UNISYS

INSIDE YOUR LIVING MCP

A TUTORIAL

B1000

CUBE

SPRING - 1987

ALS

WHO GETS A PROCESSOR

In essence: the highest priority job that needs it

GISMO "micro-scheduler" allocates processor(s).

Scheduler called

1) by SYSTEM/INIT to start the MCP
2) whenever any program (including MCP) interacts with operating system via "communicate".

Scheduler:

Checks for non-correctable memory errors and halts

Handles interrupts appropriately
(timer interrupts, I/O interrupts, console interrupt switch, port interrupts)

Works with 2 bits: "reschedule" and "timer_occurred".

Timer_occurred (every .1 seconds):

updates MPRI fields in all links
sees if any peripherals have changed state
(tape or disk mounted, card reader on etc.)
adjusts wait time for jobs hung on time
"causes" jobs if time decrements to zero

WHO GETS A PROCESSOR (continued)

Reschedule bits (one per processor)

Avoids needlessly rescheduling a processor back to the same job

Set when a higher priority job needs the processor

Hi-priority interrupt (reader-sorter)

GISMO immediately runs MMCP reader-sorter code which tanks check images and calls user's use routine

MCP first choice if "ready" to run (had been preempted by higher-priority job while running) and either

a) MCP-dispatched I/O complete

b) MCP wants control exclusively "blocking" processors

c) N-SECOND time (a periodic housekeeping time)

Otherwise, highest priority job needing processor gets it; SMCP wins all ties. Selected job marked "not queued" and reinstated.

SLAVE scheduler similar

- timer not checked or handled
- must be aware of blocking requests from master
- won't select jobs in the S_COMM_QUEUE
- takes highest-priority job in "READY QUEUE, "IOC_QUEUE," or M_COMM_QUEUE". If none, merely idles.

BRINGING UP THE SYSTEM

CLEAR/START:

- Dumps memory if requested
- Reads in SYSTEM/INIT and transfers control.

SYSTEM/INIT

- Finishes system dump - dumps certain structures and MCP's layout tables
- Locates system disk, ODT channel, master processor
- Ascertains if there's a slave processor; allocates data space for it
- Allocates memory for:
  - HINTS;
  - code directory for MCP;
  - interpreter directory;
  - CSV (and reads the structure in);
  - GISMO;
  - lamps on B1800;
  - MMCP's data area;
  - "chip" table;
  - SMCP RSN.
- Reads in GISMO; discards certain segments
- Checks firmware compatabilities
- Reads in 3 segments of SMCP
  - page zero, seg 0 (tiny, merely gets us going)
  - page one, seg 2 (our C/S code)
  - page one, seg 0 (all non-overlayable MCP code)
- Sets up memory links
- Loads SDL2 interpreter
- Fires slave and gives to GISMO
- Clears cache
- Gives master to GISMO

BRINGING UP THE SYSTEM (continued)

MCP procedure "GET_READY"

- Checks compatabilities
- Checks DD validity
- Sets up system date/time from pre-C/S value.
- Allocates space for and creates
  - MCP disk descriptor
  - SYSTEM/ODT's ODT descriptor
- Allocates space for and creates all the structures used on a running system
  - IOAT, channel tables, job queues, etc.
- Ascertains what is on every port & channel via test ops
- Elogs the C/S hardware & software configurations
- Cleans up system disk
- Initiates: PANDA for "protected" files on system disk
  - SYSTEM/ODT
  - handler
  - MCS
- Checks dumpfile for proper size
- Checks "truth table"
- Spouts "CLEAR/START" message

MCP then enters its main driving procedure:

GET_SET___GO

MCP OUTER LOOP

MCP always sitting in a wait statement just like any user program.

Waits for one of five events to occur.

The five events are:

1) TIME
2) INTERRUPT for MCP
3) READ_OK on the queue from SYSTEM/ODT
4) "S_C_Q" event: some job has need for MCP service
5) "change bit": we hope to send a job to BOJ

Jargon: it is "HUNG" on the five events.

When any program in a wait status has one of its wait events come true, GISMO "causes" the program.

"Cause" => put program into its next queue unless the "intervention bit" is set in its RSN, in which case it is given to the MCP (put in the S_COMM_QUEUE), normally for rollin.

Prioritization of events: if two or more occur before GISMO can check events, the leftmost event is caused.

THE TIME EVENT

- MCP always waiting on TIME
- Time interval recalculated every time MCP wakes up on some other event
- Arbitrary interval: 5 x jobs_running;

  5 <= interval <= 60
- We term the interval, as well as the tasks performed when the interval lapses, "N_SECOND".
- MCP can hasten the onset of N_SECOND when it is desirable not to wait for up to a minute.

  Example: setting up a pseudo reader via RN 1

  We term this "forcing N_SECOND"

When time event comes true, MCP invokes procedure EXTERMINATE_AND_N_SECOND, which:

1) Exterminates jobs that are to be automatically DSed.
    - a) Jobs that aborted with TERM or TRMD
    - b) Certain critical system programs that failed (SYSTEM/ODT for example)

2) Calls procedure N_SECOND.

N-SECOND FUNCTION

- Rolls out jobs to free up memory
  - ALL "ST"-ed jobs
  - lowest priority "waiting job" after 1 n-second grace period
- DR/TR prompt
- Adjusts various day, date, and time counters
  - "GOOD MORNING" message at midnight
- Checks jobs for exceeding time limits
- Initiates any pending delayed random I/O's
- Checks jobs waiting STARTTIME
- Initiates pseudo readers
- Transfers ELOG if it's rather full
- Updates LOG_MIX_INFO
- Queue file garbage collection
- Fires up SYSTEM/BACKUP if autobackup parameters warrant
- Fires up SYSTEM/ODT if it went down
- Updates error rate tables (ER stuff)

N-SECOND FUNCTION (continued)

- ELOGs errors from disk cache on B1990

- Handles RESTARTS if all necessary packs online

- Writes Cold Start Variables (every 4 n-seconds)

- Terminates handler if 3 n-seconds pass with no remote file open.

- Recalculates n-second for the next time.

NOTE: Certain of these functions that would entail disk allocation are supressed during a system disk squash.

THE INTERRUPT EVENT

. Set by GISMO when entry exists in interrupt queue

. What's the interrupt queue?
  62 Element array
  ultra-high memory - even MCP can't play with it
  Accessed via special S-OPS: "FETCH" in SDL2

. "WAKE UP" signal to MCP
  - MCP wants to know when an I/O completed
  - GISMO wants MCP to know about something that had not been explicitly requested by the MCP.
    Examples: User I/O had exception, requiring the MCP to do retries, elogging, etc.

    thrashing warnings

    memory parity errors for chip table

. What happens if MCP just waits for an I/O instead of requesting an interrupt?

  just mount a brand new tape and see!

ODT QUEUE EVENT

SYSTEM/ODT handles all ODT communication,
manages ODT queue, SYSTEM/ODTLOG, etc.

Broken out from MCP in 10.0 to relieve
MCP of a tedious, I/O-bound function

Communicates with MCP via two ordinary
queue files, one in each direction

This event is merely the READ-OK Boolean
for the queue from SYSTEM/ODT ---> MCP.

When TRUE, MCP does a read of the message
and passes the text to the command processor.
Commands will be scanned, parsed, and
acted upon.

COMMUNICATE EVENT

- Set by GISMO whenever some job(s) have done a "COMMUNICATE" to be handled by the SMCP. (QUEUE_ID = S_COMM_Q)

- The first 48 bits of one's ESN (ENVIRONMENT) tell the MCP what class of communicate is requested and point to the actual string of relevant fields.

```
      CLASSES:  (2 bits)
        00 =>  INTERPRETER GENERATED BEHIND USER'S BACK
        01 =>  THE USUAL SERVICES REQUESTED BY
               SOURCE LEVEL CONSTRUCTS
        10 =>  UNDEFINED; DS'es THE JOB
        11 =>  USED TO CLOSE FILES THAT ARE STILL OPEN
               WHEN A JOB GOES EOJ
```

INTERPRETER-GENERATED COMMUNICATES

Most common: PRESENCE FAULTS

User branching to some other code segment which is not in memory. MCP has to read the segment from disk, mark code segment dictionary appropriately, and reinstate the program

Same for data segments

Likewise for segments of the interpreter itself!

Trace to printer- the old trace interpreters

Certain problems: memory errors,
read out of bounds, etc.

"REAL" COMMUNICATES

- Usually generated by the compilers in response to various source-level constructs that end up requiring operating system services.

- Most I/O-related COMMS are handled directly by Gismo or MMCP. A mask set up at CLEAR/START tells who is to handle each of the 80 known communicates.

- MCP never sees READ, WRITE, SEEK, POSITION, MESSAGE-COUNT, various ISAM operations, etc.

- MCP handles such common things as:
  OPEN, CLOSE, PROGRAM DUMP, TIME/DATE, ZIP, ACCEPT, DISPLAY, SORT CALLS, and TERMINATING A PROGRAM.

- MCP also provides services to system programs
  - Accesses various system & program structures
    Disk Directory, FPBs, FIBs, memory itself, etc,.

  - Lets system programs change system data normally inaccessible (outside their own data area)

    Example: PACKCOPY must stop and resume all jobs
    SYSTEM/ODT must control the ODTL option
    DMS/REORG can restart jobs once the data-base is fixed up.
    etc.

"REAL" COMMUNICATES (continued)

How are communicates done:

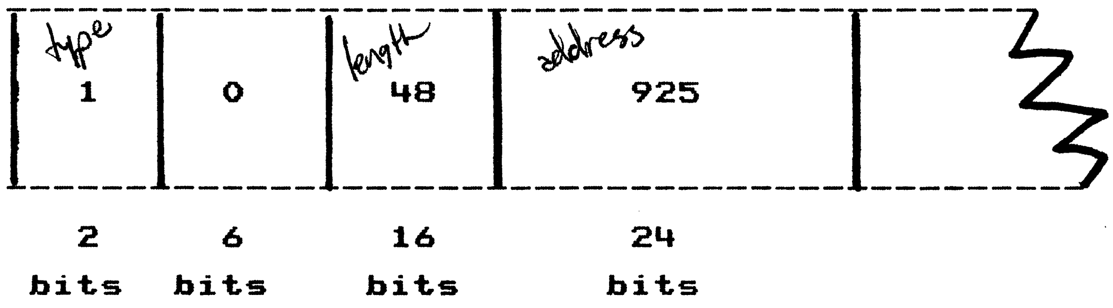
1. Compiler generates appropriate bit string somewhere in program's memory area

2. Special "COMMUNICATE" S-OP in each language causes that previously-mentioned 48-bit field to point to the data string, and control is passed to GISMO

Example: "CLOSE PAYROLL-FILE LOCK."

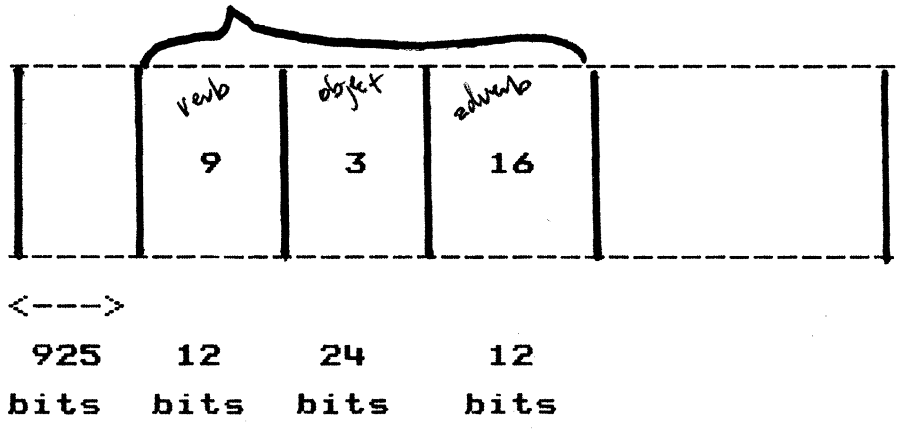
Suppose PAYROLL-FILE happens to be the 4TH file the user declared.

ESN: (base relative)



User's data area: 48 bits

happens to be @F311A6@



"REAL" COMMUNICATES (continued)

Why this string? Because the documented form for this communicate that everyone must obey, reads:

<u>CLOSE</u>

```
CT.VERB      09                              (12 bits)
CT.OBJECT    FILE.NUMBER                     (24 bits)
CT.ADVERB    BIT                             (12 bits)
             0      REEL
             1      RELEASE
             2      PURGE
             3      REMOVE
             4      CRUNCH
             5      NO REWIND
             6      OVERRIDE NAME CONVENTION AND SECURITY
             7      LOCK
             8      IF NOT CLOSED
             9      ROLLOUT
             10     AUDIT SWITCH
             11     TERMINATE
```

Let's make sure this communicate is set up correctly - Apply the documented format to the example on the previous page.

ESN

pointer to comm info	result
48	48 bits

type	length	address or actual data
8	16	24

type: tells amoung other things whether the result is in the 2nd 24 bits ("self relative") or whether the 2nd 24 bits points to the actual info somewhere in the user's data area

length: length of result in bits

2nd 24 bits: a self relative number if the info being returned can be expressed as a simple number.
Example: "ZIP" described above - how many characters?
or a relative address of longer data
Example: TIME/DATE to request name of day.
we surely can't fit "MONDAY" in 24 bits!

COMMUNICATES (continued)

Some communicates need only a verb:

Verb = 32 => program goes EOJ with "COBOL ABNORMAL END"

Verb = 38 => memory dump

Verb = 39 => give me my session #

Others have lots of necessary fields

Verb = 2 => write

Object = file #

Adverb: broken down to numerous sub fields:

1. does user have an EOF branch?
2. does user have an exception branch?
3. does user have an incomplete-IO branch?
4. printer spacing/skipping info
5. MFCU card stacking info, etc, etc.

Additional fields for

record length - how much is to be written;
address of info to be written;
key for random files;
address of 10-character key for remote/port.
length and address of "result mask" info
(for system programs handling their own
I/O exceptions);
Linage fields for printers -
page size, upper margin, lower margin,
footing, etc.

COMMUNICATES (continued)

For certain communicates, the MCP doesn't need to talk back to the program.

examples: dump
read the reader-sorter
sort

But in many cases, the MCP wants to tell what happened or return requested data.

1. What happened?
   - write: good write?
     - EOF ?
     - I/O error ?
     - incomplete?
   - Accept: how many characters did the operator AX to the program?
   - ZIP: Was the zipped text valid?
2. RETURN DATA
   - TIME: What time or date is it?
   - Complex-wait: which of n events woke up the program?
   - message count: how many messages are in my queue?

Another field in the ESN is filled in to indicate this info:

THE CHANGE BIT EVENT

The change bit is set by the MCP itself when there is at least one scheduled task that is a candidate for BOJ

"Old" MCP'S (12.0 and before): A job is in the active schedule

13.0: A job is in at least one job queue or the tasking schedule

. MCP, upon waking up on the change bit event, turns it off and tries to fire up all jobs:
  - Every job in the tasking schedule
  - Goes through each job queue, initiating jobs as long as system mixlimit ("ML") and individual job queue mixlimit permit

. The change bit is set when:
  1. A job is scheduled, E.G. EX DMPALL;
  2. When operator does a "TR" or "DR";
  3. Any job goes to EOJ;
  4. When memory is freed by rolling out a job;
  5. If the first job in any job queue is RS-ed and the queue is still non-empty;
  6. "ML" is increased (system ML or a job queue ML);
  7. Some job has its priority increased above 8;
  8. Operator does a "JS."

PROGRAM INITIATION

Two steps:

1. Scheduling - building necessary structure from the code file.
2. BOJ - creating memory-resident job image that is runnable

Code files contain three structures of interest

- Program Parameter Block (PPB)
- File Parameter Block (FPB)
- Code segment dictionary

PPB:

- First 2 sectors of every code file. Created by compiler
- Contains in a rigidly defined format:
  - a. Everything the operator can change or query about a job.
    EXAMPLES: NAME, PRIORITY, MEM requirements, # of files, switches, max lines, NODIF bit, interpreter name.
  - b. Pointers to "interesting" fields in the code file.
    EXAMPLES: First instruction, DMS path dictionary, files, layout table.
  - c. A level number to allow MCP to handle structures of different vintages.

- At schedule time, we build an expanded four-sector PPB
- First 2 sectors: exact copy of code file copy
- 3rd sector: run-time data not in code file
  EXAMPLES: Job number, session #, parent job #, sched. & BOJ dates, datacomm pointers, link to next PPB.
- 4th sector: Stores "DS or DP" message for logging and or dump analyzer
- Linked into appropriate place in given job queue or tasking schedule.

FPB

- Two segments of disk; actually 2096 bits
- One per declared file
- Extras automatically created by compiler for ISAM subfiles.
- Contains all file attributes specifiable by either actual attribute or by command syntax.
  EXAMPLE: Internal & external names, hardware device, record size, # of areas, filetype, buffers, etc.
- When job is scheduled, MCP makes copy of all FPB's, plus one for the trace file.

SEGMENT DICTIONARY:

- Most "virtual"- segmented, overlayable - structures on the B1000 managed by dictionaries.
- Dictionary = A contiguous array of "system descriptors".
- Code segment dictionary on disk: an array of relative, non-absolutized "normal descriptors."
- BOJ brings in and absolutizes ND's into SD's.
- Next page discusses SD's.
- Data segments must be overlayed to disk; code segments are non-changeable and thus do not get written back to disk.

PROGRAM INITIATION (continued)

BOJ is largely the process of getting space for and filling in fields in two structures, the "RUN STRUCTURE" and the "PRIMARY ENVIRONMENT."

The Run Structure contains information relevant to the state of the job taken as a whole.

- Overlay descriptor
- FIB dictionary
- IPC parameters
- Most interestingly, the "RUN STRUCTURE NUCLEUS" (RSN).

RSN's are linked together by descending processor priority.

RSN's frozen in memory; never rolled out or moved.

Nearly 200 fields related to the job itself, for example:
NAME, JOB #, SWITCHES, PRIORITY, CHARGE #, etc.

Many flags, pointers, and indicators to help the MCP understand the state of this job.
EXAMPLE: Invisibility flag, stopped flag, protected flag, cancelled flag, etc.

Pointers to: next RSN, AX queue, DMS globals, overlay descriptor, etc.

Event bits so GISMO can cause the job when certain things happen.

Note that the Run Structure has no user data.

SYSTEM DESCRIPTORS (a digression)

Eighty Bit Structures

The most important fields:

- Media bit: Indicates if the info is on disk
- In-process bit: Indicates that at this instant we're doing an I/O on the segment (either direction)
- Length field
- MPRI decay info.
- address.

If a segment is on disk:

MEDIA = FALSE

address = absolute disk address

If a segment is in memory:

MEDIA = TRUE

address = memory address of the info.

The information is preceeded by a memory link, of course.

Within that link:

ML_DISK is the disk address from the SD.

ML_POINTER points to the SD.

PROGRAM INITIATION (continued)

The other primary structure is the "environment".

Environment is basically the data space for a program.
Broken away from RSN in 11.0
All programs initially have one environment, termed
the "primary" environment.
Secondary environment exists for DMS and IBASIC.
IBASIC environment hides the existence of
IBASIC/RUNNER.
DMS environment allows complex DMS operations to be
interruptable at any point, because the state of
the job is stored in the "ESN" (Environment
Structure Nucleus).
Also permits DMS to execute on the slave; it is
not an MCP.

Environment concept: a separate code file (e.g.
DMS access routines), with a possibly-different
interpreter and local data space, is associated
with a user job to handle certain communicates.

Environment switching is quite efficient.
Secondary environment operates at same priority as
the user.
RSN has dictionary of all environments that have
been allocated, as well as pointer to the current
environment.
Max of four environments.

Environment consists of:

- local data area, defined differently for each
  language.
- "scratchpad" - used to save interpreter state
  upon giving up control.
- ESN.

ESN

Various pointers, counters, and flags of interest
to a program's interpreter.

Examples: communicate message pointer and reinstate
message pointer (previously discussed).
next instruction pointer.
code segment dictionary pointer.
data segment dictionary pointer.
associated RSN address.
amount of overlay disk and its address.
MEDIA Boolean - environment can be rolled
out.
rollout disk address.

Building an environment involves several steps:

- Reading initial scratchpad values from the PPB and storing them in the ESN

- Setting up the code segment dictionary if it does not yet exist (from another running copy of the same program)

- Handling initial data
  1) read in from code file if there's a data dictionary.
  2) build links for COBOL overlayable data in dynamic nenory. Note: SDL/SDL2 does this itself via intrinsics.

- Get rollout disk on system disk. If cannot be obtained, program is frozen.

- Set up interpreter dictionary if the interpreter is not currently in use by some active job.

FILE OPEN

Builds several structures in memory

- FIB (File Information Block)
- I/O descriptors
- I/O buffer

Specific peripheral assignment.

FIB contains all information needed for the operating system to do I/O's to/from the file.

Much information extracted from the FPB. Altered as necessary by the MCP.

Example: MCP would change blocking factor if

a) blocksize not integral multiple of record size (truncated)

b) to ensure backup print files blocked as necessary.

Additional fields needed to manage I/O

Example: current key, current access method for disk linage for printer files, lsn(s) for remote file, ISAM flags and table pointers.

Somewhat unusual in that the structure varies widely depending on the hardware device. Explanation: there's a lot more to be known for a disk file than a card reader.

I/O descriptors - buffer memory area pairs (one per BUFFERS value) allocated in one continuous chunk of memory. (Makes rollin and rollout easier)

FILE OPEN

OPEN EXAMPLE: opening a disk file

Why? 1. Explicit OPEN communicate generated by user's OPEN statement

2. Read or Write communicate for closed file GISMO puts job in S_COMM_QUEUE, sets RS_M_PROB_PARAMETER to a value meaning "file not yet open"

- MCP reads communicate from environment
- Calls GLOBAL_OPENERS
- Reads FPB from disk
- GENERAL_OPEN_VERIFICATION to check that various attributes are compatible.
  Example: blocksize = 0 or > 2**20 bits
  blank name
  relative non-disk file
- Certain hardware devices have unique open code (for speed)
  DISK is one of them. We call DISK_OPEN
- Reads FIB if present
- Initial disk open verifications
  arealength zero
  open input/extend
  ISAM record size < 4 bytes
- Fixup a few things for implied opens
  Example: If new, make sure it is output
- If no FIB, call ASSIGN_UNIT
  - If this is a backup file, change hardware to DISK
  - Build name, e.g. BACKUP/PRT1234
  - Usercode naming transformations
  - Security checks, e.g. non-existent usercode for MFID

FILE OPEN (continue)

- Various checks based on type
  - multi-pack duplicates
  - codefile: areas = 1 and external name from PPB.OBJ_NAME
  - create an initialized disk file header
- If there was a FIB:
  - Find the file
  - If absent, see if name transformation possible
  - If absent, either hang up the job for operator action or return "file missing" if requested by programmer
  - If present, get the header and check accessibility and security contraints
- If "old" file, change access date, user count, and several FPB fields
- If multi-pack file, create MPF table entry
- Get memory for FIB and update user's FIB dictionary
- BUILD_FIB sets up fields in FIB and builds I/O descriptors
- If input, initiate I/O's to prefill buffer(s)
- If backup, create backup file control info records
- Reinstate job into READY_QUEUE

THE DISK DIRECTORY

- Directory for each pack resides on that pack.
- Initially 16 sectors, 32 thru 47 (@20@ - @2F@).
- Each of those sectors can hold 12 names (MFID's).
- MFID'S "HASH" to one of those sectors.
- If the sector fills up with names, it is linked to an extension sector somewhere on disk.
- The directory entry points to either:
  1. The disk file header for single-named files
  2. A subdirectory (of identical format to the directory) for 2-name files.
- Subdirectories are not hashed - they are searched linearly. Their entries point to the DFH's.

- A word to the wise: Don't put "too many" names under one MFID due to the linear search. This includes usercode MFID's- WATCH IT!

```
THE DISK DIRECTORY (continued)

EXAMPLE: Find the file BOZO/WASHINGTON/SHOWITALL
  BOZO happens to be port 7, channel 10, unit 1
  so all addresses had better start with @F41@

WHY?  Disk address fields consist of a 12-bit
   PORT-CHANNEL-UNIT plus a 24-bit sector address.
   The "PCU" consists of:
       3 BIT PORT FIELD
       4 BIT CHANNEL FIELD
       1 BIT FILLER OF ZERO
       4 BIT UNIT FIELD
      ---
      12 BITS

    PORT 7:     7  = 111 in binary
CHANNEL 10:     10 = 1010 in binary
    UNIT 1:      1 = 0001 in binary

CONCATENATING  111 + 1010 + 0 + 0001
             = 111101000001 = @F41@

We hash the MFID of "WASHINGTON".  The hashing
algorithm happens to be simply dividing the
   10 character = 80 bit field into 20 4-bit
   fields, and exclusive-ORing them
"WASHINGTON" = E6 C1 E2 C8 C9 D5 C7 E3 D6 D5
```

DISK DIRECTORY (continued)

```
@E@ EXOR @6@ = @8@
@8@ EXOR @C@ = @4@
@4@ EXOR @1@ = @5@
@5@ EXOR @E@ = @B@
@B@ EXOR @2@ = @9@
@9@ EXOR @C@ = @5@
@5@ EXOR @8@ = @D@
@D@ EXOR @C@ = @1@
@1@ EXOR @9@ = @8@
@8@ EXOR @D@ = @5@
@5@ EXOR @5@ = @0@
@0@ EXOR @C@ = @C@
@C@ EXOR @7@ = @B@
@B@ EXOR @E@ = @5@
@5@ EXOR @3@ = @6@
@6@ EXOR @D@ = @B@
@B@ EXOR @6@ = @D@
@D@ EXOR @D@ = @0@
@0@ EXOR @5@ = @5@
```

From now on, we shall work entirely in hex. The disk directory goes from @20@ to @2F@. Adding our hash result of @5@ to the directory lower limit, we get @25@.

Let's look at sector @25@ on that pack:

```
@F41000025@  "4? ?&    4? ?& NEW_REL   4? ABC74ISAM   4? F?SDL2_OBJ  4? ?K"
             "MAKFILE13 4? ??SF12      4? &SCOBOOT13  4? ?BPRT13     4? ?B"
             "RLIPANDA  4? ??MMCP1313  4? T?TEST      4? !?MCPOBJ    4? ?S"
```

```
             Forward   |Backward  |  SELF  |X|
@F41000025@  F41000345|D 00000000|F4 1000025|000| D5C5E66DD9 C5D3404040 F410008182
             C3F7F4C9E2 C1D4404040 F410008622 E2C4D3F26D D6C2D14040 F41000580D2
             D4C1D2C6C9 D3C5F1F340 F410001622 E2C6F1F240 4040404040 F4100050E2
             C3D6C2D6D6 E3F1F34040 F4100031C2 D7D9E3F1F3 4040404040 F4100055C2
             D9D3C9D7C1 D5C4C14040 F410007352 D4D4C3D7F1 F3F1F34040 F41000A322
             E3C5E2E340 4040404040 F410004F22 D4C3D7D6C2 D140404040 F4100006A2
```

We don't find WASHINGTON; let's go to the forward link @F41000345@

DISK DIRECTORY (continued)

```
@F41000345@  "    ??  ?4? ?& GISM01303 4? ?SINIT1302  4? ??GISM01312 4? ??"
              "MMCP1320  4? BBOOT1304  4? CKSDL2_NEW  4? ZBMMCP1302  4? ??"
             "WASHINGTON4? ?K                                            "
```

```
                F         B         S
@F41000345@   000000000|F 41000025|F4 1000345|000 C7C9E2D4D6 F1F3F0F340 F4100059A2
              C9D5C9E3F1 F3F0F24040 F4100058B2 C7C9E2D4D6 F1F3F1F240 F410002D42
              D4D4C3D7F1 F3F2F04040 F410001F82 C2D6D6E3F1 F3F0F44040 F4100083D2
              E2C4D3F26D D5C5E64040 F41000E9C2 D4D4C3D7F1 F3F0F24040 F4100013B2
              E6C1E2C8C9 D5C7E3D6D5 F41000AA92 0000000000 0000000000 0000000000
              0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
```

Note that the backward & self pointers are correct; The forward pointer of zero implies this is the end of the chain.

Happily, we find "WASHINGTON".
Our format for a directory entry:

10 char NAME
36 bit ADDRESS
4 bit KEY: 2 = subdirectory
0 = DFH

The key is 2, meaning that the address of @F41000AA9@ points to a subdirdctory, i.e. a list of FILE-ID's with the common MFID of "WASHINGTON"

DISK DIRECTORY (continued)

The subdirectory is chained as was the main directory. Here is sector @AA9@:

```
@F41000AA9@   "4???^? ??4? ?? BALANCER  4? ?&SEQCHECK  4? ?&S.PANDA   4? ??"
              "S.SEQCHECK4? ?\S.SUX     4???\P.SORT    4???0S.REMYAKFI4???&"
              "GOODRESULT4????DESIGN    4????S.FRUIT   4????S.COMBINE 4????"
```

```
@F41000AA9@   F41001165F 41000345F4 1000AA9000 C2C1D3C1D5 C3C5D94040 F410005950
              E2C5D8C3C8 C5C3D24040 F41000AB50 E24BD7C1D5 C4C1404040 F41000FEB0
              E24BE2C5D8 C3C8C5C3D2 F41000FEE0 E24BE2E4E7 4040404040 F4100105E0
              D74BE2D6D9 E340404040 F4100108F0 E24BD9C5D4 E8C1D2C6C9 F410010A50
              C7D6D6C4D9 C5E2E4D3E3 F410010F30 C4C5E2C9C7 D540404040 F410011080
              E24BC6D9E4 C9E3404040 F410011210 E24BC3D6D4 C2C9D5C540 F410011520
```

No sign of "SHOWITALL" - link forward to sector @1165@:

```
@F41001165@  "4??L?? ?Z4???& S.GEARS6  4???0S.RECLIST 4???0B61875    4???-"
             "F12841    4????DLTWELVE  4????JOBQUEUES 4????RESPONSE  4??? "
             "OLDFRUIT  4???-STATUS    4?? {MEMO      4??[?STATUS0   4??[ "
```

```
@F41001165@   F410014A8F 41000AA9F4 1001165000 E24BC7C5C1 D9E2F64040 F4100115F0
              E24BD9C5C3 D3C9E2E340 F4100116F0 C2F6F1F8F7 F540404040 F410012660
              C6F1F2F8F4 F140404040 F410012670 C4D3E3E6C5 D3E5C54040 F410013B10
              D1D6C2D8E4 C5E4C5E240 F410013B30 D9C5E2D7D6 D5E2C54040 F410013B40
              D6D3C4C6D9 E4C9E34040 F410013B60 E2E3C1E3E4 E240404040 F4100140C0
              D4C5D4D640 4040404040 F410014A30 E2E3C1E3E4 E2D6404040 F410014A40
```

Still no luck - chain to @14A8@:

DISK DIRECTORY (continued)

```
@F410014A8@   F41001566F 41001165F4 10014A8000 D9C1C9C46D D7D9D6C240 F410014A60
              E24BE6D6D9 E3C8404040 F410014A90 C1E2E2C5E3 E240404040 F410014DD0
              D74BC8C5E7 4040404040 F410014DE0 7BE24BC6C1 C9D3E4D9C5 F410014E00
              E2C8D6E6C9 E3C1D3D340 F410014E20 C6F1F2F8F0 F640404040 F410015180
              C6F1F2F5F9 F940404040 F410015190 E24BF1F540 4040404040 F4100151C0
              C7C5C1D9E2 4040404040 F410015610 D9C5E2C9C4 C5D5E34040 F410017080
```

```
@F410014A8@  "4????? ??4??[? RAID_PROB 4??[-S.WORTH   4??[?ASSETS    4??()"
             "P.HEX     4??(\#S.FAILURE4??+ SHOWITALL 4??+?F12806     4????"
             "F12599    4????S.15       4???{GEARS     4????RESIDENT  4????"
```

EUREKA! We have TYPE = 0, meaning DFH at @F410014E2@

```
@F410014E2@   021C49F410 014E200000 1000240000 2D01000140 0000100000 A001001000
              0020000000 000000AD1B 06EF720000 0000001AC9 DACBF00000 0000000000
              0000000000 00000F4100 5641000000 0000000000 0000000000 0000000000
              0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
              0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
              0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
```

This is recognizeable as a header. Encoded therein are things you see on a KA - EOF, record size, # of areas, file type, blocking, etc.

540 bits into it is the address of the first (and only) area of actual data : @F41005641@

```
@F41005641@  "   IF CSV.SWE_OPTION COR % FOR ART'S DEBUGGING USE          "
             "             83909550                  NOT ((ZIP_RSN.RS_WFL_TAS"
             "K CAND ZIP_RSN.RS_TASK_NUMBER = 0)          83909600C0SPB61827"
```

Yep! That's my data