

rbuffer.h, un buffer tournant

Bernard Tatin

2013/2017

Voici un premier essai de *literate programming*, concept inventé par D. Knuth il y a plus de trente ans. À partir de ce seul fichier on génère la documentation et le code. Ici, je reprend du vieux code, cela m'oblige, même s'il est simple, à le repenser et donc, espérons le, à l'améliorer. Même si je passe beaucoup de temps sur la présentation...

Contents

1	rbuffer	1
1.1	premières définitions	1
1.2	la structure	2
1.2.1	les champs	2
1.2.2	remarques diverses	2
1.3	le fonctionnement	3
1.3.1	ajout d'un caractère	3
1.4	le code final	3
2	annexes	3
2.1	la ligne de commande	3
2.2	extraits de code	3
2.3	index	4

1 rbuffer

C'est un buffer tournant le plus simple possible, capable de gérer des lignes délimitées par *LF* (`'\n'`) mais *CR* (`'\r'`) n'est pas pris en compte.

1.1 premières définitions

Pour limiter les calculs, le code..., la taille du buffer est une puissance de 2 d'où la définition du nombre de bits qui ouvre le bal :

```
1  <intro-bits 1>≡  
    #define _RBUFFER_BITS 8
```

```
#define RBUFFER_SIZE    (1 << _RBUFFER_BITS)
#define RBUFFER_MASK    (RBUFFER_SIZE - 1)
```

This code is used in chunk 3a.

Defines:

```
_RBUFFER_BITS, never used.
RBUFFER_MASK, never used.
RBUFFER_SIZE, used in chunk 2a.
```

1.2 la structure

Note: tous les membres de la structure sont définis comme volatile. C'est important dans un système embarqué avec des interruptions pouvant manipuler le buffer. Sans volatile, une optimisation trop agressive pourrait placer une des valeurs entières dans un registre. En cas d'interruption modifiant cette valeur, le registre, lui, ne bougera pas et des caractères pourraient se perdre.

```
2a  <tsrbuffer 2a>≡
    /**
     * @struct TSrbuffer
     * La structure gérant le buffer tournant.
     */
    typedef struct {
        volatile int in;
        volatile int out;
        volatile int line_count;
        volatile char buffer[RBUFFER_SIZE];
    } TSrbuffer;
```

This code is used in chunk 3a.

Defines:

```
TSrbuffer, never used.
```

Uses RBUFFER_SIZE 1.

1.2.1 les champs

1.2.2 remarques diverses

On pourrait définir un VOLATILE en fonction de l'architecture du type :

```
2b  <define-volatile 2b>≡
    #if defined(__with_irqs)
        #define VOLATILE volatile
    #else
        #define VOLATILE
    #endif
```

Root chunk (not used in this document).

1.3 le fonctionnement

1.3.1 ajout d'un caractère

Le fonctionnement est le suivant pour l'ajout d'un caractère :

- on place le caractère dans le buffer à la position `in`,
- on incrémente `in`,
- si on atteint la limite du buffer, on positionne `in` à 0,
- si le caractère est `'\n'`, on incrémente `line_count`.

1.4 le code final

3a `<* 3a>≡`
`<intro-bits 1>`
`<tsrbuffer 2a>`

Root chunk (not used in this document).

2 annexes

2.1 la ligne de commande

Pour obtenir le fichier \LaTeX et le code source, voici ce qu'il faut faire depuis un terminal :

3b `<command-line 3b>≡`
`# fichier LaTeX`
`noweave -delay -index rbuffer.nw > rbuffer.tex`
`# fichier PDF`
`pdflatex rbuffer.tex && \`
`pdflatex rbuffer.tex && \`
`pdflatex rbuffer.tex`
`# le code source`
`notangle rbuffer.nw > rbuffer.h`

Root chunk (not used in this document).

2.2 extraits de code

`<* 3a> 3a`
`<command-line 3b> 3b`
`<define-volatile 2b> 2b`
`<intro-bits 1> 1, 3a`
`<tsrbuffer 2a> 2a, 3a`

2.3 index

`_RBUFFER_BITS`: [1](#)
`RBUFFER_MASK`: [1](#)
`RBUFFER_SIZE`: [1](#), [2a](#)
`TSrbuffer`: [2a](#)