```
typedef struct{
  int tid;
  matrix* matrixA;
  matrix* matrixB;
  matrix* matrixC;
}thread_args; // structure for thread function arguments

pthread_mutex_t mutex;

int numOfThread;

void *threadFunction(void *data){ // function every thread will compute
    thread_args* args=(thread_args*) data; ;

    int TID = args->tid;

    int rowPT; // thread function needs his own vars
    int colPT;
    int addPT;
    int tempSumPT;

    int matARows = args->matrixA->rows;
    int matACols = args->matrixA->cols;
    int matBCols = args->matrixB->cols;

    int numOfIters =  matARows / numOfThread; // calculate the number of iteration for every thread
    int lastNumOfIters = numOfIters + matARows - (numOfIters * numOfThread);
    int startIters = numOfIters*TID;

    if(TID==numOfThread-1){
        numOfIters=lastNumOfIters;
    }
    int endIters=startIters+numOfIters;

    for(rowPT = startIters ; rowPT < endIters ; rowPT++){
        for(colPT = 0 ; colPT < matBCols ; colPT++ ){

                tempSumPT=0;
            for( addPT = 0 ; addPT < matACols ; addPT++ ){
                tempSumPT += (int) args->matrixA->matrix[rowPT][addPT] * (int) args->matrixB->matrix
[addPT][colPT];
            }

            pthread_mutex_lock(&mutex);
              args->matrixC->matrix[rowPT][colPT] = tempSumPT;
            pthread_mutex_unlock(&mutex);

        }
    }

    pthread_exit(NULL);

}

// function for parallel multiplication with pthread
multiReturn matrixMultiPTH(matrix matrixA, matrix matrixB, matrix matrixC){

 numOfThread=numOfProc; // number of threads = number of processors

 printf("\nStart parallel multiplication with pthreads ... \n");
 double time1=omp_get_wtime();

  int t;
  pthread_t threads[numOfThread];
  thread_args args[numOfThread];

  pthread_mutex_init(&mutex,NULL);

  for(t=0;t<numOfThread;t++){ // create thread_arguments and threads
        args[t].tid=t;
        args[t].matrixA=&matrixA;
        args[t].matrixB=&matrixB;
```

```
        args[t].matrixC=&matrixC;
                pthread_create(&threads[t], NULL, threadFunction, (void *) &args[t]);
  }
  for(t=0;t<numOfThread;t++){ // wait untill all threads are finished
                pthread_join(threads[t], NULL);
  }

  double time2=omp_get_wtime();
  printf("finished\n\n");

  multiReturn mr = {"PTHREAD",time2-time1};

  return mr;
}
```