

MCR10186

Spurious “Unexpected IOM status” Messages From disk_control

Revision 1.0

Eric Swenson

October 10, 2026

1 Problem Description

[Ticket #249](#) described an issue where messages of the form

```
disk_control: Unexpected IOM status 400000000000000000000000 for dska (channel A11).  
disk_control: Unexpected IOM status 40000000000770000000000010 for dska (channel A11).
```

would appear frequently on GHM – a multi-CPU system.

They occur at random, often several at once (on GHM, in bursts at the same minute of every hour, when the system does a burst of disk I/O), never on a single-CPU system, and far more often when the simulator runs on a shared virtual server than on dedicated hardware. The simulator’s real-time mode reduces them. An email discussion in September 2026 (C. Anthony, J. Johnson) suspected thread contention and either a race in Multics or a bad residue from the simulator.

1.1 What the message means

`disk_control`’s `check_stat` logs the message, and otherwise ignores the status, when it is given a terminate status for a channel whose `chantab.active` flag is off. The status itself is not the problem: the first word, 400000000000 or 4000000000077, is a normal successful completion (terminate bit on, major status 0; 77 is the record-count residue). The remaining twelve digits are not part of the status word: `disk_control` prints 72 bits starting at the 36-bit status word, so they come from the next word of the saved status entry. Neither part selects a code path; only `chantab.active` does.

1.2 Cause

Two pieces of code read a disk channel’s status word:

- `io_manager$get_status`, used by `disk_control`’s polling (`run`) and its interrupt handler (`disk_inter`), *consumes* the status: it loads word 1 with `ldac`, which clears it, so a status is seen only once.

- The fast-path interrupt handler `dctl$disk_inter`, which is the handler actually registered for disk channels and processes every normal completion, reads word 1 with `lda` and never clears it. Its own comment at the zero-status test, “someone else processed status (`disk_control$run`)”, shows it relies on the status being cleared by whoever processed it, but it does not clear it itself.

A connect overwrites the status word, so the stale status survives only while the channel is idle. The message results when another terminate interrupt for the channel arrives in that interval. That happens in this sequence, which needs two CPUs:

1. I/O 1 completes. Its terminate interrupt is taken by CPU B, which enters `dctl$disk_inter` and waits for the disk lock, held by CPU A.
2. CPU A, holding the lock in `disk_control$run` (polling, as page control does while masked), consumes I/O 1’s status with `get_status`, posts it, and starts I/O 2 on the same channel. I/O 2 completes, storing its status and requesting a second terminate interrupt. CPU A releases the lock.
3. CPU B gets the lock, finds I/O 2’s status, sees the channel active, and processes I/O 2’s completion correctly; the channel goes idle. The status word still holds I/O 2’s status.
4. I/O 2’s own terminate interrupt is now taken. `dctl$disk_inter` finds a status (I/O 2’s, again), sees the channel is not active, and passes the interrupt to `disk_control$disk_inter`, which consumes the status and logs “Unexpected IOM status”.

The comment in `disk_control`’s `run` already anticipates this (“This race hazard . . . may also produce some situations of interrupt while not active”); the fast path’s failure to consume the status is what turns it into a logged error.

Step 2 is what makes the simulator, and especially a simulator on a shared virtual server, so susceptible. The simulator performs a disk I/O synchronously, inside the CPU’s connect instruction, so I/O 2 completes while CPU B is still waiting for the lock; on real hardware I/O 2 would take milliseconds and its status would rarely be there yet. And when the host deschedules the simulator thread for CPU A while it holds the disk lock, CPU B waits longer, widening the window. With one CPU there is no CPU B.

1.3 Effects

The messages are harmless apart from filling the log: each I/O is posted exactly once, and the stale status is ignored. It cannot be mistaken for the completion of a later I/O, because the connect for that I/O overwrites the status word first.

1.4 Measurements

`disk_meters -channels` on GHM (two CPUs or more), over 239:58 hours before the fix:

Chan	Connects	Term by RUN	Interrupt w/o term	get_io w/o term	Term w/o Active
A11	4683797	2241	594	1568	43
B11	11795	24	1	1	0

“Term w/o Active” counts the messages: the `syserr` log for the same ten days contains exactly 43, about four a day, most of them at the same minute past the hour (when GHM does an hourly burst

of disk I/O), sometimes two or three within seconds. Polling picked up a status 2241 times (“Term by RUN”), which is the precondition for the sequence below; in most of those cases the interrupt later found no status and was only counted (594 + 1568), and in 43 it produced the message. A double interrupt from the simulator would not need polling at all. GHM-Test, with two CPUs on a less loaded host, logged two messages in seven days.

2 Proposed Fix

In `dctl$disk_inter`, once all the checks for the simple case have passed and the completion is committed (“AT THIS POINT WE MAY BEGIN TO MODIFY DISK_SEG”), clear status word 1, just as `get_status` would:

```
"AT THIS POINT WE MAY BEGIN TO MODIFY DISK_SEG TO REFLECT AN I/O COMPLETION.
```

```
"    Consume the status, as io_manager$get_status (used by disk_control's
"    run and disk_inter) does. It was left in place, so when run had
"    already taken an earlier status for this channel and started this I/O,
"    this I/O's own terminate interrupt found the status again after the
"    channel had gone idle, and disk_control logged "Unexpected IOM status"
"    (ticket 249). With the status cleared that interrupt is counted as
"    chantab.no_status_terminate instead. We hold the lock, so no new
"    status can be stored until we start another I/O.
```

```
    stz    ap|chantab.statusp,*  status consumed
```

```
    lca    chantab.active+chantab.inop+1,d1
```

The status is cleared only on the fast path’s own completion path. When the fast path hands an interrupt to `disk_control$disk_inter` (errors, an inactive channel, tests, and so on), the status is left for `get_status` as before. The disk lock is held from the read of the status to the store, and a new status can be stored only after a connect, which also requires the lock.

In step 4 above, `dctl$disk_inter` then finds a zero status and takes its existing `NO_STATUS_TERM` exit, which counts the interrupt in `chantab.no_status_terminate` without a message.

2.1 Mitigation without the fix

Running the simulator in real-time mode, with enough host CPUs for all simulated CPUs, makes the window much smaller, as observed in the email discussion. It does not remove it.

2.2 Not changed

- The message itself is kept: with this change it again indicates a genuine anomaly.
- The record-count residue of 77 comes from the simulator decrementing an IDCW record count of 0 (`dps8_disk.c`); it has no effect on Multics and is not related to the message.

3 Documentation

No documentation changes.

4 Test

1. **Before.** The GHM figures above are the baseline. On a multi-CPU system that shows the messages, record `disk_meters -channels` and count the messages over a day. The “Term w/o Active” column (`terminate_not_active`) should match the number of messages, and “Term by RUN” (`status_from_run`) should be nonzero.
2. **After.** With the new `bound_page_control`, run for the same period with similar load. There are no “Unexpected IOM status” messages, “Term w/o Active” does not increase, and “Interrupt w/o term” (`no_status_terminate`) increases by about the number of messages previously seen.
3. **Stress.** Load the host (for example with `stress-ng`, as in the email discussion) while Multics does heavy disk I/O (for example several concurrent `copy` operations of large segments), and check as in step 2.
4. **Function.** Normal operation under a full day of user load, and a shutdown and reboot, show no disk errors or hangs.
5. **Single CPU.** A single-CPU boot behaves as before.

5 Files Changed

<code>bound_page_control.s.archive</code>	<code>dctl.alm</code> : consume the status in the fast-path interrupt handler
<code>bound_page_control.archive</code>	<code>dctl</code> (reassembled)
<code>>sl1</code>	<code>bound_page_control</code> (rebound). It is part of the supervisor, so it is installed with a new system tape.

6 Version History

2026-10-10	1.0	Initial version of the MCR.
------------	-----	-----------------------------