

San Francisco State University

Operating Systems

EC Project Report

23 Dec 2016

Calculating value of pie(π) with Threads & Mutex

In Operating Systems Multithreading is a specialized form of multitasking and a multitasking is the feature that allows your computer to run two or more programs concurrently. In general, there are two types of multitasking: process-based and thread-based.

Thread-based multitasking deals with the concurrent execution of pieces of the same program. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. C/C++ does not contain any built-in support for multithreaded applications. Instead, it relies entirely upon the operating system to provide this feature. We assume that you are working on Linux Environment and we are going to write multi-threaded C/C++ program using POSIX. POSIX Threads, or Pthreads provides API which are available on many Unix-like POSIX Operating systems such as FreeBSD, NetBSD, GNU/Linux, Mac OS X and Solaris.

Pie is an irrational number which means it is a real number that cannot be written as a simple fraction. Following is the formula for calculating pie:

$$\text{Pi} = 4 * (1 - 1/3 + 1/5 - 1/7 + 1/9 \dots)$$

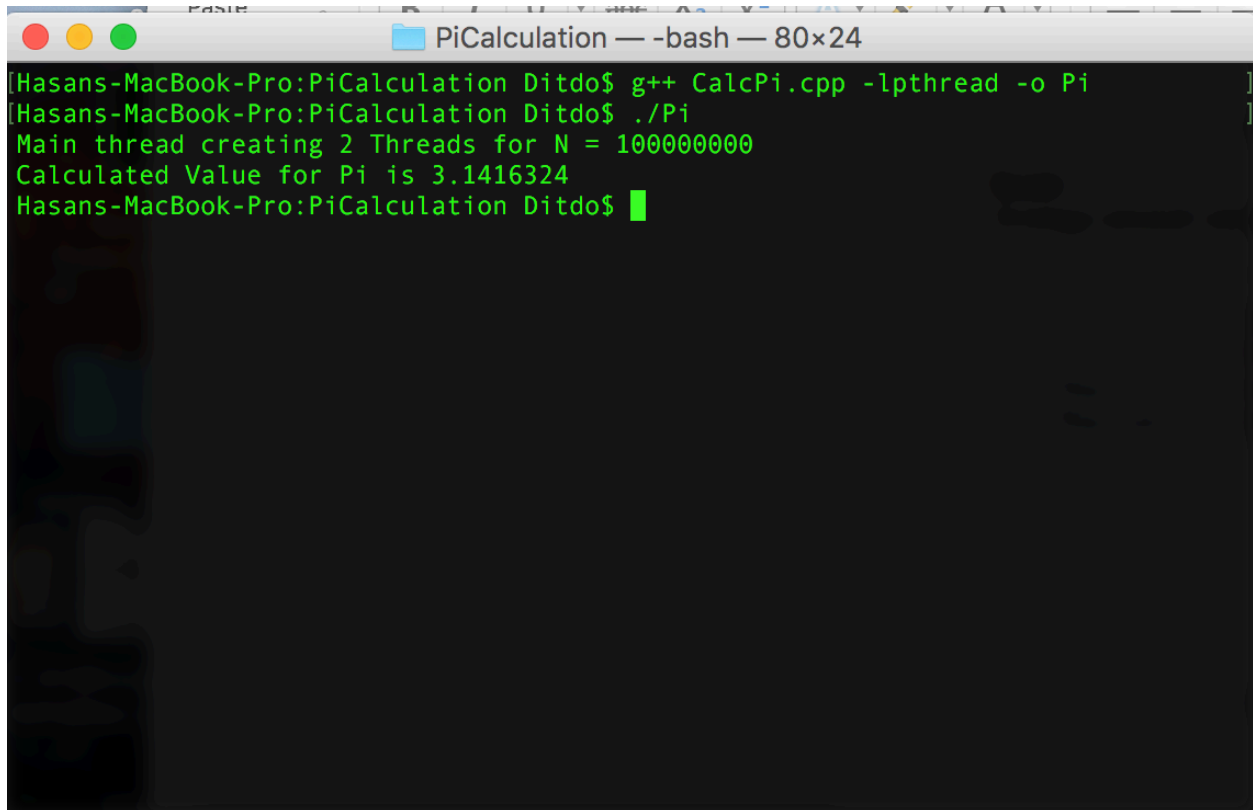
I have used 3 methods to calculate value of pie

- CalcPi (just threads are used in it)
- CalcPiSafe (threads +Mutex are used)
- CalcPiSafeFast (threads +Mutex are used with own variables)

1.CalPi

This program creates multiple threads to parallelize estimation for Pi.

This code carries possible race condition, as all threads simultaneously store the results in same variable (sum) This means, every run of this program may produce different results.

A screenshot of a macOS terminal window titled "PiCalculation — -bash — 80x24". The terminal shows the following commands and output:

```
[Hasans-MacBook-Pro:PiCalculation Ditdo$ g++ CalcPi.cpp -lpthread -o Pi ]
[Hasans-MacBook-Pro:PiCalculation Ditdo$ ./Pi ]
Main thread creating 2 Threads for N = 100000000
Calculated Value for Pi is 3.1416324
Hasans-MacBook-Pro:PiCalculation Ditdo$ █
```

Results

Average errored values, Used Dual core Intel Processor

	<i>n</i>			
	10^5	10^6	10^7	10^8
π	3.14159	3.141593	3.1415927	3.14159265
1 Thread	3.14158	3.141592	3.1415926	3.14159264
2 Threads	3.14158	3.141480	3.1413692	3.14164686

2.CalcPiSafe

This program creates multiple threads to parallelize estimation for Pi.

This code resolves the race condition problem by using Mutex. Thus, the results attained are thread safe. However, it results in sequential execution of some code segments that is all the threads will wait to enter into Mutex and it will take some time to execute, which is not an efficient solution. In practical testing it took few seconds to execute.

```
Hasans-MacBook-Pro:PiCalculation Ditdo$ g++ CalcPiSafe.cpp -lpthread -o PiSafe ]
Hasans-MacBook-Pro:PiCalculation Ditdo$ ./PiSafe ]
Main thread creating 2 Threads for N = 100000000
Calculated Value for Pi is 3.1415926
Hasans-MacBook-Pro:PiCalculation Ditdo$
```

3.CalcPiSafeFast

This program creates multiple threads to parallelize estimation for Pi.

It resolves the race condition and Mutex slow execution problem by giving every Mutex its own variables. Thus, the results attained are thread safe and execution time increased dramatically. In practical testing it took less than 1 seconds to execute.

```
[Hasans-MacBook-Pro:PiCalculation Ditdo$ g++ CalcPiSafeFast.cpp -lpthread -o PiSafeFast
[Hasans-MacBook-Pro:PiCalculation Ditdo$ ./PiSafeFast
Main thread creating 2 Threads for N = 100000000
Calculated Value for Pi is 3.1415926
Hasans-MacBook-Pro:PiCalculation Ditdo$ █
```

Functions

- **pthread_create** Creates a new posix thread, which executes the mat_calc function provided as argument # 3
- **pthread_join** Stops execution of main thread, until the joined thread (specified by argument # 1) has finished its execution.
- **pthread_mutex_init** Initializes Mutex (pthread_mutex_t) for use.
- **pthread_mutex_destroy** Frees any memory used by the Mutex.
- **pthread_mutex_lock** Allows only one thread (per Mutex) to pass through this line. Other caller threads wait until Mutex is freed by pthread_mutex_unlock.
- **pthread_mutex_unlock** Unlocks the critical section acquired by Mutex variable, which allows next caller to enter to critical section.