Exercise 4: Threads

1.1 Ans

```
nrbaboo@ubuntu: ~/Desktop/OS

nrbaboo@ubuntu:~/Desktop/OS$ ./nr
My pid is 3542
nrbaboo@ubuntu:~/Desktop/OS$
```

3.1 Ans

```
nrbaboo@ubuntu: ~/Desktop/OS
^
nrbaboo@ubuntu:~/Desktop/OS$ ./nr
HELPER
MAIN
nrbaboo@ubuntu:~/Desktop/OS$
```

How can we modify the code above to always print out "HELPER" followed by "MAIN"?

Ex4.c (~/Desktop/OS) - gedit

Open

```c
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
void *helper(void *arg) {
        printf("HELPER\n");
        return NULL;
}
int main() {
        pthread_t thread;
        pthread_create(&thread, NULL, &helper, NULL);
        pthread_join(thread, NULL);
        printf("MAIN\n");
        return 0;
}
```

3.2 Ans

```
int main() {
        pthread_t thread[1000];

        clock_t start = clock();

        for(int i=0;i<1000;i++)
        {
                pthread_create(&thread[i], NULL, &helper, NULL);
        }
        clock_t end = clock();
        float seconds = (float)(end - start) / CLOCKS_PER_SEC;

        printf("%f", seconds);
```

```
HELPER
HELPER
HELPER
HELPER
0.031078nrbaboo@ubuntu:~/Desktop/OS$
```

3.3 Ans

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

void thread1fn(void *arg) {
int i=0;
        while(1)
        {
                if(i%10000000==0)
                {
                        printf(".");
                        i = 0;
                }
                i++;
        }
        return NULL;
}

void thread2fn(void *arg) {

        while(1){
                char ch[1000];
                scanf("%s",&ch);
                printf("Thank you for your input.\n");
        }
}
int main() {
        pthread_t thread1;
        pthread_t thread2;
        pthread_create(&thread1, NULL, &thread1fn,NULL);
        pthread_create(&thread2, NULL, &thread2fn,NULL);
        pthread_join(thread1, NULL);
        pthread_join(thread2, NULL);
        return 0;
}
```

```
nrbaboo@ubuntu: ~/Desktop/OS
In file included from test1.c:2:0:
/usr/include/pthread.h:233:12: note: expected 'void * (*)(void *)' but argument
is of type 'void (*)(void *)'
 extern int pthread_create (pthread_t *__restrict __newthread,
            ^
nrbaboo@ubuntu:~/Desktop/OS$ ./test1
A
.......................................................................
.......................................................................Thank yo
u for your input.
B
.......................................................................
.......................................................................
......................Thank you for your input.
C
.......................................................................
.......................................................................
.........................................Thank you for your input.
.......................................................................
.......................................................................
```

3.4 Ans

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

int size, num_threads;
double **matrix1, **matrix2, **matrix3;

double ** allocate_matrix( int size )
{
  /* Allocate 'size' * 'size' doubles contiguously. */
  double * vals = (double *) malloc( size * size * sizeof(double) );

  /* Allocate array of double* with size 'size' */
  double ** ptrs = (double **) malloc( size * sizeof(double*) );

  int i;
  for (i = 0; i < size; ++i) {
    ptrs[ i ] = &vals[ i * size ];
  }

  return ptrs;
}

void init_matrix( double **matrix, int size )
{
  int i, j;

  for (i = 0; i < size; ++i) {
    for (j = 0; j < size; ++j) {
      matrix[ i ][ j ] = 1.0;
    }
  }
}

void print_matrix( double **matrix, int size )
{
  int i, j;

  for (i = 0; i < size; ++i) {
    for (j = 0; j < size-1; ++j) {
      printf( "%lf, ", matrix[ i ][ j ] );
    }
    printf( "%lf", matrix[ i ][ j ] );
    putchar( '\n' );
  }
}
```

```
/**
 * Thread routine.
 * Each thread works on a portion of the 'matrix1'.
 * The start and end of the portion depend on the 'arg' which
 * is the ID assigned to threads sequentially.
 */
void * worker( void *arg )
{
  int i, j, k, tid, portion_size, row_start, row_end;
  double sum;

  tid = *(int *)(arg); // get the thread ID assigned sequentially.
  portion_size = size / num_threads;
  row_start = tid * portion_size;
  row_end = (tid+1) * portion_size;

  for (i = row_start; i < row_end; ++i) { // hold row index of 'matrix1'
    for (j = 0; j < size; ++j) { // hold column index of 'matrix2'
      sum = 0; // hold value of a cell
      /* one pass to sum the multiplications of corresponding cells
              in the row vector and column vector. */
      for (k = 0; k < size; ++k) {
             sum += matrix1[ i ][ k ] * matrix2[ k ][ j ];
      }
      matrix3[ i ][ j ] = sum;
    }
  }
}

int main( int argc, char *argv[] )
{
  int i;
  double sum = 0;
  struct timeval tstart, tend;
  double exectime;
  pthread_t * threads;

  if (argc != 3) {
    fprintf( stderr, "%s <matrix size> <number of threads>\n", argv[0], argv[1] );
    return -1;
  }

  size = atoi( argv[1] );
  num_threads = atoi( argv[2] );

  if ( size % num_threads != 0 ) {
    fprintf( stderr, "size %d must be a multiple of num of threads %d\n",
                 size, num_threads );
    return -1;
  }

  threads = (pthread_t *) malloc( num_threads * sizeof(pthread_t) );

  matrix1 = allocate_matrix( size );
  matrix2 = allocate_matrix( size );
  matrix3 = allocate_matrix( size );

  init_matrix( matrix1, size );
```

```
  init_matrix( matrix2, size );

  if ( size <= 10 ) {
    printf( "Matrix 1:\n" );
    print_matrix( matrix1, size );
    printf( "Matrix 2:\n" );
    print_matrix( matrix2, size );
  }
  gettimeofday( &tstart, NULL );
  for ( i = 0; i < num_threads; ++i ) {
    int *tid;
    tid = (int *) malloc( sizeof(int) );
    *tid = i;
    pthread_create( &threads[i], NULL, worker, (void *)tid );
  }

  for ( i = 0; i < num_threads; ++i ) {
    pthread_join( threads[i], NULL );
  }
  gettimeofday( &tend, NULL );

  if ( size <= 10 ) {
    printf( "Matrix 3:\n" );
    print_matrix( matrix3, size );
  }
  exectime = (tend.tv_sec - tstart.tv_sec) * 1000.0; // sec to ms
  exectime += (tend.tv_usec - tstart.tv_usec) / 1000.0; // us to ms

  printf( "Number of MPI ranks: 0\tNumber of threads: %d\tExecution time:%.3lf sec\n",
        num_threads, exectime/1000.0);
  return 0;
}
```

This figure show that the result of multiplication matrix size 5x5