Reti Di Calcolatori

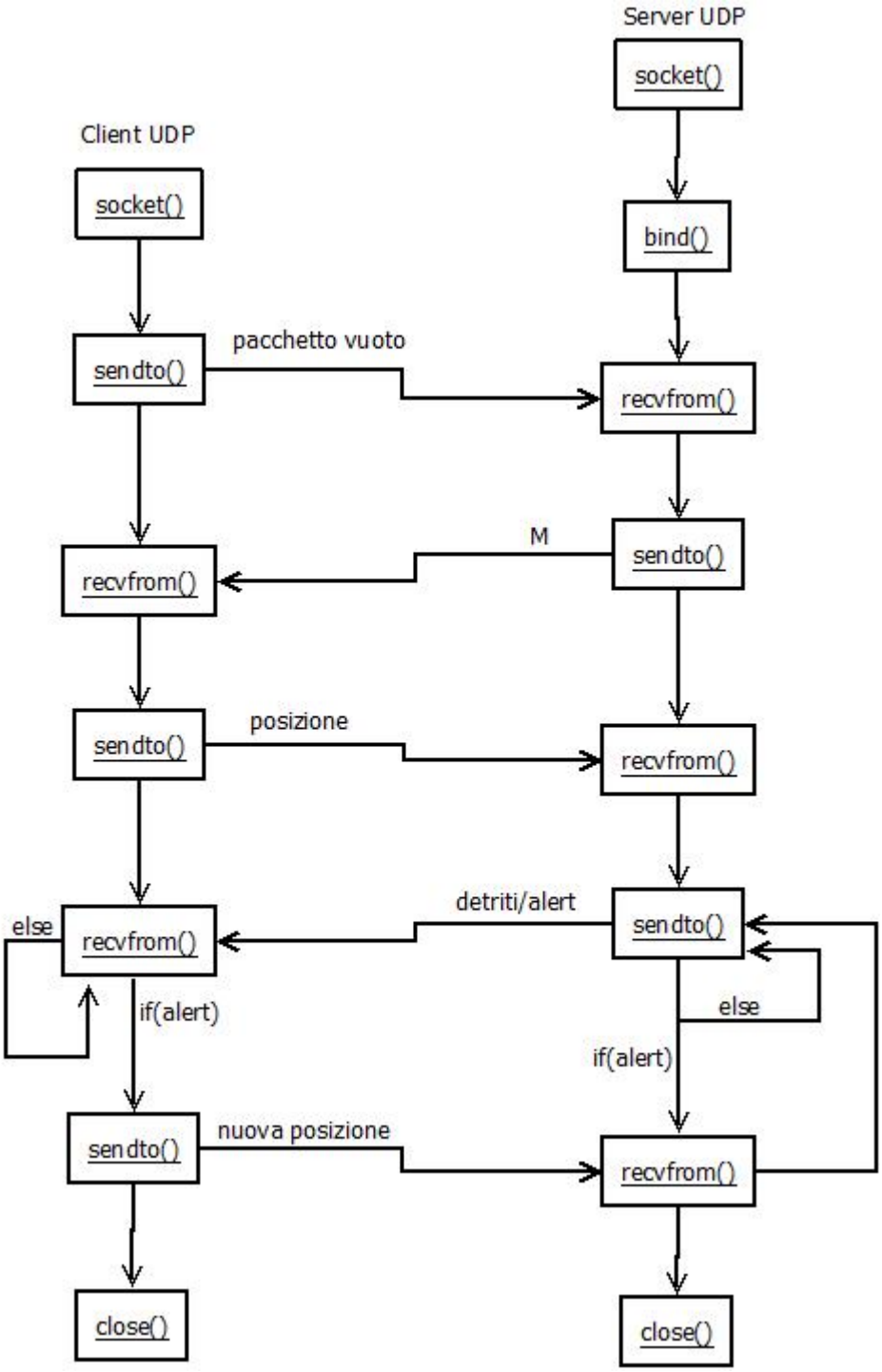
De Gaetano Giovanni 0124002431
Sparaco Asia Maria 0124002519

Descrizione del progetto

Una navicella spaziale deve evitare i detriti provenienti da una tempesta di meteoriti. La navicella (client) entra nel settore spaziale dei meteoriti (server) connettendosi in UDP. La tempesta di meteoriti genera in modo casuale n pacchetti UDP ogni 2 secondi che rappresentano i detriti in una griglia MxM. La navicella riceve un alert nel caso in cui il detrito spaziale si trovi sulla sua stessa posizione e può spostarsi prima dell'impatto. Ogni spostamento della navicella viene notificato al server il quale genera nuovi detriti nella direzione della navicella stessa.

Descrizione e schema dell'architettura

L'architettura del sistema si basa su un modello client-server. Il client comunica con il server utilizzando il protocollo UDP attraverso datagrammi inviati sulla rete.


Server UDP
socket()
Client UDP
socket()
bind()
sendto()
pacchetto vuoto
recvfrom()
recvfrom()
M
sendto()
sendto()
posizione
recvfrom()
else
recvfrom()
detriti/alert
sendto()
if(alert)
else
if(alert)
sendto()
nuova posizione
recvfrom()
close()
close()

Dettagli implementativi dei client/server

Il client e il server sono implementati in linguaggio C utilizzando le socket UDP

Client:

Librerie utilizzate

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/types.h>
```

Prototipi

```
void changePos();
```

Cambia la posizone della navicella controllando che non esca dalla griglia

```
void Sendto(int sockfd, const void *buf, size_t len, int flags,
const struct sockaddr *dest_addr, socklen_t addrlen);

void Recvfrom(int sockfd, void *buf, size_t len, int flags,
struct sockaddr *src_addr, socklen_t *addrlen);
```

Funzioni wrapper

Funzionamento

Creazione della socket UDP

```
 if((sfd = socket(AF_INET,SOCK_DGRAM,0)) < 0){
        perror("socket");
        exit(1);
    }
```

Configurazione dell'indirizzo del server

```
server.sin_family = AF_INET;
server.sin_port = htons(2309);

if(inet_pton(AF_INET,argv[1],&server.sin_addr) <= 0){
```

```
        perror("inet_pton");
        exit(1);
}
```

Server:

Librerie utilizzate

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/types.h>
```

Prototipi

```
void printGriglia(char **);
```

Data una matrice, ne stampa il contenuto

```
void generateDet(int [],int []);
```

Dati 2 array interi, che rappresentano rispettivamente la posizione della navicella e la posizione del detrito, posiziona casualmente quest'ultimo nell'intorno 3x3:

cell1	cell2	cell3
cell4	n	cell6
cell7	cell8	cell9

```
void Sendto(int sockfd, const void *buf, size_t len, int flags,
const struct sockaddr *dest_addr, socklen_t addrlen);

void Recvfrom(int sockfd, void *buf, size_t len, int flags,
struct sockaddr *src_addr, socklen_t *addrlen);
```

Funzioni wrapper

Funzionamento

Creazione della socket UDP

```
if((sfd = socket(AF_INET,SOCK_DGRAM,0)) < 0){
    perror("socket");
```

```
    exit(1);
    }
```

Configurazione del server

```
server.sin_family = AF_INET;
server.sin_port = htons(2309);
server.sin_addr.s_addr = htonl(INADDR_ANY);
```

Assegnazione indirizzo al socket

```
if(bind(sfd,(struct sockaddr*)&server,sizeof(server)) < 0){
    perror("bind");
    exit(1);
}
```

Parti rilevanti del codice sviluppato

Client:
Invio di un pacchetto UDP vuoto al server, al fine di catturare l'indirizzo del mittente

```
Sendto(sfd, buffer, sizeof(buffer), 0, (struct sockaddr*)&server,
sizeof(server));
```

Server:
Una volta ricevuto il pacchetto vuoto, invia la dimensione della griglia.

```
Recvfrom(sfd, buffer, sizeof(buffer), 0, (struct sockaddr*)&client, &len);
```

Il parametro M è definito tramite linea di comando

```
Sendto(sfd, &M, sizeof(M), 0, (struct sockaddr*)&client, sizeof(client));
```

Client:
Una volta ricevuta la dimensione della griglia, viene generata una posizione casuale per la navicella e viene inviata al server

```
Recvfrom(sfd, &M, sizeof(M), 0, (struct sockaddr*)&server, &len);
pos[0] = rand() % M;
pos[1] = rand() % M;
Sendto(sfd, pos, sizeof(pos), 0, (struct sockaddr*)&server, sizeof(server));
```

Server:
Ricevuta la posizione, inserisce la navicella nella griglia

```
Recvfrom(sfd, pos, sizeof(pos), 0, (struct sockaddr*)&client, &len);
griglia[pos[0]][pos[1]] = 'n';
```

Il server entra in ciclo infinito, nel quale vengono generati n detriti

```
for(int i=0;i<n;i++){
    // Se la navicella ha cambiato la sua posizione, i detriti vengono
    // nella direzione della navicella
    if(changePos){
        generateDet(pos, detrito);
    }
    else{// In caso contrario, vengono generati casualmente
        detrito[0] = rand() % M;
        detrito[1] = rand() % M;
    }
```

```
        griglia[detrito[0]][detrito[1]] = 'x';
        Sendto(sfd,detrito,sizeof(detrito),0,(struct sockaddr*)&client,
        sizeof(client));
        // controlla se il detrito sta per colpire la navicella
        if((detrito[0] == pos[0]) && (detrito[1] == pos[1])){
            crash = 1;
        }
}
```

Se il flag è impostato a 1, viene inviato un messaggio "alert" al client e si attende che la navicella invii la sua nuova posizione

```
if(crash){
        crash = 0;
        char alert[10] = "alert";
        Sendto(sfd,alert,strlen(alert),0,(struct sockaddr*)&client,
        sizeof(client));
        Recvfrom(sfd,pos,sizeof(pos),0,(struct sockaddr*)&client,&len);
        changePos = 1;
        griglia[pos[0]][pos[1]] = 'n';
}
```

Client:

La navicella entra in ciclo infinito nel quale riceve i pacchetti dal server: se sono detriti, li scarta; se riceve il messaggio di "alert" cambia posizione e la invia al server

```
while(1){
        Recvfrom(sfd,buffer,sizeof(buffer),0,
        (struct sockaddr*)&server,&len);
        if((strcmp(buffer,"alert")) == 0){
            changePos();
            printf("La navicella si sposta\n");
            Sendto(sfd,pos,sizeof(pos),0,(struct sockaddr*)&server,
            sizeof(server));
        }
}
```

Funzioni utilizzate:

```
void changePos(){
    int offsetRow;
    // Calcolo dell'offset in base alla posizione corrente
    if(pos[0] == 0){
    // Se la navicella si trova nella prima riga si crea un offset per
        offsetRow = rand() % 2; // tra 0 e 1;
    }
    else if(pos[0] == M-1){
    // Se la navicella si trova nell'ultima riga si crea un offset per
        offsetRow = (rand() % 2)-1; // tra -1 e 0
    }
    else{
        offsetRow = (rand() % 3) - 1; // tra -1,0 e 1
    }
    pos[0] += offsetRow;
    //Si controlla se "offsetRow" non sia uguale a 0
    //in modo tale di non avere la possibilit  che l'offset per le righ
    if(pos[1] == 0){//analogo al if precedente
        if(offsetRow == 0)
            pos[1] += 1;
        else
            pos[1] += rand() % 2; // tra 0 e 1
    }
    else if(pos[1] == M-1){
        if(offsetRow == 0)
            pos[1] += -1;
        else
            pos[1]+= (rand() % 2)-1; // tra -1 e 0
    }
    else{
        if(offsetRow == 0)
            pos[1] += 2 * (rand() % 2) -1; //tra -1 e 1
        else
            pos[1] += (rand() % 3) - 1; // tra -1,0 e 1
    }
}
```

```
void generateDet(int pos[],int detrito[]){
    int offsetRow,offsetCol;
    if(pos[0] == 0){
        offsetRow = rand() % 2; // tra 0 e 1;
    }
    else if(pos[0] == M-1){
        offsetRow = (rand() % 2)-1; // tra -1 e 0
    }
    else{
        offsetRow = (rand() % 3) - 1; // tra -1,0 e 1
    }

    if(pos[1] == 0){
       offsetCol = rand() % 2; // tra 0 e 1
    }
    else if(pos[1] == M-1){
       offsetCol = (rand() % 2)-1; // tra -1 e 0
    }
    else{
        offsetCol = (rand() % 3) - 1; // tra -1,0 e 1
    }
    detrito[0] = pos[0] + offsetRow;
    detrito[1] = pos[1] + offsetCol;
}
```

Manuale utente con le istruzioni su compilazione ed esecuzione

Compilazione:

1. gcc client.c -o client
2. gcc server.c -o server

Esecuzione:

1. ./server [size M]
2. ./client [Server's IP]