EVALUATION OF WORK YET TO BE DONE ON THE ECS SYSTEM AS OF 30 MARCH 70

STUFF NEEDED FOR THE OPERATION OF THE DISK SYSTEM

1) Allocation mod 64 for DAE map entrues (Vance)
2) Compactifier (Vance)
3) Change to mover block operation (Paul)
   a) Check map reference count
   b) Return dirty bit in X6
4) Change probe operation to return # of map refs in X7 (Paul) — cancelled later
5) New operation to turn off/on map entries for a subprocess (?)
6) Implementation of two new parameter types, block parameters and return parameters
7) Indirect C-list (Bruce) (Bruce)
8)
9) Return capability of specified type

Stuff IMPORTANT TO THE OPERATION OF THE ECS SYSTEM

1) Find descendent of subprocess (Dave)
2) Change map compiler to do F-return in case of missing map block instead of DISASTER (Paul & Bruce)
3) Change to change unique name operation vis-a-vis option bits — temporarily, do without change UN.
4) Get more system code out of central and into ECS (Vance)

STUFF WHICH WILL BE NICE WHEN IT GETS DONE, IF IT EVER DOES

1) Set temporary part of class code
2) Put check in PUTACT and PUTECS for length of ACTIONL
3) Implement accounting of CPU time
4) Reset end of path to self
5) Get the option bits into the operations (Vance)
6) Fast actions
7) Implement the error return operation
8) Fix up CCGLOA (what does this mean?)
9) General destroy operation
10) Send interrupt to pseudo-process (Howard, is this still needed?)
11) Move GLASSCNT to ECS
12) Check GARBCNT in subprocess environment establishment at the point of doing the direct access map entry
13) Fix up the 0-level file name hassle (Paul) — done
14) Fix error returns from OPINTR
15) F-return when subprocess to be deleted is not a leaf (Bruce)
16) In process and subprocess creation, correct test of lower limit for entry point
17) Design and implement display process descriptor operation
18) Incremental map compilation (Paul)

STUFF ON WHICH THERE WAS NO IMMEDIATE CONSENSUS

1) Provide date and real time
2) Move from one allocation block to another
3) Move an allocation block to another allocation block
4) What about ~~data~~ message channels?

Disagreements as to the above classifications will be cheerfully discussed. [illegible] People indicated as being somehow responsible for performing changes may try to wriggle out of it (volunteers for [illegible] unassigned projects will be courteously received).

I left the meeting without and understanding of how the file block dirty bit was to perform [illegible] its function. It was supposed to be maintained by the ECS system and somehow save the disk system the trouble of writing out blocks from read-write files unless they had actually been altered. Exactly what is the prepesal?

ALLOCATION BLOCKS

Much thinking has been going into allocation blocks, ECS space accounting, and CPU time accounting. Here is a semi-solid proposal.

1) CPU time should be taken out of allocation blocks and put , probably, into the process descripter. Several reasons
   a) AB's are really to control ECS usage and the current CPU time stuff is just a hopeful, incompletely evaluated after-thought
   b) If a process is allowed to run at different weights, the time has to be accumulated separately for the different weights (and you don't want to keep a weight in the AB)
2) CPU time should be counted down. When a process [illegible] is swapped in, if it has no time in the slot currently being charged, an error is generated and either
   a) if there's more than one pool of CPU time, control is switched to another pool to cover the processing. If the last pool runs out, it's an error error or the equivalent, and the process is shut down, perhaps destroyed.
   b) if there's only one pool of CPU time, the process is loaned epsilon time by the swapper and marked bankrupt. If it's already bankrupt, error error. The system operation which puts money in the CPU time pool clears the bankrupt signal. [illegible]

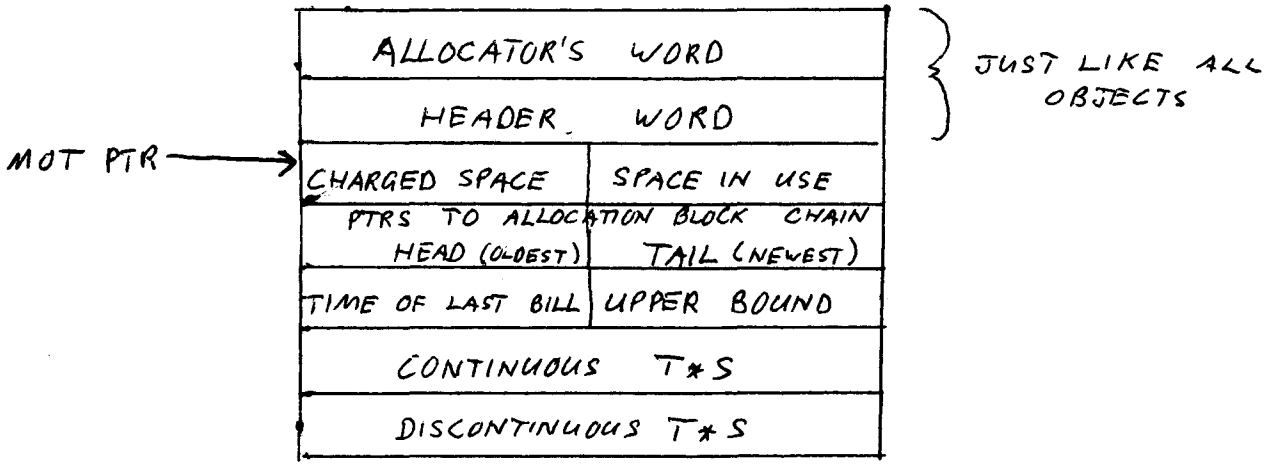
   In both methods, it is anticipated that the initial error will be intercepted by somebody competent to straighten things out, like a very priveleged system accounting subprocess. If the user intercepts the error in one of his own subprocesses and blows it, he gets had.
3) ECS space accounting in AB's is to be changed to charge for the amount of ECS that the AB has tied up, not the amount that it happens to be using. The latest model allocation block will contain 3 space parameters, 2 time-[illegible]space integrals, and the invisible time of last bill field. (See fig. 793-42B.03f)
   a) UPPER BOUND - can be set arbitrarily and doesn't reflect any real memory anywhere. It is used to control somebody you don't trust.
   b) CHARGED SPACE - this is available to the AB on demand and is the amount charged for. Space [illegible] added to this field comes from its father AB and increases may fail for lack of space in the father or for exceeding the local limit.
   c) SPACE IN USE - space currently in use, may not exceed charged space or an error is generated.

   Nice guys and poor guys will try to keep charged space down around space in use; rich guys may keep a lot of charged space in case they might need it. [illegible] Meaningful increases to charged space will presumably entail a call on a priveleged system routine to get space from a system pool, and delays may result 'cause space isn't available.
   d) CONTINUOUS T*S - starts at 0 when the AB is created and builds up continuously. Facility to display it will be provided.
   e) DISCONTINUOUS T*S - I don't like this field for reasons explained below. When it is displayed, it is reset to 0.

   A DAEMON process runs periodically and touched the AB's, to prevent deficit spending. Bruce wants to use the discontinuous field and charge the guy right away, so that if the system crashes, he stands charged for some T*S, which he

may or may not have derived any benefit from. The guy will almost certainly complain bitterly. I think that the continuous field should be used by the DAEMON to check against deficit spending, but that the DAEMON should do nothing in the normal case, leaving the log-off procedure to do all the actual charging.

FIGURE 793-42B.03f



If the time of last bill is kept in units of micro-seconds/1024, 30 bits allows about 16 days of running. If this is deemed insufficient, speak now. More bits may be used or the units can be changed.

ALLOCATION BLOCK OPERATIONS

A) Create allocation block (no change)
IP1 C: father AB (OB.CREAB)
IP2 D: C-list index for returned capability

B) Transfer charged space
IP1 C: Donor AB (OB.GIVE)
IP2 C: Donee AB (OB.GET)
IP3 D: Space to be transfered, or donation
fails if CHARGED SPACE+DONATION exceeds UPPER BOUND in donee
or DONATION exceeds CHARGED SPACE-SPACE IN USE in donor

C) Set upper bound
IP1 C: AB (new option bit)
IP2 D: new upper bound
fails (or F-returns) if new upper bound less than charged space

D) Read discontinuous T*S
IP1 C: AB (new option bit')
IP2 D: where T*S is returned (or return it in X6?)
resets discontinuous T*S to 0 and returns updated value

E) Display AB
IP1 C: AB
IP2 D: buffer
updates both versions of T*S, doesn't reset discontinuous T*S

F) Return capability for nth object on the AB (no change)
IP1 C: AB (OB.GOD)
IP2 D: full C-list index for returned capability
IP3 D: number of desired object (n)

G) Destroy AB (no change)
IP1 C: AB (OB.DSTRY)

DIRTY BITS

In order to save the disk system some unnecessary writes, it was decided to provide a dirty bit on file blocks which would enable the disk system to tell whether or not a block had to be copied back out to the disk. The final specs were:

1) File blocks are created clean
2) Blocks are dirtied by
   a) File writes, including ones with words counts of 0
   b) Being put in a map RW
   c) Being put in a direct-access map entry
3) An operation to test and reset the dirty bit will be provided.
4) Move block carries the dirty bit along with the block.

With this machinery, it is claimed that blocks from a file opened RW will not have to be written out to the disk if

1) Somebody just scans through the file and doesn't actually write in some of the blocks
2) A block hasn't been written in since it was last written out, due to a pseudo-close or somesuch mechanism

Just for some concrete examples.

DELIVERY OF INTERRUPT DATUM

It is proposed to alter the location where the interrupt datum is delivered from IPO (cell 6 of the subprocess) to cell 2 of the subprocess. Current delivery clobbers the first input parameter. Any objections?

CHANGE TO CHANGE UNIQUE NAME OPERATION

It is proposed that CUN be altered to have 2 parameters:

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX(XXXXXXXXX)X
XXXXXXXXX
IP1   C: capability for object (OB.CHNAM)
IP2   D: C-list index for return of new capability.

This is a funny thing from the point of view of the user, since the old capability becomes no good after the operation, but it allows the system to do its option bit testing in the usual place instead of in the CUN code.

CHANGE UNIQUE NAME AND MISSING MAP BLOCKS

A block refered to in a map may be caused to disappear by the use of the change unique name operation. The question is, what should the map machinery do when it encounters a map entry with miss ing blocks? The only answer se ms to be that the offending map entry should be zeroed and error processing should be initiated. This is unpleasant, as the er or is go ing to be discovered in the swapper, but it seems like there is no alternative. How abo t it?

ALLOCATION

Work on the allocator (initially undertaken to write a compactifier) has revealed certain problems:

1) The documentation is scanty and not overly helpful. For example, the purpose for the two 0-length free blocks isn't mentioned, how compactification is to be (incrementally) achieved is left as an exercise, etc.

2) There are bugs
   a) Free blocks are merged without due regard for limitations on their size
   b) Interrupt objects are scattered through core in such a way as to make keeping them fixed during compactification a somewhat bewildering problem
   c) There are miscellaneous quirks in the initialization.

3) Objects are limited to 2**17 - 1. This limits DAE's to
   2**17 - epsilon for 0-level files
   2**16 for other level files

4) The top and bottom of ECS are both fixed by various factors. This makes it difficult to dynamically change the size of ECS.

It's easy enough to fix the bugs and improve the documentation. And the top of ECS can be freed by various ploys which can be simple and inefficient or medium difficult and as efficient as at present. The stopper is item 3. Extensive rewriting will give a factor of 4; extensive rewriting plus an additional word or redsign of the allocation chain are necessary to completely unrestrict object sizes.

It is roughly true that the redesign and changes necessary to deal with 3 and 4 are internal to the allocator and can be redone later without affecting other code (the main possible exception is the file code, which shares one of the allocator's words). I feel that it is somewhat a matter of style as to whether we fix these things now or later, but I would like to have some commitment on item 3 right away.

INCREMENTAL COMPACTING

It is deemed desireable that the compactifier should be designed in such a way that some process may run while compactification is in progress. Namely, a speed freak shouldn't have to wait for compactification to complete before running. There seem to be two different schemes which allow suspension of compacting in mid-stream;

1) To tell the compactifier in advance only to do so much and then to check for speed freaks when it returns. You could tell it to collect n objects ofr example. But you have to understand that it may get into something big that it has to finish.

2) To have a flag which the compactifier looks at which tells it to stop as soon as possible. I prefer this, as it is more efficient. There is still a limit to how fast the compactifier can react, but control is better than with 1.