Linux内核中是怎么实现min和max函数

Linux内核代码有很多很经典的代码，仔细去看看，可以学到很多知识。今天说说Linux是怎么实现min和max的。max和min函数都是比较常用的，可以用函数，或者利用宏去实现，一般我们会这样去写：

```
#define min(x,y) ((x)>(y)?(y):(x))
#define max(x,y) ((x)>(y)?(x):(y))
```

但是上面的写法是有副作用的。比如输入

```
minval = min(x++, y);
```

替换宏之后，代码变成

```
minval = ((x++)>(y)?(y):(x++))
```

可以看出，如果x是最小值，那么它加了两次，很明显是不对的。现在看看Linux内核是怎么实现min和max宏的。

```
/*
 * min()/max() macros that also do
 * strict type-checking.. See the
 * "unnecessary" pointer comparison.
 */
#define min(x, y) ({                    ₩
        typeof(x) _min1 = (x);          ₩
        typeof(y) _min2 = (y);          ₩
        (void) (&_min1 == &_min2);              ₩
        _min1 < _min2 ? _min1 : _min2; })

#define max(x, y) ({                    ₩
        typeof(x) _max1 = (x);          ₩
        typeof(y) _max2 = (y);          ₩
```

```
        (void) (&_max1 == &_max2);              ₩
        _max1 > _max2 ? _max1 : _max2; })

#define min3(x, y, z) ({                        ₩
        typeof(x) _min1 = (x);                  ₩
        typeof(y) _min2 = (y);                  ₩
        typeof(z) _min3 = (z);                  ₩
        (void) (&_min1 == &_min2);              ₩
        (void) (&_min1 == &_min3);              ₩
        _min1 < _min2 ? (_min1 < _min3 ? _min1 : _min3) : ₩
                (_min2 < _min3 ? _min2 : _min3); })

#define max3(x, y, z) ({                        ₩
        typeof(x) _max1 = (x);                  ₩
        typeof(y) _max2 = (y);                  ₩
        typeof(z) _max3 = (z);                  ₩
        (void) (&_max1 == &_max2);              ₩
        (void) (&_max1 == &_max3);              ₩
        _max1 > _max2 ? (_max1 > _max3 ? _max1 : _max3) : ₩
                (_max2 > _max3 ? _max2 : _max3); })
```

里面有很多东西都没见过，但是它能解决传统的min/max宏带来的副作用。下面来一一说一下上面宏的含义：

1. typeof(xxx)含义。
   typeof(xxx)的含义是用来获取xxx的类型，比如上面的

   ```
   typeof(x) _min1 = (x);
   ```

   typeof(x)是获得x的类型，上面的含义相当于

   ```
   int _min1 = (x);
   ```

2. ({XXXX})含义。
   ({XXXX})类似与C中的逗号表达式，XXXX可以包含有多条语句(可以是变量定义、复杂的控制语句)，该表达式的值为XXXX中的最后一条语句的值。比如下面

   ```
   #include <iostream>
   ```

```
using namespace std;

int main(){
    int lastValue = 0;
 lastValue = ({
     int x = 1, y = 10;
     for(int i = 1; i <= y; i++){
      x *= i;
     }
     x;
    });

    cout << lastValue << endl;
    return 0;
}
```

结果是10的阶乘值。

结合了type和({})可以很好的消除宏的副作用。比如上面的

```
minval = min(x++, y);
```

如果代入了Linux内核中的min宏，扩展之后会变成：

```
minval = ({                     ₩
        typeof(x) _min1 = (x++);            ₩
        typeof(y) _min2 = (y);              ₩
        (void) (&_min1 == &_min2);          ₩
        _min1 < _min2 ? _min1 : _min2; })
```

可以看到，x只被加了一次，如果x是最小值，结果是正确的。

本博客文章除特别声明，全部都是原创！
原创文章版权归过往记忆大数据（过往记忆）所有，未经许可不得转载。
本文链接: 【】（）