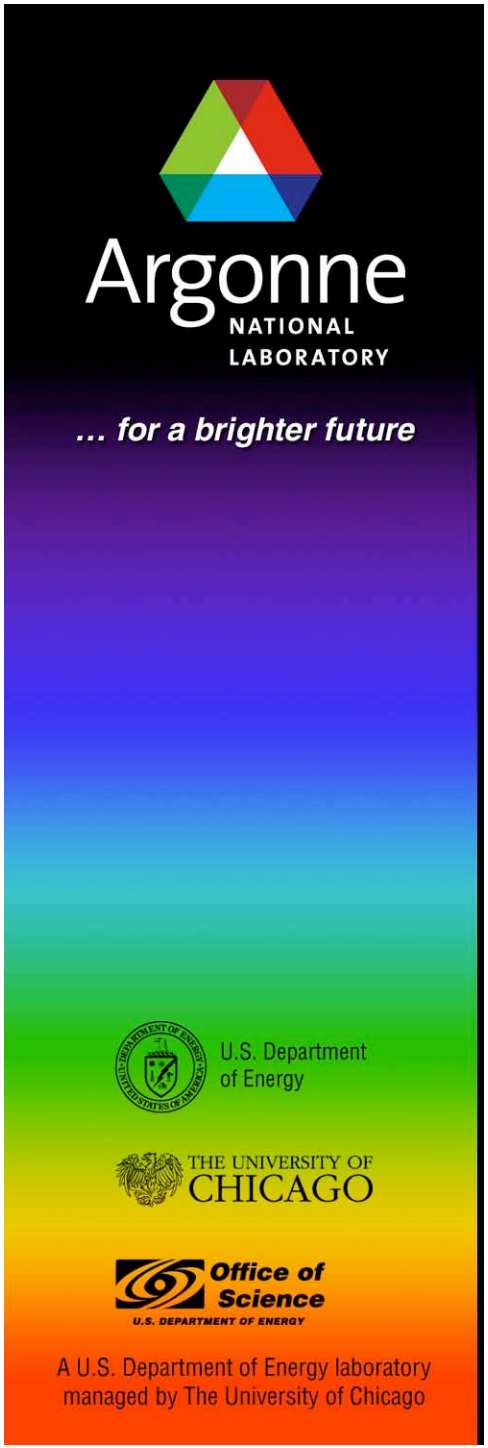


HEC POSIX I/O API Extensions

Rob Ross

Mathematics and Computer Science Division

Argonne National Laboratory

rross@mcs.anl.gov

(Thanks to Gary Grider for providing much of the material for this talk!)

POSIX Introduction

- POSIX is the IEEE Portable Operating System Interface for Computing Environments.
- “POSIX defines a standard way for an application program to obtain basic services from the operating system”
- POSIX was created when a single computer owned its own file system.
  - Network file systems like NFS chose not to implement strict POSIX semantics in all cases (e.g., lazy access time propagation)
  - Heavily shared files (e.g., from clusters) can be very expensive for file systems that provide POSIX semantics, or have undefined contents for file systems that bend the rules
- The Open Group (http://www.opengroup.org/) is responsible for the specification and any subsequent extensions.

APIs for HEC I/O

- POSIX IO APIs (open, close, read, write, stat) have semantics that can make it hard to achieve high performance when large clusters of machines access shared storage.
- A working group of HEC users is drafting some proposed API additions for POSIX that will provide standard ways to achieve higher performance.
- Primary approach is either to relax semantics that can be expensive, or to provide more information to inform the storage system about access patterns.
- The goal is to create a standard way to provide high performance and good semantics
- Three components:
  - **Good concepts** - building blocks for more effective I/O systems
  - **API definition and standardization** - well-defined and capable interfaces to use these ideas agreed upon by the community
  - **Implementations** - early prototypes to show viability, adoption in OSes and file systems to provide availability

Contributors

- Lee Ward - Sandia National Lab
- Bill Lowe, Tyce McLarty – Lawrence Livermore National Lab
- Gary Grider, James Nunez – Los Alamos National Lab
- Rob Ross, Rajeev Thakur, William Gropp, Murali Vilayannur - Argonne National Lab
- Roger Haskin – IBM
- Brent Welch, Marc Unangst - Panasas
- Garth Gibson - Carnegie Mellon University/Panasas
- Alok Choudhary – Northwestern University
- Tom Ruwart - University of Minnesota/IO Performance
- Harriet Coverston - Sun Microsystems
- Others …

Current HEC POSIX Enhancement Areas

- Current “first string”:
  - Flexible (if not concise) description of I/O operations
    - *readx(), writex()*
  - Metadata (lazy attributes, aggregation)
    - *statlite() and friends*
    - *readdirplus() and friends*
  - Coherence – (last writer wins and other such things can be optional)
    - *O_LAZY, lazyio_propagate(), lazyio_synchronize()*
  - Efficient name resolution and file open (group file opens)
    - *openg(), openfh()*
- Group locks, ACLs, QoS, and portable hinting are being investigated as well, but I will focus on the first string.

readx, writex - Efficient I/O Description

- **Syntax**

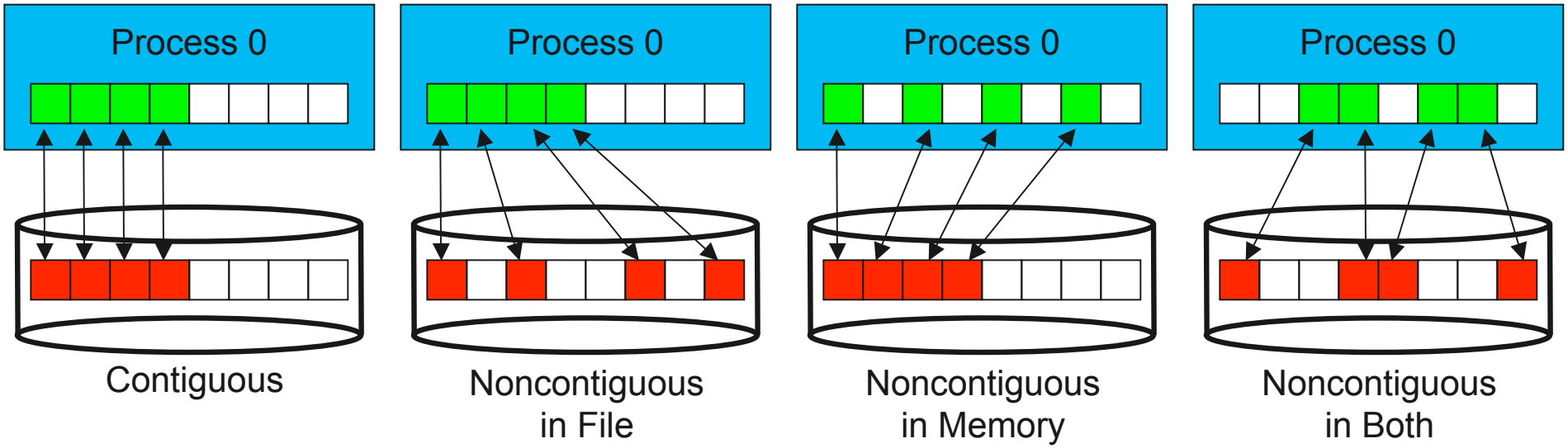
  ssize_t **readx**(int fd, const struct iovec *iov, size_t iov_count, struct xtvec *xtv, size_t xtv_count);

  ssize_t **writex**(int fd, const struct iovec *iov, size_t iov_count, struct xtvec *xtv, size_t xtv_count);

  struct xtvec { off_t xtv_off; /* Starting file offset */
  size_t xtv_len; /* Number of bytes */ };

- **Description**
  - Generalized file vector to memory vector transfer. Existing **readv**(), **writev**() specify a memory vector and do serial IO. The new **readx**(), **writex**() calls also read/write strided vectors to/from files, but regions may be processed in any order, and iov and xtv need not have the same number of elements.
  - The **readx**() function reads xtv_count blocks described by xtv from the file associated with the file descriptor fd into the iov_count multiple buffers described by iov. The file offset is not changed.
  - The **writex**() function writes at most xtv_count blocks described by xtv into the file associated with the file descriptor fd from the iov_count multiple buffers described by iov. The file offset is not changed.

Impact of readx and writex



- Patterns that are noncontiguous in memory and/or file are all supported
- Underlying implementation may choose to process the regions in any order
- Results in error cases still being ironed out…

statlite, fstatlite, lstatlite - Lazy Attributes

- **Syntax**

  int **statlite**(const char **file_name*, struct statlite **buf*);
  int **fstatlite**(int *filedes*, struct statlite **buf*);
  int **lstatlite**(const char **file_name*, struct statlite **buf*);

- **Description**
  - This family of stat calls, the lite family, is provided to allow for file I/O performance not to be compromised by frequent use of stat information lookup. Some information can be expensive to obtain when a file is busy.
  - They all return a *stat* structure, which has all the normal fields from the stat family of calls but some of the fields (e.g., file size, modify time) are optionally not guaranteed to be correct.
  - There is a litemask field that can be used to specify which of the optional fields you require to be completely correct values returned.
  - **statlite** stats the file pointed to by *file_name* and fills in *buf*.
  - **fstatlite** is identical to **stat**, only the open file pointed to by *filedes* (as returned by **open**(2)) is statlited-ed in place of *file_name*.

statlite Data Structure

```
struct statlite {
    dev_t           st_dev;      /* device */
    ino_t           st_ino;      /* inode */
    mode_t          st_mode;     /* protection */
    nlink_t         st_nlink;    /* number of hard links */
    uid_t           st_uid;      /* user ID of owner */
    gid_t           st_gid;      /* group ID of owner */
    dev_t           st_rdev;     /* device type (if inode device)*/
    unsigned long   st_litemask; /* bit mask for optional fields */
    /*************************************************************/
    /**** Remaining fields are optional according to st_litemask ***/
    off_t           st_size;     /* total size, in bytes         */
    blksize_t       st_blksize;  /* blocksize for filesystem I/O */
    blkcnt_t        st_blocks;   /* number of blocks allocated   */
    time_t          st_atime;    /* time of last access          */
    time_t          st_mtime;    /* time of last modification    */
    time_t          st_ctime;    /* time of last change          */
};
```

readdirplus, readdirlite - Aggregating Metadata Operations

- **Syntax**

  struct dirent_plus ***readdirplus**(DIR **dirp*);

  int **readdirplus_r**(DIR **dirp*, struct dirent_plus ***entry*, struct dirent_plus ****result*);

  struct dirent_lite ***readdirlite**(DIR **dirp*);

  int **readdirlite_r**(DIR **dirp*, struct dirent_lite ***entry*, struct dirent_lite ****result*);

- **Description**
  - This family of calls is provided to all the file system to return file metadata as part of the directory read process. This as a side-effect aggregates many stat operations together.
  - **readdirplus**(2) and **readdirplus_r**(2) return a directory entry plus **lstat**(2) results (like the NFSv3 READDIRPLUS command)
  - **readdirlite**(2) and **readdirlite_r**(2) return a directory entry plus **lstatlite**(2) results

readdirplus Data Structures

```
struct dirent_plus {
    struct dirent   d_dirent;  /* dirent struct for this entry */
    struct stat     d_stat;    /* attributes for this entry */
    int             d_stat_err;
};
```

- Stat structure embedded with the directory entries
- Separate error value corresponds to stat operation

O_LAZY, lazyio_propagate, lazyio_synchronize - Coherence

- **Syntax**

  Specify O_LAZY in *flags* argument to **open**(2)

  int lazyio_propagate(int *fd*, **off_t** *offset*, **size_t** *count*);

  int lazyio_synchronize(int *fd*, **off_t** *offset*, **size_t** *count*);

- **Description**
  - Requests lazy I/O data integrity. Allows network filesystem to relax data coherency requirements to improve performance for shared-write file. This is a hint only: if filesystem does not support lazy I/O integrity, does not have to do anything differently.
  - Writes may not be visible to other processes or clients until **lazyio_propagate**(2), **fsync**(2), or **close**(2) is called
  - Reads may come from local cache (ignoring changes to file on backing storage) until **lazyio_synchronize**(2) is called
  - Does not provide synchronization across processes or nodes – program must use external synchronization (e.g., pthreads, MPI, etc.) to coordinate actions.

openg, openfh - Name Space Traversal and Collective File Open

- **Syntax**

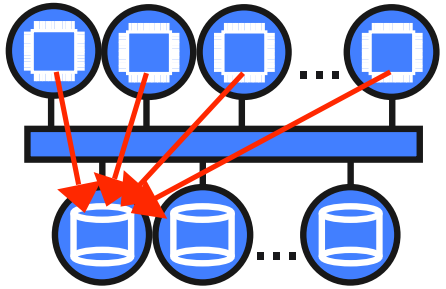
  int **openg**(char *path, int mode, fh_t *handle);

  int **openfh**(fh_t *fh);

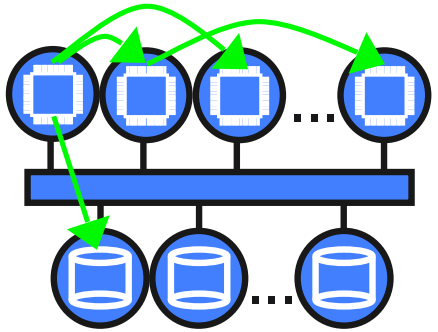
- **Description**
  - The **openg**() function opens a file named by path according to mode (e.g., O_RDWR). It returns an opaque file handle corresponding to a file descriptor. The intent is that the file handle can be transferred to cooperating processes and converted to a file descriptor with **openfh**().
  - The **openfh**() function shall create an open file descriptor that refers to the file represented by the *fh* argument. The file status flags and file access modes of the open file description shall be set according to those given in the accompanying **openg**().
  - The lifetime of the file handle is implementation specific. For example, it may not be valid once all open file descriptors derived from the handle with **openfh**() have been closed.

Impact of openg and openfh

- Calls are primarily designed to aid in efficient implementation of collective open operations (e.g. MPI_File_open)
- Rely on external communication to transfer opaque file handle to other processes
- In some cases, additional processes perform no communication with file system to create open file descriptor



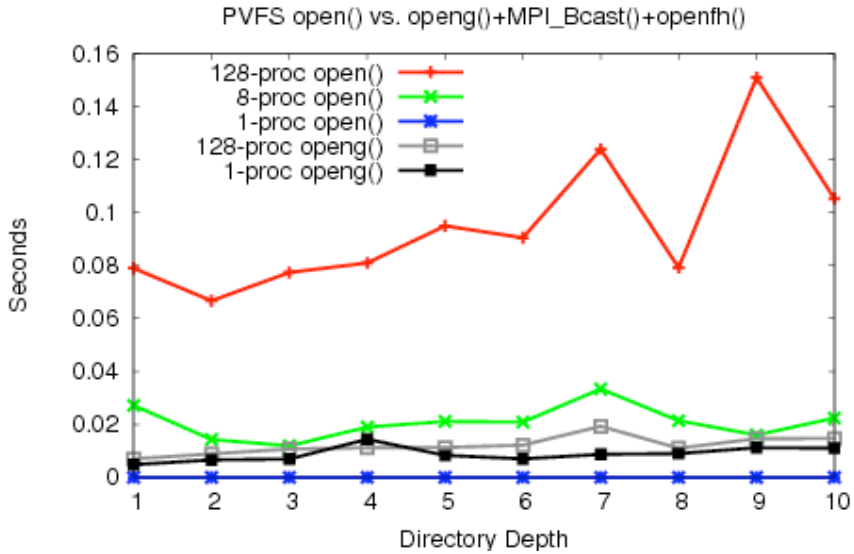
Standard POSIX open model forces all processes to open a file, causing a storm of system calls.



A single openg provides a handle that is then broadcast to the remaining processes, who call openfh.

openg and openfh Prototyped

- Here we compare time for N processes to perform independent open calls versus the openg + MPI_Bcast + N * openfh sequence
- On this system, openg/openfh become a win very quickly
  - Less expensive to “collectively open” with 128 processes than to independently open with 8!



Data from Ruth Klundt (SNL), using Darkstar cluster.

Current Status and Contact Information

- Ideas
  - Group has identified short-term and long-term goals for improvements
- Interface Specification
  - HEC Extensions working group formed with Open Group
  - Draft 0 specification nearing completion, includes calls discussed here
- Implementations
  - Prototypes of many calls have been implemented by ANL, UCSC, Sun, CFS/Cray, etc.
  - Source for many of these will be made available soon
- Go to the POSIX HPC I/O Extensions Web site for more information: www.pdl.cmu.edu/posix/