Message queue

From Wikipedia, the free encyclopedia

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. *(May 2009)*

In computer science, **message queues** and **mailboxes** aresoftware-engineering components used for inter-process communication (IPC), or for inter-thread communication within the same process. They use a queue for messaging – the passing of control or of content. Group communication systems provide similar kinds of functionality.

Contents [hide]

Overview [edit]

Message queues provide an asynchronous communications protocol, meaning that the sender and receiver of the message do not need to interact with the message queue at the same time. Messages placed onto the queue are stored until the recipient retrieves them. Message queues have implicit or explicit limits on the size of data that may be transmitted in a single message and the number of messages that may remain outstanding on the queue.

Many implementations of message queues function internally: within an operating system or within an application. Such queues exist for the purposes of that systemonly.[1][2][3]

Other implementations allow the passing of messages between different computer systems, potentially connecting multiple applications and multiple operating systems.[4]These message queueing systems typically provide enhanced resilience functionality to ensure that messages do not get "lost" in the event of a system failure. Examples of commercial implementations of this kind of message queueing software (also known as message-oriented middleware) include IBM WebSphere MQ (formerly MQ Series) and Oracle Advanced Queuing (AQ). There is a Javastandard called Java Message Service, which has severalproprietary and free software implementations.

Implementations exist as proprietary software, provided as aservice, open source software, or a hardware-based solution.

Proprietary options have the longest history, and include products from the inception of message queuing, such asIBM WebSphere MQ (formerly MQ Series), and those tied to specific operating systems, such as Microsoft Message Queuing. There are also cloud-based message queuing service options, such as Amazon Simple Queue Service(SQS), StormMQ and IronMQ.

There are a number of open source choices of messagingmiddleware systems, including Apache ActiveMQ, Apache Kafka, Apache Qpid[5], Beanstalkd, HTTPSQS[6]JBoss Messaging, JORAM, RabbitMQ, Sun Open Message Queue, and Tarantool.

In addition to open source systems, hardware-basedmessaging middleware exists with vendors like Solace Systems, Sonoa / Apigee and Tervela offering queuing through silicon or silicon/software datapaths.

Most real-time operating systems (RTOSes), such asVxWorks and QNX, encourage the use of message queueing as the primary inter-process or inter-thread communication mechanism. The resulting tight integration between message passing and CPU scheduling is attributed as a main reason for the usability of RTOSes for real time applications. Early examples of commercial RTOSes that encouraged a message-queue basis to inter-thread communication also include VRTX and pSOS+, both of which date to the early 1980s. The Erlang programming language uses *processes* to provide concurrency; these processes communicate asynchronously using message queuing.

Usage [edit]

In a typical message-queueing implementation, a system administrator installs and configures message-queueing software (a queue manager or broker), and defines a named message queue. Or they register with a message queuing service.

An application then registers a software routine that "listens" for messages placed onto the queue.

Second and subsequent applications may connect to the queue and transfer a message onto it.

The queue-manager software stores the messages until a receiving application connects and then calls the registered software routine. The receiving application then processes the message in an appropriate manner.

There are often numerous options as to the exact semantics of message passing, including:

- Durability - messages may be kept in memory, written to disk, or even committed to a DBMS if the need for reliability indicates a more resource-intensive solution.
- Security policies - which applications should have access to these messages?
- Message purging policies - queues or messages may have a "time to live"
- Message filtering - some systems support filtering data so that a subscriber may only see messages matching some pre-specified criteria of interest
- Delivery policies - do we need to guarantee that a message is delivered at least once, or no more than once?
- Routing policies - in a system with many queue servers, what servers should receive a message or a queue's messages?
- Batching policies - should messages be delivered immediately? Or should the system wait a bit and try to deliver many messages at once?
- Queuing criteria - when should a message be considered "enqueued"? When one queue has it? Or when it has been forwarded to at least one remote

queue? Or to all queues?

- Receipt notification - A publisher may need to know when some or all subscribers have received a message.

These are all considerations that can have substantial effects on transaction semantics, system reliability, and system efficiency.

Standards and protocols [edit]

Historically, message queuing has used proprietary, closed protocols, restricting the ability for different operating systems or programming languages to interact in a heterogeneous set of environments.

An early attempt to make message queuing more ubiquitous was Sun Microsystems' JMS specification, which provided aJava-only abstraction of a client API. This allowed Java developers to switch between providers of message queuing in a fashion similar to that of developers using SQLdatabases. In practice, given the diversity of message queuing techniques and scenarios, this wasn't always as practical as it could be.

Two standards have emerged which are used in open source message queue implementations:

1. Advanced Message Queuing Protocol (AMQP) – feature-rich message queue protocol
2. Streaming Text Oriented Messaging Protocol(STOMP) – simple, text-oriented message protocol

These protocols are at different stages of standardization and adoption. All of them operate at the same level as HTTP.

Some proprietary implementations also use HTTP to provide message queuing by some implementations, such asAmazon's SQS. This is because it is always possible to layer asynchronous behaviour (which is what is required for message queuing) over a synchronous protocol using request-response semantics. However, such implementations are constrained by the underlying protocol in this case and may not be able to offer the full fidelity or set of options required in message passing above.

Synchronous vs. asynchronous [edit]

Many of the more widely known communications protocols in use operate synchronously. The HTTP protocol – used in theWorld Wide Web and in web services – offers an obvious example where a user sends a request for a web page and then waits for a reply.

However, scenarios exist in which synchronous behaviour is not appropriate. For example, AJAX (AsynchronousJavaScript and XML) can be used to asynchronously send text or XML messages to update part of a web page with more relevant information. Google uses this approach for their Google Suggest , a search feature which sends the user's partially typed queries to Google's servers and returns a list of possible full queries the user might be in the process of typing. This list is asynchronously updated as the user types.

Other asynchronous examples exist in event notification systems and publish/subscribe systems.

- An application may need to notify another that an event has occurred, but does not need to wait for a response.
- In publish/subscribe systems, an application "publishes" information for any number of clients to read.

In both of the above examples it would not make sense for the sender of the information to have to wait if, for example, one of the recipients had crashed.

Applications need not be exclusively synchronous or asynchronous. An interactive application may need to respond to certain parts of a request immediately (such as telling a customer that a sales request has been accepted, and handling the promise to draw on inventory), but may queue other parts (such as completing calculation of billing, forwarding data to the central accounting system, and calling on all sorts of other services) to be done some time later.

In all these sorts of situations, having a subsystem which performs message-queuing (or alternatively, a broadcast messaging system) can help improve the behavior of the overall system.

Implementation in UNIX [edit]

UNIX implements message passing by keeping an array of linked lists as message queues. Each message queue is identified by its index in the array, and has a unique descriptor. A given index can have multiple possible descriptors. UNIX gives standard functions to access the message passing feature.[7]

- **msgget()**: This system call takes a key as an argument and returns a descriptor of the queue with the matching key if it exists. If it does not exist, and the `IPC_CREAT` flag is set, it makes a new message queue with the given key and returns its descriptor.
- **msgrcv()**: Used to receive a message from a given queue descriptor. The caller process must have read permissions for the queue. It is of two types.[8]
  - Blocking receive puts the child to sleep if it cannot find a requested message type. It sleeps till another message is posted in the queue, and then wakes up to check again.
  - Non-blocking receive returns immediately to the caller, mentioning that it failed.
- **msgctl()**: Used to change message queue parameters like the owner. Most importantly, it is used to delete the message queue by passing the `IPC_RMID` flag. A message queue can be deleted only by its creator, owner, or the superuser.

See also [edit]

- Advanced Message Queuing Protocol (AMQP)
- Amazon Simple Queue Service
- Apache ActiveMQ
- Celery Task Queue
- Gearman
- IBM WebSphere Message Broker
- IBM WebSphere MQ
- IronMQ
- Java Message Service
- Message-oriented middleware, (category)
- Microsoft Message Queuing (known colloquially as MSMQ)
- Microsoft Azure, particularly Azure storage queues andAppFabric Service Bus
- OpenEdge Sonic MQ

- QDB queues with message replay feature
- RabbitMQ
- StormMQ, an example of a message queuing service
- ØMQ
- SnakeMQ
- HornetQ
- TIBCO Enterprise Message Service

References [edit]

1. ^ Win32 system message queues. "About Messages and Message Queues" . *Windows User Interface*. Microsoft Developer Network. Retrieved April 21, 2010.
2. ^ Linux and POSIX message queues. Overview of POSIX message queues at linux.die.net
3. ^ Using Linux Message Queues. Linux message queue functions at www.civilized.com
4. ^ For example, the MSMQ product. "Message Queuing (MSMQ)" . *Network Communication*. Microsoft Developer Network. Retrieved May 9, 2009.
5. ^ Apache Qpid Project, an implementation of AMQP.
6. ^ HTTPSQS, a message queue based on HTTP GET/POST protocol.
7. ^ Bach, M.J. *The Design of the UNIX Operating System.*ISBN 9780132017992.
8. ^ Abraham Silberschatz, Peter B. Galvin. *Operating Systems Concepts*. ISBN 9780201504804.

VTE **Inter-process communication** [hide]

Data exchange among threads in computer programs

Methods	FileMemory-mapped fileMessage passing**Message queue and mailbox**Named pipeAnonymous pipePipeSemaphoreShared memorySignalSockets InternetUnix
Protocols and standards	Apple eventsCOM+CORBAD-BusDDSDCEICEOpenBinderONC RPCPOSIX (various methods)SOAPRESTThriftTIPCXML-RPC
Software libraries and frameworks	D-BuslibeventSIMPLLINX

Categories: Inter-process communication | Events (computing)