

Event-based Requests

Classic support

Thu, Jan 13, 2005

Classic protocol provides support for clock-event-based requests as well as for one-shot and periodic requests. It also provides special support for page applications that may have to await the arrival of replies from contributing external nodes to a given request. This special support comes by way of the routine `Collect`, which tries to wait for nodes that have not replied recently enough consistent with the request period. This works fine for periodic requests, but it is not currently working for event-based requests. This note explores how this waiting support in `Collect` works for periodic requests and develops how to provide the same support for event-based requests.

Cast of variables

Several key fields must be described that help provide the required waiting support. For the Type #1 requests that are created by a page application call to `ReqData`, for the case that at least one external contributing node is involved, a special structure is built for each external contributing node that is referenced in the request. Each structure is as follows:

<i>Field</i>	<i>Size</i>	<i>Meaning</i>
<code>X_NODE</code>	2	External node#
<code>X_TALLY</code>	2	Tally of idents
<code>X_STAT</code>	2	External answer fragment status
<code>X_ABUF</code>	2	Offset to external answer buffer
<code>X_SPAR</code>	2	(spare)
<code>X_SIZE</code>	2	#bytes in external answer buffer
<code>X_AGE</code>	2	Time in cycles since last update
<code>X_CNTR</code>	2	Countdown until request reissue

The `x_AGE` field is used with special care to support the waiting logic. It is set to -1 when the request is initialized, which means that a reissue is pending if no reply from the external node is received soon, say, within about 2 cycles. (A prompt reply is expected for a periodic request.) It is set to a copy of the request period (in cycles) when a reply is received from the external node. When the `update` task runs early in each cycle, and if `x_AGE` is ≥ 0 , it is decremented by 1. If `Collect` is called when `x_AGE` is zero, it tries to wait for a reply from the external node. Follow this logic for a request period of 1 cycle, meaning 15 Hz replies, and it is obvious that this logic is suitable for supporting the appropriate waiting. All this depends upon the nodes involved operating synchronously, with each node's cycle activity beginning at the same time.

<i>ms in cycle</i>	<i>x_AGE</i>	<i>Action</i>
0	1	Start of 15 Hz cycle (left over from previous cycle)
1	0	<code>update</code> task decrements <code>x_AGE</code>
4	0	Page application calls <code>Collect</code>
15	1	Arrival of reply from external node
16	1	Return from <code>Collect</code> with latest reply data

If `Collect` times out, meaning it is still waiting for a reply from an external node at 50 ms into the current cycle, the `x_AGE` field is set to -1 , with the `x_CNTR` field set to ± 30 depending on whether a reply has ever been received since the request was initialized. This gives a 2 second delay before a reissue of the request is sent. This covers the case of a node that was down when the request is initialized (returned status = 8) as well as the case of a node that died since the request was initialized (returned status = 7). If a reply is received during the 2 seconds, the pending reissue is canceled. This 2 second time out is handled by the `update`

task logic, such that if the value of `x_CNTR` is reduced to zero while the value of `x_AGE` is `-1`, implying 2 seconds has elapsed since `Collect` found an apparently missing reply, then the original request is reissued to that node in hopes that it will come alive soon and start contributing to the request. A node that is down therefore receives reminders every 2 seconds as long as the request is active.

Summarize the logic components that exist today:

<i>Task</i>	<i>Routine</i>	<i>Action for periodic/one-shot</i>	<i>Action for clock event case</i>
App1	ReqDat	<code>x_AGE = -1</code>	<code>x_AGE = -1</code>
Update	ExtCheck	decrement <code>x_AGE</code> iff ≥ 0	(ExtCheck not called)
Classic	EXTAN	<code>x_AGE = period (=1 if one-shot)</code>	<code>x_AGE = event!!</code>
App1	Collect	If <code>x_AGE = 0</code> , wait. If time-out, <code>x_AGE = -1</code> .	If <code>x_AGE = 0</code> , wait. If time-out, <code>x_AGE = -1</code> .

The present logic is not fully prepared for supporting clock event-based requests properly, which is the reason for this note.

How might similar logic proceed for the clock event case? If the `Update` task, recognizing that the event upon which a request awaits is true this cycle, sets `x_AGE` to 0, then `Collect` can enter its waiting logic. If the arrival of a new reply sets `x_AGE` to 1, then `Collect` will stop waiting and return the fresh data. If `Update`, for a cycle in which the event is false, does nothing, then it will, without waiting, build and return the same data to the caller of `Collect`. This may even be desirable. Normally, an application may want to check for the occurrence of the event of interest (via `HaveEvt`) prior to calling `Collect`.

Suggestion for improved logic components:

<i>Task</i>	<i>Routine</i>	<i>Action for periodic/one-shot</i>	<i>Action for clock event case</i>
App1	ReqDat	<code>x_AGE = 0</code>	<code>x_AGE = 2</code>
Update	ExtCheck	decrement <code>x_AGE</code> iff > 0	<code>x_AGE = 0</code> iff event true
Classic	EXTAN	<code>x_AGE = period (=1 if one-shot)</code>	<code>x_AGE = 1</code>
App1	Collect	If <code>x_AGE = 0</code> , wait. If time-out, <code>x_AGE = -1</code> .	If <code>x_AGE = 0</code> , wait. If time-out, <code>x_AGE = -1</code> .

The change in initialization of `x_AGE` means for the periodic/one-shot case that an immediate call to `Collect` will result in waiting for the reply. This can allow a call to `Collect` to immediately follow a call to `ReqData` with the result of awaiting the prompt reply. This can work because `Collect` calls `NextTask` while it is waiting in order to allow other tasks to run. But it depends upon the call to `ReqData` being able to initiate sending the request message to the network, which it does not now do. At present, the `App1` task, after the page application has returned and just before it ends its 15 Hz execution, sends a signal to the `Update` task that results in flushing the network queue and thereby transmitting the request to the external nodes. A change should be made in `ReqDat` to send this signal to `Update` before it returns.

For the clock event case, setting `x_AGE = 2` at initialization ensures that `Collect` is not set to wait if it should be called right away. This is appropriate because an event-based request does not result in a prompt reply. If an application called `Collect` right after `ReqData`, it would receive an 8 error status right away, because `ReqData` initializes the status word to 8. The data that would be returned is updated for the local node but all zero for external nodes. As mentioned above, it is a good idea for an application not to call `Collect` for an event-based request unless it knows that the event is true for the current cycle. (Setting `x_AGE = 1` at initialization would give the same result as setting it = 2, but using a different positive value for the event case may have some diagnostic interest.)

In the `update` task, for the clock event case, the `x_AGE` field is set to 0 only if the event is true. This enables the waiting logic in a call to `collect`, allowing the replies from external nodes to be received before returning the reply data to the application. As long as the event is false, the `update` task will not change it, so it likely remains the positive value it was set to when a reply was last received. If the waiting logic in `collect` is entered because `x_AGE = 0`, and it times out, it sets `x_AGE = -1`, which will result in a reissue of the request in 2 seconds, provided no reply is received within that time. If the clock event never becomes true, `x_AGE` will never be set = 0, no waiting will occur, and no reissues will occur.

Again, for the clock event case, if a reply is received, `EXTAN` sets `x_AGE = 1`. (This fixes a bug.) This ensures that `collect` will not wait. Such a reply should arrive on the cycle in which a clock event is true, when `update` will have set `x_AGE = 0`, so this is ok. Any waiting logic will thereby cease waiting.

Detailed changes

Initialization in `ReqDat` module:

If clock event case, initialize `x_AGE` to 2; otherwise, set it to 0.

Send signal to `Update` task to cause network queue to be flushed.

Routine `UDPCHAIN` in `Update` task:

Change to call `EXTCHECK` in any case, rather than avoiding it for the clock event case.

Routine `EXTCHECK` in `Update` task:

Before loop, for clock event case, set `D4.B` nonzero if clock event true, else `0x00`. If not clock event case, set `D4.B = period`.

Inside loop, if `Ext Req Block`, issue resend only if clock event case or `D4.B` nonzero. (One-shot requests do not result in reissues.)

Inside loop when `x_AGE` is not `-1`, if clock event case and event true, set `x_AGE = 0`. But if not clock event case, decrement `x_AGE` if `> 0` unless one-shot case.

Routine `EXTAN` in `Classic` task:

If clock event case, set `x_AGE` to 1; else, set `x_AGE` to request period, or to 1 if one-shot.

Server node

The logic described here for support of data requests by page applications is largely used to support Server requests, in which a client node sends a request with a special flag bit set that asks for server node support. The node forwards the request on behalf of the client, accepting all the replies from the external nodes and collecting them (via `collects`) to be delivered in one reply to the client node. The `collects` routine does not wait because it is meant to be called by the `server` task that normally runs 40 ms into the cycle, considered a deadline by which replies from all contributing nodes should have been received.

Summary

We have explored the logic related to monitoring the replies from contributing nodes in response to a Classic protocol data request. Improvements are designed to allow a page application to invoke `collect` immediately following a `ReqData` call to await the prompt reply to a one-shot or periodic request. Changes are made for handling clock event-based requests more sensibly, including support for reissuing requests to external nodes that fail to reply to such a request.