CS201 Spring 2019
Assignment 5: threads
Undergraduate assignment
15 points
Due: Friday, March 29th, at 11:59 pm

Create a multithreaded program in C, using pthreads, to compute a matrix-vector product (Ax = y).

First, get `pthreads-example.c` from the class gitlab repo. Compile this and run it. This program shows an example of how to create pthreads and pass data to them. On kaladin, you will need to compile this in the following way:

```
$ gcc pthreads-example.c -lpthread
```

Then, create your own new file `mxv.netid.c`

In your code, use this:

```
#define N 8
```

Declare a matrix, a vector, and a result vector:

```
double A[N][N];
double x[N], y[N];
```

Define a helper struct:

```
typedef struct {
  double *row;
  double *v;
  double result;
} Info;
```

Initialize each entry of your matrix to a uniformly distributed random number in the range (-0.5, 0.5):

```
for (i=0; i<N; ++i) {
  for (j=0; j<N; ++j) {
    A[i][j] = drand48() - 0.5;
  }
}
```

(See below for note about Windows.)

Initialize each entry of the vector x also in this way.

Then, create N threads to do each row-vector dot product.

Each thread will get a pointer to an Info struct that has the information filled in telling it which row it is supposed to use; and each thread puts its result in the `result` field of the struct.

In memory, A will be layed out this way: row1 row2 row3 row4 ... rowN
So the elements of A[][] will appear like this:

first row of A	second row of A	...	last row of A

More specifically:

A[0][0] A[0][1] A[0][2] ... A[0][N-1] A[1][0] A[1][1] A[1][2] ... A[1][N-1] ... A[N-1][0] A[N-1][1] A[N-1][2] ... A[N-1][N-1]

If I want to point to the fifth row of A, I can do this:

info->row = &A[4][0];

The general structure of your program will look like this:

```
Info info[N];
pthread_t tid[N];
for (i=0; i<N; ++i) {
  info[i].row = &A[i][0];
  info[i].v = x;
  pthread_create(&tid[i], NULL, multiply, &info[i]);
}

for (i=0; i<N; ++i) {
  pthread_join(tid[i], NULL);
}
```

and you will have a function like this:

```
void *multiply(void *data) {
  Info *info = (Info *) data;
  // here's where you do the work
}
```

How can you test whether your implementation is correct? In your program, you can also do the matrix-vector multiplication in a single-threaded fashion, in your main(). Then, compare results!

Here's how to test whether two floating-point numbers are approximately equal:

For two floating-point numbers x1 and x2, x1 is approximately equal to x2 if the relative difference of x1 and x2 is very small.

In other words, if |x1 - x2| / |x1| < tolerance, then x1 and x2 are close.

But, suppose that x1 itself is very small (or even zero). Then the division by |x1| is problematic.

The safe way to do this test is:

|x1 - x2| < tolerance * |x1|, with tolerance set to something like 1e-12.

If this inequality is true, then x1 is very close to x2.

Put your code in a file named mxv.netid.c

Please make sure your code will compile and run on kaladin. Up until now, we have not been taking off points for this,;but starting with this assigment, we will deduct a little if we have to modify your code to make it compile and/or run on kaladin.

Note for Sindows users: drand48() is not available in Windows. Instead use rand(). The rand() function returns a pseudorandom integer in the range 0 to RAND_MAX (32767). So to get a uniformly distributed random number in the range (-0.5, 0.5), do this: r = ((double) rand() / RAND_MAX) - 0.5