Distributed Auction Bidding System

By: Albert Lluberes &
Adonis Payton

Table of contents

Overview of Auction System

Technology used

Architecture

Internode communication

Multithreading/Multiprocessing

Interprocess Communication

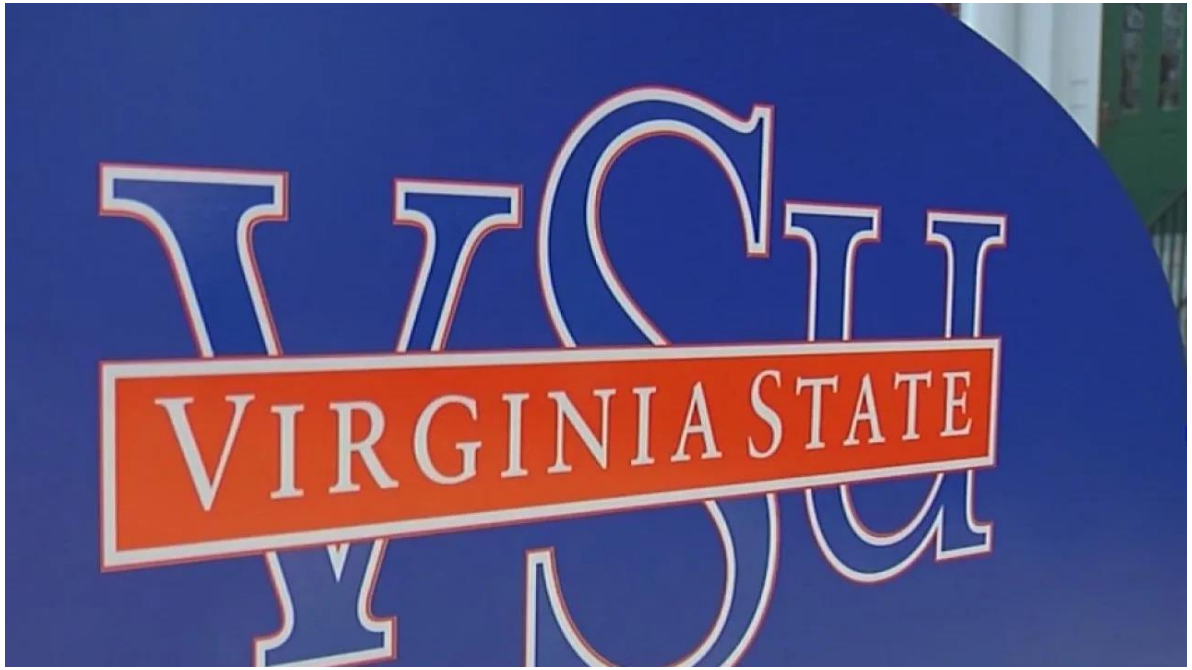
Distributed Computing

Demo and Results

Overview of App

VSU Online Auction

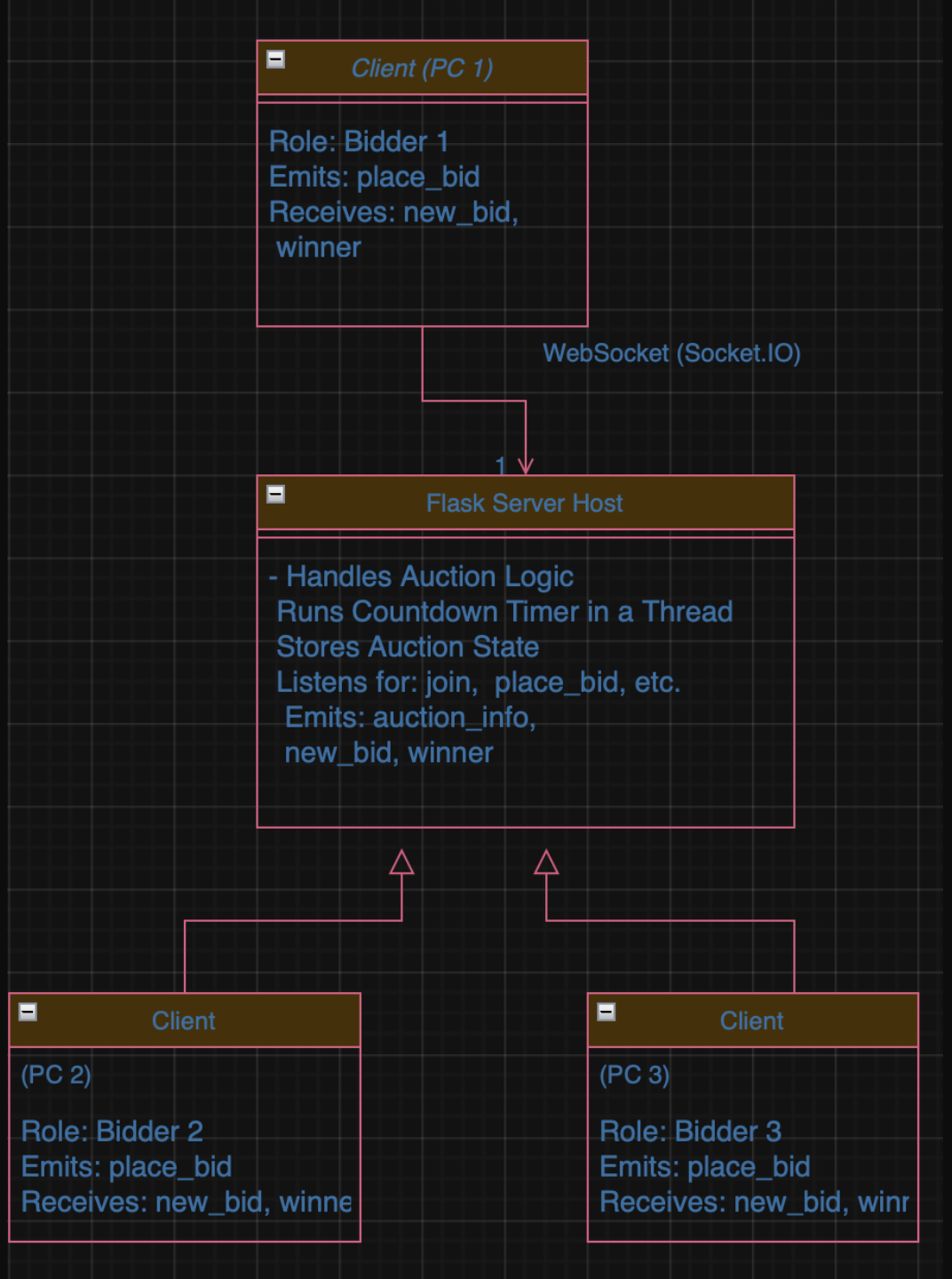
A platform where Virginia State University students can bid and win exclusive VSU collectables. This project brings students together, increasing university engagement by at least 50%.



Technologies Used

- **Python** (Flask Framework)
- **Flask-SocketIO** for real-time communication
- **JavaScript, HTML, CSS** for frontend
- **Multiprocessing, Multithreading** for server operations
- **WebSockets (Socket.IO)** for internode communication

Architecture Diagram



Internode Communication

- Nodes are the Clients (PCs) and the Server. (Individual nodes)
- Communication between nodes is handled via Socket.IO over WebSocket.
- Real-time messages (e.g., join, place bid, auction start, auction end) are passed across nodes.

Example Messages Passed:

- join — A user joins the auction.
- place_bid — A user places a new bid.
- auction_info — The server sends auction details to all users.
- For example, when a user joins or places a bid, that data is immediately broadcasted to all connected clients.

Multithreading

When the auction is started by the host, a new thread is created. The countdown timer for the auction runs in a separate thread. This allows users to continue placing bids without blocking the server

Multithreading allows the auction timer to run in the background while multiple users place bids simultaneously. This showcases concurrency without blocking the main server logic because the thread runs in the background without blocking user interaction.

```
78  @socketio.on('start_auction')
79  def start_auction():
80      global auction_started, bids, winner_announced, auction_ended
81      if request.sid != host_sid:
82          return
83      auction_started = True
84      auction_ended = False
85      winner_announced = False
86      bids = []
87      socketio.emit('auction_started')
88      thread = threading.Thread(target=countdown_and_announce)
89      thread.start()
```

Multi-Processing

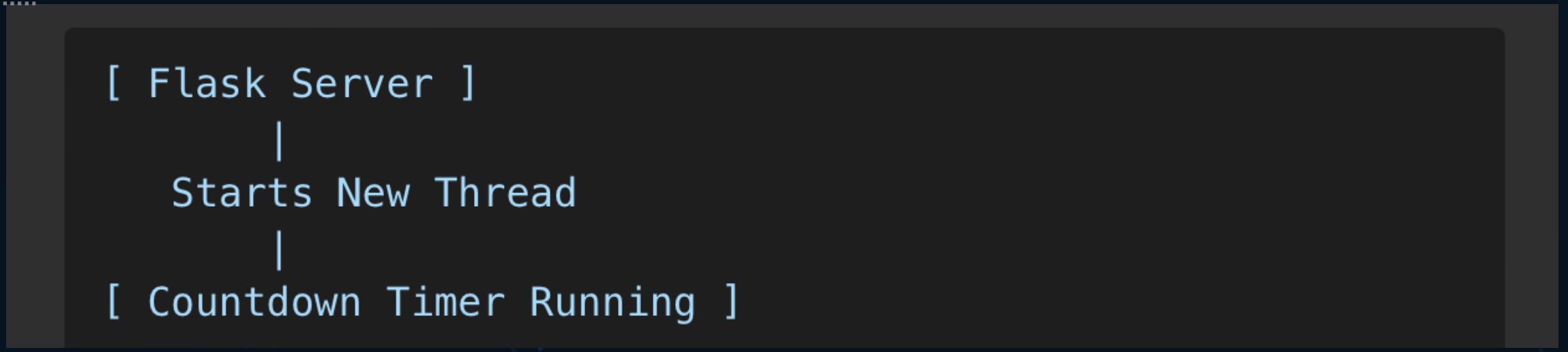
```
4   from flask import Flask, render_template, request
5   from flask_socketio import SocketIO, emit
6   import threading
7   import multiprocessing
```

```
129   def handle_auction_end():
130       p = multiprocessing.Process(target=announce_winner)
131       p.start()
132
```

```
113   @socketio.on('end_auction')
114   def end_auction():
115       global auction_ended
116       if request.sid == host_sid and not auction_ended:
117           handle_auction_end()
```

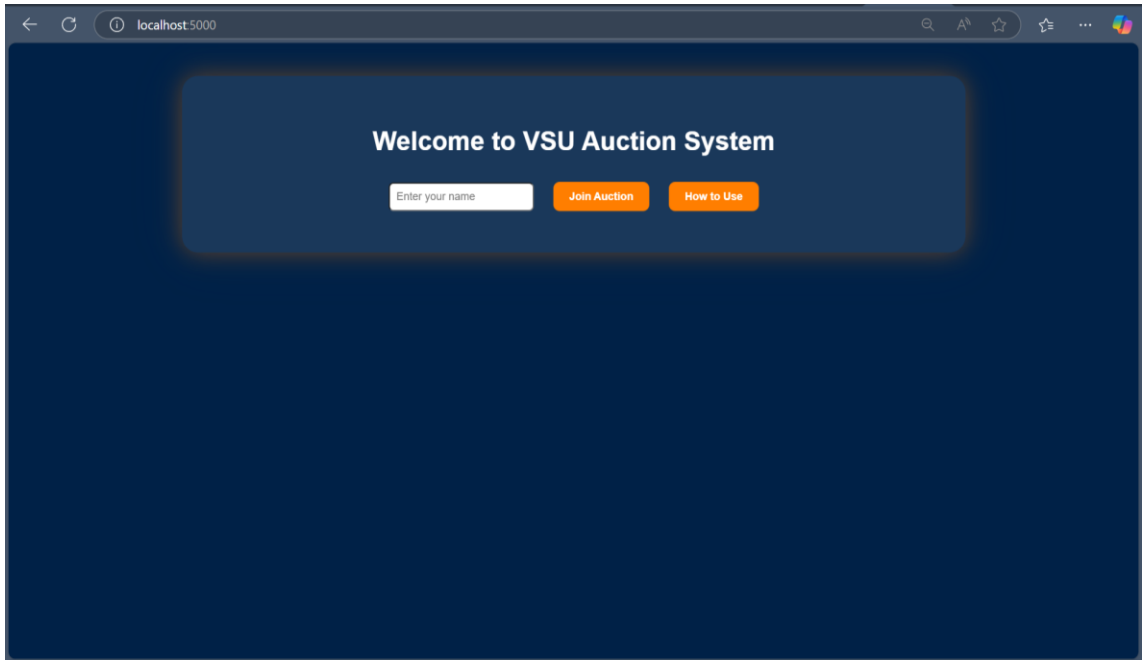
How Distributed Computing is Used

- Clients (bidders) are running on different machines over the network.
- All clients communicate with a central Flask server using Socket.IO (WebSocket technology).
- Real-time communication allows distributed users to bid, receive updates, and see results.

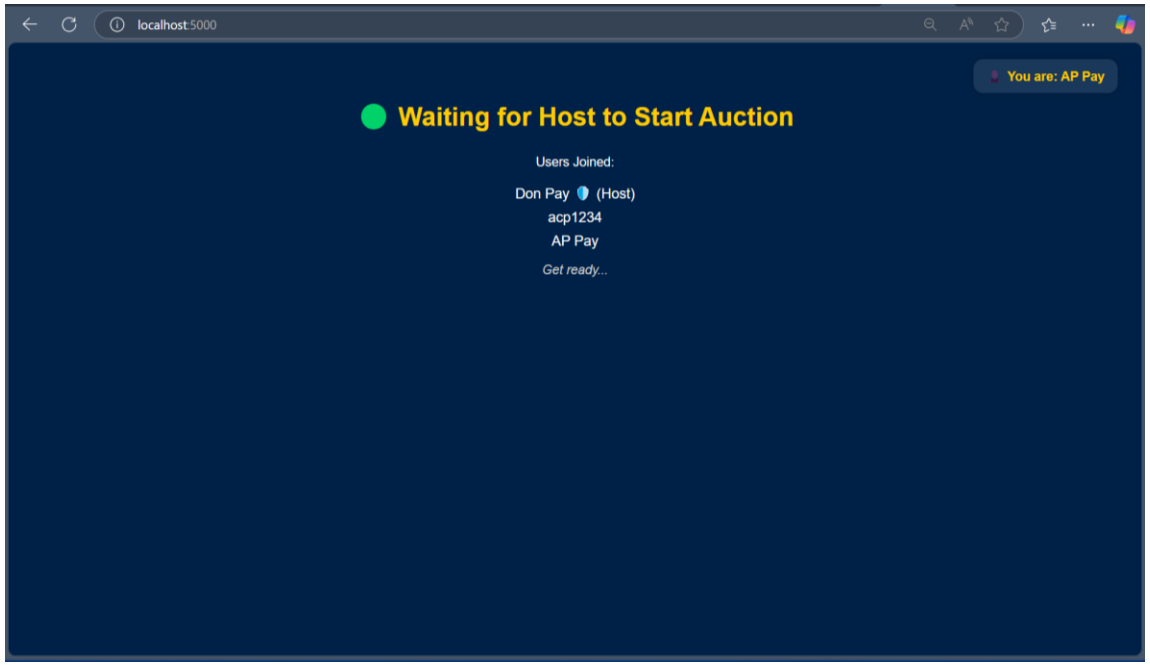


Interprocess Communication(ICP)

Interprocess communication occurs when the countdown timer, running on a separate thread, signals the main server process to finalize the auction and notify all bidders with the results


localhost:5000
Welcome to VSU Auction System
Enter your name
Join Auction
How to Use


localhost:5000
You are: Don Pay
Item: VSU Shirt
Min Price: $10
Type: First-Price
Users: Don Pay
Time Remaining: --
Start Auction
End Auction
You are the host.


localhost:5000
You are: AP Pay
Waiting for Host to Start Auction
Users Joined:
Don Pay (Host)
acp1234
AP Pay
Get ready...

Demo