Assignment #3: Threads

CS201 Fall 2022
25 points, due Saturday, Mar. 4th, 11:59 pm
second chance (for 22.5 points): due TBD

1 Threads

You'll write a multithreaded program to look for the longest sequence of digits in the first 10000 digits of pi such that the last digit in the sequence is the sum of the previous digits in the sequence. You may work alone or with a partner.

You'll do this in by writing two functions:

```
int readFile(char *filemame, int *numChars, char *buffer);
void *findMaxSumSeq(void *param);
```

These functions are described below.

1.1 Example program

First, get `pthreads-example-simpler.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 silk, you will need to compile this in the following way:

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

You can also run this on macOS (and possibly on Windows?). To compile it at the command line on macOS:

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

2 Reading a File

Create your own file `threads.netid.c`

Then, write the function `int readFile(char *fileName, int *numChars)`. This file will read lines from the specified file and will put each of these lines into a large character buffer. On success, it will return zero; otherwise, it will return a nonzero value.

Here's how you'll call your function:

```
#define BUFLEN 10100

int main(int argc, char *argv[]) {
  int rc, numChars;
  char buffer[BUFLEN];

  if (argc < 2) {
    printf("ERROR: need a filename\n");
    return 8;
  }
```

```
  rc = readFile(argv[1], &numChars, buffer);
  if (rc != 0) {
    return 8;
  }

  // here's where you'll create the threads that do the actual searching

  return 0;
}
```

2.1 Opening a file for reading

Open (initialize) the file for reading this way:

```
FILE *fp = fopen(filename, "r"); // filename is the char* passed to your readFile() function
if (fp == NULL) {
  printf("ERROR: cannot open file '%s'\n", filename);
  return 1;
}
// now you can read from the file
```

2.2 Reading a line from a file

Use the function `char *fgets(char *s, int size, FILE *fp)`. This function will read at most `size-1` characters from the open file `fp` into the buffer `s`. In a while loop, call this function to read a single line from the input file. Read each line into a buffer of size 256 bytes. Make a local variable in your `readFile()` function: `char line[LINELEN]`, and `#define LINELEN 256` near the top of your `threads.netid.c` file.

You can than call `fgets()` this way:

```
char *chp = fgets(line, LINELEN, fp);
```

If the return value from `fgets()` is `NULL`, then you've reached the end of file—this tells you to break out of your while loop. Otherwise, put the characters from `line` at the end of `buffer`. In this way, you'll read line after line from the file, adding the characters you read each time to `buffer`, so that at the end, `buffer` will have every character from the file.

There is a quirk about `fgets()` that you'll have to deal with: it will read characters from the file until it has read a newline character or `size-1` characters, whichever comes first. So for example, if `size` is 80 and there is a line containing fewer than 80 characters, then the string it returns will have a newline character (`'\n'`) at the end. So, after you read a line into a `line`, check whether `line[strlen(line)-1]` is `'\n'`. If it is, then truncate the string before the newline character. Here's an example:

```
chp = fgets(line, LINELEN, fp);
if (chp == NULL) {
  printf("file is empty\n");
  fclose(fp);
  return 8;
}

while (chp != NULL) {
  len = strlen(line);
  if (line[len-1] == '\n') {
    line[len-1] = '\0';
    len = len - 1;
  }
```

```
  for (i=0; i<len; ++i) {
    // append the contents of the line[] buffer to the big buffer
  }
  chp = fgets(line, LINELEN, fp);
} // while not at end of file
```

Test your program with a short file: `alphabet.asc`. This contains the 26 letters of the alphabet, spread over several lines. After you read this file, you should have the string `abcdefghijklmnopqrstuvwxyz` in your `buffer`.

2.3 Converting text to numbers

You must convert the ASCII digits to integers. The digit "0" in ASCII is 48; the digit "1" is 49, etc. So to convert, subtract 48 from each element of your array after you read it in. You can still store the converted numerical digits in a `char[]` array, because each digit will represent a number in the range from zero to nine (and will thus fit in an eight-byte `char`). Ignore the decimal point in the input.

2.4 Creating threads

Put this line in your program:

```
#define NUM_THREADS 4
```

This is the number of threads you'll create.

Create `NUM_THREADS` threads, using `pthread_create()` to create each thread.

Use this structure to pass information to each thread:

```
typedef struct {
  char *A;     // the digits themselves--this is a pointer to your big buffer
  int start;   // first position that thread should look at
  int end;     // last position that this thread should look at
  int bestpos; // ending position of the best sequence found by this thread
  int max;     // length of the best sequence found by this thread
} ThreadInfo;
```

Declare an array of these structs, of length `NUM_THREADS`. Also make an array of `pthread_t`, to hold each thread ID that `pthread_create()` will give you.

Each thread will look at a different region of the `buffer[]`. Think about what the start index and end index should be for each thread.

What if there is a sequence that spans two regions? A crude way to account for this is to pad each region by a margin value. If we think that the length of the best sequence will be at most 10, then pad each region by 10, so that each thread looks at a little bit of its neighbors' region also. (But pay attention to the overall start and end indices of the buffer; in other words, don't look at any characters before the start or after the end.)

Then, think what needs to happen after each thread has looked in its region.

Important: do not use any global variables for this assignment. Any information from the `main()` that the threads need must be passed through the `ThreadInfo` structure.

2.5 Output

Print out the longest sequence found and the position of the final element of the sequence.

3 Testing

Test this on a small file first: `pi100.asc`. The longest subsequence is four: 6208, and the 8 is in position 77. (There is also a subsequence of length four, 3238, and the 8 is in position 18.)

Then, try it on `pi10000.asc`. The longest subsequence is of length nine: 141010029, and the 9 is position 7774.

4 Key Points

Here are the key points and things to remember for this assignment:

- You'll specify a maximum value for the number of characters you'll read from a file (`BUFLEN`), but you might actually read fewer—use the `numChars` value you get from the call to your `readFile()` function.
- Every thread needs its own individual instance of `ThreadInfo`. The best way to achieve this is with an array.
- Threads are independent—we cannot predict the relative order of their start and their processing.
- Any work that needs to be done after all threads are complete must wait until they are complete—the `pthread_join()` calls are the key.

5 What to Submit

Submit your `threads.netid.c` file.

6 Graduate Students

Graduate students, and undergraduates for extra credit: Look for sequences where the final element of the sequence is a two-digit number.

There is a run of length 32 in `pi10000.asc`: 40317211860820419000422966171196, and the 96 starts at position 4806.

In addition, do experiments with runtime and thread count. For this, generate a large array of random digits to use as input (rather than reading digits from a file). Do several runs of your program using various numbers of threads and various sizes of input (for example, one million, ten million); and time how long your program takes, using `gettimeofday()`. I've put an example program in gitlab that shows how to use `gettimeofday()`. Plot your data (elapsed time vs. #threads, for various problem sizes).

Put your code in a file named `threads-grad.netid.c`.