Linux编程笔记

任何一个操作系统必须要关注的几个点

- 设备驱动
- 进程管理
- 内存管理
- 文件目录系统管理
- IO

分布式与中间件（tuxedo/corba/mq）

内存管理

- 硬件层次

  > 内存结构管理

- 内核层次

  > 内存映射 堆扩展

- 数据结构层次

  > STL 智能指针

- 语言层次

  > C：malloc C++:new

linux对内存的结构描述

> /proc/&{pid}/ 存放进程运行时的所有信息。

linux中4k的空间为一页

任何一个程序的内存空间分成4个基本部分：

1. 代码区
2. 全局栈区
3. 堆
4. 局部栈

进程查看： ps a/u/e

> 每一个程序都有一个自加载器，叫/lib/ld_linux.so.2,是一个标准的执行程序，负责把执行程序拷贝到代码空间，并且负责把指针指向main()

```
#include <stdio.h>
#include <stdlib.h>
int add(int a,int b)
{
    return a+b;
}

int a1 = 1;
static int a2 = 2;
const int a3 = 3;
main()
{
    int b1 = 4;
    static b2 = 5;
    const b3 = 6;

    int *p1 = malloc(4);

    printf("a1:%p\n",&a1);
    printf("a2:%p\n",&a2);
```

```
    printf("a3:%p\n",&a3);
    printf("b1:%p\n",&b1);
    printf("b2:%p\n",&b2);
    printf("b3:%p\n",&b3);
    printf("p1:%p\n",p1);
    printf("main:%p\n",&main);
    printf("add:%p\n",&add);


}
```

linux中代码的首地址是固定的。全局常量放在代码区，局部常量放在局部栈区。main在代码区，add在代码区。a1,a2在全局区。p1在堆里。

malloc的工作原理：

> malloc使用一个数据结构（链表）来维护分配空间联保的构成：分配的空间，上一个空间数据，下一个空间数据，空间大小。对malloc分配的空间不要越界访问，会导致后台维护的结构，使得mallooc,free,realloc,calloc不能正常工作。

C++的new和malloc的关系

malloc	new\new[]
realloc	new()(定位分配)
calloc	new[]
free	delete

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
    int *p1 = (int *)malloc(4);
    int *p2 = new int;
    int *p3 = (int *)malloc(4)l
    int *p4 = new int;
    int *p5 = new int;
    int *p6 = new int;

    printf("%p\n",p1);
    printf("%p\n",p2);
    printf("%p\n",p3);
    printf("%p\n",p4);
    printf("%p\n",p5);
    printf("%p\n",p6);

    return 0;
}
```

> new的实现使用的是malloc，但是有个区别：new使用malloc以后还要初始化空间，如果是基本类型，就直接初始化成默认值，UDT类型（用户自定义类型）调用指定的构造器。
>
> delete调用free实现，free直接释放，delete负责调用析构器，然后再调用free。
>
> **new与new[]的区别：** new调用一个构造器初始化，new[]循环对每个区域调用构造器。
>
> **delete与delete[]的区别：** delete只会调用首地址的析构器，而delete[]会依次调用序列中每个对象的析构器。
>
> 定位分配
>
> ```
> #include <stdio.h>
> #include <stdlib.h>
> #include <unistd.h>
> int main()
> ```

```
int main()
{
    char a[20];
    int *p = new(a) int;
    return 0;
}
```