取决于在第 6 行和第 10 行发生的 malloc 和 free 的调用模式，当程序在第 14 行引用 x[i]时，数组 x 可能是某个其他已分配堆块的一部分了，因此其内容被重写了。和其他许多与内存有关的错误一样，这个错误只会在程序执行的后面，当我们注意到 y 中的值被破坏了时才会显现出来。

### 9.11.10 引起内存泄漏

内存泄漏是缓慢、隐性的杀手，当程序员不小心忘记释放已分配块，而在堆里创建了垃圾时，会发生这种问题。例如，下面的函数分配了一个堆块 x，然后不释放它就返回：

```
1   void leak(int n)
2   {
3       int *x = (int *)Malloc(n * sizeof(int));
4
5       return;  /* x is garbage at this point */
6   }
```

如果经常调用 leak，那么渐渐地，堆里就会充满了垃圾，最糟糕的情况下，会占用整个虚拟地址空间。对于像守护进程和服务器这样的程序来说，内存泄漏是特别严重的，根据定义这些程序是不会终止的。

## 9.12 小结

虚拟内存是对主存的一个抽象。支持虚拟内存的处理器通过使用一种叫做虚拟寻址的间接形式来引用主存。处理器产生一个虚拟地址，在被发送到主存之前，这个地址被翻译成一个物理地址。从虚拟地址空间到物理地址空间的地址翻译要求硬件和软件紧密合作。专门的硬件通过使用页表来翻译虚拟地址，而页表的内容是由操作系统提供的。

虚拟内存提供三个重要的功能。第一，它在主存中自动缓存最近使用的存放磁盘上的虚拟地址空间的内容。虚拟内存缓存中的块叫做页。对磁盘上页的引用会触发缺页，缺页将控制转移到操作系统中的一个缺页处理程序。缺页处理程序将页面从磁盘复制到主存缓存，如果必要，将写回被驱逐的页。第二，虚拟内存简化了内存管理，进而又简化了链接、在进程间共享数据、进程的内存分配以及程序加载。最后，虚拟内存通过在每条页表条目中加入保护位，从而了简化了内存保护。

地址翻译的过程必须和系统中所有的硬件缓存的操作集成在一起。大多数页表条目位于 L1 高速缓存中，但是一个称为 TLB 的页表条目的片上高速缓存，通常会消除访问在 L1 上的页表条目的开销。

现代系统通过将虚拟内存片和磁盘上的文件片关联起来，来初始化虚拟内存片，这个过程称为内存映射。内存映射为共享数据、创建新的进程以及加载程序提供了一种高效的机制。应用可以使用 mmap 函数来手工地创建和删除虚拟地址空间的区域。然而，大多数程序依赖于动态内存分配器，例如 malloc，它管理虚拟地址空间区域内一个称为堆的区域。动态内存分配器是一个感觉像系统级程序的应用级程序，它直接操作内存，而无需类型系统的很多帮助。分配器有两种类型。显式分配器要求应用显式地释放它们的内存块。隐式分配器(垃圾收集器)自动释放任何未使用的和不可达的块。

对于 C 程序员来说，管理和使用虚拟内存是一件困难和容易出错的任务。常见的错误示例包括：间接引用坏指针，读取未初始化的内存，允许栈缓冲区溢出，假设指针和它们指向的对象大小相同，引用指针而不是它所指向的对象，误解指针运算，引用不存在的变量，以及引起内存泄漏。

## 参考文献说明

Kilburn 和他的同事们发表了第一篇关于虚拟内存的描述[63]。体系结构教科书包括关于硬件在虚拟内存中的角色的更多细节[46]。操作系统教科书包含关于操作系统角色的更多信息[102，106，113]。Bovet 和 Cesati [11]给出了 Linux 虚拟内存系统的详细描述。Intel 公司提供了 IA 处理器上 32 位和 64 位