Emerging Real-Time Communication Standards

Arkady Kanevsky

8 September 1997

MITRE

Overview

- Introduction
- Highlights of MPI/RT and POSIX.21
- Real-Time Philosophies
- Communication Paradigms
- Real-Time Paradigms - Outline
- Break
- Real-Time Paradigms - Details
- Buffer Management
- Collective Operations and Group Services
- Clock Synchronization
- Break
- Protocols, Addressing Issues
- Fault Handling, Instrumentation and Other Issues
- Conclusion

MITRE

Introduction

- Real-Time Standardization
  - Operating Systems: C real-time extensions: POSIX.1b, POSIX.1d, POSIX.1j, Ada real-time extensions: POSIX.5b; POSIX.1c - threads; IEEE 2003.1 - test methods for POSIX.1b; POSIX.13 - real-time profiles.
  - Communication
    - Network Layers
      ATM, FDDI-II, Token Ring, Token Bus.
    - Middleware Layers
      POSIX.21 - Real-time Distributed Systems Communication
      MPI/RT - Real-Time Message Passing Interface
    - Application/Object Layers
      Real-Time CORBA
      Real-Time JAVA

MITRE

POSIX.21 Highlights

- Draft IEEE standard intended for the real-time distributed system communication
- Key features:
  - Multiple Communication Models (unicast, multicast, broadcast, labeled messages)
  - Message Integer and/or Deadline priorities supported
  - Asynchronous and Synchronous interaction with implementation
  - Zero copy buffers possible
  - Lightweight directory services; protocol mapping
  - Event management
  - Connection management
  - Dynamic group management

MITRE

POSIX.21 Status

- POSIX - an IEEE standards group for developing standards for a *Portable Operating System Interface (PASC).* Portability, interoperability, and existing practice are important goals.
  - (ISO/IEC JTC1/SC22/WG 15 US TAG SD11)
  - http://www.pasc.org/
  - http://standards.ieee.org/
- Fast-tracked to become ANSI and ISO standards.
- History:
  - January 1993: Officially approved as *WG*
  - September 1996: IEEE ballot of *Language Independent Specification (LIS)*: over 1500 comments received, more than 60 balloters
  - Fall 1997: IEEE ballot LIS Recirculation
- Language bindings for *C* and *ADA* in progress

MITRE

POSIX.21 Participation

- WG members come from many places:
  - Industry (Lockheed-Martin, Boeing, Raytheon, Northrop-Grumman, Hughes, CDI, TI)
  - DoD Labs (UK MoD, NRaD, NSWC, NUWC)
  - Universities and FFRDCs (MITRE, CMU SEI, John Hopkins APL)
- Email Exploder:
  - *posix-rtds@sei.cmu.edu*
- Anonymous FTP site *ftp.sei.cmu.edu* contains an archive of material including meeting minutes, drafts of LIS, requirements, models and other documents, and prototype implementations
- More than 200 people on exploder

MITRE

MPI History

- MPI is a de facto Message Passing Standard
- MPI-1 completed 5/94, revised 6/95 & 7/97
- MPI-2 involves extensions to MPI-1
- MPI/RT is a part of MPI-2 process but not part of standard released 7/97 - real-time specification and features
- MPI/RT is not specifically a subset or superset of MPI-1 or MPI-2.
- De facto standards body (MPI-F, 1992-date)

MITRE

MPI/RT History

- Constituted informally in March/April, 1995
- Government, University, Commercial participants
- Has had “special status” in MPI process to allow time for concepts to mature properly
- Has broad scope in terms of augmenting and modifying MPI
- Has focused goal of providing standardized programming environment for RT
- Met at all MPI Forum meetings, and in between
- Like rest of MPI, an open forum, now stand-alone

MITRE

Existing MPI-1 Implementations

- General:
  - IBM, SGI and CRAY, HP and Convex, Fujitsu, NEC, Intel, SUN, MPI Software Technologies, Parsytec
- Embedded:
  - Mercury, Alacron, Hughes, Avalon, CSPI, MPI Software Technology, Lincoln-Labs, Lockheed-Martin

MITRE

MPI-1 Highlights

- Reliable, point-to-point and collective message passing operations
- Group-oriented communication with safety
- 2-sided communication scoped to ranks within groups
- Non-blocking and blocking, synchronized and buffered point-to-point models
- Blocking collective operations to groups
- Supports SPMD/MIMD parallel models with static processes
- LIS with C and Fortran 77 bindings
- Thread-safe API
- 129 functions with few key concepts to master

MITRE

MPI-2 Highlights

- Superset of MPI-1.2
- Dynamic process management
- 1-sided communication (put/get model, windows)
- I/O for flat file structures (including collective)
- External interfaces (for tools)
- Extended collective operations (intra group communications)
- C++ and Fortran 90 bindings for MPI-1 and MPI-2

MITRE

MPI/RT Highlights

- GOAL: *To develop application programming interface standard for message passing communication domain for real-time parallel and cluster systems*
- Embraces MPI-1 as base API and programming strategy, but modifies it for real-time
- Higher optimizability than MPI-1 and MPI-2
- Designed to support several RT paradigms
- Uses channels (point-to-point and collective) with quality of service
- Provides timeouts where appropriate
- Has profile stages to support cost-effective vendor development

MITRE

MPI/RT Participants

- Government Labs:
  - MITRE, NRaD, NUWC, MIT Lincoln Labs, Rome Labs, Sandia Labs
- Universities:
  - MSU, Caltech, Syracuse, Denver, Michigan, Maryland
- Industry:
  - Intel, Hughes, Mercury, SKY, CSPI, Lockheed-Martin, Avalon, Alacron, Raytheon, Khoral Research, SCA
- More than 200 people on exploder

MITRE

Being Plugged-in to MPI/RT

- Reflector
  - subscribe to *mpi-realtime-request@mcs.anl.gov* ("subscribe" in the first line)
  - send comments to *mpi-realtime@mcs.anl.gov*
- Web page
  - http://www.cs.msstate.edu/mpirt
  - latest and earlier versions of the standard document
  - schedule of meetings
  - related papers and pointers

MITRE

MPI/RT Technical Highlights

- Object-oriented design
  - Cloning and composition
  - Templates, objects, requests
  - Persistent generalized requests
- Deferred early binding philosophy
  - Maximum flexibility to optimize resources
- Attention to multithreaded applications
- Guaranteed QoS for data transfer operations

MITRE

Scope of MPI/RT Specification

- A full *draft* specification exists, under revision
- Timing requirements of implementation
- API for functions added or modified (to MPI)
- Channel, buffering, and admission test syntax/semantics
- Discussion of three real-time programming models
- Discussion of QoS specifications
- Fault tolerance / fault detection [early stages]
- Instrumentation[early stages]
- Profiles
- LIS specification, language bindings for C, C++, F77/F90 - maybe

MITRE

Scope of MPI/RT Functionality

- Channels
- Admission Tests
- Reliable, QoS-oriented data transfer and operations.
- Point-to-point operations on channels
- Collective operations on collective channels
- Admissions tests specified in a novel way to the system
- Both on-line and off-line mechanisms for establishing channels+QoS are to be entertained.
- Support for use of base MPI for non-real-time programming within the RT world.

MITRE

Current Real-Time Development Philosophy

- Current application development process
  - Step 1 - Application developers become platform experts
    - Platform provides message passing API
  - Step 2 - Application developers are responsible for achieving desired application performance and real-time guarantees
    - It is up to an application developer to get desired QoS and performance guarantees, by experimentation using platform-dependent API

MITRE

POSIX.21 Philosophy

- POSIX. 21 standardize existing practice.
- Adopt existing distributed system practice.
  - Separate system design and controls from application design and control
  - Add appropriate application controls without guarantees (Priorities)
  - Add existing features that improve performance
  - Provide mapping to the most widely used underlying transport layer protocols

MITRE

MPI/RT Philosophy - Rationale

- Current Philosophy is inherently platform-dependent, non-portable, and trial-and-error oriented
- Designers and implementers of a platform are better platform experts than users
  - Operating systems
  - Communication layers
  - Middleware with MPI/RT
- Platform and middleware designers know better what it takes to provide QoS
  - Providing QoS guarantee is a difficult schedulability problem:
    - Involves scheduling different resources
    - Different resources are scheduled using real-time paradigms and different scheduling algorithms

MITRE

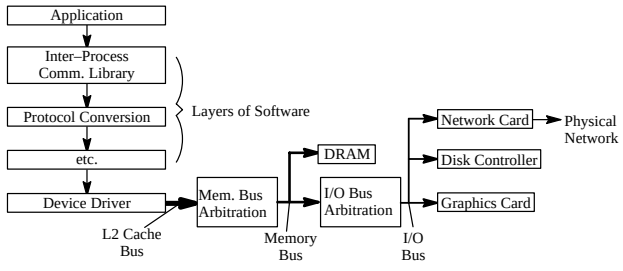
MPI/RT Philosophy Rationale - QoS Guarantees

- Providing QoS guarantee is a difficult schedulability problem, as it involves scheduling different resources:
  - CPUs where send/receiver applications are running
  - Data busses (e.g., PCI setup)
  - DMA engines
  - Memory (SRAM, DRAM)
  - Network interface chips (NICs)
  - Physical network (topology dependencies)

MITRE

Typical Node Architecture

- Most network co-processors reside on the peripheral bus!



1

MITRE

Peripheral Bus Penalties

- Cache bandwidth typically >> than memory bus bandwidth
- Memory bus bandwidth typically order of magnitude >> than peripheral bus
- Cache/Memory Bus/Peripheral Bus latencies follow similar relationship
- Example: Alpha 21164
- Bandwidth:
  - Cache bandwidth: 2.5 Gbyte/sec
  - Memory bus bandwidth: 1.2 Gbyte/sec
  - PCI bus bandwidth: 132 Mbyte/sec
- Latency:
  - B-cache latency: 26 nsec
  - Main memory latency: 253 nsec
  - Peripheral bus latency: 5 µsec

MITRE

MPI/RT Philosophy

- Difficult for application developers to coordinate resource usage to guarantee application QoS
  - Difficult schedulability problem, user solutions not portable
- Users provide detailed application info. - platform either satisfies user requests (dedicates resources) or states that it cannot support requested quality of service
- Allows MPI/RT implementers best to utilize platform data transfer primitives: two-sided, one-sided, and zero-sided communications
  - MPI/RT encourages this viewpoint but does not require it (that is, one can use priority model, or use “low QoS” specifications)
- Platform and Middleware Designers Exploit this information plus their inherently superior platform knowledge to achieve best performance portability

MITRE

MPI/RT Philosophy - Semantics

- MPI/RT provides both
  - Early binding
    - Supports implementation guaranteed user QoS - Channels
  - Late binding
    - Provides API to help application developers to satisfy user QoS requirements
  - MPI/RT provides an API for time-outs, handlers, instrumentation

MITRE

MPI Communication "Sidedness"

- MPI-1 supports 2-sided point-to-point.
- MPI/RT supports 2-sided, non-real-time modes of MPI-1.
- MPI/RT supports 1-sided point-to-point puts (and gets) on channels.
- MPI/RT supports 0-sided point-to-point transfers on time-based channels with periodic transfer.
- MPI-2 supports 1-sided window-oriented DSM. MPI/RT has not embraced this.

MITRE

MPI/RT Communication "Sidedness"

- 2-sided communication
  - point-to-point send/receive operations
  - collective operations
    - inter and intra communication in groups
      scatter, gather, reduce, allreduce, alltoall, scan
      MPI/RT proposes new: transpose-and-reshape
- 1-sided communication
  - put or get
- 0-sided communication
  - no application calls
  - on channels with time-based QoS with periodic transfers

MITRE

POSIX.21 Communication Paradigms

- 2-sided communication: one side "sends" another/other side(s) receive(s)
  - Unicast (point-to-point) specify "address"
  - Multicast (group) specify "address"
  - Broadcast (to every endpoint that registered for broadcast messages)
  - Labeled Messages (to every endpoint that registered for messages with this label)
- Connection (persistent communication)
  - Establish connection
  - Still 2-sided communication (still uses "address")
  - One connection per endpoint. The same endpoint can not be used for other data transfers while the connection is on.

MITRE

POSIX.21 Unicast Example

- **Application Sending:**
  - **Initialize communication specifying buffer pool sizes.**
  - **Create endpoint.**
  - **Use directory services to look up the name *fred.* The application is returned a local identifier for the name *fred.***
  - **Request a buffer for a message specifying message *send_control_block (destination_id* (represents *fred*), *message_length, priority, reliability).***
  - **Construct a message content in the buffer.**
  - **Send the buffer using endpoint resources.**

MITRE

POSIX.21 Unicast Example (continued)

- **Application Receiving:**
  - **Initialize communication specifying buffer pool sizes.**
  - **Create endpoint.**
  - **Associate the logical name *fred* with the endpoint. This makes the logical name available to the sending application (through the use of directory services).**
  - **Initiate reception of a message using buffered (address is returned to the application) or unbuffered semantics (application specifies an address where the message should be copied).**

MITRE

POSIX.21 Broadcast Example

- Application Sending:
  - Initialize communication specifying buffer pool sizes.
  - Create endpoint.
  - Request a buffer for a message without message *send_control_block.*
  - Construct a message content in the buffer.
  - Send the buffer using endpoint resources to the destination defined by the constant *BROADCAST_DESTINATION*.
- Application Receiving:
  - Initialize communication specifying buffer pool sizes.
  - Create endpoint.
  - Register endpoint to receive broadcast messages.
  - Initiate reception of a message.

MITRE

POSIX.21 Multicast Example

- Application Sending:
  - Initialize communication specifying buffer pool sizes.
  - Create endpoint.
  - Use directory services to look up the multicast group name *jim.* The application is returned a local identifier for the multicast group name *jim.*
  - Request a buffer for a message specifying message *send_control_block (destination_id* (represents *jim*), *message_length, priority, reliability).*
  - Construct a message content in the buffer.
  - Send the buffer using endpoint resources.

MITRE

POSIX.21 Multicast Example (continued)

- Application Receiving:
  - Initialize communication specifying buffer pool sizes.
  - Create endpoint.
  - Look up the multicast group name *jim* using directory services the multicast group lookup operation. The application will be returned a local identifier that could be used for multicast group management operations.
  - Join the endpoint to the multicast group *jim* using the identifier obtained from the lookup operation.
  - Initiate reception of a message using buffered (address is returned to the application) or unbuffered semantics (application specifies an address where the message should be copied).

MITRE

POSIX.21 Use of Message Labels Example

- Application Sending:
  - Initialize communication specifying buffer pool sizes.
  - Create endpoint.
  - Request a buffer for a message without message *send_control_block.*
  - Construct a message content in the buffer.
  - Request that the buffer be sent with the value of the message label to the implementation using endpoint resources.

MITRE

POSIX.21 Use of Message Labels Example

- Application Receiving:
  - Initialize communication specifying buffer pool sizes.
  - Create endpoint.
  - Register to receive messages of the specified message label.
  - Initiate reception of a message using buffered (address is returned to the application) or unbuffered semantics (application specifies an address where the message should be copied) possibly specifying the value of the message label that it will receive.

MITRE

Real-Time Paradigms

- Time-Driven
- Event-Driven
- Priority-Driven
- Best-Effort (Soft real-time)

MITRE

Triggering Entities

- Event-triggered system:
  - telephone systems
  - “drive/fly-by-wire” systems
  - plant automation
  - command-and-control
- Time-triggered system:
  - radar systems
  - plant automation
  - signal/sensor processing

MITRE

Real-Time Quality of Service

- Deadline
- Priority
- Starting Time
- Confusion between
  - real-time paradigms
  - real-time triggering mechanisms
  - real-time quality of service

MITRE

POSIX.21 Real-Time Paradigms (Priorities)

- Message assigned priorities
  - Integer priority
    - Implementation specifies the range
    - Minimal required range [1-32]
    - Higher number represents higher priority
  - Deadline priority
    - Semantics defined by application
    - Representation 64-bit integer
      High 32-bits - seconds since Epoch
      Low 32-bits - microseconds since Epoch
    - Message dropped if deadline expires before the message is queued.

MITRE

POSIX.21 Real-Time Paradigms (queues)

- Endpoints define queuing semantics
  - FIFO
  - Prioritized
  - Extra flag for receiving messages by label
    - for optimization

MITRE

MPI/RT Real-Time Paradigms

- Real-time paradigms - user perspective:
  - Time-Driven (purely calendar / periodic)
  - Mixed Time-Driven / Event-Driven (Time as special event)
  - Priority Channels with Event-Driven Delivery
  - Priority-Driven with polled delivery
  - High-Level Event-driven Model (Multicast events)
  - Other/Mixed Paradigms (Still being discussed/debated/designed)
  - Soft QoS (best effort) (still undefined)

MITRE

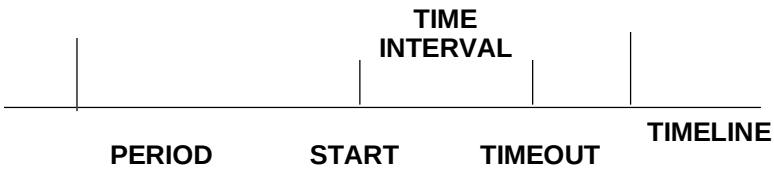
MPI/RT Real-Time Paradigms (continued)

- Each paradigm has specific QoS characteristics
- Each paradigm has unique additional functionality
- Lower level QoS parameters that are resource specific are not part of the API:
  - Examples: bandwidth, jitter, latency
  - Presented as advice to implementers

MITRE

Time-Driven MPI/RT Real-Time Paradigm

- Application activities are satisfied within the requested time interval
- Quality of Service parameters:
  - Time interval for data transfer
  - Period of data transfer for periodic computations
  - Relationship between the time interval and the period



MITRE

Time-Driven MPI/RT Paradigm (continued)

- User activities are done within the requested time interval.
- MPI/RT introduces two new parameters for data transfer operations:
  - Starting time of the data transfer (trigger)
  - Timeout of the data transfer (deadline - QoS and trigger)
- By specifying time interval an application guarantees that data is ready in the "send message buffer" to be sent and "receive message buffer" is ready to receive.
- Application guarantees that no system/implementation resources should be used on behalf of the time-driven message transfer outside the specified time interval.

MITRE

Time-Driven MPI/RT Paradigm (continued)

- 1-sided and 0-sided data transfer modes.
- Implicit MPI "ready" mode data transfer.
- Eliminates the need for queues and system buffers.
- No handshaking is required on behalf of the application.
- Each end of the persistent channel specifies QoS error handler that is automatically invoked by the MPI/RT implementation if QoS of data transfer is not met.
- Application does not need to know that message is transferred successfully.
  - By the end of the data transfer time interval either the message was transferred successfully or the error handler was invoked.

MITRE

MPI/RT Time Specification

- MPIRT_TIME_OBJECT
  - MPIRT_TIME_OBJECT_TYPE
    - Absolute
    - Relative
  - MPIRT_TIME_OBJECT_TIME
    - double precision floating point number

MITRE

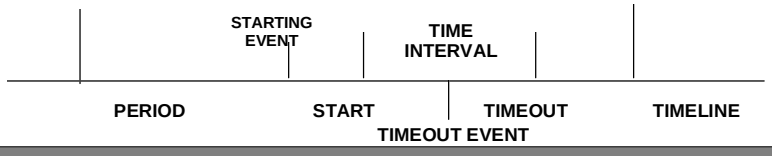
Example of the Use of Time-Driven Paradigm

- Establish point-to-point channel with single buffer in the buffer pool, no queuing strategy and QoS with period and time interval within it for data transfer
  - This defines a single step of the data flow pipeline
  - Establish each stage of the pipeline using a collection of point-to-point channels over the same communicator
    - Buffers are not shared between channels
- Start the pipeline
  - Create data in the "send message buffer" of a channel prior to the start time of the channel data transfer within the channel's period
  - Use the data in the "receive message buffer" of the channel after the channel's timeout within the period

MITRE

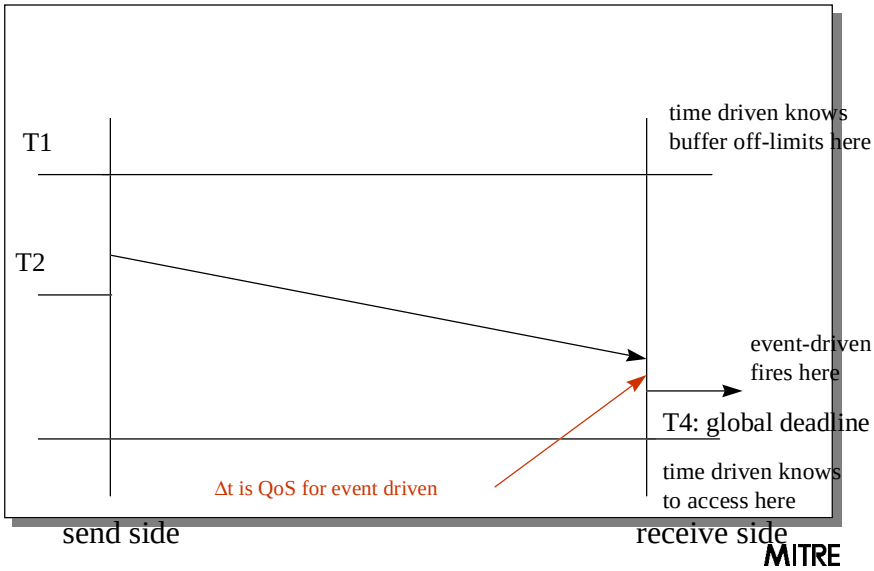
Event-Driven MPI/RT Real-Time Paradigms

- Application activities are bounded by time intervals, guarded by user specified events
- Quality of Service parameters:
  - Upper bound for an activation/deactivation time of an activity in response to an event:
- Events can be local (for low-level and high-level event driven paradigms) or group scoped (for high-level event-driven paradigm)



MITRE

Comparing Time- and Event-Driven Data Transfers



MITRE

MPI/RT Event-Driven Paradigms (continued)

- **Low Level - a mechanism for scheduling an application handler with QoS upon completion of data transfer operations**
  - **similar to *hrecv, hsend* operations**
- **High Level - a mechanism for scheduling any application activity with QoS:**
  - **Application activity:**
    - **MPI/RT data transfer (channel use)**
    - **Application thread, process, or function**
  - **Triggering Events:**
    - **System Event (OS captures)**
    - **Communication Event (MPI/RT captures)**
    - **Application Event(Application generates)**

MITRE

Low Level MPI/RT Events

- MPI/RT defines two events for completion of data transfer
  - Local completion of data transfer (message buffer can be reused)
  - Global completion of data transfer (channel resources can be reused)
  - Start of data transfer (future consideration)

MITRE

Low Level MPI/RT Event-Driven Paradigm Request Parameters

Request	MPI (MPI/RT) handle for the channel
Request_Cond	Requested channel condition
Cond_Handler_Fn	Requested condition handler function
Failure_Fn	QoS failure function
Extra_State	User supplied function state
Handler_QoS	Quality of service of the handler. Either: Priority or Deadline

MITRE

High Level MPI/RT Event-Driven Paradigm

- An application specifies *interval* of time *guarded* by the specified events in order to bound the resource usage of communication and computation application activities.
- Rationale:
  - Current Application Control
    - Application waits on system events or user control message
    - Application schedules a handler that in turn schedules application activities (functions, processes, thread/tasks, data transfers)
    - MPI provides an interface for data flow control
  - High level MPI/RT event-driven paradigm provides an interface for control flow.

MITRE

MPI/RT High Level Events

- Events can be persistent or one time only
- Events are not necessarily local to a process or a node
- Event Types:
  - System Events - MPI/RT implementation is aware of them.
  - Communication Events - MPI/RT generates and captures them.
  - Application Events - Application generates them and has to pass this information to MPI/RT.
- MPI/RT associates a name with each event to handle persistent/periodic events.
- MPI/RT associate a name with a channel to handle persistent channel data transfer events.

MITRE

MPI/RT High Level Event-Driven Paradigm Intervals

- Application events have meaning only to the application.
- MPI/RT is just a mechanism to match user activities with user requests to "monitor" events; a mechanism for event delivery; and a mechanism for triggering/timing out application activities.
- Application specifies events that guard start and timeout of an activity.
- Interval of time for time-driven paradigm is a subset of the high-level event-driven real-time paradigm interval of time guarded by the events.
- Critical Difference between time and event paradigms:
  - application's inability to schedule communication and computation activities,
  - Synchronization
  - Time interval is both triggers and QoS

MITRE

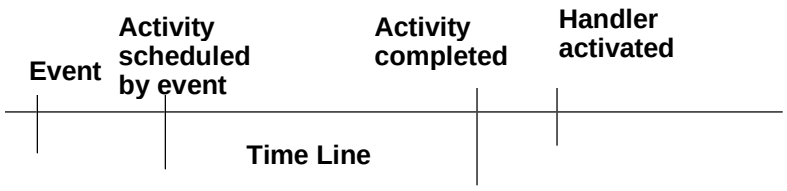
MPI/RT High Level Event-Driven Operations

- MPIRT_REQUEST_GUARDED - allows an application to bind a guarded activity associated with the request with the interval of time guarded by the events.
- MPIRT_EVENTNAMES_REGISTER and MPIRT_EVENTNAMES_DEREGISTER - allows an application to specify events that it would like MPI/RT to monitor and that are used to guard some process activity, and events that this process generates.
- MPIRT_EVENT_GENERATED - allows an application to generate event.

MITRE

MPI/RT QoS for Handlers

- For channel data transfer:
  - integer priority
  - deadline
- For Handler and Event Deliver analogous QoS.



MITRE

MPI/RT Priority Real-Time Paradigm

- Priority is defined for a channel for all message data transfer of the channel not individual messages.
- Priority just specifies QoS.
- Combining with Event-Driven Paradigm that specifies Triggers defines full channel real-time specification.
- No relationship to the process priorities.
  - Channel priorities defines order for multiple resources, and not only OS as process priorities do.
- Channels with priority mean using multiple channels for multiple priority levels
- MPI/RT doesn't give specific meaning to specific priority values
- Emphasizes static priority channels

MITRE

MPI/RT Point-To-Point Channels

- Communicators are used to describe groups and set up real-time communication
- Sets of channels, each with QoS are requested, and *collective admission tests* are performed by the implementation to define channels
- Point-to-point channels are uni-directional, reliable, packet passing mechanisms with QoS, buffering schemes, and fixed end-points

MITRE

MPI/RT Deferred Early Binding

- Collect all information for all the channels over a communicator and then create them all at once - to optimize resource usage and to provide timing and QoS guarantees.
- Templates collect information:
  - Operations: create, set/get parameters, duplicate, free
  - *Bufpool*
    - *count, datatype, bufcount*
    - *bases* - application provides or system creates when the channels are established
  - *Bufqueue*
    - *bufpool, length, strategy* (buffer iterators)

MITRE

MPI/RT Deferred Early Binding (continued)

- *Channel*
  - *bufqueue, rank, direction, channel_qos, error_fn, handler_templates, channel_name*
- *Channel_set*

- *Channels_create: comm, channel_set* template - creates channels, channelset, bufqueue, bufpool, buffers, handlers

MITRE

POSIX.21 Unicast Model (Sending)

- Data transfer operations have the same format for unicast, multicast, broadcast and labeled messages.
  - Asynchronously or synchronously data transfers, with timeout
  - *Send_control_block (create/delete)*
    - endpoint - through which a message will be sent
    - destination_id - for the message
    - message_length
    - message_priority
    - reliability
    - [label] - for labeled messages
  - *Send_message_with_send_control_block*
  - *Send_message_with_send_control_block _and_buffer*
  - *Send_destination_message*

MITRE

POSIX.21 Unicast Model (Receiving)

- Asynchronously or synchronously data transfers, with timeout
- With or without buffer
  - *Receive_message* specifies length to be copied
  - *Receive_message_with_buffer* returns message length
  - *Receive_message_from_set* of endpoints
- Events
  - *message_lost*
  - *message_discarded*

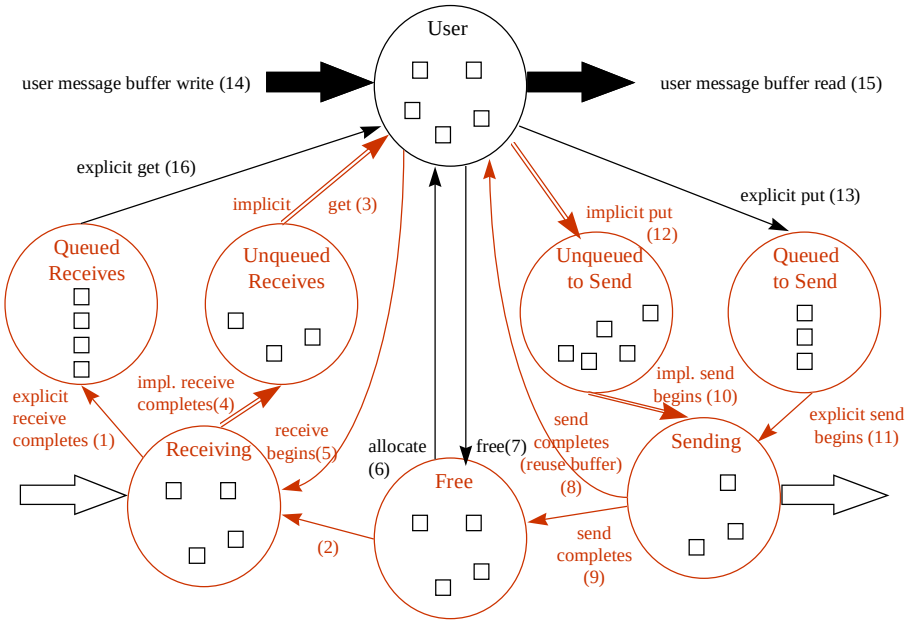
MITRE

MPI/RT Buffers, Buffer Pools, and Queues

- Buffer pools support abstraction that manages multiple buffers for an endpoint of a channel
- API specifies implementation buffer management strategy and application buffer management strategy
  - implementation: e.g., overwrite/non-overwrite of buffers when out of space
  - application: e.g., random access to buffers (not necessarily FIFO or LIFO)
- Concepts such as “double buffering” and “data fusion” are supported
- Buffer state transitions carefully specified to provide clear behavior
- Work on “cut-through”-like properties still under discussion

MITRE

The Bufferpool/Buffer State Transition Diagram for MPI/RT



10/21/97 17:11 66

State Transition Legend

(1)	Explicit receive completes
(2)	Allocate Buffer from Buffer Pool Free List for Receive operation
(3)	Implicit get of buffer asynchronously from user activity
(4)	Implicit receive completes
(5)	Receive begins (buffered receive)
(6)	Allocate Buffer from Buffer Pool Free List
(7)	Free Buffer to in Buffer Pool
(8)	Send completes (with buffer returned to user intact for reuse)
(9)	Send completes (buffer returned to Buffer Pool Free List)
(10)	Implicit send begins
(11)	Explicit send begins
(12)	Implicit put of Buffer
(13)	Explicit put
(14)	User message buffer write (user gathers in a buffer to program space)
(15)	User message buffer read (user scatters out a buffer to program space)
(16)	Explicit get of buffer by user

MITRE

POSIX.21 Buffers, Queues and Buffer Pools

- Buffer can be shared between all the endpoint of a process.
- Application requests sizes for buffer pools sizes for sending buffers and receiving buffers. Implementation is allowed to add extra for message headers and other info.
- Application can receive a message in a buffer through an endpoint, modify it, and send the same buffer through any endpoint of the process.
- Application requests a buffer for a message sending, or a receive message buffer and gets back a handle of the buffer.
- Message buffer is comprised of
  - message length
  - number of buffers
  - message buffer array
    - address and length

MITRE

POSIX.21 Buffers, Queues and Buffer Pools (Continued)

- Application can reuse the same buffer many times (send)
- Buffers are managed by Implementation on behalf of the application.
- Buffers are process wide (exception Fork)
- Buffer Management Operations:
  - *GET_MESSAGE_BUFFER*
  - *GET_MESSAGE_BUFFER_WITH_SEND_CONTROL_BLOCK*
  - *RELEASE_MESSAGE_BUFFER*
  - Inquiry:
    - get send/receive buffer space total/available

MITRE

POSIX.21 Endpoint Parameters

- Queue Lengths: Event, Send, Receive
- Endpoint Reliability
- Maximum Receive Message Length
- Receive Message Queue Policy: FIFO or Priority
- Maximum Send/Receive Message Length: Maximum number of pending transmissions
- Flags: Read Only Receives,
  Receive by Label
- Endpoint can be closed gracefully or abruptly, synchronously or asynchronously
- Purge Endpoint - all queues
- Endpoint can enable or disable event queue
- Inquiry: number of pending sends/receives
- Endpoints, Buffer pools and other POSIX.21 objects are shared over *FORK*. The process that *EXEC* inherits open endpoints.

MITRE

POSIX.21 Events

- Three types of events:
  - Message transmission event
    - message lost, discarded
  - Multicast event
    - member joins, leaves; group deleted
  - Connection event
    - abortive close receive, graceful close request, connection lost
- Operations:
  - Mask/Unmask event types
  - Get number of pending events
  - get next event synchronously/asynchronously
- Events queued in FIFO order

MITRE

MPI/RT Admission Tests

- Admission tests over sets of channels is supported as a collective construction process. This is asserted to be a new way to present the real-time requirements to the message passing subsystem.
- A collective operation makes a set of point-to-point channels
- A collective operation defines a single collective channel, with a specific operation.
- Systems like ATM, Tenet and U. of Michigan group use admission tests on single channels, with hop-by-hop tests. MPI/RT specifies the admission test at a higher level.

MITRE

MPI/RT Collective Channels

- Collective channels use a fixed group of participants to define a collective operation as a single channel abstraction, offering the same type of quality of service as point-to-point channels
- For each collective operation, a collective channel with quality of service can be established. The MPI-type collective operation is executed on demand (or as scheduled) with the particular QoS.
- Collective channels are persistent and use “early binding” to negotiate once and use often a pattern of communication that is not binary, but rather involves the whole group of processes.
- MPI and MPI/RT collective operations support one-to-many, may-to-one, and many-to-many models of communication that are widely used for parallel computing

MITRE

MPI/RT Collective Operations

- Quality of service for clique and bi-partite collectives, in form of collective channels
- Early-binding semantics allows for use of poly-algorithms, and reduces per-use overheads
- Since existing MPI collectives (even MPI-2 extensions) have wrong abstraction level for transpose, we are developing specific transpose for RT, to be more easily used and optimized.

MITRE

POSIX.21 Collective Models (Multicast)

- One-to-many: a source entity interacts with some number of other entities. Traditional Multicast model of distributed computing.
- Multicast: based upon communication among a known set of members that comprise a multicast group. Collective Operations and Group Services
- Sender based model.
  - Create Multicast Group, Join, Leave, Delete,
  - Obtain multicast group identifier, Obtain destination identifier, Release multicast group identifier
  - Inquiry functions:
    - Get number of members,
    - Determine Membership in a multicast group
    - Get Members of a multicast group
  - Generate events for Join, Leave, Group Deleted.

MITRE

POSIX.21 Collective Models (Labeled Messages)

- Labeled messages: the case where a message is sent with an associated label. The message will be received by those destinations that have registered to receive a particular message label.
- Receiver based paradigm
  - No management information available
  - Receiver knows the identity of the sender but the sender does not know the identity of receivers.
  - Operations:
    - Register/Deregister label / range / set
    - Send labeled message / with destination
    - Receive labeled message: 1 endpoint, set of endpoints
    - Receive message by label: 1 or set of endpoints

MITRE

POSIX.21 Collective Models (Broadcast)

- One-to-all model
  - Destinations not known
  - Endpoints Register/Deregister to receive broadcast messages
  - Standard send/receive operation used
    - BROADCAST_DESTINATION used for send

MITRE

Clock Synchronization

- Provides precise timing correctness
  - Synchronized clocks to support quality of service (QoS) timing requirements
  - Platform portability achieved - performance tuning using QoS parameters.
- Support fine grained instrumentation
  - delicate tuning for optimal performance
- Predictable timing behavior

MITRE

POSIX Clocks

- POSIX.21 does not specify synchronized clocks
- POSIX.1d defines synchronized clocks and POSIX.21 is an extension/modification to POSIX.1 and POSIX.5.
- POSIX standards are not allowed to define performance parameters.
- Implementations of POSIX standards can define system constant that provide performance parameters and guarantees.

MITRE

MPI/RT Clocks

- The MPI/RT clock is a synchronized clock.
  - The design of the synchronized clock is left to the implementers. MPI/RT Clocks may be local or global.
- Require that clocks be monotonically non-decreasing.

MITRE

MPI/RT Clock Parameter Definitions

- *Drift* - clock rate error bound
- *Skew* - maximum bound on absolute value of difference between simultaneous values of synchronized clock in distinct nodes
- *Accuracy* - maximum bound on absolute value of the difference between simultaneous values of synchronized clock and ideal clock, started at the synchronized clock's hypothetical start time.
- *Resolution (Tick)* - time between two successive updates
- *Access Time* - maximum bound on time to execute a call of MPI_WTIME, the synchronized clock access function.
- Formats of clock attributes are POSIX 1.b compliant.

MITRE

MPI/RT Instrumentation

- Monitoring/Instrumentation (Medium term)
  - provide portable linkage to system behavior
  - still being specified
- Allows monitoring of user and MPI/RT -specified information.
  - Monitoring and instrumentation of Channels (point-to-point, collective) (data transfers)
  - Communicators
  - External Monitors for Application activities
- Application specified metrics
  - all complete information
  - single numbers (max, min, average, percentage QoS met)
  - distribution results

MITRE

MPI/RT Fault Tolerance

- Fault Handling (Long term)
  - early explorations into making an MPI specification with fault tolerance or just fault detection
  - timeouts are first step in this direction
  - handlers as mechanism to associate fault handling with MPI requests (channels, MPI data transfers, others)
  - Possible use of Instrumentation and Monitoring for application feedback and control mechanisms
  - will leave most work for future
  - different audience with different requirements
- Shadow Channels with different QoS

MITRE

Addressing issues

- MPI/RT addressing issues are outside the standard
  - application refers to ranks within the communicator
- POSIX.21 supports both location-independent view (logical names), and location-dependent view (Protocol Dependent Addresses (PDA)).
- *Logical_name*
  - *Destination_ID*
  - *Protocol_dependent_address*

MITRE

POSIX.21 Directory Services

- *Associate_logical_name* to endpoint
- *Lookup_logical_name* returns destination_id
- *Get_logical_name* takes destination_id and returns logical_name
- *Bind_endpoint* to a protocol dependent address
- *Get_identifier_for_protocol_dependent_address* returns destination_id
- *Get_protocol_dependent_address* from destination_id
- *Release_destination_id*
- *Convert_endpoint_to_destination* within the same process

MITRE

Protocol Mappings

- MPI/RT - no protocol mapping. Left to the implementation. No common high performance protocols. Current work on *Packetway* IETF standard that replaces to some degree IP. No TCP, or UDP equivalents. Current work on transport layers for Myrinet.
- POSIX.21 - defines mappings to TCP, UDP and IP.
  - Define protocol headers for interoperability
  - Operations:
    - *Get_available_protocols* for an application
    - *Get_protocols* bound to a specific endpoint
    - *Get_protocol_state*
    - *Get_protocol_parameters* for a specified protocol
    - *Set_protocol_parameters* for a specified protocol

MITRE

Implementation Activities

- POSIX.21 Prototypes (partial)
  - CMU SEI
  - Texas Instrument (Raytheon)
  - Lockheed-Martin (Syracuse)
- MPI/RT Prototype (starting activity)
  - MSU
    - IneRTia - Provides no QoS, but provides complete API (Fall1997)
    - Action - Provides simple QoS on specific systems with single channels. Linux/RT OS (Summer 1998)
    - Momentum - Robust prototype that includes significant admission test logic.

MITRE

Conclusion

- Standardization activity reaches real-time area.
- Helps real-time move to the main stream computing
- What standards should I consider for my application?
- For communication domain when should I use POSIX.21 and when should I use MPI/RT
  - Timing granularity?
  - Throughput requirements?
  - Static vs. dynamic data flows?
  - Distributed vs. parallel models of computing?
  - Sender based vs. Receiver based models?
    - High performance, static pipelines, high utilization, tighter timing constraints - MPI/RT
    - Lower performance, dynamic data flow, looser timing constraints - POSIX.21

MITRE