4/17/70

Directory structure

I) implemented as a low level disk file, 0-level
contains 2 major parts

A) allocation and accounting control

associated with certain directories is something like an ECS allocation block. This data will either be stored in some data words in the first part of the low level disk file, or will be pointed to by a word in 1st part of the low level file. Each directory not having an associated "allocation block" will point to its 1st ancestor that does. Thus each directory will have either a direct or an indirectly associated "allocation block". Any object owned by a directory will have its space requirements paid for by the allocation block either directly or indirectly associated with that directory. this allocation block will be pointed to by the object itself.
(note: the ancestor of a directory is that directory owning the given directory)

example

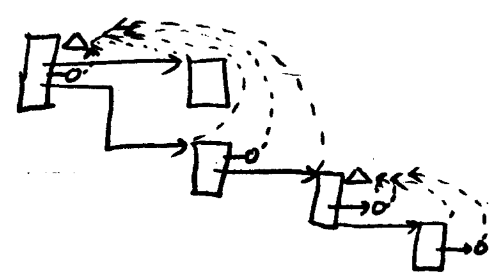
▯ directory

o object (other than directory)

△ allocation block

→ ownership pointer

--→ controlling allocation block



B) directory body

contains names and access control for various objects, i.e. "entries"

II) directory entries

each entry contains a number of parts

A) name

Text name of This entry. The form of The name as well as The algorithm for finding an entry for a given input name are in dispute.

some possibilities are:

1) The name is a sequence of printable characters.
   (There may be a fixed maximum length, or arbitrary length)
   recognition could be by either exact match,(a)
   or The unique entry the 1^{st} part of whose name exactly matches the given input name(b)
   or The 1^{st} entry the 1^{st} part of whose name exactly matches the given input name(c)

examples	entries
Directory 1	alpha 1
	Beta
	alpha

~~Directory 2~~ ~~[illegible]~~

in directory 1 under schemes a), b), c) The following given inputs get The entries as shown

	a)	b)	c)
A	no match	no match	alpha 1
alpha	alpha	no match	alpha1
alpha1	alpha1	alpha1	alpha 1

a better scheme might be the following:

If There is an exact match, take that; if no exact match, if There is a unique entry whose 1st part exactly matchs The given input name, take That; otherwise no match.(d)

	d)
A	no match
alpha	alpha
alpha1	alpha1
B	Beta

The difficulty with schemes That use initial abreviation (schemes b, c and d above) is that frequently one uses The last part of a name to distinguish files, e.g. alpha, alpha1, alpha2. In this case The abreviation is of no help.

2) The name is divided into parts and possibly subparts. The parts are supposed to indicate what kind of object is being named. for example, with a 2 part nameing system, The second part could be

binary

fortran

compass

etc. to indicate That The file is binary input for a loader, or source input for fortran or compass.

Each part can now be abreviated seperatly.
an algorithm might be as follows:

construct a name part by part as follows:

for each part if The given input name part exactly matchs some corresponding part in an entry, take that. else if There is some unique corresponding part in an entry an initial segment of which matches The given input name part, take That, otherwise no match.

having constructed a name, if There is a unique entry which matches the constructed name, take it, otherwise no match.

The name can be divided into 2 major parts, each of which has subparts. Then corresponding parts are i^{th} subpart of j^{th} major part where $j = 1$ or 2 and $i = 1, 2, \ldots$

an attempted example follows, using the 2 major part scheme. major parts divided by colon, minor parts divided by period.

directory

BLAST.1A: Fortran
BLAST.1A: Binary
BLAST.2 : Fortran
ALPHA : Fortran
BAKER.2 : Binary

input	match
B.1:F	(no match)
BL.1:F	Blast.1A: Fortran
BA.2:B	Baker.2 : Binary

It would appear that the 1st example should have picked up "Blast.1A: Fortran" as that is in some sense a unique match. But I do not now know how to state an algorithm I would accept which would work this way.

B) <u>object</u>

This part of the entry comes in 3 flavors

1) <u>ownership</u>

for some kinds of objects there are no ownership entries, for others there is exactly one directory with exactly one ownership entry for the object. (The later include at least disk files, i.e. directories, and subprocess descriptors)

An ownership entry includes some sort of unique designation of the object (for disk files i.e. directories and subprocess descriptors this is the unique name and disk address.)

Further specification will be given when the various kinds of object are described

2) <u>non ownership</u> or <u>Hard link</u>

consists of the same unique designation that an ownership entry would contain.

3) <u>Soft link</u>

consists of
a) Hard link to a directory
b) Hard link to an access key
c) text name

The obvious look up in the specified directory is made.

c) access list

This is a list (of arbitrary length) of pairs.

The 1st member of each pair is an access key number

The 2nd member of each pair is the set of allowed options ~~for the designated object~~ to be permitted using this access key.

(This second member may also contain the type field of the object, for convenience of implementation, but this is invisible to the user)

D) scratch bit

~~One of the option bits associated with a directory as a whole is implicit access. A reference to an ownership type entry via a directory capability with the implicit access bit on~~ ~~associated with each entry, as if it were in the access list,~~ ~~is an implicit access key~~.

associated with each ownership entry ~~as if it were in the access list~~ is an implicit access key. If the scratch bit is off, the associated ~~[illegible]~~ options ~~[illegible]~~ permit all actions. If the scratch bit is on, they permit only destruction.