



UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Università degli Studi di Napoli "Federico II"
Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Corso di Laurea in Informatica

LABORATORIO DI SISTEMI OPERATIVI:

Progetto:

SISTEMA DI DIFESA AEREA

Docente:

Clemente Galdi

Studenti:

Paolo Perrotta N86/23

Giovanni Di Matteo N86/1490

Calise Giuseppe N86/1334



Indice

1 - Descrizione del progetto.....	2
2 - Guida d'uso	3
2.1 - Server di Rendezvous	4
2.2 - Server - Torretta	4
2.3 - Client - Bersaglio.....	5
3 - Uso.....	5
3.1 - Server di Rendezvous	5
3.2 - Server - Torretta	6
3.3 - Client - Bersaglio.....	6
3.4 – Esempio di utilizzo	7
4 – Protocollo di comunicazione	8
4.1 – Comunicazione Rendezvous - Server	8
4.2 – Comunicazione Client - Server.....	8
4.3 – Comunicazione Server - Server	9
5 – Dettagli implementativi.....	10
6 – Appendice.....	12

1 - 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, dopodiché 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.

2 - Guida d'uso

L'applicazione è sviluppata su piattaforma UNIX rispettando lo Standard POSIX. Per la compilazione e l'esecuzione di essa, bisogna aver installato sulla macchina un Sistema Operativo Unix/Linux ed il compilatore *gcc* (GNU Compiler Collection). Inoltre, tutti i sorgenti del progetto devono essere compilati.

- 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:

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

Il Server:

Parametri	Descrizione
ID	L'ID assegnato alla Torretta – Server
IP: Porta Rendezvous	L'IP e porta del Server di Rendezvous a cui connettersi
Porta Server	La porta da assegnare al Torretta-Server
Raggio	Il raggio entro cui la Torretta- Server può colpire i bersagli
Numero Missili	Il numero di missili a disposizione della Torretta-Server
Posizione (X,Y)	La posizione della Torretta sulla "scacchiera"

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"

2.1 - Server di Rendezvous

Una volta raggiunta la cartella adeguata tramite il comando: **cd Scrivania/LSO**, il sorgente deve essere compilato nel modo seguente:

Compilazione: gcc -pthread -o r Rendezvous.c

L'opzione "-o" permette di rinominare il sorgente; l'opzione "-pthread" è necessaria per segnalare al *linker* l'inclusione della libreria pthread.h; l'eseguibile verrà generato all'interno della directory in cui ci si trova e per avviarne l'esecuzione bisogna specificare la porta.

Esecuzione: ./r <Numero di Porta>

Esempio: ./r 4000

In caso di parametro non specificato, verrà generato un messaggio di errore.

2.2 - Server - Torretta

Una volta raggiunta la cartella adeguata tramite il comando: **cd Scrivania/LSO**, dopo aver avviato Server di Rendezvous e Server, un messaggio notificherà all'utente che il Server è connesso ed in stato di attesa.

La compilazione avverrà nel modo seguente:

Compilazione: gcc -pthread -o s Server.c -lm

Oltre all'opzione "-pthread", "-lm" è necessaria per segnalare al *linker* l'inclusione della libreria math.h; l'eseguibile verrà generato all'interno della directory in cui ci si trova e per avviarne l'esecuzione bisogna specificare ID, IP e porta di Rendezvous, coordinata x e coordinata y, numero missili, raggio e porta del Server.

Esecuzione:

./s<ID><IP:Porta_Rendezvous><X,Y><Numero_Missili><Raggio><Porta_Server>

Esempio: ./s 10 127.0.0.01:4000 5,5 20 10 5000

Se il numero dei parametri è inferiore a 7, verrà visualizzato a video un messaggio di errore.

2.3 - Client

Una volta raggiunta la cartella adeguata tramite il comando: **cd Scrivania/LSO**, il sorgente deve essere compilato nel modo seguente:

Compilazione: gcc -o c Client.c

L'eseguibile verrà generato all'interno della directory in cui ci si trova e per avviarne l'esecuzione bisogna specificare l'ID del bersaglio, IP Server, porta del Server, coordinata x e coordinata y.

Esecuzione: ./c <ID><IP_Server><Porta_Server><X,Y>

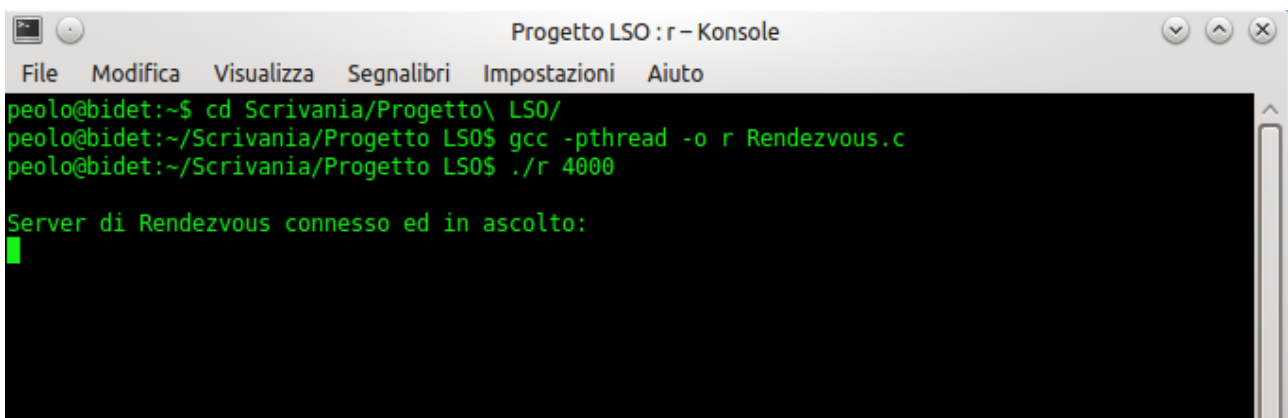
Esempio: ./c 100 127.0.0.1 5000 7,8

In caso di parametri non specificati oppure inferiori a 4, verrà generato un messaggio di errore.

3 - Uso

3.1 - Server di Rendezvous

Avviato il Server di Rendezvous si visualizzerà in output l'effettiva sua connessione e fase di 'ascolto' di nuovi Server. Inoltre, alla connessione di un Server, sarà possibile visualizzare ID e IP della torretta attiva in rete.

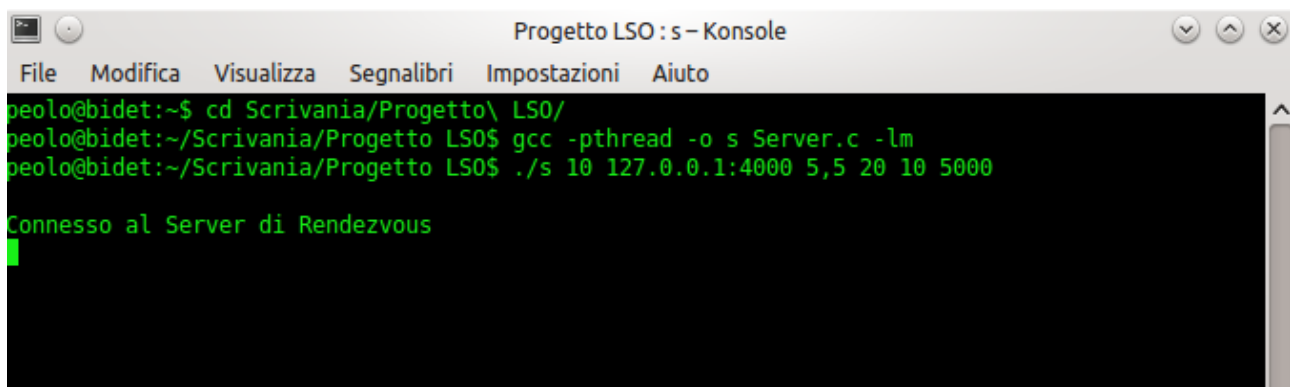


```
Progetto LSO : r - Konsole
File  Modifica  Visualizza  Segnalibri  Impostazioni  Aiuto
peolo@bidet:~$ cd Scrivania/Progetto\ LSO/
peolo@bidet:~/Scrivania/Progetto LSO$ gcc -pthread -o r Rendezvous.c
peolo@bidet:~/Scrivania/Progetto LSO$ ./r 4000

Server di Rendezvous connesso ed in ascolto:
█
```

3.2 - Server – Torretta

Avviato il Server si visualizzerà in output l'esito positivo della connessione al Server di Rendezvous e sarà successivamente visualizzabile una lista delle eventuali torrette che entreranno in rete.

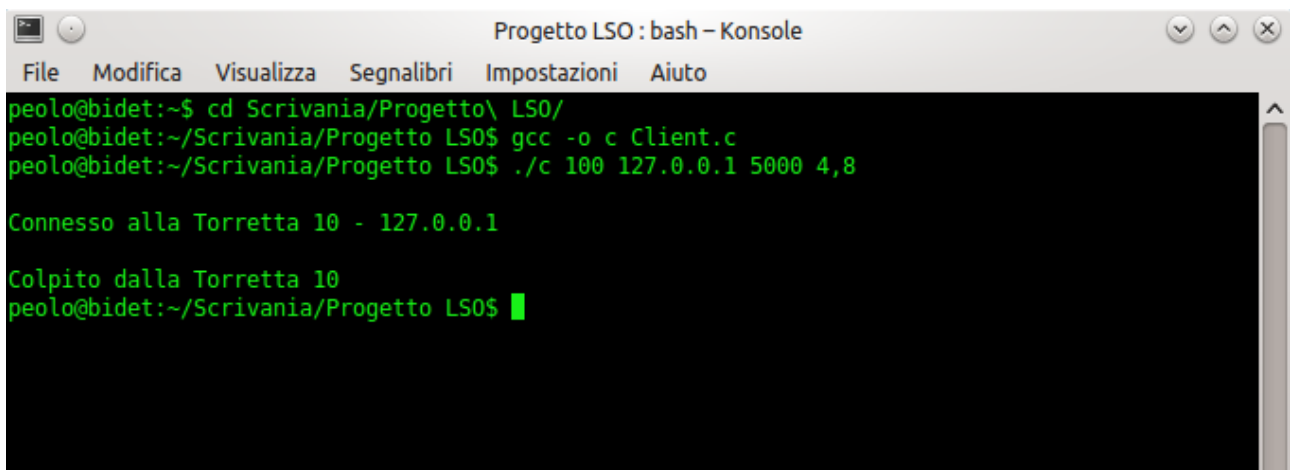


```
peolo@bidet:~$ cd Scrivania/Progetto\ LSO/
peolo@bidet:~/Scrivania/Progetto LSO$ gcc -pthread -o s Server.c -lm
peolo@bidet:~/Scrivania/Progetto LSO$ ./s 10 127.0.0.1:4000 5,5 20 10 5000

Connesso al Server di Rendezvous
```

3.3 - Client – Bersaglio

Avviato il Client si visualizzerà in output la connessione alla torretta ed il suo esito.



```
peolo@bidet:~$ cd Scrivania/Progetto\ LSO/
peolo@bidet:~/Scrivania/Progetto LSO$ gcc -o c Client.c
peolo@bidet:~/Scrivania/Progetto LSO$ ./c 100 127.0.0.1 5000 4,8

Connesso alla Torretta 10 - 127.0.0.1

Colpito dalla Torretta 10
peolo@bidet:~/Scrivania/Progetto LSO$
```

3.4 – 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.

Di seguito uno Screen catturato durante l'esecuzione che mostra il Rendezvous in attesa di una eventuale connessione di una o più torrette, che verranno poco dopo connesse e visualizzate.

Dopodichè si collega un Client, la sua posizione viene visualizzata dal Server e, tramite un apposito algoritmo, riceve l'esito del bersaglio.

Il server mostra quale torretta lo ha abbattuto ed il numero di missili, mentre il bersaglio visualizza il suo esito con relativo messaggio.

```
Progetto LSO : r - Konsole
peolo@bidet:~$ cd Scrivania/Progetto\ LSO/
peolo@bidet:~/Scrivania/Progetto LSO$ gcc -pthread -o r Rendezvous.c
peolo@bidet:~/Scrivania/Progetto LSO$ ./r 4000

Server di Rendezvous connesso ed in ascolto:

Connessione della Torretta 10 - 127.0.0.1
Connessione della Torretta 11 - 127.0.0.1
[]

Progetto LSO : s - Konsole
peolo@bidet:~$ cd Scrivania/Progetto\ LSO/
peolo@bidet:~/Scrivania/Progetto LSO$ gcc -pthread -o s Server.c -lm
peolo@bidet:~/Scrivania/Progetto LSO$ ./s 10 127.0.0.1:4000 5,5 20 10 5000

Connesso al Server di Rendezvous

Bersaglio 100 da colpire in posizione (4,8)
Bersaglio 100 abbattuto dalla Torretta 10
Numero missili Torretta 10: 19
Server 127.0.0.1 connesso
[]

Progetto LSO : bash - Konsole
peolo@bidet:~$ cd Scrivania/Progetto\ LSO/
peolo@bidet:~/Scrivania/Progetto LSO$ gcc -o c Client.c
peolo@bidet:~/Scrivania/Progetto LSO$ ./c 100 127.0.0.1 5000 4,8

Connesso alla Torretta 10 - 127.0.0.1

Colpito dalla Torretta 10
peolo@bidet:~/Scrivania/Progetto LSO$ []

Progetto LSO : s - Konsole <2>
peolo@bidet:~$ cd Scrivania/Progetto\ LSO/
peolo@bidet:~/Scrivania/Progetto LSO$ ./s 11 127.0.0.1:4000 6,7 10 5 6000

Connesso al Server di Rendezvous

Parametri ricevuti sul Server 127.0.0.1
[]
```


4–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.

4.1 – 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.

4.2 – 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.

4.3 – 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.

5–Dettagli implementativi

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

Parametri	Descrizione
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>)

Parametri	Descrizione
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_equal
- System Call per la gestione della maschera dei segnali bloccati:

```
sigset_t set;  
sigaddset(&set, SIGPIPE); /* Blocca il segnale SIGPIPE in caso di server disconnesso */  
sigprocmask(SIG_SETMASK, &set, NULL);
```

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*:

```
/* Si evita di indicare la porta in ascolto per un altro Server, inviandolo direttamente al Rendezvous*/  
if(getsockname(sock_server, (struct sockaddr *)&listen_server, (socklen_t *)&lung_server)<0)  
{  
    write( 1, "\nErrore getsockname", strlen("\nErrore getsockname") );  
    exit(EXIT_FAILURE);  
}
```

- 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**:

```
/* Attende che terminano gli altri thread */  
pthread_join(th,NULL);  
pthread_join(th2,NULL);  
pthread_join(th3,NULL);
```

6–Appendice

```
/* Rendezvous */

/* Inclusione librerie */

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/un.h>

#include <arpa/inet.h>

#include <pthread.h>

#include <signal.h>

#include <unistd.h>

#include <fcntl.h>

/* Struttura che contiene informazioni sul Server connesso */

typedef struct lista_server

{

    /* Tid processo */

    pthread_t thr;

    /* ID della torretta connessa */

    int id_torretta;

    /* Struttura che contiene le informazioni da inviare ai Server */

    struct sockaddr_in info;

}lista_server;

/* Dichiarazione variabili globali */

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; /* Semaforo che gestisce la sincronizzazione tra thread */

pthread_mutex_t mutex_delete = PTHREAD_MUTEX_INITIALIZER; /* Semaforo che impedisce la cancellazione dei parametri dalla 'lista_server' durante l'utilizzo del master thread */

pthread_cond_t cond = PTHREAD_COND_INITIALIZER; /* Cond è utilizzato per far inviare i parametri al nuovo Server connesso */

pthread_cond_t cond2 = PTHREAD_COND_INITIALIZER; /* Cond2 segnala al master thread che i Server hanno inviato le informazioni all'ultimo thread connesso */

lista_server ult_info; /* Variabile usata per memorizzare le informazioni sull'ultimo server connesso */

lista_server *list; /* Lista che contiene le informazioni dei Server connessi */
```

```

int cont_thread_info = 0; /* Contatore dei thread che hanno inviato le informazioni all'ultimo Server connesso */

int cont_elem = 0; /* Contatore degli elementi connessi */

/* Prototipi */

void *gestione_server(void* fd); /* Funzione che gestisce la comunicazione di un Server */

void *invia_info(void* fd); /* Funzione che invia nuove informazioni al Server */

/* Parametri: (1) Porta Server di Rendezvous */

int main(int argc, char **argv)

{
    int i, sock, *sock_accept = NULL, lung_host = sizeof(struct sockaddr_in);

    pthread_t *th; /* Id thread che gestira' le comunicazioni tra un server e il server di rendezvous */

    struct sockaddr_in rendezvous; /* Contiene le informazioni sul Server di Rendezvous */

    struct sockaddr_in host; /* Contiene le informazioni sul Server connesso */

    struct sockaddr_in fine_lista; /* Messaggio di fine lista */

    int port[2]; /* Contiene ID e numero porta del nuovo Server connesso */

    char stampa_info[100] = ""; /* Visualizza le informazioni dell'ultimo server connesso */

    int dim_list = 100; /* Dimensione della lista_server */

    sigset_t set;

    sigaddset(&set,SIGPIPE); /* Blocca il segnale SIGPIPE in caso di server disconnesso */

    sigprocmask(SIG_SETMASK,&set,NULL);

    /* Inizializza il messaggio di fine lista */

    fine_lista.sin_addr.s_addr = 0;

    fine_lista.sin_port = 0;

    /* Controlla il numero dei parametri da linea di comando */

    if( argc < 2)

    {

        write( 1, "\nParametri non inseriti da linea di comando\n", strlen("\nParametri non inseriti da linea di comando\n"));

        exit(EXIT_FAILURE);

    }

    list = malloc(sizeof(lista_server)*dim_list);

    /* Inizializza comunicazione con protocollo TCP/IP e numero di porta inserito da riga di comando */

    rendezvous.sin_family = AF_INET;

    rendezvous.sin_port = htons(atoi(argv[1]));

    rendezvous.sin_addr.s_addr = htonl(INADDR_ANY);

```

```

/* Crea socket per le comunicazioni */
if(( sock=socket(AF_INET, SOCK_STREAM, 0)) == -1 )
{
    perror("\nErrore Socket");
    exit(EXIT_FAILURE);
}

/* Bind */
if( (bind (sock, (struct sockaddr *)&rendezvous, sizeof(rendezvous))) == -1 )
{
    perror("\nErrore Bind");
    exit(EXIT_FAILURE);
}

/* Listen */
if( listen(sock,100) == -1 )
{
    perror("\nErrore listen");
    exit(EXIT_FAILURE);
}

write( 1, "\nServer di Rendezvous connesso ed in ascolto:\n", strlen("\nServer di Rendezvous connesso ed in
ascolto:\n") );

while(1)
{
    /* Alloca spazio per un nuovo file descriptor */
    sock_accept = malloc(sizeof(int));

    /* Accept */
    *sock_accept = accept( sock, (struct sockaddr *)&host,(socklen_t*)&lung_host);

    /* Legge ID e numero porta del nuovo Server*/
    read( *sock_accept , port,2*sizeof(int));

    /* Scrive le informazioni dell'ultimo Server in una variabile globale per poter essere lette da tutti gli altri
    Server della lista */
    ult_info.info.sin_addr.s_addr = host.sin_addr.s_addr;

    ult_info.info.sin_family = AF_INET;

    ult_info.info.sin_port = htons(port[0]);

    ult_info.id_torretta= port[1];
}

```

```

/* Acquisizione del mutex */

pthread_mutex_lock(&mutex_delete);

/* Controlla se la lista può ospitare un altro elemento */

if( dim_list == cont_elem )

{

    /* Altrimenti si raddoppia la dimensione della lista */

    dim_list = dim_list*2;

    list = (lista_server*)realloc( list, sizeof(lista_server)*dim_list );

}

/* Verifica che il server appena connesso abbia un ID diverso dai Server nella lista */

for( i=0; (i<cont_elem) && (list[i].id_torretta!=ult_info.id_torretta); i++ );

if(list[i].id_torretta == ult_info.id_torretta )

{

    /* Se l'ID non è univoco, rifiuta la connessione */

    write( *sock_accept, "NOT",strlen("NOT"));

    pthread_mutex_unlock(&mutex_delete);

    continue;

}

write( *sock_accept, "YES",strlen("YES")); /* Se l'ID è univoco permette la connessione */

sprintf( stampa_info, "\nConnessione della Torretta %d -
%s\n",port[1],inet_ntoa(ult_info.info.sin_addr));

write( 1, stampa_info, strlen(stampa_info));

/* Invia all'ultimo Server connesso le informazioni sui Server connessi */

for( i=0; i < cont_elem; i++ )

    write( *sock_accept, &list[i].info, sizeof(struct sockaddr_in));

write( *sock_accept, &fine_lista, sizeof(struct sockaddr_in) );/* Invia il messaggio di fine lista */

cont_thread_info = 0; /* Reset del contatore */

pthread_cond_broadcast(&cond); /* Risveglia i thread in attesa di leggere le informazioni sull'ultimo
Server connesso */

/* Attende che tutti i thread inviino le informazioni sull'ultimo Server connesso */

while( cont_thread_info < cont_elem )

    pthread_cond_wait(&cond2,&mutex);

th = malloc(sizeof(pthread_t));

```



```

/* Crea un thread per gestire le comunicazioni con il nuovo Server */
if((pthread_create( th, NULL, (void*) &gestione_server, sock_accept)) == -1)
{
    perror("\nErrore pthread_create");
    continue;
}

/* Copia le informazioni del nuovo Server nella lista */
memcpy(&(list[cont_elem].thr),th,sizeof(pthread_t));
list[cont_elem].info.sin_addr.s_addr = ult_info.info.sin_addr.s_addr ;
list[cont_elem].info.sin_family = ult_info.info.sin_family;
list[cont_elem].info.sin_port = ult_info.info.sin_port;
list[cont_elem].id_torretta= ult_info.id_torretta;
cont_elem++; /* Incrementa il numero di Server connessi */

/* Rilascia i mutex */
pthread_mutex_unlock(&mutex);
pthread_mutex_unlock(&mutex_delete);
}

return 0;
}

void *gestione_server(void* fd)
{
    int sock = *(int*)fd, n, i;
    pthread_t tid;
    char check, da_canc[100];
    free(fd);
    pthread_create( &tid, NULL, (void*)invia_info, &sock ); /* Crea un thread per inviare informazioni al Server */
    n=read( sock, &check, 1 ) ; /* Rimane in ascolto */

    /* Gestione Socket broken */
    if( n == 0 )
    {
        /* Blocca il mutex_delete per evitare problemi di inconsistenza della lista quando viene usata dal
        master thread */
        pthread_mutex_lock(&mutex_delete);

        /* Ricerca delle informazioni sul server gestito */

```

```

for( i = 0; pthread_equal(pthread_self(), list[i].thr) == 0 ; i++ );

sprintf(da_canc, "\nServer %s disconnesso\n", inet_ntoa(list[i].info.sin_addr));

write( 1, da_canc, strlen(da_canc) );

/* Scambia l'ultimo elemento della lista con le informazioni */

memcpy(&(list[i].thr), &(list[cont_elem-1].thr), sizeof(pthread_t));

list[i].info.sin_addr.s_addr = list[cont_elem-1].info.sin_addr.s_addr;

list[i].info.sin_port = list[cont_elem-1].info.sin_port;

list[i].id_torretta = list[cont_elem-1].id_torretta;

/* Azzera l'ultimo elemento */

list[cont_elem-1].info.sin_addr.s_addr = 0;

list[cont_elem-1].info.sin_port = 0 ;

list[cont_elem-1].id_torretta = 0;

cont_elem--; /* Aggiorna il numero di elementi nella lista_server */

pthread_mutex_unlock(&mutex_delete); /* Rilascia il mutex_delete */

close(sock); /* Chiusura del socket broken */

pthread_cancel( tid ); /* Cancella il thread 'invia_info' */

pthread_exit(NULL); /* Termina thread */

}

return NULL;

}

/* Invia al Server le informazioni del nuovo Server connesso */

void *invia_info(void* fd)

{

    while(1)

    {

        pthread_cond_wait( &cond, &mutex); /* Attende che il master-thread faccia leggere le informazioni in 'ult_info' */

        write( *(int*)fd, &ult_info.info, sizeof(struct sockaddr_in)); /* Invia le informazioni */

        cont_thread_info++; /* Incrementa il contatore dei thread che hanno inviato le informazioni */

        pthread_mutex_unlock(&mutex); /* Rilascia il mutex */

        pthread_cond_broadcast(&cond2); /* Risveglia il master thread */

    }

}

```

```

/* Server - Torretta */

/* Inclusione librerie */

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/un.h>

#include <arpa/inet.h>

#include <pthread.h>

#include <signal.h>

#include <math.h>

#include <time.h>

#include <unistd.h>

#include <fcntl.h>

/* Definizione costante */

#define MAX_THREAD 500

/* Struttura che contiene le mie informazioni */

typedef struct postazione
{
    int id;

    int posizione[2];

    int numero_missili;

    int raggio;
}postazione;

/* Dichiarazione variabili globali */

postazione pos;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

pthread_mutex_t mutex2 = PTHREAD_MUTEX_INITIALIZER;

pthread_mutex_t mutex_esito = PTHREAD_MUTEX_INITIALIZER;

pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

pthread_cond_t cond_esito = PTHREAD_COND_INITIALIZER;

pthread_cond_t cond2 = PTHREAD_COND_INITIALIZER;

```

```

int ult_info[3]; /* Variabile che attende l'esito del bersaglio: [0] = id_bersaglio, [1] = coordinata X, [2] = coordinata Y */

int cont_server_connessi = 1; /* Conta il numero di Server connessi */

int cont_server_pos = 0; /* Conta il numero dei server a cui è stata inviata la posizione del bersaglio */

int **lista_candidati; /* Contiene le risposte dei Server: [0] = id_server, [1] = risultato algoritmo che decide il bersaglio da colpire */

int destino[2]; /* Contiene l'esito dell'ultimo bersaglio */

int dim = 100; /* Dimensione di 'lista_candidati' */

/* Prototipi */

void *comunicazione_server(void *fd);

void *accetta_server(void *fd);

void *accetta_bersaglio(void *fd);

void *lettura_rend(void *fd);

void *gestione_bersaglio(void *fd);

int esito(int x, int y, int r);

void *gestisci_server(void* fd);

void *bersaglio_avvistato(void* fd);

void stampa_esito(int id_bersaglio, int id_server, int esito);

/* Parametri: (1) ID (2) IP:Porta Rendezvous (3) X,Y (4) Numero missili (5) Raggio (6) Porta Server*/

int main(int argc, char **argv)

{

    int sock_rendezvous, sock_server, sock_client, *connect_server = NULL, port[2], i, lung_server = sizeof(struct
sockaddr_in);

    struct sockaddr_in rendezvous; /* Contiene IP e numero di porta del Server di Rendezvous */

    struct sockaddr_in listen_server; /* Contiene i dati per la connessione dei Server */

    struct sockaddr_in listen_client; /* Contiene i dati per la connessione dei Client */

    struct sockaddr_in ricevuto; /* Contiene le informazioni sui Server inviati dal Rendezvous */

    char *ip_rend, *porta_rend, stampa_ultimo[100], esito_connessione[3]="\0";

    pthread_t th, th2, th3;

    sigset_t set;

    /* Blocca il segnale SIGPIPE in caso di disconnessione di un Server */

    sigaddset(&set,SIGPIPE);

    sigprocmask(SIG_SETMASK,&set,NULL);

    /* Controlla il numero dei parametri da riga di comando */

    if( argc < 7)

```

```

{
    write( STDERR_FILENO, "\nNon sono stati inseriti tutti i parametri da riga di comando\n", strlen("\nNon
sono stati inseriti tutti i parametri da riga di comando\n"));

    exit(EXIT_FAILURE);
}

/* Inizializza i dati della postazione */

pos.id = atoi(argv[1]); /* Estrae ID da riga di comando */
ip_rend = strtok(argv[2], ":\n"); /* Estrae IP da riga di comando */
porta_rend = strtok(NULL, ":\n"); /* Estrae il numero di porta da riga di comando */
pos.posizione[0] = atoi(strtok(argv[3], ":\n")); /* Estrae la coordinata x da riga di comando */
pos.posizione[1] = atoi(strtok(NULL, ":\n")); /* Estrae la coordinata y da riga di comando */
pos.numero_missili = atoi(argv[4]); /* Estrae il numero dei missili da riga di comando */
pos.raggio = atoi(argv[5]); /* Estrae il range da riga di comando */
port[1] = atoi(argv[1]);

/* Controlla se l'ID è maggiore di 0 */
if( pos.id < 1 )
{
    write( 1, "\nL'ID deve essere maggiore di 0\n", strlen("\nL'ID deve essere maggiore di 0\n") );
    exit(EXIT_FAILURE);
}

/* Creazione socket per la connessione con il Server di Rendezvous */
if(( sock_rendezvous=socket(AF_INET, SOCK_STREAM, 0)) == -1 )
{
    perror("\nErrore creazione socket Rendezvous");
    exit(EXIT_FAILURE);
}

/* Creazione socket per la connessione ai Server */
if(( sock_server=socket(AF_INET, SOCK_STREAM, 0)) == -1 )
{
    perror("\nErrore creazione socket Server");
    exit(EXIT_FAILURE);
}

/* Creazione socket per la connessione al Client */
if(( sock_client=socket(AF_INET, SOCK_STREAM, 0)) == -1 )

```

```

{
    perror("\nErrore creazione socket Client");
    exit(EXIT_FAILURE);
}

/* Inizializza la struttura sockaddr_in per la connessione al Server di Rendezvous */
rendezvous.sin_addr.s_addr = inet_addr(ip_rend);
rendezvous.sin_family = AF_INET;
rendezvous.sin_port = htons(atoi(porta_rend));

/* Inizializza la struttura sockaddr_in per accettare le connessioni da altri Server */
listen_server.sin_family = AF_INET;
listen_server.sin_addr.s_addr = htonl(INADDR_ANY);

/* Inizializza la struttura sockaddr_in per accettare le connessioni dai Client */
listen_client.sin_family = AF_INET;
listen_client.sin_port = htons(atoi(argv[6]));
listen_client.sin_addr.s_addr = htonl(INADDR_ANY);

/* Listen senza bind per avere una porta libera */
if( listen(sock_server, MAX_THREAD) == -1 )
{
    perror("\nErrore listen client");
    pthread_exit(NULL);
}

/* Si evita di indicare la porta in ascolto per un altro Server, inviandolo direttamente al Rendezvous */
if(getsockname(sock_server, (struct sockaddr *)&listen_server,(socklen_t *)&lung_server)<0)
{
    write( 1, "\nErrore getsockname", strlen("\nErrore getsockname") );
    exit(EXIT_FAILURE);
}

port[0] = htons(listen_server.sin_port); /* Inserisce la porta in una stringa */

/* Bind Client */
if( (bind (sock_client, (struct sockaddr *)&listen_client, sizeof(listen_client))) == -1 )
{
    perror("\nErrore bind client");
    exit(EXIT_FAILURE);
}

```

```

}

/* Connessione al Server di Rendezvous */
if (connect(sock_rendezvous , (struct sockaddr *)&rendezvous , sizeof(rendezvous)) < 0)
{
    perror("\nErrore connessione Server di Rendezvous");
    exit(EXIT_FAILURE);
}

write( 1, "\nConnesso al Server di Rendezvous\n", strlen("\nConnesso al Server di Rendezvous\n") );
write( sock_rendezvous, port, 2*sizeof(int)); /* Invia ID e numero di porta al Server di rendezvous */
read( sock_rendezvous, esito_connessione,3);
esito_connessione[2]='\0';

/* Se il Rendezvous rifiuta la connessione */
if(strcmp(esito_connessione,"NOT") == 0)
{
    write( 1, "\nID appartenente ad un altro Server\n", strlen("\nID appartenente ad un altro Server\n") );
    exit(EXIT_FAILURE);
}

/* Creazione thread che gestisce le connessioni */
if((pthread_create(&th ,NULL,(void*)accetta_server,&sock_server)) == -1 )
{
    perror("\nErrore accetta_server");
    exit(EXIT_FAILURE);
}

/* Attende finchè non riceve end_of_line */
while(read(sock_rendezvous, &ricevuto, sizeof(struct sockaddr_in))>0 && ricevuto.sin_port!=0)
{
    connect_server = malloc(sizeof(int));

    /* Creazione socket per la connessione al Server */
    if(( *connect_server=socket(AF_INET, SOCK_STREAM, 0)) == -1 )
    {
        perror("\nErrore creazione socket Server");
        continue;
    }
}

```

```

/* Connessione ad un Server della lista */

if (connect(*connect_server , (struct sockaddr *)&ricevuto , sizeof(ricevuto)) < 0)

{

    perror("\nErrore connessione al Server");

    continue;

}

/* Crea un thread con ogni Server in comunicazione */

if((pthread_create( &th, NULL, (void*) &gestisci_server, connect_server)) == -1)

{

    perror("\nErrore pthread_create");

    continue;

}

sprintf( stampa_ultimo , "\nParametri ricevuti sul Server %s\n", inet_ntoa(ricevuto.sin_addr) );

write( 1, stampa_ultimo, strlen(stampa_ultimo) ); /* Visualizza IP e numero di porta */

cont_server_connessi++; /* Incrementa il contatore dei Server connessi */

}

dim = cont_server_connessi + 100; /* Aggiunge in 'dim' lo spazio sufficiente per ricevere le informazioni dai
Server */

lista_candidati =(int**) malloc ( dim * sizeof( int *));

for( i=0; i<dim ; i++)

    lista_candidati[i] = malloc(sizeof(int)*2);

/* Thread per gestire le connessioni ai bersagli */

if((pthread_create(&th2 ,NULL,(void*)&accetta_bersaglio, &sock_client)) == -1 )

{

    perror("\nErrore accetta_bersaglio");

    exit(EXIT_FAILURE);

}

/* Thread creato per gestire i messaggi del Rendezvous sui nuovi Server connessi */

if((pthread_create(&th3 ,NULL,(void*)&lettura_rend, &sock_rendezvous)) == -1)

{

    perror("\nErrore sock_rendezvous");

    exit(EXIT_FAILURE);

}

```



```

/* Attende che terminano gli altri thread */

pthread_join(th,NULL);

pthread_join(th2,NULL);

pthread_join(th3,NULL);

}

/* Funzione che accetta connessioni da altri Server */

void *accetta_server(void *fd)

{

    int *accept_fd;

    pthread_t th;

    struct sockaddr_in server;

    int lung_server = sizeof(struct sockaddr_in);

    while(1)

    {

        accept_fd = malloc(sizeof(int));

        if((*accept_fd = accept(*(int*)fd, (struct sockaddr *)&server,(socklen_t*)&lung_server)) == -1 )

        {

            write( 1, "\nErrore accept server\n", strlen("\nErrore accept server\n") );

            continue;

        }

        if( (pthread_create( &th, NULL, (void*)bersaglio_avvistato,(void*) accept_fd)) == -1)

            write( 1, "\nErrore connessione Server\n", strlen("\nErrore connessione Server\n") );

    }

    return NULL;

}

/* Funzione che notifica l'attacco di una torretta con l'esito del bersaglio */

void *bersaglio_avvistato(void *fd)

{

    int sock = *(int*)fd, dati_ricevuti[3], n, controllo, risultato[2], destino[2];

    char stampa_bersaglio[100], status[100], id[100];

    struct timespec t; /* Struttura usata da timedlock */

    free(fd);

    risultato[0] = pos.id;

```

```

while(1)
{
    n=read( sock, dati_ricevuti, sizeof(int)*3 );

    /* Verifica se il Server in comunicazione è attivo */

    if( n != 0 )
    {
        sprintf(stampa_bersaglio, "\nBersaglio %d in posizione
        (%d,%d)\n", dati_ricevuti[0], dati_ricevuti[1], dati_ricevuti[2]);

        write( 1, stampa_bersaglio, strlen(stampa_bersaglio) );

        clock_gettime(CLOCK_REALTIME, &t);

        t.tv_sec += 2;

        /* Cerca di acquisire il lock entro un certo intervallo di tempo */

        controllo = pthread_mutex_timedlock(&mutex2, &t);

        /* Se il mutex2 è stato acquisito */

        if( controllo == 0 )
        {
            risultato[1] = esito(dati_ricevuti[1], dati_ricevuti[2], pos.raggio); /* Calcola l'esito del
            bersaglio */

            write(sock, risultato, sizeof(int)*2); /* Invia l'esito al Server */

            read(sock, destino, sizeof(int)*2); /* Legge il destino del bersaglio */

            /* Controlla se i parametri sono sufficienti per abbattere il bersaglio */

            if( destino[0]==pos.id && pos.numero_missili > 0 )

                pos.numero_missili--; /* Decrementa il numero dei missili */

            pthread_mutex_unlock(&mutex2); /* Rilascia il mutex */

        }

        /* Altrimenti notifica il Server che non è disponibile ad attaccare il bersaglio */

        else
        {
            risultato[1] = -2;

            write(sock, risultato, sizeof(int)*2);

            read(sock, destino, sizeof(int)*2);

        }

        /* Stampa l'esito del bersaglio */

        stampa_esito(dati_ricevuti[0], destino[0], destino[1]);
    }
}

```

```

        sprintf(status, "\nNumero missili Torretta %d: %d\n", destino[0], pos.numero_missili);

        write( 1, status, strlen(status) );

    }

    /* Altrimenti visualizza un messaggio se la connessione è stata persa con il Server */

    else

    {

        write( 1, "\nTorretta disconnessa\n", strlen("\nTorretta disconnessa\n") );

        close(sock);

        pthread_exit(NULL);

    }

}

/* Funzione che gestisce la comunicazione con altri Server con l'esito dei Server candidati */

void *gestisci_server(void* fd)

{

    int sock = *(int*)fd, n, ricevuto[2];

    while(1)

    {

        pthread_cond_wait(&cond,&mutex); /* Thread sbloccato da 'gestione_client' */

        write(sock, ult_info, sizeof(int)*3); /* Manda i parametri sul bersaglio al server di cui gestisce il file descriptor */

        n=read(sock, ricevuto, sizeof(int)*2); /* Legge l'esito e lo inserisce nella lista dei candidati */

        if( n == 0 )

        {

            cont_server_connessi--;

            pthread_mutex_unlock(&mutex);

            pthread_cond_broadcast(&cond2);

            close(sock);

            write( 1, "\nConnessione persa col Server\n", strlen("\nConnessione persa col Server\n") );

            pthread_exit(NULL);

        }

        /* Copia dati ricevuti dalla lista */

        lista_candidati[cont_server_pos][0]=ricevuto[0];

```

```

        lista_candidati[cont_server_pos][1]=ricevuto[1];

        cont_server_pos++;

        pthread_mutex_unlock(&mutex); /* Rilascia il mutex per poter scrivere nella lista */

        pthread_cond_broadcast(&cond2); /* Notifica il thread 'gestione_bersaglio' che ha scritto nella lista */

        pthread_cond_wait(&cond_esito,&mutex_esito); /* Attende che 'gestione_server' calcoli l'esito */

        write(sock, destino, sizeof(int)*2); /* Invia l'esito */

        cont_server_pos++; /* Indica che ha inviato l'esito al Server in comunicazione */

        pthread_mutex_unlock(&mutex_esito); /* Rilascia il mutex per consentire ad altri di inviare l'esito */

        pthread_cond_broadcast(&cond2); /* Segnala a 'gestione_bersaglio' che ha inviato l'esito */

    }

}

/* Ascolta sulla porta dei missili e li gestisce */

void *accetta_bersaglio(void *fd)
{
    int *accept_fd;

    pthread_t th;

    struct sockaddr_in client;

    int lung_client = sizeof(struct sockaddr_in);

    if( listen(*(int*)fd, MAX_THREAD) == -1 )
    {
        perror("\nErrore listen client");

        pthread_exit(NULL);

    }

    while(1)
    {
        accept_fd = malloc(sizeof(int));

        if((*accept_fd = accept(*(int*)fd, (struct sockaddr *)&client,(socklen_t*)&lung_client)) < 0 )
        {
            write( 1, "\nErrore accept client\n", strlen("\nErrore accept client\n") );

            continue;

        }

        /* Crea thread per la gestione del bersaglio */

        if((pthread_create( &th, NULL, (void*)gestione_bersaglio, accept_fd)) == -1 )

```

```

        write( 1, "\nErrore gestione missile\n", strlen("\nErrore gestione missile\n") );

    }

    return NULL;

}

/* Gestione client */

void *gestione_bersaglio(void *fd)
{
    int sock = *(int*)fd, dati_bersaglio[3], i, max = -1, id_vincitore = -1, risultato;

    char stampa_bersaglio[100], status[100];

    /* Legge i dati inviati dal Client secondo un formato prestabilito [0] = id_bersaglio, [1] = coordinata X, [2] = coordinata Y */

    if( (read( sock, dati_bersaglio, sizeof(int)*3)) < 0)

    {
        perror("\nErrore lettura dati client");

        pthread_exit(NULL);

    }

    pthread_mutex_lock(&mutex2); /* Acquisisce il lock per gestire un missile alla volta */

    pthread_mutex_lock(&mutex); /* Acquisisce il lock per modificare 'ult_info' */

    sprintf(stampa_bersaglio, "\nBersaglio %d da colpire in posizione (%d,%d)\n", dati_bersaglio[0], dati_bersaglio[1], dati_bersaglio[2]);

    write( 1, stampa_bersaglio, strlen(stampa_bersaglio) );

    /* Inserisce nella variabile globale le informazioni del missile */

    ult_info[0] = dati_bersaglio[0]; /* ID */

    ult_info[1] = dati_bersaglio[1]; /* Coordinata x */

    ult_info[2] = dati_bersaglio[2]; /* Coordinata y */

    cont_server_pos = 0;

    /* Esegue l'algoritmo e l'inserimento in cima alla 'lista_candidati' ed incremento di 'cont_server_pos' */

    risultato = esito( dati_bersaglio[1], dati_bersaglio[2], pos.raggio);

    lista_candidati[0][0] = pos.id;

    lista_candidati[0][1] = risultato;

    cont_server_pos++;

    pthread_mutex_unlock(&mutex);

    pthread_cond_broadcast(&cond); /* Risveglia tutti i thread in attesa */

```

```

/* Attende che tutti abbiano inviato il messaggio con l'esito in 'lista_candidati' */
while( cont_server_pos < cont_server_connessi)

pthread_cond_wait(&cond2,&mutex);

/* Scandisce la lista dei candidati per determinare l'esito del bersaglio */
for ( i = 0; i < cont_server_connessi; i++ )
{
    if( lista_candidati[i][1] > max )
    {
        id_vincitore = lista_candidati[i][0];
        max = lista_candidati[i][1];
    }
}

/* Scrive il destino del bersaglio in una variabile globale */
destino[0] = id_vincitore;
destino[1] = max;

cont_server_pos = 1; /* Visualizza 1 perchè ha letto il destino del bersaglio */

pthread_cond_broadcast(&cond_esito); /* Consente a tutti i thread di inviare il destino del bersaglio */

/* Attende che tutti abbiano inviato il destino del bersaglio ai Server */
while( cont_server_pos < cont_server_connessi)

pthread_cond_wait(&cond2,&mutex_esito);

if(id_vincitore == pos.id && pos.numero_missili>0)

    pos.numero_missili--;

write( sock, destino, sizeof(int)*2 ); /* Invia al Client il suo destino */

stampa_esito(dati_bersaglio[0], id_vincitore, max); /* Stampa l'esito del bersaglio */

/* Stampa lo stato */

sprintf(status,"\nNumero missili Torretta %d: %d\n",pos.id,pos.numero_missili);

write( 1, status, strlen(status) );

pthread_mutex_unlock(&mutex2); /* Rilascia il mutex */

pthread_mutex_unlock(&mutex_esito); /* Rilascio il mutex che conferma l'esito da parte di tutti */

close(sock); /* Chiude il socket di comunicazione con il Client */

pthread_exit(NULL);
}

```

```

/* Funzione che riceve le informazioni sui Server connessi */

void *lettura_rend(void *fd)
{
    int sock_rend = *(int*)fd, n, i, *sock;

    struct sockaddr_in ricevuto;

    char stampa_mess[100];

    pthread_t th;

    while( (n=read(sock_rend , &ricevuto, sizeof(struct sockaddr_in))) > 0 )
    {
        sprintf( stampa_mess , "\nServer %s connesso\n",inet_ntoa(ricevuto.sin_addr));

        sock = malloc(sizeof(int));

        *sock = socket(AF_INET, SOCK_STREAM, 0);

        /* Connessione ad un Server appena connesso */

        if (connect(*sock , (struct sockaddr *)&ricevuto , sizeof(ricevuto)) < 0)
        {
            perror("\nErrore connessione Server");

            continue;

        }

        pthread_mutex_lock(&mutex);

        /* Se è stato raggiunto il limite, si deve aumentare la dimensione della lista */

        if( cont_server_connessi == dim )
        {
            dim = dim*2; /* La dimensione viene raddoppiata */

            lista_candidati=(int**) realloc ( lista_candidati,dim * sizeof( int *) );

            for( i=(dim/2) ; i<dim; i++)

                lista_candidati[i] = malloc(sizeof(int)*2);

        }

        /* Thread creato per gestire la connessione con un Server appena connesso */

        if((pthread_create(&th ,NULL,(void*)gestisci_server,(void*)sock)) == -1 )
        {
            perror("\nErrore accetta_server");

            pthread_mutex_unlock(&mutex);

            continue;
        }
    }
}

```

```

    }

    else

    {

        if( n==sizeof(struct sockaddr_in))

        {

            write( 1, stampa_mess, strlen(stampa_mess) );

            cont_server_connessi++;

            pthread_mutex_unlock(&mutex);

        }

    }

    write( 1, "\nConnessione persa con il Server di Rendezvous\n", strlen("\nConnessione persa con il Server di
    Rendezvous\n") );

    close(sock_rend);

    pthread_exit(NULL);

}

/* Funzione che calcola l'esito di un bersaglio */

int esito(int x, int y, int r)

{

    if((sqrt((pow(x-pos.posizione[0],2)) + (pow(y-pos.posizione[1],2)))) <= r )

        return pos.numero_missili;

    return -1;

}

/* Funzione che visualizza l'esito del bersaglio */

void stampa_esito(int id_bersaglio, int id_server, int esito)

{

    char stampa_bersaglio[100];

    if( esito == -1)

    {

        sprintf( stampa_bersaglio, "\nBersaglio %d non raggiungibile da nessuna Torretta\n", id_bersaglio);

        write( 1, stampa_bersaglio, strlen(stampa_bersaglio) );

    }

    else if( esito == 0 )

    {

        sprintf( stampa_bersaglio, "\nBersaglio %d non abbattibile da nessuna Torretta attiva\n", id_bersaglio);

```



```
        write( 1, stampa_bersaglio, strlen(stampa_bersaglio) );  
    }  
    else  
    {  
        sprintf( stampa_bersaglio, "\nBersaglio %d abbattuto dalla Torretta %d\n", id_bersaglio, id_server);  
        write( 1, stampa_bersaglio, strlen(stampa_bersaglio) );  
    }  
}
```

```

/* Client - Bersaglio */

/* Inclusione librerie */

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/un.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <fcntl.h>

/* Prototipo */

void stampa_esito(int id_server, int esito); /* Funzione per la stampa dell'esito */

/* Parametri in input (1) ID (2) IP Server (3) Porta Server (4) X,Y */

int main(int argc, char **argv)

{

    int sock_server, parametri[3], esito[2];

    struct sockaddr_in server;

    char *server_ip = (char*)malloc(sizeof(100)), *server_port = (char*)malloc(sizeof(100)), *x, *y, id_serv[100];

    if( argc < 5 )

    {

        write( 1, "\nNon sono presenti tutti i parametri su riga di comando\n",

            strlen("\nNon sono presenti tutti i parametri su riga di comando\n") );

        exit(EXIT_FAILURE);

    }

    parametri[0] = atoi(argv[1]); /* Estrae ID missile da riga di comando */

    strcpy(server_ip, argv[2]); /* Estrae ip da riga di comando */

    strcpy(server_port, argv[3]); /* Estrae la porta da riga di comando */

    x = strtok(argv[4],",\n"); /* Estrae la coordinata x da riga di comando */

    y = strtok(NULL,",\n"); /* Estrae la coordinata y da riga di comando */

    parametri[1] = atoi(x);

    parametri[2] = atoi(y);

    /* Creazione socket per connessione al Client */

    if(( sock_server=socket(AF_INET, SOCK_STREAM, 0)) == -1 )

```

```

{
    perror("\nErrore socket");
    exit(EXIT_FAILURE);
}

/* Inizializzazione della struttura sockaddr_in */
server.sin_addr.s_addr = inet_addr(server_ip);
server.sin_family = AF_INET;
server.sin_port = htons(atoi(server_port));

/* Connessione al Server */
if (connect(sock_server , (struct sockaddr *)&server , sizeof(server)) < 0)
{
    perror("\nErrore connessione");
    exit(EXIT_FAILURE);
}

/* Invia i parametri (id, x, y) al Server */
write( sock_server, parametri, sizeof(int)*3 );

/* Riceve l'esito dal Server */
read( sock_server, esito, sizeof(int)*2);
sprintf(id_serv, "\nConnesso alla Torretta %d - %s\n", esito[0],server_ip);
write( 1, id_serv, strlen(id_serv) );

/* Visualizza l'esito del bersaglio */
stampa_esito(esito[0], esito[1]);

return 0;
}

void stampa_esito(int id_server, int esito)
{
    char stampa_bersaglio[100];

    if( esito == -1)
    {
        sprintf( stampa_bersaglio, "\nNon raggiungibile da nessuna Torretta\n");
        write( 1, stampa_bersaglio, strlen(stampa_bersaglio) );
    }
}

```

```
else if( esito == 0 )
{
    sprintf( stampa_bersaglio, "\nMissili della torretta %d insufficienti\n", id_server);
    write( 1, stampa_bersaglio, strlen(stampa_bersaglio) );
}
else
{
    sprintf( stampa_bersaglio, "\nColpito dalla Torretta %d\n", id_server);
    write( 1, stampa_bersaglio, strlen(stampa_bersaglio) );
}
}
```