Practical session for blockchain

(4) Auction

Giyeong Lee[1] Jungyoon Song[1]

[1]Financial Risk Engineering Lab.
Seoul National University

April 27, 2022

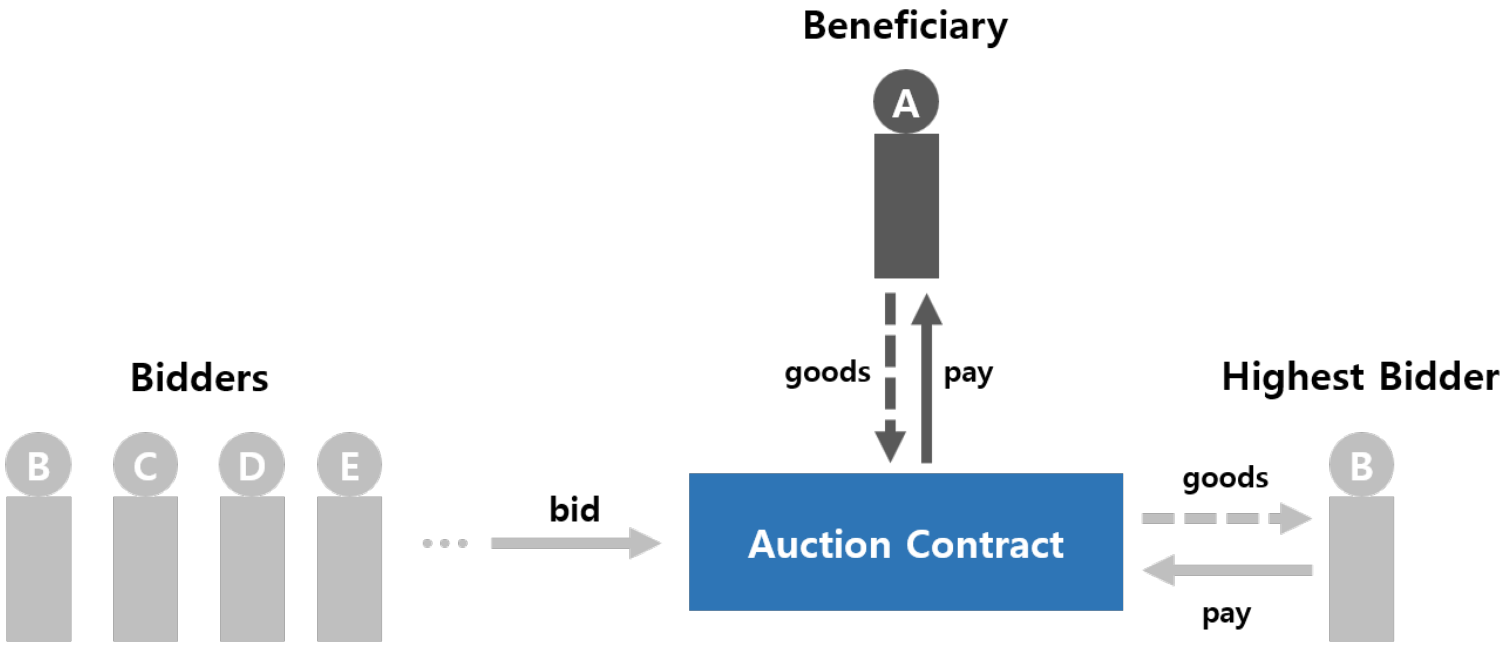
Auction

- **Auction** or **public sale** is a publicly held sale at which goods or services are sold to the (usually) highest bidder.
- Blockchain can be applied to prevent tamper and manipulation of bids in untact auctions.
- In this lecture, we implement the following types of auctions.
  1. English auction: the most common type of auction.
  2. Blind auction: no bidder knows the bid of any other participant.

Simple Auction

- Bidders bid and deposit the bid price.
- The highest bidder pays and takes goods.
- The beneficiary is paid the highest bid.
- The delivery of goods will be omitted.

Simple Auction(Cont.)



State Variables for Auction

- Addresses of beneficiary and bidders.
- The address of highest bidder and the highest bid price.
- Information of bidders' deposit.
- When to be closed and whether the auction is closed or not.

Implementation: Simple Auction

[Ex. 1] Simple Auction

```
pragma solidity >0.7.0 <0.8.0;

contract SimpleAuction{
    address payable public beneficiary;
        uint public auctionEndTime;

        address public highestBidder;
        uint public highestBid;

        mapping(address => uint) pendingReturns;

        bool isAuctionEnded;

        event BidUpdated(address bidder, uint amount);
        event AuctionEnded(address winner, uint amount);

        constructor(
        uint _biddingTime,    // in seconds
        address payable _beneficiary
    ) {
        beneficiary = _beneficiary;
        auctionEndTime = block.timestamp + _biddingTime;
        }
```

Implementation: Simple Auction (Cont.)

[Ex. 1] Simple Auction

```
function bid() public payable {
    require(block.timestamp <= auctionEndTime, "The auction is closed.");
    require(msg.value > highestBid, "The current bid is higher than your bid.");

    if (highestBid != 0){
        pendingReturns[highestBidder] += highestBid;
    }
    highestBidder = msg.sender;
    highestBid = msg.value;

    emit BidUpdated(msg.sender, msg.value);
}

function withdraw() public returns (bool) {
    require(isAuctionEnded, "You can withdraw the deposit after the auction ends.");
    uint amount = pendingReturns[msg.sender];
    if (amount > 0) {
        pendingReturns[msg.sender] = 0;
        if (!msg.sender.send(amount)) {
            pendingReturns[msg.sender] = amount;
            return false;
        }
    }

    return true;
}
```

Implementation: Simple Auction (Cont.)

[Ex. 1] Simple Auction

```
    function auctionEnd() public {
        require(block.timestamp >= auctionEndTime, "The auction is not ended yet.");
        require(!isAuctionEnded, "'auctionEnd' has already been called.");

        beneficiary.transfer(highestBid);
        emit AuctionEnded(highestBidder, highestBid);
        isAuctionEnded = true;
    }
}
```

Test Auction Contract

- Create and deploy *SimpleAuction* contract.
- Try *bid* function with the various addresses.
- After *auctionEndTime*, closed the auction and withdraw the deposits.
- (Optional) Try *withdraw* and *auctionEnd* functions before the auction is closed.

Blind Auction

- In a blind auction, no bidders knows the bid price of any other participant.
- But we leave that information open to make it easier to see how it works.
  - Which part of the code should be modified to make that information sealed?
- Additionally, we set the *minimum bid price*.

Implementation: Blind Auction

[Ex. 2] Blind Auction

```
pragma solidity >0.7.0 <0.8.0;

contract BlindAuction {
    struct Bid {
        uint bidPrice;
        uint deposit;
    }

    address payable public beneficiary;
    uint public biddingEnd;
    uint public revealEnd;
    bool public isAuctionEnded;

    address[] private bidders;

    mapping(address => Bid) public bids;
    address public highestBidder;
    uint public highestBid;
    uint public minimumBid;

    event AuctionEnded(address winner, uint highestBid);

    modifier onlyBefore(uint time) {
        require(block.timestamp <= time, "You called the method too late.");
        _;
    }
    modifier onlyAfter(uint time) {
        require(block.timestamp >= time, "You called the method too early.");
        _;
    }
```

Implementation: Blind Auction

[Ex. 2] Blind Auction

```
constructor(uint _biddingTime, uint _revealTime, uint _minimumBidInEther,
            address payable _beneficiary) {
    beneficiary = _beneficiary;
    minimumBid = _minimumBidInEther * 1 ether;
    biddingEnd = block.timestamp + _biddingTime;
    revealEnd = biddingEnd + _revealTime;
}

function bid(uint _bidPriceInEther) external payable onlyBefore(biddingEnd) {
    uint bidPrice = _bidPriceInEther * 1 ether;
    if (bids[msg.sender].deposit == 0) {
        require(msg.value >= bidPrice, "You need to deposit at least the bid price.");
        bids[msg.sender] = Bid({bidPrice: bidPrice,
                                deposit: msg.value
                                });
        bidders.push(msg.sender);
    } else {
        require(bids[msg.sender].deposit + msg.value >= bidPrice,
                "Your total deposit should be at least the bid price.");
        require(bids[msg.sender].bidPrice < bidPrice,
                "You have already bid on a higher price.");
        bids[msg.sender].bidPrice = bidPrice;
        bids[msg.sender].deposit += msg.value;
    }
}
```

Implementation: Blind Auction

[Ex. 2] Blind Auction

```
function reveal() external onlyAfter(biddingEnd) onlyBefore(revealEnd) {
    uint length = bidders.length;
    uint refund;
    for (uint i = 0; i < length; i++) {
        Bid storage bidToCheck = bids[bidders[i]];
        uint bidPriceToCheck = bidToCheck.bidPrice;

        if (bidPriceToCheck > highestBid && bidPriceToCheck >= minimumBid){
            // Refund the deposit of the previous highest bidder.
            if (highestBidder != address(0)) {
                refund = bids[highestBidder].deposit;
                bids[highestBidder].bidPrice = 0;
                bids[highestBidder].deposit = 0;
                payable(highestBidder).transfer(refund);
            }

            // Updated new highest bidder and bid price.
            highestBidder = bidders[i];
            highestBid = bidPriceToCheck;
        } else {
            refund = bids[bidders[i]].deposit;
            payable(bidders[i]).transfer(refund);
        }
    }
}
```

Implementation: Blind Auction

[Ex. 2] Blind Auction

```
    function withdraw() external onlyBefore(biddingEnd) {
        uint amount = bids[msg.sender].deposit;
        if (amount > 0) {
            bids[msg.sender].bidPrice = 0;
            bids[msg.sender].deposit = 0;

            payable(msg.sender).transfer(amount);
        }
    }

    function auctionEnd() external onlyAfter(revealEnd) {
        require(!isAuctionEnded, "'auctionEnd' has already been called.");
        emit AuctionEnded(highestBidder, highestBid);

        beneficiary.transfer(highestBid);
        payable(highestBidder).transfer(bids[highestBidder].deposit - highestBid);
        isAuctionEnded = true;
    }
}
```

References

- Solidity official documentation - Solidity by Example, https://docs.soliditylang.org/en/v0.8.13/solidity-by-example.html