Message queues

7 minutes

A message queue is a technique used for inter-process communication (also known as IPC), or between various components of an application, or across applications. Message queues provide a protocol or interface to enable message passing.

Message queues for IPCs within a single machine have been made available through the UNIX `<sys.msg.h>` and POSIX `mqueue.h` system libraries, where they can be used in both synchronous and asynchronous modes. A synchronous message requires the sending or receiving processes to be blocked while the message is being sent, until it has been acknowledged by the receiver. Asynchronous messages do not block the sending or receiving processes, and can be buffered for delayed delivery.

For applications that span multiple machines, a number of proprietary and open-source message queue systems have been developed, such as IBM's WebSphere MQ, the Java Message Service (JMS), and RabbitMQ. All of these systems provide the ability to send, manage, and receive messages.

Employing message queues can add complexity to the application design and increase the number of services that need to be managed when compared to traditional request and response messaging. However, at a certain scale, and for applications such as stream processing, distributed message queue systems are needed to manage the added complexity and scale.

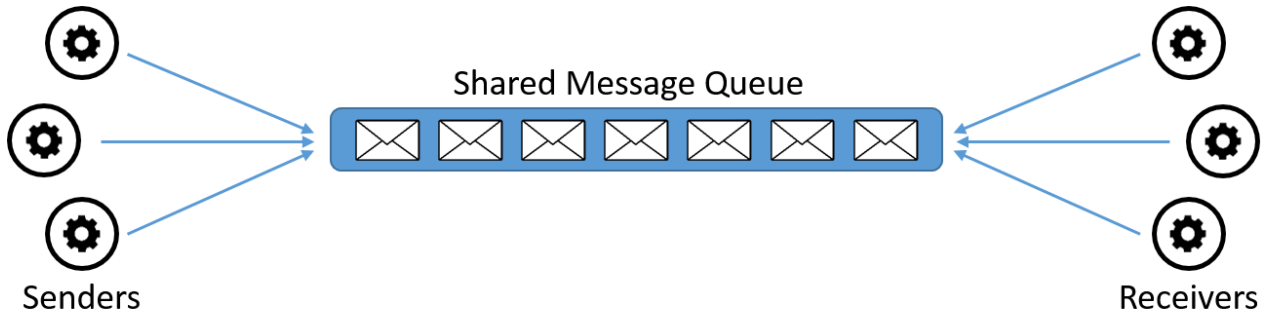

Figure 1: An interface connects to senders (or producers), which create messages, and to receivers (or consumers), which can receive the messages.

Characteristics of message queues

Message queues and their novelty can be understood and appreciated when contrasted with traditional RPC and request-response mechanisms. The primary features of a message queue are as follows:

Storage: Unlike traditional request and response systems that rely on basic TCP and UDP protocols over sockets, message queues typically store messages in some type of buffer until they have either been read by a destination process or been explicitly removed from the message queue.

Asynchronicity: In contrast to request and response systems, the buffering of messages allows message queues to expose a degree of asynchronicity in applications, allowing source processes to send messages and let them accumulate in a queue while destination processes pick them up to process. This allows for applications to function under certain failure scenarios, such as intermittent connectivity or failure of either the source process or the destination process.

Routing: Message queues may also provide routing functionality, where multiple processes can read or write messages in the same queue, allowing for broadcast or unicast communication paradigms.

Advantages of message queues

The primary advantage of a message queue is that it provides loose coupling between various entities in a distributed application. This allows for asynchronous non-blocking communication that provides a higher level of tolerance against failures of processes.

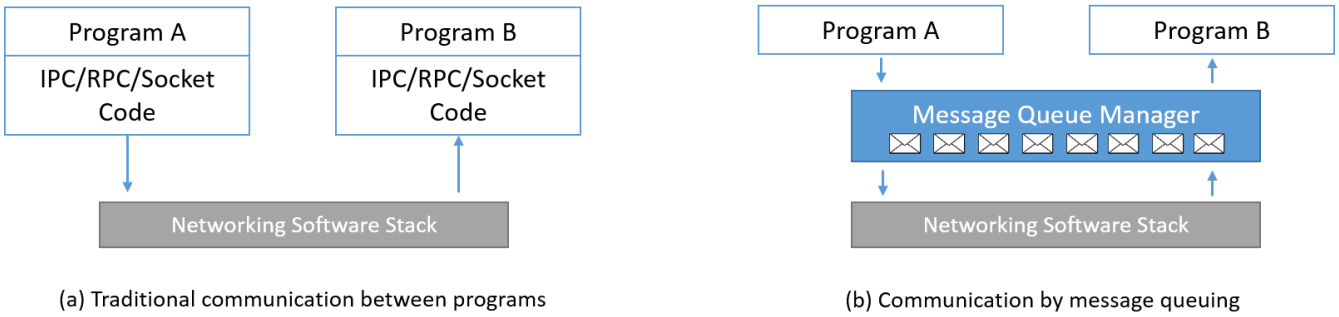

Figure 2: Message queues provide an interface that allows for programs to communicate with each other.

- **No direct connections between programs**: Message queuing allows for indirect program-to-program communication. Senders do not necessarily need to know the receivers of messages and vice versa. This allows for the logic of routing messages (and potentially work) to be decided by the message queueing system.
- **Communication between programs can be independent of time**: Message queues typically buffer messages between programs so they do not have to block or interrupt

their execution to send or receive messages. They can do other work, and process the reply either when a message arrives or at a later time. When writing a messaging application, you do not need to know (or be concerned) when a program sends a message, or when the target is able to receive the message. The message is not lost; it is retained by the queue manager until the target is ready to process it. The message stays on the queue until it is removed by a program. This means that the sending and receiving programs are decoupled; the sender can continue processing without waiting for the receiver to acknowledge receipt of the message. The target application does not even have to be running when the message is sent. It can retrieve the message after it has been started.

- **Work can be carried out by small, self-contained programs**: Message queuing allows you to use the advantages of using small, self-contained programs. Instead of a single, large program performing all the parts of a job sequentially, you can spread the job over several smaller, independent programs. The requesting program sends messages to each of the separate programs, asking them to perform their function. When each program is complete, the results are sent back as one or more messages.
- **Communication can be driven by events**: Programs can be controlled according to the state of queues. For example, you can arrange for a program to start as soon as a message arrives on a queue. Or you can specify that the program does not start until there are, for example, 10 messages above a certain priority on the queue, or 10 messages of any priority on the queue.
- **Applications can assign a priority to a message**: A program can assign a priority to a message when it puts the message on a queue. This determines the position in the queue at which the new message is added. Programs can get messages from a queue either in the order in which the messages are in the queue, or by getting a specific message. (A program might want to get a specific message if it is looking for the reply to a request that it sent earlier.)
- **Recovery support**: Many message queues offer persistence and logging, allowing for a recovery of state and messages in the queue during failures.
- **Multiple queues**: Some systems allow for multiple queues that can be defined and configured by the application developer. This allows for messages to be routed to the necessary entities based on a publisher or subscriber relationship. Apache Kafka is an example of this.
- **Ease of scalability**: Message queues can horizontally scale to deal with an increase in the message load, as opposed to tightly coupled systems, where scaling and managing the communication traffic and endpoints is harder.

Check your knowledge

1. Which of the following is **not** an advantage of message queues over traditional messaging approaches?

- [x] Message queue systems are simpler to set up and less complex architecturally. ✓

Correct! Message queues add complexity to a system's architecture.

- [ ] Message queue systems allow for loose coupling between communicating entities.
- [ ] Message queue systems provide asynchronous communication through buffering.

Next unit: Message queues: Case study

Continue >