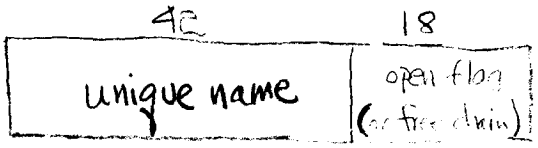
Name Tag System

Dec 10, 1970

1. Global Data Structures

A. Static Name Tag Division

A C-list of length n (where n is [illegible]) and a parallel, single level list file comprise the static name tag table. Words in the file have the following structure:



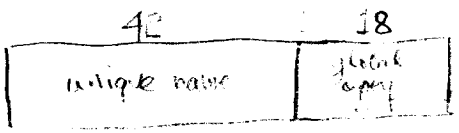
If this entry of the static name tag table (SNT) is in use, the open flag is 1 or 0 according as an object is or is not in the corresponding c-list index. If this entry is not in use, the unique name field contains the next value to use when creating a dynamic name tag with this index, and the lower 18 bits of the file word contain the index of the next free entry.

( )

There is an event channel associated with the SNT. It holds a single word which [illegible] read or write access to the SNT is made.

B. Dynamic Name Tag Division

The dynamic name tag table (DNT) consists of a c-list of length m (where m could be a creation-time parameter) and a parallel ecs file, used as a linear hash table. Words in the file are either 0, indicating an empty DNT entry, or have the following structure:

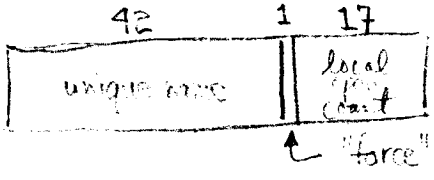


The local open count, always greater than 0, represents the number of processes which have opened the dynamic name tag specified by the unique name field more times than they have closed it.

An event channel is associated with the DNT just as with the SNT, to synchronize access.

2. Local Data Structure (Dynamic name tag division only)

A local open list is maintained in each process, with an entry for every dynamic name tag currently held open. Entry format is:

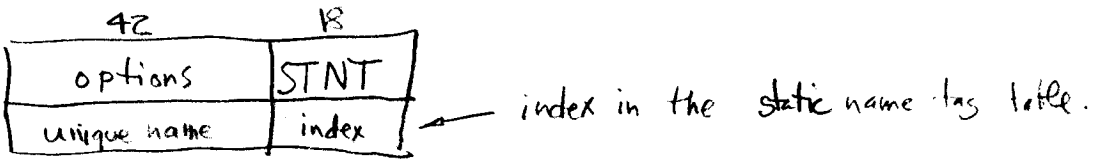


The local open count, always greater than zero, is the number of times the designated tag has been opened by this process minus the number of closes by this process. The "force" flag is described later (section 4I, 4J).

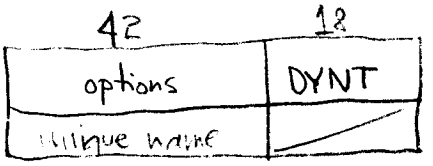
Unused entries in the local open list are maintained on a linked free list. The first implementation will probably not allow the list space size to vary dynamically

3. Capability Formats

A. Static Name Tag:



B. Dynamic Name Tag:



4. Actions

A. Create Static Name Tag []

If the SNT is full, an error results. Otherwise an entry is set up with a new unique name and a zero open flag. Then a capability of type STNT with the new unique name and c-list index as datum is created and returned.

B. Create Dynamic Name Tag []

A capability of type DYNT with a new unique name is created and returned.

C. Open Static Tag with Object [static name tag; any object]

If the static name tag is valid (its unique name matches the one at the SNT entry at the index specified by the static name tag), the given object is placed in the SNT entry, and the open flag of this entry is set to 1 if it is equal to zero.

D. Open Static Tag without Object [static name tag]

If the tag is invalid, an error results. Otherwise, the open flag of the SNT entry is examined. If it is zero, the action fails, otherwise a capability for the object in the SNT entry is returned. (options ANDed with those in the tag)

E. Open Dynamic Tag with Object [dynamic name tag ; arbitrary object]

If the DNT already contains an entry for the given name tag, the action fails. Otherwise, a new entry is made (unless the DNT is full, in which case an error results) for the tag, with a global open count of one and containing the given object. Additionally an entry is made in the local open list, with count = 1.

F. Open Dynamic Tag without Object [dynamic name tag]

If an entry in the local open list for the given tag exists, its open count is incremented by one. Otherwise, the DNT is searched for an entry for this tag. If none exists, the action fails. If one is found, its open count is incremented, and an entry in the local open list with open count of one is created for the tag. The object in the DNT entry is returned, its options ANDed with those in the given name tag.

G. Destroy Static Name Tag [static name tag]

If the given tag is valid, its entry in the SNT is returned to the free list (and the entry's unique name is incremented) making further opens, of either sort, illegal.)

H. Close Dynamic Name Tag [dynamic name tag]

If there is no entry in the local open list for this tag, an error results. Otherwise, the local open count of the entry for this tag is decremented by one. If the result is zero, the entry is returned to the free list and the DNT entry is located. Its open count is decremented by one, and if the result is zero, this entry is returned to the free list and the action fails.

I. Close All Locally Opened Dynamic Name Tags []

A close action, as described in H) above, is performed for each entry in the local open list whose "force" flag is equal to zero. The action as a whole does an freturn if one or more of the tags were removed from the DNT.

J. Set Force Flag on Locally Opened Dynamic Name Tag [dynamic name tag; integer]

If the given dynamic name tag is not in the local open list, an error results. Otherwise the "force" flag in the local entry for this tag is set to 0 or 1 according as the given integer is equal to zero or not equal to zero.

I' Close All Locally Opened Dynamic Name Tags, regardless of Force Flag []

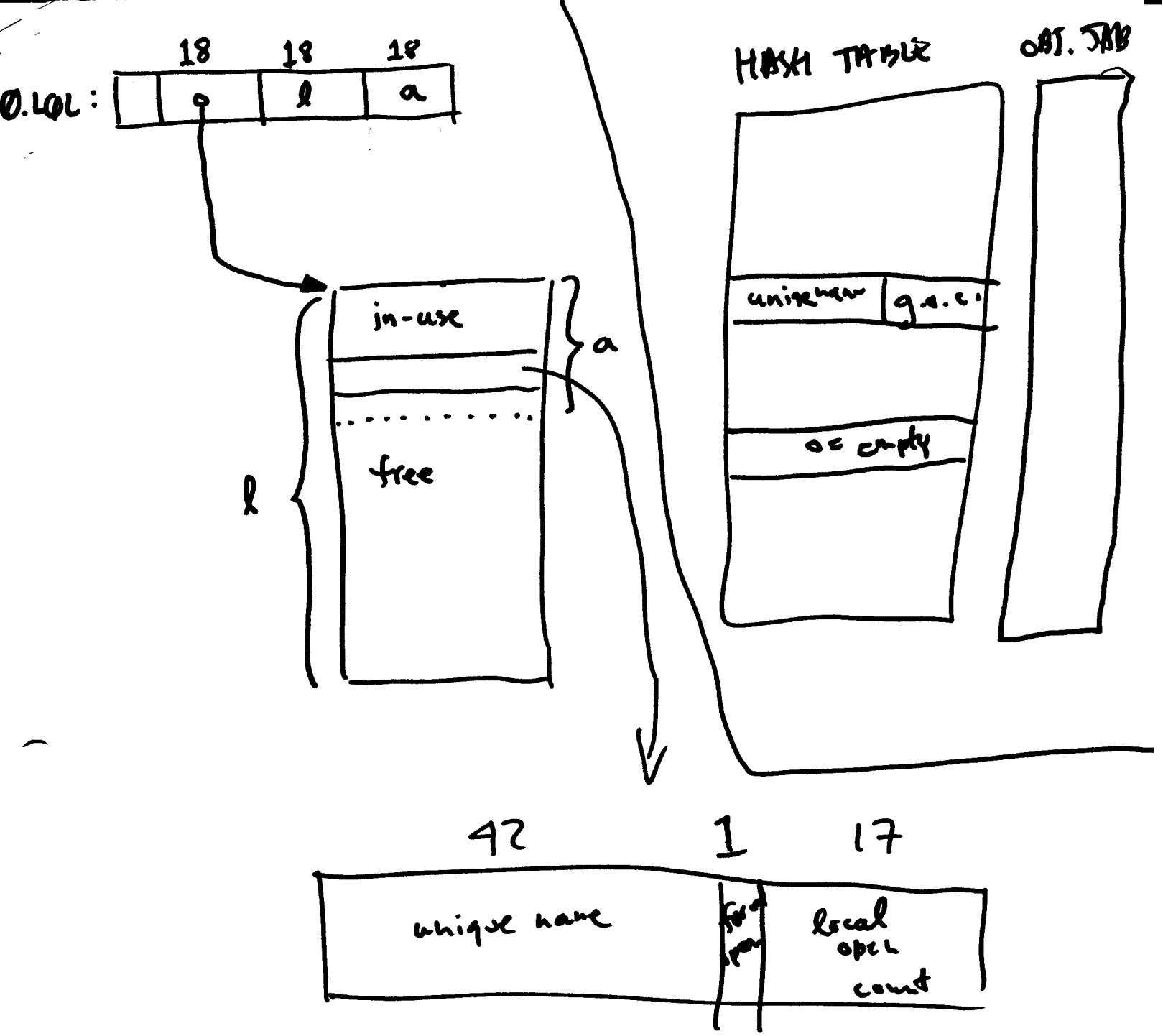
Appendix A

Since the file of the SNT, which also contains three unique name counters, must be preserved from deadstart to deadstart, it must be a disk file, and it is important that the disk copy be a "good" representation of the ccs version of the file. A scheme for insuring that a unique name is never reissued (due to a system crash between updates of the disk file), is to initiate an update of the disk file every time any of the counters becomes equal to zero, modulo 2^n (some small n). Further, whenever the system is started, the counters in the disk file are increased by $k * 2^n$, for suitable k (depending on the likelihood of completion of the last issued update(s) of the disk file.)

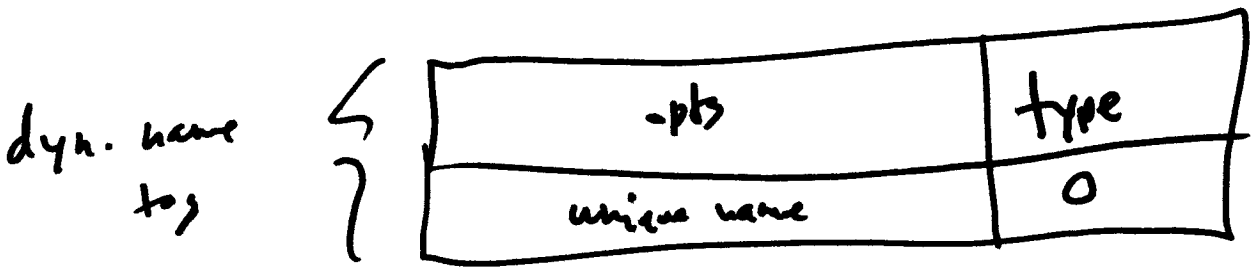
Questions: When a single-level disk file is pseudo-closed, can completion be synchronous? How often will access-keys and name tags be created? Answer – infrequently (mostly at beginning of academic quarters.)

Appendix B

It is proposed that the hash algorithm for the DNT be to take the name tag unique name modulo the table size, use a linear search to resolve collisions, and upon each deletion to rehash linearly downward until the next empty entry is encountered.


18
18
18
O.VOL :
o
l
a
HASH TABLE
OBJ. TABLE
in-use
a
free
l
uniquename
g.a.c.
= empty

42
1
17
unique name
local
count


dyn. name
to
-ptr
type
unique name
0