1.

alsa:

snd\_pcm\_writei();

实践结果来看，函数进行写操作时，如果一次性写的Bytes超过一个周期的Bytes时，最后一次读写可能会产生实际写入与真实数据大小不匹配的错误（也可能不会产生），虽然声音文件听起来好像已经播放完。

2.

子进程对以阻塞模式打开的音频设备调用snd\_pcm\_writei();进行写操作时，如果父进程调用snd\_pcm\_drop()或snd\_pcm\_drain()或者snd\_pcm\_close()等函数来停止写操作时，子进程的行为不明，据实际测试代码来看，好像在snd\_pcm\_writei();停止了一样。

3.疑惑点依然存在：

①buffer\_time和buffer\_size

period\_time和period\_size有关联嘛？

测试过程中，buffer\_time从5us到0.5s不等进行设置，buffer\_size没有任何的改变，period也类似

②snd\_pcm\_hw\_params\_set\_rate\_resample（最后一个参数为0）设置之后rate变成固定值，同时buffer\_size和period\_size会随之改变，这是什么原因?

③snd\_pcm\_sw\_params\_get\_boundary获取的boundary到底是什么？设置之后有什么意义？

4．

int fseek(FILE \*stream, long offset, int fromwhere)返回值为error的值有两个，一个是由于stream不能seek导致，另一个是由于fromwhere不是SEEK\_SET,SEEK\_CUR,SEEK\_END导致。

5．

#define [min\_t](http://lxr.free-electrons.com/ident?i=min_t)([type](http://lxr.free-electrons.com/ident?i=type), [x](http://lxr.free-electrons.com/ident?i=x), [y](http://lxr.free-electrons.com/ident?i=y)) ({ \

[type](http://lxr.free-electrons.com/ident?i=type) \_\_min1 = ([x](http://lxr.free-electrons.com/ident?i=x)); \

[type](http://lxr.free-electrons.com/ident?i=type) \_\_min2 = ([y](http://lxr.free-electrons.com/ident?i=y)); \

\_\_min1 < \_\_min2 ? \_\_min1 : \_\_min2; })

可以用这种方法来定义变量。

GCC提供的typeof，实际上是在预编译时处理的，最后实际转化为数据类型被编译器处理。用法上也和上述语言不太一样。

基本用法是这样的

|  |  |
| --- | --- |
| 1  2  3 | int a;  typeof(a) b; //这等同于int b;  typeof(&a) c; //这等同于int\* c; |

/\*

 \* Check at compile time that something is of a particular type.

 \* Always evaluates to 1 so you may use it easily in comparisons.

 \*/

#define typecheck(type,x) \

({  type \_\_dummy; \

    typeof(x) \_\_dummy2; \

    (void)(&\_\_dummy == &\_\_dummy2); \

    1; \

})

/\*

 \* Check at compile time that 'function' is a certain type, or is a pointer

 \* to that type (needs to use typedef for the function type.)

 \*/

#define typecheck\_fn(type,function) \

({  typeof(type) \_\_tmp = function; \

    (void)\_\_tmp; \

})

这个例子里面不要求x和y是严格等于type类型，只要x和y能够安全地完成隐式类型转换为type就可以安全通过编译，否则会抛出warning。

另外一个非常经典的例子就是交换变量。

[?](http://deltamaster.is-programmer.com/posts/37253.html)

|  |  |
| --- | --- |
| 1  2  3  4  5 | /\*   \* swap - swap value of @a and @b   \*/  #define swap(a, b) \      do { typeof(a) \_\_tmp = (a); (a) = (b); (b) = \_\_tmp; } while (0) |

试想如果没有typeof，要怎么在C语言中实现这种类似C++模板的特性呢？

最后不得不提的就是container\_of宏，在kernel中也被广泛使用。

[?](http://deltamaster.is-programmer.com/posts/37253.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | /\*\*   \* container\_of - cast a member of a structure out to the containing structure   \* @ptr:    the pointer to the member.   \* @type:   the type of the container struct this is embedded in.   \* @member: the name of the member within the struct.   \*   \*/  #define container\_of(ptr, type, member) ({          \      const typeof( ((type \*)0)->member ) \*\_\_mptr = (ptr); \      (type \*)( (char \*)\_\_mptr - offsetof(type,member) );}) |

比如内核的task\_struct数据结构中有一个member是sched\_entity类型的se，这个member常常被调度器使用来决定进程的调度顺序，那么如果要根据这个se来获取包含它的task\_struct，就可以使用container\_of(p, task\_struct, se)来实现（假设p是指向这个sched\_entity的指针）。原理是先产生一个指针指向member，然后将这个指针减去member在这个struct中的偏移量，指针自然就指向了包含该member的对象了（这个地方用到了offsetof，含义一看便知，我就不再细说了）。

6.

alsa驱动程序里有:

runtime->twake = min\_t(snd\_pcm\_uframes\_t, size,

runtime->control->avail\_min ? : 1);

三目运算符第二个参数缺省，经测试默认为1；测试代码如下：

int a = 2;

typeof(a) b = (a == 2) ? : 4;

int c = (a != 2) ? : 5;

printf(" b: %d\n c: %d\n", b, c);