIST
: named file exists

PIOUS_ENOENT
: path prefix component does not exist or *path* is the empty string

PIOUS_ENAMETOOLONG
: path or path component name too long

PIOUS_ENOTDIR
: a component of the path prefix is not a directory

PIOUS_ENOSPC
: no space to extend directory or file system

PIOUS_EINSUF
: insufficient system resources to complete operation

PIOUS_ETPORT
: error condition in underlying transport system

PIOUS_EUNXP
: unexpected error condition encountered

PIOUS_EFATAL
: fatal error; check data server error logs

SEE ALSO

pious_open(3PIOUS)

NAME

pious_open – open a file for access

SYNOPSIS C

int pious_popen(char *group, char *path, int view, pious_sizet map, int faultmode, int oflag, pious_modet mode, int seg);

int pious_spopen(struct pious_dsvec *dsv, int dsvcnt, char *group, char *path, int view, pious_sizet map, int faultmode, int oflag, pious_modet mode, int seg);

int pious_open(char *path, int oflag, pious_modet mode);

int pious_sopen(struct pious_dsvec *dsv, int dsvcnt, char *path, int oflag, pious_modet mode);

SYNOPSIS FORTRAN

subroutine piousfpopen(group, path, view, map, faultmode, oflag, mode, seg, rc)

subroutine piousfspopen(hname, spath, lpath, dsvcnt, group, path, view, map, faultmode, oflag, mode, seg, rc)

subroutine piousfopen(path, oflag, mode, rc)

subroutine piousfsopen(hname, spath, lpath, dsvcnt, path, oflag, mode, rc)

integer dsvcnt, view, map, faultmode, oflag, mode, seg, rc

character*(*) hname, spath, lpath, group, path

DESCRIPTION

pious_popen() opens a file descriptor for file *path,* declustered on the default data servers, for the process in group *group.* The file is opened for access with view *view,* access mapping *map,* fault tolerance mode *faultmode,* and an access mode determined by *oflag.* If the file does not exist and *oflag* specifies file creation, then a file with *seg* data segments is created, with permission (access control) bits set according to *mode.* If the file exists and *oflag* indicates truncation, then all data segments in the file are truncated if the process is the first (regardless of group) to open the file.

pious_spopen() performs the same action as pious_popen(), except that the data servers on which the file is declustered are specified by the *dsvcnt* element vector *dsv.*

pious_open() is equivalent to pious_popen(group, path, PIOUS_INDEPENDENT, 1, PIOUS_VOLATILE, oflag, mode, seg), where *group* is a system supplied unique group name and *seg* is equal to the number of default data servers.

pious_sopen() performs the same action as pious_open(), except that the data servers on which the file is declustered are specified by the *dsvcnt* element vector *dsv.*

Corresponding Fortran functions accept argument values of type integer and character, as required, and return the result code in *rc.* In specifying a set of data servers, *hname, spath,* and *lpath* take the place of the C structure *dsv.*

The PIOUS API defines two-dimensional file objects. Each PIOUS file is composed of one or more physically disjoint data segments, where each data segment is a linear sequence of zero or more bytes. The number of data segments in a file is specified at the time of creation and does not change for the life of the file.

The argument *view* specifies one of the three supported file views: PIOUS_INDEPENDENT, PIOUS_GLOBAL, and PIOUS_SEGMENTED. In both the independent and global views, a file appears as a linear sequence of data bytes; in the independent view each process maintains a local file pointer, and in the global view all processes in a group share a single file pointer. In the segmented view, the segmented (two-dimensional) nature of the file is exposed; each process accesses a specified segment via a local pointer.

For the global and independent file views, a linear sequence of bytes is defined by ordering the bytes in a file by fixed size blocks taken round-robin from each segment; i.e. data is striped across segments. The stripe unit size, in bytes, is specified via the *map* argument.

For the segmented view, the argument *map* specifies the particular data segment that the process is to access. Data segments are number starting from zero (0).

Note that a PIOUS file view only defines an access mapping and does not alter the physical representation. Thus a file can always be opened with any view, though all processes in a group must use the same view.

PIOUS provides linearizable (and hence sequentially consistent) access to files opened under any view, guaranteeing that both data access and file pointer update are atomic with respect to concurrency.

The file fault tolerance mode is specified by the argument *faultmode* as either PIOUS_STABLE or PIOUS_VOLATILE. PIOUS_STABLE guarantees file consistency in the event of a system failure; PIOUS_VOLATILE does not. Presently, only PIOUS_VOLATILE is supported.

The *oflag* argument determines the file access mode and is the bitwise inclusive OR (addition in Fortran) of exactly one of PIOUS_RDONLY (read-only), PIOUS_WRONLY (write-only), or PIOUS_RDWR (read-write), and any combination of PIOUS_CREAT (create file) and PIOUS_TRUNC (truncate file). In specifying *oflag* for Fortran functions, each constant must appear at most once.

The argument *mode* sets the file permission (access control) bits at the time a file is created, and is specified as the bitwise inclusive OR (addition in Fortran) of any of: PIOUS_IRUSR, PIOUS_IWUSR, PIOUS_IXUSR, PIOUS_IRWXU, PIOUS_IRGRP, PIOUS_IWGRP, PIOUS_IXGRP, PIOUS_IRWXG, PIOUS_IROTH, PIOUS_IWOTH, PIOUS_IXOTH, PIOUS_IRWXO. In specifying *mode* for Fortran functions, each constant must appear at most once, and the constants PIOUS_IRWX* must not be used with other constants from the same group; e.g. (PIOUS_IRWXU + PIOUS_IRUSR) is invalid. File permission bits have identical meaning to the similarly named POSIX.1 file permission bits.

The number of data segments in a file is specified at creation time via the argument *seg.* PIOUS declusters a file, thus permitting parallel file access, by mapping file data segments to data servers in a round-robin fashion. Note that if the number of file data segments is a multiple of the number of data servers on which the file is declustered, then a linear view of the file results in an access pattern that is equivalent to traditional disk striping.

Finally, the functions pious_spopen() and pious_sopen() allow an application to directly specify the set of data servers on which a file is declustered via the *dsv* argument. Each of the *dsvcnt* elements of the *dsv* array provides the specification for a single data server via the structure members:

*char *hname*
: host name string

*char *spath*
: search root path string

*char *lpath*
: log directory path string

Entries in *dsv* are checked to insure that no duplicate hosts are specified, that no host name is null, and that all host search root and log directory paths are non-empty with '/' as the first character. A data server will be automatically spawned on each host that does not already have a data server executing.

In specifying a set of data servers for Fortran functions, the character variables *hname, spath,* and *lpath* take the place of the C structure *dsv. Hname* consists of a set of *dsvcnt* consecutive substrings,

each of length PIOUS_NAME_MAX, defining host names. *Spath* and *lpath* each consist of a set of *dsvcnt* consecutive substrings, of length PIOUS_PATH_MAX, defining search root and log directory paths, respectively. Corresponding entries in *hname, spath,* and *lpath* form the specification for a single data server; e.g. the first data server specification in the set is hname(1:PIOUS_NAME_MAX), spath(1:PIOUS_PATH_MAX), and lpath(1:PIOUS_PATH_MAX). For simplicity, *hname, spath,* and *lpath* can be defined as equivalent to character arrays of the proper length character elements.

Because PIOUS files are segmented and declustered, a system failure or unexpected denial of access during an open-create operation can result in a partial file remaining. PIOUS recognizes partial files and treats them as not extant. The remains of a partial file can be removed by simply unlinking the file via pious_unlink() or pious_sunlink(), as appropriate.

RETURN VALUES

Upon successful completion, a non-negative file descriptor is returned. Otherwise, a negative value is returned indicating an error condition.

ERRORS

The following error code values can be returned.

PIOUS_EINVAL
: invalid argument value, or PIOUS_TRUNC specified with PIOUS_RDONLY, or file does not exist and PIOUS_CREAT specified with an access mode not allowed by the permission bits, or no data server executable on host

PIOUS_EACCES
: path prefix search permission denied, or file exists and *oflag* access mode denied or *view* or *faultmode* inconsistent, or file does not exist and write permission to create denied

PIOUS_ENAMETOOLONG
: path or path component name too long

PIOUS_ENOTDIR
: a component of the path prefix is not a directory

PIOUS_ENOENT
: file does not exist on data servers and create not specified or *path* is the empty string

PIOUS_EPERM
: create operation not permitted due to *path* conflict on one or more data servers

PIOUS_ENOSPC
: no space to extend directory or file system

PIOUS_EINSUF
: file table full, or too many groups with file open, or too many processes in group, or insufficient system resources to complete operation

PIOUS_ETPORT
: error condition in underlying transport system

PIOUS_EUNXP
: unexpected error condition encountered

PIOUS_EFATAL
: fatal error; check data server error logs

SEE ALSO

pious_close(3PIOUS), pious_read(3PIOUS), pious_write(3PIOUS), pious_lseek(3PIOUS)

NAME

pious_read – read from a file

SYNOPSIS C

pious_ssizet pious_read(int fd, char *buf, pious_sizet nbyte);

pious_ssizet pious_oread(int fd, char *buf, pious_sizet nbyte, pious_offt *offset);

pious_ssizet pious_pread(int fd, char *buf, pious_sizet nbyte, pious_offt offset);

SYNOPSIS FORTRAN

subroutine piousfread(fd, buf, nbyte, rc)

subroutine piousforead(fd, buf, nbyte, offset, rc)

subroutine piousfpread(fd, buf, nbyte, offset, rc)

integer fd, nbyte, offset, rc

FortranType buf(*)

DESCRIPTION

pious_read() attempts to read *nbyte* bytes from the file associated with the open file descriptor *fd* into the buffer *buf.* Starting offset is determined by the file pointer associated with *fd.* Upon successful completion, the file pointer is incremented by the number of bytes actually read.

pious_oread() performs the same action as pious_read(), and returns the starting offset, as determined by the file pointer, in *offset.*

pious_pread() performs the same action as pious_read() except that the starting offset is determined by *offset;* the file pointer associated with *fd* remains unaffected.

Corresponding Fortran functions accept argument values of type integer, except *buf* which is an array of any data type, and return the result code in *rc.*

Any error in reading implies that the access, and associated user-transaction if any, is aborted or that the PIOUS system state is inconsistent.

Note that the contents of *buf* is undefined from the byte immediately following the last byte read up to the byte position intended for the last byte of the read request.

RETURN VALUES

Upon successful completion, a non-negative value indicating the number of bytes read is returned. Otherwise, a negative value is returned indicating an error condition.

ERRORS

The following error code values can be returned.

PIOUS_EBADF
: *fd* is not a valid descriptor open for reading

PIOUS_EINVAL
: *offset, nbyte,* or *buf* argument not a proper value or exceeds system constraints

PIOUS_EPERM
: file and user-transaction faultmode inconsistent

PIOUS_EABORT
: access/user-transaction aborted normally

PIOUS_EINSUF
: insufficient system resources to complete operation

PIOUS_ETPORT
: error condition in underlying transport system

PIOUS_EUNXP
: unexpected error condition encountered

PIOUS_EFATAL
: fatal error; check data server error logs

SEE ALSO

pious_open(3PIOUS), pious_lseek(3PIOUS), pious_tbegin(3PIOUS), pious_tabort(3PIOUS), pious_sysinfo(3PIOUS)

NAME

pious_rmdir – delete a directory file

SYNOPSIS C

int pious_rmdir(char *path);

int pious_srmdir(struct pious_dsvec *dsv, int dsvcnt, char *path);

SYNOPSIS FORTRAN

subroutine piousfrmdir(path, rc)

subroutine piousfsrmdir(hname, spath, lpath, dsvcnt, path, rc)

integer dsvcnt, rc

character*(*) hname, spath, lpath, path

DESCRIPTION

pious_rmdir() deletes directory file *path* from all default data servers. If directory *path* does not exist on a subset of the data servers, then the directory file is deleted at the remaining servers.

pious_srmdir() performs the same action as pious_rmdir(), except that the directory file is deleted from the data servers specified by the *dsvcnt* element vector *dsv.*

Corresponding Fortran functions accept argument values of type integer and character, as required, and return the result code in *rc.* In specifying a set of data servers, *hname, spath,* and *lpath* take the place of the C structure *dsv.*

See pious_open() for a discussion on valid values for the *dsv (hname, spath, lpath)* argument(s).

RETURN VALUES

Upon successful completion, a value of PIOUS_OK (0) is returned. Otherwise, a negative value is returned indicating an error condition.

Because a single error code is returned from potentially multiple operations, the value returned is the first error encountered other than PIOUS_ENOENT, if any.

ERRORS

The following error code values can be returned.

PIOUS_EINVAL
: invalid argument value, or no data server executable on host

PIOUS_EACCES
: path prefix search permission or write permission denied

PIOUS_EBUSY
: directory in use

PIOUS_ENOTEMPTY
: directory not empty

PIOUS_ENOENT
: directory file does not exist or *path* is the empty string

PIOUS_ENAMETOOLONG
: path or path component name too long

PIOUS_ENOTDIR
: a component of the path prefix is not a directory

PIOUS_EINSUF
: insufficient system resources to complete operation

PIOUS_ETPORT
: error condition in underlying transport system

PIOUS_EUNXP
: unexpected error condition encountered

PIOUS_EFATAL
: fatal error; check data server error logs

SEE ALSO

pious_open(3PIOUS)

NAME

pious_setcwd – set the current working directory path

SYNOPSIS C

int pious_setcwd(char *path);

SYNOPSIS FORTRAN

subroutine piousfsetcwd(path, rc)

integer rc

character*(*) path

DESCRIPTION

pious_setcwd() sets the current working directory path string for the calling process to *path.* The current working directory string is automatically concatenated, as a prefix, with all file pathnames to form the full file pathname string.

The corresponding Fortran function accepts argument values of type integer and character, as required, and returns the result code in *rc.*

Note that the current working directory path string is not checked for correctness or accessability until used to resolve a file pathname as discussed above.

By default, the current working directory path string is the null string.

RETURN VALUES

Upon successful completion, a value of PIOUS_OK (0) is returned. Otherwise, a negative value is returned indicating an error condition.

ERRORS

The following error code values can be returned.

PIOUS_EINVAL
invalid *path* argument

PIOUS_EINSUF
insufficient system resources to complete operation

PIOUS_EUNXP
unexpected error condition encountered

SEE ALSO

pious_getcwd(3PIOUS)

NAME

pious_shutdown – shutdown PIOUS

SYNOPSIS C

int pious_shutdown(void);

SYNOPSIS FORTRAN

subroutine piousfshutdown(rc)

integer rc

DESCRIPTION

pious_shutdown() requests that PIOUS be shutdown, causing all data servers and the service coordinator to complete and exit. PIOUS will only shutdown if all files open by all processes have been closed.

The corresponding Fortran function accepts argument values of type integer, and returns the result code in *rc*.

Successful completion of the pious_shutdown() operation does not guarantee that the system has been shutdown, only that a shutdown request has been successfully issued.

Note that it is not necessary to shutdown PIOUS via pious_shutdown() in order to maintain file consistency; this is a clean-up operation only.

RETURN VALUES

Upon successful completion, a value of PIOUS_OK (0) is returned. Otherwise, a negative value is returned indicating an error condition.

ERRORS

The following error code values can be returned.

PIOUS_EPERM
: calling process has open files or a user-transaction in progress

PIOUS_EINSUF
: insufficient system resources to complete operation

PIOUS_ETPORT
: error condition in underlying transport system

PIOUS_EUNXP
unexpected error condition encountered

SEE ALSO
pious_close(3PIOUS)

NAME

pious_sysinfo – query system configuration/status

SYNOPSIS C

long pious_sysinfo(int name);

SYNOPSIS FORTRAN

subroutine piousfsysinfo(name, rc)

integer name, rc

DESCRIPTION

pious_sysinfo() returns the system configuration/status information indicated by *name*.

The corresponding Fortran function accepts argument values of type integer, and returns the result code in *rc*.

Valid system parameters to query include:

PIOUS_DS_DFLT
: number of default data servers

PIOUS_OPEN_MAX
: maximum number of open file descriptors for a given process

PIOUS_TAG_LOW
: lower bound of message tag range reserved for system

PIOUS_TAG_HIGH
: upper bound of message tag range reserved for system

PIOUS_BADSTATE
: true (1) if system state inconsistent, false (0) otherwise

Note that the reserved message tag range is only of interest in metacomputing environments that do not support message contexts, such as PVM.

RETURN VALUES

Upon successful completion, a non-negative integer representing the system parameter value is returned. Otherwise, a negative value is returned indicating an error condition.

ERRORS

The following error code values can be returned.

PIOUS_EINVAL
: invalid *name* argument

PIOUS_EINSUF
: insufficient system resources to complete operation

PIOUS_ETPORT
: error condition in underlying transport system

PIOUS_EUNXP
: unexpected error condition encountered

SEE ALSO

pious_fstat(3PIOUS)

NAME

pious_tabort – abort a user-transaction

SYNOPSIS C

```
int pious_tabort(void);
```

SYNOPSIS FORTRAN

```
subroutine piousftabort(rc)
integer rc
```

DESCRIPTION

pious_tabort() aborts a user-transaction. If successful, the effects of all file write and seek updates performed within the context of the user-transaction are undone.

The corresponding Fortran function accepts argument values of type integer, and returns the result code in *rc*.

Any error implies that the PIOUS system state is inconsistent.

RETURN VALUES

Upon successful completion, a value of PIOUS_OK (0) is returned. Otherwise, a negative value is returned indicating an error condition.

ERRORS

The following error code values can be returned.

PIOUS_EINSUF
: insufficient system resources to complete operation

PIOUS_ETPORT
: error condition in underlying transport system

PIOUS_EUNXP
: unexpected error condition encountered

PIOUS_EFATAL
: fatal error; check data server error logs

SEE ALSO

pious_tbegin(3PIOUS), pious_tend(3PIOUS), pious_sysinfo(3PIOUS)

NAME

pious_tbegin – begin a user-transaction

SYNOPSIS C

int pious_tbegin(int faultmode);

SYNOPSIS FORTRAN

subroutine piousftbegin(faultmode, rc)

integer faultmode, rc

DESCRIPTION

pious_tbegin() marks the beginning of a user-transaction. All read, write, and seek accesses performed after a successful pious_tbegin() and prior to a pious_tend() or pious_tabort() are atomic with respect to concurrency. Any number of files can be accessed within the context of a user-transaction, and any PIOUS function can be called, though certain actions may be denied.

The user-transaction fault tolerance mode is specified by the argument *faultmode* as either PIOUS_STABLE or PIOUS_VOLATILE. PIOUS_STABLE guarantees file consistency in the event of a system failure; PIOUS_VOLATILE does not. Presently, only PIOUS_VOLATILE is supported.

Note that within the context of a user-transaction, the fault tolerance mode of any file accessed, as specified by pious_open(), must be the same as that of the user-transaction.

The corresponding Fortran function accepts argument values of type integer, and returns the result code in *rc*.

RETURN VALUES

Upon successful completion, a value of PIOUS_OK (0) is returned. Otherwise, a negative value is returned indicating an error condition.

ERRORS

The following error code values can be returned.

PIOUS_EINVAL
invalid *faultmode* argument

PIOUS_EPERM

user-transaction already in progress

PIOUS_EUNXP

unexpected error condition encountered

SEE ALSO

pious_tabort(3PIOUS), pious_tend(3PIOUS), pious_open(3PIOUS), pious_read(3PIOUS), pious_write(3PIOUS), pious_lseek(3PIOUS)

NAME

pious_tend – end a user-transaction

SYNOPSIS C

int pious_tend(void);

SYNOPSIS FORTRAN

subroutine piousftend(rc)

integer rc

DESCRIPTION

pious_tend() completes a user-transaction. If successful, all file updates are installed atomically, in accordance with the fault tolerance mode specified by the corresponding pious_tbegin(), and become visible.

The corresponding Fortran function accepts argument values of type integer, and returns the result code in *rc*.

Any error implies that the user-transaction is aborted or that the PIOUS system state is inconsistent.

RETURN VALUES

Upon successful completion, a value of PIOUS_OK (0) is returned. Otherwise, a negative value is returned indicating an error condition.

ERRORS

The following error code values can be returned.

PIOUS_EABORT
: user-transaction aborted normally

PIOUS_EINSUF
: insufficient system resources to complete operation

PIOUS_ETPORT
: error condition in underlying transport system

PIOUS_EUNXP
: unexpected error condition encountered

PIOUS_EFATAL
fatal error; check data server error logs

SEE ALSO

pious_tbegin(3PIOUS), pious_tabort(3PIOUS), pious_sysinfo(3PIOUS)

NAME

pious_umask – set the file mode creation mask

SYNOPSIS C

int pious_umask(pious_modet cmask, pious_modet *prevcmask);

SYNOPSIS FORTRAN

subroutine piousfumask(cmask, prevcmask, rc)

integer cmask, prevcmask, rc

DESCRIPTION

pious_umask() sets the file mode creation mask for the calling process to *cmask,* returning the previous value of the mask in *prevcmask.* Only the file permission bits of *cmask* are used, all others are ignored.

The corresponding Fortran function accepts argument values of type integer, and returns the result code in *rc.*

The file mode creation mask is used by the pious_open() and pious_mkdir() functions to turn off permission bits in the *mode* argument. Permission bits set in *cmask* are cleared for a file to be created. See pious_open() for a discussion on file permission bit values.

By default, the file mode creation mask is zero (0).

RETURN VALUES

Upon successful completion, a value of PIOUS_OK (0) is returned. Otherwise, a negative value is returned indicating an error condition.

ERRORS

The following error code values can be returned.

PIOUS_EINVAL
: invalid *prevcmask* argument

PIOUS_EUNXP
: unexpected error condition encountered

SEE ALSO

pious_open(3PIOUS), pious_mkdir(3PIOUS)

NAME

pious_unlink – delete a file

SYNOPSIS C

int pious_unlink(char *path);

int pious_sunlink(struct pious_dsvec *dsv, int dsvcnt, char *path);

SYNOPSIS FORTRAN

subroutine piousfunlink(path, rc)

subroutine piousfsunlink(hname, spath, lpath, dsvcnt, path, rc)

integer dsvcnt, rc

character*(*) hname, spath, lpath, path

DESCRIPTION

pious_unlink() deletes the regular file *path* declustered on the default data servers.

pious_sunlink() performs the same action as pious_unlink(), except that the data servers on which the file is declustered are specified by the *dsvcnt* element vector *dsv.*

Corresponding Fortran functions accept argument values of type integer and character, as required, and return the result code in *rc.* In specifying a set of data servers, *hname, spath,* and *lpath* take the place of the C structure *dsv.*

See pious_open() for a discussion on valid values for the *dsv (hname, spath, lpath)* argument(s).

Because PIOUS files are segmented and declustered, a system failure or unexpected denial of access during an unlink operation can result in a partial file remaining. PIOUS recognizes partial files and treats them as not extant. The remains of a partial file can be removed by simply re-unlinking the file.

Note that pious_unlink() only operates on regular PIOUS files. To delete a directory file use pious_rmdir().

RETURN VALUES

Upon successful completion, a value of PIOUS_OK (0) is returned. Otherwise, a negative value is returned indicating an error condition.

ERRORS

The following error code values can be returned.

PIOUS_EINVAL
: invalid argument value, or no data server executable on host

PIOUS_EACCES
: path prefix search permission or write permission denied or insufficient privileges to perform operation

PIOUS_ENOENT
: file does not exist on data servers or *path* is the empty string

PIOUS_EBUSY
: file in use; operation denied

PIOUS_ENAMETOOLONG
: path or path component name too long

PIOUS_ENOTDIR
: a component of the path prefix is not a directory

PIOUS_EINSUF
: insufficient system resources to complete operation

PIOUS_ETPORT
: error condition in underlying transport system

PIOUS_EUNXP
: unexpected error condition encountered

PIOUS_EFATAL
: fatal error; check data server error logs

SEE ALSO

pious_open(3PIOUS), pious_rmdir(3PIOUS)

NAME

pious_write – write to a file

SYNOPSIS C

pious_ssizet pious_write(int fd, char *buf, pious_sizet nbyte);

pious_ssizet pious_owrite(int fd, char *buf, pious_sizet nbyte, pious_offt *offset);

pious_ssizet pious_pwrite(int fd, char *buf, pious_sizet nbyte, pious_offt offset);

SYNOPSIS FORTRAN

subroutine piousfwrite(fd, buf, nbyte, rc)

subroutine piousfowrite(fd, buf, nbyte, offset, rc)

subroutine piousfpwrite(fd, buf, nbyte, offset, rc)

integer fd, nbyte, offset, rc

FortranType buf(*)

DESCRIPTION

pious_write() attempts to write *nbyte* bytes to the file associated with the open file descriptor *fd* from the buffer *buf.* Starting offset is determined by the file pointer associated with *fd.* Upon successful completion, the file pointer is incremented by the number of bytes actually written.

pious_owrite() performs the same action as pious_write(), and returns the starting offset, as determined by the file pointer, in *offset.*

pious_pwrite() performs the same action as pious_write() except that the starting offset is determined by *offset;* the file pointer associated with *fd* remains unaffected.

Corresponding Fortran functions accept argument values of type integer, except *buf* which is an array of any data type, and return the result code in *rc.*

Any error in writing implies that the access, and associated user-transaction if any, is aborted or that the PIOUS system state is inconsistent.

Note that the contents of the file is undefined from the byte immediately following the last byte written up to the byte position intended for the last byte of the write request.

RETURN VALUES

Upon successful completion, a non-negative value indicating the number of bytes written is returned. Otherwise, a negative value is returned indicating an error condition.

ERRORS

The following error code values can be returned.

PIOUS_EBADF
: *fd* is not a valid descriptor open for writing

PIOUS_EINVAL
: *offset, nbyte,* or *buf* argument not a proper value or exceeds system constraints

PIOUS_EPERM
: file and user-transaction faultmode inconsistent

PIOUS_EABORT
: access/user-transaction aborted normally

PIOUS_EINSUF
: insufficient system resources to complete operation

PIOUS_ETPORT
: error condition in underlying transport system

PIOUS_EUNXP
: unexpected error condition encountered

PIOUS_EFATAL
: fatal error; check data server error logs

SEE ALSO

pious_open(3PIOUS), pious_lseek(3PIOUS), pious_tbegin(3PIOUS), pious_tabort(3PIOUS), pious_sysinfo(3PIOUS)