Contiki Libraries

Contiki ringbuf

- Interrupt-safe way to pass bytes from interrupt to non-interrupt code
- Ring buffer size must be even power of two
- Ring buffer can only pass bytes (uint8_t)

ringbuf

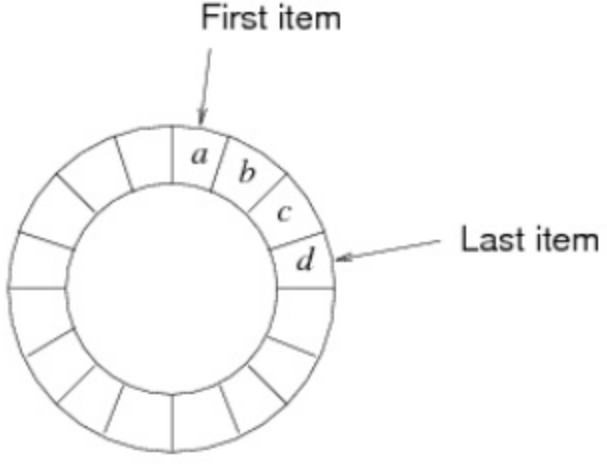
```
Void ringbuf_init (struct ringbuf *r,
         uint8_t *a, uint8_t size_power_of_two);

int ringbuf_put (struct ringbuf *r, uint8_t c);

int ringbuf_get (struct ringbuf *r);

int ringbuf_size (struct ringbuf *r);

int ringbuf_elements (struct ringbuf *r);
```

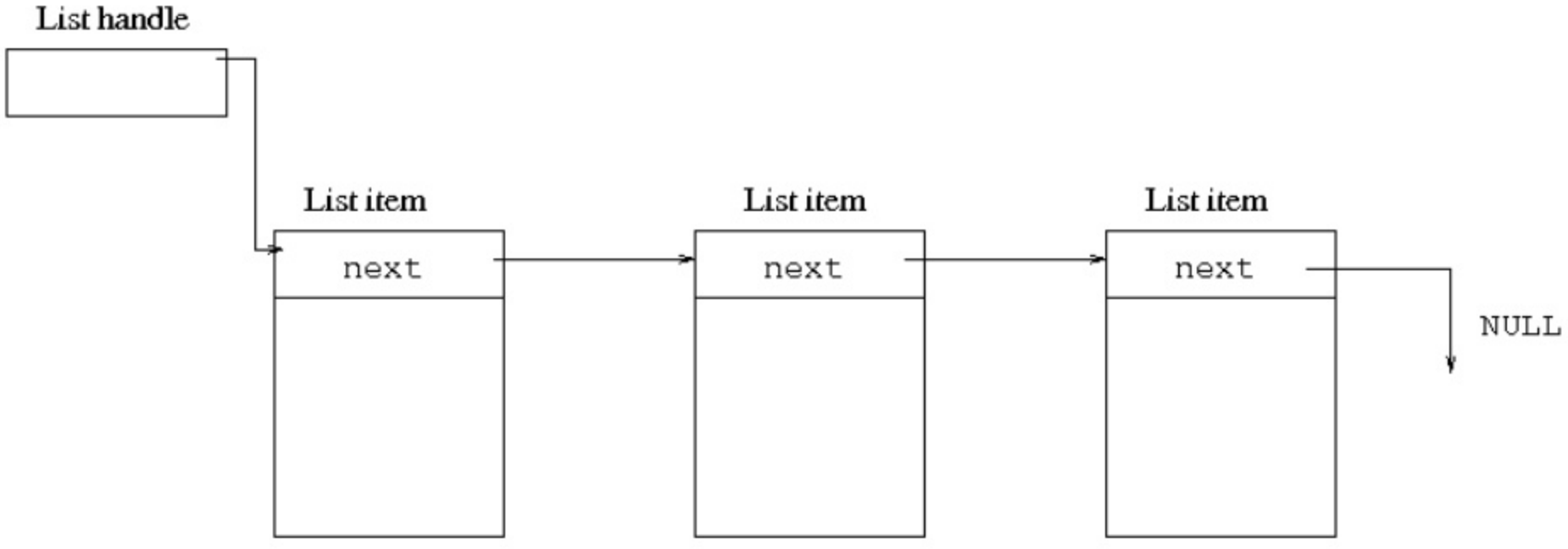


The list library

- Convenient way to keep arbitrary structs on a list
- Only requirement is a **`->next`** pointer at beginning of the struct
- Example:

```
struct my_data {
    struct my_data *next;
    int some_data;
    uint8_t more_data[8];
}
```

The list library



The list library

`void list_init (list_t list)`
Initialize a list.

`void *list_head (list_t list)`
Get a pointer to the first element of a list.

`void list_push (list_t list, void *item)`
Add an item to the start of the list.

`void *list_pop (list_t list)`
Remove the first object on a list.

`void *list_item_next (void *item)`
Get the next item following this item.

The list library

```
#include "lib/list.h"

struct example_list_struct {
  struct *next;
  int number;
};

LIST(example_list);

static void my_function(void) {
  struct example_list_struct *s;

  list_init(example_list);

  list_add(example_list, &element1);
  list_add(example_list, &element2);

  for(s = list_head(example_list); s != NULL; s = list_item_next(s)) {
    printf("List element number %d\n", s->number);
  }
}
```

The memb library

- Manage a chunk of memory
- A fixed set of structs that can be allocated and deallocated
- Size of set specified at compile time

The memb library

- `MEMB(name, structure, num)`
  - Declare a memory block.
- `void memb_init(struct memb *m)`
  - Initialize a memory block.
- `void *memb_alloc(struct memb *m)`
  - Allocate a memory block.
- `int memb_free(struct memb *m, void *ptr)`
  - Free a memory block.

The memb library

```
struct my_structure {
   int a, b;
}

MEMB(my_mem, struct my_structure, NUM_STRUCTS);

static void my_function(void) {
   memb_init(&my_mem);

   struct my_structure *s = memb_alloc(&my_mem);

   memb_free(s);
}
```

Memory allocation with a list

- Typical usage pattern:
  - Allocate memory from memb
  - Maintain on a list
- Makes it easy to keep track of things to deallocate them later

Memory allocation with a list

```
struct example_num {
  struct example_num *next;
  int num;
};

#define MAX_NUMS 16
LIST(num_table);
MEMB(num_mem, struct example_num, MAX_NUMS);

void init_num(void) {
  memb_init(&num_mem);

  list_init(neighbor_table);
}
```

Memory allocation with a list

```
void add_num(int num) {
  e = memb_alloc(&neighbor_mem);
  if(e != NULL) {
    e->num = num;
    list_add(num_table, e);
  }
}

struct example_num {
void remove_num(struct example_num *e) {
  list_remove(num_table, e);
  memb_free(&num_mem, e);
}
```

Memory allocation with a list

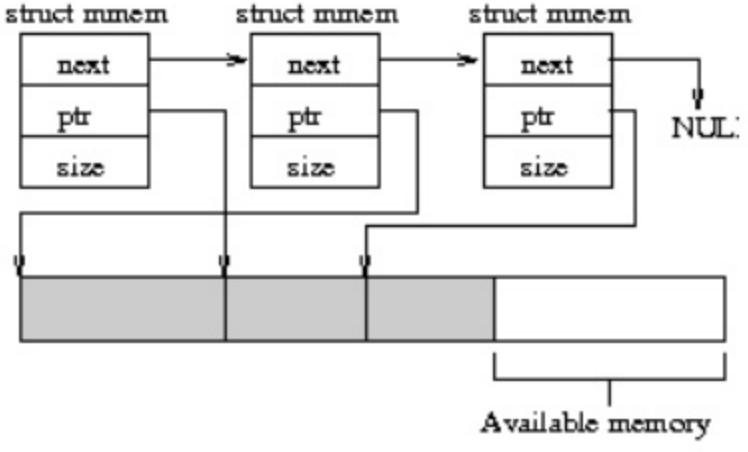
```
struct example_num *find_num(int num) {
  struct example_num *n;

  for(n = list_head(num_table); n != NULL; n = list_element_next(n)) {
    if(n->num == num) {
      return n;
    }
  }
  return NULL;
}
```

The mmem library

- Managed memory allcator
- Maintains a fragmentation-free memory area
- Sometimes useful
- Somewhat tricky to use

The mmem library



The mmem library

- **`MMEM_PTR(m)`**
  - Provide a pointer to managed memory.
- **`int mmem_alloc(struct mmem *m, unsigned int size)`**
  - Allocated managed memory.
- **`void mmem_free(struct mmem *)`**
  - Free managed memory.
- **`void mmem_init(void)`**
  - Initialize the managed memory library.

The lower layers of the netstack

The radio driver

- Input
  - Read packet from the radio into packetbuf
  - Call NETSTACK_RDC.input();
- Output
  - Prepare radio for sending
    - NETSTACK_RADIO.prepare()
  - Send the packet
    - NETSTACK_RADIO.transmit()

Radio driver gotchas

- Radio driver works in two layers
  - Interrupt context
  - Contiki context
- SPI bus must be protected
  - Disable interrupts during an SPI transfer
- Radio core must be protected
  - Maintain flag to avoid interrupts when radio driver is active

Radio driver energy estimator

- The Contiki energest module keeps track of energy consumption
  - `ENERGEST_ON(ENERGEST_TYPE_LISTEN);`
  - `ENERGEST_OFF(ENERGEST_TYPE_LISTEN);`
  - `ENERGEST_ON(ENERGEST_TYPE_TRANSMIT);`
  - `ENERGEST_OFF(ENERGEST_TYPE_TRANSMIT);`

queuebufs

- “Enough” queuebufs are needed
- Difficult to say beforehand how many that is
- Trial and error may be needed

Hands-on: HTTP POST

Build big-red-button.c as a firmware image

- Copy big-red-button.c from demo.thsq.io
- Compile and upload on the hardware
- Connect the button
- Sniff the packets on screen
- Code walk-through

More



http://thingsquare.com