DRS: Project Submission

Lt Luis Sepulveda

Lt. Rseky Anderson

Daniel (Dare) E Oke

DRS: Project Submission

CS689-01

Distributed Software Systems

Introduction

Prime Factorization

Prime numbers are a set of all numbers that can only be equally divided by one and themselves. Examples of prime numbers include 2, 3,5 and 7. What very little people know is the importance of prime numbers and how the mathematical logic behind them has resulted in vital applications in our modern history today. Mathematician have been able to show that any whole number can be expressed as a product of primes. Only primes and nothing else. This rule is called the prime factorization rule.

The task of prime factorization may seem like a cool mathematical oddity at first but as the number to be factorized gets bigger, it gets more challenging. The best mathematicians and scientist have been able to determine that it is totally impossible to find a completely efficient algorithm for factorizing large numbers into primes. There is some limit to the size of number we can factorize into prime numbers. This fact is absolutely key to modern computer security. Modern encryption algorithms exploit the fact we can easily take two large primes and multiply them together to get a much larger number but that no computer has yet been created that can take the much larger number and very quickly figure out which of the two primes went into making it.

Pollards Rho Algorithm of Factorization

The Pollard’s Rho prime factorization is very fast for large composite numbers with small factors. This paper will describe our distributed system which performs prime factorization on large numbers using Pollard’s Rho algorithm. Let us look at some of the key concepts employed.

* Two numbers are said to congruent modulo n (x = y modulo n) if
  + Each leave the same reminder when divided by n
  + The absolute of their difference is a multiple of n
* Birthday Paradox: When assessing a set of people, the probability of two persons having the same birthday is high.
* Floyd’s Cycling Algorithm. If the tortoise and hare start at the same point and move in a cycle, as such as the hare goes twice the speed of the tortoise, they are bound/sure to meet at some point.

Our Algorithm is shown below

Start with random x and c. Take y equal to x and f(x) = x2 + c.

While a divisor isn’t obtained

Update x to f(x) (modulo n) [Tortoise Move]

Update y to f(f(y)) (modulo n) [Hare Move]

Calculate GCD of |x-y| and n

If GCD is not 1

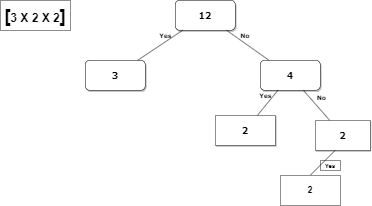
If GCD is n, repeat from step 2 with another set of x, y and c

Else GCD is our answer

Once we are given a number N, our system generates two sets of random numbers using a random number generator. The two random Numbers are set to variables x and c. Variable x is copied into a variable named y. While a divisor is not yet found, the tortoise move is performed on x and the hare moves is performed on y. The greatest common denominator of the difference of x-y and n is computed. If the GCD is equal to n repeat steps (see above). If GCD is not n and it is not 1, it must be our answer!

Testing Performed

Our team multiplied all the numbers from our output from application to ensure that we arrived back at N. More detailed explanation will be given is subsequent chapters. See diagram below to illustrate (we assume N = 12)



Concepts (Background)

An iterative Server is a server that handles request and returns a response to the requesting client. It iterates through each client, handling it one at a time. The main server (DivFinder.cpp) in our project was modeled as an iterative server.

A concurrent Server handles multiple clients request at the same time by passing them to a thread or a process after which it waits for next incoming request. The server may call a fork function, creating one child process for each child. An alternative is also to pass client request to threads. This utilized three concurrent server which objects of DivFinderSP.cpp, each of which spun several threads based on messages passed to them from the main server.

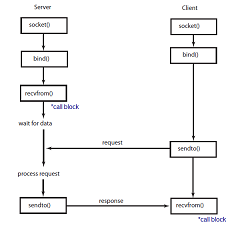
What do we actually mean by the term concurrency? When we talk about concurrency when it relates to computers (servers/clients), we are referring to single systems performing multiple independent activities in parallel rather than sequentially. Historically, most computers had just one processor with a single unit processing core; although some these computers still exist today. Such machines, although may have appeared mildly fast, could really only perform one task at a time. The trick was that it could switch between tasks many times per seconds. This is called task switching. The task switch provides an illusion to both the user and the application itself. Because there’s an illusion of concurrency behavior maybe subtly different when executing in a single process task switching environment compared to when executing in an environment with true concurrency. Computers containing multiple processors (cores) a now mostly being used for servers and high-performance computing tasks. The PCs are capable of running more than one task in parallel

Threads in distributed systems

A useful property of threads in distributed systems is that it can provide a convenient means of allowing blocking system calls with blocking the entire process in which the thread is running. This makes threads commonly used in distributed systems because it is easier to express communication by maintaining multiple logical connections at the same time. Our concurrent servers in our project utilizes this thread concept

Inter-process Communication with Sockets.

Sockets are an abstract endpoint of communication between a pair of processes. Developed by Berkley Software Distribution as a part of BSD UNIX, sockets are part of the I/O of an operating system. There are two types of sockets; datagram sockets and stream sockets. TCP/IP sockets are stream sockets while UDP/IP sockets are datagram sockets. The diagram below briefly describes the basic client-server communication.



Communication? Simple transient messaging with sockets? TCP sockets

Threads in distributed System.

Layered Architectural Style

Wrappers?

Iterative Server or Concurrent Server

Multicasting?

Naming?

Synchronization and co-ordination

Design and Methodology

describe the architecture of your system, design considerations,  
design of your testing methodology and justification for all

Results

data produced by your testing and analysis – how did changing parameters affect  
performance?

Creating a large

Conclusion

 final analysis, “future work” (i.e. ways your system would be expanded) and wrap  
things up

Appendix A

software dependencies (libraries, OS, compiler, etc) and compile/install instructions

Appendix B

Any raw data or more-detailed data

References

Anthony Williams (2012). *C++ Concurrency in Action: Practical Multithreading*. *Manning Publications*

https://www.extremetech.com/extreme/219570-what-are-prime-numbers-and-why-are-they-so-vital-to-modern-life

https://www.geeksforgeeks.org/pollards-rho-algorithm-prime-factorization/