4 MACHINE RESOURCE MANAGEMENT (TRAP #1)

The QDOS procedures, within this classification, control the allocation of machine resources. QDOS supports multi-tasking (i.e., the pseudo-concurrent execution of multiple jobs) and the appropriate allocation of both CPU and RAM resources is, therefore, of prime importance.

4.1 QDOS multi-tasking

System calls to QDOS may either be treated as atomic or partially atomic. Most QDOS routines are atomic in nature. Atomic routines are executed with the 68000 processor in supervisor mode. In this mode no other job can take priority over use of the processor and, therefore, the routine will be executed from start to finish before being 'swapped-out'. Note that this is the general case only; the routine could be interrupted by an interrupt service procedure. Routines which are partially atomic will complete some sort of primary operation, but will then allow another job to swap-out the original calling process until a later moment in time. All the I/O calls are partially atomic unless specifically accessed as being fully atomic. Scheduler calls are partially atomic.

JOB STATUS

When a job is set up by QDOS procedures it can exist in a number of states. First, the job can be active. This means that the job has a priority, within the multi-tasking environment, and will obtain a share of the 68000 CPU resources in line with that priority. If the job has a low priority then it will be allocated a relatively small percentage of CPU resources.

Second, the job may be suspended, either for a limited time or for an indefinite period. Jobs are normally suspended to force them to wait for some I/O or another job.

Third, the job may be inactive. The job will still use up space within the memory, but it will never be allocated any CPU resources. A job at priority level 0 is identical to an inactive job. A major difference between an inactive job and a job that has been suspended indefinitely is that the latter cannot be removed by the simple version of the 'remove job' call (i.e., MT.RJOB (TRAP #1, D0=4)). The reason for the

difference between suspended/released jobs and inactive/active jobs is as follows. Suspending and releasing a job does not alter the flow of execution of code; it merely interrupts it. On the other hand, an inactivated job has completed its execution. When re-activated, the job will start again at the beginning.

SCHEDULING

The allocation of CPU resources to jobs, in line with job priorities, is known as scheduling. QDOS re-scheduling is related to the frame rate of the monitor (i.e., 50/60 Hz). Certain QDOS routines will cause re-scheduling to take place (e.g., MT.SUSJB (TRAP #1, D0=8)).

TIMEOUTS

A number of QDOS calls permit a timeout to be specified. The procedure to suspend a job (i.e., MT.SUSJB) is one example. The timeout period is a multiple of the monitor frame rate (i.e., 50/60 Hz). A timeout period of unity is therefore equivalent to 20 ms for a 50 Hz timebase, and 16.666 ms for a 60 Hz timebase.

With respect to QDOS procedures, a timeout value of -1 signifies that an indefinite period is required. No other negative value should be used. A maximum timeout period of 32767 times the unit timebase period is permitted. This gives a maximum period of 10 minutes, 55.3 seconds for a 50 Hz timebase, and 9 minutes, 6.1 seconds for a 60Hz timebase.

4.2 Use of 68000 registers

The TRAP #1 procedures are accessed with register D0 (byte) indicating which particular call is required. This register is also used to return an error status (long-word) to the calling process. If the error code returned is not zero then an error has occurred. Small negative error codes are used to indicate standard errors. These error codes are listed in Appendix C. If the trap call invoked some form of additional device driver, the error code returned can be a pointer to a specific error message. In order that the two types of error return code might never be confused, the pointer type error code is in fact a pointer to an address $8000 below that of the true error message. Potentially, all QDOS routines can return the error 'ERR.BP' (-15), signifying 'bad parameter'. The full descriptions of the TRAP #1 procedures state which additional errors can be returned. It would of course be wise to check for any errors after the trap call has been made.

In addition to the use of register D0, data registers D1 to D3 and address registers A0 to A3 are variably used to pass values to and from the QDOS procedures. When the appropriate registers have been set for any one call the appropriate routine is accessed by simply executing the TRAP #1 instruction. In cases where the data size qualifier (i.e., '.B',

'.W', or '.L') is not specified within the description, the default is long-word (i.e., '.L').

MT.INF $00 (0)

Get system information

Entry parameters:	none
Return parameters:	D1.L Current job ID D2.L QDOS version (in ASCII) A0 Pointer to system variables
Affected registers:	D1, D2, A0
Additional errors:	none

Description

MT.INF returns the specified system information. The version number of QDOS is returned as a 4-byte ASCII string in the form:

v.xx

where 'v' is the major revision, and 'xx' is the update code. The most significant word of D2 will contain the major revision code and the full stop. The update code will be in the least significant word of D2.

The system variable pointer returned in A0 is the value of the base pointer SV.BASE (normally $28000).

MT.CJOB $01 (1)

Create a job in transient program area

Entry parameters:	D1.L	Owner job ID
	D2.L	Code length (bytes)
	D3.L	Data length (bytes)
	A1	Start address
Return parameters:	D1.L	Job ID
	A0	Base of area allocated
Affected registers:	D1, A0	
Additional errors:	OM (-3)	out of memory
	NJ (-2)	not a valid job

Description

QDOS jobs are created in the transient program area. Each job has a fixed allocation of memory, which must allow for all data and working areas (including stack space). It is advisable to use at least an extra 64 bytes on top of any calculated stack space, to allow for QDOS utility stack requirements. On entry to MT.CJOB, registers D2 and D3 will specify the total memory requirement for the job. Stack space is included in the data length specification.

The specified start address will be zero if the start address is at the base of the job. Any other address specified must be absolute. The owner job ID will be zero if the job is to be independent. If the current job is to be this new job's owner then the owner job ID may be passed as -1 (i.e., a negative value).

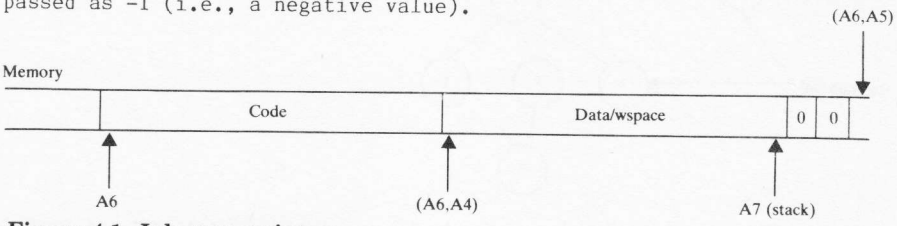


Figure 4.1 Job area pointers

Note that this procedure does not load or execute the actual job; it simply attempts to allocate space in the TPA and set up a job entry in the scheduler tables. The job program would normally be loaded by another job after this call had been successfully completed. Job programs must be written in position independent code. When a job is made active, register A6 will be pointing to the base of the job area, A6 indexed by A4 will be pointing to the base of the data area, and A6 indexed by A5 will be pointing to the top of the area. The stack pointer

register A7 will, in the simple case, be pointing to two words of zero placed on the stack by the MT.CJOB procedure (see Fig.4.1).

The two words of zero placed on the stack are a standard format information packet. In non-simple cases these zeros would be replaced by more detailed information packets consisting of, for example, a word of data representing the number of channels opened for the job, long-word channel IDs, and a job command string. The command string would also be in standard format, i.e., a word of data representing the length of the command string followed by the string itself.

Note that the SuperBASIC command EXEC performs the setting up of a TPA, and the loading and activation of a job program, all in one go! Unless there is the requirement for one job to set up and control its own sub-job, the use of EXEC would be the normal way to invoke a transient program (see Chapter 8).

MT.JINF $02 (2)

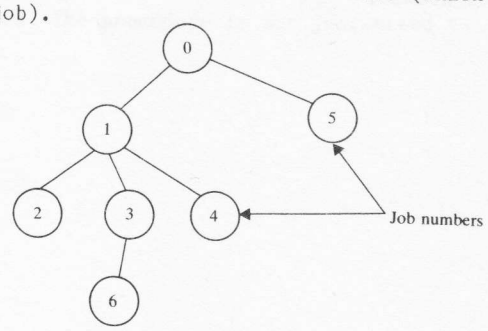
Get job information

Entry parameters:	D1.L	Job ID
	D2.L	Job ID at top of tree
Return parameters:	D1.L	Next job ID in tree
	D2.L	Owner job ID
	D3.L	Status/priority
	A0	Base address of job
Affected registers:	D1 - D3, A0, A1	
Additional errors:	NJ (-2) not a valid job	

Description

This procedure returns the status of a specified job. Jobs may be independent or they may be owned by other jobs (except for job zero which cannot be owned by any other job). The structure of job ownership can be viewed as a tree (see Fig.4.2). It is possible, using this procedure, to scan the status of an entire tree of jobs. To do so is a simple matter of setting D1 and D2 to the top of the tree, and then continuously calling MT.JINF until D1 is returned as zero (which signifies that there is no next job).

Figure 4.2 Job ownership tree



On exit, D2 will contain zero if the scanned job is independent. The most significant byte of D3 will be negative if the job is suspended, and the least significant byte of D3 will contain the priority of the job.

MT.RJOB $04 (4)

Remove inactive job from TPA

Entry parameters:	D1.L	Job ID
	D3.L	Error code
Return parameters:	none	
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	NJ (-2)	not a valid job
	NC (-1)	job is not inactive

Description

This procedure removes a specified job from the transient program area. Any sub-jobs owned by the job will also be removed. For the procedure to work, the job(s) must be inactive. On entry, D3 should contain the error code that is to be returned from the activation call which created the job (see MT.ACTIV; D0=$0A). If no such error exists then D3 will contain zero.

Note that job zero cannot be removed. The procedure is not guaranteed to be atomic.

MT.FRJOB $05 (5)

Force remove job(s) from TPA

Entry parameters:	D1.L	Job ID
	D3.L	Error code
Return parameters:	none	
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	NJ (-2) not a valid job	

Description

This procedure will inactivate and remove a complete job tree. On entry, D1 may be negative if the job to remove is the current job. Also, D3 should contain the error code that is to be returned from the activation call which created the job (see MT.ACTIV; D0=$0A). If no such error exists then D3 will contain zero.

If there is another job waiting for the completion of the job being removed, it will be released with D0 set to the error code that was initially returned from the activation call which created the job.

Note that job zero cannot be removed. The procedure is not guaranteed to be atomic.

MT.FREE $06 (6)

Length of largest space in TPA

Entry parameters: none

Return parameters: D1.L Length of space found

Affected registers: D1 - D3, A0 - A3

Additional errors: none

Description

This will return the length of the largest contiguous area of memory that could be subsequently allocated to a transient program area. Note that the value returned can only be used as a guide if there are numerous active jobs. The scheduling system may have allowed another job to grab some (or all) of the free memory in between the time this call was made and the time the current job attempts to use it.

MT.TRAPV $07 (7)

Set job trap vector pointer

Entry parameters:	D1.L	Job ID
	A1	Pointer to vector table
Return parameters:	D1.L	Job ID
	A0	Base address of job
Affected registers:	D1, A0, A1	
Additional errors:	none	

Description

The traps and exception vectors of the 68000 CPU that are not used by QDOS may be redirected through a table set up by a particular job. If a job does set up such a table, the table vectors will be used while the job is being executed. Additionally, any job set up by a job with its own vector table will automatically adopt the same table until it is redefined locally.

Vector tables set up by jobs (with the exception of adoption as mentioned above) are entirely local and do not affect tables within other jobs. If on entry, D1 is negative, the table will be set up for the job that called the procedure. The vector table pointed to by A1 on entry must contain long-word addresses for each trap and exception. The table order, together with the offset address for each vector (with respect to the base of the table), is as follows:

Offset	Vector/exception	Offset	Vector/exception
00	address error	28	TRAP #7
04	illegal instruction	2C	TRAP #8
08	zero divide	30	TRAP #9
0C	CHK	34	TRAP #10
10	TRAPV	38	TRAP #11
14	privilege violation	3C	TRAP #12
18	trace	40	TRAP #13
1C	interrupt level 7	44	TRAP #14
20	TRAP #5	48	TRAP #15
24	TRAP #6	< end of table >	

MT.SUSJB $08 (8)

Suspend a job

Entry parameters:	D1.L	Job ID
	D3.W	Timeout period
	A1	Flag byte address
Return parameters:	D1.L	Job ID
	A0	Base of job control area
Affected registers:	D1, A0	
Additional errors:	NJ (-2) not a valid job	

Description

This procedure will suspend a job, either indefinitely or for a given time. If on entry, D1 is negative, the current job will be suspended. Suspension for an indefinite time will occur if the word value of D3 is passed as -1. No other negative value should be used. Suspending an already suspended job will have the effect of resetting the suspension period.

The flag byte, which exists in the job control area, will be cleared when the job is later released. The flag is used to indicate to a job, which is suspending another job, that either the suspension has timed-out, or yet another job has released the suspended job. Use of the job control area is rather specialised and it is, therefore, not normally accessed by the applications programmer. As such register A1, on entry to the procedure, should be set to zero.

All jobs will be re-sheduled and therefore (as a result of accessing the scheduler) MT.SUSJB cannot be fully atomic.

MT.RELJB $09 (9)

Release a job and re-schedule

Entry parameters:	D1.L	Job ID
Return parameters:	D1.L	Job ID
	A0	Base of job control area
Affected registers:	D1, A0	
Additional errors:	NJ (-2) not a valid job	

Description

This call will release (i.e., un-suspend) a specified job and cause all jobs to be re-scheduled. The act of releasing a job does not imply that the job will become active. Job activity is also related to job priority, and a job that has a priority of zero will be inactive.

Because all jobs are re-sheduled MT.RELJB cannot be fully atomic.

MT.ACTIV $0A (10)

Activate a job

Entry parameters:	D1.L	Job ID
	D2.B	Job priority (0 to 127)
	D3.W	Timeout (0 or -1)
Return parameters:	D1.L	Job ID
	A0	Base of job control area
Affected registers:	D1, A0, A3	
Additional errors:	NJ (-2)	not a valid job
	NC (-1)	job already active
	<any job run-time error>	

Description

The specified job, in the transient program area, will be made active. Execution, on obtaining CPU resources, will begin at the address specified when the job was created (see MT.CJOB (TRAP #1, D0=1)). A priority of 127 is the highest priority possible.

Two timeout options are available. First, if a timeout of zero is given, execution of the current (calling) job will continue. The newly activated job will begin execution at some later moment in time, as and when the scheduler invokes it. In this case the two errors, ERR.NJ (-2) and ERR.NC (-1), are the only additional errors that will be returned.

Second, if a timeout of -1 is given, the current job will be suspended until the newly activated job has finished execution. In this case any error could be returned when the job completes (i.e., when the job removes itself, or is removed by some other job. See MT.RJOB; D0=4, and MT.FRJOB; D0=5).

MT.PRIOR $0B (11)

Change job priority

Entry parameters:	D1.L	Job ID
	D2.B	Job priority (0 to 127)
Return parameters:	D1.L	Job ID
	A0	Base of job control area
Affected registers:	D1, A0	
Additional errors:	NJ (-2)	not a valid job

Description

This call can be used to change the priority of a job. If, on entry, D1 is negative, the priority of the current job will be changed. A priority of zero will effect inactivation. This procedure invokes the scheduler and therefore a job will immediately inactivate itself if it sets its own priority to zero.

MT.ALRES $0E (14)

Allocate resident procedure area

Entry parameters:	D1.L	Number of bytes required
Return parameters:	A0	Base address of area
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	OM (-3) out of memory NC (-1) TPA not empty	

Description

This procedure is used to allocate memory to the resident procedure area. It should only be used when the TPA is completely empty (i.e., when no transient programs exist).

Note that the SuperBASIC function RESPR is normally used to perform this task from a 'BOOT' device or file (see Chapter 8).

MT.RERES $0F (15)

Release resident procedure area

Entry parameters: none

Return parameters: none

Affected registers: D1 - D3, A0 - A3

Additional errors: NC (-1) TPA not empty

Description

This procedure will release the resident procedure area. A paradox is evident here. The call cannot be made if the TPA is not empty, but a program must exist to make the call. It is no good having the call as part of a program in the resident procedure area because it will annihilate itself in the process!

There is a way of circumventing this apparent paradox but it is not to be encouraged. In practice, therefore, this call will never be used and the resident procedure area will only be released or reset by re-booting the entire system.

MT.DMODE $10 (16)

Set/read display mode

Entry parameters:	D1.B	Set/read display mode flag
	D2.B	Set/read display type flag
Return parameters:	D1.B	Display mode
	D2.B	Display type
Affected registers:	D1, D2, A4	
Additional errors:	none	

Description

This procedure is used for one of two purposes. First, if D1 and D2 are set to -1 on entry, the display mode (i.e., four or eight colour) and display type (i.e., TV or monitor) will be returned in the respective registers. The return values are as follows:

Display mode:	0	- 4 colour
	8	- 8 colour
Display type:	0	- monitor
	1	- TV (625 line)

Second, if one each of the above display mode and display type codes are placed into the respective registers on entry to the call, the display will be set accordingly. This form of the procedure should only be invoked when there are no other jobs attempting to access the display. All windows are cleared and the character sizes may change. In some cases (e.g., as shown in the example 'clock' program in Chapter 9) programs may be written such that the display mode may be changed by a job, without serious side-effects.

MT.IPCOM $11 (17)

Intelligent peripheral controller (IPC) command (keyboard row scan, sound)

Entry parameters: A3 Pointer to command

Return parameters: D1.B Return parameter

Affected registers: D1, D5, D7

Additional errors: none

Description

Extreme care must be taken when using this procedure. IPC communication is entirely unprotected and a total loss of machine operations may occur if an error exists in the call. Additionally, most of the IPC commands are for sole use by QDOS, and any attempt to use such commands is likely to effect a loss of data or something equally belligerent.

Three commands to the IPC are usable. Each command is a string of bytes consisting of a command byte, a parameter block, and a reply-length byte. The parameter block consists of a parameter byte count (in one byte), a long-word holding up to 16 2-bit codes, and the actual parameter bytes. The 2-bit codes are used to determine how many bits of each parameter byte should be sent to the IPC, as follows:

```
00    - send least significant four bits
01    - send nothing
10    - send all eight bits
```

Bits 1,0 of the long-word refer to parameter byte 0. Bits 3,2 refer to parameter byte 1, and so on. The final reply-length byte of the command is encoded in a similar fashion using bits 1,0 (i.e., a byte value of $02 will signify that the return value in register D1 should be eight bits long).

The three commands available are:

1. Keyboard row scan
   The command byte is $09. There is one parameter of four bits specifying the row to scan. The reply is eight bits long and it has one bit set for each column position that has a key pressed. The relationship between rows, columns, and actual keys can be found in the QL User Manual in the section on the SuperBASIC command KEYROW.

2. Create sound
   The command byte is $0A. There are eight parameters:

pitch-1	8 bits
pitch-2	8 bits
step interval	16 bits
duration	16 bits
pitch step	4 bits
wrap	4 bits
randomness	4 bits
fuzz	4 bits

   There is no reply.

3. Kill sound
   The command byte is $0B. There are no parameters and no reply is given.

MT.BAUD $12 (18)

Set baud rate

Entry parameters: D1.W Baud rate

Return parameters: none

Affected registers: D1

Additional errors: none

Description

This procedure is used to set the baud rate for the two serial interfaces, SER1 and SER2. The same baud rate must be applied to both interfaces. The rate is passed to the procedure as a pure binary value. For example, to set a rate of 9600 baud the following could be used:

```
:
move.w    #9600,d1      ;9600 baud
move.b    #$12,d0       ;Set baud rate proc.
trap      #1
:
```

MT.RCLCK $13 (19)

Read real-time clock

Entry parameters: none

Return parameters: D1.L Time in seconds

Affected registers: D1, D2, A0

Additional errors: none

Description

This procedure will return the time in seconds and may be used in one of two ways. First, it can be used in conjunction with the clock set and adjust traps, and date, day, and time utilities to obtain a true calendar clock for as long as the machine is turned on. In this case, time is assumed to have started at 00.00 hours on 1 January 1961.

Second, the value returned can be used simply to measure elapsed time between two successive calls.

MT.SCLCK $14 (20)

Set real-time clock

Entry parameters: D1.L Time in seconds

Return parameters: D1.L Time in seconds

Affected registers: D1 - D3, A0

Additional errors: none

Description

This procedure is used to set the real-time clock so that utilities can be used to display a true calendar clock. Time is assumed to have started at 00.00 hours on 1 January 1961.

MT.ACLCK $15 (21)

Adjust real-time clock

Entry parameters:	D1.L	Adjustment in seconds
Return parameters:	D1.L	Time in seconds
Affected registers:	D1 - D3, A0	
Additional errors:	none	

Description

This call can be used to adjust the real-time clock. A negative adjustment can be made. Because it takes a significant time to set the clock, no adjustment will be made if D1 is zero on entry to the call.

MT.ALCHP $18 (24)

Allocate common heap area

Entry parameters:	D1.L	Number of bytes required
	D2.L	Owner job ID
Return parameters:	D1.L	Number of bytes allocated
	A0	Base address of area
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	OM (-3)	out of memory
	NJ (-2)	not a valid job

Description

Space may be grabbed from the common heap area by jobs. The space may be allocated by one job on behalf of another job, and all heap space allocated using this procedure will be released when the owner job (rather than the 'allocater' job) is removed. The allocated space is cleared and all of it is available to the job.

MT.RECHP $19 (25)

Release common heap area

Entry parameters: A0 Base of area to be freed

Return parameters: none

Affected registers: D1 - D3, A0 - A3

Additional errors: none

Description

This procedure will release the specified space from the common heap area. It is the programmer's responsibility to ensure that the area is completely finished with before this procedure is called.