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

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 rejetté.

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 :

2a ⟨*intro-bits* 2a⟩≡
```
#define _RBUFFER_BITS   8
#define RBUFFER_SIZE    (1 << _RBUFFER_BITS)
#define RBUFFER_MASK    (RBUFFER_SIZE - 1)
```
This code is used in chunk 4a.
Defines:
_RBUFFER_BITS, never used.
RBUFFER_MASK, used in chunk 3b.
RBUFFER_SIZE, used in chunks 2b and 3a.

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.

2b ⟨*tsrbuffer* 2b⟩≡
```
/**
 * @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;
```
Root chunk (not used in this document).
Defines:
TSrbuffer, used in chunk 3b.
Uses RBUFFER_SIZE 2a 2a.

1.2.1 les champs

1.2.2 remarques diverses

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

2c ⟨*define-volatile* 2c⟩≡
```
#if defined(__with_irqs)
  #define VOLATILE volatile
#else
  #define VOLATILE
#endif
```
This code is used in chunk 4a.

Ce qui donnerait au final :

3a ⟨*tsrbuffer-final* 3a⟩≡
```
/**
 * @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 4a.
Defines:
TSrbuffer, used in chunk 3b.
Uses RBUFFER_SIZE 2a 2a.

1.3 le fonctionnement

1.3.1 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`.

3b ⟨*add-char* 3b⟩≡
```
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 4a.
Defines:
rbf_add_char, never used.
Uses RBUFFER_MASK 2a 2a and TSrbuffer 2b 2b 3a 3a.

1.4 le code final

4a ⟨* 4a⟩≡
⟨*intro-bits* 2a⟩
⟨*define-volatile* 2c⟩
⟨*tsrbuffer-final* 3a⟩
⟨*add-char* 3b⟩

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 :

4b ⟨*command-line* 4b⟩≡

```
# 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).

3 tables et index

3.1 table des extraits de code

3.2 index