64 位的应用程序可以直接访问 4EB 的内存和文件大小最大达到 4 EB（2 的 63 次幂）；可以访问大型数据库。本文介绍的是 64 位下 C 语言开发程序注意事项。

1. 32 位和 64 位 C 数据类型

32 和 64 位 C 语言内置数据类型，如下表所示：

C TYPE	ILP32	LP64	LLP64	ILP64
char	8	8	8	8
short	16	16	16	16
float	32	32	32	32
double	64	64	64	64
int	32	32	32	64
long	32	64	64	64
long long	64	64	64	64
pointer	32	64	64	64

上表中第一行的大写字母和数字含义如下所示：

I 表示：int 类型

L 表示：long 类型

P 表示：pointer 指针类型

32 表示：32 位系统

64 表示 64 位系统

如：LP64 表示，在 64 位系统下的 long 类型和 pointer 类型长度为 64 位。

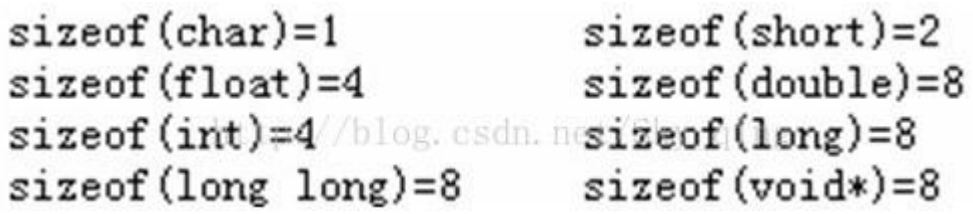
64 位 Linux 使用了 LP64 标准，即：long 类型和 pointer 类型长度为 64 位，其他类型的长度和 32 位系统下相同类型的长度相同，32 位和 64 位下类型的长度比较见上图的蓝色部分。

下图为在 32 和 64 位 linux 系统下使用 sizeof 检测出的数据类型的长度。

32 位平台下结果：

```
sizeof(char)=1          sizeof(short)=2
sizeof(float)=4         sizeof(double)=8
sizeof(int)=4           sizeof(long)=4
sizeof(long long)=8     sizeof(void*)=4
```

64 位平台下结果：



2. 64 系统下开发注意事项：

2.1 格式化字符串：long 使用%ld，指针使用%p，例如：

[cpp] view plain copy print?

```
1.  char *ptr = &something;
2.  printf (%x\n", ptr);
```

上面的代码在 64 位系统上不正确，只显示低 4 字节的内容。正确的方法是：使用 %p，如下：

[cpp] view plain copy print?

```
1.  char *ptr = &something;
2.  printf (%p\n", ptr);
```

2.2 数字常量：常量要加 L

例 1，常数 0xFFFFFFFF 是一个有符号的 long 类型。在 32 位系统上，这会将所有位都置位（每位全为 1），但是在 64 位系统上，只有低 32 位被置位了，结果是这个值是 0x00000000FFFFFFFF。

例 2，在下面的代码中，a 的最大值可以是 31。这是因为 1 << a 是 int 类型的。

[cpp] view plain copy print?

```
1.  long l = 1 << a;
```

要在 64 位系统上进行位移，应使用 1L，如下所示：

[cpp] view plain copy print?

```
1.  long l = 1L << a;
```

2.3 符号扩展：避免有符号数与无符号数运算，例如：

[cpp] view plain copy print?

```
1.  int i = -2;
2.  unsigned int j = 1;
```

```
3.  long l = i + j;
4.  printf("Answer: %ld\n",l);
```

32 位下是-1，在 64 位下是 4294967295。原因在于表达式(i+j)是一个 unsigned int 表达式，但把它赋值给 k 时，符号位没有被扩展。要解决这个问题，两端的操作数只要均为 signed 或均为 unsigned 就可。

2.4 转换截断：

转换截断发生在把 long 转换成 int 时，如下例：

[cpp] view plain copy print?

```
1.  int length = (int) strlen(str);
```

strlen 返回 size_t（它在 LP64 中是 unsigned long），当赋值给一个 int 时，截断是必然发生的。而通常，截断只会在 str 的长度大于 2GB 时才会发生，这种情况在程序中一般不会出现。虽然如此，也应该尽量使用适当的多态类型（如 size_t、uintptr_t 等等）。

2.5 赋值：

不要交换使用 int 和 long 类型，例如：

[cpp] view plain copy print?

```
1.  int i;
2.  time_t l;
3.  i = l;
```

不要使用 int 类型来存储指针，例如：

[cpp] view plain copy print?

```
1.  unsigned int i, *ptr;
2.  i = (unsigned) ptr;
```

不要使用指针来存放 int 类型的值。例如：

[cpp] view plain copy print?

```
1.  int *ptr;
2.  int i;
3.  ptr = (int *) i;
```

2.6 移植倒 64 位环境下的性能：

移植到 64 位平台后，性能实际上降低了。原因是 64 位中的指针长度和数据大小有关，并由此引发的缓存命中率降低、数据对齐等问题。通过改变结构中数据排列的先后顺序，会因为少了填充数据，存储空间也随之减少。如：

```
struct MyStruct
{
    int i;
    long l;
    int j;
    char* p;
};  // sizeof(MyStruct)=32
```

```
struct MyStruct
{
    int i;
    int j;
    long l;
    char* p;
};  // sizeof(MyStruct)=24
```

http://blog.csdn.net/qiyiqing

2.7 程序中链接到的库要使用 64 位的库。

由上可见所有的问题都是由 **long** 和指针长度改变引起，在开发过程中只有牢记 **long** 和指针类型的长度。