NAME

SIMPLEQ_EMPTY, SIMPLEQ_ENTRY, SIMPLEQ_FIRST, SIMPLEQ_FOREACH, SIMPLEQ_HEAD, SIMPLEQ_HEAD_INITIALIZER, SIMPLEQ_INIT, SIMPLEQ_INSERT_AFTER, SIMPLEQ_INSERT_HEAD, SIMPLEQ_INSERT_TAIL, SIMPLEQ_NEXT, SIMPLEQ_REMOVE, SIMPLEQ_REMOVE_HEAD, STAILQ_CONCAT, STAILQ_EMPTY, STAILQ_ENTRY, STAILQ_FIRST, STAILQ_FOREACH, STAILQ_HEAD, STAILQ_HEAD_INITIALIZER, STAILQ_INIT, STAILQ_INSERT_AFTER, STAILQ_INSERT_HEAD, STAILQ_INSERT_TAIL, STAILQ_NEXT, STAILQ_REMOVE, STAILQ_REMOVE_HEAD, − implementation of a singly linked tail queue

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

#include <sys/queue.h>

STAILQ_ENTRY(TYPE);

STAILQ_HEAD(HEADNAME, TYPE);
STAILQ_HEAD STAILQ_HEAD_INITIALIZER(STAILQ_HEAD *head***);**
void STAILQ_INIT(STAILQ_HEAD *****head*);**

int STAILQ_EMPTY(STAILQ_HEAD *****head*);**

void STAILQ_INSERT_HEAD(STAILQ_HEAD *****head*,**
struct TYPE *****elm*, **STAILQ_ENTRY** *NAME*);
void STAILQ_INSERT_TAIL(STAILQ_HEAD *****head*,**
struct TYPE *****elm*, **STAILQ_ENTRY** *NAME*);
void STAILQ_INSERT_AFTER(STAILQ_HEAD *****head*, **struct TYPE *****listelm***,**
struct TYPE *****elm*, **STAILQ_ENTRY** *NAME*);

struct TYPE *STAILQ_FIRST(STAILQ_HEAD *****head*);**
struct TYPE *STAILQ_NEXT(struct TYPE *****elm*, **STAILQ_ENTRY** *NAME*);

STAILQ_FOREACH(struct TYPE *****var*, **STAILQ_HEAD *****head***, **STAILQ_ENTRY** *NAME*);

void STAILQ_REMOVE(STAILQ_HEAD *****head*, **struct TYPE *****elm***, **TYPE,**
STAILQ_ENTRY *NAME*);
void STAILQ_REMOVE_HEAD(STAILQ_HEAD *****head*,**
STAILQ_ENTRY *NAME*);

void STAILQ_CONCAT(STAILQ_HEAD *****head1*, **STAILQ_HEAD *****head2***);**
Note: Identical macros prefixed with SIMPLEQ instead of STAILQ exist; see NOTES.

DESCRIPTION

These macros define and operate on singly linked tail queues.

In the macro definitions,*TYPE* is the name of a user -defined structure, that must contain a field of type *STAILQ_ENTRY*, named *NAME*. The argument *HEADNAME* is the name of a user-defined structure that must be declared using the macro **STAILQ_HEAD**().

Creation

A singly linked tail queue is headed by a structure defined by the **STAILQ_HEAD**() macro. This structure contains a pair of pointers, one to the first element in the tail queue and the other to the last element in the tail queue. The elements are singly linked for minimum space and pointer manipulation overhead at the expense of O(n) removal for arbitrary elements. New elements can be added to the tail queue after an existing element, at the head of the tail queue, or at the end of the tail queue. A *STAILQ_HEAD* structure is declared as follows:

```
STAILQ_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 tail queue. A pointer to the head of the tail queue can later be declared as:

```
struct HEADNAME *headp;
```

(The names *head* and *headp* are user selectable.)

STAILQ_ENTRY() declares a structure that connects the elements in the tail queue.

STAILQ_HEAD_INITIALIZER() evaluates to an initializer for the tail queue *head*.

STAILQ_INIT() initializes the tail queue referenced by *head*.

STAILQ_EMPTY() evaluates to true if there are no items on the tail queue.

Insertion

STAILQ_INSERT_HEAD() inserts the new element *elm* at the head of the tail queue.

STAILQ_INSERT_TAIL() inserts the new element *elm* at the end of the tail queue.

STAILQ_INSERT_AFTER() inserts the new element *elm* after the element *listelm*.

Traversal

STAILQ_FIRST() returns the first item on the tail queue or NULL if the tail queue is empty.

STAILQ_NEXT() returns the next item on the tail queue, or NULL this item is the last.

STAILQ_FOREACH() traverses the tail queue referenced by *head* in the forward direction, assigning each element in turn to *var*.

Removal

STAILQ_REMOVE() removes the element *elm* from the tail queue.

STAILQ_REMOVE_HEAD() removes the element at the head of the tail queue. For optimum efficiency, elements being removed from the head of the tail queue should use this macro explicitly rather than the generic **STAILQ_REMOVE**() macro.

Other features

STAILQ_CONCAT() concatenates the tail queue headed by *head2* onto the end of the one headed by *head1* removing all entries from the former.

RETURN VALUE

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

STAILQ_FIRST(), and **STAILQ_NEXT**() return a pointer to the first or next *TYPE* structure, respectively.

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

STANDARDS

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

NOTES

Some BSDs provide SIMPLEQ instead of STAILQ. They are identical, but for historical reasons they were named differently on different BSDs. STAILQ originated on FreeBSD, and SIMPLEQ originated on NetBSD. For compatibility reasons, some systems provide both sets of macros. glibc provides both STAILQ and SIMPLEQ, which are identical except for a missing SIMPLEQ equivalent to **STAILQ_CONCAT**().

BUGS

STAILQ_FOREACH() doesn't allow *var* to be removed or freed within the loop, as it would interfere with the traversal. **STAILQ_FOREACH_SAFE**(), which is present on the BSDs but is not present in glibc, fixes 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;
    STAILQ_ENTRY(entry) entries;        /* Singly linked tail queue */
};

STAILQ_HEAD(stailhead, entry);

int
main(void)
{
    struct entry *n1, *n2, *n3, *np;
    struct stailhead head;                  /* Singly linked tail queue
                                               head */

    STAILQ_INIT(&head);                     /* Initialize the queue */

    n1 = malloc(sizeof(struct entry));      /* Insert at the head */
    STAILQ_INSERT_HEAD(&head, n1, entries);

    n1 = malloc(sizeof(struct entry));      /* Insert at the tail */
    STAILQ_INSERT_TAIL(&head, n1, entries);

    n2 = malloc(sizeof(struct entry));      /* Insert after */
    STAILQ_INSERT_AFTER(&head, n1, n2, entries);

    STAILQ_REMOVE(&head, n2, entry, entries); /* Deletion */
    free(n2);

    n3 = STAILQ_FIRST(&head);
    STAILQ_REMOVE_HEAD(&head, entries);     /* Deletion from the head */
    free(n3);

    n1 = STAILQ_FIRST(&head);
    n1->data = 0;
    for (unsigned int i = 1; i < 5; i++) {
        n1 = malloc(sizeof(struct entry));
        STAILQ_INSERT_HEAD(&head, n1, entries);
        n1->data = i;
    }
                                            /* Forward traversal */
    STAILQ_FOREACH(np, &head, entries)
        printf("%i\n", np->data);
                                            /* TailQ deletion */
    n1 = STAILQ_FIRST(&head);
    while (n1 != NULL) {
        n2 = STAILQ_NEXT(n1, entries);
        free(n1);
        n1 = n2;
    }
    STAILQ_INIT(&head);

    exit(EXIT_SUCCESS);
}
```

SEE ALSO

insque(3), **queue**(7)