csc456 Reliable Systems
Project Requirements

Dr. Steven P. Crain

In this assignment, you will gain experience designing a reliable system.

1 Collaboration

This assignment is to be done in groups of 3–6 students. You are welcome to discuss the project widely, but you may not borrow the architecture and details of another team's design. One student should submit the report, including the names of everyone on the team. Every student should submit a brief statement identifying who you worked with and what your responsibility was in the assignment.

2 Overview

You will design a program that simulates a voting machine. It will read a file containing the votes that users cast, give each voter a receipt to provide accountability and tally the votes for each candidate at the end.

The system you design will be modified to simulate a database with relatively high failure rate. The database will corrupt information at a rate of approximately 1 fault per 1000 bits stored or retrieved.

3 Functional Requirements

1. The system will be provided with an input file containing a sequence of commands to be processed in sequence.
2. The operators must be able to terminate the program prematurely.

3. When the program terminates (either from reaching the end of the input file or from an early termination) the program must create a file containing the number of voters processed and the number of votes received by each each candidate in each position.
4. The program must produce an output log containing the simulated "receipts" that were given to voters.
5. The program must support a configuration command for passing configuration parameters to the database subsystem.
6. The program must support a voter command to start the simulation for a voter.
7. The program must support a vote command to simulate the voter selecting a candidate for a position.
8. The program must support a cast command to cast the votes that the voter specified.
9. The program must generate a unique vote record number to identify the votes cast by a voter.
10. The program must support an inquire command to list the votes associated with a specific vote record number.

4 Non-Functional Requirements

1. The system must have a MTTF<1,000,000 votes when the database system has a failure rate of 1 corruption per 500 bits of data stored or retrieved.
2. If a candidate received k votes for an office but the system reports that the candidate received n votes, this is counted as $|n - k|$ failures.
3. If the system reports votes for an invalid candidate to a real office, this is counted as 1 fault regardless of the number of votes reported.
4. If the system reports votes for an invalid office, this is not considered a fault.
5. If an inquiry reports incorrect votes for a voter, this is a single fault no matter how many errors there are or what kind they are.
6. The program must not keep any state information after processing any cast command.
7. The program must close the database subsystem before terminating.

8. The program must be given the name of the input file when it starts.
9. The voter receipt log should be named `source-votelog.txt`, where `source` is replaced with the main part of the input filename. For example, if the input filename is `smallinput.txt`, the coter receipt log should be named `smallinput-votelog.txt`.
10. The voter receipt log should use the same format as the command log, except that the vote record number should be appended on the voter command.
11. The results file should be named `source-results.txt`, where `source` is replaced with the main part of the input filename.
12. The results file should begin with a tally command containing the total number of voters who cast votes.
13. The results file should contain tally commands for each office and candidate, in any order.
14. If there has been a voter command that has not been followed by a cast command when a new voter command comes in, the first voter has abandoned the booth and the votes must not be counted.
15. Vote record numbers do not necessarily need to be really numbers, but may contain other characters that do not interfere with parsing of the files.
16. The program may support extensions to allow receipts for abandoned voter sessions and separately tallying the votes of abandoned sessions.
17. The program must be able to handle 1,000,000 voters.
18. The program must be able to handle 200 offices, with an average name length of 20 characters. The average number of offices voted per voter is 15.
19. The program must be able to handle 2000 candidates per office, with an average name length of 25 characters.
20. Every voter will vote for each office at most once. The program is free to respond to a violation of this rule in any way.
21. Input will contain only ordinary characters (US-ASCII). The program is free to respond to a violation of this rule in any way.

5 Commands

The following are the details for the commands.

5.1 Configuration command

The configuration command begins with the command `CONF`. The whole line that starts with this command should be passed to the database subsystem without examining the rest of the line.

Example: `CONF cache-false-rate 0.00001`

5.2 Voter command

The voter command contains only the command `VOTER`.

5.3 Vote command

The vote command begins with the command `VOTE`, followed by the office and the selected candidate, all separated by tab characters.

5.4 Cast command

The cast command contains only the command `CAST`. It indicates that the user is ready to cast the votes that were identified. The votes must not be added to the tallies for the candidates until this command is received. When output in the vote receipt log, it is followed by a tab and then the vote record number.

After this command is processed, the program is required to purge all of its variables, except that it can keep the files and database subsystem open. This requirement is to ensure that you do not cheat by circumventing the noisy database subsystem.

Example from the input file:

```
VOTER
VOTE    Govenor  Silly   Hen
VOTER
VOTE    Govenor  Mickey  Mouse
VOTE    Mayor    Ronald  Goose
CAST
```

Example from the vote receipt log:

```
VOTER
VOTE    Govenor  Mickey  Mouse
VOTE    Mayor    Ronald  Goose
CAST  8948298145031
```

5.5 Inquiry command

The inquiry command contains the command `INQ`, followed by a tab and then the vote record number. To process this command, the program should look up the votes cast for the specified voter record number, and print a copy of the voter receipt on the voter receipt log file.

Example from input file:

```
INQ     8948298145031
```

This should result in the following displayed in the vote receipt log:

```
VOTER
VOTE    Govenor  Mickey  Mouse
VOTE    Mayor    Ronald  Goose
CAST  8948298145031
```

6 Database Subsystem Design

You do not need to understand how the database subsystem works to engineer a solution for this problem, but it may help some groups to have the implementation details. This may enable you to make informed choices about the best way to introduce data redundancy into the program.

The database subsystem uses 4 components.

6.1 Noisy Channel

A noisy channel is communication pathway that is subject to faults. Data is put in to the channel, and then comes out the other side. These noisy channels support several different configurable failure modes, each with its own failure rate, expressed in faults per bit of message length.

- Bit flip. When a bit flip occurs, one bit of the data is changed.

- Clone. When a clone occurs, the entire message is replaced with the previous message passed on the channel.
- Scramble. When a scramble occurs, the bits in the message are shuffled into a different order. Exactly one leading 0 bit is included in the scrambling.

6.2 Noisy Cache

A cache allows temporary storage of data that has been recently used so that it can be reused more efficiently. This cache has 2^k slots for holding key-value pairs. Each key is mapped into exactly one slot using the lowest magnitude k bits of the key. The cache supports two operations: store a key-value pair in the cache (replacing any existing key-value pair in the appropriate slot) and retrieving the value associated with a key, which returns wither the value for that key or an indicator that the key is not present.

The noisy cache supports several failure techniques, each with a rate of occurence measured in faults per bit proccessed, where the bits processed include the bits in the key and value combined.

- False hit. The cache returns the value stored in the appropriate slot without checking that the key matches.
- Random hit. The cache returns the value stored in a randomly selected slot.
- Key half-write. The cache stores the new key in the slot, but does not store the new value. Suppose that the slot previously contained ‘cat’:‘Tom’ and we are attempting to store ‘dog’:‘Fido’ in the same slot. After this fault, the slot will contain ‘dog’:‘Tom’ and a subsequent attempt to retrieve the value of ‘dog’ will yield ‘Tom’ instead of ‘Fido.’
- Value half-write. The cache stores the new value in the slot, but does not store the new key. Suppose that the slot previously contained ‘cat’:‘Tom’ and we are attempting to store ‘dog’:‘Fido’ in the same slot. After this fault, the slot will contain ‘cat’:‘Fido’ and a subsequent attempt to retrieve the value of ‘cat’ will yield ‘Fido’ instead of ‘Tom.’

6.3 Database

The database is a well-behaved dictionary. When the program exits, the contents of the dictionary are written to a file named `source-votes.txt`. When the program starts, the contents of this file, if present, are ready into the dictionary. The file consists of one line per database entry, containing the key, then a tab, then the value. As such, tabs are not permitted in keys or values. The entries may appear in this file in any order.

6.4 Broker

The database system is managed by a broker. The broker accepts commands from the program, attempts to satisfy them from cache if possible, and ensures that the database is up-to-date. The broker supports a few failure modes of its own.

- Write failure. The broker sends the data to the cache, but fails to send the data to the database.
- Read failure. The broker returns that there is no data, without checking the cache or database first.
- Replay failure. On reading from the database, the broker returns whatever it last returned instead of checking the cahce or database.

6.5 Architecture

When the program invokes the function to set the value of a key, the following components are active:

1. The input key is passed through a noisy channel to the broker.
2. The input value is passed through a noisy channel to the broker.
3. The key is passed through a noisy channel from the broker to the cache.
4. The value is passed through a noisy channel from the broker to the cache.
5. The cache stores the key and value in the appropriate slot, discarding whatever was already there.
6. The key is passed through a noisy channel from the broker to the database.
7. The value is passed through a noisy channel from the broker to the database.
8. The key, value pair is stored in the database, replacing any key-value pair that had the same key.

When the program invokes the function to append a value to the list associated with a key, the following components are active:

1. The input key is passed through a noisy channel to the broker.
2. The input value is passed through a noisy channel to the broker.
3. The key is passed through a noisy channel from the broker to the cache.
4. The cache looks up the value associated with the key, if any.
5. If the value was found in the cache:
    (a) The value is passed through a noisy channel from the cache to the broker.
    (b) If the cached value is a tuple, the broker appends the input value to it. Otherwise, the broker makes a tuple containing the cached value and the input value.
6. If the value was not found in the cache:
    (a) The key is passed through a noisy channel from the broker to the database.
    (b) The database looks up the value associated with the key, if any.
    (c) If the value was found in the database:
        i. The value is passed through a noisy channel from the database to the broker.
        ii. If the retrieved value is a tuple, the broker appends the input value to it. Otherwise, the broker makes a tuple containing the retireved value and the input value.
    (d) If the value was not found in the database:
        i. The broker creates a tuple containing the input value.
7. The key is passed through a noisy channel from the broker to the cache.
8. The tuple is passed through a noisy channel from the broker to the cache.
9. The cache stores the key and tuple in the appropriate slot, discarding whatever was already there.
10. The key is passed through a noisy channel from the broker to the database.
11. The tuple is passed through a noisy channel from the broker to the database.
12. The key, value pair is stored in the database, replacing any key-value pair that had the same key. If there were no faults, this has the effect of replacing the tuple that was there with the tuple that has had the input value appended.

When the program invokes the function to retrieve the value associated with a key:

1. The input key is passed through a noisy channel to the broker.
2. The key is passed through a noisy channel from the broker to the cache.
3. The cache looks up the value associated with the key, if any.
4. The value, if found, is passed through a noisy channel from the cache to the broker.
5. If the value was not found in the cache:
   (a) The key is passed through a noisy channel from the broker to the database.
   (b) The database looks up the value associated with the key, if any.
   (c) If the value was found in the database:
      i. The value is passed through a noisy channel from the database to the broker.
      ii. The key is passed through a noisy channel from the broker to the cache.
      iii. The value is passed through a noisy channel from the broker to the cache.
      iv. The cache stores the key and value in the appropriate slot, discarding whatever was already there.
6. The found value, if any, is passed through a noisy channel from the broker to the program.

When the program invokes the function to delete a key and its associated value:

1. The input key is passed through a noisy channel to the broker.
2. The key is passed through a noisy channel from the broker to the cache.
3. The cache looks for and removes the key from the cache if found.
4. The key is passed through a noisy channel from the broker to the database.
5. The database looks up the value associated with the key, if any.
6. If the value was found in the database:
   (a) The key and its value are removed from the database.
   (b) The value is passed through a noisy channel from the database to the broker.
   (c) The found value is passed through a noisy channel from the broker to the program.

7 Analysis

Prepare an analysis of the MTTF of your program to demonstrate that the design is expected to achieve the require MTTF. **You do not need to achieve this level of reliability yet. You just need to make a reasonable first attempt at desiging a program that will be reliable, and calculate how well it worked.** You will be tweaking the design if necessary to achieve a higher reliability in the next assignment. To do this analysis, you will need to:

1. Select the types of data redundancy to use.
2. Define the data structures you will use to store votes and tallies in the database, with redundancy.
3. Calculate the average number of bits that you will need to store per vote. Keep in mind the size of the keys and values needed for your data structures. If your redundancy involves keeping multiple copies of individual records in the database, you will want to calculate the number of bits per copy instead of the total number of bits.
4. Convert the failure rate from failures per bit to failures per vote.
5. Calculate the MTTF of your program, keeping in mind the effect of your redundancy.
6. Compare the MTTF that you achieved with the target MTTF of 1,000,000 votes cast. (That is votes, not voters.)

8 Submission

Please submit your answer on Moodle and bring a printed copy to class. You will be reviewing the designs of another group during class.

Your design should include, at a minimum:

1. (20 pts) An executive summary that describes in well-written paragraphs the problem you are solving, a high-level description of your solution and mentions the MTTF that your solution achieves.
2. (20 pts) A diagram showing the overall architecture of your system.
3. (40 pts) Design detail for the data structures you are using. What kinds of objects are you storing in the database? How do you construct the objects using key-value pairs? Be specific on what the keys and values are. Make sure that the data structures include substantial redundnacy and will support the inquiry and reporting requirements.

4. (20 pts) An analysis of the reliability of your system, with explanation and calculations.