从一定程度上来说，实际上是个实时系统，即便通用系统也是实时系统。当中断发生时，操作系统可能只有若干微秒去完成特定的操作，否则就会丢失关键的信息。在任意时刻启动垃圾回收功能是不可接受的。

1.8.2 头文件

一个操作系统项目通常包括多个目录，每个目录都含有许多.c文件，这些文件中存有系统某个部分的代码，而一些.h 头文件则包含供一个或多个代码文件使用的声明以及定义。头文件还可以包括简单的宏，诸如

```
#define BUFFER_SIZE 4096
```

宏允许程序员命名常数，这样在代码中出现的BUFFER_SIZE，在编译时该常数就被数值4096所替代。良好的C程序设计实践应该除了0，1和-1之外命名所有的常数，有时把这三个数也进行命名。宏可以附带参数，例如

```
#define max(a, b)(a > b ? a: b)
```

这个宏允许程序员编写

```
i = max(j, k+1)
```

从而得到

```
i= (j > k+1 ? j : k+1)
```

将j与$k+1$之间的较大者存储在i中。头文件还可以包含条件编译，例如

```
#ifdef PENTIUM
intel_int_ack();
#endif
```

如果宏PENTIUM有定义，而不是其他，则编译进对intel_int_ack函数的调用。为了分割与结构有关的代码，大量使用了条件编译，这样只有当系统在Pentium上编译时，一些特定的代码才会被插入，其他的代码仅当系统在SPARC等机器上编译时才会插入。通过使用#include指令，一个.c文件体可以含有零个或多个头文件。

1.8.3 大型编程项目

为了构建操作系统，每个.c被C编译器编译成一个目标文件。目标文件使用后缀.o，含有目标机器的二进制代码。它们可以随后直接在CPU上运行。在C的世界里，没有类似于Java字节代码的东西。

C编译器的第一道称为C预处理器。在它读入每个.c文件时，每当遇到一个#include指令，它就取来该名称的头文件，并加以处理、扩展宏、处理条件编译（以及其他事务），然后将结果传递给编译器的下一道，仿佛它们原先就包含在该文件中一样。

由于操作系统非常大（五百万行代码是很寻常的），每当文件修改后就重新编译是不能忍受的。另一方面，改变了用在成千个文件中的一个关键头文件，确实需要重新编译这些文件。没有一定的协助，要想记录哪个目标文件与哪个头文件相关是完全不可行的。

幸运的是，计算机非常善于处理事务分类。在UNIX系统中，有个名为make的程序（其大量的变体如gmake、pmake等），它读入Makefile，该Makefile说明哪个文件与哪个文件相关。make的作用是，在构建操作系统二进制码时，检查此刻需要哪个目标文件，而且对于每个文件，检查自从上次目标文件创建之后，是否有任何它依赖（代码和头文件）的文件已经被修改了。如果有，目标文件需要重新编译。在make确定了哪个.o文件需要重新编译之后，它调用C编译器重新编译这些文件，这样，就把编译的次数减少到最低限度。在大型项目中，创建Makefile是一件容易出错的工作，所以出现了一些工具使该工作能够自动完成。

一旦所有的.o文件都已经就绪，这些文件被传递给称为linker的程序，将其组合成一个单个可执行的二进制文件。此时，任何被调用的库函数都已经包含在内，函数之间的引用都已经解决，而机器地址也都按需要分配完毕。在linker完成之后，得到一个可执行程序，在UNIX中传统上称为a.out文件。这个过程中的各种部分如图1-30所示，图中的一个程序包含三个C文件，两个头文件。这里虽然讨论的是有关操作系统的开发，但是所有内容对开发任何大型程序而言都是适用的。