Bionic Buffalo Tech Note #36:

Introduction to the Alarm Clock Protocol (ACP)

last revised Friday 12 June 1998

Overview

The Alarm Clock Protocol (ACP) defines communication with an Alarm Clock Client and an Alarm Clock Server. (In this document, these will be called, simply, the *client* and the *server*.) A server provides time-related services to its clients.

Although ACP may be used in a wide range of applications, the most common use is expected to be in local networks of co-operating, embedded systems, within single platforms among tasks or processes, and among tasks or processes within coupled, parallel processors.

A client may communicate with a server using connectionless or connection-oriented protocols. Each message from the client to the server consists of a sequence of tagged fields. Multiple requests, responses, and notifications can be aggregated within the same message. Aggregation improves efficiency and can be used insure that the tagged fields depart the sender or arrive at the server at the same time. (Aggregation may also occur when the server implementation processes requests in groups, for implementation reasons unrelated to the semantics of the requests.)

There are other protocols, which provide some of these services. However, they were deemed inadequate for the requirements and were rejected. BBC plans to make the ACP standard public, since it fills a niche not occupied by any other known standard.

ACP is designed to inter-operate with software written to POSIX.4. ACP remains aware of multiple clocks, and specifies time values to the nearest nanosecond. On the other hand, ACP's tag-based design allows future extensions for greater precision or other features.

Time, Date, and Interval Specification

ACP time values are specified to the nearest nanosecond, using a seconds/nanoseconds pair as in POSIX.4. However, ACP uses a different scheme for dates. ACP dates are based on the Astronomical Julian Day, a simple count of days since 1 January, 4713 BC. These day numbers advance at noon (rather than at midnight).

The use of day numbers avoids the problem, which will occur in the year 2036 with systems using a count of seconds since the beginning of the year 1900. Since ACP was designed for use by embedded systems (many of which will survive the year 2036), this achieves a significant design goal. A 32-bit signed integer is used for day numbers.

ACP allows negotiation for specific levels of accuracy and precision. Although an interval may be written in terms of nanoseconds, clients and servers may understand that less accuracy or precision may be expected.

Absolute times may be specified in terms of any of several clocks.

Network Delays

ACP does not measure or allow for network delays. Any client wishing to account for delays, jitter, congestion, or other such values should use another protocol (such as NTP) to determine appropriate adjustments.

Although a server recognizes multiple clocks, it makes a “best effort” attempt to perform its tasks as quickly as possible, without anticipating errors, which might be expected to occur from local system performance or network speed. ACP allows for messages to be time-stamped, giving enough information to clients to permit adjustment for these factors.

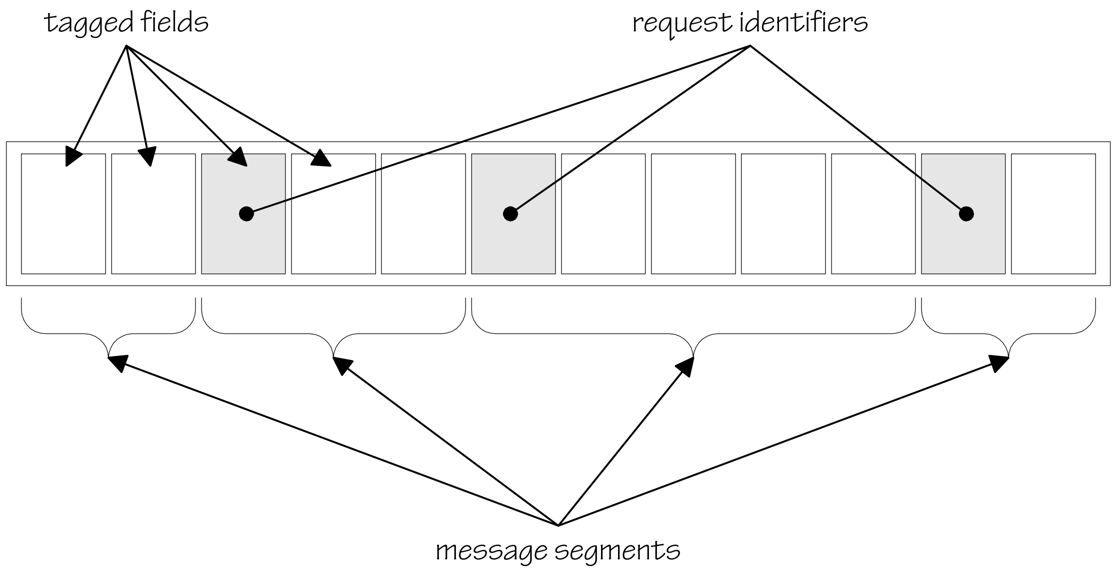
Basic Protocol

All messages use a common format. Each message consists of zero or more tagged fields. A tagged field contains of a tag followed by data. The tag determines the form and meaning of its associated data item.

One type of tag specifies a *request identifier*, which identifies a specific request which is being made or which has been made by a client to a server. The client assigns half of the request identifier when the request is made, and the server assigns the second half when responding. A client or server may choose to use either or both halves of the request identifier when comparing or testing its value. If no response is requested, then the client may not know the server’s half of the identifier, and the server cannot rely on the client’s half to be unique among all clients.

The request identifier tags divide a message into segments, each beginning with a request identifier and ending just prior to the next request identifier.

The tagged fields in a message segment apply to the request identified at the beginning of the segment. If any tagged fields appear before the first request identifier, then those tagged fields apply to all requests in the message.



A new request is created when a client sends a message containing a new request identifier. Each request identifier has a finite lifetime. The lifetime of a request identifier is defined by the client, and may be extended later by the client. Clients which power down or which otherwise separate from the network might leave out-standing requests, but eventually those requests will expire.

Each request specifies an action to be performed by the server. The available services are listed in the next section.

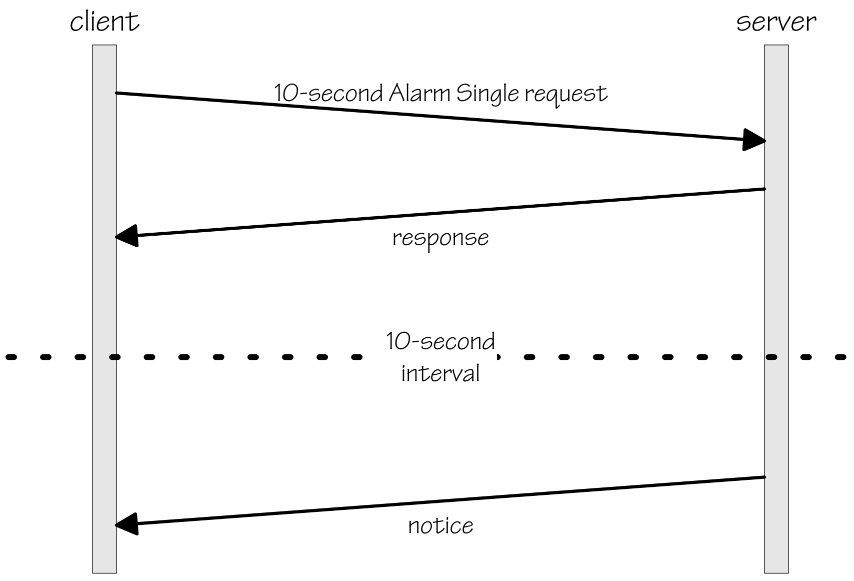
A message segment (associated with a specific request identifier) is identified as a *request*, a *response*, or a *notice*.

- A request is used to ask the server to perform a certain action. Note that a subsequent, related request uses a separate request identifier. For example, a request for an ALARM SINGLE might have identifier 1234, while a request to CANCEL the first request might have identifier 5678.

- A response optionally may be sent in response to any request. It may contain requested information, status, or exception codes. If the request involves a delay, then a response is sent after the requested service is scheduled, otherwise the response is sent after the service is performed.

- A notice is a message sent by the server after an intentional delay. For example, if an Alarm Single is requested after an interval of 10 seconds, then the response would be sent immediately and the notice 10 seconds later.

The following diagram illustrates this example for the case when the request is *not* cancelled:



Note that this diagram shows the flow of message segments. There may also be other message segments within each message.

Notice or notification messages may optionally be broadcast, rather than directed at specific machines. This may be more efficient when multiple machines are listening together for the same notification (of events, alarms, or lifetime expirations).

Certain requests may be broadcast to all available servers. This may be used to discover the available servers.

The Alarm Clock Protocol can sit above most transport protocols, including TCP/IP, UDP/IP, and message passing operating systems. If the transport mechanism does not allow the server to determine the origin of a message, then tagged fields within the message may contain this information so the server can respond to the client.

Two well-known addresses are required to implement ACP. Clients use one well-known address for two purposes:

- clients have a known destination for request messages when using datagrams
- clients have a known destination to establish a connection with a server (in fact, this is simply a specialized form of the first use)

Servers use the second well-known address when notification messages are to be broadcast simultaneously to multiple clients.

Services

This section lists the services available from a server, which supports the Alarm Clock Protocol.

Alarm Periodic

The server sends a sequence of messages to the client at fixed intervals.

Alarm Single

The server sends a message to the client after a specified delay or at a specified time. The message may contain a data field optionally provided by the client at the time of the request. This function may be used to provide timeouts for processes which anticipate being blocked during message read operations.

Cancel

The server cancels the specified request.

Connect

The server provides a specific address or port for a connection. As with other requests, connections expire after the stated interval and must be renewed before expiration.

If there is no connection, then the server operates in connectionless (datagram) mode.

Disconnect

The server terminates the connection.

Event Define

The server returns a handle for a new event. An event may be singular (occurring not more than once) or recurring (occurring zero or more times).

Event Subscribe

Requests that a message be sent when an event is triggered. The client provides the event handle as part of the request. If the event has not been triggered at the time it's lifetime expires, then the server optionally will send an expiration notice to subscribers.

Event Trigger

The server notifies subscribers that an event has occurred.

Renew

The server extends the lifetime of the specified request.

Request Status

The server returns the status of the specified request.

Resource List

The server provides a list of resources of the specified types. If clocks are listed, then precision, accuracy, and base will be provided in the list.

Time of Day

The server returns the current time and date.

Application Examples

This section describes three reference applications using the Alarm Clock Protocol. These applications defined the original design goals of the protocol.

Embedded System

The embedded system reference application consists of multiple, networked embedded systems. Typically, these occur inside a complex machine such as an automobile, aircraft, or in manufacturing machinery.

There may be from several to hundreds of processors in such an environment. These processors usually require some degree of time or event synchronization. Although many designs implement centralized control of synchronized operations, that approach does not always easily permit add-on components which were not anticipated in the original design. ACP provides a standardized design for components, which can be used in diverse systems.

ACP allows individual nodes to be designed without precision clocks, since all nodes can share a common, precision timebase. The event subscription services also provide a way for an application simply to trigger preset operations on multiple nodes.

Portable Application

When writing an extremely portable application using multiple processes or tasks, the design must be based on the “lowest common denominator” services expected to be available on potential target platforms. This includes the following restrictions:

- applications can use, *but may not require*, multiple threads of execution within a process
- timeouts for i/o operations may not be available
- shared memory may not be available

The resulting application is usually based on a message-passing paradigm, which can be efficiently implemented with shared memory when shared memory is available. In this context, the use of messages for timer services fits well.

A network read can be timed out by using the Alarm services. By scheduling a notification message to arrive periodically or after a designated time, an application will not expect to wait indefinitely on a port if the expected message does not arrive.

Office or Home System

Office or home environments consist of numerous devices of varying levels of complexity. These may consist of appliances, computers, peripherals, and network components.

In this environment, the primary motivation for ACP is the provision of *standardized* services, so additional devices may be added over time. Each home or office might be expected to have at least one ACP server, which can synchronize the clocks on appliances and computers, and provide a conventional means for co-ordinating operations performed by co-operating devices.

For instance, a standardized appliance expected to turn itself on or off at specified times might find, at a well-known address, a centralized controller which has acquired an event handle in advance. The ACP server (which well might be hosted in the same central controller) would send the notices as required when the events were triggered.

Additional Considerations

ACP may need future expansion to address some issues, which have not been discussed here. Among those issues are:

1. *Security*. Although an underlying transport may be used to enforce certain security rules, such a secure transport is not always available, nor may it be flexible enough to provide the desired protection. Future versions of the protocol may add tagged fields, which allow authentication operations between client and server.

2. *Fault tolerance*. If a server fails or is taken down, there is no mechanism for transferring pending requests to a new server. Such a mechanism would require the introduction of new service types, with a “transfer” or “forward” request being made by the server to the client.

3. *Quality-of-service negotiation*. There is a need to provide a mechanism for QOS negotiations between client and server.

4. *Load management*. There is no mechanism defined to balance, limit, or otherwise manage the load on a server. The quality of service may vary with the load, and among servers due to uneven loading.

5. *Server synchronization*. Although *ad hoc* mechanisms may be used to synchronize servers with one another, a standardized procedure would reduce application complexity, make behaviour more deterministic, and simplify management.

The current version of the protocol includes tagged fields for revision information. As much as possible, future revisions will be made compatible with the current version. No special compatibility, interoperability, or upgrade problems are anticipated.

For More Information

For more detailed information, the reader is referred to the *Alarm Clock Protocol Reference Guide*, by Bionic Buffalo Corporation. The reference guide includes detailed protocol specifications, message formats, and a standard API for applications which use the protocol.

In addition, Bionic Buffalo licenses uncommented source code to implement the protocol on a wide range of platforms. Executable code for specific environments is also available.