

rbuffer, 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...

Ce code est très orienté *ligne de caractères* et a servi, entre autre, à la gestion de modems sur des systèmes embarqués. On notera l'absence de gestion de trop plein du buffer, *i.e.* de l'écrasement de caractères lors du remplissage. Certains systèmes embarqués m'en ont découragés par manque de mémoire et une commande de modem écrasée était une commande modem mal formée... Jeu dangereux qui a finalement bien fonctionné.

Pour finir cette présentation, les fichiers créés à partir du document maître comme les sources et le PDF sont inclus dans ce dépôt pour permettre une visualisation simple des résultats obtenus sans avoir à installer quoique ce soit de spécial dont **noweb** et L^AT_EX.

Contents

1	rbuffer	2
1.1	premières définitions	2
1.2	la structure	3
1.2.1	les champs	3
1.2.2	remarques diverses	3
1.3	le fonctionnement	5
1.3.1	types et modificateurs	5
1.3.2	ajout d'un caractère	6
1.4	le code final	10
1.4.1	standard.h	10
1.4.2	rbuffer.h	11
1.4.3	rbuffer.c	11
2	annexes	12
2.1	la ligne de commande	12

3	tables et index	13
3.1	table des extraits de code	13
3.2	index	13

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, plus exactement, il est rejeté.

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, et en tenant compte du fait que cette définition peut-être donnée en paramètre du préprocesseur :

2a `<intro-bits 2a>≡`
`# if !defined(_RBUFFER_BITS)`
`#define _RBUFFER_BITS 8`
`#endif`

This definition is continued in chunk 2.

This code is used in chunk 11a.

Defines:

`_RBUFFER_BITS`, used in chunk 2b.

La taille du buffer sera donc :

2b `<intro-bits 2a>+≡`
`#define RBUFFER_SIZE (1 << _RBUFFER_BITS)`

This code is used in chunk 11a.

Defines:

`RBUFFER_SIZE`, used in chunks 2–4.

Uses `_RBUFFER_BITS` 2a.

Et le masque permettant un rapide *modulo* arithmétique avec un **and** binaire :

2c `<intro-bits 2a>+≡`
`#define RBUFFER_MASK (RBUFFER_SIZE - 1)`

This code is used in chunk 11a.

Defines:

`RBUFFER_MASK`, used in chunks 6b and 7a.

Uses `RBUFFER_SIZE` 2b.

1.2 la structure

La voici :

3a $\langle tsrbuffer\ 3a \rangle \equiv$

```
typedef struct {
    volatile int in;
    volatile int out;
    volatile int line_count;
    volatile char buffer[RBUFFER_SIZE];
} TSrbuffer;
```

Root chunk (not used in this document).

Defines:

TSrbuffer, used in chunks 6–9.

Uses **RBUFFER_SIZE** 2b.

1.2.1 les champs

1.2.2 remarques diverses

Tous les membres de la structure sont définis comme **volatile int**. 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. On pourrait définir un **VOLATILE** en fonction de l'architecture du type :

3b $\langle define-volatile\ 3b \rangle \equiv$

```
#if defined(__with_irqs)
    #define VOLATILE volatile
#else
    #define VOLATILE
#endif
```

This code is used in chunk 11a.

Defines:

__with_irqs, , never used.

VOLATILE, used in chunk 4b.

L'utilisation du type **int** permet d'utiliser le type entier permettant en général le meilleur compromis vitesse/taille. Mais ce n'est pas toujours vrai, tout dépend de l'architecture du processeur et des choix des concepteurs du compilateur. On va donc utiliser un **define** ce qui autorise la définition sur la ligne de commande du compilateur, contrairement au **typedef**:

4a $\langle \text{define-int } 4a \rangle \equiv$

```
#if !defined(INT)
#define INT int
#endif
```

This code is used in chunk 11a.

Defines:

INT, used in chunks 4b and 7–9.

Ce qui donnerait au final :

4b $\langle \text{tsrbuffer-final } 4b \rangle \equiv$

```
typedef struct {
    VOLATILE INT in;
    VOLATILE INT out;
    VOLATILE INT line_count;
    VOLATILE char buffer[RBUFFER_SIZE];
} TSrbuffer;
```

This code is used in chunk 11a.

Defines:

TSrbuffer, used in chunks 6–9.

Uses **INT** 4a 4a, **RBUFFER_SIZE** 2b, and **VOLATILE** 3b.

Quoiqu'il en soit, il est fortement recommandé de lire la définition exacte du `VOLATILE` de votre compilateur, certaines variations pouvant rendre votre code totalement inefficace. Et d'autant plus que votre compilateur cible un système embarqué où les variations autour des standards sont choses communes.

Cependant, nous ne résolvons pas tous les problèmes, en particuliers ceux du *multi-threading* qui sont laissés à l'utilisateur dans cette version.

1.3 le fonctionnement

1.3.1 types et modificateurs

On utilise `bool` qui n'est pas défini avant *C11* :

```
5 <type-bool 5>≡
  #ifndef no_c11
    #include <stdbool.h>
  #else
    typedef enum {
      false = 0,
      true = 1
    } bool;
  #ifndef no_inline
    #define no_inline
  #endif
  #endif
```

This code is used in chunk 10.

Defines:

- `bool`, never used.
- `false`, never used.
- `no_inline`, used in chunk 6a.
- `true`, never used.

Certaines fonctions sont **inline**. La diversité des compilateurs nous obligent à définir un **INLINE** ainsi (et nous ne couvrons pas tous les cas, loin de là):

```
6a  <define-inline 6a>≡
    #if defined(with_watcominline)
        #define INLINE __inline
    #elif !defined(no_inline)
        #define INLINE inline
    #else
        #define INLINE
    #endif
```

This code is used in chunk 10.

Defines:

INLINE, used in chunks 6 and 7.

Uses **no_inline** 5.

1.3.2 ajout d'un caractère

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

- si le caractère est '\r', on ne fait rien,
- 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**.

```
6b  <add-char 6b>≡
    static INLINE void rbf_add_char(TSrbuffer *rb, const char c) {
        if (c != '\r') {
            rb->buffer[rb->in++] = c;
            rb->in &= RBUFFER_MASK;
            if (c == '\n') {
                rb->line_count++;
            }
        }
    }
```

This code is used in chunk 11a.

Defines:

rbf_add_char, used in chunk 8b.

Uses **INLINE** 6a, **RBUFFER_MASK** 2c, and **TSrbuffer** 3a 4b 4b.

La récupération d'un caractère dans le buffer est l'algorithme inverse :

7a $\langle \text{get-char } 7a \rangle \equiv$

```
static inline char rbf_get_char(TSrbuffer *rb) {
    INT out = rb->out;
    char c = rb->buffer[out++];
    out &= RBUFFER_MASK;
    rb->out = out;
    if (c == '\n' && rb->line_count) {
        rb->line_count--;
    }
    return c;
}
```

This code is used in chunk 11a.

Defines:

rbf_get_char, used in chunk 9.

Uses **INLINE** 6a, **INT** 4a 4a, **RBUFFER_MASK** 2c, and **TSrbuffer** 3a 4b 4b.

Il est cependant très important de déterminer si des caractères sont présents dans le buffer :

7b $\langle \text{has-chars } 7b \rangle \equiv$

```
static inline bool rbf_has_chars(TSrbuffer *rb) {
    return rb->in != rb->out;
}
```

This code is used in chunk 11a.

Defines:

bool, never used.

Uses **INLINE** 6a and **TSrbuffer** 3a 4b 4b.

Le marquage d'une fin de ligne se fait par un '\0' :

7c $\langle \text{end-of-line } 7c \rangle \equiv$

```
static inline void rbf_end_of_line(TSrbuffer *rb) {
    rb->buffer[rb->in] = 0;
    rb->line_count++;
}
```

Root chunk (not used in this document).

Defines:

rbf_end_of_line, used in chunk 8b.

Uses **INLINE** 6a and **TSrbuffer** 3a 4b 4b.

Ces fonctions, nécessitant une boucle, ne sont pas déclarées `INLINE` :

8a `<more-functions-h 8a>≡`
 void rbf_add_line(TSrbuffer *rb, char *line);
 INT rbf_get_line(TSrbuffer *rb, char *line);
This code is used in chunk 11a.
Uses **INT** 4a 4a, **rbf_add_line** 8b, **rbf_get_line** 9, and **TSrbuffer** 3a 4b 4b.

L'ajout d'une ligne est *simple* :

8b `<more-functions-c 8b>≡`
 void rbf_add_line(TSrbuffer *rb, char *line) {
 char c;

 while ((c = *(line++)) != 0) {
 rbf_add_char(rb, c);
 }
 rbf_end_of_line(rb);
 }
This definition is continued in chunk 9.
This code is used in chunk 11b.
Defines:
 rbf_add_line, used in chunk 8a.
Uses **rbf_add_char** 6b, **rbf_end_of_line** 7c, and **TSrbuffer** 3a 4b 4b.

Et la lecture d'une ligne :

9 *<more-functions-c 8b>+≡*

```
INT rbf_get_line(TSrbuffer *rb, char *line) {
    char c;
    INT r = 0;

    while (rbf_has_chars(rb)) {
        c = rbf_get_char(rb);

        if (c == '\n') {
            break;
        }
        if (c != 0) {
            *(line++) = c;
        }
        r++;
    }
    *line = 0;
    return r;
}
```

This code is used in chunk 11b.

Defines:

rbf_get_line, used in chunk 8a.

Uses **INT** 4a 4a, **rbf_get_char** 7a, and **TSrbuffer** 3a 4b 4b.

1.4 le code final

1.4.1 standard.h

10 $\langle standard.h\ 10 \rangle \equiv$

```
/*
 * _standard.h
 * generated by noweb
 */
#ifndef INCLUDE__COMPAT__STANDARD_H_
#define INCLUDE__COMPAT__STANDARD_H_

#include <stdlib.h>
#include <stdarg.h>

 $\langle type\text{-}bool\ 5 \rangle$ 

 $\langle define\text{-}inline\ 6a \rangle$ 

#endif /* INCLUDE__COMPAT__STANDARD_H_ */
```

Root chunk (not used in this document).
Defines:
INCLUDE__COMPAT__STANDARD_H_, never used.

1.4.2 rbuffer.h

11a `<rbuffer.h 11a>≡`
`/*`
 `* rbuffer.h`
 `* generated by noweb`
 `*/`

`#if !defined(__rbuffer_h__)`
`#define __rbuffer_h__`

`<intro-bits 2a>`

`<define-volatile 3b>`
`<define-int 4a>`

`<tsrbuffer-final 4b>`

`<add-char 6b>`

`<get-char 7a>`

`<has-chars 7b>`

`<more-functions-h 8a>`

`#endif // __rbuffer_h__`
 Root chunk (not used in this document).
 Defines:
 `__rbuffer_h__`, never used.

1.4.3 rbuffer.c

11b `<rbuffer.c 11b>≡`
`/*`
 `* rbuffer.c`
 `* generated by noweb`
 `*/`

`#include "rbuffer.h"`

`<more-functions-c 8b>`
 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 :

```
12 <command-line 12>≡  
# fichier LaTeX  
noweave -delay -autodefs c -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).

L'option `-autodefs c` permet à `noweave` de déterminer lui-même les éléments du langage C. Sans cette option, dans le cadre de ce fichier, les définitions de `intro-bits` ne seraient pas visibles.

3 tables et index

3.1 table des extraits de code

`<add-char 6b>` [6b](#), [11a](#)
`<command-line 12>` [12](#)
`<define-inline 6a>` [6a](#), [10](#)
`<define-int 4a>` [4a](#), [11a](#)
`<define-volatile 3b>` [3b](#), [11a](#)
`<end-of-line 7c>` [7c](#)
`<get-char 7a>` [7a](#), [11a](#)
`<has-chars 7b>` [7b](#), [11a](#)
`<intro-bits 2a>` [2a](#), [2b](#), [2c](#), [11a](#)
`<more-functions-c 8b>` [8b](#), [9](#), [11b](#)
`<more-functions-h 8a>` [8a](#), [11a](#)
`<rbuffer.c 11b>` [11b](#)
`<rbuffer.h 11a>` [11a](#)
`<standard.h 10>` [10](#)
`<tsrbuffer 3a>` [3a](#)
`<tsrbuffer-final 4b>` [4b](#), [11a](#)
`<type-bool 5>` [5](#), [10](#)

3.2 index

`__rbuffer_h__`: [11a](#)
`__with_irqs,` [3b](#)
`_RBUFFER_BITS`: [2a](#), [2b](#)
`bool`: [5](#), [5](#), [7b](#)
`false`: [5](#)
`INCLUDE__COMPAT__STANDARD_H_`: [10](#)
`INLINE`: [6a](#), [6b](#), [7a](#), [7b](#), [7c](#)
`INT`: [4a](#), [4a](#), [4b](#), [7a](#), [8a](#), [9](#)
`no_inline`: [5](#), [6a](#)
`rbf_add_char`: [6b](#), [8b](#)
`rbf_add_line`: [8a](#), [8b](#)
`rbf_end_of_line`: [7c](#), [8b](#)
`rbf_get_char`: [7a](#), [9](#)
`rbf_get_line`: [8a](#), [9](#)
`RBUFFER_MASK`: [2c](#), [6b](#), [7a](#)
`RBUFFER_SIZE`: [2b](#), [2c](#), [3a](#), [4b](#)
`true`: [5](#)

TSrbuffer: 3a, 4b, 4b, 6b, 7a, 7b, 7c, 8a, 8b, 9
VOLATILE: 3b, 4b