```
**********************************************************
   10613 Fri Dec  4 18:09:24 2015
new/usr/src/man/man3lib/libavl.3lib
6498 typo in libavl(3LIB) man page
Reviewed by: Marcel Telka <marcel@telka.sk>
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
**********************************************************
.\"
.\" This file and its contents are supplied under the terms of the
.\" Common Development and Distribution License ("CDDL"), version 1.0.
.\" You may only use this file in accordance with the terms of version
.\" 1.0 of the CDDL.
.\"
.\" A full copy of the text of the CDDL should have accompanied this
.\" source.  A copy of the CDDL is also available via the Internet at
.\" http://www.illumos.org/license/CDDL.
.\"
.\"
.\" Copyright 2015 Joyent, Inc.
.\"
.Dd Dec 04, 2015
.Dd May 07, 2015
.Dt LIBAVL 3LIB
.Os
.Sh NAME
.Nm libavl
.Nd generic self-balancing binary search tree library
.Sh SYNOPSIS
.Lb libavl
.In sys/avl.h
.Sh DESCRIPTION
The
.Nm
library provides a generic implementation of AVL trees, a form of
self-balancing binary tree. The interfaces provided allow for an
efficient way of implementing an ordered set of data structures and, due
to its embeddable nature, allow for a single instance of a data
structure to belong to multiple AVL trees.
.Lp
Each AVL tree contains entries of a single type of data structure.
Rather than allocating memory for pointers for those data structures,
the storage for the tree is embedded into the data structures by
declaring a member of type
.Vt avl_node_t .
When an AVL tree is created, through the use of
.Fn avl_create ,
it encodes the size of the data structure, the offset of the data
structure, and a comparator function which is used to compare two
instances of a data structure. A data structure may be a member of
multiple AVL trees by creating AVL trees which use different
offsets (different members) into the data structure.
.Lp
AVL trees support both look up of an arbitrary item and ordered
iteration over the contents of the entire tree. In addition, from any
node, you can find the previous and next entries in the tree, if they
exist. In addition, AVL trees support arbitrary insertion and deletion.
.Ss Performance
AVL trees are often used in place of linked lists. Compared to the
standard, intrusive, doubly linked list, it has the following
performance characteristics:
.Bl -hang -width Ds
.It Sy Lookup One Node
.Bd -filled -compact
Lookup of a single node in a linked list is
.Sy O(n) ,
whereas lookup of a single node in an AVL tree is
```



```
.Sy O(log(n)) .
.Ed
.It Sy Insert One Node
.Bd -filled -compact
Inserting a single node into a linked list is
.Sy O(1) .
Inserting a single node into an AVL tree is
.Sy O(log(n)) .
.Pp
Note, insertions into an AVL tree always result in an ordered tree.
Insertions into a linked list do not guarantee order. If order is
required, then the time to do the insertion into a linked list will
depend on the time of the search algorithm being employed to find the
place to insert at.
.Ed
.It Sy Delete One Node
.Bd -filled -compact
Deleting a single node from a linked list is
.Sy O(1),
whereas deleting a single node from an AVL tree takes
.Sy O(log(n))
time.
.Ed
.It Sy Delete All Nodes
.Bd -filled -compact
Deleting all nodes from a linked list is
.Sy O(n) .
With an AVL tree, if using the
.Xr avl_destroy_nodes 3AVL
.Xr avl_delete_nodes 3AVL
function then deleting all nodes
is
.Sy O(n) .
However, if iterating over each entry in the tree and then removing it using
a while loop,
.Xr avl_first 3AVL
and
.Xr avl_remove 3AVL
then the time to remove all nodes is
.Sy O(n\ *\ log(n)).
.Ed
.It Sy Visit the Next or Previous Node
.Bd -filled -compact
Visiting the next or previous node in a linked list is
.Sy O(1) ,
whereas going from the next to the previous node in an AVL tree will
take between
.Sy O(1)
and
.Sy O(log(n)) .
.Ed
.El
.Pp
In general, AVL trees are a good alternative for linked lists when order
or lookup speed is important and a reasonable number of items will be
present.
.Sh INTERFACES
The shared object
.Sy libavl.so.1
provides the public interfaces defined below. See
.Xr Intro(3)
for additional information on shared object interfaces. Individual
functions are documented in their own manual pages.
.Bl -column -offset indent ".Sy avl_is_empty" ".Sy avl_destroy_nodes"
.It Sy avl_add Ta Sy avl_create
.It Sy avl_destroy Ta Sy avl_destroy_nodes
```

```
.It Sy avl_find Ta Sy avl_first
.It Sy avl_insert Ta Sy avl_insert_here
.It Sy avl_is_empty Ta Sy avl_last
.It Sy avl_nearest Ta Sy avl_numnodes
.It Sy avl_remove Ta Sy avl_swap
.El
.Pp
In addition, the library defines C pre-processor macros which are
defined below and documented in their own manual pages.
.\"
.\" Use the same column widths in both cases where we describe the list
.\" of interfaces, to allow the manual page to better line up when rendered.
.\"
.Bl -column -offset indent ".Sy avl_is_empty" ".Sy avl_destroy_nodes"
.It Sy AVL_NEXT Ta Sy AVL_PREV
.El
.Sh TYPES
The
.Nm
library defines the following types:
.Lp
.Sy avl_tree_t
.Lp
Type used for the root of the AVL tree. Consumers define one of these
for each of the different trees that they want to have.
.Lp
.Sy avl_node_t
.Lp
Type used as the data node for an AVL tree. One of these is embedded in
each data structure that is the member of an AVL tree.
.Lp
.Sy avl_index_t
.Lp
Type used to locate a position in a tree. This is used with
.Xr avl_find 3AVL
and
.Xr avl_insert 3AVL .
.Sh LOCKING
The
.Nm
library provides no locking. Callers that are using the same AVL tree
from multiple threads need to provide their own synchronization. If only
one thread is ever accessing or modifying the AVL tree, then there are
no synchronization concerns. If multiple AVL trees exist, then they may
all be used simultaneously; however, they are subject to the same rules
around simultaneous access from a single thread.
.Lp
All routines are both
.Sy Fork-safe
and
.Sy Async-Signal-Safe .
Callers may call functions in
.Nm
from a signal handler and
.Nm
calls are all safe in face of
.Xr fork 2 ;
however, if callers have their own locks, then they must make sure that
they are accounted for by the use of routines such as
.Xr pthread_atfork 3C .
.Sh EXAMPLES
The following code shows examples of exercising all of the functionality
that is present in
.Nm .
It can be compiled by using a C compiler and linking
against
```



```
.Nm .
For example, given a file named avl.c, with gcc, one would run:
.Bd -literal
$ gcc -Wall -o avl avl.c -lavl
.Ed
.Bd -literal
/*
 * Example of using AVL Trees
 */

#include <sys/avl.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>

static avl_tree_t inttree;

/*
 * The structure that we're storing in an AVL tree.
 */
typedef struct intnode {
        int in_val;
        avl_node_t in_avl;
} intnode_t;
```

______***unchanged_portion_omitted_***