Documentatie ChessS

Scutaru Iulian-Tudor - Grupa B5

Universitatea "Alexandru Ioan Cuza", Iasi, secretariat@info.uaic.ro
https://www.info.uaic.ro

Abstract. Documentul dat contine documentatia proiectului ChessS, inclusiv tehnologiile folosite in cadrul acestuia si modul de prezentare si desfasurare a jocului de sah intre clientii serverului.

Keywords: TCP/IP · Retele de calculatoare · Threading.

1 Introducere

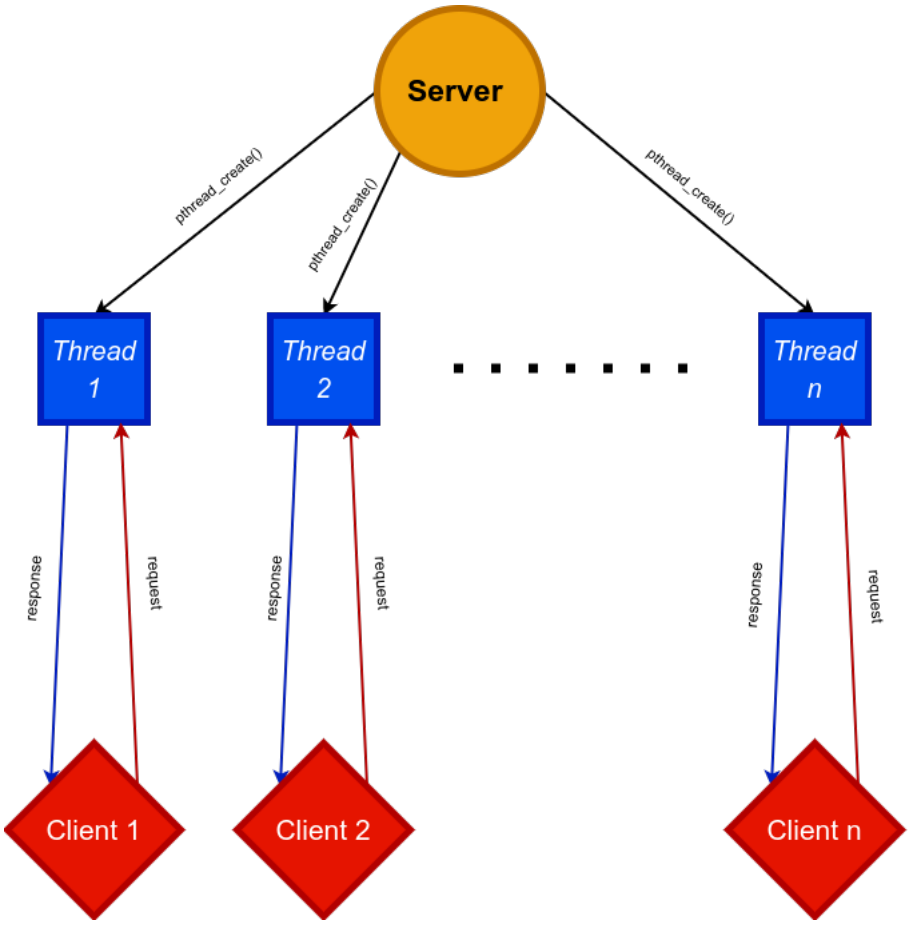
Proiectul **ChessS** presupune crearea unui server concurent ce admite conectarea a cel putin 2 utilizatori cu scopul de a asigura, observa si ulterior decide un castigator conform modului de desfasurare jocului, totul in cadrul terminalului, avand totusi o reprezentare pseudo-grafica, prin caractere speciale in forma unicode.

2 Tehnologii

In cadrul tehnilogiilor utilizate se numara protocolul **TCP** (Transfer Control Protocol) si **IP** (Internet Protocol), deoarece un joc online intre 2 clienti necesita o transmitere sigura, intacta si ordonata a datelor, pentru a tine cont de miscarile fiecarui jucator si cand este sau nu randul lui sa faca o miscare pe tabla de joc. Aceste lucruri ar fi incompatibile cu protocolul **UDP** (User Datagram Protocol), ce presupune conectare neorientata.
Proiectul utilizeaza si multi-threading, optand ca fiecare client sa fie deservit de catre un thread, in loc de un proces copil, evitand astfel alocarea unor noi zone de memorie separate. Prin consecinta, sunt prezente si **Mutexes** (Mutual Exclusion Lock) pentru a solutiona orice eventual race condition care apare.

3 Arhitectura

Proiectul presupune si existenta unor clienti care se conecteaza la server, acestia fiind de tipul Telnet pentru convenienta. Astfel serverul, prin apelul primitivei *pthread_create*, creaza cate un thread pentru fiecare client pentru a le deservi cererile, adica comenzile pentru mutari ale pieselor sau deconectarea de la server. Conform diagramei:


Server
pthread_create()
pthread_create()
pthread_create()
Thread 1
Thread 2
Thread n
response
request
response
request
response
request
Client 1
Client 2
Client n

4 Detalii de implementare

1) Initial, Serverul, dupa ce i s-a creat socket-ul propriul si i s-a atribuit o adresa si un port, v-a face **listen()** pentru a astepta cereri de conectare de la potentiali clienti.

```
int s_listen()
{
struct addrinfo hostinfo, *res;

int sock_fd;
int server_fd; //  fd-ul la care serverul asculta
int placeholder;

memset(&hostinfo, 0, sizeof(hostinfo));
hostinfo.ai_family = AF_UNSPEC;
hostinfo.ai_socktype = SOCK_STREAM;
hostinfo.ai_flags = AI_PASSIVE;

getaddrinfo(NULL, PORT, &hostinfo, &res);


    server_fd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
    if(server_fd < 0)
    {
      perror("Eroare la socket()\n");
      return errno;
    }
    placeholder = bind(server_fd, res->ai_addr, res->ai_addrlen);
    if(placeholder != 0)
    {
        perror("Eroare la bind \n");
        return -1 ;
    }

    placeholder = listen(server_fd, waitingList);
    if(placeholder < 0)
    {
      perror("Eroare la listen()\n");
      return errno;
    }


return server_fd;

}
```

2) In momentul in care un client incearca sa se conecteaza sub forma *telnet [adresa] [port]* , serverul va executa functia *conn_server* care apeleaza primitiva accept pentru stabilirea conexiunii, urmata de primitiva *getpeername* pentru a putea arata in terminal si adresa si portul de la care s-a conectat clinetul, prin conversiile de rigoare *ntohs* si *inet_ntop*.

```
int conn_server(int server_fd)
{
    char ipstr[INET_ADDRSTRLEN];
    int port;


    int new_sd;
    struct sockaddr_storage remote_info ;
    socklen_t addr_size;

    addr_size = sizeof(addr_size);
    new_sd = accept(server_fd, (struct sockaddr *) &remote_info, &addr_size);
    if (new_sd < 0)
    {
      perror("Eroare la accept()\n");
      return errno;
    }
    return new_sd;

}
```

3) Functia anterioara se realizeaza in cadrul unui *while*, intru-cat acesta continua sa primeasca cereri de conectarea, insa accepta pana la un anumit numar maxim de descriptori, ca un mecanism de protectie, intr-ucat foloseste multithreading, iar un numar imens de clienti ar afecta performanta. De mentionat ca in momentul adaugarii descriptorului noului client, se realizeaza un blocaj pe mutex, pentru a se asigura ca noul descriptor este adaugat doar o data, si ulterior, dupa deblocare, se creaza un thread nou pentru clientul respectiv.

```
int main()
{

    board.resize(8);
    for (int i = 0; i < 8; i++)
        board[i].resize(8);

     startPhase(); // Umplem tabela cu piese

    cout << "Serverul este on" << endl ;

    int server_fd = s_listen() ;
    if (server_fd == -1)
    {
        cout << "Eroare!" ;
        return 1 ;
    }


    string whitePlayer;
    string blackPLayer;

    string options \\comenzile valide

    pthread_t threads[maxFD];

    FD_ZERO(&stare);

    while(1)
    {
        int readFD;
        void *arg;

        readFD = conn_server(server_fd);

        if (readFD >=0)
        {
```

```
            write_to_client(readFD,"\nAveti␣descriptorul:
                ␣");
            write_to_client(readFD, to_string(readFD));
            if (readFD > maxFD)
            {
                write_to_client(readFD, "\nPrea␣multi␣
                    clienti␣sunt␣deja␣in␣asteptare.\n");
                close(readFD);
                continue;
            }

            pthread_mutex_lock(&mlock);

            FD_SET(readFD, &stare);
            pthread_mutex_unlock(&mlock);

            pthread_create(&threads[readFD], NULL,
                read_from_client, (void *) (intptr_t)
                readFD);
        }
        if(readFD<6)
        {
            write_to_client(readFD, options);
            if (readFD=4)
                write_to_client(readFD, whitePlayer);

            if (readFD=5)
                write_to_client(readFD, blackPLayer);

        }
        else
        {
            write_to_client(readFD, "\nUn␣joc␣este␣deja␣
                in␣desfasurare.␣Va␣rugam␣sa␣asteptati␣
                finalizarea␣lui.\n");
        }

    }
    return 0;
}
```

4) Prin intermediul a 2 functii de scrierei spre client si citire de la client, se realizeaza transferul de date de la client la server, si in sens invers, ceea ce asigura si comunicarea client-client, in care serverul este punctul de legatura.

```
int write_to_client(int fd, string date)
{
    printf("checkpoint write to client");
    int memorie;
    memorie = send(fd, date.c_str(), strlen(date.c_str()),0);

    return 0;
}

void *read_from_client(void *argument)
{
    int readFD;
    int writeFD;

    char mesaj[maxlength];
    int mesajLength;

    readFD = static_cast<int>(reinterpret_cast<intptr_t>(
        argument));
    while (1)
    {
        mesajLength=read(readFD, mesaj, sizeof(mesaj));
        if (mesajLength <1)
        {
            cout << "S-a deconectat clientul, deci se
                elibereaza descriptorul. " << readFD <<
                endl;
            pthread_mutex_lock(&mlock);
            FD_CLR(readFD, &stare);
            pthread_mutex_unlock(&mlock);
            close(readFD);
            pthread_exit(NULL);
        }

        pthread_mutex_lock(&mlock);

        for (writeFD=4; writeFD<maxFD; writeFD++)
        {
            if (FD_ISSET(writeFD, &stare) && (readFD!=
                writeFD))
            {
                write_to_client(writeFD, mesaj);
```

```
            }
        }

        pthread_mutex_unlock(&mlock);

        if((readFD<6) && (readFD>3))
        {
            jucator(mesaj,readFD);
        }
        else write_to_client(readFD,"Ne pare rau. Un meci
            este deja in desfasurare. Va rugam sa
            asteptati.");
    }

    return NULL;
}
```

5) Odata stabilita conexiunea, orice jucator va primi un mesaj cu optiunile pe care le are la indemana:

print - printeaza in terminal tabla de saj, piesele si patratele albe si negre fiind reprezentate prin coduri UNICODE corespunzatoare.

```
void startPhase()
{

    pthread_mutex_lock(&mutex_table);

    captures.clear();

    for (int i=0; i<8; i++)
    {
        for (int j=0; j<8; j++)
        {
            board[i][j] = "-"; // consideram - ca fiind
                spatiu gol.
        }
    }


 //Piese albe
    for (int i=0; i<8; i++)
    {
        board[1][i] = white_p;
    }
    board[0][0]= white_r;
    board[0][7]= white_r;
    board[0][1]= white_k;
    board[0][6]= white_k;
    board[0][2]= white_b;
    board[0][5]= white_b;
    board[0][3]= white_q;
    board[0][4]= white_king;


 //Piese Negre
    for (int i=0; i<8; i++)
    {
        board[6][i] = black_p;
    }
    board[7][0]= black_r;
    board[7][7]= black_r;
```

```
    board[7][1]= black_k;
    board[7][6]= black_k;
    board[7][2]= black_b;
    board[7][5]= black_b;
    board[7][3]= black_q;
    board[7][4]= black_king;

    pthread_mutex_unlock(&mutex_table);

}


void showTable(int client) // va afisa in terminalul
    fiecarui client o tabla de joc
{
    char coordonate_numerice;

    string coordonate= culoare_coordonate + "  A  B  C  D
        E  F  G  H\n";
    write_to_client(client, "\n" + coordonate);

    for (int j=7; j>=0; j--)
    {
        coordonate_numerice= j + 49; // int to char
        string coordonate_numerice_str="";
        coordonate_numerice_str += coordonate_numerice;
            //char to string
        write_to_client(client, coordonate_numerice_str);

        for (int i=0; i<8; i++)
        {
            if (board[i][j]=="-")
            {
                if (((j % 2 == 0) && (i % 2 == 0)) || ((j
                    % 2 == 1) && (i % 2 == 1))) //
                    verifica daca patratul este alb
                {
                    board[i][j]=whiteSquare;
                }
                else
                    board[i][j]=blackSquare;

            }
```

```
            write_to_client(client, " " + board[i][j] + "
                ");
        }

        write_to_client(client, culoare_coordonate +
            coordonate_numerice_str + "\n");

    }

    write_to_client(client, coordonate + "\n\n");

    string placeholder="Piese capturate: ";
    for (nr_capturi = captures.begin(); nr_capturi!=
        captures.end(); ++nr_capturi)
        placeholder = placeholder + *nr_capturi;

    write_to_client(client, placeholder + culoare_text +
        "\n");

}
```

quit - optiunea jucatorului de a se deconecta de la server, prin urmare incheierea meciului.

```
int quit(int client)
{
    write_to_client(client, "Ati fost deconectat.\n");
    close(client);
    return 0;
}
```

reset - optiunea de a reseta jocul, adica readucerea tablei de joc la starea initiala, in care inca nu s-a realizat nici o mutare.

```
void resetTable()
{
    startPhase();
    pasi=0;
    for (int writeFD =4; writeFD<maxFD; writeFD++)
        {
            if(FD_ISSET(writeFD, &stare))
            {
                showTable(writeFD);
            }
        }
}
```

Input de forma *a5a6* - coordonatele pentru efectuarea unei mutari, piesa de pe pozitia a5 se muta pe pozitia a6, iar serverul verifica daca a fost capturata o piesa a adversarului, eventual daca a fost capturat regele, deci terminand jocul.

```
void jucator(char * input, int client)
{
    char p, q, r;

    if (input[0] == 'p' || input[0]=='P') showTable(
        client);

    if (input[0] == 'r' || input[0]=='R') resetTable();

    if (input[0] == 'q' || input[0]=='Q')  quit(client);

    if (input[0]>='A' &&
        input[0]<='H' &&
        input[1]>='1' &&
        input[1]<='8' &&
        input[2]>='A' &&
        input[2]<='H' &&
        input[3]>='1' &&
        input[3]<='8')
        {
            if(client==4)
            {
                if (valid_step(input,client) && pasi
                    %2==0)
                {
                    step(input);
                    pasi++;
                    verifyWinner(client);
                    write_to_client(client, "\n Mutare: "
                        );
                    write_to_client(client, input);
                    write_to_client(client, "\n");

                }
                else
                {
                    if (valid_step(input,client) && pasi
                        %2==1)
                        write_to_client(client, "Este
                            randul oponentului de a muta.
                            \n");
```

```
                    if (valid_step(input,client)==false
                        && pasi%2==0)
                        write_to_client(client, "Miscarea
                            nu corespunde regulamentului
                            SAU ati incercat sa mutati
                            piesa adversarului.  \n");
                }
            }


            if(client==5)
            {
                if (valid_step(input,client) && pasi
                    %2==1)
                {
                    step(input);
                    pasi++;
                    verifyWinner(client);
                    write_to_client(client, "\nMutare: ")
                        ;
                    write_to_client(client, input);
                    write_to_client(client, "\n");

                }
                else
                {
                    if (valid_step(input,client) && pasi
                        %2==1)
                        write_to_client(client, "Este
                            randul oponentului de a muta.
                            \n");

                    if (valid_step(input,client)==false
                        && pasi%2==0)
                        write_to_client(client, "Miscarea
                            nu corespunde regulamentului
                            SAU ati incercat sa mutati
                            piesa adversarului. \n");
                }
            }



        }
}
```

Functii ce iau in calcul validitatea miscarilor pe tabla:

```
bool valid_queen(char * move)
{
    bool realizare = false;


    int a, b , c ,d ;

    a = move[0]-65; //coloana
    b = move[1]-49; //linie
    c = move[2]-65; //coloana
    d = move[3]-49; //linie

    if (b == d || a == c || abs(d - b) == abs(c - a))
       realizare=true;
    return realizare;

}

bool valid_king(char * move)
{
    bool realizare = false;


    int a, b , c ,d ;

    a = move[0]-65; //coloana
    b = move[1]-49; //linie
    c = move[2]-65; //coloana
    d = move[3]-49; //linie

    if (abs(d-b)==2 <=1 && abs(c-a)<=1) realizare=true;
    return realizare;

}

bool valid_knight(char * move)
{
    bool realizare = false;


    int a, b , c ,d ;

    a = move[0]-65; //coloana
```

```
    b = move[1]-49; //linie
    c = move[2]-65; //coloana
    d = move[3]-49; //linie

    if (abs(d-b)==2 && abs(c-a)==1 || abs(d-b)==1 && abs(
        c-a)==2) realizare=true;
    return realizare;

}

bool valid_bishop(char * move)
{
    bool realizare = false;


    int a, b , c ,d ;

    a = move[0]-65; //coloana
    b = move[1]-49; //linie
    c = move[2]-65; //coloana
    d = move[3]-49; //linie

    if (abs(d-b)==abs(c-a)) realizare=true;

    return realizare;

}

bool valid_rook(char * move)
{
    bool realizare = false;


    int a, b , c ,d ;

    a = move[0]-65; //coloana
    b = move[1]-49; //linie
    c = move[2]-65; //coloana
    d = move[3]-49; //linie

    if (b==d || a==c) realizare=true;
    return realizare;

}
```

```
bool valid_pawn(char * move)
{
    bool realizare = false;


    int a, b , c ,d ;

    a = move[0]-65; //coloana
    b = move[1]-49; //linie
    c = move[2]-65; //coloana
    d = move[3]-49; //linie

    if (a==c) realizare=false;

    if (d=b && c==a+1 || d==b && c==a-1) realizare = true
        ;

    if ((a==6 || a==1) && d==b && c==a+2) realizare=true;

    if (abs(d-a)==1 && c==a+1 || abs(d-b)==1 && c==a-1)
        realizare=true;

    return realizare;

}

bool valid_step(char * move, int client)
{
    bool realizare = true;


    int a, b , c ,d ;

    a = move[0]-65;
    b = move[1]-49;
    c = move[2]-65;
    d = move[3]-49;

    if ((board[a][b] == blackSquare) && (board[a][b] ==
        whiteSquare))
    {
        realizare=false;
    }
    else if (client=4)
    {
```

```
        if(board[a][b]==white_b && valid_bishop(move)||
                board[a][b]==white_k && valid_knight(move)
                    ||
                board[a][b]==white_p && valid_pawn(move)||
                board[a][b]==white_q && valid_queen(move)
                    ||
                board[a][b]==white_r && valid_rook(move)||
                board[a][b]==white_king && valid_king(move
                    ))
                {
                  realizare=true;
                }
                else realizare=false;
    }
    else if (client=5)
    {
        if(board[a][b]==black_b && valid_bishop(move) ||
                board[a][b]==black_k && valid_knight(move)
                    ||
                board[a][b]==black_p && valid_pawn(move)||
                board[a][b]==black_q && valid_queen(move)
                    ||
                board[a][b]==black_r && valid_rook(move)||
                board[a][b]==black_king && valid_king(move
                    ))
                {
                  realizare=true;
                }
                else realizare=false;
    }

    return realizare;
}
```

Functii pentru executarea unei mutari si verificarea castigatorului:

```
 void checkWinner(int client)
{
    bool white_king_alive=false;
    bool black_king_alive=false;

    for (int writeFD =4; writeFD<maxFD; writeFD++)
    {
        if (FD_ISSET(writeFD, &stare))
        {
            showTable(writeFD);
            for (int i=0; i<8; i++)
            {
                for (int j=0; j<8; j++)
                {
                    if(board[i][j]==white_king)
                        white_king_alive=true;
                    else if (board[i][j]==black_king)
                        black_king_alive=true;
                }
            }

            if (white_king_alive==true &&
                black_king_alive==false)
            {
                write_to_client(client, "Felicitari, ati
                    castigat cu Alb!");
                close(client);
            }
            else if (black_king_alive==true &&
                white_king_alive==false)
            {
                write_to_client(client, "Felicitari, ati
                    castigat cu Negru!");
                close(client);
            }

        }
    }
}
```

```
void step(char * move)
{
    bool realizare = false;

    int a, b , c ,d ;

    a = move[0]-65;
    b = move[1]-49;
    c = move[2]-65;
    d = move[3]-49;

    if (!(board[a][b] == blackSquare) && !(board[a][b] ==
        whiteSquare))
    {
        realizare = true;
        pthread_mutex_lock(&mutex_table);

        if (board[c][d] != "-" &&
            board[c][d] != blackSquare &&
            board[c][d] != whiteSquare)
            {
                captures.insert(captures.begin(), board[c
                    ][d]);
            }

        board[c][d]=board[a][b];
        board[a][b]="-";
        pthread_mutex_unlock(&mutex_table);

    }

}
```

5 Concluzii

Eventuale imbunatariti ar fi un mecanism in care clientii fortati sa astepte terminarea unei partide de sah in desfasurare ar putea in schimb sa o urmareasca ca si spectator. De asemenea. Serverul ar putea fi proiectat sa monitorizeze si alte tipuri de jocuri asemanatoare cu sahul, precum jocul de dame, care utilizeaza aceiasi tabla de joc, cu acelasi sistem de coordonate. Orice alta variatie de sah, precum RANDOM PIECE POSITION chess ar putea fi implementat, cu optiunea ca la conectare, clientul sa aleaga ce variatie doreste.

References

1. Georgiana Calancea, RC-home
https://profs.info.uaic.ro/ gcalancea/rc-home.html
2. Retele de calculatoare
https://profs.info.uaic.ro/ computernetworks/
3. Stack Overflow
https://stackoverflow.com/questions/5947742/how-to-change-the-output-color-of-echo-in-linux
4. GNU Chess manual
https://www.gnu.org/software/chess/manual/
5. The Unicode Standard
https://www.unicode.org/charts/PDF/U2600.pdf
6. CMU School of Computer Science
https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html
7. Diagram Software and Flowchart Maker
https://www.diagrams.net