程序 14-24 linux/include/linux/math_emu.h

```
 1 /*
 2  * linux/include/linux/math_emu.h
 3  *
 4  * (C) 1991 Linus Torvalds
 5  */
 6 #ifndef _LINUX_MATH_EMU_H
 7 #define _LINUX_MATH_EMU_H
 8
 9 #include <linux/sched.h>  // 调度程序头文件。定义了任务结构 task_struct、任务 0 的数据，
                             // 还有一些有关描述符参数设置和获取的嵌入式汇编函数宏语句。
10
   // CPU 产生异常中断 int 7 时在栈中分布的数据构成的结构，与系统调用时内核栈中数据分布类似。
11 struct info {
12         long ___math_ret;      // math_emulate()调用者（int7）返回地址。
13         long ___orig_eip;      // 临时保存原 EIP 的地方。
14         long ___edi;           // 异常中断 int7 处理过程入栈的寄存器。
15         long ___esi;
16         long ___ebp;
17         long ___sys_call_ret; // 中断 7 返回时将去执行系统调用的返回处理代码。
18         long ___eax;           // 以下部分（18--30 行）与系统调用时栈中结构相同。
19         long ___ebx;
20         long ___ecx;
21         long ___edx;
22         long ___orig_eax;      // 如不是系统调用而是其它中断时，该值为-1。
23         long ___fs;
24         long ___es;
25         long ___ds;
26         long ___eip;           // 26 -- 30 行 由 CPU 自动入栈。
27         long ___cs;
28         long ___eflags;
29         long ___esp;
30         long ___ss;
31 };
32
   // 为便于引用 info 结构中各字段（栈中数据）所定义的一些常量。
33 #define EAX (info->___eax)
34 #define EBX (info->___ebx)
35 #define ECX (info->___ecx)
36 #define EDX (info->___edx)
37 #define ESI (info->___esi)
38 #define EDI (info->___edi)
39 #define EBP (info->___ebp)
40 #define ESP (info->___esp)
41 #define EIP (info->___eip)
42 #define ORIG_EIP (info->___orig_eip)
43 #define EFLAGS (info->___eflags)
44 #define DS (*(unsigned short *) &(info->___ds))
45 #define ES (*(unsigned short *) &(info->___es))
46 #define FS (*(unsigned short *) &(info->___fs))
47 #define CS (*(unsigned short *) &(info->___cs))
48 #define SS (*(unsigned short *) &(info->___ss))
```

```
49
   // 终止数学协处理器仿真操作。在 math_emulation.c 程序中实现(L488 行）。
   // 下面 52-53 行上宏定义的实际作用是把__math_abort 重新定义为一个不会返回的函数
   // （即在前面加上了 volatile）。该宏的前部分：
   // (volatile void (*)(struct info *,unsigned int))
   // 是函数类型定义，用于重新指明 __math_abort 函数的定义。后面是其相应的参数。
   // 关键词 volatile 放在函数名前来修饰函数，是用来通知 gcc 编译器该函数不会返回，
   // 以让 gcc 产生更好一些的代码。详细说明请参见第 3 章 $3.3.2 节内容。
   // 因此下面的宏定义，其主要目的就是利用__math_abort，让它即可用作普通有返回函数，
   // 又可以在使用宏定义 math_abort() 时用作不返回的函数。
50 void __math_abort(struct info *, unsigned int);
51
52 #define math_abort(x,y) \
53 (((volatile void (*)(struct info *,unsigned int)) __math_abort)((x),(y)))
54
55 /*
56  * Gcc forces this stupid alignment problem: I want to use only two longs
57  * for the temporary real 64-bit mantissa, but then gcc aligns out the
58  * structure to 12 bytes which breaks things in math_emulate.c. Shit. I
59  * want some kind of "no-alignt" pragma or something.
60  */
   /*
    * Gcc 会强迫这种愚蠢的对齐问题：我只想使用两个 long 类型数据来表示 64 比特的
    * 临时实数尾数，但是 gcc 却会将该结构以 12 字节来对齐，这将导致 math_emulate.c
    * 中程序出问题。唉，我真需要某种非对齐“no-align”编译指令。
    */
61
   // 临时实数对应的结构。
62 typedef struct {
63       long a,b;          // 共 64 比特尾数。其中 a 为低 32 位，b 为高 32 位（包括 1 位固定位）。
64       short exponent;    // 指数值。
65 } temp_real;
66
   // 为了解决上面英文注释中所提及的对齐问题而设计的结构，作用同上面 temp_real 结构。
67 typedef struct {
68       short m0,m1,m2,m3;
69       short exponent;
70 } temp_real_unaligned;
71
   // 把 temp_real 类型值 a 赋值给 80387 栈寄存器 b (ST(i))。
72 #define real_to_real(a,b) \
73 ((*(long long *) (b) = *(long long *) (a)),((b)->exponent = (a)->exponent))
74
   // 长实数（双精度）结构。
75 typedef struct {
76       long a,b;          // a 为长实数的低 32 位；b 为高 32 位。
77 } long_real;
78
79 typedef long short_real;  // 定义短实数类型。
80
   // 临时整数结构。
81 typedef struct {
82       long a,b;          // a 为低 32 位；b 为高 32 位。
```

```
 83      short sign;       // 符号标志。
 84 } temp_int;
 85
    // 80387 协处理器内部的状态字寄存器内容对应的结构。（参见图 11-6）
 86 struct swd {
 87      int ie:1;         // 无效操作异常。
 88      int de:1;         // 非规格化异常。
 89      int ze:1;         // 除零异常。
 90      int oe:1;         // 上溢出异常。
 91      int ue:1;         // 下溢出异常。
 92      int pe:1;         // 精度异常。
 93      int sf:1;         // 栈出错标志，表示累加器溢出造成的异常。
 94      int ir:1;         // ir, b: 若上面 6 位任何未屏蔽异常发生，则置位。
 95      int c0:1;         // c0--c3: 条件码比特位。
 96      int c1:1;
 97      int c2:1;
 98      int top:3;        // 指示 80387 中当前位于栈顶的 80 位寄存器。
 99      int c3:1;
100      int b:1;
101 };
102
    // 80387 内部寄存器控制方式常量。
103 #define I387 (current->tss.i387)          // 进程的 80387 状态信息。参见 sched.h 文件。
104 #define SWD (*(struct swd *) &I387.swd)   // 80387 中状态控制字。
105 #define ROUNDING ((I387.cwd >> 10) & 3)   // 取控制字中舍入控制方式。
106 #define PRECISION ((I387.cwd >> 8) & 3)   // 取控制字中精度控制方式。
107
    // 定义精度有效位常量。
108 #define BITS24 0              // 精度有效数：24 位。（参见图 11-6）
109 #define BITS53 2              // 精度有效数：53 位。
110 #define BITS64 3              // 精度有效数：64 位。
111
    // 定义舍入方式常量。
112 #define ROUND_NEAREST  0      // 舍入方式：舍入到最近或偶数。
113 #define ROUND_DOWN     1          // 舍入方式：趋向负无限。
114 #define ROUND_UP       2          // 舍入方式：趋向正无限。
115 #define ROUND_0                3          // 舍入方式：趋向截 0。
116
    // 常数定义。
117 #define CONSTZ   (temp_real_unaligned) {0x0000,0x0000,0x0000,0x0000,0x0000}  // 0
118 #define CONST1   (temp_real_unaligned) {0x0000,0x0000,0x0000,0x8000,0x3FFF}  // 1.0
119 #define CONSTPI  (temp_real_unaligned) {0xC235,0x2168,0xDAA2,0xC90F,0x4000}  // Pi
120 #define CONSTLN2 (temp_real_unaligned) {0x79AC,0xD1CF,0x17F7,0xB172,0x3FFE}  // Loge(2)
121 #define CONSTLG2 (temp_real_unaligned) {0xF799,0xFBCF,0x9A84,0x9A20,0x3FFD}  // Log10(2)
122 #define CONSTL2E (temp_real_unaligned) {0xF0BC,0x5C17,0x3B29,0xB8AA,0x3FFF}  // Log2(e)
123 #define CONSTL2T (temp_real_unaligned) {0x8AFE,0xCD1B,0x784B,0xD49A,0x4000}  // Log2(10)
124
    // 设置 80387 各状态
125 #define set_IE() (I387.swd |= 1)
126 #define set_DE() (I387.swd |= 2)
127 #define set_ZE() (I387.swd |= 4)
128 #define set_OE() (I387.swd |= 8)
129 #define set_UE() (I387.swd |= 16)
```

```
130 #define set_PE() (I387.swd |= 32)
131
    // 设置 80387 各控制条件
132 #define set_C0() (I387.swd |= 0x0100)
133 #define set_C1() (I387.swd |= 0x0200)
134 #define set_C2() (I387.swd |= 0x0400)
135 #define set_C3() (I387.swd |= 0x4000)
136
137 /* ea.c */
138
    // 计算仿真指令中操作数使用到的有效地址值，即根据指令中寻址模式字节计算有效地址值。
    // 参数：__info - 中断时栈中内容对应结构；__code - 指令代码。
    // 返回：有效地址值。
139 char * ea(struct info * __info, unsigned short __code);
140
141 /* convert.c */
142
    // 各种数据类型转换函数。在 convert.c 文件中实现。
143 void short_to_temp(const short_real * __a, temp_real * __b);
144 void long_to_temp(const long_real * __a, temp_real * __b);
145 void temp_to_short(const temp_real * __a, short_real * __b);
146 void temp_to_long(const temp_real * __a, long_real * __b);
147 void real_to_int(const temp_real * __a, temp_int * __b);
148 void int_to_real(const temp_int * __a, temp_real * __b);
149
150 /* get_put.c */
151
    // 存取各种类型数的函数。
152 void get_short_real(temp_real *, struct info *, unsigned short);
153 void get_long_real(temp_real *, struct info *, unsigned short);
154 void get_temp_real(temp_real *, struct info *, unsigned short);
155 void get_short_int(temp_real *, struct info *, unsigned short);
156 void get_long_int(temp_real *, struct info *, unsigned short);
157 void get_longlong_int(temp_real *, struct info *, unsigned short);
158 void get_BCD(temp_real *, struct info *, unsigned short);
159 void put_short_real(const temp_real *, struct info *, unsigned short);
160 void put_long_real(const temp_real *, struct info *, unsigned short);
161 void put_temp_real(const temp_real *, struct info *, unsigned short);
162 void put_short_int(const temp_real *, struct info *, unsigned short);
163 void put_long_int(const temp_real *, struct info *, unsigned short);
164 void put_longlong_int(const temp_real *, struct info *, unsigned short);
165 void put_BCD(const temp_real *, struct info *, unsigned short);
166
167 /* add.c */
168
    // 仿真浮点加法指令的函数。
169 void fadd(const temp_real *, const temp_real *, temp_real *);
170
171 /* mul.c */
172
    // 仿真浮点乘法指令。
173 void fmul(const temp_real *, const temp_real *, temp_real *);
174
```

```
175 /* div.c */
176
    // 仿真浮点除法指令。
177 void fdiv(const temp_real *, const temp_real *, temp_real *);
178
179 /* compare.c */
180
    // 比较函数。
181 void fcom(const temp_real *, const temp_real *);    // 仿真浮点指令 FCOM，比较两个数。
182 void fucom(const temp_real *, const temp_real *);   // 仿真浮点指令 FUCOM，无次序比较。
183 void ftst(const temp_real *);                // 仿真浮点指令 FTST，栈顶累加器与 0 比较。
184
185 #endif
186
```