**The Concept Crew**

Crew Members: Ryan Norton

Brianna Estrada

Quan Le

Barbara Lyn Crotty

Saurav Gautam

Reliable Systems Project: Course Number CSC 456

Voting System

**Deliverable Date: October 26 , 2016**

# Executive Summaryvoters at polls.JPG

## Introduction

The objective of this assignment is to design and implement a python program that will simulate a voting machine while striving for a high reliability when our system is modified to simulate a database with relatively high failure rate. To avoid redundancies with our program, and also to check for errors, we will be using checksums. A combination of functional and nonfunctional requirements exists which must be met satisfactorily to be considered successful.

## 

## Functional Requirements

In a nutshell, our voting system will cast and tally votes, ultimately providing both a ‘simulated’ voter ballot receipt and contain a program results file. Our voting system must be prepared to work with the provided input file’s sequence commands, and process them in the proper order. During processing, the voting system shall be able to be terminated by the operators at any time. Additionally, our voting system will need to be able to handle that termination gracefully, by either backing out (rolling back) or completing (committing). Finally, a unique vote record identifier is necessary with the ability for our voting system to inquire regarding this number and determine the associated votes. The use of libraries within python are something to explore for the purpose of implementation as a potential validation checks.

## Non-Functional Requirements

The accuracy of vote counting depends upon many factors. Our voting system shall be simple in both design and usability. Redundancy will be utilized to increase reliability against creative failure rates. It will be capable of handling at least one million voters, 200 offices and 2000 candidates. Additionally, US-ASCII characters will be the only accepted characters. Stipulated file names exist and our voting system shall be in alignment with these named conventions. As the primary function of the voting system is to cast votes, once a vote is cast and the cast has been processed, state information will not be retained. And, when the voting system is to be terminated, the database subsystem will be closed prior to termination.

## Commands

There are requirements for numerous commands in order for our voting system to be in communication with the database subsystem. These included: CONF, VOTER, VOTE, CAST, INQ. Our voting system will be in compliance with the with the specified parameters as we do not intend to circumvent the machinations providing the challenge for which this project has been intended.

Receipt log

Results file

Abandoned session handling

## Definitions

Failure - [n - k] failures [if a candidate received *k* votes for an office but the system reports that the candidate received *n* votes]

Fault - (1) invalid candidate voted to real office (regardless of number of votes reported) (1) if an inquiry report incorrect votes for a voter (no matter how many errors there are or what kind they are)

# Functions

1. **Casting/Recording Votes:** The primary function of the machine is to cast the voters votes. There are four typical outcomes for a cast vote: a vote, overvote, undervote, and unintended choice. A vote is one which an individual cast a vote for a candidate for an office and it is recorded as such. An overvote is one in which the same individual casts a vote for the same office, twice, such as voting two candidates for President. An over vote is an error which needs handling. An undervote is one in which a voter casts no vote where one is permitted, thus fewer votes than permitted are cast. Undervotes are permitted; however, they may errors. An undervote could be a vote that was mean but did not cast. Unintended choices are errors by accident, as our voters are from an input file, any unintended choices would be errors.

# 

# Diagram

# 

# 

# Design Details

## Database objects being stored

For our design we are assuming per entry there will be 8 bits per character. The users voter id will be a 5 character string consisting of letters and numbers. The positions will be associated with a number rather than a word so the bit count isn't as high. The candidate will also be associated with a number to reduce the bit count and increase reliability

## 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 redundancy and will support the inquiry and reporting requirements.

Input Algorithm:

Start command

Input –Voter Id: Type US-ASCII (~20bits)

#Show user the result on the left side of =

Vote =>Cand-Position-1- (1 or 2) = Trump, President or Bernie, President Cand-Position-2- (1)

Vote =>Cand-Position-2 (1)= Hillary, Vice President

Vote =>Cand-Position-3(1,2 or 3)= Ronaldo, Messi or Nani

Status: ABCDF voting – Done => Save vote Into library and print recipe

ABCDF voting- Not done => Return to the Picking session

Voting System Objects

Office object: How many position we have in the poll

Structure:

Office – End = N #N-1 is total of position

Office – (N-1)= Position-1

Office – (N-2) = Position-2

Office – (N-3) = Position-3

#and go on depend on N

Cand object: It’s about the candidate and the position they are running

Structure:

Cand- Position-1 – End = 3 #this mean there are only 2 candidates running for this position

Cand – Position-1 - 1 = Trump – President #Value of this key is Trump as President

Cand – Position-1 - 2= Bernie – President

Cand-Position-2-End =2 #this mean there are only 1 candidate running for this position

Cand – Position-2 - 1= Hillary – Vice President

Cand-Position-3-End = 4 #this mean there are 3 candidates running for this position

Cand – Position-3 – 1= Ronaldo - Secretary

Cand – Position-3 – 2= Messi- Secretary

Cand – Position-3 – 3= Nani- Secretary

Receipt Algorithm:

From noisy cache import key-value:

Print “Voter Id”

Show- Position-1-1 =Trump as President #Show user Trump as President

Show- Position-2-1= Hillary as Vice President #Show user Hillary as Vice President

Show-Position-3-3 = Nani as Secretary #Show user Nani as Secretary

\*Subsystem Database\*

\*\*Noisy Channel Algorithm:

//Receiving Cand-Position from input file, broker//

For (+1 input):

Clone (Vote -VoterID-n : Cand-position) x 10 #Duplicate the candidate-position key and its value to increase reliability

Send Cand-Position to Cache/Database

\*\*Broker Algorithm:

Send (Vote -VoterID-n : Cand-position) to Cache/Database through Noisy Channel

Return (Vote -VoterID-n : Cand-position) from Cache/Database through Noisy Channel

\*\*Noisy Cache Algorithm:

For(+1 input):

Update (Vote -VoterID-n : Cand-position) (replace same key-value)

\*\*Database Algorithm

Replace same (Vote -VoterID-n : Cand-position)  
Create Source-vote.txt

Ex: Cand-Position-1-1-340 = Trump- President 340 votes

Update: Cand-position-1-1-340+1= Trump-President 341 votes

# Reliability Analysis

We are assuming there will be 8 bits per character. Each vote that goes into our database will look like this “1-12ks3-n (Vote -VoterID-n)”, which consists of 9 characters. This vote statement is saying the candidate for position 1 associated with value 1 (say Trump) will be getting one more. This gives us a total of 72 bits per vote. Since one bit per 1000 bits will be corrupted we are expecting a MTTF of 13 votes. This also gives us a failure rate of 0.08. The failure rate of the parallel is .08/(1+½+⅓+¼+...1/10)=.03 Using this we can now calculate the reliability of the vote: Rvote = e(-0.03\*1000000) which gives us a reliability of 0.