CS7610: Lab 1 report

Madhukara S Holla (sholla.m@northeastern.edu)

The implemented program facilitates coordination among multiple processes using UDP messages. Each process communicates its readiness to start a task by sending and receiving heartbeat messages. The program reads a configuration file to identify the peers it needs to communicate with and ensures that it receives acknowledgment from all peers before proceeding.

Initialization

1. The program checks for command line arguments to obtain the path to the hosts file containing peer information.
2. It then reads the configuration file and dynamically allocates memory to store the peer information, including hostnames, IP addresses, and received message counts.
3. The program obtains its hostname by using **gethostname()** system call - this is used to exclude the program from sending messages to itself.
4. A UDP listener socket is created to receive messages from peers, and the program binds the socket to port 8080.

Loop

After initialization, the program enters a loop where it sends heartbeat messages to all peers and performs peer discovery at regular intervals (1 second)

Sending Heartbeats

1. The program attempts to resolve each hostname into an IP address and sends a simple string ("HEARTBEAT") via the UDP socket.
2. Upon successfully sending a message, the program stores the IP address-to-hostname mapping.

Peer discovery

1. After sending a round of heartbeat messages, the program listens for incoming messages from peers.
2. When a message is received:
    a. It checks if the message is a heartbeat.
    b. If it is, the program checks the sender's IP address and updates the corresponding peer's status.
    c. The program tracks unique heartbeats from each peer, ensuring that it only acknowledges the first message from each peer.

Ready state

Once the program has received heartbeat messages from all peers (n-1), it prints "READY" to stderr and continues to send heartbeat messages indefinitely until terminated.

Cleanup

1. The program runs indefinitely until stopped by the user (e.g., by killing the container).
2. Memory and socket cleanup code is included for future use when a proper termination logic is implemented (ex: with signal handlers).

Key Considerations

1. **Non-blocking Reception:** The program uses non-blocking socket reception to avoid missing messages if peers send them before the program is ready to receive..
2. **Dynamic Memory Management:** The program dynamically allocates and reallocates memory for peer information, ensuring efficient use of resources..
3. **Error Handling:** The program includes error handling for file operations, memory allocation, and socket operations, logging errors to aid debugging.