【注意事項】
RX ファミリ用 C/C++コンパイラパッケージ

R20TS0246JJ0100
Rev.1.00
2017.12.16 号

概要

RX ファミリ用 C/C++コンパイラパッケージ V.1～V.2 の使用上の注意事項を連絡します。

1. 標準ライブラリ関数を用いて文字列を数値に変換した場合の注意事項 (RXC-0046)
2. 文字列操作用ライブラリ関数を使用する場合、またはアセンブラソースで suntil、smovu、scmpu 命令を使用する場合の注意事項 (RXC-0047)

注: 注意事項の後ろの番号は、注意事項の識別番号です。

1. 標準ライブラリ関数を用いて文字列を数値に変換した場合の注意事項 (RXC-0046)

1.1 該当製品

RX ファミリ用 C/C++コンパイラパッケージ全バージョン

（V.1.00 Release 00～V.2.07 Release 00)

1.2 内容

strtol() などの標準ライブラリ関数で、第 1 引数の文字列を数値に変換した場合に、規格どおりでないポインタが第 2 引数に設定される場合があります。

1.3 発生条件

以下の条件をすべて満たしたときに発生する場合があります。

(1) 次のいずれかの標準ライブラリ関数を使用している。

- strtol()
- strtoll()
- strtoul()
- strtoull()
- strtod()

(2) (1) の第 1 引数に、次の (a)および (b)の条件を満たす文字列を指定している。

(a) 先頭に空白類文字（注 1）が 1 つ以上ある。

(b) (a) の空白類文字（注 1）に続いて整数を構成しない文字（注 2）がある。

注 1：空白 (' ')、水平タブ ('¥t')、改行 ('¥n')、垂直タブ ('¥v')、書式送り ('¥f')、復帰 ('¥r') の文字を示します。

注2：第3引数が10～36 の間のとき、一部または全部のアルファベットが整数を構成する文字として扱われます。

1.4 発生例

発生条件の該当箇所を赤文字で説明しています。

■ 発生ソース例

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
long test;
const char* str = "   Hello World!"; /* (発生条件(2-a)(2-b)) */
void main(void) {
   char* endptr = NULL;
   test=strtol(str, &endptr, 0); /* (発生条件(1)) */
   printf(" str=¥"%s¥"¥n endptr=¥"%s¥"¥n", str, endptr);
}
```

■ 正しい出力結果

仕様上では endptr == str になるようにポインタが設定され、以下のように正しく出力される。

```
 str="   Hello World!"
 endptr="   Hello World!"
```

■ 実際の出力結果

空白を飛ばしたポインタを設定し、以下のように誤って出力される。

```
str="   Hello World!"
endptr="Hello World!"
```

➢ 補足：strtol()の規格

文字列のはじめの数字の部分を base で指定された基数で long 型に変換します。文字列中に変換不可能な文字があった場合には、その文字列へのポインタを endptr に格納します。空白の後が数値であった場合は、数値の後まで読み飛ばしますが、数値でないときは空白を読み飛ばさずに endptr に格納します。

strtol() の詳細については、以下の URL をご参照ください。

https://www.renesas.com/ja-jp/doc/products/tool/doc/009/r20ut3248jj0104_ccrx.pdf
CC-RX コンパイラユーザーズマニュアル
7.4 ライブラリ関数

■ 使用方法

```
long strtol(
    const char *nptr, /* 文字列 */
    char **endptr,
     /*整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ */
    int base /* 基数*/
);
```

■ 使用ソース例

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
long test;
const char* str = "   123 Hello 456 World!";
void main(void) {
    char* endptr = NULL;
    test=strtol(str, &endptr, 0);
    printf(" str=¥"%s¥"¥n endptr=¥"%s¥"¥n test=%ld¥n",str,endptr,test);
}
```

■ 出力結果

```
 str="   123 Hello 456 World!"
 endptr=" Hello 456 World!" /* 数値の後のアドレスがポインタに格納される*/
 test=123/*戻り値として変換された値を long 型で返却*/
```

1.5 回避策

以下の方法により回避可能です。

発生条件 (1) の第 1 引数に渡す文字列の先頭に空白類文字を含めないようにする。

■ 回避例

```
const char* str = "□□□Hello World!"; /*空白を 3 つ含む*/
void main(void) {
    char* endptr = NULL;
    (void)strtol(&str[3], &endptr, 0);
    /*空白を含めないようにすることで 発生条件(2)を回避*/
        ...
```

1.6 恒久対策

改修時期は、現在検討中です。回避策で対応をお願い致します。

2. 文字列操作用ライブラリ関数を使用する場合、またはアセンブラソースで suntil、smovu、scmpu 命令を使用する場合の注意事項 (RXC-0047)

2.1 該当製品

RX ファミリ用 C/C++コンパイラパッケージ全バージョン

（V.1.00 Release 00～V.2.07 Release 00)

2.2 内容

文字列操作用ライブラリ関数またはアセンブラソースで suntil、smovu、scmpu 命令のいずれかを使用したとき、文字列以外の記憶領域にアクセスし、正しく動作しない場合があります。

文字列以外の記憶領域へのアクセス先と動作の詳細は以下のとおりです。

(1) 予約領域へアクセスした場合

不正アドレスアクセスエラーにより割り込みが発生します。

(2) MPU(Memory Protection Unit)で保護された領域へアクセスした場合

データメモリプロテクションエラーにより例外処理が発生します。

2.3 発生条件

以下の条件をすべて満たしている場合に発生することがあります。

(1) (1-1)に示す文字列操作用ライブラリ関数を使用している。またはアセンブラソースでアセンブラ命令を使用している。

(1-1) 文字列操作用ライブラリ関数

memchr()

strlen()

strcmp()

strncmp()

strcpy()

strncpy()

strcat()

strncat()

(1-2) アセンブラ命令

suntil

smovu

scmpu

(2) 以下のいずれかの条件に該当している。

(2-1) (1-1) に示す文字列操作用ライブラリ関数を使用している場合

引数で指定する文字列のアドレス(注) が 4 バイトアライメントではない。

注：文字列のアドレスを指定する引数は以下のとおりです。

文字列操作用ライブラリ関数	文字列のアドレスを指定する引数
memchr(const void *s, long c, size_t n);	s
strlen(const char *s)	s
strcmp(const char *s1, const char *s2)	s1、s2
strncmp(const char *s1, const char *s2, size_t n);	s1、s2
strcpy(char *s1, const char *s2)	s2
strncpy(char *s1, const char *s2, size_t n);	s2
strcat(char *s1, const char *s2)	s1、s2
memchr(const void *s, long c, size_t n);	s

(2-2) (1-2) に示すアセンブラ命令を使用している場合

レジスタで指定する文字列のアドレス (注) が 4 バイトアライメントではない。

注：文字列のアドレスを指定するレジスタ情報は以下のとおりです。

アセンブラ命令	文字列のアドレスを指定するレジスタ情報
suntil	R1 レジスタで示される比較先番地
smovu	R1 で示される転送先番地が対象
scmpu	R1 で示される比較元番地、 R2 で示される比較先番地

(3) (1)の文字列操作用ライブラリ関数またはアセンブラソースでアセンブラ命令を使用した際、

引数で指定する文字列の終端文字（'¥0'）が、予約領域や MPU で保護された領域の直前 3 バイト以内に配置されている。

2.4 発生例

以下に発生例を記します。

- 文字列操作用ライブラリ関数“strlen()”を使用して文字列 s の文字数（文字“ab”）を計算する場合

  (1) 文字列 s の配置アドレスが 4 バイトアライメントではない 0x0d 番地にある。

  (2) 終端文字（'¥0'）が、予約領域の直前 3 バイト以内に配置される。

  (1) (2)の条件を満たし、予約領域や MPU で保護された領域であるアクセス禁止領域の 0x10 番地までアクセスしてしまう。

発生条件の該当箇所を赤文字で説明しています。

```
==========================================================
  size = strlen(s);       //  発生条件(1)

  0x0d 番地      アクセス可領域      'a'              発生条件(2) s のアドレスが 4 バイト
                                                     アライメントではない 0x0d
  0x0e 番地      アクセス可領域      'b'
  0x0f 番地      アクセス可領域      '¥0' (終了条件)  発生条件(3)
  ----- 4 バイト境界 ---
  0x10 番地      アクセス禁止領域（予約領域）
==========================================================
```

2.5 回避策

以下のいずれかの方法で回避可能です。

(1) 発生条件(2)の文字列操作の対象データを、予約領域や MPU で保護された領域の直前に
配置しないように文字列の配置を変更する。

(2) CC-RX V2.07.00 からサポートしている-avoid_cross_boundary_prefetch オプションを使用する。

2.6 恒久対策

改修時期は、現在検討中です。回避策での対応をお願い致します。

以上

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2017.12.16	-	新規発行

ルネサスエレクトロニクス株式会社
〒135-0061　東京都江東区豊洲 3-2-24（豊洲フォレシア）

■総合お問い合わせ先
https://www.renesas.com/contact/