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 'CIRCLEQ_FOREACH_REVERSE.3' command

$ man CIRCLEQ_FOREACH_REVERSE.3

CIRCLEQ(3) Linux Programmer's Manual CIRCLEQ(3)

NAME

CIRCLEQ_EMPTY, CIRCLEQ_ENTRY, CIRCLEQ_FIRST, CIRCLEQ_FOREACH, CIR?

CLEQ_FOREACH_REVERSE, CIRCLEQ_HEAD, CIRCLEQ_HEAD_INITIALIZER, CIR?

CLEQ_INIT, CIRCLEQ_INSERT_AFTER, CIRCLEQ_INSERT_BEFORE, CIRCLEQ_IN?

SERT_HEAD, CIRCLEQ_INSERT_TAIL, CIRCLEQ_LAST, CIRCLEQ_LOOP_NEXT, CIR?

CLEQ_LOOP_PREV, CIRCLEQ_NEXT, CIRCLEQ_PREV, CIRCLEQ_REMOVE - implemen?

tation of a doubly linked circular queue

SYNOPSIS

```
#include <sys/queue.h>

int CIRCLEQ_EMPTY(CIRCLEQ_HEAD *head);

CIRCLEQ_ENTRY(TYPE);

struct TYPE *CIRCLEQ_FIRST(CIRCLEQ_HEAD *head);

CIRCLEQ_FOREACH(struct TYPE *var, CIRCLEQ_HEAD *head,
        CIRCLEQ_ENTRY NAME);

CIRCLEQ_FOREACH_REVERSE(struct TYPE *var, CIRCLEQ_HEAD *head,
        CIRCLEQ_ENTRY NAME);

CIRCLEQ_HEAD(HEADNAME, TYPE);

CIRCLEQ_HEAD CIRCLEQ_HEAD_INITIALIZER(CIRCLEQ_HEAD head);

void CIRCLEQ_INIT(CIRCLEQ_HEAD *head);

void CIRCLEQ_INSERT_AFTER(CIRCLEQ_HEAD *head, struct TYPE *listelm,
        struct TYPE *elm, CIRCLEQ_ENTRY NAME);

void CIRCLEQ_INSERT_BEFORE(CIRCLEQ_HEAD *head, struct TYPE *listelm,
```

struct TYPE *elm, CIRCLEQ_ENTRY NAME);

void CIRCLEQ_INSERT_HEAD(CIRCLEQ_HEAD *head, struct TYPE *elm,
CIRCLEQ_ENTRY NAME);

void CIRCLEQ_INSERT_TAIL(CIRCLEQ_HEAD *head, struct TYPE *elm,
CIRCLEQ_ENTRY NAME);

struct TYPE *CIRCLEQ_LAST(CIRCLEQ_HEAD *head);

void CIRCLEQ_LOOP_NEXT(CIRCLEQ_HEAD *head, struct TYPE *elm,
CIRCLEQ_ENTRY NAME);

void CIRCLEQ_LOOP_PREV(CIRCLEQ_HEAD *head, struct TYPE *elm,
CIRCLEQ_ENTRY NAME);

struct TYPE *CIRCLEQ_NEXT(struct TYPE *elm, CIRCLEQ_ENTRY NAME);

struct TYPE *CIRCLEQ_PREV(struct TYPE *elm, CIRCLEQ_ENTRY NAME);

void CIRCLEQ_REMOVE(CIRCLEQ_HEAD *head, struct TYPE *elm,
CIRCLEQ_ENTRY NAME);

DESCRIPTION

These macros define and operate on doubly linked circular queues.

In the macro definitions, TYPE is the name of a user-defined structure, that must contain a field of type CIRCLEQ_ENTRY, named NAME. The argu? ment HEADNAME is the name of a user-defined structure that must be de? clared using the macro CIRCLEQ_HEAD().

A circular queue is headed by a structure defined by the CIRCLEQ_HEAD() macro. This structure contains a pair of pointers, one to the first element in the circular queue and the other to the last element in the circular queue. The elements are doubly linked so that an arbitrary element can be removed without traversing the circular queue. New ele? ments can be added to the circular queue after an existing element, be? fore an existing element, at the head of the circular queue, or at the end of the circular queue. A CIRCLEQ_HEAD structure is declared as follows:

CIRCLEQ_HEAD(HEADNAME, TYPE) head;

where struct HEADNAME is the structure to be defined, and struct TYPE is the type of the elements to be linked into the circular queue. A pointer to the head of the circular queue can later be declared as:

struct HEADNAME *headp;

(The names head and headp are user selectable.)

The macro CIRCLEQ_HEAD_INITIALIZER() evaluates to an initializer for the circular queue head.

The macro CIRCLEQ_EMPTY() evaluates to true if there are no items on the circular queue.

The macro CIRCLEQ_ENTRY() declares a structure that connects the ele? ments in the circular queue.

The macro CIRCLEQ_FIRST() returns the first item on the circular queue.

The macro CIRCLEQ_FOREACH() traverses the circular queue referenced by head in the forward direction, assigning each element in turn to var. var is set to &head if the loop completes normally, or if there were no elements.

The macro CIRCLEQ_FOREACH_REVERSE() traverses the circular queue refer? enced by head in the reverse direction, assigning each element in turn to var.

The macro CIRCLEQ_INIT() initializes the circular queue referenced by head.

The macro CIRCLEQ_INSERT_HEAD() inserts the new element elm at the head of the circular queue.

The macro CIRCLEQ_INSERT_TAIL() inserts the new element elm at the end of the circular queue.

The macro CIRCLEQ_INSERT_AFTER() inserts the new element elm after the element listelm.

The macro CIRCLEQ_INSERT_BEFORE() inserts the new element elm before the element listelm.

The macro CIRCLEQ_LAST() returns the last item on the circular queue.

The macro CIRCLEQ_NEXT() returns the next item on the circular queue, or &head if this item is the last one.

The macro CIRCLEQ_PREV() returns the previous item on the circular queue, or &head if this item is the first one.

The macro CIRCLEQ_LOOP_NEXT() returns the next item on the circular queue. If elm is the last element on the circular queue, the first el?

ement is returned.

The macro CIRCLEQ_LOOP_PREV() returns the previous item on the circular queue. If elm is the first element on the circular queue, the last el? ement is returned.

The macro CIRCLEQ_REMOVE() removes the element elm from the circular queue.

RETURN VALUE

CIRCLEQ_EMPTY() returns nonzero if the queue is empty, and zero if the queue contains at least one entry.

CIRCLEQ_FIRST(), CIRCLEQ_LAST(), CIRCLEQ_NEXT(), and CIRCLEQ_PREV() re? turn a pointer to the first, last, next or previous TYPE structure, re? spectively.

CIRCLEQ_HEAD_INITIALIZER() returns an initializer that can be assigned to the queue head.

CONFORMING TO

Not in POSIX.1, POSIX.1-2001 or POSIX.1-2008. Present on the BSDs (CIRCLEQ macros first appeared in 4.4BSD).

BUGS

The macros CIRCLEQ_FOREACH() and CIRCLEQ_FOREACH_REVERSE() don't allow var to be removed or freed within the loop, as it would interfere with the traversal. The macros CIRCLEQ_FOREACH_SAFE() and CIRCLEQ_FORE? ACH_REVERSE_SAFE(), which are present on the BSDs but are not present in glibc, fix this limitation by allowing var to safely be removed from the list and freed from within the loop without interfering with the traversal.

EXAMPLES

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/queue.h>
struct entry {
    int data;
    CIRCLEQ_ENTRY(entry) entries;          /* Queue. */
```

```
};
CIRCLEQ_HEAD(circlehead, entry);
int
main(void)
{
    struct entry *n1, *n2, *n3, *np;
    struct circlehead head;                 /* Queue head. */
    int i;
    CIRCLEQ_INIT(&head);                    /* Initialize the queue. */
    n1 = malloc(sizeof(struct entry));      /* Insert at the head. */
    CIRCLEQ_INSERT_HEAD(&head, n1, entries);
    n1 = malloc(sizeof(struct entry));      /* Insert at the tail. */
    CIRCLEQ_INSERT_TAIL(&head, n1, entries);
    n2 = malloc(sizeof(struct entry));      /* Insert after. */
    CIRCLEQ_INSERT_AFTER(&head, n1, n2, entries);
    n3 = malloc(sizeof(struct entry));      /* Insert before. */
    CIRCLEQ_INSERT_BEFORE(&head, n2, n3, entries);
    CIRCLEQ_REMOVE(&head, n2, entries);     /* Deletion. */
    free(n2);
                                /* Forward traversal. */
    i = 0;
    CIRCLEQ_FOREACH(np, &head, entries)
        np->data = i++;
                                /* Reverse traversal. */
    CIRCLEQ_FOREACH_REVERSE(np, &head, entries)
        printf("%i\n", np->data);
                                /* Queue deletion. */
    n1 = CIRCLEQ_FIRST(&head);
    while (n1 != (void *)&head) {
        n2 = CIRCLEQ_NEXT(n1, entries);
        free(n1);
        n1 = n2;
    }
```

```
    CIRCLEQ_INIT(&head);
    exit(EXIT_SUCCESS);
}
```

SEE ALSO

insque(3), queue(7)

COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.