Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'bsearch.3p' command

$ man bsearch.3p

BSEARCH(3P) POSIX Programmer's Manual BSEARCH(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

bsearch ? binary search a sorted table

SYNOPSIS

```
#include <stdlib.h>
void *bsearch(const void *key, const void *base, size_t nel,
    size_t width, int (*compar)(const void *, const void *));
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1?2017 defers to the ISO C standard.

The bsearch() function shall search an array of nel objects, the ini? tial element of which is pointed to by base, for an element that matches the object pointed to by key. The size of each element in the array is specified by width. If the nel argument has the value zero, the comparison function pointed to by compar shall not be called and no match shall be found.

The comparison function pointed to by compar shall be called with two arguments that point to the key object and to an array element, in that order.

The application shall ensure that the comparison function pointed to by compar does not alter the contents of the array. The implementation may reorder elements of the array between calls to the comparison function, but shall not alter the contents of any individual element.

The implementation shall ensure that the first argument is always a pointer to the key.

When the same objects (consisting of width bytes, irrespective of their current positions in the array) are passed more than once to the com? parison function, the results shall be consistent with one another.

That is, the same object shall always compare the same way with the key.

The application shall ensure that the function returns an integer less than, equal to, or greater than 0 if the key object is considered, re? spectively, to be less than, to match, or to be greater than the array element. The application shall ensure that the array consists of all the elements that compare less than, all the elements that compare equal to, and all the elements that compare greater than the key ob? ject, in that order.

RETURN VALUE

The bsearch() function shall return a pointer to a matching member of the array, or a null pointer if no match is found. If two or more mem? bers compare equal, which member is returned is unspecified.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

The example below searches a table containing pointers to nodes con? sisting of a string and its length. The table is ordered alphabetically on the string in the node pointed to by each entry.

The code fragment below reads in strings and either finds the corre?

sponding node and prints out the string and its length, or prints an error message.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TABSIZE    1000
struct node {                /* These are stored in the table. */
    char *string;
    int length;
};
struct node table[TABSIZE];    /* Table to be searched. */
    .
    .
    .
{
    struct node *node_ptr, node;
    /* Routine to compare 2 nodes. */
    int node_compare(const void *, const void *);
    .
    .
    .
    while (scanf("%ms", &node.string) != EOF) {
        node_ptr = (struct node *)bsearch((void *)(&node),
              (void *)table, TABSIZE,
              sizeof(struct node), node_compare);
        if (node_ptr != NULL) {
            (void)printf("string = %20s, length = %d\n",
                node_ptr->string, node_ptr->length);
        } else {
            (void)printf("not found: %s\n", node.string);
        }
        free(node.string);
    }
```

```
}
/*
    This routine compares two nodes based on an
    alphabetical ordering of the string field.
*/
int
node_compare(const void *node1, const void *node2)
{
    return strcoll(((const struct node *)node1)->string,
        ((const struct node *)node2)->string);
}
```

APPLICATION USAGE

The pointers to the key and the element at the base of the table should be of type pointer-to-element.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being com? pared.

In practice, the array is usually sorted according to the comparison function.

RATIONALE

The requirement that the second argument (hereafter referred to as p) to the comparison function is a pointer to an element of the array im? plies that for every call all of the following expressions are non-zero:

```
( (char *)p - (char *)base ) % width == 0
(char *)p >= (char *)base
(char *)p < (char *)base + nel * width
```

FUTURE DIRECTIONS

None.

SEE ALSO

hcreate(), lsearch(), qsort(), tdelete()

The Base Definitions volume of POSIX.1?2017, <stdlib.h>

COPYRIGHT