

Larger Internal Ptrs

Suggested ideas

Tue, Oct 8, 2002

Suppose it were important to support internal ptrs that require more than 32 bits to express. One could imagine allocating a block of memory that could contain whatever is needed. The internal ptr would be the address of the allocated block. To make this work, though, we need to find a way to make sure that the allocated blocks are freed when the request is canceled. There may have to be some clean up code that is executed when a request is freed, according to the particular read-type routine used in the request. ??

A particular listype could have another attribute, one that specifies whether such block releasing is needed. Its job would be simple; merely traverse through the internal ptr array, and free each block referenced. Or, maybe we can add another field to a request block which can be a header for a linked list of block pointers. While initializing the request, if a block must be allocated, add its address to the linked list. When freeing this kind of block, the cancel code would check this header, and if it is non-NULL, free all the blocks in the linked list.

A less elaborate approach might be to merely double the size of the internal ptr. This would affect a lot of code if it were done in general. Can it be done by the listype? Right now, the determination of the space required for an array of internal ptrs is fixed and done before actually generating the internal ptrs. We would need a way for that code, which may be preparing to allocate the request support block, to know how much space to set aside for each internal ptr for each ident. Then the ptr-type routine and the read-type routine would be the only ones to know how much space is needed for each.

Suppose the upper two bits in the byte that contains the ptr-type code were used to specify any additional longwords needed for internal ptrs. These bits would normally be zero, but they could be used to add 4, 8, or 12 extra bytes to the internal ptr field. We need to examine what code cares about the ptr-type field. The `LTTPTT` routine could function as now, but return only the low 6 bits of the ptr-type byte. Another routine, `LTTPSZ`, could be defined to return the new 2-bit field. An alternate way to do this via a new `LTTPSZ` is to use another array that is indexed by ptr-type value. Or even simpler, one can add code whenever a new ptr-type routine needs more room for its internal ptr structure.

Only `LTTPSZ` should be the source of information about the size of an internal ptr, and it could always depend upon the ptr type. What's more, the ptr-types need only be those that are 32 and above. Right now, the maximum ptr-type number is 46, so we are only talking about 15 different values. This scheme would assume that ptr-types that correspond to system tables do not have a need for an internal ptr format longer than 4 bytes. Armed with `LTTPSZ`, whose argument should be the listype id, any code would need to scan through all listypes to build a count of longwords, say, needed for the complete internal ptr array needed by the request. Code that does this is both `ReqDat` and `PReqDat`, and `ACReq` and `FTPMan`. The `ACReq` code already uses 8-byte internal ptrs in some cases, so this new scheme should not mess with that logic, such as for building an average of data when requested at reply rates of less than 7.5 Hz.

At the time the size of the internal ptr structure is required, only the listype number is available. It has not been broken down into a listype-id, ident-type, or ptr-type yet. So, even though the logical argument to `LTTPSZ` is the ptr-type number, it could be the listype number

instead. It could return -1 , say, for an error; otherwise, it can return the number of longwords needed for the internal ptr structure for the given listype. This result would nearly always be 1, but it could be more than 1 for some listypes.

Example of larger internal ptr

The motivation for this suggested support for larger internal ptrs is the support required for fast digitizers. The internal ptr structure needs to retain the location where in a circular buffer the sampling of data last left off. For the 1 KHz digitizer, this offset is a 16-bit byte offset in a 64K byte circular buffer. For the 10 KHz digitizer, we need to retain at least 20 bits as a word offset within a 2 MB, or 1 MW, circular buffer. Because we need to also retain an 8-bit clock event number, we feel the squeeze. There also needs to be an id to indicate which block of consecutive channels is referenced, since it is necessary to know where the circular buffer is located and also where the registers are located and where the list of time-stamps for each slot. One also needs to know the number of sets of data within the circular buffer. The code that supplies the return data needs access to all of these parameters, which may vary depending on the ident. So, what format might a larger internal ptr structure take?