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 'insque.3p' command

$ man insque.3p

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

insque, remque ? insert or remove an element in a queue

SYNOPSIS

```
#include <search.h>
void insque(void *element, void *pred);
void remque(void *element);
```

DESCRIPTION

The insque() and remque() functions shall manipulate queues built from doubly-linked lists. The queue can be either circular or linear. An application using insque() or remque() shall ensure it defines a struc? ture in which the first two members of the structure are pointers to the same type of structure, and any further members are application-specific. The first member of the structure is a forward pointer to the next entry in the queue. The second member is a backward pointer to the previous entry in the queue. If the queue is linear, the queue is ter? minated with null pointers. The names of the structure and of the pointer members are not subject to any special restriction.

The insque() function shall insert the element pointed to by element into a queue immediately after the element pointed to by pred.

The remque() function shall remove the element pointed to by element from a queue.

If the queue is to be used as a linear list, invoking insque(&element, NULL), where element is the initial element of the queue, shall ini? tialize the forward and backward pointers of element to null pointers.

If the queue is to be used as a circular list, the application shall ensure it initializes the forward pointer and the backward pointer of the initial element of the queue to the element's own address.

RETURN VALUE

The insque() and remque() functions do not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Creating a Linear Linked List

The following example creates a linear linked list.

```
#include <search.h>
...
struct myque element1;
struct myque element2;
char *data1 = "DATA1";
char *data2 = "DATA2";
...
element1.data = data1;
element2.data = data2;
insque (&element1, NULL);
insque (&element2, &element1);
```

Creating a Circular Linked List

The following example creates a circular linked list.

```
#include <search.h>
...
```

```
struct myque element1;
struct myque element2;
char *data1 = "DATA1";
char *data2 = "DATA2";
...
element1.data = data1;
element2.data = data2;
element1.fwd = &element1;
element1.bck = &element1;
insque (&element2, &element1);
```

Removing an Element

The following example removes the element pointed to by element1.

```
#include <search.h>
...
struct myque element1;
...
remque (&element1);
```

APPLICATION USAGE

The historical implementations of these functions described the arguments as being of type struct qelem * rather than as being of type void * as defined here. In those implementations, struct qelem was commonly defined in <search.h> as:

```
struct qelem {
    struct qelem  *q_forw;
    struct qelem  *q_back;
};
```

Applications using these functions, however, were never able to use this structure directly since it provided no room for the actual data contained in the elements. Most applications defined structures that contained the two pointers as the initial elements and also provided space for, or pointers to, the object's data. Applications that used these functions to update more than one type of table also had the problem of specifying two or more different structures with the same

name, if they literally used struct qelem as specified.

As described here, the implementations were actually expecting a struc? ture type where the first two members were forward and backward point? ers to structures. With C compilers that didn't provide function proto? types, applications used structures as specified in the DESCRIPTION above and the compiler did what the application expected.

If this method had been carried forward with an ISO C standard compiler and the historical function prototype, most applications would have to be modified to cast pointers to the structures actually used to be pointers to struct qelem to avoid compilation warnings. By specifying void * as the argument type, applications do not need to change (unless they specifically referenced struct qelem and depended on it being de? fined in <search.h>).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

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

COPYRIGHT