Bounded Buffer

Generated by Doxygen 1.8.17

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

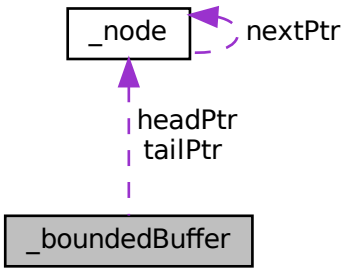
Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 _boundedBuffer Struct Reference

Collaboration diagram for _boundedBuffer:



Public Attributes

- size_t capacity

  A concurrent buffer with limited capacity.
- size_t numElements
- size_t dataSize
- struct _node * headPtr
- struct _node * tailPtr
- pthread_mutex_t mutex
- pthread_cond_t empty
- pthread_cond_t full

3.1.1 Member Data Documentation

3.1.1.1 capacity

```
size_t _boundedBuffer::capacity
```

A concurrent buffer with limited capacity.

Maximum number of elements that can be in the buffer at once

3.1.1.2 dataSize

```
size_t _boundedBuffer::dataSize
```

Size of the data type of the elements in the buffer

3.1.1.3 empty

```
pthread_cond_t _boundedBuffer::empty
```

Condition variable used to track whether the buffer is empty

3.1.1.4 full

```
pthread_cond_t _boundedBuffer::full
```

Condition variable used to track whether the buffer is full

3.1.1.5 headPtr

```
struct _node* _boundedBuffer::headPtr
```

Pointer to first element

3.1.1.6 mutex

```
pthread_mutex_t _boundedBuffer::mutex
```

A mutex for ensuring mutual exclusion access to the buffer

3.1.1.7 numElements

```
size_t _boundedBuffer::numElements
```

Current number of elements in the buffer

3.1.1.8 tailPtr

```
struct _node* _boundedBuffer::tailPtr
```

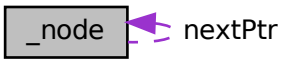
Pointer to last element

The documentation for this struct was generated from the following file:

- boundedbuffer.c

3.2 _node Struct Reference

Collaboration diagram for _node:



Public Attributes

- void * data

  A buffer node.
- struct _node * **nextPtr**

The documentation for this struct was generated from the following file:

- boundedbuffer.c

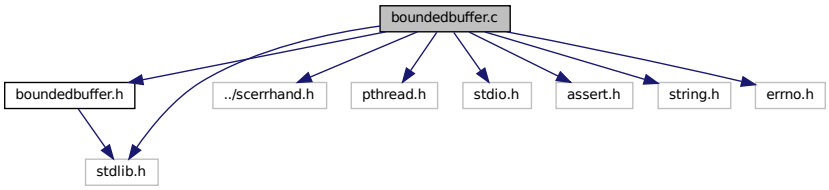
Chapter 4

File Documentation

4.1 boundedbuffer.c File Reference

```
#include "boundedbuffer.h"
#include "../scerrhand.h"
#include <pthread.h>
#include <stdio.h>
#include <assert.h>
#include <string.h>
#include <errno.h>
```

Include dependency graph for boundedbuffer.c:



Classes

- struct _node
- struct _boundedBuffer

Macros

- #define **MIN**(a, b) (a) <= (b) ? (a) : (b)

Functions

- static struct _node * _allocNode (void *data, size_t dataSize)
- BoundedBuffer * allocBoundedBuffer (size_t capacity, size_t dataSize)
- int dequeue (BoundedBuffer *buf, void *dest, size_t destSize)
- int destroyBoundedBuffer (BoundedBuffer *buf)
- int enqueue (BoundedBuffer *buf, void *data)

4.1.1 Function Documentation

4.1.1.1 _allocNode()

```
static struct _node* _allocNode (
            void * data,
            size_t dataSize )  [static]
```

Allocates a new node for the bounded buffer, initializing its value to the given one, and returns it.

Parameters

data	A pointer to the data the new node is going to have
dataSize	The size of the data

Returns

A pointer to the new node

NULL if the node could not be allocated.

4.1.1.2 allocBoundedBuffer()

```
BoundedBuffer* allocBoundedBuffer (
            size_t capacity,
            size_t dataSize )
```

Initializes and returns a new empty buffer with the given capacity.

Parameters

capacity	Maximum capacity of the buffer
dataSize	Size of the elements in the buffer

Returns

A pointer to the newly created buffer upon success, NULL on error (sets `errno`)

Upon error, `errno` will have one of the following values:
`ENOMEM`: memory for the buffer couldn't be allocated
`EINVAL`: invalid parameter(s) were passed

4.1.1.3 dequeue()

```
int dequeue (
            BoundedBuffer * buf,
```

```
            void * dest,
            size_t destSize )
```

Pops the node at the head of the buffer and returns its value. If the buffer is empty, waits until there is at least one element in it.

Parameters

buf	A pointer to the buffer from which to dequeue the node
dest	A pointer to a location to save the popped data. Can be NULL if data isn't to be saved but rather just destroyed

Returns

0 on success, -1 on error (sets `errno`)

Upon error, `errno` will have one of the following values:
`EINVAL`: invalid parameter(s) were passed

4.1.1.4 destroyBoundedBuffer()

```
int destroyBoundedBuffer (
            BoundedBuffer * buf )
```

Frees every remaining element in the buffer, then frees the buffer.

Parameters

buf	Pointer to the buffer to free

Returns

0 on success, -1 on error (sets `errno`)

Upon error, `errno` will have one of the following values:
`EINVAL`: invalid parameter(s) were passed

4.1.1.5 enqueue()

```
int enqueue (
            BoundedBuffer * buf,
            void * data )
```

Allocates a new node with the given value and pushes it to the tail of the bounded buffer.

Parameters

buf	is the buffer the data is going to be pushed to
data	is a pointer to the data to be pushed

If the buffer is full, waits until there is at least one free spot.

Returns

0 on success, -1 if it is unable to allocate memory for the new node.

Allocates a new node with the given value and pushes it to the tail of the bounded buffer. If the buffer is full, waits until there is at least one free spot.

Parameters

buf	is the buffer the data is going to be pushed to
data	is a pointer to the data to be pushed

Returns

0 on success, -1 on error (sets `errno`)

Upon error, `errno` will have one of the following values:
`ENOMEM`: memory for the new node couldn't be allocated
`EINVAL`: invalid parameter(s) were passed

Index