**体系结构 第四章**

**陈彦帆**

1. 需要。上半部为确定中断类型为网络收包，关闭网卡中断使能，跳转到网卡中断处理程序。处理程序通知NAPI有新任务。 下半部为开中断，采用轮询的方式收包，解码，中断返回。

2. (1) 如下所示，两个线程都想对var加1，期望的结果应该是var被加了两次1。但若运行的顺序为1,4,2,5,3,6, 则var只被加了一次1，结果出错。

// thread 1

1: lw $t0, var

2: addiu $t0, 1

3: sw $t0, var

// thread 2

4: lw $t0, var

5: addiu $t0, 1

6: sw $t0, var

(2)

// thread 2

try:

ll $t0, var

addiu $t0, 1

sc $t0, var

beqz $t0, try

// thread 1

try:

ll $t0, var

addiu $t0, 1

sc $t0, var

beqz $t0, try

3.

编写函数

static int do\_nothing(int a)

{

    return a;

}

然后调用:

do\_nothing(1);

反汇编结果：

(O0):

调用者：

ffffffffa08000b8: 24040001 li a0,1

ffffffffa08000bc: 0c200449 jal ffffffffa0801124 <do\_nothing>

ffffffffa08000c0: 00000000 nop

do\_nothing:

ffffffffa0801124: 67bdffe0 daddiu sp,sp,-32

ffffffffa0801128: ffbe0018 sd s8,24(sp)

ffffffffa080112c: 03a0f02d move s8,sp

ffffffffa0801130: 0080102d move v0,a0

ffffffffa0801134: 00021000 sll v0,v0,0x0

ffffffffa0801138: afc20000 sw v0,0(s8)

ffffffffa080113c: 8fc20000 lw v0,0(s8)

ffffffffa0801140: 03c0e82d move sp,s8

ffffffffa0801144: dfbe0018 ld s8,24(sp)

ffffffffa0801148: 67bd0020 daddiu sp,sp,32

ffffffffa080114c: 03e00008 jr ra

ffffffffa0801150: 00000000 nop

(O1及以上):

调用者：

无

do\_nothing:

ffffffffa0800cc0: 03e00008 jr ra

ffffffffa0800cc4: 0080102d move v0,a0

可以看出，O0无优化时被调函数包括开栈空间，存s8到栈中，更新s8为sp，然后存参数到栈中，再从栈中取参数，计算返回值，把返回值存到栈中，再从栈中取返回值。恢复s8和栈空间，返回。

调用者压入参数，然后jal。

而O1及以上优化，被调函数的以上过程只剩下计算返回值和返回（此处无须开辟栈空间）。

而调用者的调用动作直接被优化省略了。

4.

(1)

#include <stdio.h>

struct tt{

    char a;

    short b;

    int c[2];

    long long d;

};

int main()

{

    printf("SIZE:%lu\n",sizeof(struct tt));

    return 0;

}

OUTPUT:

SIZE:24

(2) 对齐方式为：结构体内每种类型按各自的对齐方式对齐，整个结构体按其中最大的对齐方式对齐。

以下举三个例子具体分析如下：

①

struct tt{

    char a;

    long long d;

    short b;

    int c[2];

};

32

对齐方式：

char 0 long long 8-15 short 16-17 int [2] 20-27

同时，整个结构体按long long 8字节对齐到32-1

②

struct tt{

    char a;

    short b;

    long long d;

    int c[2];

};

24

对齐方式：

char 0 short 1-2 long long 8-15 int [2] 16-23

同时，整个结构体按long long 8字节对齐到24-1

③

struct tt{

    long long d;

    short b;

    char a;

    int c[2];

};

24

对齐方式：

long long 0-7 short 8-9 char 10 int [2] 12-19

同时，整个结构体按long long 8字节对齐到24-1

5.

函数调用：调用者需要保存abi中调用者保存的寄存器，即at, v\*, a\*, t\*,hi,lo寄存器，被调用者需要保存abi中被调用者保存的寄存器即s\* ra, s8, gp,sp寄存器。

系统调用：上下文切换时，需要保存sp, $0, a\*, jp, gp, ra, cp0\_status, cp0\_cause, cp0\_epc。

中断：上下文切换时，需要保存除k\*外的所有通用寄存器和hi,lo, cp0\_status, cp0\_cause, cp0\_epc。

函数调用时，被调用者保存寄存器到栈中。而系统调用和中断中，内核把上下文保存到内核栈中。

函数调用时，被调用者只需保存abi中被调者保存的寄存器。而系统调用中，还需根据约定保存a\*寄存器。由于内核可能发生切换，还需保存cp0的部分寄存器。而中断可能发生在函数的任意位置，故需要保存所有通用寄存器（除k\*外）和部分cp0寄存器。