Voting

Core Idea

The core idea is to distribute the voting mechanism in phases with each phase being implemented using a smart contract. We define a few states in the voting to ensure fairness in the process: OPEN, CLOSED, FINISHED. It prevents the owner and users from executing functions that cannot be executed in that particular state.

Ballot phase

We deal with the voting situation with two candidates only. For this, the user can enter a Boolean value which is True and False. This will classify whether the vote will go the person1 or person2. Voting is a basic right for all individuals so no fees are required for the user to vote.

Every individual can vote only once, so the first vote of the user will be considered with no option to revote or change voting preferences. The blockchain stores the smart contract takes the user vote and registers it.

The vote count given to each person among the 2 contenders is also counted simultaneously with the vote registering.

The vote chosen by a voter is kept secret and can only be accessed by that voter as the hash of the smart contract is visible to others and not the actual link of the vote with the voter. The current vote count is also not displayed to a generic voter. The voter can only put his vote and wait for the results to come.

In no way can he make a decision based on the current vote count and influence the election. The only option for the voter is to put the vote and wait for the results to be announced. This will always result in a fair and impartial election.

In this phase the voting is OPEN.

Winner/Tie announcement

In this phase the voting is CLOSED.

The blockchain has stored all the votes entered by millions of users who have registered their votes. The counts have also been stored.

The winner of the voting is the person that has the maximum number of votes. It is computed as a separate function and announced.

It is also possible that the vote counts are the same for both parties. This is the case of a tie situation. In such a case, a random function is used to choose any winner randomly among both parties. This random function will choose among both the address of the contenders (addresses will always be different) and will classify a winner of the election.

Since it is a random function, both the parties have an equal probability of 0.5 to be chosen. This ensures that fairness is not lost during the computation of the winner.

It terminates with the FINISHED state as all voting computations are completed.

We also implement a change state mechanism to prevent changes that are prohibited which ensures that no action is taken to spoil the voting system. The only valid state change is from OPEN to CLOSED to FINISHED. All other state changes are invalid and need to be disregarded.

Smart Contract Implementation

```
// Create a smart contract
contract Voting{

    // Store the addresses of contenders
    address address0 = ADDRESS_0;
    address address1 = ADDRESS_1;

    // Store the initial vote count of contenders
    uint256 person0 = 0;
    uint256 person1 = 0;

    // The total number of voters
    uint256 NO_OF_VOTERS = MAX_VOTERS;

    // Stores whether the voter has voted or not
    bool[] voted;

    // Mark that all the voters have not voted
    for (uint256 i = 0; i < NO_OF_VOTERS; i++){

        // Not voted
        voted.push(false);
    }

    // OPEN = 1, CLOSED = 2, FINISHED = 3
    // Only valid order is 1 -> 2 -> 3
    // Initially Voting is OPEN
    uint Votingstate = 1;

    // Function for voter to put his ballet
    function ballet(bool val, address voter){
        // Requirements

        // Voting must be OPEN to vote
        require Votingstate == 1;
        // Voter must have not have voted before (no revoting)
        // Case of the first and only vote
        require voted[voter] == false;

        // Vote is given to person1
        if (val){

            // Vote count of person1 is incremented
            person1++;
        }

        // Vote is given to person0
        else{

            // Vote count of person0 is incremented
            person0++;
        }

        // The voter has completed voting, cannot be allowed to revote
        voted[voter] = true;
    }

    // Announce the winner
    function winner(){

        // As the winner is being computed, we cannot consider more votes
        // Voting is CLOSED
        changestate(2);

        // Voting must be CLOSED to evaluate the winner
        require Votingstate == 2;

        // Person0 has more votes than Person1, he is declared the winner
        if (person0 > person1){

            // The winner is displayed
            return address0;
        }

        // Person1 has more votes than Person0, he is declared the winner
        if (person1 > person0){

            // The winner is displayed
            return address1;
        }

        // Since both the contenders have equal votes, the situation of tie arises
        if (person0 == person1){

            // Use a random function to select the address of the winning person among the 2
            // Random function gives an equal probability of 0.5 for both the contenders
            // Ensures a winner is declared in the tie situation
            return generaterandomamong(address0, address1);
        }

        // As the winner has been computed,
        // Voting is FINISHED
        changestate(3);
    }

    // Valid change of the Voting state
    function changestate(uint state){

        // Either we can move 1 -> 2 or 2 -> 3
        require (Votingstate == 1 && state == 2) ||
                (Votingstate == 2 && state == 3);

        // Change the current Voting state
        Votingstate = state;
    }
}
```