```

#include <stdio.h>
#include <math.h>
#include <pthread.h>  // required for threads usage

#define MAX_N 100000000
#define MAX_THREADS 25

// shared variables
int nthreads,  // number of threads (not counting main())
    n,  // range to check for primeness
    prime[MAX_N+1],  // in the end, prime[i] = 1 if i prime, else 0
    nextbase;  // next sieve multiplier to be used
// lock for the shared variable nextbase
pthread_mutex_t nextbaselock = PTHREAD_MUTEX_INITIALIZER;
// ID structs for the threads
pthread_t id[MAX_THREADS];

// "crosses out" all odd multiples of k
void crossout(int k)
{  int i;
   for (i = 3; i*k <= n; i += 2)  {
      prime[i*k] = 0;
   }
}

// each thread runs this routine
void *worker(int tn)  // tn is the thread number (0,1,...)
{  int lim,base,
       work = 0;  // amount of work done by this thread
   // no need to check multipliers bigger than sqrt(n)
   lim = sqrt(n);
   do  {
      // get next sieve multiplier, avoiding duplication across threads
      // lock the lock
      pthread_mutex_lock(&nextbaselock);
      base = nextbase;
      nextbase += 2;
      // unlock
      pthread_mutex_unlock(&nextbaselock);
      if (base <= lim)  {
         // don't bother crossing out if base known composite
         if (prime[base])  {
            crossout(base);
            work++;  // log work done by this thread
         }
      }
      else return work;
   } while (1);
}

main(int argc, char **argv)
{  int nprimes,  // number of primes found
       i,work;
   n = atoi(argv[1]);
   nthreads = atoi(argv[2]);
   // mark all even numbers nonprime, and the rest "prime until
   // shown otherwise"
   for (i = 3; i <= n; i++)  {
      if (i%2 == 0) prime[i] = 0;
      else prime[i] = 1;
   }
   nextbase = 3;
   // get threads started
   for (i = 0; i < nthreads; i++)  {
      // this call says to create a thread, record its ID in the array
      // id, and get the thread started executing the function worker(),
      // passing the argument i to that function
      pthread_create(&id[i],NULL,worker,i);
   }

   // barrier, to wait for all done
   for (i = 0; i < nthreads; i++)  {
      // this call said to wait until thread number id[i] finishes
      // execution, and to assign the return value of that thread to our
      // local variable work here
      pthread_join(id[i],&work);
      printf("%d values of base done\n",work);
   }

   // report results
   nprimes = 1;
   for (i = 3; i <= n; i++)
      if (prime[i])  {
         nprimes++;
      }
   printf("the number of primes found was %d\n",nprimes);

}




// this include file is mandatory
#include <mpi.h>

#define MAX_N 100000
#define PIPE_MSG 0  // type of message containing a number to
                       be checked
#define END_MSG 1  // type of message indicating no more data will
                      be coming

int NNodes,  /* number of nodes in computation*/
    N,  /* find all primes from 2 to N */
    Me,  /* my node number */
    ToCheck;  /* current number to check for passing on to next node;
                 stylistically this might be nicer as a local in
                 Node*(), but I have placed it here to dramatize
                 the fact that the globals are NOT shared among
                 the nodes */

double T1,T2;  /* start and finish times */

Init(Argc,Argv)
   int Argc;  char **Argv;

{  int DebugWait;

   N = atoi(Argv[1]);
   DebugWait = atoi(Argv[2]);

   /* this loop is here to synchronize all nodes for debugging;
      if DebugWait is specified as 1 on the command line, all nodes
      wait here until the debugging programmer starts GDB at all
      nodes and within GDB sets DebugWait to 0 to then proceed */
   while (DebugWait) ;

   /* mandatory to begin any MPI program */
   MPI_Init(&Argc,&Argv);

   /* puts the number of nodes in NNodes */
   MPI_Comm_size(MPI_COMM_WORLD,&NNodes);
   /* puts the node number of this node in Me */
   MPI_Comm_rank(MPI_COMM_WORLD,&Me);

   /* OK, get started; first record current time in T1 */
   if (Me == 2) T1 = MPI_Wtime();
}
```

```

Node0()

{  int I,Dummy,
       Error;  /* not checked in this example */
   for (I = 1; I <= N/2; I++)  {
      ToCheck = 2 * I + 1;
      if (ToCheck > N) break;
      /* MPI_Send  --  send a message
               parameters:
               pointer to place where message is to be drawn from
               number of items in message
               item type
               destination node
               message type ("tag") programmer-defined
               node group number (in this case all nodes) */
      if (ToCheck % 3 > 0)
         Error = MPI_Send(&ToCheck,1,MPI_INT,1,PIPE_MSG,MPI_COMM_WORLD);
   }
   Error = MPI_Send(&Dummy,1,MPI_INT,1,END_MSG,MPI_COMM_WORLD);
}

Node1()

{  int Error,  /* not checked in this example */
       Dummy;
   MPI_Status Status;  /* see below */

   while (1)  {
      /* MPI_Recv  --  receive a message
               parameters:
               pointer to place to store message
               number of items in message (see notes on
                  this at the end of this file)
               item type
               accept message from which node(s)
               message type ("tag"), programmer-defined (in this
          case any type)
               node group number (in this case all nodes)
               status (see notes on this at the end of this file) */
      Error = MPI_Recv(&ToCheck,1,MPI_INT,0,MPI_ANY_TAG,
                  MPI_COMM_WORLD,&Status);
      if (Status.MPI_TAG == END_MSG) break;
      if (ToCheck % 5 > 0)
         Error = MPI_Send(&ToCheck,1,MPI_INT,2,PIPE_MSG,MPI_COMM_WORLD);
   }
   /* now send our end-of-data signal, which is conveyed in the
      message type, not the message (we have a dummy message just
      as a placeholder */
   Error = MPI_Send(&Dummy,1,MPI_INT,2,END_MSG,MPI_COMM_WORLD);
}

Node2()

{  int ToCheck,  /* current number to check from Node 0 */
       Error,  /* not checked in this example */
       PrimeCount,I,IsComposite;
   MPI_Status Status;  /* see below */

   PrimeCount = 3;  /* must account for the primes 2, 3 and 5, which
                    won't be detected below */
   while (1)  {
      Error = MPI_Recv(&ToCheck,1,MPI_INT,1,MPI_ANY_TAG,
                  MPI_COMM_WORLD,&Status);
      if (Status.MPI_TAG == END_MSG) break;
      IsComposite = 0;
      for (I = 7; I*I <= ToCheck; I += 2)
         if (ToCheck % I == 0)  {
    IsComposite = 1;
    break;
 }
      if (!IsComposite) PrimeCount++;
   }
   /* check the time again, and subtract to find run time */
   T2 = MPI_Wtime();
   printf("elapsed time = %f\n",(float)(T2-T1));
   /* print results */
   printf("number of primes = %d\n",PrimeCount);
}

main(argc,argv)
   int argc; char **argv;

{  Init(argc,argv);
   /* note:  instead of having a switch statement, we could write
      three different programs, each running on a different node */
   switch (Me)  {
      case 0:  Node0();
               break;
      case 1:  Node1();
               break;
      case 2:  Node2();
   };
   /* mandatory for all MPI programs */
   MPI_Finalize();
}

```