Message Passing with Queues

Rosemary C. Leary

University of Rochester
Laboratory for Laser Energetics
250 East River Road
Rochester, NY 14623

Donald P. McClimans

Inti Electronics Corp.
108 Boardman
Rochester, NY 14607

Abstract

General message passing software for intertask communication has been written in URTH using first-in first-out queues. These queues can be of any length, and either memory or disk resident. The disk resident queues may be several blocks long. The width of a queue entry is also arbitrary, and can range from a single byte to the queue length. This software was successfully used for intertask communication among the seventeen tasks of the Omega Laser Alignment System.

The Omega Laser Alignment System is a Forth real-time application that controls large numbers of devices such as shutters, mirrors, detectors, and focusing lenses. It is implemented on a PDP-11/23 computer with 256 K bytes of memory and a 5 M byte disk. Seventeen different tasks in a multi-tasking configuration perform the various functions needed by the laser system.

The alignment system software has a number of requirements for general message passing, including messages between tasks, messages to output devices (displays), and messages from input devices (keyboard). The queue software described here allows all message passing applications to use the same vocabulary, and allows message buffers to be either disk or memory resident.

In order to be general, messages are not associated with a task or operation. Rather, we define a QUEUE to be a first-in first-out (FIFO) buffer which any task, program, or word can put entries into and/or take entries out of. Entries can be put into the queue by one task and taken out by the same or a different task. More than one task can insert or remove entries from the same queue.

Queues can be any length. Disk resident queues can span several blocks if necessary. The smallest item that can be placed in a queue is one byte long, and queues are, generally, byte oriented. However, entries in the queue can be any size, from one byte up to the entire queue size.

Overview of Queue Words

A number of queue operations are required to handle queues in a general fashion. In addition, variations of the basic functions are necessary to handle different applications. The message passing software includes words to:

1. Define a queue, allocating memory and/or disk space for the buffer.
2. Select a queue for activity. All future queue words will operate on this queue.
3. Store an entry into a queue.
4. Retrieve an entry from a queue.
5. Retrieve the status of a queue. Status information includes whether the queue is full or empty, the number of spaces left in the queue, and the total size of the queue.

Different words are used for different sizes and types of queue entries. Entries may be one byte long, one word long, or an entire buffer. Multiple byte entries may be processed in one of two forms: in standard FORTH format (address and byte count on the stack), or in string format (address on stack, count in first byte).

Some of the words have 'special-function' versions. These special versions allow you to examine the next item in the queue without removing it from the queue, and to handle 'queue-full' and 'queue-empty' conditions in a variety of ways.

Queue Naming Conventions

Queues can be associated with any name that the programmer chooses, simply by declaring a queue of that name using the QUEUE word. However, because queues are often associated with tasks and task scheduling, for the alignment system we have adopted some naming conventions for common queue uses.

Any task which has a single input queue for input messages from other tasks, will normally name that queue

taskname-MSGQ

where 'taskname' is the name of the task as specified in the BLDTASK word.

All queue operation words, including the queue names and their associated buffers, are in the URTH vocabulary, and are known to all programs. The state vector variable CURQ points to the currently selected queue for each task, or is zero if no queue is selected.

Queue Entry Removal

Normally, when you remove an entry from a queue, it is permanently removed and is inaccessible to any other task or word which may want to access it. If, however, you wish to retrieve an entry from a queue, but leave that entry in the queue, you can specify one of the words QGET-NR, QWGET-NR, QBGET-NR, or QSGET-NR. The suffix '-NR' specifies 'no removal', and means that the next entry is retrieved from the queue, but that no changes are made to entries in the queue, and nothing is removed from it.

Queue Space Considerations

In theory, queues have infinite size, never overflow, and always contain at least as many entries as you want to remove. In practice, however, we implement queues using some sort of finite size buffer, and it is quite possible for queues to not have enough room left for your entry (queue-full), or to not have any entries left to remove (queue-empty).

In some applications, the standard queue processing words provide acceptable default actions in these situations. For applications that require more sophisticated action regarding queue-full or queue-empty conditions, words are defined which provide the information needed.

The queue store words, QPUT, QWPUT, QBPUT, QSPUT, Q", and QS", perform the following action when they encounter a queue which does not have enough room to store the entry: they simply PAUSE, and then retry the entry when control is returned. If room is still not available, they continue to PAUSE. Note that this can lead to infinite suspension, if no other task of the same or higher priority is removing items from the queue.

The words QPUT, QSPUT, Q", and QS" will PAUSE unless they can put the entire message into the queue. This insures that the message goes into the queue without intervening characters. Similarly, QWPUT will PAUSE unless it can put the entire word into the queue.

If you require better control over queue-full conditions, you can use the words QPUT-FULL?, QWPUT-FULL?, QBPUT-FULL?, and QSPUT-FULL?. These words attempt to enter the item into the queue only once. If they are successful, they return the value 0 (false) to the top-of-stack. If they are unsuccessful, they return the value 1 (true) to the top-of-stack. Thus, you can use code such as:

```
... QPUT-FULL? IF ( code for queue-full ) THEN ...
```

Alternatively, you can use the words QSPACE and QFULL? to determine if there is enough room in the queue for your entry.

The queue retrieval words, QGET, QWGET, QBGET, and QSGET perform the following action when they encounter a queue which does not contain enough bytes to satisfy the request: they simply PAUSE, and then retry the request when control is returned. If enough bytes are still not available, they continue to PAUSE. Note that this can lead to infinite suspension if no other task of the same or higher priority is putting items into the queue.

The word QGET will PAUSE unless it can return as many bytes as were requested. Similarly, QWGET will PAUSE unless it can retrieve an entire word from the queue.

If you require better control over queue-empty conditions, you can use the words QGET-EMPTY?, QWGET-EMPTY?, QBGET-EMPTY?, and QSGET-EMPTY?. These words attempt to retrieve the item from the queue only once. If they are successful, they return the value 0 (false) to the top-of-stack. If they are unsuccessful, they return the value 1 (true) to the top-of-stack. Thus, you can use code such as:

```
... QGET-EMPTY? IF ( code for queue-empty ) THEN ...
```

Alternatively, you can use the words QLENGTH and QEMPTY? to determine if there is an entry in the queue for you.

Queue Word Descriptions

The following words form the standard QUEUE processing words for the URTH Alignment System. They are all a part of the URTH vocabulary.

QUEUE [residency] [size] --- 'Q-name'

QUEUE is a defining word which creates a queue. The 'residency' argument tells whether this queue is to be disk or memory resident. If the residency word is less than 5000, the queue is disk resident. Otherwise it is memory resident in mapped memory "user" space, and the residency word contains the "user" space virtual address of the start of the queue.

In either case, the word 'Q-name' is entered into the dictionary. The residency word is saved as the first parameter, with the SIZE, FIRST, LENGTH, MAXLEN, and TASK parameters following it. If the queue is memory resident, the queue buffer will be in mapped memory. If it is disk resident, the buffer will be on disk.

The 'size' argument specifies the size of the queue buffer in bytes. This will be at least two bytes and is rounded up to the nearest even number. Typical values for 'size' are in the range of 100 to 1000 bytes.

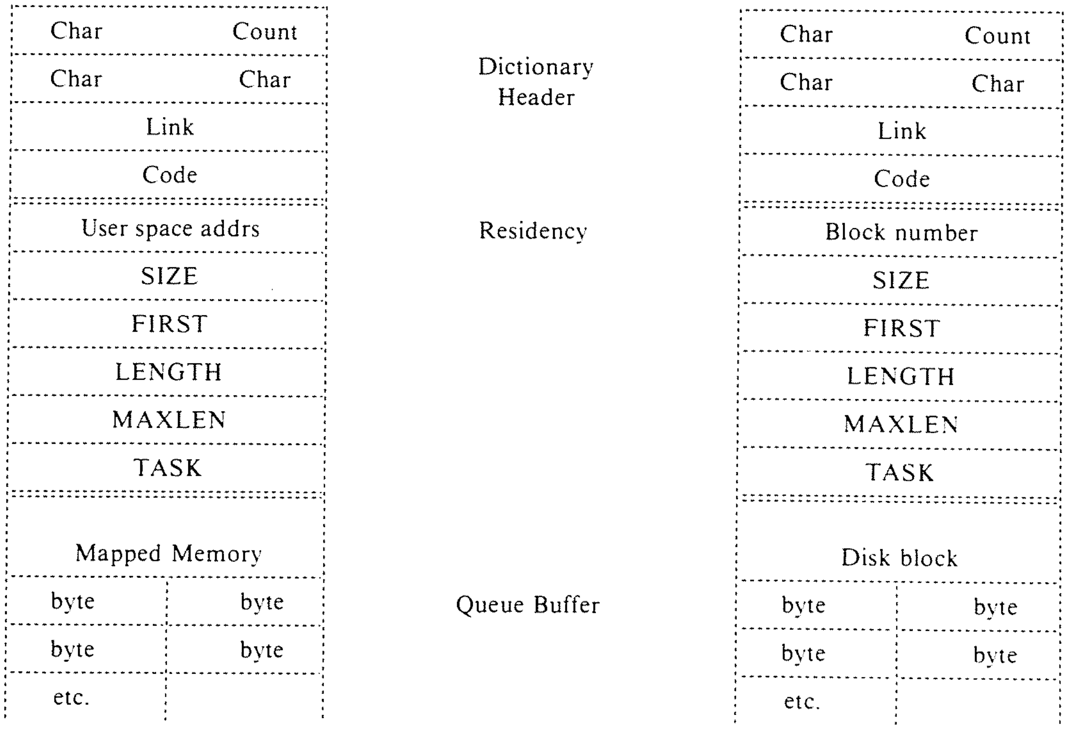

Figure 1. Memory Resident Queue Format

Figure 2. Disk Resident Queue Format.

QUEUE has two additional effects besides simply allocating buffer space:

1. The currently selected queue for this task is set to be Q-name (see QSELECT below).
2. When the word Q-name is invoked after being defined in this manner, it will place the address of the beginning of the queue buffer header (residency word) onto the top-of-stack.

QSELECT [Q-address] --- [ ]

QSELECT sets the specified queue to be the current queue by storing its address in the state vector variable CURQ. All further words which operate on queues will operate on this queue until a new queue is QSELECTed or defined using QUEUE. Since each task has its own copy of CURQ, each task that uses queues must create or QSELECT its own queue. The same queue can be used by more than one task, however.

QBPUT [byte] --- [ ]

QBPUT moves one 8-bit byte from the TOS to the next entry in the currently selected queue. If the queue is full, QBPUT PAUSEs until there is space available.

QBGET [ ] --- [byte]

QBGET moves one 8-bit byte from the beginning of the currently selected queue to the TOS. If the queue is empty, this word PAUSEs until there is a byte in the queue.

QWPUT [word] --- [ ]

QWPUT moves one 16-bit word from the TOS to the next entry in the currently selected queue. If there is not enough room in the queue for another full word entry, this word PAUSEs until there is space available.

QWGET [ ] --- [word]

QWGET moves one 16-bit word from the beginning of the currently selected queue to the TOS. If there is not a full word in the queue, this word PAUSEs until a full word item is available in the queue.

QPUT [address] [count] --- [ ]

QPUT moves a buffer, in its entirety, to the next entry in the currently selected queue. It expects a byte count on the TOS, and the address of the first word in the buffer one beneath the TOS. If there is not enough room in the queue for this entry, QPUT will PAUSE until there is space available. Note that if the length of the buffer is longer than the size of the queue, QPUT will PAUSE forever.

QGET [address] [count] --- [ ]

QGET moves a multi-byte message, in its entirety, from the currently selected queue to a specified buffer. It expects a byte count to be on the TOS, and the address of the first word of the destination buffer to be one beneath the TOS. If there are not enough bytes in the queue to satisfy the request, this word PAUSEs until enough bytes become available.

QSPUT [address] --- [ ]

QSPUT moves a buffer containing a string in the conventional string format into the currently selected queue. It expects the address of the string on the TOS. The byte count must be in the first byte at that address. If there is not enough space in the queue for the whole string, this word will PAUSE until space becomes available.

QSGET [address] --- [ ]

QSGET removes a string in the conventional string format from the currently selected queue. It expects on the stack the address of the buffer into which to put the string. The byte count of the string must be in the first available byte from the queue, and the remaining characters must follow. If the number of bytes available is less than the byte count, this word will PAUSE until they become available.

Q"	Q" text"

Q" moves ASCII characters directly from the input stream into the currently selected queue. It is used in conjunction with a trailing double-quote, which marks the end of the character string. All characters immediately following the space after the Q", up until the following ", will be placed into the queue. For example, the code:

Q" ERROR MSG"

will place the character string 'ERROR MSG' into the currently selected queue. Like QPUT, this word will PAUSE if there is not enough room in the queue for the entire character string.

QS"	QS" text"

QS" is similar to Q" except that it puts the string of characters into the queue preceded by the byte count, in the conventional string format.

QPUT-FULL?	[address] [count] --- [T/F]
QWPUT-FULL?	[word] --- [T/F]
QBPUT-FULL?	[byte] --- [T/F]
QSPUT-FULL?	[address] --- [T/F]

These four words act similarly to the words QPUT, QWPUT, QBPUT, and QSPUT. They differ in that they never PAUSE due to a queue-full condition, and they leave a value on the TOS which indicates whether the requested action was successfully performed. A value of 0 (false) indicates that the PUT operation was successful, while a value of 1 (true) indicates that there was not enough room left in the queue to add the items. See the earlier discussion of queue-full conditions.

QGET-EMPTY?	[address] [count] --- [T/F]
QWGET-EMPTY?	[ ] --- [word] [T/F]
QBGET-EMPTY?	[ ] --- [byte] [T/F]
QSGET-EMPTY?	[address] --- [T/F]

These four words act similarly to the words QGET, QWGET, and QBGET. They differ in that they never PAUSE due to a queue-empty condition, and they leave a value on the TOS which indicates whether the requested action was successfully performed. A value of 0 (false) indicates that the GET operation was successful, while a value of 1 (true) indicates that there was not enough data in the queue to satisfy the request. See the earlier discussion of queue-empty conditions.

QGET-NR	[address] [count] --- [ ]
QWGET-NR	[ ] --- [word]
QBGET-NR	[ ] --- [byte]
QSGET-NR	[address] --- [ ]

These four words are similar to the words QGET, QWGET, and QBGET. They differ in that they do not actually remove any bytes from the queue. This allows you to inspect items in the queue before removing them. See the earlier discussion on Entry Removal. These words will PAUSE if there is not enough data in the queue to satisfy the request.

QEMPTY?	[ ] --- [T/F]

QEMPTY? is used to tell whether the currently selected queue is empty. It returns 1 (true) to the TOS if the queue is empty, and 0 (false) if it is not.

QFULL?	[ ] --- [T/F]

QFULL? is used to tell whether the currently selected queue is full. It returns 1 (true) to the TOS if there is no room left in the queue, and 0 (false) if there is room for at least one byte more.

QSPACE [ ] --- [available space in bytes]

QSPACE returns the amount of space available in the currently selected queue. It places the number of bytes remaining in the queue on the TOS.

QLENGTH [ ] --- [current length in bytes]

QLENGTH returns the length of the currently selected queue. It places the number of bytes now being used on the TOS.

QSIZE [ ] --- [buffer size in bytes]

QSIZE returns the total size of the currently selected queue. It places the number of bytes in the entire queue (ie, the maximum length) onto the TOS. Note that we have:

$$QSIZE = QLENGTH + QSPACE$$

QMAX [ ] --- [longest length in bytes]

QMAX returns the maximum size that the queue has grown to since it was created. This value is useful for 'tuning' queue sizes.

.QUEUE [ ] --- [ ]

.Q is a debugging aid. It prints the name, residency, header words, and contents of the currently selected queue.

QPRINT [ ] --- [ ]

QPRINT empties the currently selected queue and prints the contents as 16-bit integers.

Queue Implementation

Queues are implemented using circular buffers. A buffer is allocated of a certain length, with pointers to the first and last bytes in the queue. The last byte of the buffer is logically followed by the first byte of the buffer, in a 'circular' fashion.

When a queue is created, a maximum length must be specified as a parameter to the defining word QUEUE. This maximum length is known as the SIZE of the queue.

The created queue consists of five header words and a queue buffer containing the required number of bytes. The queue buffer must be at least two bytes long and is rounded up to the nearest word.

The queue header consists of the following:

Word 0 RESIDENCY	If this word is less than 5000, the queue is disk resident, and this word contains the block number. Otherwise it is resident in mapped memory and this word contains the address in "user" mode virtual address space.
Word 1 SIZE	The maximum length of the buffer in bytes, not including the first five words. This is equal to the SIZE specified for the defining word QUEUE, rounded up to an even number of bytes. The minimum value of SIZE is 2.
Word 2 FIRST	A byte pointer to the first entry in the queue. The byte pointed to by FIRST is the next to be removed from the queue.
Word 3 LENGTH	A byte counter of the current number of bytes in the queue.

Word 4 MAXLEN	A byte counter of the maximum length of the queue. Whenever a byte is added to the queue, MAXLEN is set to the maximum of its old value and the new value of LENGTH.
Word 5 TASK	Address of the task to be DISPATCHed when anything is put into this queue. If this parameter is zero, no task will be dispatched.

The following conventions are used to determine whether the queue is full or empty:

1. If LENGTH is equal to 0 the queue is empty.
2. If LENGTH is equal to SIZE the queue is full.

If neither of these conditions holds, then the buffer contains some data, but is not full.

The number of bytes remaining in the queue can be calculated by:

SPACE = SIZE - LENGTH

The next item to be removed from the queue is given by the value of FIRST. After removing a byte from the queue, FIRST is incremented modulo SIZE.

The next item to be added to the queue is added in the location pointed to by:

NEXT = (FIRST + LENGTH) modulo SIZE

After adding a byte to the queue, LENGTH is incremented modulo SIZE.

Figures 3 and 4 give examples of different queues in different conditions.

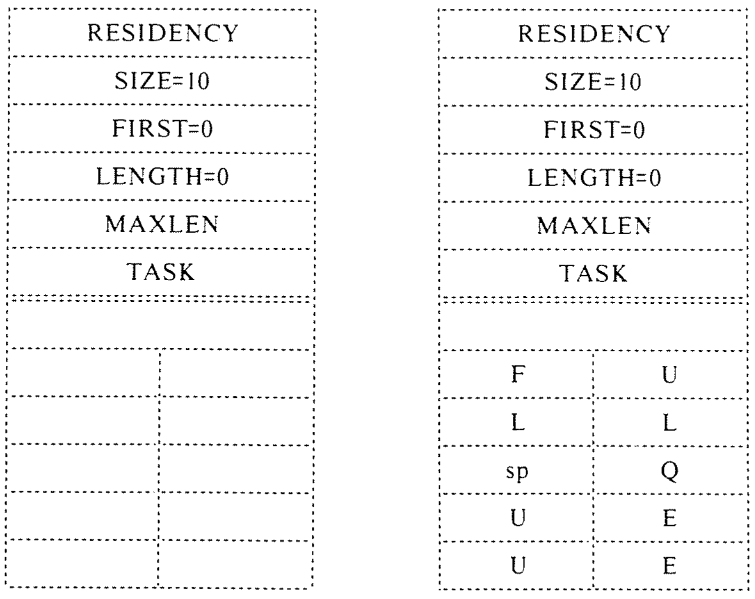

Figure 3. Examples of Empty and Full Queues.

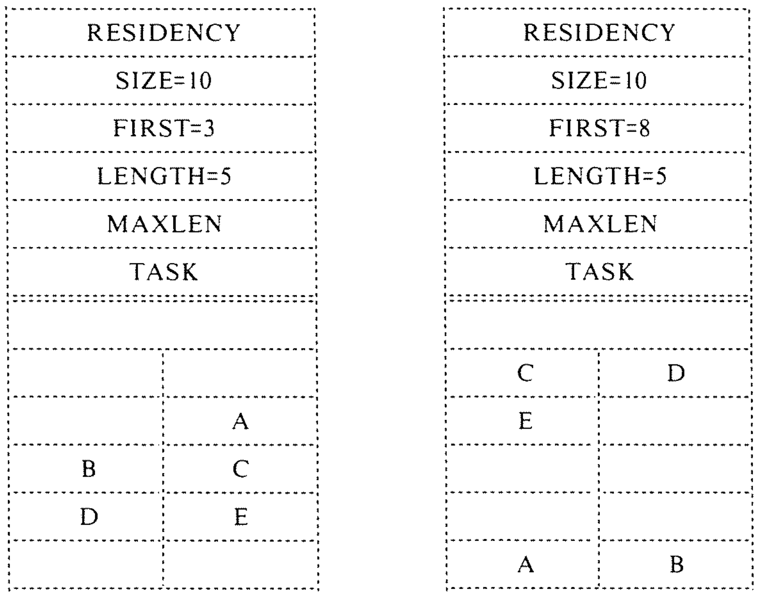

Figure 4. Examples of Queues Containing Five Bytes "ABCDE".

Note

The first byte of a word is shown here in the left byte for readability. Since PDP-11's are byte address machines, the first byte is actually in the right byte (bits 0-7), which has the lower address.

Conclusion

The queue technique implemented here has provided us with a flexible but consistent method of interprocess communication for this complex control system.

Ms. Leary is a lead computer systems analyst at the Laboratory for Laser Energetics, where she is responsible for the Omega Laser and target system computer configuration and hardware. Ms. Leary also supervises programming for the Laboratory Operations Group.

Mr. McClimans is president of Inti Electronics Corp., where he is consultant to the process control and laboratory automation industries. He is interested in real-time systems, microprocessor controllers, and device interfacing.

References

[FOR77] Forsley, L., Hardwick, K., and Berg, R., The URTH Tutorial, 1977.

[LEA82] Leary, R., and McClimans, D., The Omega Alignment System Software Maintenance Manual, 1982.

[LEA81] Leary, R., and Winkler, C., "Mapped Memory Management Techniques in Forth", Forth Dimensions, III, 4, 1981.

```
BLOCK # 350
( QUEUES - QSELECT )

: >QUEUES ;
: QSELECT  ( [QUEUE ADRS]---[  ] SAVES ADRS OF 'QUEUE' IN SVAR * )
   CURQ ! ;

- -> Original version      02/01/80  Rosemary C. Leary
     Task dispatching      02/23/81  Don McClimans
     Mapped memory         02/23/82  Rosemary Leary

BLOCK # 351
( QUEUES - QUEUE                          23FEB82 RCL )
: QUEUE  ( [RESIDENCY] [ADRS] [SIZE] [TASK] --- 'NAME' * )
   ( CREATES A QUEUE BUFFER IN MEMORY OR ON DISK * )
   ( RESIDENCY:  0=KERNEL MEMORY RESIDENT, 1=USER MEMORY RES. )
   (     2=DISK RESIDENT )
   ( WHEN EXECUTED, 'NAME' RETURNS THE ADRS OF THE RES WORD  * )
   ( Any put to this queue will dispatch TASK, unless TASK=0 )
   4 PICK CONSTANT                ( CREATE HDR, PARAM 1 = RESDNCY )
   HERE 2 - QSELECT               ( SELECT THIS QUEUE )
   3 PICK ,                       ( SAVE ADRS )
   SWAP 1+ 2/ 2* 2 MAX DUP ,      ( SIZE )
   0 , 0 , 0 , SWAP ,             ( FIRST, LENGTH, MAX, TASK )
- ->

BLOCK # 352
( QUEUES - QUEUE, CON'T                    23FEB82 RCL )
   ROT ROT OR 0=                  ( KERNAL MEMORY RESIDENT? )
   IF HERE CURQ @ 2 + !           ( STORE 'HERE' IN 'QADRS' )
     2/ DP+!                      ( CREATE BUFFER )
   ELSE DROP
   THEN  ; :  ;
- ->

BLOCK # 353
( QUEUES - QRES-ERROR )
: QRES-ERROR  ( [RESIDENCY]---[  ] )
   T" INVALID QUEUE RESIDENCY WORD" .
   T" QUEUE=" CURQ @ HPRINT
   T"  TASK=" SV HPRINT ;
- ->
```

```
BLOCK # 354
( QUEUES - QRES. QADRS. QSIZE. QFIRST. QLENGTH. QMAX. QTASK )
: QRES      ( [  ]---[RESIDENCY] RETURNS RESIDENCY WORD )
   CURQ @ @ ;
: QADRS     ( [  ]---[ADRS] RETURNS BLOCK# OR BUFFER STARTING ADRS )
   CURQ @ 2 + @ ;
: QSIZE     ( [  ]---[SIZE] RETURNS QUEUE MAXIMUM LENGTH IN BYTES )
   CURQ @ 4 + @ ;
: QFIRST    ( [  ]---[FIRST] RETURNS BYTE # OF FIRST ENTRY * )
   CURQ @ 6 + @ ;           ( RELATIVE TO WORD 4 OF BUFFER * )
: QLENGTH   ( [  ]---[LENGTH] RETURNS QUEUE LENGTH IN BYTES * )
   CURQ @ 8 + @ ;
: QMAX      ( [  ]---[MAX] RETURNS MAX LENGTH QUEUE HAS REACHED * )
   CURQ @ 10 + @ ;          ( SINCE IT WAS CREATED * )
: QTASK     ( [  ]---[ADRS] RETURNS SV ADRS OF TASK )
   CURQ @ 12 + @ ;
- ->

BLOCK # 355
( QUEUES - QB@ )
: QB@  ( [BYTE#]---[DATA] GETS BYTE FROM QUEUE BUFFER * )
   QADRS CURQ @ @ DUP 0=            ( ADRS & RESIDENCY )
   IF DROP + C@                     ( KERNEL MEMORY RESIDENT )
   ELSE DUP 1 =                     ( USER MEMORY RESIDENT? )
     IF DROP + DUP 1 BIC            (      CHECK EVEN OR ODD BYTE )
       U@ SWAP 1 AND IF 8 ->L THEN          ( FETCH WORD. SHIFT )
       3770 AND                     (      ISOLATE BYTE )
     ELSE DUP 2 =                   ( DISK RESIDENT? )
       IF DROP SWAP 1024 /MOD       (      FIND BLOCK OFFSET )
         ROT + BLOCK + C@           (      FETCH WORD )
       ELSE QRES-ERROR              ( BAD RESIDENCY WORD )
       THEN
     THEN
   THEN ;
- ->

BLOCK # 356
( QUEUES - QB! )
: QB!  ( [DATA][BYTE#]---[  ] STORES BYTE IN QUEUE BUFFER )
   QADRS CURQ @ @ DUP 0=            ( ADRS & RESIDENCY )
   IF DROP + C!                     ( KERNEL MEMORY RESIDENT )
   ELSE DUP 1 =                     ( USER MEMORY RESIDENT? )
     IF DROP + SWAP OVER 1 BIC      ( FORCE EVEN ADDRESS )
       U@ 3 PICK 1 AND              ( ODD OR EVEN BYTE? )
       IF 3770 AND SWAP 8 <-L OR    ( ODD, SHIFT UP )
       ELSE 1774000 AND SWAP 3770 AND OR  ( EVEN, MASK )
       THEN SWAP 1 BIC U!           ( STORE BYTE )
     ELSE DUP 2 =                   ( DISK RESIDENT? )
       IF DROP SWAP 1024 /MOD       ( GET BLOCK OFFSET )
         ROT + BLOCK + C! UPDATE    ( STORE BYTE, SET FLAG )
       ELSE QRES-ERROR THEN         ( BAD RESIDENCY WORD )
     THEN
   THEN                             - ->
```

```
BLOCK # 357
( QUEUES - QB! , CON'T )
    CURQ @ 12 + @ DUP 0 <>           ( IS TASK ASSIGNED? )
    IF DISPATCH                      ( YES, DISPATCH TASK )
    ELSE DROP                        ( NO, IGNORE )
    THEN ;
-->

BLOCK # 358
( QUEUES - QFULL?, QEMPTY?, QSPACE )
: QFULL?    ( [  ]---[T/F] RETURNS TRUE IF QUEUE IS FULL * )
    QSIZE QLENGTH - 0= ;
: QEMPTY?   ( [  ]---[T/F] RETURNS TRUE IF QUEUE IS EMPTY * )
    QLENGTH 0= ;
: QSPACE    ( [  ]---[#BYTES] RETURNS SPACE LEFT IN QUEUE * )
    QSIZE QLENGTH - ;

-->

BLOCK # 359
( QUEUES - PUT-UPDATE, GET-UPDATE, BTOQ, BFRQ )
: PUT-UPDATE    ( [COUNT]---[  ] UPDATES QUEUE HEADER * )
    CURQ @ SWAP OVER 8 + DUP         ( ADD COUNT TO LENGTH )
      @ ROT + DUP ROT !              ( SAVE NEW LENGTH )
    SWAP 10 + DUP @ ROT MAX          ( UPDATE MAX )
      SWAP ! ;                       ( SAVE NEW MAX )
: GET-UPDATE    ( [COUNT]---[  ] UPDATES QUEUE HEADER * )
    DUP CURQ @ SWAP OVER 6 +         ( ADD COUNT TO FIRST )
      DUP @ ROT + QSIZE MOD SWAP !     ( MOD QSIZE )
    8 + DUP @ ROT - SWAP ! ;         ( SUBTRACT COUNT FROM LENGTH )
: BTOQ          ( [DATA BYTE]---[  ] BYTE TO QUEUE * )
    QFIRST QLENGTH + QSIZE MOD QB! ;   ( GET POINTER, STORE )
: BFRQ          ( [  ]---[DATA BYTE] BYTE FROM QUEUE * )
    QFIRST  QB@ ;                    ( GET POINTER, READ )
-->

BLOCK # 360
( QUEUES - WTOQ, WFRQ )
: WTOQ          ( [DATA WORD]---[  ] WORD TO QUEUE * )
    DUP 8 ->L SWAP 255 &             ( SEPARATE BYTES )
    QFIRST QLENGTH + SWAP OVER       ( GET BYTE POINTER )
    QSIZE MOD QB!                    ( STORE FIRST BYTE )
    1+ QSIZE MOD QB! ;               ( STORE SECOND BYTE )
: WFRQ          ( [  ]---[DATA WORD] WORD FROM QUEUE * )
    QFIRST DUP QB@ SWAP              ( GET FIRST BYTE )
    1+ QSIZE MOD QB@ 8 <-L OR ;      ( SECOND BYTE, MERGE )
-->
```

```
BLOCK # 361
( QUEUES - TOQ, FRQ )
: TOQ            ( [ADRS] [COUNT]---[COUNT] STRING TO QUEUE * )
   0 MAX                             ( IGNORE NEGATIVE COUNTS )
   DUP ROT QFIRST QLENGTH +          ( GET BYTE POINTER )
   ROT 0 DO                          ( LOOP FOR COUNT )
     OVER I + C@                     ( GET DATA BYTE )
     OVER I + QSIZE MOD QB!          ( GET ADRS, STORE )
   LOOP 2DROP ;                      ( LEAVE COUNT ON STACK )
: FRQ            ( [ADRS] [COUNT]---[COUNT] STRING FROM QUEUE * )
   0 MAX                             ( IGNORE NEGATIVE COUNTS )
   DUP ROT QFIRST SWAP               ( GET BYTE POINTER )
   ROT 0 DO                          ( LOOP FOR COUNT )
     OVER I + QSIZE MOD QB@          ( GET BYTE FROM QUEUE )
     OVER I + C!                     ( STORE )
   LOOP 2DROP ;                      ( LEAVE COUNT )
-->

BLOCK # 362
( QUEUES - QPUT-FULL?, QGET-EMPTY? )
: QPUT-FULL?     ( [ADRS] [CNT]---[T/F] STRING TO QUEUE, NO WAIT * )
   ( RETURNS TRUE IF NOT ENOUGH SPACE IN QUEUE * )
   DUP QSIZE QLENGTH - >             ( ENOUGH SPACE? )
   IF 2DROP 1                        ( NO, LEAVE TRUE )
   ELSE TOQ PUT-UPDATE 0             ( YES, TRANSFER, LEAVE FALSE )
   THEN ;
: QGET-EMPTY?    ( [ADRS] [CNT]---[T/F] STRING FROM QUEUE, NO WAIT * )
   ( RETURNS TRUE IF NOT ENOUGH BYTES IN QUEUE * )
   DUP QLENGTH >                     ( ENOUGH ENTRIES? )
   IF 2DROP 1                        ( NO, LEAVE TRUE )
   ELSE FRQ GET-UPDATE 0             ( YES, TRANSFER, LEAVE FALSE )
   THEN ;
-->

BLOCK # 363
( QUEUES - QPUT, QGET, QGET-NR )
: QPUT           ( [ADRS] [CNT]---[ ] STRING TO QUEUE - WAIT IF FULL * )
   WHILE-TRUE 2DUP QPUT-FULL?        ( STUFF IF ROOM ENOUGH )
   PERFORM PAUSE                     ( OTHERWISE PAUSE )
   ENDWHILE 2DROP ;                  ( CLEAN STACK WHEN DONE )
: QGET           ( [ADRS] [CNT]---[ ] STRING FROM QUEUE - WAIT IF EMPTY * )
   WHILE-TRUE 2DUP QGET-EMPTY?       ( FETCH IF ENOUGH BYTES )
   PERFORM PAUSE                     ( OTHERWISE PAUSE )
   ENDWHILE 2DROP ;                  ( CLEAN STACK WHEN DONE )
: QGET-NR        ( [ADRS] [CNT]---[ ] READ STRING BUT DO NOT REMOVE * )
   BEGIN 2DUP DUP QLENGTH >          ( ENOUGH BYTES? )
     IF PAUSE 2DROP 0                ( NO, LOOP AGAIN )
     ELSE FRQ DROP 1                 ( YES, GET BYTES )
     THEN
   END 2DROP ;
-->
```

```
BLOCK # 364
( QUEUES - QWPUT-FULL?, QWGET-EMPTY? )
: QWPUT-FULL?   ( [DATA WORD]---[T/F] WORD TO QUEUE, NO WAIT * )
   QSPACE 2 <                        ( ENOUGH SPACE? )
   IF DROP 1                         ( NO, LEAVE TRUE )
   ELSE WTOQ 2 PUT-UPDATE 0          ( YES, TRANSFER, LEAVE FALSE )
   THEN ;
: QWGET-EMPTY? ( [  ]---[DATA WORD] [T/F] WORD FROM QUEUE, N/W * )
   QLENGTH 2 <                       ( ENOUGH BYTES? )
   IF 1                              ( NO, LEAVE TRUE )
   ELSE WFRQ 2 GET-UPDATE 0          ( YES, TRANSFER, LEAVE FALSE )
   THEN ;
-->

BLOCK # 365
( QUEUES - QWPUT, QWGET, QWGET-NR )
: QWPUT          ( [DATA WORD]---[  ] WORD TO QUEUE, WAIT TIL READY * )
   WHILE-TRUE DUP QWPUT-FULL?        ( STUFF IF SPACE ENOUGH )
   PERFORM PAUSE                     ( OTHERWISE PAUSE )
   ENDWHILE DROP ;                   ( CLEAN STACK WHEN DONE )
: QWGET          ( [  ]---[DATA WORD] WORD FROM QUEUE, WAIT TIL READY * )
   WHILE-TRUE QWGET-EMPTY?           ( FETCH IF WORD AVAILABLE )
   PERFORM PAUSE                     ( OTHERWISE PAUSE )
   ENDWHILE ;
: QWGET-NR       ( [  ]---[DATA WORD] WORD FROM QUEUE, DO NOT REMOVE * )
   BEGIN QLENGTH 2 <                 ( ENOUGH BYTES? )
     IF PAUSE 0                      ( NO, LOOP AGAIN )
     ELSE WFRQ 1                     ( YES, TRANSFER )
     THEN
   END ;
-->

BLOCK # 366
( QUEUES - QBPUT-FULL?, QBGET-EMPTY? )
: QBPUT-FULL?   ( [DATA BYTE]---[T/F] BYTE TO QUEUE, NO WAIT * )
   QFULL?                            ( ENOUGH SPACE? )
   IF DROP 1                         ( NO, LEAVE TRUE )
   ELSE BTOQ 1 PUT-UPDATE 0          ( YES, TRANSFER, LEAVE FALSE )
   THEN ;
: QBGET-EMPTY? ( [  ]---[DATA BYTE] [T/F] BYTE FROM QUEUE, N/W * )
   QEMPTY?                           ( ANY BYTES? )
   IF 1                              ( NO, LEAVE TRUE )
   ELSE BFRQ 1 GET-UPDATE 0          ( YES, TRANSFER, LEAVE FALSE )
   THEN ;
-->
```

BLOCK # 367

```
( QUEUES - QBPUT, QBGET, QBGET-NR )
: QBPUT           ( [DATA BYTE]---[  ] BYTE TO QUEUE, WAIT TILL READY * )
    WHILE-TRUE DUP QBPUT- FULL?       ( STUFF IF ROOM ENOUGH )
    PERFORM PAUSE                     ( OTHERWISE PAUSE )
    ENDWHILE DROP ;
: QBGET           ( [  ]--[DATA BYTE] BYTE FROM QUEUE, WAIT TILL READY * )
    WHILE-TRUE QBGET-EMPTY?           ( FETCH IF BYTE AVAILABLE )
    PERFORM PAUSE                     ( OTHERWISE PAUSE )
    ENDWHILE ;
: QBGET-NR        ( [  ]--[DATA BYTE] BYTE FROM QUEUE, DO NOT REMOVE * )
    BEGIN QEMPTY?                     ( ENOUGH DATA? )
      IF PAUSE 0                      ( NO, LOOP AGAIN )
      ELSE BFRQ 1                     ( YES, TRANSFER )
      THEN
    END ;
-->
```

BLOCK # 368

```
( QUEUES - QSPUT-FULL?, QSGET-EMPTY? )
: QSPUT-FULL?   ( [ADRS]---[T/F] STRING TO QUEUE, NO WAIT * )
    ( COUNT IN FIRST BYTE, RETURNS TRUE IF NOT ENOUGH SPACE * )
    DUP C@ 1+                         ( GET BYTE COUNT, ADD 1 )
    QPUT-FULL? ;                      ( ATTEMPT TO STUFF STRING )
: QSGET-EMPTY? ( [ADRS]---[T/F] STRING FROM QUEUE, NO WAIT * )
    ( COUNT IF FIRST BYTE, RETURNS TRUE IF NOT ENOUGH BYTES * )
    QBGET-NR 1+ QGET-EMPTY? ;         ( GET COUNT, FETCH STRING )
-->
```

BLOCK # 369

```
( QUEUES - QSPUT, QSGET, QSGET-NR )
: QSPUT           ( [ADRS]---[  ] STRING TO QUEUE - WAIT IF FULL * )
    WHILE-TRUE DUP QSPUT-FULL?        ( STUFF IF ROOM ENOUGH )
    PERFORM PAUSE                     ( OTHERWISE PAUSE )
    ENDWHILE DROP ;                   ( CLEAN STACK WHEN DONE )
: QSGET           ( [ADRS]---[  ] STRING FROM QUEUE - WAIT IF EMPTY * )
    WHILE-TRUE DUP QSGET-EMPTY?       ( FETCH IF BYTES ENOUGH )
    PERFORM PAUSE                     ( OTHERWISE PAUSE )
    ENDWHILE DROP ;                   ( CLEAN STACK WHEN DONE )
: QSGET-NR        ( [ADRS]---[  ] READ STRING BUT DO NOT REMOVE * )
    DUP C@ SWAP 1- SWAP               ( RETRIEVE COUNT, ADJUST ADRS )
    QGET-NR ;                         ( GET DATA )
-->
```

```
BLOCK # 370
( QUEUES - XLITST )
   ( DUPLICATE OF THIS IN COLOR CRT DRIVER )
CODE XLITST     ( JUMP INTERPRETER POINTER OVER STRING * )
   R1    RP )   MOV,          ( SAVED INTERPRETER POINTER TO R1 )
   R0 / R1 )    MOV,          ( BYTE COUNT OF STRING TO R0 )
   RP ) R0      ADD,          ( ADD BYTE COUNT TO INTERPRETER PTR )
   RP ) 2 #     ADD,          ( ADD TWO FOR POSSIBLE ODD NUMBER )
   RP ) 1 #     BIC,          ( MAKE EVEN ADDRESS )
   R1           INC,          ( BUMP STRING ADDRESS OVER CHAR COUNT )
   S -) R0      MOV,          ( PUSH CHAR COUNT TO STACK )
   PUSH         J,            ( PUSH STRING ADDRESS TO STACK )
-->

BLOCK # 371
( QUEUES - Q" )
: QPUT-L          ( [CNT][ADRS]---[  ] QPUT WITH REVERSE ARGUMENTS * )
   SWAP QPUT ;
: QLIT            ( LITERAL CHARACTERS * )
   XLITST QPUT-L ;
: Q"
   ' QLIT ' QPUT-L 34 CLIT ;    IMP Q"
: QSPUT-L         ( [CNT][ADDRS] --- [  ]. QSPUTS STRING FOR QS" * )
     SWAP DROP  1- QSPUT ;
: QSLIT           ( LITERAL CHARACTERS. POINTS TO QSPUT-L * )
     XLITST QSPUT-L ;
: QS"             ( LITERAL STRING TO QUEUE - WITH LEADING BYTE CNT * )
   ' QSLIT ' QSPUT-L 34 CLIT ;    IMP QS"
;S

BLOCK # 372
( DEBUG - ECNT, OCNT, DPRINT, .LINK )
: ECNT            ( [TICK ADRS]---[RIGHT BYTE] GETS BYTE FROM HEADER * )
   8 - C@ 1775770 AND ;
: OCNT            ( [TICK ADRS]---[LEFT BYTE] GETS BYTE FROM HEADER * )
   7 - C@ ;
: HPRINT          ( [' ADRS]---[  ] PRINTS NAME * )
   DUP OCNT TCH                    ( CHAR 1 )
   DUP 2 + ECNT TCH                ( CHAR 2 )
   DUP 2 + OCNT TCH                ( CHAR 3 )
   ECNT 3 - DUP 0 >                ( FILL OUT MISSING CHARACTERS )
     IF 0 DO 1370 TCH LOOP
     ELSE DROP
     THEN ;
 -->
```

```
BLOCK # 373
( QUEUES - .QUEUE                                25FEB82 RCL )
: >QPRINT ;
: .QUEUE            ( PRINTS WHOLE QUEUE * )
    CR CURQ @ DUP HPRINT T"            "        ( PRINT HEADER )
    @ DUP 0 = IF T" KERNEL MEMORY RESIDENT" THEN
       DUP 1 = IF T" USER MEMORY RESIDENT" THEN
            2 = IF T" DISK RESIDENT" THEN CR
    T" SIZE= " QSIZE .                        T"  FIRST= " QFIRST .
    T"  LENGTH=" QLENGTH .                    T"  MAX= " QMAX .
    QTAST DUP 0<>                             ( PRINT TASK NAME )
    IF DUP MASTER =
      IF      T"    TASK=MASTER" DROP
      ELSE  T"    TASK=" HPRINT
      THEN
    ELSE DROP
    THEN                                      - ->

BLOCK # 374
( QUEUES - .QUEUE                               25FEB82 RCL )
    QLENGTH 0<>                               ( PRINT CONTENTS )
    IF QLENGTH 0 DO                           ( EACH BYTE IN QUEUE )
        I 10 MOD 0 = IF CR THEN               ( 10 PER LINE )
        QFIRST I + QSIZE MOD QB@              ( GET BYTE )
        DUP 5 .V SPACE TCH                    ( PRINT DECIMAL & TYPE ASCII )
      LOOP
    THEN CR ;
- ->

BLOCK # 375
(Print and empty a queue )

: QPRINT          ( [   ] --- [   ]. PRINTS AND EMPTIES CURRENT QUEUE. )
     QLENGTH DUP 0> IF
           2 / 0 DO     QWGET . LOOP CR
     ELSE DROP THEN ;
;S
```