```
18
19  /* Thread routine */
20  void *thread(void *vargp)
21  {
22      int myid = *((int *)vargp);
23      printf("Hello from thread %d\n", myid);
24      return NULL;
25  }
```

code/conc/race.c

图 12-42　（续）

问题是由每个对等线程和主线程之间的竞争引起的。你能发现这个竞争吗？下面是发生的情况。当主线程在第 13 行创建了一个对等线程，它传递了一个指向本地栈变量 i 的指针。在此时，竞争出现在下一次在第 12 行对 i 加 1 和第 22 行参数的间接引用和赋值之间。如果对等线程在主线程执行第 12 行对 i 加 1 之前就执行了第 22 行，那么 myid 变量就得到正确的 ID。否则，它包含的就会是其他线程的 ID。令人惊慌的是，我们是否得到正确的答案依赖于内核是如何调度线程的执行的。在我们的系统中它失败了，但是在其他系统中，它可能就能正确工作，让程序员“幸福地”察觉不到程序的严重错误。

为了消除竞争，我们可以动态地为每个整数 ID 分配一个独立的块，并且传递给线程例程一个指向这个块的指针，如图 12-43 所示(第 12～14 行)。请注意线程例程必须释放这些块以避免内存泄漏。

code/conc/norace.c

```
1   #include "csapp.h"
2   #define N 4
3
4   void *thread(void *vargp);
5
6   int main()
7   {
8       pthread_t tid[N];
9       int i, *ptr;
10
11      for (i = 0; i < N; i++) {
12          ptr = Malloc(sizeof(int));
13          *ptr = i;
14          Pthread_create(&tid[i], NULL, thread, ptr);
15      }
16      for (i = 0; i < N; i++)
17          Pthread_join(tid[i], NULL);
18      exit(0);
19  }
20
21  /* Thread routine */
22  void *thread(void *vargp)
23  {
24      int myid = *((int *)vargp);
25      Free(vargp);
26      printf("Hello from thread %d\n", myid);
27      return NULL;
28  }
```

code/conc/norace.c

图 12-43　图 12-42 中程序的一个没有竞争的正确版本