Open Source and UNIX portability

Chinmay Ghosh

OpenVMS Engineering

Agenda

- Shared Stream IO (SSIO)
- PIPE
- BASH
- Miscellaneous

Shared Stream IO (SSIO)

SETTING THE CONTEXT

File System I/O

- **Programs use file system APIs for file I/O**
- **OpenVMS traditional file system APIs**

  Record: SYS$OPEN, SYS$GET, SYSPUT, SYSCLOSE

  Low level: IO$_ACCESS, IO$_READVBLK, etc

- **OpenVMS supports POSIX APIs too**
- **POSIX APIs provided by library – CRTL**
  - open(), read(), write(), fsync(), close(), etc.
- **CRTL uses OpenVMS native file system APIs**

WHAT IS THE PROBLEM?

The problem

- **On OpenVMS, concurrent POSIX write() calls to the same file can corrupt data**
  - POSIX I/O on OpenVMS not atomic
    - Data updates can get lost
    - Disk can get mixed data from overlapping writes
- **Consistency not guaranteed for files opened for shared write**
- **Victims: UNIX programs ported to OpenVMS**
- **Programs must provide atomicity on their own**
- ***Stated formally:***
  - *OpenVMS does not provide POSIX-compliant shared read/write to byte stream files*

Atomicity: OpenVMS and POSIX (1/2)

- **OpenVMS is <u>record</u>-atomic**
- **POSIX is <u>byte-stream</u>-atomic**
- **Block I/O is not atomic**
  - Ultimately, all disk I/O is done this way
  - *<u>Caller</u>* (file system) expected to manage concurrency

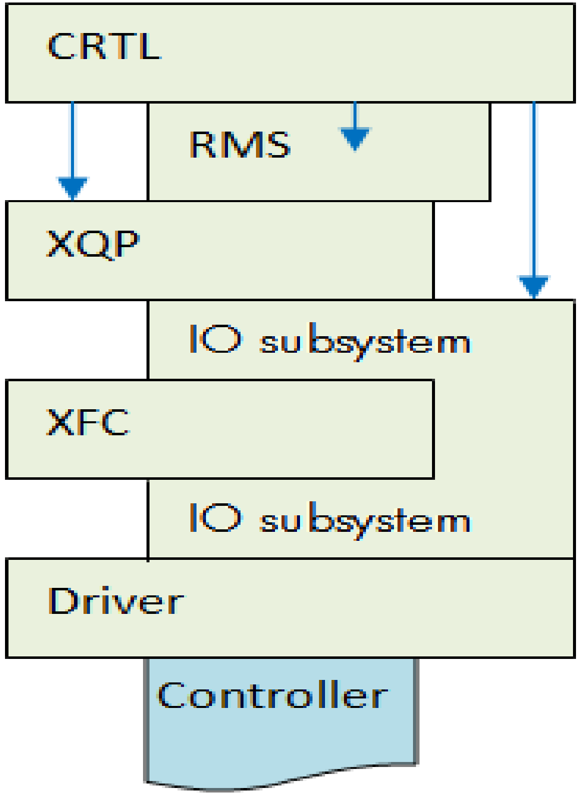
	Record I/O	Byte-stream I/O	Block I/O
OpenVMS	Atomic	N/A	Not atomic
POSIX	N/A	Atomic	Not atomic

Atomicity: OpenVMS and POSIX (2/2)

- **Byte-range I/O on UNIX**
  - UNIX FS converts byte-stream I/O to block I/O
  - Provides atomicity, designed for this
- **Record I/O via RMS**
  - RMS converts record I/O to block I/O
  - Provides atomicity, designed for this
- **POSIX I/O on OpenVMS**
  - CRTL converts byte-stream I/O to block I/O
  - Design not geared to provide atomicity
  - Buffers not system-wide or cluster-wide

OpenVMS file system layering

- **CRTL**
  - victim of no-synch-for-block-I/O
- **RMS**
  - provides synch transparently, but for record I/O only!
- **XQP**
  - basic synch, user program must still do some synch
- **XFC**
  - no API, no synch, parallel writes can mix
- **IO subsystem**
  - no synch, parallel writes can mix



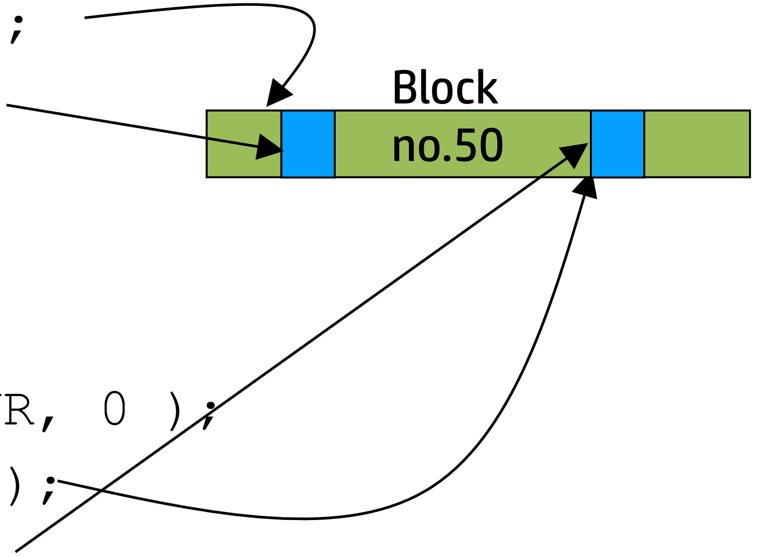
Example victim program

Process A:

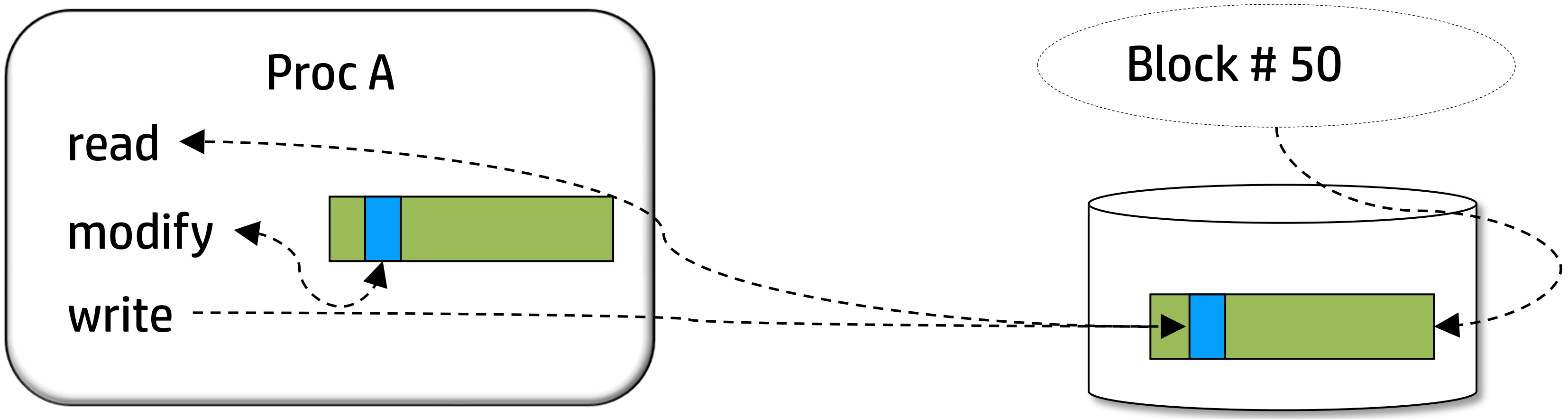
```
fd = open( "x.dat", O_RDWR, 0 );
lseek( fd, 10, SEEK_SET );
write( fd, data1, 20 );
```

Process B:

```
fd = open( "x.dat", O_RDWR, 0 );
lseek( fd, 400, SEEK_SET );
write( fd, data2, 20 );
```



Block I/O: Read-modify-write sequence

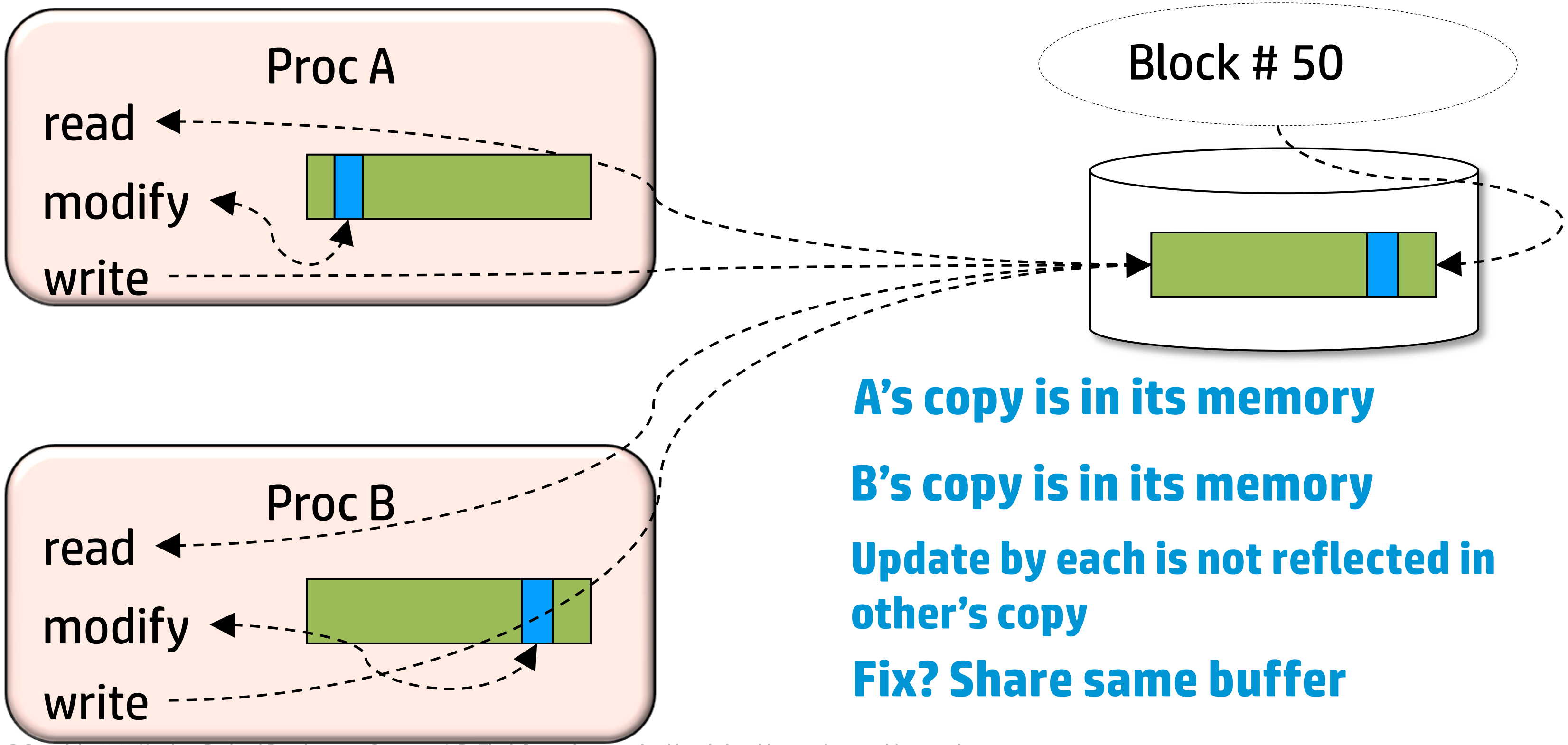


I/O is done in units of blocks, not bytes

To modify part of block:

- Read whole block (green)
- Modify desired bytes (blue)
- Write whole block (blue + green)

Block I/O: Lost update problem



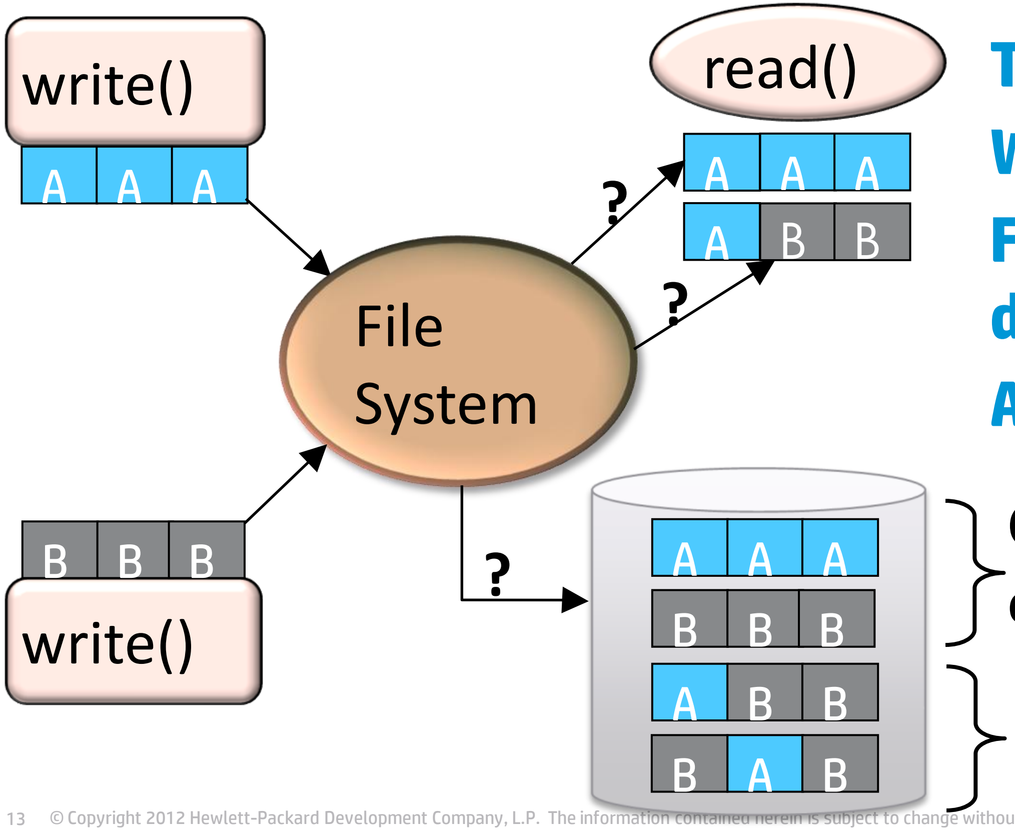
A's copy is in its memory

B's copy is in its memory

Update by each is not reflected in other's copy

Fix? Share same buffer

Block I/O: Mixed data problem



Two processes do write

Write blocks 50, 51, 52

File system updates blocks on disk

A third process reads

Examples of Proc A and Proc B

- **Competing writers**
  - E.g. transaction processing, database system
  - Processes A and B attempt to update same ‘record’
- **Workers with common parent**
  - ‘forked’ by common parent (e.g. smbd)
  - Proc A writes to file; Proc B reads from same file
- **Parent – child**
  - Processes A and B append to same log file via same FD
  - Each is affected by the other’s EOF update

Impact of the problem

- **More effort porting UNIX program to OpenVMS**
  - Extra coding by programmer to assure data integrity

- **Performance is lower**
  - Extra code executed for synchronization
  - Extra I/O done to disk – frequent calls to fflush

- **Spend extra effort and get a slower program!**

- **One of the blockers for a conforming UNIX fork**
  - Parent – child sharing same FD

Impact: Specific examples

- **Java (CIFS too) uses a work-around**
  - Does open+read/write+close for every read/write!
  - Restores current file offset after each close+open
  - Significant performance issue
- **Oracle problem with log and trace files**
  - Single writer with multiple readers
- **Apache's use of log files sub-optimal**
  - V1.3 places restriction
  - V2.0 uses a work-around

Key learnings

On OpenVMS application is responsible to provide atomicity for block I/O

- OpenVMS doesn’t guarantee atomicity for block I/O

Lost update problem:

- Process I/O buffers must be shared system-wide to avoid lost updates

Mixed data problem:

- Programs doing block I/O must synch among themselves to prevent mixed data

Today’s solutions:

- Flush after every write
- Exclusively-lock file when doing write

Solution - SSIO

How SSIO solves the problem

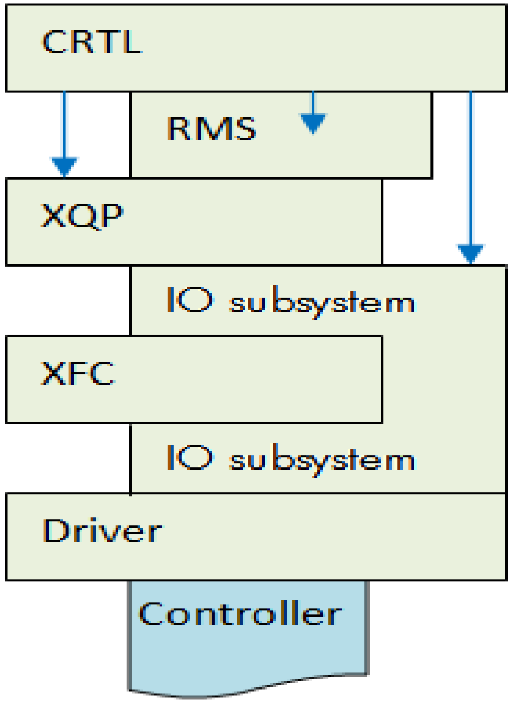
- **Lost update problem:**
  - XFC provides the new API
  - Programs pass byte-offset via new API
  - New code in XFC to update only part of a block
- **Mixed data problem:**
  - XFC will lock all affected buffers during block I/O
  - Atomic up to SSIO_MAX_ATOMIC_IO bytes

SSIO components

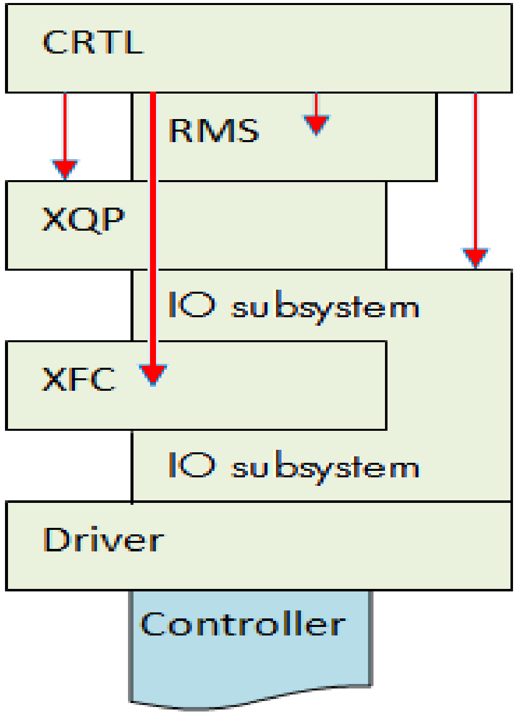
- **XFC**
  - Excellent buffer management, enhanced for byte-range
  - Would provide new, byte-range I/O API
  - Existing code for native OpenVMS I/O remains unchanged
- **CRTL**
  - Would call new XFC API to do byte-range I/O
- **RMS, XQP**
  - Minor, necessary changes to support SSIO operations
    - RMS: SYS$OPEN, etc
    - XQP: IO$_ACCESS, etc
  - Supports current APIs with no behavior change

Current and proposed designs

Current design



SSIO design



Additional benefit: Performance

- **XFC could also provide performance boost**
  - Dirty data caching to avoid frequent writes
  - Append optimization
  - Caching dirty data after file close
  - Fine-grained locking

What remains unchanged

- **Existing APIs, options remain unchanged**
- **Applications using RMS, QIO APIs**
  - Will not need any code changes
  - Will not see any behavior changes
- **Applications using CRTL (POSIX) API**
  - Will continue to work without code changes
  - Will run faster with new CRTL, with extra synch code removed

SSIO – V1.0 (Beta) release (1/2)

- **Data consistency is guaranteed**
  - For shared access to non overlapping byte boundaries with in the same block
- **Standalone implementation**
- **Write though cache**
- **Impacted CRTL APIs**
  - open(), create(), read(), write(), lseek(), Fcntl(), truncate(), ftruncate(), fsync()
- **Supported record formats**
  - STREAM, STREAM_CR, STREAM_LF, UNDEFINED
- **To Enable SSIO**
  - Use logical DECC$SSIO
  - Use argument "fop=ssio“ with open() or create()

SSIO – V1.0 (Beta) release (2/2)

- **Requirements**
  - XFC Caching has to be enabled
  - SSIO mode should not be mixed with NON SSIO mode

- **Restrictions**
  - files to be opened and accessed in shared mode
  - Define DECC$FILE_SHARING 1
  - Use "shr=val,val,…" in create() and open() call
  - Specify fop="ssio,cbt" in create() and open()

SSIO – upcoming release

- Cluster aware
- Performance Improvement

Benefits

- **Porting becomes easier technically**
  - No writing of extra code to assure data integrity
- **Customers get Open Source products quicker**
  - New product versions can be ported faster
- **Faster performance of POSIX products**
  - Java, Oracle
  - CIFS, CSWS, GNV, etc
- **Reduced porting cost to HP, partners**
  - Lesser time, skills for porting

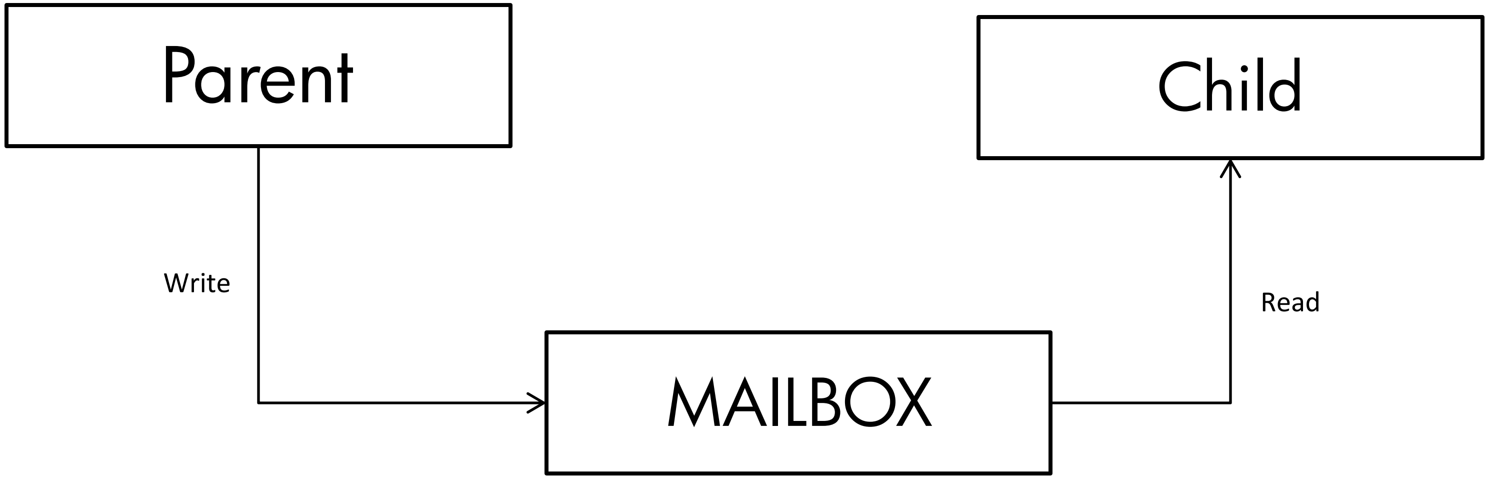
SSIO PROMOTES UNIX PORTABILITY

PIPE

- Unidirectional interprocess communication
- Has a *read* end and a *write end*
- Data written to the write end can be read from the read end
- No message boundaries

PIPE – current implementation

- pipe() is Implemented in CRTL using MAILBOX

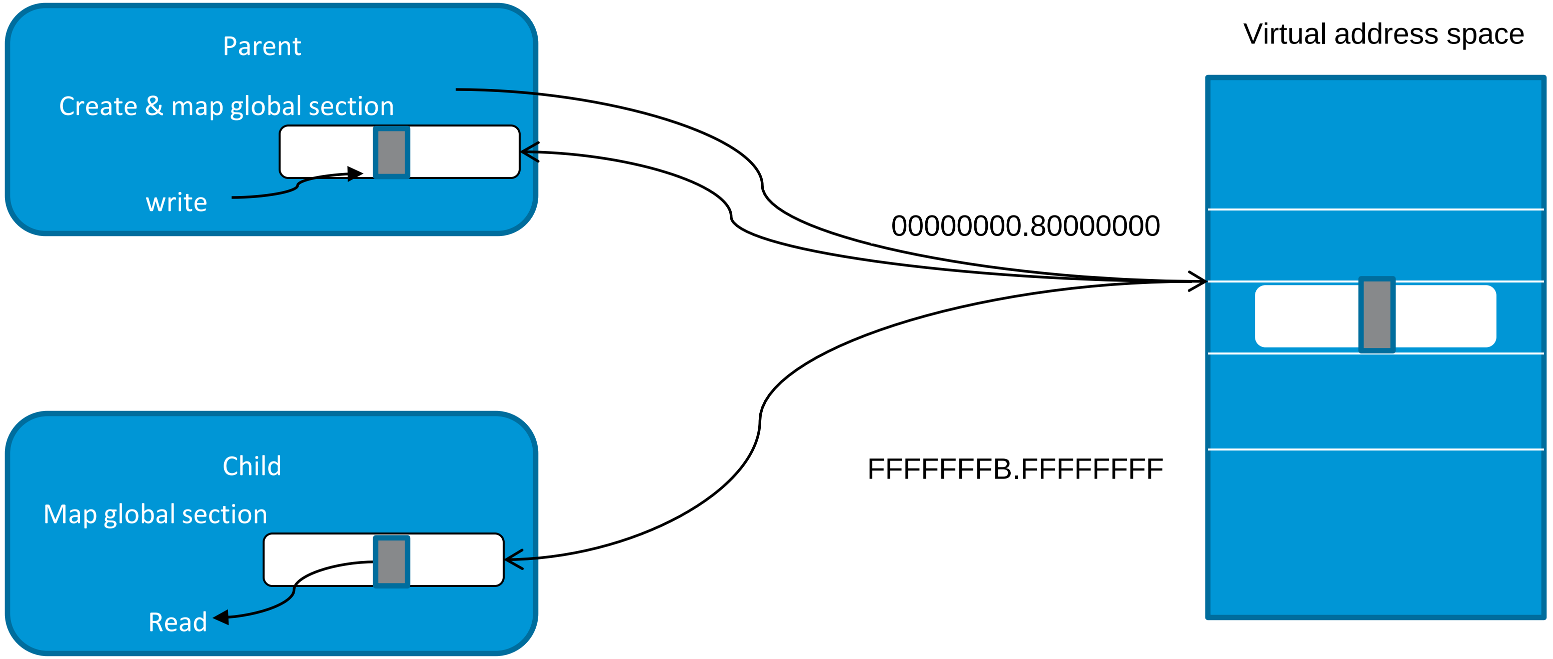


- Maximum mailbox size = 64 KB
- Consumes S0 (32 bit) limited address space

PIPE – Planned new implementations

- **Use global section backed by page file**
  - Mapped to P2 space (64 bit address space)
- **Use UNIX Domain Sockets**
- **Use 2 separate mailboxes**
  - One for data, other to notify
  - Store data in P2 space when mailbox is full
  - Reader will notify for more data using the 2nd mailbox
- **Advantages**
  - Larger size, more than 64K
  - Improved performance
  - Compatible with UNIX/Linux
  - Backward compatibility
  - Doesn't consume S0 space

PIPE – Using Global Section



GNV BASH 4.2

- Based on GNU BASH 4.2
- Contribution from opensource community
- Available at http://h71000.www7.hp.com/opensource/opensource.html

GNV BASH 4.2 – New features (1/2)

Upgrade from GNV BASH 1.6

>100 new features and bug fixes

- External commands
  - 2 ways to run external commands with $ or single quote

```
badresult=$(./ex17.sh)
goodresult=`./ex17.sh something`
echo "\"./ex17.sh\" gave: $badresult"
echo "\"./ex17.sh something\" gave: $goodresult“
```

- supports \u and \U Unicode escape
- can dynamically load built-ins at run time
  - Loaded using command “enable -f filename builtin-name”
  - Will speed up execution

GNV BASH 4.2 – New features (2/2)

- Negative array indices
- Negative parameter in string-extraction construct
- new `-g' option with declare/typeset to creates variables in the global scope in a shell
- `exec -a foo' now sets $0 to `foo‘
- Corrected permission problem with history file (.bash_history)

GNV BASH 4.2 – Restrictions

- Does not support for the 'fg' 'bg', and '&'
- DCL fallback is not implemented
- Bash currently uses the same control characters as OpenVMS, Control-Z is EOF
- ulimit builtin command is only partially implemented
- "test -x" does not append ".EXE"
  - Supposed to retry by appending .EXE with filename
  - Common practice to use filename without extension as hardlinks
  - Compatibility issues with other test options

To become a GNV developer

Subscribe to mailing list: https://lists.sourceforge.net/lists/listinfo/gnv-develop

Send a mail to : hp-gnv-devlp@users.sourceforge.net

Miscellaneous

MAKE utility

PostgreSql

Q&A

Thank you