**Descrizione del Progetto**

Il progetto consiste in un’applicazione Client/Server scritta in linguaggio C e con protocollo TCP, che simula un sistema di difesa di uno spazio aereo. Questo sistema prevede *n* torrette, un bersaglio ed una “stazione di controllo” per le torrette.

La Stazione di Controllo, una volta stabilita la porta da utilizzare, è rappresentata dal Server di Rendezvous che resta in ascolto sulla porta selezionata.  
Si occupa principalmente di mettere in comunicazione contemporaneamente una o più torrette.

Il Server-Torretta possiede un ID, un indirizzo IP, la porta del Server di Rendezvous, una propria porta, il range entro cui può colpire il bersaglio, il numero di missili a sua disposizione e la propria posizione sulla “scacchiera” (espressa tramite coordinate).

Il Client che si connette ad uno dei Server connessi rappresenta il bersaglio;   
possiede a sua volta un ID, IP e porta del server a cui si connette e le coordinate in cui appare (X, Y).

All’avvio dell’applicativo il bersaglio si connette ad una delle torrette attive inviando le sue informazioni, dopodichè la torretta comunica tramite il Rendezvous i dati ricevuti alle altre torrette. Ogni torretta controlla, tramite un apposito algoritmo, se il bersaglio rientra nel suo raggio e se ha un numero di missili maggiore di zero; se le condizioni sono soddisfatte viene eseguito l’algoritmo che calcola l’esito del bersaglio distinguendo 3 casi possibili:

* Bersaglio fuori raggio
  + Nessuna delle torrette ha raggio sufficiente per abbattere il bersaglio, quindi il bersaglio sopravvive.
* Numero missili torrette insufficienti
  + Nessuna delle torrette in raggio ha missili disponibili, quindi il bersaglio sopravvive.
* Bersaglio distrutto.

Inoltre, quando un bersaglio rientra nel raggio di più torrette connesse in rete, il Server con il maggiore numero di missili abbatte il Client e tutte le torrette attive visualizzano l’esito del bersaglio.

Il Server di Rendezvous fornisce il servizio di connessione tra le diverse torrette. Quando riceve una connessione, invia alla torretta attiva in rete IP e porta del nuovo Server.

Quando una torretta (Server) identifica un bersaglio (Client), invia ID e posizione del bersaglio a tutti gli altri Server connessi.

Il Client si collega al Server inviando ID, Indirizzo IP, la porta del Server e la sua posizione, ricevendo poi l’esito del bersaglio, cioè se è stato abbattuto o meno.

**Parametri generali - Tabelle riassuntive**

Il Server di Rendezvous:

Numero porta Il numero della porta da assegnare al Server di Rendezvous

Il Server:

Il Client:

|  |  |
| --- | --- |
| Parametri | Descrizione |
| ID | L’ID assegnato al Client - Bersaglio |
| IP Server | L’IP del Server - Torretta a cui connettersi |
| Porta Server | La Porta assegnata al Server - Torretta |
| Posizione (X, Y) | La posizione in cui appare il bersaglio sulla “scacchiera” |

**Esempio di utilizzo**

Una volta che il Rendezvous è attivo ed in ascolto, connettendo in rete uno o più Server, questi ultimi invieranno i loro ID e IP al Rendezvous e verranno inseriti in una lista visualizzata da quest’ultimo.

L’ultimo Server connesso, invece invierà le sue informazioni alle altre torrette attive che ne visualizzeranno la connessione.

Il Client, dopo essersi collegato ad un Server, specificando il suo ID, IP, Porta Server e posizione, attende l’esito da una delle torrette con tre possibili casi:

1. Non raggiungibile da nessuna Torretta;
2. Missili della Torretta <ID> insufficienti (<ID>: identificativo del Server);
3. Colpito dalla Torretta <ID>.

Inoltre, il Server a cui si è connesso il bersaglio visualizza a video la posizione del Client, il Server che ha abbattuto il bersaglio ed il numero dei missili rimasti a disposizione.

Invece, in caso di disconnessione di una delle torrette, verranno notificati sia gli altri Server rimasti in rete sia il Rendezvous e visualizzeranno entrambi l’IP del Server disconnesso.

**Protocollo di comunicazione**

Il **Server di Rendezvous**, stazione principale per la gestione della rete, crea un Socket per la connessione tra se stesso ed i Server che si connetteranno successivamente in rete, associando al Socket un numero di porta inserito da riga di comando.

Il **Server** o torretta, gestisce tre tipi di connessioni: una con il Rendezvous, una con un Server ed una con un Client.

Server – Rendezvous: gestita tramite un unico Socket.

Server – Server: gestita tramite un Socket dedicato per la gestione del bersaglio in caso di avvistamento ed un altro per la gestione server, avendo così una connessione Peer-to-Peer nella quale un Server ha il ruolo sia di Server che di Client.

Client – Server: gestione tramite un Socket per la comunicazione con il bersaglio.

Il **Client** o bersaglio, crea un unico Socket e si collega alla torretta che vuole attaccare.

**Comunicazione Rendezvous – Server**

Una volta connesso, il Rendezvous si mette in ascolto, controlla di avere spazio sufficiente a memorizzare l’ultimo Server connesso, controlla se l’ID assegnatogli è già presente nella lista (nel caso rifiutandone la connessione), aggiunge il nuovo Server alla fine della propria lista e gli comunica la lista delle torrette precedentemente connesse.

Infine invia alle vecchie torrette le informazioni dell’ultimo Server collegato.

**Comunicazione Client – Server**

Il Client, crea un Socket di comunicazione, si collega al Server (connect) ed invia le sue informazioni alla torretta, il quale, tramite un algoritmo basato sulla distanza geometrica delle coordinate sulla scacchiera, verifica l’esito del bersaglio. I possibili casi sono:

* Se l’esito ritorna -1, il bersaglio non è raggiungibile da nessuna torretta attiva;
* Se ritorna 0, il bersaglio è fuori raggio;
* Altrimenti il bersaglio viene abbattuto da una delle torrette connesse.

**Comunicazione Server – Server**

Il Server crea un Socket, controlla se l’ID è univoco e maggiore di zero, dopodichè effettua la connessione al Rendezvous.

Una volta che il Server è connesso al Rendezvous, crea altri due Socket: uno per la connessione agli altri Server e l’altro per la connessione al Client.

Tramite la lista memorizzata nel Rendezvous, inviata su “sock\_rendezvous”, la torretta crea per ogni membro della lista un thread associato alla funzione “gestisci\_server”; la procedura invia le informazioni sul bersaglio alle altre torrette, poi attende la loro risposta.

In caso di risposta affermativa (numero missili e raggio sufficienti) aggiunge la torretta alla lista dei candidati per la distruzione del bersaglio. Contemporaneamente monitora la connessione/disconnessione dei Server attivi in modo da gestire la disconnessione improvvisa di uno dei server, per esempio dovuta all’invio di un SIG\_INT forzato da tastiera.

Il Socket connesso al Server di Rendezvous è associato al thread che gestisce la procedura “lettura\_rend”; la procedura controlla se c’è spazio sufficiente per il nuovo Server nella lista dopodichè genera un thread che si occupa della connessione al nuovo Server arrivato (con il metodo gestisci\_server sopra citato).

Quando una torretta invia un segnale ad un altro Server, questo verifica se il motivo del suo segnale sia l’avvistamento di un bersaglio.

La verifica viene eseguita dalla funzione “bersaglio\_avvistato” associata al thread.

Una volta avviata la funzione, il Server ha un time lapse di circa 2 secondi per prendere il lock del mutex (System Call *pthread\_mutex\_timedlock*), se ci riesce allora invia l’esito del bersaglio dal suo punto di vista alla torretta iniziale (rappresentato da un intero), altrimenti Invia “-2” per indicare “server occupato”.

Questa gestione serve ad evitare il blocco di più Torrette che avvistano contemporaneamente diversi bersagli.

Se il Server in comunicazione è attivo, viene calcolata la distanza geometrica e poi calcolato l’esito del bersaglio, ritornando -1 nel caso in cui il bersaglio non viene abbattuto; altrimenti se l’esito è minore di zero, risulterà fuori raggio, se uguale a zero i missili non sono sufficienti per abbattere il bersaglio e se maggiore di zero allora il bersaglio è stato abbattuto.

**Dettagli implementativi**

**Librerie System Call – Tabelle riassuntive ( <sys/socket.h> )**

socket() Crea un Socket di comunicazione (permette di specificare il tipo)

bind() Associa il Socket alla struttura Sockaddr\_in

listen() Il Socket viene settato in ascolto

accept() Accetta la connessione da chi ne fa richiesta

connect() Si connette ad un Socket in ascolto

getsockname() Restituisce la struttura Sockaddr\_in relativa al Socket corrente

**Librerie System Call – Tabelle riassuntive ( <pthread.h> )**

pthread\_create() Genera un thread

pthread\_mutex\_lock() Prende il lock sul mutex

pthread\_mutex\_unlock() Rilascia il mutex

pthread\_cond\_signal() Risveglia un singolo thread in attesa su una “condition variable”

pthread\_cond\_wait() Attende che qualcuno rilasci una condition variable

pthread\_join() Attende la terminazione di un altro thread

pthread\_self() Restituisce il tid del thread corrente

pthread\_equal() Confronta due tid

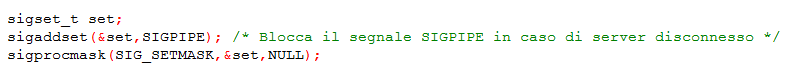
pthread\_cancel() Elimina un thread

pthread\_exit() Termina il thread corrente

Tratteremo in questa sezione nello specifico alcune scelte implementative di cui non abbiamo parlato finora.

Per quanto riguarda il Server di Rendezvous le principali System Call utilizzate sono:

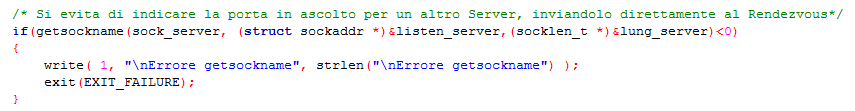
* Funzioni di gestione thread
  + ESEMPIO: pthread\_cancel, pthread\_self, pthread\_equals
* System Call per la gestione della maschera dei segnali bloccati:



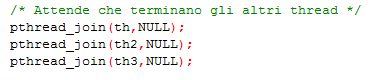
È stato necessario aggiungere **SIGPIPE** (segnale associato alle comunicazioni PIPE/Socket) ai segnali bloccati in quanto era il segnale inviato di default da un Server appena disconnesso.

Per quanto riguarda il Server le principali System Call utilizzate sono:

* “getsockname”: avendo scelto di utilizzare due porte differenti di comunicazione (Client/Server), quella su cui comunica il Server viene decisa dal Sistema Operativo al momento dell’esecuzione della System Call “listen” su un indirizzo di porta non specificato. Di conseguenza risulta necessario recuperare la porta associata dal Sistema Operativo utilizzando appunto *getsockname:*



* Il funzionamento di tutti i thread in contemporanea è garantito da una serie di System Call relative ai “semafori”:
  + Le variabili di tipo **pthread\_mutex** sono utilizzate per garantire il monopolio temporaneo della cosiddetta “sezione critica” (compresa tra ***pthread\_mutex\_lock*** e ***pthread\_mutex\_unlock***);
  + Le variabili di tipo **pthread\_cond** sono utilizzate per generare una sezione critica in caso di verifica di una determinate condizione; per attivare la condition variable, abbiamo utilizzato ***pthread\_cond\_broadcast***.
  + Per garantire la terminazione contemporanea di tutti i thread attivi nell’applicativo abbiamo utilizzato una serie di System Call del tipo   
    ***pthread\_join***:



**6 – Appendice**