1. src/lz4.c:221: warning: 'LZ4\_readLE16' defined but not used

解释：

"Function defined but not used" warning is only issued for functions with internal linkage, i.e. functions that are declared as static. These functions are only accessible in one translation unit, so the compiler always knows whether they are used (in the program) or not. If you don't reference these functions in their translation unit, these functions are known to be unused, and the warning is generated.

You are saying that these functions "are not used in a.c, but used in b.c". This is not true. When you declare (and define) a function as static in the header file, each translation unit that includes that header file gets its own internal copy of the function. Even though these functions look absolutely the same, they are still separate, completely independent functions. The fact that they have the same name and consist of the same code means nothing to the compiler. So, in b.c you got a completely independent copy of the function, which is used (as you say), but the completely independent copy in a.c is still not used.

The question in this case is why you are doing this. Why on Earth are you defining static functions in the header file? If you really need to do this (i.e. if you really want to spawn a separate internal "clone" of this function in each translation unit), you can work around the warning by using some compiler-specific means. Like in GCC, for example, you can declare the function with \_\_attribute\_\_((unused)) an the warning for this function will no longer be issued.

But normally one wouldn't need to define functions in the header file. Normally one'd use a function with external linkage (i.e. no static), define it in one of the .c files and put the declaration (prototype) in the header file. The compiler will not issue any warnings in this case, even if the function is declared but not used in some translation unit.

“函数已定义但未使用”警告仅针对具有内部链接的函数发出，即声明为静态的函数。这些函数只能在一个翻译单元中访问，所以编译器总是知道它们是否被使用（在程序中）。如果您没有在它们的翻译单元中引用这些函数，那么这些函数将被认为是未使用的，并且会生成警告。

你说这些函数“不在a.c中使用，而在b.c中使用”。事实并非如此。当您在头文件中声明（并定义）一个函数为静态时，包含该头文件的每个翻译单元将获得它自己的函数的内部副本。尽管这些函数看起来完全一样，但它们仍然是独立的、完全独立的函数。它们具有相同的名称和由相同的代码组成的事实，对编译器来说毫无意义。因此，在b.c中，您得到了该函数的完全独立的副本，该副本是使用的（正如您所说的），但是a.c中的完全独立的副本仍然没有使用。

现在的问题是你为什么这么做。到底为什么要在头文件中定义静态函数？如果您确实需要这样做（例如，如果您确实想在每个翻译单元中生成这个函数的一个单独的内部“克隆”），那么您可以使用一些特定于编译器的方法来解决这个警告。例如，与在GCC中一样，可以使用\_\_attribute\_\_((unused)) 来声明函数，此函数的警告将不再发出。

但是通常不需要在头文件中定义函数。通常使用带有外部链接（即没有静态）的函数，将其定义在一个.c文件中，并将声明（原型）放入头文件中。在这种情况下，编译器不会发出任何警告，即使函数是声明的，但在某些翻译单元中没有使用。

解决：把对应方法前面的static修饰符去掉

1. [root@localhost Desktop]# make

gcc -o replace replace.c

replace.c:4: error: expected ‘=’, ‘,’, ‘;’, ‘asm’ or ‘\_\_attribute\_\_’ before ‘newMemory’

replace.c:12: error: expected ‘=’, ‘,’, ‘;’, ‘asm’ or ‘\_\_attribute\_\_’ before ‘uintToStr’

replace.c:30: error: expected ‘=’, ‘,’, ‘;’, ‘asm’ or ‘\_\_attribute\_\_’ before ‘replace’

make: \*\*\* [replace] Error 1

解释：这三个函数的返回值类型为bool，而C语言里面没有bool类型,除此之外，byte,true,false也没有

解决：

#define bool char

#define true 1

#define false 0

typedef char byte;

1. 为什么函数一返回，其形参指针指向的内容就变成了乱码？

解释：假设是这样

void get(int \*p,int num)

{

p=(int \*)malloc(num\*sizeof(int));

return;

}

调用函数

int \*pi;

get(pi,10);

get函数里的这个参数只是pi的一个副本，get函数结束后就没了，malloc分配的内存首地址也没带出来，pi还是原来的pi没变，还发生了内存泄漏

想用一级指针当参数 可以这样

int\* get(int \*p,int num) //返回指针

{

p=(int \*)malloc(num\*sizeof(int));

return p;

}

这样看起来是返回了一个局部变量的地址，但这个地址是个堆地址，函数结束后任然有效，只要记得释放就行了

那么就可以这样了

int\* pi;

pi=get(pi, 10);

//是不是有点脱裤放屁的感觉，还不如直接pi=(int\*)malloc(10\*sizeof(int))

最后在适当的时候释放 free(pi)

解决：传一个二级指针进去，或者将指针返回

1. src/monitor.c:178: warning: implicit declaration of function `pthread\_create'

解释：在monitor.c中调用了pthread\_create()函数，但是在C语言中调用函数需要先声明再调用

解决：在monitor.c文件的前面进行函数声明

int pthread\_create(pthread\_t \*tidp,const pthread\_attr\_t \*attr,(void\*)(\*start\_rtn)(void\*),void \*arg);

1. undefined reference to `dlclose'

undefined reference to `dlerror'

undefined reference to `dlsym'

undefined reference to `dlopen'

解释：dl库没有包进去，dl库是用来做动态库加载的（dynamic load）

解决：添加链接选项-ldl

比较常见的undefined reference问题：

* pthread库

undefined reference to ‘pthread\_create’

undefined reference to ‘pthread\_xxxx‘

增加-lpthread链接选项

* rt库

undefined reference to `clock\_gettime’

增加-lrt链接选项

1. [root@ktcatv log]$./producer-arm341

INFO Producer success to enqueue message (22 byte) for topic eocDevRxCFO1

%3|32094042.797|FAIL|rdkafka#producer-1| [thrd:162.105.155.71:9092/bootstrap]: 162.105.155.71:9092/bootstrap: Connect to ipv4#162.105.155.71:9092 failed: No route to host

%3|32094042.797|ERROR|rdkafka#producer-1| [thrd:162.105.155.71:9092/bootstrap]: 162.105.155.71:9092/bootstrap: Connect to ipv4#162.105.155.71:9092 failed: No route to host

INFO Producer success to enqueue message (207 byte) for topic eocDevRxCFO

解释1：可以ping通，但是实际发报文的时候显示No route to host，因为报文被服务器阻挡

解决1：在服务器上暂时关闭防火墙，service iptables stop

解释2：显示的日志说明rd\_kafka\_produce()已经成功调用，但是消息并没有发送出去。该函数是个非阻塞函数，只是让消息进入发送队列，并没有真正发送，如果调用rd\_kafka\_produce()后程序很快就退出了，那么消息很可能不会被发送

解决2：可以在调用rd\_kafka\_produce()后紧接着调用rd\_kafka\_flush()，让生产者把发送队列清空再进行下一步，使用非阻塞的API就要关注它是否把功能都执行完毕了

1. 程序执行到某一函数，立即崩溃，segment fault，调试发现没有进入该函数，似乎看到该函数就崩溃了

解释：在被调用的函数内部有这样一条语句：char cfgCode[3\*1024\*1024];直接在栈上申请了3M的空间，导致栈溢出(对于 x86 和 x64 计算机，默认堆栈大小为 1 MB)。可以使用malloc()这种方式在堆上申请大内存。

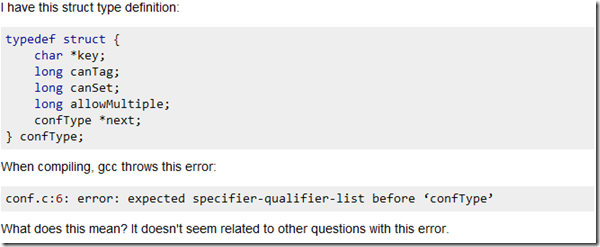
解决：在函数内使用较大内存时使用malloc()函数

1. expected specifier-qualifier-list before sth

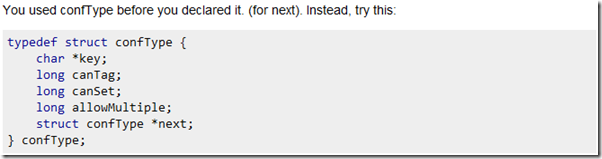
解释：在使用GCC时经常会遇到expected specifier-qualifier-list before sth之类的错误。specifiers是指void、char、struct Foo等词汇；qualifiers是指像const和volatile一类的关键字。一个词汇再未定义之前就使用就会出项这种错误，可以通过typedef进行定义以后再使用。

解决：

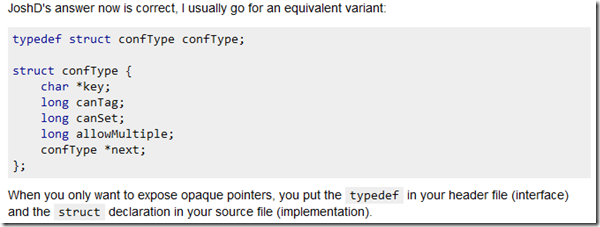
错误实例：



解决办法1:



解决办法2：



1. Select error when recv packet,exit!select()返回-1

解释：#include <sys/select.h>

int select(int maxfdp1, fd\_set \*readset, fd\_set \*writeset, fd\_set \*exceptset,struct timeval \*timeout);

timeout == NULL 等待无限长的时间。等待可以被一个信号中断。当有一个描述符做好准备或者是捕获到一个信号时函数会返回。如果捕获到一个信号， select函数将返回 -1,并将变量 erro设为 EINTR。

setitimer(ITIMER\_REAL, &value, NULL);//每隔interval，产生一个SIGALRM信号

select()捕获到了settimer()产生的时钟信号，慎用信号，容易发生信号泄漏，导致奇怪的错误

解决：如果要实现定时执行某个函数，可以试试sleep()