Subject: MCR-10053, Fix exercise_disk and rdisk_ Event Wait Channel Processing

Author: Eric Swenson

Date: April 21, 2019

Problem Description

Multics Change Ticket: http://multics-trac.swenson.org/ticket/163 describes an issue with the execise_disk command and the rdisk_ i/o module. The issue is that the code that does i/o using ioi_$connect does not always call ipc_$block. Not calling ipc_$block results in an event message remaining in the ECT for the event channel, which may or may not get processed for the next ioi_$connect call. Unprocessed event messages may accumulate, overflowing the process directory in which the area that holds the ECT lives.

Each time ioi_$connect is called, an interrupt from the IOM will be sent to Multics indicating the status of the i/o. The status can indicate an error or success, but this interrupt is converted into a wakeup event message for the event wait channel associated with the IOI device. This event wait channel is established in the initial rcp_$attach (or rcp_priv_$attach) call made to attach the device for i/o. The expectation is that each event message will be read from the ECT during a call to ipc_$block and then freed from the area in which the ECT lives.

The ECT is the event channel table, and it lives in an per-ring area in the process directory. A field in each ring's stack header points to the ECT area. When the first event channel is created in a ring, ipc_real_ calls an internal procedure, find_ectp, which checks the value of the ect_ptr in the stack header for the ring. If it is null, it calls ipc_util_$create_ect to create the ECT. This latter entry allocates an area for the ECT (ect_area) in the system free area. It then calls define_area_ to convert the original area into an extensible area. An extensible area can grow beyond a segment. The area manager knows how to add new segments to the area (which is called a multisegment or extensible area) when there is not enough room in the area for a new allocation. An internal procedure of ipc_util_$create_ect allocates an ect_header in the extensible area, initializes the ECT, and sets a pointer to this ect_header in the stack header (for the current ring).

Ring-0 IPC allocates ITT (Interprocess Transmission Table) messages in this area when the user-ring IPC calls hcs_$read_events. These ITT messages are threaded into either an event call or event wait message thread in the ECT by user-ring IPC. When ipc_$block is called in the user ring, the code retrieves the first event message associated with the event channel passed in a structure parameter to ipc_$block. If no call to ipc_$block is made (nor the event channel drained with ipc_$drain_chn) then the event message stays in the ECT.

The completion status of IOI i/o is reflected in a status queue in the IOI workspace and updated automatic by the interrupt termination support in ioi_masked$interrupt. This status queue is directly accessible to user-ring code, and can therefore be checked after a call to ioi_$connect. It is possible that this status can get updated after a call to ioi_$connect and before a call to ipc_$block to (possibly) wait for the termination status. The code in exercise_disk and rdisk_ has an optimization — it checks the status for completion and errors and if already updated, skips the call to ipc_$block. Unfortunately, this has the side effect of leaving the IOI event in the ECT where it will get picked up the next time ipc_$block is called.

If this happens, the event message returned from ipc_$block will NOT correspond to the immediately preceding call to ioi_$connect, but to an earlier call. And the other side effect is that the event messages may accumulate in the ECT.

The emulator for Multics has the ability to support synchronous i/o (as well as, in a newer release, asynchronous i/o). With synchronous i/o, the emulator posts the terminate interrupt before returning from the "connect" signal sent by Multics to begin the i/o. This results in completion status being available very rapidly — possibly just after the call to ioi_$connect and before the call to ipc_$block. So sometimes, completion status is available when exercise_disk or rdisk_ checks the status word. And in this case, the code doesn't call ipc_$block, with the consequence that event messages accumulate in the ECT.

execise_disk takes a long time to run—even for smaller 451 disks. In a recent run it took 13 hours to run (I was running a debug version of the emulator, which is slower than a non-debug version, so this time might have been longer as a result). Well before exercise_disk finishes, when synchronous emulator i/o is used, the number of event messages that accumulate in the ECT causes the quota on the process directory to overflow (my process directory quota is 2000 records). exercise_disk does a lot of i/o operations and therefore one can very quickly overflow quota in the process directory. Each new ITT message allocated in the ECT area can potentially grow the area. And because the area is an extensible area, it adds new segments as needed. These segments consume more and more quota in the process directory.

If asynchronous i/o is enabled in the emulator, no quota overflow occurs. While is it possible, due to the speed of the underlying hardware and the emulator that some completion status is posted before the call to ipc_$block is made, and thus some event messages remain in the ECT when they shouldn't, the number must be far less or perhaps even zero, because quota does not appear to grow.

The problematic code in exercise_disk is:

```
do while (again);                          /* I/O loop */

     completion.st = "0"b;                          /* initialize status entry */
     completion.run = "1"b;

     call ioi_$connect (devx, dcw_offset, code);    /* Start I/O */
     if code ^= 0 then call io_err ("0"b); /* didn't get away from the starting line */

     do while (^completion.st & completion.run);    /* while connected and no status */

        call ipc_$block (addr (wait_list), addr (event_info), code); /* wait for completion */
        if code ^= 0 then call io_err ("0"b);      /* No loiterers?? */

     end;
```

As can be seen, the call to ipc_$block is skipped if there completion status or if the i/o is no longer in progress. This is the case where the event message is not extracted from the ECT and therefore accumulates.

Proposed Changes

The proposed change is to add a flag, called_block, which is set to false at each ioi_$connect iteration, and which is set to true when ipc_$block is called.

The above code is changed to:

```
do while (again);                                   /* I/O loop */

        completion.st = "0"b;                         /* initialize status entry */
        completion.run = "1"b;

        call ioi_$connect (devx, dcw_offset, code);  /* Start I/O */
        if code ^= 0 then call io_err ("0"b);         /* didn't get away from the starting line */

        called_block = "0"b;
        do while (^completion.st & completion.run);  /* while connected and no status */

             call ipc_$block (addr (wait_list), addr (event_info), code); /* wait for completion */
             if code ^= 0 then call io_err ("0"b);    /* No loiterers?? */
             called_block = "1"b;

        end;
        if ^called_block then
             call ipc_$drain_chn (event_info.chan_id, (0));
```

The event wait channel used by exercise_disk.pl1 and rdisk_.pl1 is the same event channel used by RCP when attaching the device. That event channel becomes the IOI event channel used for i/o. While there is no evidence that the same race condition described above that can leave event messages in the ECT for this event channel occurs during RCP attachment, in the event that any event messages are left in the channel after the RCP attachment, a defensive additional call to ipc_$drain_chn is added at the tail end of the “mount” procedure — which performs the RCP attachment. This guarantees that prior to the first ioi_$connect call to perform i/o, there are no event messages in the ECT for the event channel.

The compare_ascii output for the change follows:

```
Inserted in B:
B109                dcl     called_block           bit (1);
Preceding:
A104                dcl     code                   fixed bin (35);


Inserted in B:
B1369               dcl     ipc_$drain_chn         entry (fixed bin(71), fixed bin(35));
Preceding:
A1363               dcl     ioi_$set_status        entry (fixed bin, fixed bin (18), fixed bin (8), fixed bin (35));


Inserted in B:
B1978                    called_block = "0"b;
Preceding:
A1971                    do while (^completion.st & completion.run);  /* while connected and no status */


A1975
A1976                    end;
Changed by B to:
B1983                         called_block = "1"b;
B1984
B1985                    end;
B1986                    if ^called_block then                       /* drain events in case we didn't call ipc_$block
*/
B1987                         call ipc_$drain_chn (wait_list.ev_chan, (0));
```

```
Inserted in B:
B2507               /* drain event wait channel of any events that might have been read during rcp attachment */
B2508               call ipc_$drain_chn (ev_chan, (0));
B2509
Preceding:
A2496     detach: proc;

Comparison finished: 6 differences, 42 lines.
r 14:58 1.027 13
```

A similar, corresponding change will be made to rdisk_,which has the same problem. The compare_ascii output for the change follows:

```
Inserted in B:
B385      dcl  ipc_$drain_chn                 entry (fixed bin(71), fixed bin(35));
Preceding:
A380      dcl  iox_$err_no_operation          entry options (variable);


Inserted in B:
B1625     dcl  called_block                   bit (1);               /* keep track of whether we
picked up terminate event */
B1626
Preceding:
A1619               true_len = min (4 * block_len, data_left);        /* set true amount to xmit
*/


Inserted in B:
B1757                    called_block = "0"b;
Preceding:
A1749                      do while (^completion.st & completion.run);  /* while connected and no
status */


A1754
A1755                         end;
A1756
Changed by B to:
B1763                            called_block = "1"b;                     /* remember that we've
retrieved termination event */
B1764                    end;
B1765                     if ^called_block then                          /* drain events in case we
didn't call ipc_$block */
B1766                         call ipc_$drain_chn (wait_list.ev_chan, (0));
B1767


Inserted in B:
B2794                 /* drain event wait channel of any events that might have been read during
rcp attachment */
B2795               call ipc_$drain_chn (ev_chan, (0));
B2796
Preceding:
A2783     detach:

Comparison finished: 8 differences, 40 lines.
r 15:00 1.752 43
```

Documentation

The proposed changes do not affect the function and descriptions of either exercise_disk or rdisk_ and therefore no documentation changes are necessary.

Testing

The changes to exercise_disk have already been tested by running the program on a 451 disk. Prior to the changes, running exercise_disk provoked a record_quota_overflow signal. After the changes, the program completed successfully.

rdisk_ will be tested by performing a reload_volume, which uses rdisk_ to restore a volume from volume backup tapes.

Version History

Date	Revision	Author	Comment
2019-04-21	1.0	Eric Swenson	Initial revision of the MCR.
2019-04-21	1.1	Eric Swenson	Added description of ECT area explain how the ECT can grow beyond one segment, and contribute to a record_quota_overflow.
2019-05-18	1.2	Eric Swenson	Updated compare_ascii output for exercise_disk.pl1 based on review comments from Gary. Added rdisk_.pl1 compare_ascii output. Added description and justification for additional call to ipc_$drain_chn called at tail end of mount procedure.