in include/linux/err.h

```
#define IS_ERR_VALUE(x) unlikely((x) >= (unsigned long)-MAX_ERRNO)

#define MAX_ERRNO    4095
```

(unsigned long)-MAX_ERRNO) = 0xffff,f000

IS_ERR_VALUE(x) macro的逻辑是在返回值(return value)是pointer的情况下，合法的pointer是不可能 >= 0xffff,f000的。

在32-bit kernel情况下，这显然是成立的。另外，kernel的error value只能在1 to MAX_ERRNO之间，那么

0xffff,f000 to 0xffff,ffff (-1 to -4095)可以作为**ERR**空间。

```
static inline void * __must_check ERR_PTR(long error)
{
        return (void *) error;
}
```

convert error value into pointer

```
static inline long __must_check PTR_ERR(__force const void *ptr)
{
        return (long) ptr;
}
```

convert pointer into error value

```
static inline bool __must_check IS_ERR(__force const void *ptr)
{
        return IS_ERR_VALUE((unsigned long)ptr);
}
```

通过判断ptr指针是否落在0xffff,f000 and 0xffff,ffff的space中来check ptr是正常pointer还是带有出错码的fake pointer.

for example,

```
struct device *foo()
{
    ......

    if(... )
    {
        // error handling
        ERR_PTR(-EIO ) ;     (1)
    }

    ......
}
```

foo() function return value type is strcut device *.

```
struct device *d;
d = foo();            (2)
if(IS_ERR(d))         (3)
{
    long err = PTR_ERR(d);    (4)
}
```

(1)

foo() function在出错的情况下，需要告诉caller出错code(这里的EIO)，并且按照函数签名，需要返回的是pointer。

ERR_PTR(-EIO) 一举两得。

(2)

d 获得的是pointer

(3)

通过IS_ERR()来确定是否foo()是否成功返回

(4)

从ptr中提取出error code。