```
17      if (argc != 3) {
18          printf("Usage: %s <nthreads> <log_nelems>\n", argv[0]);
19          exit(0);
20      }
21      nthreads = atoi(argv[1]);
22      log_nelems = atoi(argv[2]);
23      nelems = (1L << log_nelems);
24      nelems_per_thread = nelems / nthreads;
25      sem_init(&mutex, 0, 1);
26
27      /* Create peer threads and wait for them to finish */
28      for (i = 0; i < nthreads; i++) {
29          myid[i] = i;
30          Pthread_create(&tid[i], NULL, sum_mutex, &myid[i]);
31      }
32      for (i = 0; i < nthreads; i++)
33          Pthread_join(tid[i], NULL);
34
35      /* Check final answer */
36      if (gsum != (nelems * (nelems-1))/2)
37          printf("Error: result=%ld\n", gsum);
38
39      exit(0);
40  }
```

code/conc/psum-mutex.c

图 12-31　（续）

图 12-32 给出了每个对等线程执行的函数。在第 4 行中，线程从线程参数中提取出线程 ID，然后用这个 ID 来决定它要计算的序列区域(第 5～6 行)。在第 9～13 行中，线程在它的那部分序列上迭代操作，每次迭代都更新共享全局变量 gsum。注意，我们很小心地用 P 和 V 互斥操作来保护每次更新。

code/conc/psum-mutex.c

```
1   /* Thread routine for psum-mutex.c */
2   void *sum_mutex(void *vargp)
3   {
4       long myid = *((long *)vargp);          /* Extract the thread ID */
5       long start = myid * nelems_per_thread; /* Start element index */
6       long end = start + nelems_per_thread;  /* End element index */
7       long i;
8
9       for (i = start; i < end; i++) {
10          P(&mutex);
11          gsum += i;
12          V(&mutex);
13      }
14      return NULL;
15  }
```

code/conc/psum-mutex.c

图 12-32　psum-mutex 的线程例程。每个对等线程将各自的和累加进一个用互斥锁保护的共享全局变量中