Brute Force Cracking of Keys

When in doubt, use brute force -- Ken Thompson.

Objective

For this project you are to design, implement and document a parallel shared memory program that does a brute force search for a cryptographic key.

Background

This project is based upon the RSA secret-key challenge. The goal of the contests organised as part of the challenge is:

... to quantify the security offered by the government-endorsed data encryption standard (DES) and other secret-key ciphers with keys of various sizes. The information obtained from these contests is anticipated to be of value to researchers and developers alike as they estimate the strength of an algorithm or application against exhaustive key-search.

Unlike the RSA challenge, you will be working with the Blowfish algorithm. Blowfish is a variable key length, symmetric block cipher, designed in 1993 by Bruce Schneier and included in a large number of cipher suites and encryption products.

You will be provided with a ciphertext, the expected plaintext and a range of the key space to search.

Your key search system will find the key used to encrypt the plaintext/decrypt the ciphertext.

The search will be parallelised by distributing the work across multiple clients.

Requirements

Your implementation will use a “bag of tasks” architecture. There will be a single key manager and a dynamic set of clients.

The system should meet the following minimum requirements:

1. Clients only need to be aware of the location of the key manager.
2. Clients can join or leave but will complete the work they have been requested.
3. Clients request work from the key manager and return results to it.
4. Connections between clients and the Masters only exist long enough to request work or to return results.
5. When the key is found, the key manager will shutdown.

You will implement two programs (KeyManager and Client) and evaluate their performance.

Key Manager

The key manager handles clients joining the system, allocates them key spaces to search and reports the results.

The key manager is invoked from the command line (as shown below) and **does not have a GUI interface**.

```
java KeyManager initial-key keysize ciphertext
```

Where the arguments are:

initial-key is the starting point for the search, the key is expressed as an integer value.
keysize defines the key space (expressed in number of bytes)
ciphertext is the ciphered text encoded as a base64 string

When started the manager should print out a random port number that will be used by clients wanting to request work.

When the key has been found you should print a message on the command line indicating success and print out the key itself, similarly a message indicating failure should be printed when no key has been found and the key space has been exhausted.

In addition, the manager should keep track of how long it takes to either find the key or come to the end of the key space and print this when it exits.

The Client Program

The client program is also invoked from the command line be able to be invoked as follows from the command line:

```
java Client hostname portnumber chunksize
```

Where the arguments are:

hostname is the name of the machine where the key manager is running
portnumber is the port number used by the key manager service
chunksize is the number of keys that the client will request each time from the client

It is assumed that a client will keep requesting tasks until no more tasks are available from the key manager or the user makes the client shutdown (using CONTROL C).

Tasks

Although we specify different submission items you should compile the different written components into a report. The code for the programs should be included in the report itself as well as being submitted independently so we can run the code if necessary.

Task 1: Understanding the code [10 marks].

Download project2.jar and test that the following works.

Demonstrate your understanding by modifying EchoClient and EchoServer to implement a different knock knock joke and Search to search a user specified number of keys instead of the current fixed ones.

Submit:

1. The modified EchoClient and EchoServer as well as the modified Search.
2. Include a README file that highlights the changes that you have made and evidence that the programs function correctly.

Task 2: Design your system and document the design [30 marks].

Design your system in terms of what each program does and the messages exchanged between the programs over the network.

Submit:

1. An outline of the control flow in each program and the messages exchanged between the two programs. This should be in English and the level of abstraction should be quite high. For example, client sends a work request for chunksize keys and waits for a response specifying X.

2. A short document explaining how this design satisfies the each of the system requirements.

Task 3: Implement your system and test its functionality [30 marks].

Implement the programs and test their functionality. We assume that unless you provide convincing evidence that we will assume it doesn't implement a given piece of functionality.

Submit:

1. The code for the programs, the programs should be documented and well-structured.

2. A test plan that shows how you test that all of the system requirements have been satisfied and that you have met the individual program requirements.

3. Evidence that you have carried out the tests and an evaluation of whether your programs are correct or not.

Task 4: Performance evaluation [15 marks].

Investigate how the performance of your system changes (in terms of total time taken to find a key) when you (a) change the number of clients, and ; (b) change the amount of work done by each. You should run each combination 30 times and calculate the average time as well as the standard deviations to allow you to determine if there are statistically significant differences between the variables.

It would be sufficient to test three by three conditions.

Submit:

A short report specifying: (a) how you conducted the experiments; (b) the results of the experiments; and, (c) your interpretation of the results.

Task 5: Design and implement a more realistic system [15 marks].

The current requirement that clients will never fail after taking on a job is overly simplistic.

It is quite possible that clients will take tasks and not complete them (especially in volunteer computing scenario).

Design and implement a more realistic system. Provide evidence that your system meets the new requirement.

Submit:

4. An outline of the control flow in each program and the messages exchanged between the two programs. This should be in English and the level of abstraction should be quite high. For example, client sends a work request for chunksize keys and waits for a response specifying X.

5. A short document explaining how this design satisfies the each of the system requirements.

6. The code for the programs including documentation.

7. A test plan that shows how you test that all of the system requirements have been satisfied and that you have met the individual program requirements.

8. Evidence that you have carried out the tests and an evaluation of whether your programs are correct or not.