```
#include <pthreads/pthread.h>

void new_thread(arg)
  int *arg;
{
  printf("new thread argument = %d\n", *arg)
}

main()
{
  pthread_t th;
  int i = 1;

  pthread_create(&th, NULL, new_thread, &i);
  pthread_join(th, NULL);
  pthread_detach(&th);
}
```

```
typedef struct list {
  struct list *next;
  item t;
} *list_t;

typedef struct shared_list {
  list_t head;
  pthread_mutex_t lock;
} *shared_list_t;

void shared_list_add(q, t)
  shared_list_t q;
  item t;
{
  list_t *new;

  new = malloc(sizeof(*new));
  pthread_mutex_lock(&q->lock);
  new->next = q->head;
  q->head = new;
  pthread_mutex_unlock(&q->lock);
}
```

```
    Program compute_pi
    integer n, i
    double precision w, x, sum, pi, f, a
c function to integrate
    f(a) = 4.d0 / (1.d0 + a*a)
    print *, ‘Enter number of intervals: ‘
    read *,n
c calculate the interval size
    w = 1.0d0/n
    sum = 0.0d0
!$OMP PARALLEL DO PRIVATE(x), SHARED(w)
!$OMP& REDUCTION(+: sum)
    do i = 1, n
    x = w * (i - 0.5d0)
    sum = sum + f(x)
    enddo
    pi = w * sum
    print *, ‘computed pi = ‘, pi
    stop
    end
```

**Computing PI serially and in parallel using OpenMP.**

```c
#include <pthread.h>
#include <stdio.h>
pthread_mutex_t reduction_mutex;
pthread_t *tid;
int n, num_threads;
double pi, w;
double f(a)
double a;{ return (4.0 / (1.0 + a*a));}
---------------------------------------------------------------
void *PIworker(void *arg){
     int i, myid;
     double sum, mypi, x;
     /* set individual id to start at 0 */
     myid = pthread_self()-tid[0];
     /* integrate function */
     sum = 0.0;
     for (i=myid+1; i<=n; i+=num_threads) {
          x = w*((double)i - 0.5);
          sum += f(x);
     }
     mypi = w*sum;
     /* reduce value */
     pthread_mutex_lock(&reduction_mutex);
     pi += mypi;
     pthread_mutex_unlock(&reduction_mutex);
     return(0);
}
-----------------------------------------------------------------
void main(argc,argv)
int argc;
char *argv[];
{
     int i;
     /* get num intervals and num threads from command line */
     n = atoi(argv[1]);
     num_threads = atoi(argv[2]);
     w = 1.0 / (double) n;
     pi = 0.0;
     tid = (pthread_t *) calloc(num_threads, sizeof(pthread_t));
     /* initialize lock */
     if (pthread_mutex_init(&reduction_mutex, NULL))
          fprintf(stderr, “Cannot init lock\n”), exit(1);
     /* create the threads */
     for (i=0; i<num_threads; i++)
          if(pthread_create(&tid[i], NULL, PIworker, NULL))
               fprintf(stderr,”Cannot create thread %d\n”,i), exit(1);
     /* join threads */
     for (i=0; i<num_threads; i++)
          pthread_join(tid[i], NULL);
     printf(“computed pi = %.16f\n”, pi);
}
```
**Computing PI in parallel using pthreads.**