## Παράδειγμα 1Α

Δημιουργία ενός αριθμού νημάτων (ο αριθμός καθορίζεται από όρισμα της γραμμής εντολών) και εμφάνιση μηνύματος που περιέχει τον αριθμό νήματος για κάθε νήμα.

```c
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>

// Pacheco (program 4.1 )

int thread_count;

void *hello(void *rank) {
  long myrank = (long)rank;
  pthread_t thread_internal_id = pthread_self();
  printf("Hello from thread %ld (opaque thread id: %lu) of %d\n", myrank,
         (unsigned long)thread_internal_id, thread_count);
  return NULL;
}

int main(int argc, char *argv[]) {
  long thread;
  thread_count = strtol(argv[1], NULL, 0);
  pthread_t *thread_handles;
  thread_handles = malloc(thread_count * sizeof(pthread_t));
  for (thread = 0; thread < thread_count; thread++) {
    pthread_create(&thread_handles[thread], NULL, hello, (void *)thread);
  }
  printf("Hello from the main thread\n");
  for (thread = 0; thread < thread_count; thread++) {
    pthread_join(thread_handles[thread], NULL);
  }
  free(thread_handles);
  return 0;
}
```

pthreads_example01a.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
gcc pthreads_example01a.c -o pthreads_example01a -lpthread
```

```
./pthreads_example01a 4
```

```
Hello from thread 0 (opaque thread id: 123145492553728) of 4
Hello from thread 1 (opaque thread id: 123145493090304) of 4
Hello from thread 2 (opaque thread id: 123145493626880) of 4
Hello from the main thread
Hello from thread 3 (opaque thread id: 123145494163456) of 4
```

## Παράδειγμα 1Β

Δημιουργία 4 νημάτων. Κάθε νήμα "παράγει" έναν αριθμό που τον επιστρέφει στο κύριο νήμα. Εμφανίζονται όλες οι τιμές που παράγονται και επιστρέφονται.

```c
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>

#define THREAD_COUNT 4

void *work(void *tid)
{
  // έστω ότι οι ακόλουθες τιμές υπολογίζονται με χρονοβόρα επεξεργασία από κάθε thread
  // δλδ το thread 0 υπολογίζει την τιμή 23, το thread 1 την τιμή 12 κοκ
  long dummy_values[] = {23, 12, 78, 90};
  long mytid = (long)tid;
  long r = dummy_values[mytid];
  printf("Thread %ld produces %ld\n", mytid, r);
  pthread_exit((void *)r);
}

int main()
{
  long t;
  pthread_t thread_handles[THREAD_COUNT];
  for (t = 0; t < THREAD_COUNT; t++)
    pthread_create(&thread_handles[t], NULL, work, (void *)t);

  for (t = 0; t < THREAD_COUNT; t++)
  {
    void *r;
    pthread_join(thread_handles[t], &r);
    printf("Thread %ld returns %ld\n", t, (long)r);
  }
  return 0;
}
```

pthreads_example01b.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
gcc pthreads_example01b.c -o pthreads_example01b –lpthread
```

```
./pthreads_example01b
```

```
Thread 0 produces 23
Thread 1 produces 12
Thread 2 produces 78
Thread 3 produces 90
Thread 0 returns 23
Thread 1 returns 12
Thread 2 returns 78
Thread 3 returns 90
```

## Παράδειγμα 2

Πολλαπλασιασμός ενός πίνακα A με m γραμμές και n στήλες με ένα διάνυσμα n στοιχείων

### Σειριακή λύση

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define a 10.0

/*
 Πίνακας A με διαστάσεις m x n
 Διάνυσμα x με n στοιχεία
 Σειριακός πολλαπλασιασμός του πίνακα A με το διάνυσμα x
*/
void generate_random_data(double **A, double *x, int m, int n) {
  srand(time(NULL));
  for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
      A[i][j] = a * (double)rand() / (double)RAND_MAX;
  for (int i = 0; i < n; i++)
    x[i] = a * (double)rand() / (double)RAND_MAX;
}

int main(int argc, char **argv) {
  double **A;
  double *x, *y;
  int m, n;

  if (argc != 3) {
    printf("Usage: %s m n\n", argv[0]);
    exit(-1);
  }
  m = atoi(argv[1]);
  n = atoi(argv[2]);

  x = (double *)malloc(sizeof(double) * n);
  // για επιπλέον τρόπους δέσμευσης δισδιάστατου πίνακα στη C δείτε
  // http://www.geeksforgeeks.org/dynamically-allocate-2d-array-c/
  A = (double **)malloc(sizeof(double *) * m);
  for (int i = 0; i < m; i++)
    A[i] = (double *)malloc(sizeof(double) * n);
  y = (double *)malloc(sizeof(double) * m);

  generate_random_data(A, x, m, n);

  for (int i = 0; i < m; i++) {
    y[i] = 0.0;
    for (int j = 0; j < n; j++)
      y[i] += A[i][j] * x[j];
  }
```

```
  int k = m;
  if (k > 10)
    k = 10;
  for (int i = 0; i < k; i++)
    printf("%d -> %.2f\n", i, y[i]);

  free(x);
  free(y);
  for (int i = 0; i < m; i++)
    free(A[i]);
  free(A);
  return 0;
}
```

pthreads_example02a.c (σειριακή λύση)

Μεταγλώττιση – εκτέλεση – έξοδος

```
gcc pthreads_example02a.c -o pthreads_example02a
```

```
time ./pthreads_example02a 100000 1000
```

```
0 -> 25925.70
1 -> 25116.42
2 -> 24880.10
3 -> 26421.22
4 -> 24739.83
5 -> 24421.85
6 -> 25089.27
7 -> 25201.41
8 -> 25585.28
9 -> 25770.27
./pthreads_example02a 100000 1000  1.22s user 0.29s system 99% cpu 1.520 total
```

## Παράλληλη λύση

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>

#define a 10.0

double **A;
double *x, *y;
int m, n, thread_count;

void generate_random_data(double **A, double *x, int m, int n) {
  srand(time(NULL));
  for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
      A[i][j] = a * (double)rand() / (double)RAND_MAX;

  for (int i = 0; i < n; i++)
    x[i] = a * (double)rand() / (double)RAND_MAX;
```

```
}

void *Pth_mat_vect(void *rank) {
  long my_rank = (long)rank;
  int i, j;
  int local_m = m / thread_count;
  int my_first_row = my_rank * local_m;
  int my_last_row = (my_rank + 1) * local_m - 1;

  for (i = my_first_row; i <= my_last_row; i++) {
    y[i] = 0.0;
    for (j = 0; j < n; j++)
      y[i] += A[i][j] * x[j];
  }
  return NULL;
}

int main(int argc, char **argv) {
  if (argc != 4) {
    printf("Usage: %s m n t\n", argv[0]);
    printf("m = # of rows, n= # of columns, t = # of threads\n");
    exit(-1);
  }
  long thread_id;
  pthread_t *thread_handles;
  m = atoi(argv[1]);
  n = atoi(argv[2]);
  thread_count = atoi(argv[3]);

  x = (double *)malloc(sizeof(double) * n);
  A = (double **)malloc(sizeof(double *) * m);
  for (int i = 0; i < m; i++)
    A[i] = (double *)malloc(sizeof(double) * n);
  y = (double *)malloc(sizeof(double) * m);

  generate_random_data(A, x, m, n);

  thread_handles = malloc(sizeof(pthread_t) * thread_count);

  for (thread_id = 0; thread_id < thread_count; thread_id++)
    pthread_create(&thread_handles[thread_id], NULL, Pth_mat_vect,
                   (void *)thread_id);

  for (thread_id = 0; thread_id < thread_count; thread_id++)
    pthread_join(thread_handles[thread_id], NULL);

  int k = m;
  if (k > 10)
    k = 10;
  for (int i = 0; i < k; i++)
    printf("%d -> %.2f\n", i, y[i]);
```

```
  free(x);
  free(y);
  for (int i = 0; i < m; i++)
    free(A[i]);
  free(A);
  return 0;
}
```

pthreads_example02b.c (παράλληλη λύση)

Μεταγλώττιση – εκτέλεση – έξοδος

```
gcc pthreads_example02b.c -o pthreads_example02b
```

```
time ./pthreads_example02b 100000 1000 4
```

```
time ./pthreads_example02b 100000 1000 4
0 -> 24343.15
1 -> 24505.63
2 -> 24254.85
3 -> 24339.83
4 -> 25015.67
5 -> 24040.35
6 -> 24510.99
7 -> 24588.91
8 -> 25797.50
9 -> 24254.58
./pthreads_example02b 100000 1000 4  1.52s user 0.27s system 141% cpu 1.269 total
```

## Παράδειγμα 3

Υπολογισμός του π ως ένα άθροισμα σειράς (βλ. Pacheco σελ 229)

- pthreads_example03a.c (σειριακός κώδικας, παράλληλος κώδικας χωρίς συγχρονισμό)
- pthreads_example03b.c (παράλληλος κώδικας με busy wait)
- pthreads_example03c.c (παράλληλος κώδικας με mutex)

### Σειριακός κώδικας και παράλληλος κώδικας χωρίς συγχρονισμό

```
// Pacheco example 4.3
// the value of pi computed by the parallel code is not computed correctly
// due to race conditions

#include <stdio.h>
#include <pthread.h>

#define T 10      // number of threads
#define N 1000000 // number of terms of the series to use

double sum;

void *compute_pi_parallel(void *rank);
double compute_pi_serial(int);

int main() {
  long thread;
  pthread_t thread_handles[T];
```

```
  sum = 0.0;
  for (thread = 0; thread < T; thread++) {
    pthread_create(&thread_handles[thread], NULL, compute_pi_parallel,
                   (void *)thread);
  }

  for (thread = 0; thread < T; thread++) {
    pthread_join(thread_handles[thread], NULL);
  }
  sum = 4 * sum;
  printf("Pi value computed using %d terms and %d threads  = %.12f\n", N, T,
         sum);
  printf("Pi value computed using %d terms and serial code = %.12f\n", N,
         compute_pi_serial(N));
  return 0;
}

void *compute_pi_parallel(void *rank) {
  long my_rank = (long)rank;
  double factor;
  long i;
  long my_n = N / T;
  long my_first_i = my_n * my_rank;
  long my_last_i = my_first_i + my_n;
  if (my_first_i % 2 == 0)
    factor = 1.0;
  else
    factor = -1.0;
  for (i = my_first_i; i < my_last_i; i++, factor = -factor) {
    sum += factor / (2 * i + 1);
  }
  return NULL;
}

double compute_pi_serial(int n) {
  int i;
  double factor = 1.0;
  double sum = 0.0;
  for (i = 0; i < n; i++, factor = -factor) {
    sum += factor / (2 * i + 1);
  }
  return 4.0 * sum;
}
```

pthreads_example03a.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
gcc pthreads_example03a.c -o pthreads_example03a -lpthread
```

```
./pthreads_example03a
```

```
Pi value computed using 1000000 terms and 10 threads  = 3.144836482933
Pi value computed using 1000000 terms and serial code = 3.141591653590
```

Παράλληλος κώδικας με busy wait

```
// Pacheco example 4.5
// the value of pi computed by the parallel code is correct because race
// conditions are handled using busy waiting

#include <stdio.h>
#include <pthread.h>

#define T 10      // number of threads
#define N 1000000 // 10^6, number of terms of the series to use

double sum;
int flag;

void *compute_pi_parallel(void *rank);

int main() {
  long thread;
  pthread_t thread_handles[T];

  sum = 0.0;
  flag = 0;
  for (thread = 0; thread < T; thread++) {
    pthread_create(&thread_handles[thread], NULL, compute_pi_parallel,
                   (void *)thread);
  }

  for (thread = 0; thread < T; thread++) {
    pthread_join(thread_handles[thread], NULL);
  }
  sum = 4 * sum;
  printf(
      "Pi value computed using %d terms and %d threads  (busy wait) = %.12f\n",
      N, T, sum);
  return 0;
}

void *compute_pi_parallel(void *rank) {
  long my_rank = (long)rank;
  double factor;
  long i;
  long my_n = N / T;
  long my_first_i = my_n * my_rank;
  long my_last_i = my_first_i + my_n;
  double my_sum = 0.0;

  if (my_first_i % 2 == 0)
    factor = 1.0;
  else
    factor = -1.0;
```

```
  for (i = my_first_i; i < my_last_i; i++, factor = -factor) {
    my_sum += factor / (2 * i + 1);
  }
  // busy waiting
  while (flag != my_rank);
  sum += my_sum;
  flag++;
  return NULL;
}
```

pthreads_example03b.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
gcc pthreads_example03b.c -o pthreads_example03b –lpthread
```

```
./pthreads_example03b
```

```
Pi value computed using 1000000 terms and 10 threads (busy wait) = 3.141591653590
```

### Παράλληλος κώδικας με mutex

```
// Pacheco example 4.3
// the value of pi computed by the parallel code is correct because race
// conditions are handled by a mutex

#include <stdio.h>
#include <pthread.h>

#define T 10      // number of threads
#define N 1000000 // 10^6, number of terms of the series to use

// shared variables
double sum;
pthread_mutex_t mutex;

void *compute_pi_parallel(void *rank);

int main() {
  long thread;
  pthread_t thread_handles[T];
  pthread_mutex_init(&mutex, NULL);

  sum = 0.0;
  for (thread = 0; thread < T; thread++) {
    pthread_create(&thread_handles[thread], NULL, compute_pi_parallel,
                   (void *)thread);
  }

  for (thread = 0; thread < T; thread++) {
    pthread_join(thread_handles[thread], NULL);
  }
  sum = 4 * sum;
  printf("Pi value computed using %d terms and %d threads (mutex) = %.12f\n", N,
         T, sum);
```

```
  pthread_mutex_destroy(&mutex);
  return 0;
}

void *compute_pi_parallel(void *rank) {
  long my_rank = (long)rank;
  double factor;
  long i;
  long my_n = N / T;
  long my_first_i = my_n * my_rank;
  long my_last_i = my_first_i + my_n;
  double my_sum = 0.0;

  if (my_first_i % 2 == 0)
    factor = 1.0;
  else
    factor = -1.0;
  for (i = my_first_i; i < my_last_i; i++, factor = -factor) {
    my_sum += factor / (2 * i + 1);
  }
  pthread_mutex_lock(&mutex);
  sum += my_sum;
  pthread_mutex_unlock(&mutex);
  return NULL;
}
```

pthreads_example03c.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
gcc pthreads_example03c.c -o pthreads_example03c –lpthread
./pthreads_example03c
Pi value computed using 1000000 terms and 10 threads (mutex) = 3.141591653590
```

## Παράδειγμα 4Α (μεταβλητές υπό συνθήκη)

Εννέα threads παράγουν από έναν αριθμό (π.χ. τον αριθμό 1) και τον τοποθετούν σε διαφορετικές θέσεις ενός κοινόχρηστου πίνακα. Ένα επιπλέον thread λειτουργεί ως καταναλωτής των τιμών που παρήγαγαν τα νήματα παραγωγοί τις οποίες και διπλασιάζει. Το κύριο πρόγραμμα αθροίζει όλες τις τιμές του κοινόχρηστου πίνακα.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define T 10

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond_var = PTHREAD_COND_INITIALIZER;

int c = 0;
int values[T - 1];
void *produce(void *tid) {
  long mytid = (long)tid;
  printf("Thread %ld produces value 1\n", mytid);
```

```
  values[mytid] = 1;
  pthread_mutex_lock(&lock);
  c++;
  if (c == T - 1)
    pthread_cond_signal(&cond_var);
  pthread_mutex_unlock(&lock);
  return NULL;
}

void *consume(void *tid) {
  long mytid = (long)tid;
  pthread_mutex_lock(&lock);
  if (c < T - 1)
    pthread_cond_wait(&cond_var, &lock);
  pthread_mutex_unlock(&lock);
  for (int i = 0; i < T - 1; i++)
    values[i] *= 2;
  printf("Consumer thread %ld doubled all values\n", mytid);
  return NULL;
}

int main() {
  pthread_t thread_handles[T];
  for (long t = 0; t < T - 1; t++)
    pthread_create(&thread_handles[t], NULL, produce, (void *)t);
  pthread_create(&thread_handles[T - 1], NULL, consume, (void *)T - 1);

  for (long t = 0; t < T; t++)
    pthread_join(thread_handles[t], NULL);

  int sum = 0;
  for (int i = 0; i < T - 1; i++)
    sum += values[i];
  printf("Main: The sum is %d\n", sum);
  pthread_mutex_destroy(&lock);
  pthread_cond_destroy(&cond_var);
  return 0;
}
```

pthreads_example04a.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
gcc pthreads_example04a.c -o pthreads_example04a -lpthread
```

```
./pthreads_example04a
```

```
Thread 0 produces value 1
Thread 1 produces value 1
Thread 2 produces value 1
Thread 3 produces value 1
Thread 4 produces value 1
Thread 5 produces value 1
Thread 6 produces value 1
Thread 7 produces value 1
```

```
Thread 8 produces value 1
Consumer thread 9 doubled all values
Main: The sum is 18
```

### Παράδειγμα 4Β (μεταβλητές υπό συνθήκη)

Ένα thread παράγει μια τιμή και εννέα άλλα threads περιμένουν έτσι ώστε να παραχθεί και στη συνέχεια να την εμφανίσουν.

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> // sleep

#define T 10

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond_var = PTHREAD_COND_INITIALIZER;

int shared_value = -1;
int flag = 0;
void *produce(void *tid) {
  long mytid = (long)tid;
  printf("Producer thread %ld is busy, consumers have to wait\n", mytid);
  sleep(2);
  shared_value = 42;
  printf("Thread %ld produced shared value %d\n", mytid, shared_value);
  pthread_mutex_lock(&lock);
  flag = 1;
  if (flag == 1)
    pthread_cond_broadcast(&cond_var);
  pthread_mutex_unlock(&lock);
  return NULL;
}

void *consume(void *tid) {
  long mytid = (long)tid;
  pthread_mutex_lock(&lock);
  if (flag == 0)
    pthread_cond_wait(&cond_var, &lock);
  pthread_mutex_unlock(&lock);
  printf("Consumer thread %ld reads shared value %d\n", mytid, shared_value);
  return NULL;
}

int main() {
  pthread_t thread_handles[T];
  pthread_create(&thread_handles[0], NULL, produce, (void *)0);
  for (long t = 1; t < T; t++)
    pthread_create(&thread_handles[t], NULL, consume, (void *)t);

  for (long t = 0; t < T; t++)
```

```
    pthread_join(thread_handles[t], NULL);

  printf("Main: shared value %d\n", shared_value);
  pthread_mutex_destroy(&lock);
  pthread_cond_destroy(&cond_var);
  return 0;
}
```

pthreads_example04b.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
gcc pthreads_example04b.c -o pthreads_example04b -lpthread
```

```
./pthreads_example04b
```

```
Producer thread 0 is busy
Thread 0 produced shared value 42
Consumer thread 2 reads shared value 42
Consumer thread 3 reads shared value 42
Consumer thread 1 reads shared value 42
Consumer thread 4 reads shared value 42
Consumer thread 5 reads shared value 42
Consumer thread 6 reads shared value 42
Consumer thread 7 reads shared value 42
Consumer thread 8 reads shared value 42
Consumer thread 9 reads shared value 42
Main: shared value 42
```

## Παράδειγμα 5Α (barriers)

Δέκα νήματα ξεκινούν και καθένα από αυτά εμφανίζει πρώτα το μήνυμα Phase A και μετά το μήνυμα Phase B. Θέλουμε πρώτα να εμφανιστούν όλα τα μηνύματα A και μετά όλα τα μηνύματα B.

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define T 10
pthread_barrier_t bar;

void *thread_func(void *tid) {
    long mytid = (long)tid;
    printf("Thread %ld phase A\n",mytid);
    pthread_barrier_wait(&bar);
    printf("Thread %ld phase B\n",mytid);
    return NULL;
}

int main() {
  pthread_barrier_init(&bar, NULL, T);
  pthread_t thread_handles[T];
  for (long t = 0; t < T; t++)
    pthread_create(&thread_handles[t], NULL, thread_func, (void *)t);

  for (long t = 0; t < T; t++)
```

```
    pthread_join(thread_handles[t], NULL);
  pthread_barrier_destroy(&bar);
  return 0;
}
```

pthreads_example05a.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
gcc pthreads_example05a.c -o pthreads_example05a -lpthread
```

```
./pthreads_example05a
```

```
Thread 0 phase A
Thread 1 phase A
Thread 2 phase A
Thread 4 phase A
Thread 5 phase A
Thread 6 phase A
Thread 7 phase A
Thread 8 phase A
Thread 3 phase A
Thread 9 phase A
Thread 4 phase B
Thread 1 phase B
Thread 6 phase B
Thread 5 phase B
Thread 8 phase B
Thread 7 phase B
Thread 3 phase B
Thread 0 phase B
Thread 2 phase B
Thread 9 phase B
```

## Παράδειγμα 5Β (barriers)

Μια κοινόχρηστη μεταβλητή με αρχική τιμή 0 πρώτα αυξάνεται κατά 1 από 10 νήματα και εφόσον ολοκληρωθεί η φάση αυτή καθένα από τα 10 νήματα διπλασιάζει την τιμή της.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define T 10

pthread_barrier_t bar;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
int shared_variable=0;

void *thread_func(void *tid) {
    long mytid = (long)tid;
    pthread_mutex_lock(&lock);
    shared_variable++;
    printf("Thread %ld added 1 to the shared value, shared value=%d\n",mytid,shared_vari
able);
    pthread_mutex_unlock(&lock);
```

```c
    pthread_barrier_wait(&bar);
    pthread_mutex_lock(&lock);
    shared_variable*=2;
    printf("Thread %ld doubled shared value, shared value=%d\n",mytid,shared_variable);
    pthread_mutex_unlock(&lock);
    return NULL;
}

int main() {
  pthread_barrier_init(&bar, NULL, T);
  pthread_t thread_handles[T];
  for (long t = 0; t < T; t++)
    pthread_create(&thread_handles[t], NULL, thread_func, (void *)t);
  for (long t = 0; t < T; t++)
    pthread_join(thread_handles[t], NULL);
  pthread_barrier_destroy(&bar);
  printf("The final result is %d\n", shared_variable);
  return 0;
}
```

pthreads_example05b.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
gcc -pthread pthreads_example05b.c -o pthreads_example05b –lpthread
```

```
./pthreads_example05b
```

```
Thread 0 added 1 to the shared value, shared value=1
Thread 1 added 1 to the shared value, shared value=2
Thread 2 added 1 to the shared value, shared value=3
Thread 3 added 1 to the shared value, shared value=4
Thread 4 added 1 to the shared value, shared value=5
Thread 5 added 1 to the shared value, shared value=6
Thread 6 added 1 to the shared value, shared value=7
Thread 7 added 1 to the shared value, shared value=8
Thread 8 added 1 to the shared value, shared value=9
Thread 9 added 1 to the shared value, shared value=10
Thread 0 doubled shared value, shared value=20
Thread 8 doubled shared value, shared value=40
Thread 1 doubled shared value, shared value=80
Thread 2 doubled shared value, shared value=160
Thread 3 doubled shared value, shared value=320
Thread 4 doubled shared value, shared value=640
Thread 5 doubled shared value, shared value=1280
Thread 6 doubled shared value, shared value=2560
Thread 7 doubled shared value, shared value=5120
Thread 9 doubled shared value, shared value=10240
The final result is 10240
```