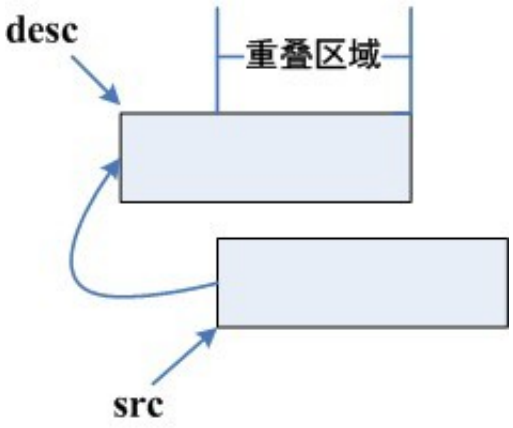
Linux库memmove函数实现

在/archives/227
主要介绍了memcpy函数的实现，并说明了memcpy函数的局限性。今天来介绍一下和memcpy函数功能类似的函数memmove。memmove函数和memcpy函数的原型为

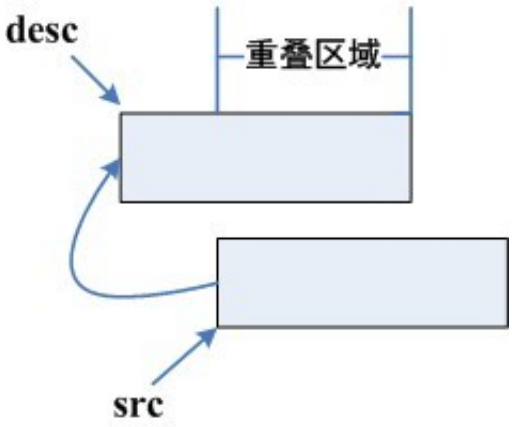
```
#include <string.h>

void *memcpy(void *dest, const void *src, size_t n);
void *memmove(void *dest, const void *src, size_t n);
```

memmove英文介绍，里面很详细的介绍了memmove函数的功能：
Copies the values of n bytes from the location pointed by src to the memory block pointed by dest. Copying takes place as if an intermediate buffer were used, allowing the destination and source to overlap.
The underlying type of the objects pointed by both the source and destination pointers are irrelevant for this function; The result is a binary copy of the data.
The function does not check for any terminating null character in source - it always copies exactly num bytes.
To avoid overflows, the size of the arrays pointed by both the destination and source parameters, shall be at least num bytes.
实际上memmove()与memcpy()一样都是用来拷贝src所指的内存内容前n个字节到dest所指的地址上。不同的是,当src和dest所指的内存区域重叠时,memmove()仍然可以正确的处理,不过执行效率上会比使用memcpy()略慢些。那么，memmove是怎么来避免memcpy的缺陷的呢？

1. 如果dest <= src，如下图所示：



   这种情况下，我们需要从src开始的部分复制东西到desc中去，这样就不会导致区域重叠而产生错误。
2. 如果dest > src，如下图所示：



这时候，我们需要先把src和desc的指针都移到末尾去，然后把src倒数的n个字符拷贝到desc的末尾去。

实现：

```
#ifndef __HAVE_ARCH_MEMMOVE
/**
 * memmove - Copy one area of memory to another
 * @dest: Where to copy to
 * @src: Where to copy from
 * @count: The size of the area.
 *
 * Unlike memcpy(), memmove() copes with overlapping areas.
 */
void *memmove(void *dest, const void *src, size_t count)
{
	char *tmp;
	const char *s;

	if (dest <= src) {
		tmp = dest;
		s = src;
		while (count--)
			*tmp++ = *s++;
	} else {
		tmp = dest;
		tmp += count;
		s = src;
		s += count;
		while (count--)
			*--tmp = *--s;
	}
	return dest;
}
EXPORT_SYMBOL(memmove);
```

#endif