Using Access Control Lists (ACLs) [3]

This chapter describes the discretionary access controls used by a UNICOS system. The following topics are described:

- Overview of ACLs on a UNICOS system
- How you can create, display, remove, and duplicate access control lists (ACLs) for your files
- The order in which access control lists (ACLs) are checked to determine the allowed access

3.1 Overview of ACLs

On a traditional UNIX system, discretionary access to a named object is controlled by the object's permission bits. These permission bits define the read (r), write (w), and execute (x) access granted to the owning user, owning group, and all others. Using only this mechanism is too restrictive because it is impossible to define the allowed access for multiple users and/or groups of users.

The UNICOS MLS discretionary access control mechanism uses the access control list (ACL), which provides a method to extend the traditional discretionary access mechanisms. ACLs can be applied to all named objects.

An ACL contains one or more ACL entries; each ACL entry has the following information (shown here in pseudo code for the sake of convenience):

user:*group*:*permissions*:

The *user* field defines the user's login name. The *group* field defines the group name, while the *permissions* field is used to define any combination of r, w, x, or no (n) access. A user's *current groups* are defined as a combination of the effective group and all of the user's group list entries; current groups is also referred to as *member of a specific group* and *belonging to a group* in the following paragraphs.

The ACL entries are intersected with the file's group (mask) bits to determine the type of access allowed; this is called the *effective permissions*. The term *absolute permissions* refers to the permissions defined in an ACL entry. The ACL entry types are shown in the following list:

Entry type	Description
User-only	The absolute permissions defined for a specific user, regardless of the user's current groups. The format for this type is `uid:*:`.
User-group	The absolute permissions defined for a specific user when that user is a member of a specific group. The format for this type is `uid:gid:`.
Group-only	The absolute permissions defined for any user that is a member of the specified group. The format for this type is `*:gid:`.
Owning-group	The absolute permissions defined for the group that owns the file. The format for this type is `*::`.

The owning-group entry type defines the file's group permission bits as the ACL mask. This mask defines the maximum permissions allowed for the object group (an object group being all the ACL entries for an object).

Because no group is specified in the owning-group entry, this entry always pertains to the group that owns the object at the time the discretionary access check is made. If the owning group is changed (by using the `chgrp`(1) command), this owning-group entry then pertains to the new owning group.

Note: When removing an ACL from a file, the owning group's permission is set to the permission granted the owning group with the ACL set. The owning group's permission does not change in the process of removing the ACL.

If there is a group-only entry for the group that owns the file, it is also used to determine the owning-group access. If neither an owning-group or group-only entry from the group that owns the file are found in the ACL, the owning-group permissions default to the ACL mask bits. See Section 3.3, page 55, for more information on the masking operation and the order in which entries are checked.

When you want file access to be controlled by an ACL, you must do the following:

1. Create an intermediate ACL file (by using the `spacl`(1) command).
2. Apply the intermediate ACL file to the file (by using the `spset`(1) command).

An intermediate ACL file is different from the system block where the information is stored. The intermediate file is a formatted version of the ACL that is created and displayed with the `spacl` command. The intermediate file information is copied to the system block when you use the `spset` command. Any changes to an intermediate ACL file are not copied into the system block unless you execute the `spset` command again. See Section 3.2.5, page 48, for more information on these differences.

ACLs can also be applied to Cray/REELlibrarian files and volumes. See the *Cray/REELlibrarian (CRL) User's Guide*, Cray Research publication SG–2126, and the `spset`(1) and `spacl`(1) man pages, for more information.

ACLs can also be applied to IPC objects (shared memory segments, message queues, and semaphores). See the `spset`(1) and `spclr`(1) man pages for more information.

3.2 Maintaining ACLs

You can use the `spacl` command to create, display, remove, or duplicate entries for a specified ACL. The `spset` command applies the ACL to a file.

The following sections explain how to use these commands. Each example in the following sections is built on knowledge defined in the previous section. Examining the examples out of sequence is not recommended.

3.2.1 Creating entries in an intermediate ACL file

The `spacl` command uses the following options to create entries in an intermediate ACL file:

- The `-a` *aclfile* option interactively adds entries to an intermediate ACL file. The *aclfile* argument is the name of the resultant intermediate ACL file.
- The `-i` *modfile aclfile* option inputs ACL text file changes from *modfile* into *aclfile*. The *modfile* argument is a file with add and/or remove statements. The *aclfile* argument is the name of the resultant intermediate ACL file.
- The `-t` *tmodfile aclfile* option inputs ACL text file changes from *tmodfile* into *aclfile*. The *tmodfile* argument is a file of an ACL display format from a previous execution of a `spget -a` or `spacl -l` command. The *aclfile* argument is the name of the resultant intermediate ACL file. The `-t` option is explained in Section 3.2.7, page 52.

3.2.1.1 Interactive creation of intermediate ACL files (`spacl -a`)

The `spacl -a` command interactively adds new entries to the intermediate ACL file (*aclfile*) by prompting you for the user name, group name, and permissions for each ACL entry. You can use the wildcard (`*`) character to specify all users in a group or a specific user in any group; specifying a wildcard character for both the user and group (`*:*`) in a single entry is not allowed. Specifying `*::` defines the owning-group entry.

Example 6 shows how to create an intermediate ACL file called `myacl` with four entries. At the `enter user's name OR an *` prompt in the first entry, the user called `tnn` is specified. The `spacl` command checks the specified user name against the valid choices listed in the user database (UDB). If you specify an invalid choice, the system prompts you for another choice.

At the `enter group name OR an *` prompt in the first entry, the wildcard (`*`) character specifies that `tnn` can belong to any group. The group entry is checked against the entries in `/etc/groups` to ensure it is a valid choice. If you specify an invalid choice, the system prompts you for another choice.

Finally, at the `enter access mode (n = none)` prompt, you must specify the access modes for the entry. You can specify any combination of `r`, `w`, and `x`, or `n`. The `rw` definition in the example results in absolute read and write access being allowed for `tnn` to the file protected by `myacl`.

The second entry allows a user called `jog`, who must belong to the `trng` group, to be allowed absolute permissions of read and write.

The third entry allows any user (specified by use of the wildcard character), who belongs to the `trng` group, to be allowed absolute permission of read.

The fourth entry defines the owning-group entry. This entry allows the owning group absolute permission of read and write.

As shown in the fifth entry, you cannot specify a wildcard character for both the user name and group. If you do, the `NO user or group name entered` message appears. Also, as shown in the sixth entry, duplicate user name and group entries are not allowed. If you try to enter duplicate entries, the `duplicate entry NOT accepted` message appears.

Example 6: Creating ACL entries (`spacl -a`)

```
$ spacl -a myacl

ENTER "quit" to END SESSION, "ctrl-c" to ABORT

enter user’s name OR an *......tnn
enter group name OR an *......*
enter access mode (n = none)......rw


ADD MODE

enter user’s name OR an *......jog
enter group name OR an *......trng
enter access mode (n = none)......rw

ADD MODE

enter user’s name OR an *......*
enter group name OR an *......trng
enter access mode (n = none)......r

ADD MODE

enter user’s name OR an *......*
enter group name OR an *.....
enter access mode (n = none)......rw
```

Entering `quit` at any point saves the changes and exits the interactive session (as shown in the last entry). Executing a `CONTROL-C` discards all changes and exits the session.

The absolute permissions defined in ACL entries are always intersected with the file’s mask bits to determine the requester’s effective access permissions. See Section 3.3, page 55, for more information on the masking operation and the order in which entries are checked. All of the following examples in this section are showing the absolute permissions.

3.2.1.2 Creation of intermediate ACL file entries using an input file (`spacl -i`)

The `-i` *modfile aclfile* option inputs ACL edit statements through *modfile* into *aclfile*. The following is the format for adding a record (line) in *modfile*:

`a:`*user_name*`:`*group_name*`:`*access_mode*`:`

Example 7, page 42 shows how to use the `spacl -i` command. When using the `-i` option, the user name, group name, and permission guidelines presented in the previous section apply.

In Example 7, page 42 the intermediate ACL file called `myacl2` is created and contains the same entries as the intermediate ACL file called `myacl` in the previous example, except for the last entry, which denies `root` access. See Section 3.3.2, page 59, for more information.

Example 7: Creating ACL entries (`spacl -i`)

```
$ cat modfile
a:tnn:*:rw:
a:jog:trng:rw:
a:*:trng:r:
a:*::rw:
a:root:*:n:
$ spacl -i modfile myacl2
spacl: end of file on modfile
```

3.2.2 Displaying intermediate ACL files (`spacl -l`)

You can display the contents of an intermediate ACL file by using one of the following methods:

- The `spacl -l` *aclfile* command, which lists a long version of the ACL entry.
- The `spacl -s` *aclfile* command, which lists a condensed version of the output shown when using the `-l` option.

The display of these two commands is shown in Example 8. The main difference between the two displays is that the uid and gid numbers (that correspond to the user name and group name, respectively) are shown in the `-l` display, but not in the `-s` display. The uid and gid numbers are unique identification numbers assigned to users and groups by your system administrator.

Example 8: Displaying an intermediate ACL file (`spacl -l` and `-s`)

```
$ spacl -l myacl
# ACL owner’s uid: 10505
# ACL owner’s name: ben

uid:1822     gid: -   user = tnn    group = *    mode = rw-
uid:927      gid: 28  user = jog    group = trng mode = rw-
uid:-        gid: 28  user = *      group = trng mode = r--
uid:-        gid: -   user = *      group =      mode = rw-

$ spacl -s myacl
# ACL owner’s name: ben

user = tnn     group = *      mode = rw-
user = jog     group = trng   mode = rw-
user = *       group = trng   mode = r--
user = *       group =        mode = rw-
```

3.2.3 Removing entries in intermediate ACL file

You can use the following `spacl` options to remove entries from a specified intermediate ACL file:

- The `-r` *aclfile* option interactively removes entries from an intermediate ACL file. The *aclfile* argument is the name of the resultant intermediate ACL file.
- The `-i` *modfile1 aclfile* option inputs ACL text file changes from *modfile1* into *aclfile*. The *modfile1* argument is a file with add or remove statements. The *aclfile* argument is the name of the resultant intermediate ACL file.

3.2.3.1 Interactive removal of intermediate ACL files (`spacl -r`)

The `-r` option removes entries from the *aclfile*. As with the `-a` option, you are prompted for a user name/group name pair that exists in the specified ACL. The permission prompt does not appear when using the `-r` option.

When removing entries, a question mark (`?`) is used as a wildcard character; specifying `?:?` is not allowed. The `*` is used to match user names and/or group names that were specified that way in the original entry.

Use of the `-r` option is shown in Example 9, page 45. First, the contents of `myacl2` (created in Section 3.2.1.2, page 42) are displayed by using the `spacl -l` command.

Then, by using the `spacl -r` command, the first entry removes the user called `tnn` in any group (specified by the `*`) from `myacl2`. The second entry removes all users (specified by the `?`) in the `trng` group from `myacl2`, which results in both the `jog:trng` and `*:trng` entries being removed. The third entry removes the owning-group entry.

These modifications produce an intermediate ACL file with only one entry, as shown in Example 9. This is the entry that denies `root` access.

Executing a `CONTROL-C` discards all changes and exits the interactive session. Entering `quit` saves the changes and exits the interactive session.

Example 9: Removing an intermediate ACL file entry (`spacl -r`)

```
$ spacl -l myacl2
# ACL owner’s uid: 10505
# ACL owner’s name: ben

uid:1822     gid: -   user = tnn    group = *    mode = rw-
uid:927      gid: 28  user = jog    group = trng mode = rw-
uid:0        gid: -   user = root   group = *    mode = n
uid:-        gid: 28  user = *      group = trng mode = r--
uid:-        gid: -   user = *      group =      mode = rw-
$ spacl -r myacl2

ENTER "quit" to END SESSION, "ctrl-c" to ABORT

REMOVE MODE

enter user’s name OR a ? OR an *......tnn
enter group name OR a ? OR an *.......*

REMOVE MODE

enter user’s name OR a ? OR an *......?
enter group name OR a ? OR an *.......trng

REMOVE MODE

enter user’s name OR a ? OR an *......*
enter group name OR a ? OR an *.......

REMOVE MODE

enter user’s name OR a ? OR an *......quit
entry session terminated

$ spacl -l myacl2
# ACL owner’s uid: 10505
# ACL owner’s name: ben

uid:0        gid: -   user = root   group = *    mode = n
```

3.2.3.2 Removing entries in intermediate ACLs files using an input file (`spacl -i`)

The `-i` *modfile aclfile* option inputs ACL edit statements through *modfile* into *aclfile*. Using this option to remove entries is similar to the add example shown in Section 3.2.1.2, page 42. The following is the format for removing a record (line) in *modfile*:

`r:`*user_name*`:`*group_name*`:`

You can use the question mark (`?`) when removing all instances of a user or group, as shown in the `spacl -r` example; specifying `?:?` is not allowed.

3.2.4 Modifying intermediate ACL files

If you want to modify an existing intermediate ACL file, you can do so by in one of the following ways:

- If you are adding a new entry, use the `spacl -a` command as described in Section 3.2.1.1, page 40.
- If you want to delete an entry, use the `spacl -r` command as described in Section 3.2.3.1, page 43.
- If you want to change the access mode of an existing ACL entry, you must first remove the entry by using the `spacl -r` command and then recreate the entry with the new access mode by using the `spacl -a` command.

Example 10 shows how to modify the ACL entry for `jog`. First, the entry for `jog` is created to allow read access by using the `spacl - a` command. Second, the entry for `jog` is removed by using the `spacl -r` command. Last, the entry for `jog` is recreated (again, by using the `spacl -a` command), but this time both read and write access are allowed.

Example 10: Modifying an existing ACL entry

```
$ spacl -a myacl2

ENTER "quit" to END SESSION, "ctrl-c" to ABORT

ADD MODE

enter user’s name OR an *......jog
enter gruop name OR an *......trng
enter access mode (n = none)...r

enter user’s name OR an *......quit
entry session terminated

& spacl -r myacl2

ENTER "quit" to END SESSION, "ctrl-c" to ABORT

REMOVE MODE

enter user’s name OR a ? OR an *......jog
enter gruop name OR an *......trng

REMOVE MODE

enter user’s name OR a ? OR an *......quit
entry session terminated

& spacl -a myacl2

ENTER "quit" to END SESSION, "ctrol-c" to ABORT

ADD MODE

enter user’s name or an *......jog
enter group name OR an *......trng
enter access mode (n=none).....rw

ADD MODE

enter user’s name OR a ? OR an *.....quit
entry session terminated
```

3.2.5 Applying ACLs to files (`spset -a`)

Once an intermediate ACL file has been created or modified, you must apply it to the file by using the `spset -a` *aclfile files* command. You must be the owner of the file (or the security administrator) to apply an ACL to a file. The `spset -a` command copies the contents of an intermediate ACL file to an ACL system block. The system block is defined by a pointer in the inode of the ACL-protected file.

An ACL system block is not shared by multiple files in the file system, even though the same intermediate ACL file can be applied to multiple files by using the `spset -a` command.

As shown in Figure 8, `myacl` is the *aclfile* (intermediate ACL file). It has an inode associated with it, which identifies the data block(s) from which the intermediate ACL file information is copied when the `spset -a` command is executed.

Figure 8 also shows that a system block is allocated for `file1` and `file2`. The address of the system block is stored in the inode of each file; this is the pointer to the ACL. The system block ACL information is transparent to you as a user and can be accessed only by the system kernel.

The contents of `myacl` are converted into binary and copied into the system block for each file each time the `spset -a` command is executed. If any of the entries in `myacl` are modified, the new information is not automatically copied into the system block unless you use the `spset -a` command again.

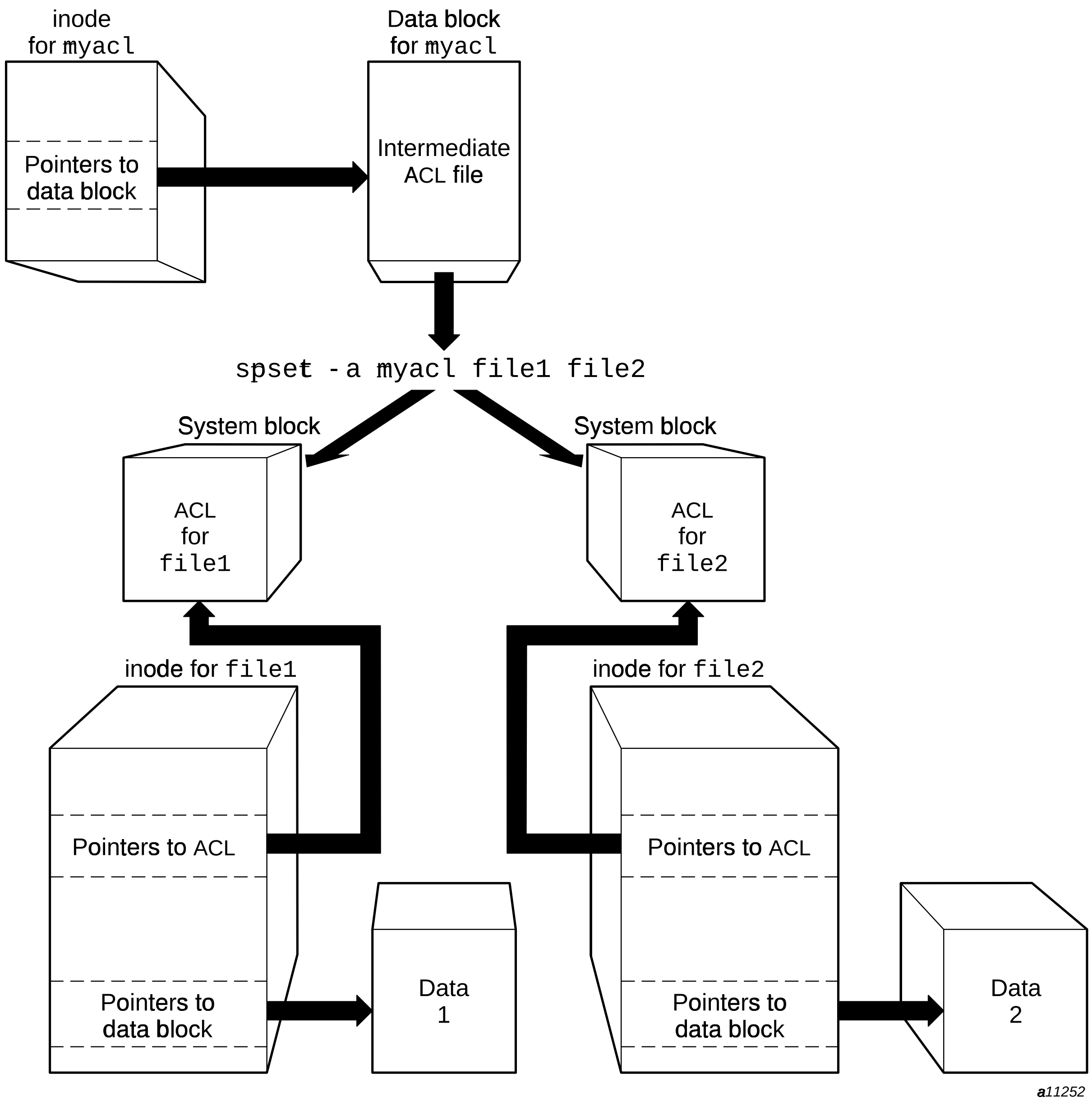

Figure 8. Connections between ACLs and files

Example 11 shows how to use the `spset -a` command. In this example, `myacl` is first displayed by using the `spacl -l` command and then applied to the file called `newfile` by using the `spset -a` command.

As stated previously, any changes made to an intermediate ACL file are not automatically copied into the system block. In Example 12, changes are made to `myacl` by using the `spacl -a` command to add the `jack:*:r` entry.

To update the system block with this new information, the `spset -a` command must be executed again. Because this file is already protected by an ACL, the system will ask you if you want to replace it. Notice that the last step in Example 12 removes `myacl`. This does not affect `newfile` because it has its own copy of the ACL entries.

Example 11: Applying ACLs (`spset -l`)

```
$  spacl -l myacl

#ACL owner's uid: 10505
#ACL owner's name: ben

uid:1822     gid:-     user = tnn    group = *     mode = rw-
uid:927      gid:28    user = jog    group = trng  mode = rw-
uid:-        gid:28    user = *      group = trng  mode = r--
uid:-        gid:-     user = *      group =       mode = rw-
$ spset -a myacl newfile
```

Example 12: Applying ACLs (`spset -a`)

```
$ spacl -a myacl

ENTER "quit" to END SESSION, "ctrl-c" to ABORT

ADD MODE

enter user’s name OR an *......jack
enter group name OR an *.......*
enter access mode (n=none).....r

ADD MODE

enter user’s name OR an *......quit
entry session terminated
$ spset -a myacl newfile
spset: replace existing acl for file? y
$ rm myacl
```

3.2.6 Displaying ACLs applied to files (`spget -a`)

If you are unsure which files have ACLs applied to them, use the `ls -e` command as shown in Example 13. An `a` appears prior to the file name if an ACL is applied. If you use the `ls -le` command, the `a` appears immediately after the UNICOS mode permissions. See Section 5.2, page 78, for more information on the `ls` command.

You can display the ACL entries by using the `spget -a` or `spget -ar` commands, as shown Example 14. The `spget -ar` command displays a reduced format, which is useful when duplicating ACL entries (an example of this is shown in the next section).

Example 13: Displaying ACL entries (`spget -e`)

```
$ ls -e
  0 file
  0 modfile
  0 myacl2
a 0 newfile
  0 testfile
$ ls -le newfile
-rw-------a  0   1 ben   trng     60  May 16 15:10  newfile
```

Example 14: Displaying ACL entries (`spget -a`)

```
$ spget -a newfile
# ACL Information for: newfile
uid:1822     gid:-     user = tnn    group = *     mode = rw-
uid:196      gid:-     user = jack   group = *     mode = r--
uid:927      gid:28    user = jog    group = trng  mode = rw-
uid:-        gid:28    user = *      group = trng  mode = r--
uid:-        gid:-     user = *      group =       mode = rw-

$ spget -ar newfile
#ACL information for: newfile
user = tnn     group = *     mode = rw-
user = jack    group = *     mode = r--
user = jog     group = trng  mode = rw-
user = *       group = trng  mode = r--
user = *       group =       mode = rw-
```

3.2.7 Duplicating ACLs (`spset -d` or `spacl -t`)

It may be necessary to duplicate the contents of an ACL. An example of this is if you want to apply an ACL to another of your files, but you cannot locate the intermediate ACL file.

There are two ways you can duplicate an ACL from one file and apply it to another file. The first way is to use the `spset -d` command; the second way is a three-step process using the `spget -ar`, `spacl -t`, and the `spset -a` commands. The second method generates an intermediate ACL file.

To duplicate the ACL permissions found in the ACL applied to one file and apply them to another file, use the `spset -d` *file1 file2* command, as shown in Example 15. In this example, *file1* is the file that has the ACL that you want to duplicate for *file2*.

Example 15: Duplicating ACLs (`spset -d`)

```
$ spget -a newfile
# ACL Information for: newfile
uid:1822     gid:-     user = tnn    group = *     mode = rw-
uid:196      gid:-     user = jack   group = *     mode = r--
uid:927      gid:28    user = jog    group = trng  mode = rw-
uid:-        gid:28    user = *      group = trng  mode = r--
uid:-        gid:-     user = *      group =       mode = rw-
$ spset -d newfile testfile
$ spget -a testfile
# ACL Information for: testfile
uid:1822     gid:-     user = tnn    group = *     mode = rw-
uid:196      gid:-     user = jack   group = *     mode = r--
uid:927      gid:28    user = jog    group = trng  mode = rw-
uid:-        gid:28    user = *      group = trng  mode = r--
uid:-        gid:-     user = *      group =       mode = rw-
$ ls -le testfile
-rw-------a  0   1 jack  trng     77  May 16 15:15  testfile
```

To duplicate an ACL from one file to another, even though the intermediate ACL file has been removed from the directory, you can use the following three-step process (shown in Example 16):

1. Create a file that contains the ACL information in a reduced format. You do this by using the `spget -ar` command to redirect the output of a file's ACL into a text file. The `-r` option (which can only be used with the `-a` option) gets a reduced version of the ACL information.

2. Create an intermediate ACL file. You do this by using the `spacl -t` command to convert the reduced format from step 1 into an intermediate ACL file.

3. Apply the ACL file created in step 2 to the new file. You do this by using the `spset -a` command.

When using this method, you can edit the reduced format `text` before issuing the `spacl -t` command. Use a text editor to add or remove entries or modify existing entries. Thus, a similar, but not identical ACL is applied to `file` when the `spset -a` is executed.

Example 16: Duplicating ACLs (`spacl -t`)

```
$ spget -ar testfile > text
$ spacl -t text myacl
spacl:  end of file on text
$ spacl -l myacl
#ACL owner’s uid: 10505
#ACL owner’s name: ben

uid:1822     gid:-     user = tnn    group = *     mode = rw-
uid:196      gid:-     user = jack   group = *     mode = r--
uid:927      gid:28    user = jog    group = trng  mode = rw-
uid:-        gid:28    user = *      group = trng  mode = r--
uid:-        gid:-     user = *      group =       mode = rw-
$ spset -a myacl file
$ spget -a file
# ACL Information for: file
uid:1822     gid:-     user = tnn    group = *     mode = rw-
uid:196      gid:-     user = jack   group = *     mode = r--
uid:927      gid:28    user = jog    group = trng  mode = rw-
uid:-        gid:28    user = *      group = trng  mode = r--
uid:-        gid:-     user = *      group =       mode = rw-
```

3.2.8 Removing ACLs (`spclr -a`)

You can use the `-a` option of the `spclr`(1) command to remove an ACL from one or more files, as shown in Example 17.

The `spclr` command can fail for the following reasons:

- You are not the owner of the file or you are not the security administrator.
- An ACL is not applied to the file.
- An invalid ACL was detected.

If you do not have the correct permission to access the directory in which the file is located, you will get a `permission denied` message. If you are not the owner of the file (or are not an active security administrator), and you try to remove an ACL, you will get a `spclr:  acl remove error for` *filename* message.

Example 17: Removing ACLs from files (`spclr -a`)

```
$ ls -le newfile
-rw-------a  0   1 ben  trng     77  May 16 15:15  newfile
$ spclr -a newfile
$ ls -le newfile
-rw-------   0   1 ben  trng     77  May 16 15:15  newfile
```

3.3 How ACLs are checked

The order in which ACL entries are searched and the masking operation is shown in Figure 9. The following explanation follows the numbered sequence shown in the figure.

When you attempt to gain access to a file, the ACL of the file is searched. The first check made in the ACL is to see if you are `root` (this step is not shown in Figure 9). Depending on your system configuration, one of the following occurs:

- On systems using the super-user mechanism, `root` is always granted DAC access to a file, even if there is an entry for `root` in the ACL.
- On systems using the PAL-based privilege mechanism, `root` is not given any special consideration for DAC access. It is treated the same as an entry for any user and the ACL can be used to determine (or deny) access for `root`. A process is allowed to override the DAC restrictions only if the process belonging to `root` has the `PRIV_DAC_OVERRIDE` privilege in its effective privilege set.

If you are not `root` or your system is using the PAL-based privilege mechanism, then a check is made to determine if you are the owner of the file (shown in step 1 of Figure 9). If you are, the access specified by the file's owner permission bits are granted. The ACL is not checked in this case; regardless of the ACL entry, the type of access that a file's owner is granted can be determined by using the `ls -l` command.

If you are not the owner of the file, then the ACL is searched for the set of `uid:gid:` and/or `uid:*:` entry or entries that relates to you and your current groups (shown in step 2 of Figure 9). The current groups are your effective gid and your group list. If there is a `uid:gid:` entry or entries for you and any of your current groups, the entry or a union of these entries is intersected with the file's mask bits to determine the type of access allowed.

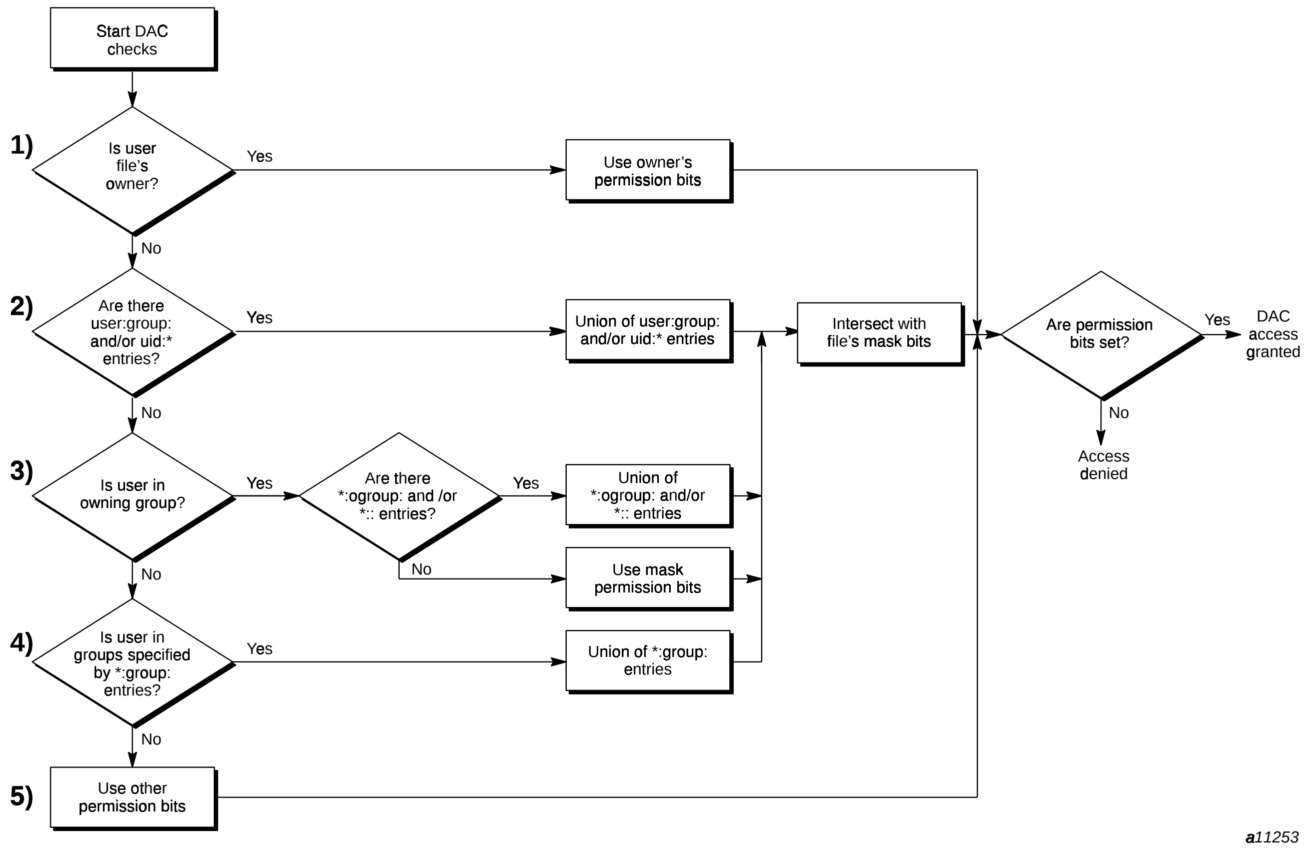
Start DAC checks
1)
Is user file's owner?
Yes
Use owner's permission bits
No
2)
Are there user:group: and/or uid:* entries?
Yes
Union of user:group: and/or uid:* entries
Intersect with file's mask bits
Are permission bits set?
Yes
DAC access granted
No
Access denied
3)
Is user in owning group?
Yes
Are there *:ogroup: and /or *:: entries?
Yes
Union of *:ogroup: and/or *:: entries
No
Use mask permission bits
4)
Is user in groups specified by *:group: entries?
Yes
Union of *:group: entries
No
5)
Use other permission bits
a11253

Figure 9. Flowchart of how UNICOS ACLs are checked

An example of this check is shown with the following ACL entries for a user named Jack with groups training and testing wants to access a file that has its mask bits set to `r-x`. Assume for this example that Jack has only the following entries defined for him:

```
user: jack   group: training   mode: r--
user: jack   group: testing    mode: -w-
user: jack   group: *          mode: --x
```

The union of these entries (which would be `rwx`) is intersected with the file's mask bits, resulting in Jack having `r-x` access to that file. If Jack requests write access to this file, it is denied.

The union of multiple entries is used because of the fact that the UNICOS operating system supports multiple groups. Thus, users can simultaneously belong to all of their groups and no single group takes precedence over any other group.

If no match was found in step 2, then a check is made to see if you belong to the owning group of the file (shown in step 3 of Figure 9). If yes, a search is made for a `*:ogroup:` (where `ogroup` is the group-only entry for the owning group) and/or a `*::` entry (where `*::` is the owning-group entry). If either of these entries is found, the entry or a union of these entries is intersected with the file's mask bits to determine the type of access allowed.

If neither a `*:ogroup:` or `*::` entry is found, then the file's mask permissions are used to determine the type of access allowed to the member of the owning group.

If you do not belong to the owning group, then the ACL is checked for a `*:group:` entry that matches your current groups (shown in step 4 of Figure 9). If one or more matching entries are found, then the entry or a union of all `*:group:` entries are intersected with the file's mask bits to determine the type of access allowed.

If no matching `*:group` entry is found, then access is granted according to the file's other permission bits (shown in step 5 of Figure 9). There is no intersection with the file's mask bits in this check.

A user or group can be denied permission by entering `n` in the ACL, as shown in the following example:

```
jack : * : n
```

As explained previously, multiple entries for the same user or group are combined for the masking operation. When specifying `n` (no) access, be certain that multiple matching entries are not specified in the ACL. See Section 3.2.1, page 39, for more information on adding entries to an ACL.

3.3.1 Displaying masked ACL permissions (`spget -ae`)

You can use the `spget -ae` command to show the masked ACL entries. Execution of this command masks each ACL entry's mode bits against the file's mask permissions and displays the resultant mode bits. Example 18, page 58 shows how the `-ae` option works (the `-e` option can be used only with the `-a` option).

In Example 18, page 58, execution of the `ls -le` command reveals the standard permission bits for `testfile` allow read and write access for the user only (`-rw-------a`). Executing the `spget -a` command displays the absolute permissions defined in the ACL entries for `testfile`.

Because the mask permission bits for `testfile` do not allow access for any group, the masking operation results in no access for any user listed in the ACL. This result is shown in the display of the `spget -ae` command.

In Example 19, page 59, the file's permission bits are changed by executing `chmod g+r`, which results in the mask permissions for `testfile` allowing read access. The masking operation results in read access for all the users. This result is shown in the display of the `spget -ae` command.

Example 18: Displaying the masked ACL mode bits (`spget -ae`)

```
$ ls -le testfile
-rw-------a  0   1 ben  trng     77  May 16 15:15  testfile
$
$ spget -a testfile
# ACL Information for: testfile
uid:1822     gid:-     user = tnn    group = *     mode = rw-
uid:196      gid:-     user = jack   group = *     mode = r--
uid:927      gid:28    user = jog    group = trng  mode = rw-
uid:-        gid:28    user = *      group = trng  mode = r--
uid:-        gid:-     user = *      group =       mode = rw-
$
$ spget -ae testfile
# ACL Information for: testfile
uid:1822     gid:-     user = tnn    group = *     mode = n
uid:196      gid:-     user = jack   group = *     mode = n
uid:927      gid:28    user = jog    group = trng  mode = n
uid:-        gid:28    user = *      group = trng  mode = n
uid:-        gid:28    user = *      group =       mode = n
```

Example 19: Displaying the masked ACL mode bits (`spget -ae`)

```
$ chmod g+r testfile
$ ls -le testfile
-rw-r-----a  0   1 ben  trng     77  May 16 15:15  testfile
$
$ spget -ae testfile
# ACL Information for: testfile
uid:1822     gid:-     user = tnn    group = *     mode = r--
uid:196      gid:-     user = jack   group = *     mode = r--
uid:927      gid:28    user = jog    group = trng  mode = r--
uid:-        gid:28    user = *      group = trng  mode = r--
uid:-        gid:-     user = *      group =       mode = r--
```

In Example 20, page 59, the file's permission bits are changed by executing `chmod g+w`, which results in the mask permissions for `testfile` allowing read and write access. The masking operation results in read and write access for the `tnn:*`, `jog:trng`, and owning-group entries. The `jack:*` and `*:trng` entries are not granted write access because the ACL entries defined an absolute access of read only.

Example 20: Displaying the masked ACL mode bits (`spget -ae`)

```
$ chmod g+w testfile
$ ls -le testfile
-rw-rw----a  0   1 ben  trng     77  May 16 15:15  testfile
$
$ spget -ae testfile
# ACL Information for: testfile
uid:1822     gid:-     user = tnn    group = *     mode = rw-
uid:196      gid:-     user = jack   group = *     mode = r--
uid:927      gid:28    user = jog    group = trng  mode = rw-
uid:-        gid:28    user = *      group = trng  mode = r--
uid:-        gid:-     user = *      group =       mode = rw-
```

3.3.2 ACLs and `root` access

On a system using the PAL-based privilege mechanism, `root` is subject to all ACL rules. An ACL entry for `root` is treated the same as for any other user.

On a system using the super-user mechanism, `root` can override the ACL, even if it contains an entry for `root`.

If the file's ACL does contain an entry for `root`, then the entry is masked against the file's mask permissions as explained in Section 3.3, page 55. This means that you can deny `root` access by entering `n` in the ACL, as shown in Example 7, page 42, and in the following example:

```
root:*:n
```

Using this type of entry does not completely prevent `root` access for the following reasons:

- `root` has the ability to bypass a file's ACL protection by changing the permission mode, owner, and/or owning group of a file.
- `root` is allowed to access file system data through device special files, thus allowing `root` the ability to grant itself discretionary access to an object by changing the contents of the object's inode.

On a system using the super-user mechanism, if a file does not have an ACL, then `root` is automatically granted rwx access if all the mandatory access control checks are passed. For systems using the PAL-based privilege mechanism, if the file does not have an ACL, `root` is granted access through the permission mode bits.

3.3.3 umask(1) default access permissions

On a UNICOS system, the default setting for `umask` is 077 (instead of 027 as on previous non-MLS systems). This default setting is necessary to meet the TCSEC requirements. This default must be used on a Cray ML-Safe configuration of the UNICOS system.

Use of this default affects you in the following ways:

- If you want new files to automatically have group and/or world access, you must manually set the file access permission bits to enable this type of access.
- If you want to use a default other than 077, place `umask` in your `$HOME/.profile` or `$HOME/.login` file.