MEDIUM TERM SCHEDULER

- Job Attributes and States
- Priorities
- Load
- Scheduling Decisions
- Scheduling Parameters
- Diagnosing Problems
- Scheduler Anomalies
- details in System Management Utilities

JOB ATTRIBUTES AND STATES

- Job Kind
  - Provided to scheduler from environment; biases scheduling decisions
  - Core Editor, Object Editor, Attached, Detached, Server, Queued
- Job State
  - Run: job is runnable
  - Wait: job is runnable but being withheld
  - Idle: job is runnable, but not using any time
  - Queued: job is detached and queued in a stream
  - Disabled: job has been externally disabled
  - Terminated: job has finished (sort of)
- see !Commands.Scheduler

PRIORITIES

- CPU Priorities
  - 16 Priority Levels
  - 0 is best priority, 15 is worst
  - Short Term Scheduler implemented in microcode
  - Strict pre-emptive priority-based scheduling
- MTS Priorities
  - 7 priority levels
  - 6 is best priority, 0 is worst
  - Only 6 and 0 supported for Foreground and Background
  - Serves as a base for Ada priorities

PRIORITIES cont'd...

- Ada Task Priorities
  - LRM does not specify a required priority range, but does define how priority levels interact: lower priority task cannot be executing when a higher priority task could execute (note that in face of virtual memory, this cannot always be guaranteed)
  - Priority is inherited from creating task; in Delta, the default priority is 1
  - Use Pragma Priority to change priority; R1000 CG defines range of 0..5 (see LRM, appendix F)
  - Task priorities are mapped to CPU priorities via MTS base priority

PRIORITY RELATIONSHIPS

CPU	MTS	ADA	
0			Kernel
1			Environment
2			
3			
4		5	Foreground Range
5		4	- core editors
6		3	- object editors
7		2	- attached jobs
8		1	
9	6	0	
10		5	Background Range
11		4	- aged attached jobs
12		3	- detached jobs
13		2	- servers
14		1	- batch jobs
15	0	0	

LOAD

- Represents number of tasks attempting to use a resource
- Maintained for running tasks, page I/O waits, and withheld tasks
- Latest value plus averages for last 1, 5, and 15 minutes
- Withheld jobs and disabled jobs included in withheld load
- What.Load, use Verbose option or Scheduler.State for detailed load

SCHEDULING DECISIONS

- Schedules jobs not tasks
  - Attempts to provide fair sharing of resources: CPU time, disk I/O, and memory
- Reviews job activity every 100 ms
  - Decisions made in response to past behavior
  - Does not guarantee any particular job will get time
- Microcode charges jobs for CPU time consumed
  - All tasks are grouped into one charge for the job
  - System time is charged back to the user job, affects rendezvous only

SCHEDULING DECISIONS cont'd...

- Job Groups
  - Job groups are given equal time
  - Core Editor is root of a job group; all jobs started in the session are mapped to this job
  - Job Groups have a budget of CPU time which is shared by all foreground jobs in the group (CE, OE, attached)
  - Budget is limited to a range specified by MTS parameters
  - As a job group executes its budget is debited; when it waits for disk or memory its budget is credited
  - Job Groups with a positive budget are not withheld until budget becomes negative
  - Allows jobs to "burst" after being idle, up to the limit set by the budget parameters

SCHEDULING DECISIONS cont'd...

- Percent For Background
  - — Foreground jobs are withheld to allow background jobs to run
  - — Includes server jobs; will impact print spooler and network activity
- Foreground Time Limit
  - — Limits elapsed time that a job may consume foreground resources
  - — Job will become "aged" and treated as background
  - — Intended to maintain interactive performance by limiting time-consuming foreground jobs
  - — Requires sensible stream parameters

MICROCODE SUPPORT

- — Independent of scheduling decisions
- — Remembers tasks that must be paged out
- — Does delays for paged out tasks
- — Provides throttling function for disabled tasks which are still paging
- — Job activity may be distorted due to microcode requirements

SCHEDULING PARAMETERS

```
-- Default values shown
-- Usually adjusted by !Machine.Initialize_Housekeeping

Cpu_Scheduling    : Enabled
Disk_Scheduling   : Enabled
Memory_Scheduling : Enabled

Percent_For_Background          : 20%
Min_ and Max_Foreground_Budget :-250 .. 250 milliseconds
Withhold_Run_Load               : 130
Withhold_Multiple_Jobs          : FALSE

Environment_Wsl            : 11000 pages
Daemon_Wsl                 : 200 pages
Min_ and Max_Ce_Wsl        : 150 .. 500 pages
Min_ and Max_Oe_Wsl        : 75 .. 750 pages
Min_ and Max_Attached_Wsl  : 50 .. 2000 pages
Min_ and Max_Detached_Wsl  : 50 .. 4000 pages
Min_ and Max_Server_Wsl    : 75 .. 1000 pages
Min_Available_Memory       : 1024 pages
Wsl_Decay_Factor           : 50 pages/5 seconds
Wsl_Growth_Factor          : 50 pages/100 milliseconds
Page_Withdrawal_Rate       : 1*640 pages/second

Min_ and Max_Disk_Load : 200 .. 250

Foreground_Time_Limit    : 1800 seconds
Background_Streams       : 3
Strict_Stream_Policy     : FALSE
Stream_Time and _Jobs 1  : 2 minutes, 2 jobs
Stream_Time and _Jobs 2  : 8 minutes, 1 job
Stream_Time and _Jobs 3  : 50 minutes, 0 jobs
```

DIAGNOSING PROBLEMS

- Talk to users
- Scheduler.Display, Scheduler.State
  - — Shows parameters, job state, load, and streams
  - — Check for strange combinations, unusual load values
- What.Jobs, What.Users, What.Load
  - — Observe job states and percentages assigned to jobs
- Show_Jobs, Show_Tasks
  - — Find out what job is waiting on
- Use Kernel Command Interpreter
  - — Show_Mts_Params, Jobs_Mts, Load, MtsQ
  - — Enable_Job, Disable_Job

SCHEDULER ANOMALIES

- Priority Conflicts
  - interfere with CE, gets equal time with CE input/output tasks; normally, only other CE jobs compete
- Editor Operations
  - Large search and replace
  - Command.Spawn: job detaches and is queued before CE finishes execution; results in hung session; wait or kill job from another session
- Memory Scheduling
  - Archive Server - bumps into page limits
  - MTS doesn't know about page creates; can cause poor choice of victim
- Cross-VP charge back
  - Disable a job stuck in rendezvous will have no effect

Topics

- Scheduler
- Disk Garbage Collection
- Disk Errors
- Remote Debugger
- Miscellaneous Topics

Disk Garbage Generation

- Everything you do creates garbage
  - — Commiting a unit creates grabage
  - — Promoting a 100K unit to installed creates 50-200K of garbage
  - — Coding a unit leaves the previous code segment as garbage that is not reclaimed until the next boot
  - — Elaborating a big program whose size in N Mbytes whose working set limit is L Mbytes will generate N-L Mbytes of garbage
  - — We suggest that customers use small packages for editing and compilation efficiency. Since each package consumes several pages at a minimum, this makes runtime performance worse and creates more garbage.

Disk Garbage Generation cont'd...

— Consider the following:

```
subtype Length_Type is Positive range 1..65536;
type Bounded_String (Limit : Length_Type := Length'last) is
  record
    Length   : Length_Type;
    Chars    : string (1..Limit);
end record;
Table  : array (1..250) of Bounded_String;
```

— This will initialize 16 Mbytes of data stack, or hit it's job page limit. This will often hit the disk at the rate of several Mbytes per minute.

— Job disable will not disable the job before the space is allocated because the allocation is done by a single instruction.

— Resolving names creates garbage

— Temp Heaps

- The environment uses lots of temp heaps. These become garbage if not explicitly deleted. Thus, killing jobs, force logoff, and bugs leave garbage temp heaps which are not reclaimed until the next system boot.

Collector Operation

- Zapping moldy vps and spaces
  - — Delete stuff that the kernel knows is to be recycled
- Traversing
  - — Search to find all allocated disk space
  - — If disk is empty, this is fast
- Reclaiming
  - — Return garbage to the free space map

GC Priority

- Priority -1
  - — Backoff during traversal phase if load is "big". Restart if load is "small".
    - Big: Withheld_Last_Sample > 0 or Run_Last_Sample > 2.0
    - Small: Withheld_Last_5_Min < .75 and Run_Last_5_Min < .75
  - — Backup parameters are set by operations in !Tools.Disk_Daemon
- Priority 0
  - — No backoff
  - — Collector makes progress; some performace degradation
  - — Doesn't make much progress if load is moderate to high

GC Priority cont'd...

- Priority 2
  - — Will preempt most background jobs. Runs on par with a background job that uses the best 'priority.
- Priority 3
  - — Runs on par with most foreground jobs. Tends to have a big impact on performance, since it will compete with commands.
- Priority 4
  - — Preempts most foreground jobs. Should still be able to edit. But commands will run VERY slowly.
- Priority 6
  - — No backoff; Preempts virtually all activity, except that from the console.
  - — No guarantee of progress in face of high CPU load or Job 4 activity.

Collector Thresholds

- Start GC Threshold
  - — When disk space on a volume drops below its start threshold, the GC starts at priority -1. Recall that the GC is a single task and can only collect one volume at a time. picks lowest free space
- Raise Priority
  - — Priority is raised so that collector can make better progress
- Stop Jobs
  - — Stop all jobs > 5.
  - — Kill current GC and start again at higher priority.
- Suspend System
  - — Don't allocate any more space on the volume
  - — Typically hangs the machine
  - — Must reboot to make progress if this happens

Interaction with Backup

— GC and backup cannot run concurrently. This is limitation of the retained snapshot mechanism.

— By default, GC will run a job.kill on an in-progress backup which has not yet requested the blue tape

— To change this, use Disk_Daemon.Set_Backup_Killing in !Tools
not preserved between boots

Checking Remaining Disk Space

- Operator.Disk_Space

```
Volume  Capacity  Available   Used   % Free
======  ========  =========  ======  ======
1        515889    244737    271152   47
2        269280    218664     50616   81
3        269280    215190     54090   79
4        269280    236713     32567   87

Total   1323729    915304    408425   69
```

- Kernel: Show_Volume_Summary

```
Kernel: show_volume_summary
Volume Status Summary

 Vol     Total     Unused      Rate
 Num   Capacity   Capacity   Blks/Min
--------------------------------------
   1    515889     244690          2
   2    269280     218579          5

low space thresholds for volume 1:
    START_COLLECTION threshold at 25% (waiters exist)
    RAISE_PRIORITY threshold at 15% (waiters exist)
    STOP_JOBS threshold at 12% (waiters exist)
    SUSPEND_SYSTEM threshold at 7% (waiters exist)
    SPACE_04 threshold at 0% (no waiters)
    next trigger at 128972 blocks
low space thresholds for volume 2:
    START_COLLECTION threshold at 25% (waiters exist)
    RAISE_PRIORITY threshold at 15% (waiters exist)
    STOP_JOBS threshold at 10% (waiters exist)
    SUSPEND_SYSTEM threshold at 8% (waiters exist)
    SPACE_04 threshold at 0% (no waiters)
    next trigger at 67320 blocks

Debugging information:
    OUT_OF_SPACE_EVENT_PAGE_ADDR => ( 1023, DATA, 259, 504)
```

Out of Disk Space?

- Check error log
- Kernel: Show_Volume_Summary
- Messages in message windows
- Kernel Show_Volume_Summary says 0
  - This means that the suspend system threshold has been crossed.
  - Reboot immediately (to EEDB). If a snapshot goes by, even more space will be lost, putting the system into peril.
  - System is hung, so no loss by rebooting

Is GC Running?

- Daemon.Status("Disk")
  - Shows what phase collector is in and how far it's got
- Check error log

  Kernel: show_GC_state
- It's running, but is it making progress?
  - Look at CPU and disk use on Job 5 with (kernel) Job or (environment) Show_Jobs.
- Waiters
  - Check the show_volume_summary kernel display. Does it indicate waiters for each volume? If not, GC is probably running on the volume that says "no waiters".

Manual Controls

- Start GC
  - Daemon.Run("Disk") - runs on all volumes in order of which has the least available space
  - Daemon.Collect(volume#) - runs GC on specified volume.
- Priority Control
  - Daemon.Set_Priority(pri) - set priority of a currently running GC
- Other operations
  - See !Tools.Disk_Daemon

Disk Eaters

- To find a job that is consuming disk space
  - — If space is dropping and/or keeping the GC running, here are some ways to find the job that is doing it
  - — Run Show_Jobs (or kernel Jobs). Look for:
    - High Disk Wait count or D/S. Each new page allocated requires a few disk waits
    - Large Job Segment
    - Large Disk Page Count

file I/o only shows in D/S
list part of Job Segment

Disk Eaters cont'd...

- Try disabling or killing such a job to see if allocation stops
  - — Don't be afraid to disable editor jobs
  - — The simplest strategy is to leave the jobs disabled until the next boot.
  - — Disabling a job will not recovery space it has comsumed.

Disk Eaters cont'd...

- Job 4 Problems
  - —If the consumer appears to be job 4, it is harder to locate the actual cause.
  - —If disk space is very low, crash the system. This will stop allocation and disk collection can start when it reboots.
  - —You cannot disable job 4, and disabling the job "responsible" for the allocation will have no effect.

Disk Eaters cont'd...

- The GC itself will consume disk space during its operation
  - —It should not run the system out of space
  - —It will stop before it is done collecting if space gets too low.
  - —The GC will have to run a second time to complete collection

GC Threshold Settings

- Suggested values
  - Start - 25%
  - Raise priority - 15%
  - Stop Jobs - 10%
  - Suspend System - 8%
- Volume 1 is more critical
  - Stop Jobs at 12%

Suspend Threshold

- System Hangs
  - Users will probably know why - messages displayed
  - Check error log and Show_Volume_Summary command
- Reboot
  - There is no other recovery from the suspend system threshold

Suspend Threshold cont'd...

- Recovery procedure
  - — Boot the Kernel configuration
  - — Using the kernel command interpreter, lower the Suspend_System threshold for the affected volume (from 8% to 3% is good)

```
Kernel: change_gc_thresholds
VOLUME_NUMBER [1]: 2
THRESHOLD [START_COLLECTION]: suspend_system
REMAINING CAPACITY (%) [10]: 3
```

Suspend Threshold cont'd...

- — Start virtual memory: Defaults command (privileged)
- — Once EEDB is up, elaborate to the disk cleaner subsystem.
  - Build a configuration DDC (if necessary) and elaborate it:

```
EEDB: running
  D_9_21_1
EEDB: build ddc
Existing Configuration: d_9_21_1
Parent subsystem: ddc
Subsystem.Version:
```

  - The DDC configuration is usually shipped with the system

Suspend Threshold cont'd...

- — The GC will start running and should complete successfully
- — Reset the Suspend_System threshold higher, and elaborate the rest of the configuration

- Don't be shy about calling for help if things don't go well

Action to Free Space

- If the Stop_Jobs threshold has been reached, take some action to reduce the disk usage:
  - — Increase the Start_GC threshold so that there is better warning before space runs out.
  - — Run Lib.Expunge to free space held by deleted version
  - — Redistribute worlds to better balance disk utilization
  - — Lower retention counts
  - — Demote old units to Archived
  - — Delete unneeded views and worlds

GC Notes

— NEVER configure a system to automatically boot without operator intervention. Since the system creates garbage at each boot, this might run the system out of space.

— Space gets harder to reclaim the less there is of it. Don't procrastinate!

— Assuming there is at least 15% space remaining, feel free to reboot then system when the Stop_Jobs threshold is reached. This looses the reboot time, but definitely stops any jobs that were consuming the space.

— Reboot weekly to reclaim temp heaps and code segments

Finding space

- Lib.Space

```
Lib.Space (For_Object => "!users",
           Levels => 2,
           Recursive => True,
           Each_Object => False,
           Each_Version => False,
           Space_Types => False,
           Response => "<PROFILE>",
           Options => "");
```

display only - always looks for values

- Show_Memory_Hogs

  — Scans memory for large objects. See example

  — Also available in Kernel, Hogs command

  — Need to improve it some

Lib.Space Output

direct ⌐sub structure

```
    Object Total
Vol  Size   Size     Object Name
--- ------ ------ ----------------------------------------------
                  !USERS
                  .USERS
                    .CLP
       63             .BIN                          (DIRECTORY)
 1   1352             .LASER_STUFF                  (WORLD)
 1     76   1491    .CLP                            (WORLD)
 4    283           .GURU_COURSE                    (WORLD)
                    .JIM
      109             .LOGIN_STUFF                  (DIRECTORY)
      467             .PROGRAMS                     (DIRECTORY)
 1    384             .UOC                          (WORLD)
 1     48   1008    .JIM                            (WORLD)
                    .MARLIN
      558             .COV_TEMP                     (DIRECTORY)
 1   1525             .DISASSEMBLER                 (WORLD)
 1     10             .EDIT                         (WORLD)
 1    279             .ENV                          (WORLD)
 1    437   1594      .HISTOGRAM                    (WORLD)
 1   2289   8130      .MTS                          (WORLD)
 1    143             .PERFORMANCE                  (WORLD)
 1     67             .SUPPORT                      (WORLD)
 1     57             .TEST_ERROR_LOG               (WORLD)
 1    346             .XRAY                         (WORLD)
 1   1208  13928    .MARLIN                         (WORLD)
                    .OPERATOR
 4     11             .TEST                         (WORLD)
 1    284    295    .OPERATOR                       (WORLD)
                    .PHIL
 1    237    247      .ACCESS_CONTROL               (WORLD)
     3558             .CRUD_1_ARCHIVE               (DIRECTORY)
      399             .CRUD_RELEASE_NOTES_D_9_20_2  (DIRECTORY)
 1    208             .DEBUGGER_COURSE              (WORLD)
      134             .DELTA_RELEASE_NOTE           (DIRECTORY)
 1   1038   1049      .GURU_COURSE                  (WORLD)
 1    141             .SPOOLER_INIT                 (WORLD)
 4     58             .TEST_AREA                    (WORLD)
 4    407             .UNCHECKED_CONVERSION         (WORLD)
 1    453   7099    .PHIL                           (WORLD)
 1     10           .RATIONAL                       (WORLD)
                    .SMP
 4     10    416      .DELTA                        (WORLD)
 1  55601  56017    .SMP                            (WORLD)
                    .SRP
        8     59      .DOCUMENTATION                (DIRECTORY)
 1     26    415      .EHR_OE_TESTS                 (WORLD)
 1    101    575    .SRP                            (WORLD)
 1     35  85979  .USERS                            (WORLD)
```

pages

Show_Memory_Hogs output

```
      Show_Memory_Hogs (Vp => 256, Volume => 1,
                        Size_Threshold => 250);

( 256,MODULE, 74471)  commit_time: 205
        page_count: 273  mark:  227: Image: Permanent editor buffers
( 256,MODULE, 77631)  commit_time: 236
        page_count: 350  mark:  151: ADA Data
( 256,MODULE, 82945)  commit_time: 278
        page_count: 53577  mark:  153: FILE Data
( 256,MODULE, 83024)  commit_time: 278
        page_count: 1608  mark:  153: FILE Data
( 256,MODULE, 95839)  commit_time: 469
        page_count: 262  mark:  151: ADA Data

-- Ask kernel to convert virtual address to object id

Kernel: enable_priv_cmds
*Kernel: show_space_info
VPID [0]: 256
KIND [MODULE]:
SEGMENT [0]: 82945
SNAPSHOT_NUMBER [0]: 278
THE_SPACE        => ( 256,MODULE, 82945)
COMMIT_TIME      => 278
...
DELETED          => FALSE
USER_DATA        => 153: FILE Data
OBJECT           => Manager 3Instance 13680

-- Next convert object id to name

    Action_Utilities.Display_Object (3, 13680, 1);

-- Answer:

!USERS.SMP.DATA'V(1)
```

Diagnostic Tool Summary

- System Maintenance
  - Show_Groups - access control
  - Show_Identity - access control
  - Show_Jobs - system resources and activity
  - Show_Job_Names - system resources and activity
  - Show_Locks - object locking; job deadlock
  - Show_Machine_Id - hardware; product authorization
  - Show_Memory_Hogs - disk utilization
  - Show_Stats - job statistics
  - Show_Tasks - job activity and status
  - Check_Universe_Acls - access control
  - Set_Universe_Acls - access control
  - Show_Error_Log - error log display

Diagnostic Tool Summary cont'd...

- Kernel
  - Jobs/Job - resource utilization and job activity
  - Job_Names/Job_Name - job information
  - Jobs_Mts - scheduling
  - Show_Volume_Summary - disk space and GC info
  - Show_Gc_State - GC info
  - Show_Disk_Summary - disk use and errors
  - Enable/Disable_Job - job control
  - Show_Task_States - status of interesting tasks
  - Show_Error_Log - error log display
  - Show_Vps - vp -> disk mapping
  - Show_Space_Info - map from VM address to object
  - Roust - roust a task out of wait service

— Hogs - find large memory use

— LMR/LMW - read/write memory

— Abort_Task - kill any task (or job)

Diagnostic Tool Summary cont'd...

- System Availability
  - System_Report.Generate

Diagnostic Tool Summary cont'd...

- Environment Commands
  - — Daemon.Status - daemon information
  - — Daemon.Status("Disk") - disk collection info if running
  - — Action_Utilities.Lock_Information - super what.locks
  - — Action_Utilities.Display_Object - object id to name conversion

Miscellaneous Topics

- — OM tests   OM_Tests : read   gives to EEDB if appropriate
  stores if for OM
- — OM file system   vd $\frac{+}{*}$?
  OM state as files
  can be deleted
  secure systems
  copyright file
- — Editor connect$\overset{e}{\text{d}}$ terminal model
  attached
  job attached to terminal
  not free for input   - no job started,
  just CE active
  ↑G   User Interrupt   (Job N. Queued)
  ↑G   "   "

- Dir tester
  - never be used in field
    not reported - unknown bugs
  - dangerous

Diagnostic Tool Summary cont'd...

- Environment Commands
  - Daemon.Status - daemon information
  - Daemon.Status("Disk") - disk collection info if running
  - Action_Utilities.Lock_Information - super what.locks
  - Action_Utilities.Display_Object - object id to name conversion

Disk Errors

— Prior to the message "the virtual memory system is up", disk errors appear only on the console.

— After the message, disk errors appear on both the console, and in the system error log.

— Log entries in the error log identify, among other things, the virtual memory address and disk block address involved in the failed IO.

— ATTEMPT: (1023, DATA, 259, 10234) <== (3, 10234)

— 4-tuple gives the virtual address of the page involved in the IO

— -tuple gives the disk block address.

— "arrow" identifies whether the IO is a read/write.

— Above example: read from block 10234 from volume 3 into virtual page 10234 of data segment 259 of vp 1023.

Disk Driver Logic

- Write
  - — Try to write the addressed disk location. Bad status results in up to 30 retries
  - — Then: the block is retargeted (described further below). Unless retargeting fails, a write will always be "successful".

Disk Driver Logic cont'd...

- Read
  - — Try to read the addressed disk location. Initial good status causes the read to be considered "successful". Up to 10 retries.
  - — Offset heads "advanced". 10 retries
  - — Offset heads "retarded". 10 retries
  - — Two successive tries yielding the "same result" after ECC cause the read to be considered "successful".
  - — If this fails, the block is considered unrecoverable
  - — If a successful read encountered one or more non seek errors, the block is retargeted

Retargeting

— Write the good data to a new location, and redirect all future IO to the old location to the new location.

— Displaying retarget database:

- Kernel: Show_Bad_Blocks
- No output means database is empty

— Messages sent to all users when a retarget occurs

— Machine calls Rational, also

After an Unrecoverable Error

— System continues to do IO to volume with error

— System will usually hang eventually

- Error on kernel VP - will cause snapshots to hang
- Page replacement policy will stumble on bad page

— Error log entries will be made unless disk error is in error log

— Job involved in bad page will hang immediately. The job cannot be killed.

After an Unrecoverable Error cont'd...

— Kernel Show_Disk_Summary command shows number of disk errors

```
Kernel: show_disk_summary
DISK STATUS SUMMARY

               Q   IOP    Total      Total    Seek  Soft  Hard   Un    Total
~~Total~~
  Vol  Unt   Len   Len    Reads      Writes   Errs  Ecc   Ecc   Recov  Errs
~~Recov   Errs~~
---------------------------------------------------------------------------
---------
    1    0     0     0    311312      98342     0     0     0     ϕ     ϕ
~~0~~          0
    2    1     0     0    102090     119296     0     0     0     ϕ     ϕ
~~0~~          0
    3    2     0     0    103753     127907     0     0     0     ϕ     1
~~0~~          1
    4    3     0     0     68767     213141     0     0     0     ϕ     ϕ
~~0~~          ~~0~~

no disk IO in progress

Debugging information:
Ready_Volume mask => 0
Busy_Event_Page => ( 1023, DATA, 259, 241)
Volume_Offline_Event_Page => ( 1023, DATA, 259, 242)
```

After an Unrecoverable Error cont'd...

— To find if the system is hung due to a disk error:

- Run Show_Disk_Summary, Show_Task_States (cache), Show_Disk_Summary
- If the 2 Show_Disk_Summary commands have the same values for Total Reads/Writes and the Show_Task_States command shows one or more modules in disk wait, then the system is probably hung do an unrecoverable disk error. As further confirmation, one could compare the page address printed by the Show_Task_States command with those printed in the disk error log entries.

This note describes procedures for handling unrecoverable disk read errors in the R1000 file system under Delta0 software.

This note does NOT cover the following topics: (a) Dealing with disk errors in the iop file system. (b) Diagnosing drive/controller faults. That is, this note is not going to tell you how (from the status messages, or other information) to ascertain whether the problem stems from software, controller, drive, hda, media, etc. Assuming that you have already determined that the problem is a relatively isolated media defect, then this note will (hopefully) help you.

Prior to the message "the virtual memory system is up", disk errors appear only on the console. After the message, disk errors appear on both the console, and in the system error log.

The disk error entries in the system error log identify, among other things, the virtual memory address and disk block address involved in the failed IO. The log entry will contain a line which looks like:

```
ATTEMPT: (1023, DATA, 259, 10234) <== (3, 10234)
```

The 4-tuple gives the virtual address of the page involved in the IO, the 2-tuple gives the disk block address. The "arrow" identifies whether the IO is a read/write. In this example, the IO is a read from block 10234 from volume 3 into virtual page 10234 of data segment 259 of vp 1023.

The device driver logic for a write (from memory to disk) is basically as follows: Try to write the addressed disk location. Bad status results in up to 30 retries. If the retry limit is reached, or non seek errors occurred, the block is retargeted (described further below). Unless retargeting fails, a write will always be "successful".

The device driver logic for a read (from disk to memory) is basically as follows: Try to read the addressed disk location. Initial good status causes the read to be considered "successful". Bad status results in up to 30 retries. The first 10 retries are without offset heads. The next 10 retries have the heads advanced. The next 10 retries have the heads retarded. Two successive tries yielding the "same result" cause the read to be considered "successful". By same result we mean the value returned by the controller, after soft ecc correction, if necessary. Otherwise, after exhausting the 30 retries, the read is considered "unrecoverable". If a successful read encountered one or more non seek errors, the block is retargeted, as discussed below.

By retargeting we mean write the good data to a new location, and redirect all future IO to the old location to the new location. The retarget database can be examined by using the Show_Bad_Blocks command in the kernel command interpreter, supplying "Retarget" as the answer to the "Kind" prompt. If the command prints nothing, the retarget database is empty.

With the Eagle drives, there have been 3 cases where the presence of multiple retarget database entries have predicted a future severe disk problem (head crash, multiple hard ecc errors, etc). Personal opinion: If multiple retarget entries showed up on my machine, I would take incrementals twice a day until either the drive crapped out or several weeks had passed without additional errors.

The system will continue to do disk IO to a volume that has experienced

an unrecoverable error. This will typically allow log entries to be made even though volume 1 is experiencing unrecoverable disk errors.

However, it will still be the case that after an unrecoverable error the system may eventually hang. This will typically happen for one of the following reasons: (1) If the error involved a kernel disk mapping page (vp = 1023), snapshot will get hung (waiting for IO to complete, which will never complete). So, within 2 snapshots, the system will certainly be tangled up in disk wait. (2) Regardless of what kind of page was involved in the unrecoverable disk error, the page is left in the cache "in transit". The page replacement policy may eventually stumble across this in transit page, causing jobs to become forever stuck waiting for the IO to complete. If the above technical explanation for the hang behaviour doesn't make any sense, just ignore it.

Regardless of the reason a job gets hung in disk wait, the job cannot be killed, short of crashing the machine.

The Show_Disk_Summary command in the kernel command interpreter can be used to display the number of disk errors. In particular, the second to last column (labelled "Un Recov") shows the number of unrecoverable errors which have occurred on the volume since last boot. A non-zero value in any row of this column indicates that the system is or will eventually become hung.

One can determine whether the system is currently hung from unrecoverable errors by the following procedure: Do a Show_Disk_Summary command. Do a Show_Task_States (with Cache default) command. Do another Show_Disk_Summary command. If the 2 Show_Disk_Summary commands have the same values for Total Reads/Writes and the Show_Task_States command shows one or more modules in disk wait, then the system is probably hung do an unrecoverable disk error. As further confirmation, one could compare the page address printed by the Show_Task_States command with those printed in the disk error log entries.

Given the page address involved in the error, one can often discover the identity of the object via this procedure. The vpid cannot be 1023. Use the Show_Space_Info (privileged) command:

The procedure for identifying the object (damaged by an unrecoverable disk error) and recovering is as follows:

```
    case vp (from virtual address) is
        when 4 .. 5 =>
            The segment kind should be one of CONTROL, TYP, DATA, QUEUE,
            or IMPORT.  The error occurred in a runtime module/import_space
            of the environment.  Increased likelyhood of system hang.
            Problem is corrected by rebooting.

        when 8 .. 26 =>
            The segment kind should be CODE. The error occurred in an
            environment code segment.  Increased likelyhood of system hang.
            Problem is corrected by (a) rebooting to EEDB, (b) deleting the
            bad segment (see example at end of document), (c) reloading all of
            the appropriate AE tapes, and (d) elaborating the environment.

        when 27 .. 255 =>
            The segment kind should be one of CONTROL, TYP, DATA, QUEUE, or
            IMPORT.  The error occurred in a runtime module/import_space of
            the job (whose number is the same as the vp).  This job cannot be
        killed.  Problem is corrected by rebooting.

    when 256 .. 1022 =>
        The segment kind should be CODE or DATA.

        In the first case (CODE), the error occurred in the code
        segment of some coded Ada unit, somewhere in the machine.

        In the second case (DATA), the error occurred in some object,
        somewhere in the machine.

        Recommended recovery procedure:  Reboot to EEDB.  At the kernel
        command interpreter, use the Show_Space_Info (privileged) command:

             *Kernel: show_space_info
             VPID [4]: <vp>
             KIND [MODULE]:
             SEGMENT [0]: <segment>
             SNAPSHOT_NUMBER [0]:

        If the command produces no output, then the disk error occurred in
        a temporary or superseded segment, and the problem has gone away.
        Otherwise, the command will print
             most recent generation: <snap #>
        Re-execute the command, supplying <snap #> to the last prompt.
        It will print out a bunch of stuff.  The last 2 lines of output
        should look like:

             user_data => <name> (<mark>)
             object    => manager <m> instance <I>

        For example, if the object was an Ada unit, the last lines
        would look like:

             user_data => Ada Data (150)
             object    => Manager 1 Instance 457

        If the object_id is not 0, and the mark does not identify manager
        state, then with high probability you can (a) elaborate the
        environment, (b) use lib.resolve ("<[m, i, 1]>") to get the
        full pathname of the object in question.  For vanilla files,
        you can simply delete the damaged object.  You can sometimes
        delete damaged Ada units.

        If the above fails, further identification and recovery proceeds
        as follows:

            case <mark> is
                when 99 .. 100 | 114..119 | 200..1023 =>
                    Should not see these, since a mark of this class
                    identifies a temporary segment, which should have
                    been deleted by the reboot.

                when 101..113 =>
                    Error occurred in permanent environment state.
                    Follow procedure outlined below for 120..149.

                when 120..149 =>
                    Error occurred in object management state.  If you feel
                    lucky, run all the compaction daemons, followed by disk
```

```
                    garbage collection on all volumes.  If this doesn't
                    stumble across the disk error, then the disk error
                    occurred in object management state which is deleted
                    by crash recovery, and the problem has gone away.
                    Otherwise, the recommended recovery procedure is to
                    restore the system from backup tapes.  Feel free to
                    try to source archive recent work; the source archive
                    may or may not run across the bad block.

                when 150..179 =>
                    Error occurred in an object with a directory
                    pathname.  Locate the object via the following
                    command (from an editor command window):
                          Disk_Space.Name_Space (Vp => <vp>,
                                                 Kind => Disk_Space.Data,
                                                 Segment => <segment>,
                                                 Vol_Hint => <volume>);
                    where <vp>, <segment> and <volume> come from the
                    information contained in the appropriate entry in the
                    system error log.  The command will run for a long
                    time (on the order of an hour).

                    If the command produces no useful output, then the
                    error occurred in some object which is permanent
                    garbage, or simply not understood by the command.
                    Two options are available: leave the unknown object
                    damaged, or source archive and restore from backup
                    and archive tapes.

                    If the output of the command identifies the object(s)
                    corresponding to the page address involved in the
                    disk error then the recovery procedure is:

                        case <mark> is
                            when 150 =>
                                The error is in an Ada unit.  Deleting
                                the unit may or may not stumble across
                                the bad block.  If the unit state is
                                installed or better, deleting the unit
                                will almost certainly hit the bad block.
                                Two basic options are available: leave
                                the unit damaged, or source archive
                                around it and restore from backup and
                                archive tapes.

                            when 152 =>
                                The error occurred in some "file".
                                Simply delete the file.  Note that the
                                file may actually contain switches, etc.

                            when 160 =>
                                The error occurred in the link pack of
                                some world.  Further compilation commands
                                in the world may stumble across the bad
                                block.  Two basic options are available:
                                leave the world damaged, or source archive
                                around it and restore from backup and
                                archive tapes.
                        end case;
            end case;
```

```
        when 1023 =>
            The segment kind should be DATA.

            The error occurred in a kernel data structure.  Reboot to
            EEDB.  If the boot process stumbles across the bad block, then
            the only recovery procedure is to restore from backup tapes.

            Otherwise, chances are very good that the error occurred in the
            index structure for some virtual memory segment.  Use the
            Zero_Block command (in the kernel command interpreter); give it the
            disk block address from the error log entry.  Better be careful,
            or you could really trash permanent state!  Elaborate up to
            DDC.  Set the Start_Gc threshold low enough to trigger the
            garbage collector on the volume containing the error.  If you get
            a software crash, the only recovery procedure is to restore from
            backup tapes.  If the garbage collector finishes without producing
            any unusual messages, then the disk error occurred in a temporary
            or superseded segment, and the problem has therefore gone away.
            Or, the garbage collector may identify the segment(s) corresponding
            to the zeroed block.  In this case, feed the segment name(s) to
            the Disk_Space.Name_Space procedure (from a command window in the
            editor) following the procedure described above (under vp in the
            range 256..1022). Note that in this last case, the garbage
            collector will die as part of identifying the bad segment(s).
            After zeroing the index block, if you "touch" the segment, the
            machine will probably crash; touch includes definition, trying
            to delete the object, etc.

    end case;
```

Should you choose the option of leaving the damaged object on the system, you will need to perform the following, to prevent system backup from stumbling across the bad block:

```
      if the block is in a kernel data structure
              (the vp of the page was 1023) then

          *kernel: delete_space
          VPID [4]: <vp>
          KIND [MODULE]: <kind>
          SEGMENT [0]: <segment>

          *kernel: create_empty_space
          VPID [4]: <vp>
          KIND [MODULE]: <kind>
          SEGMENT [0]: <segment>
          where the <vp> <kind> and <segment> values are those corresponding
          to the segment to which the bad index block belongs; recall that
          the segment name was identified by running gc.

      else
          use the Zero_Block command on the bad block (the one identified
          in the disk error status messages).
      end if;
```

If you choose the recovery from backup tape option instead of the leave it damaged option, it is recommended that you temporarily use the leave it damaged option, in order to take a full system backup. That way, if the restore (to previous state) fails, this extra backup can be used to return

to the current state (with the damaged object). Of course, following this recommendation implies that you have to follow the above procedure for taking the leave it damaged option.

Hint: when using the kernel command interpreter, use it via EEDB, since it is more forgiving.

Hint: many of the correction procedures indicate to reboot. Feel free to do a shutdown by Quit'ing from EEDB; if that doesn't work (typically because the system hangs), one can always resort to crashing it.

Hint: The Disk_Space.Name_Space command can be used to compute the segment address for an object by taking default parameters, setting Root to the name of the object, and Traversals to Directory_Only.