NAME

archive_entry_linkresolver, **archive_entry_linkresolver_new**, **archive_entry_linkresolver_set_strategy**, **archive_entry_linkresolver_free**, **archive_entry_linkify** — hardlink resolver functions

LIBRARY

library “libarchive”

SYNOPSIS

#include <archive_entry.h>

*struct archive_entry_linkresolver **
archive_entry_linkresolver_new(*void*);

void
archive_entry_linkresolver_set_strategy(*struct archive_entry_linkresolver *resolver*
    int format);

void
archive_entry_linkresolver_free(*struct archive_entry_linkresolver *resolver*);

void
archive_entry_linkify(*struct archive_entry_linkresolver *resolver*,
    *struct archive_entry **entry*, *struct archive_entry **sparse*);

DESCRIPTION

Programs that want to create archives have to deal with hardlinks. Hardlinks are handled in different ways by the archive formats. The basic strategies are:

1. Ignore hardlinks and store the body for each reference (old cpio, zip).

2. Store the body the first time an inode is seen (ustar, pax).

3. Store the body the last time an inode is seen (new cpio).

The **archive_entry_linkresolver** functions help by providing a unified interface and handling the complexity behind the scene.

The **archive_entry_linkresolver** functions assume that *archive_entry* instances have valid nlinks, inode and device values. The inode and device value is used to match entries. The nlinks value is used to determined if all references have been found and if the internal references can be recycled.

The **archive_entry_linkresolver_new**() function allocates a new link resolver. The instance can be freed using **archive_entry_linkresolver_free**(). All deferred entries are flushed and the internal storage is freed.

The **archive_entry_linkresolver_set_strategy**() function selects the optimal hardlink strategy for the given format. The format code can be obtained from `archive_format`(3). The function can be called more than once, but it is recommended to flush all deferred entries first.

The **archive_entry_linkify**() function is the core of **archive_entry_linkresolver**. The **entry**() argument points to the *archive_entry* that should be written. Depending on the strategy one of the following actions is taken:

1. For the simple archive formats **entry* is left unmodified and **sparse* is set to `NULL`.

2. For tar like archive formats, **sparse* is set to `NULL`. If **entry* is `NULL`, no action is taken. If the hardlink count of **entry* is larger than 1 and the file type is a regular file or symbolic link, the internal list is searched for a matching inode. If such an inode is found, the link count is decremented and the file size of **entry* is set to 0 to notify that no body should be written. If no such inode is found, a copy of the entry is added to the internal cache with a link count reduced by one.

3. For new cpio like archive formats a value for ∗*entry* of `NULL` is used to flush deferred entries. In that case ∗*entry* is set to an arbitrary deferred entry and the entry itself is removed from the internal list. If the internal list is empty, ∗*entry* is set to `NULL`. In either case, ∗*sparse* is set to `NULL` and the function returns. If the hardlink count of ∗*entry* is one or the file type is a directory or device, ∗*sparse* is set to `NULL` and no further action is taken. Otherwise, the internal list is searched for a matching inode. If such an inode is not found, the entry is added to the internal list, both ∗*entry* and ∗*sparse* are set to `NULL` and the function returns. If such an inode is found, the link count is decremented. If it remains larger than one, the existing entry on the internal list is swapped with ∗*entry* after retaining the link count. The existing entry is returned in ∗*entry*. If the link count reached one, the new entry is also removed from the internal list and returned in ∗*sparse*. Otherwise ∗*sparse* is set to `NULL`.

The general usage is therefore:

1. For each new archive entry, call **`archive_entry_linkify`**().
2. Keep in mind that the entries returned may have a size of 0 now.
3. If ∗*entry* is not `NULL`, archive it.
4. If ∗*sparse* is not `NULL`, archive it.
5. After all entries have been written to disk, call **`archive_entry_linkify`**() with ∗*entry* set to `NULL` and archive the returned entry as long as it is not `NULL`.

RETURN VALUES

`archive_entry_linkresolver_new`() returns `NULL` on `malloc`(3) failures.

SEE ALSO

`archive_entry`(3)