Read\_config { ( pass in address to put it in and file name?)

Read up config file and put into table(s) in memory used in many

If not all routines

}

Set\_security\_user {

Save current user info from syscall into saved\_user

Set userid to requesting user (in fuse request structure)

Return

}

Set\_security\_saved {

Set userid to saved\_user using syscall

Return

}

Expand\_path\_info { (pass path, address to stuff, batch/interactive, working with existing file basically do we need to call stat\_xattr or not flag)

Take user supplied path in fuse request structure to look up info,

using MAR\_mnttop and look up in MAR\_namespace array,

fill in all MAR\_namespace and MAR\_repo info

into a big structure to hold everything we know about that path,

you need to pass in is this interactive (fuse) or batch

so you can use iperms or bperms as the perms to use)

if this is an existing file operation (you need to pass in if this may be a

existing file op) and if it is you need to pull in the Xattrs from

the existing file (use stat\_xattr() and put them

into the structure as well, so you have

everything needed to deal with this operation, whatever it might be

the reason you need to get the xattrs from the existing file for some ops

is that you need that info for how to do the subsequent read op

\*\*\* another part of this routine is to check for if the user is trying to get into the trash directory. If a user is trying to get into a trash directory, we need to deny it. This can be done by returning an error from this routine, that tells the calling routine that a permissions error occurred. This type of error may occur anyway in several ways since this routine may call stat\_xattr and that routine may get a permissions error which needs to be passed up.

The return is standard file ops return based on errors,

The expand\_Path\_info structure is essentially a completely resolved set of substructures:

* Expanded path
* Namespace info resolved to this path
* Repo info resolved to this repo if this is an existing file we are working on I guess?
* A stat structure (filled in only if you need to do stat\_xattr)
* A structure for the 3 potential reserved xattrs (filled in only if you need to do stat\_xattr)

return

}

read\_multi\_objid { pass in expanded path info structure

malloc space to hold the objid’s using the MAR\_objidbytes xattr from

the structure passed in. read the objid info from the metadata file,

put the address of that information into the expanded path info structure

return

}

Trash\_file { (renames file to trashfile which keeps all attrs)

(pass in expanded\_path\_info\_structure and file name to be trashed

rename mdfile (with all xattrs) into trashmdnamepath,

uniqueify name somehow with time perhaps == trashname,

rename file to trashname

trash\_name() record the full path in a related file in trash

return

}

Trashdup\_file { (copies trashed file into trash area, does NOT unlink original)

(pass in expanded\_path\_info\_structure and file name to be trashed

rename mdfile (with all xattrs) into trashmdnamepath,

If this has no xattrs (its just a normal file using the md file for data) just trunc the file and return – we have nothing to clean up, too bad for the user as we aren’t going to keep the trunc’d file.

uniqueify name somehow with time perhaps == trashname,

you already have all xattrs passed in expanded\_path\_info\_structure

open trashsir/trashname

if no xattr objtype

copy file data to file

if xattr objtype is multipart

copy file data using numobj and chunkinfobytes as a way to know how many bytes is chunk info

update trash file mtime to original mtime

trunc trash file to same length as original

set all reserved xattrs like original

close

trunc\_xattr()

trash\_name()

return

}

trunc\_xattr {

(pass in expanded\_path\_info\_structure and file name)

you have all xattrs passed in

trunc file to zero

remove (not just reset but remove) all reserved xattrs

return

}

Trash\_name {

(pass in expanded\_path\_info\_structure and file name and trashname)

make file in trash that has the full path of the file name and the inode number in the file data (having inode helps you not have to walk the tree for gpfs to do garbage collection/reclaim, you know object id, object type (uni, multi, packed, striped), file name, and inode. Gpfs has ilm that will list all but the file path very quickly.)

and call it $trashname.path

return

}

stat\_xattr { (stat the file and get all reserved xattrs

(pass in file path and a way to return stat and xattr info)

pull the three xattrs by name

push any errors up the call chain

return

}