ChessS

Gălățianu Mihai

December 2023

1 Introducere

Am decis să realizez documentația proiectului ChessS.Acest proiect are ca scop implementarea unei aplicații de tip *client/server* unde fiecare jucător va putea iniția sau se va putea alătura unei partide de șah, beneficiind de o tablă de joc virtuală și de un sistem care supraveghează desfășurarea jocului și determină momentul încheierii partidei, anunțând câștigătorul la finalul acesteia.

2 Tehnologii utilizate

2.1 Server TCP concurent

Aplicația folosește protocolul TCP (Transmission Control Protocol) pentru a gestiona comunicarea între server și clienți. Protocolul TCP este ales pentru fiabilitatea sa în asigurarea unei comunicări ordonate și corecte deoarece acesta se caracterizeaza prin livrarea mesajelor în ordinea corectă și verificarea dacă informațiile au ajuns complet și fără erori. TCP contribuie la eficiența aplicației prin asigurarea unei comunicări full-duplex, permitând transmiterea de date bidirecțională și simultană între client și server. Această caracteristică este crucială pentru un joc de șah interactiv, unde jucătorii pot face mutări, pot primi actualizări de stare și pot comunica în timp real. Mai mult, garanția livrării pachetelor și gestionarea eficientă a congestiei rețelei sunt avantaje cheie ale folosirii TCP, reducând riscul de întârzieri sau pierderi de date care ar putea afecta desfășurarea partidei.

2.2 Socket-uri

Socket-urile sunt puncte finale pentru trimiterea și primirea datelor între dispozitive pe o rețea, funcționând ca un canal de comunicare între programe sau procese care rulează pe computere diferite în cadrul aceleiași rețele sau în rețele diferite. În cazul acestui proiect, socket-urile permit serverului și clienților (jucătorilor) să trimită și să primească mișcările de șah, actualizările stării jocului și alte informații relevante. Folosind protocolul TCP/IP, socket-urile asigură o conexiune stabilă și ordonată, identificată unic prin adresa IP

și numărul de port al fiecărui client. Această tehnologie este esențială pentru interacțiunea în timp real în jocuri, oferind o bază solidă pentru comunicarea eficientă și fiabilă în aplicația de șah.

2.3 Thread-uri

Thread-urile pot fi văzute ca cea mai mică secțiune de cod ce poate fi administrată, programată de către sistemul de operare;. În contextul unei aplicații de șah server-client, utilizarea thread-urilor permite serverului să gestioneze multiple partide de șah în același timp. Fiecare conexiune cu un client poate fi tratată de un thread separat, astfel încât serverul să poată servi mai mulți jucători fără a aștepta finalizarea unei partide pentru a începe alta. Acest lucru asigură o experiență de joc fluidă și responsivă pentru toți utilizatorii.

3 Arhitectură aplicației

Aplicația de șah este structurată într-un model client-server, unde serverul poate gestiona multiple conexiuni de la diferiți utilizatori simultan. Comunicarea dintre client și server se realizează prin intermediul socket-urilor pentru a asigura un flux coerent și eficient de informații.

3.1 Concepte implicate

- **Server**
  Serverul în aplicația noastră de șah reprezintă nucleul funcționalității, gestionând logica jocului, validarea mutărilor, starea partidelor și interacțiunile dintre jucători. El acceptă cereri de la clienți și le servește, oferindu-le posibilitatea de a începe, juca și finaliza partidele de șah.

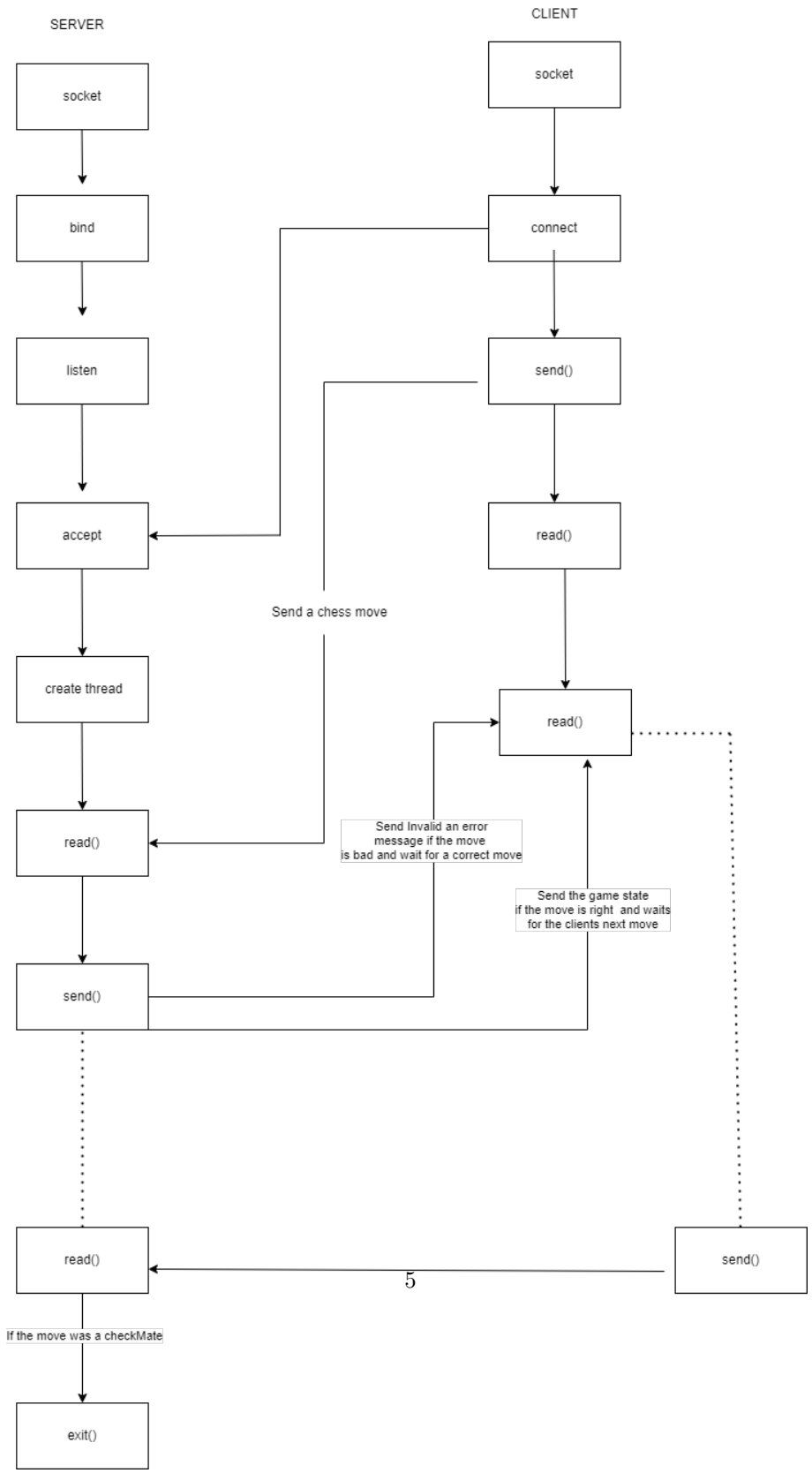
- **Client**
  Clientul este interfața prin care utilizatorii interacționează cu jocul de șah. Inițiază comunicarea cu serverul pentru a se alătura jocului, executa mutări și primește actualizări ale stării jocului.

- **Adresa IP**
  Adresa IP este folosită pentru a identifica dispozitivele conectate la rețea, inclusiv serverul și clienții de șah. Prin intermediul adreselor IP, serverul poate determina sursa și destinația datelor, asigurându-se că informațiile despre joc ajung corect la fiecare jucător.

- **Portul**
  Portul reprezintă partea dintr-o adresă de rețea de calculatoare care determină atribuirea conexiunilor TCP/UDP și a pachetelor de date către serverele și programele client de către sistemele de operare.Portul este reprezentat printr-un număr pe 16 biți și va fi atașat conexiunii utilizate de aplicație.Porturile cu numere între 0 și 1023 sunt rezervate unor diferite procese de sistem.

- **Interfata**
  Interfața reprezintă punctul de interacțiune între utilizator și aplicație. Pentru clienți, interfața este adesea un GUI (Graphical User Interface) sau un CLI (Command Line Interface), unde jucătorii pot vedea tabla de șah și pot face mutări.

3.2 Diagrama aplicație



4 Detalii de implementare

Comunicare dintre client și server se realizează prin socket-uri.Acest lucru avantajează scrierea programului deoarece nu mai trebuie să închidem capetele canalului de transmisie.

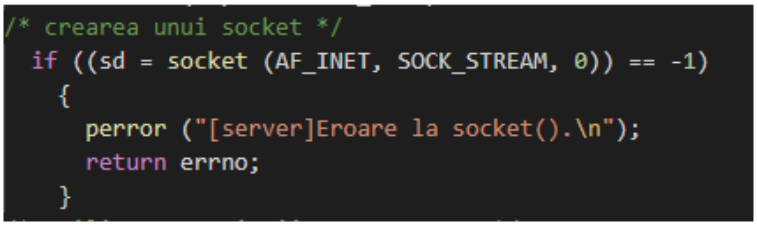


Figure 2:

Configurarea inițială a socket-ului server în aplicația de șah este un pas crucial pentru a asigura comunicarea eficientă între client și server. Mai jos este prezentat codul pentru inițializarea socket-ului și pregătirea acestuia pentru a accepta conexiuni.

Figure 3:

Comanda care are rolul de a asocia adresa IP și portul serverului cu socket-ul, pregătindu-l să primească conexiuni.

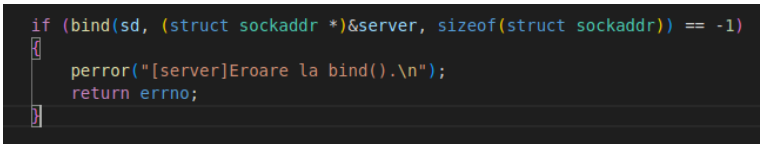


Figure 4:

După atașarea portului, serverul va trebui să aștepte viitoare conexiuni de la diverși clienți și să le rezolve cererile. Pentru aceasta, se utilizează apelul listen() urmat de accept().

```
if (listen(sd, 5) == -1)
{
    perror("[server]Eroare la listen().\n");
    return errno;
}
```

Figure 5:

Pentru a gestiona starea jocului și a comunica eficient între client și server, aplicația de șah folosește mai multe structuri de date, fiecare având un rol specific în implementarea logicii jocului.

- **ChessPiece**
  Această structură reprezintă o piesă de șah individuală. Este definită prin două câmpuri: type, care stochează tipul piesei (rege, regină, turn, nebun, cal sau pion) și color, care indică culoarea piesei (alb sau negru).

- **Move**
  Structura Move este utilizată pentru a reprezenta o mutare în joc. Aceasta include coordonatele de start (startX, startY) și de sfârșit (endX, endY) ale mutării pe tablă.

- **ChessBoard**
  ChessBoard este structura care modelează tabla de șah. Aceasta conține o matrice bidimensională de ChessPiece pentru a reprezenta poziția pieselor pe tablă și un întreg currentPlayer, care indică jucătorul curent ce trebuie să mute.

- **GameSession**
  Structura GameSession reprezintă o sesiune de joc între doi jucători. Include identificatori pentru primul și al doilea jucător (player1, player2), o instanță a structurii ChessBoard pentru a stoca starea actuală a jocului și un întreg pentru currentPlayer care indică al cărui jucător este rândul să mute.

- **thData:**
  Structura thData este destinată a fi folosită pentru gestionarea datelor specifice fiecărui thread. Aceasta include un identificator pentru thread (idThread), un descriptor al clientului (cl) și un pointer către o sesiune de joc (gameSessionId), permițând astfel thread-ului să gestioneze sesiunile de joc ale clientului.

```c
typedef struct
{
    char type;
    int color;
} ChessPiece;

typedef struct
{
    int startX;
    int startY;
    int endX;
    int endY;
    char promPiece;
} Move;

typedef struct
{
    ChessPiece board[BOARD_SIZE][BOARD_SIZE];
    int currentPlayer;
} ChessBoard;

typedef struct
{
    int player1;
    int player2;
    ChessBoard game;
    int currentPlayer;


} GameSession;

typedef struct thData
{
    int idThread;
    int cl;
    GameSession *gameSessionId;
} thData;
```

Figure 6:

4.1 Implementare functii

- **Functia initializeGame**
  Funcția initializeGame este responsabilă pentru inițializarea tablei de șah. În cadrul acestei funcții, fiecare celulă a tablei de șah este inițial setată ca fiind goală (ii atribuim tipul L si culoarea 2).După aceea, funcția plasează piesele conform regulilor sahului, setandule culoarea si tipul piesei (0 pentru alb si 1 pentru negru).

- **Functia validateMove**
  Funcția validateMove examinează daca o mutare de sah este valida, aceasta verifica pozițiile inițiale și finale să fie în limitele tablei și că piesa mutată aparține jucătorului curent. Dacă destinația este ocupată de o piesă de aceeași culoare, mutarea e respinsă. Funcția testează apoi dacă mutarea pune regele celui care a mutat este în continuare în șah, caz în care mutarea este, de asemenea, invalidă. Dacă niciuna dintre aceste condiții nu este îndeplinită, funcția validează mutarea specifică pentru fiecăre tip de piesă.

- **Functiile de forma validatePiesaMove**
  Functiile de tipul "validatePiesaMove" sunt o generalizare pentru validarea mutarilor pentru diferite tipuri de piese de sah. Ea urmeaza un principiu similar cu functia "validatePawnMove", ajustand regulile in functie de capacitatea unica de miscare a fiecarei piese. In functie de tipul piesei (pion, turn, nebun, cal, regina sau rege), functia examineaza daca mutarea respecta regulile de miscare specifice fiecarei piese: de exemplu, inainte sau pe diagonala pentru pioni, orizontal sau vertical pentru turnuri, forma de "L" pentru cai, diagonale pentru nebuni, o combinatie pentru regina, si o singura casuta in orice directie pentru rege. Functia va returna un raspuns afirmativ daca mutarea este in conformitate cu regulile de miscare si nu incalca nicio regula (cum ar fi sa lase regele in sah); in caz contrar, va returna un raspuns negativ.

- **Functia parseMove**
  Această funcție este o funcție ajutătoare care ne ajută să convertim mutarile de forma e2e4 in coordonate pe care le putem folosi.

- **Functia updateGameState**
  Functia "updateGameState" actualizeaza starea tablei de sah dupa o mutare valida. Initial, functia copiaza piesa mutata din pozitia de start in pozitia de sfarsit pe tabla de joc. Apoi, ea "curata" locatia initiala a piesei, marcand-o ca fiind goala. De asemenea, functia schimba jucatorul curent, alternand intre jucatorul alb (0) si negru (1). Prin aceste actiuni, functia reflecta efectul unei mutari complete asupra starii jocului.

- **Functia serializeGameSession**

Functia "serializeGameSession" converteste starea curenta a unei sesiuni de joc de sah intr-un sir de caractere pentru a facilita procesul de comunicare dintre server si client.

- **Functiile de tipul attackedByPiesa**
  Functiile de tipul "attackedByPiesa" (de exemplu, "attackedByPawn," "attackedByKnight," "attackedByQueen," etc.) au rolul de a verifica daca regele este amenintat de un anumit tip de piesa adversa pe tabla de sah. Aceste functii determina daca o piesa specificata (pion, cal, regina, etc.) pune in pericol siguranta regelui.

- **Functiile de tipul canPiesaBlockCheck**
  Functia "canPieceBlockCheck" are rolul de a verifica daca orice piesa de pe tabla poate bloca un sah dat regelui sau poate captura piesa care da sah. Aceasta functie parcurge toate piesele de pe tabla de sah si, pentru fiecare piesa, analizeaza miscarile posibile conform regulilor specifice fiecarei piese. Pentru fiecare piesa, functia simuleaza mutarea pe tabla si verifica daca aceasta mutare elimina amenintarea asupra regelui. Daca gaseste o mutare care protejeaza regele fie prin blocarea ataculuI, fie prin capturarea piesei atacatoare, functia returneaza 1, indicand ca sahul poate fi blocat sau prevenit. Daca nicio piesa nu poate bloca sau preveni sahul, functia returneaza 0, indicand ca regele ramane in sah si jucatorul trebuie sa gaseasca o alta solutie pentru a-l proteja.

- **Functia treat**

  Funcția treat gestionează interacțiunea cu un client în cadrul unui joc de șah . Funcția se execută într-un thread separat pentru fiecare client și are rolul de a procesa mutările jucătorului, să actualizeze starea jocului și să comunice înapoi rezultatele în urma acestora.

- **Functia main**
  Functia "main" initializeaza si ruleaza serverul pentru un joc de sah. Acesta creeaza un socket si il configureaza pentru a accepta conexiuni. Apoi, intr-o bucla continua, asteapta si accepta clientii. Cand doi clienti se conecteaza, initializeaza o sesiune de joc noua, atribuie fiecarui client rolul de jucator si trimite starea initiala a jocului. Pentru fiecare client, creeaza un thread care va gestiona comunicarea si logica jocului prin functia "treat".

```
53  static void *treat(void *);
54  void initializeGame(ChessBoard *);
55  void printBoard(ChessBoard *);
56  int validateMove(ChessBoard *board, Move move, int playerColor);
57  int validatePawnMove(ChessBoard *board, Move move, int color);
58  int validateRookMove(ChessBoard *board, Move move, int color);
59  int validateKnightMove(ChessBoard *board, Move move, int color);
60  int validateBishopMove(ChessBoard *board, Move move, int color);
61  int validateQueenMove(ChessBoard *board, Move move, int color);
62  int validateKingMove(ChessBoard *board, Move move, int color);
63
64  int parseMove(char *input, Move *move);
65  void updateGameState(ChessBoard *game, Move move);
66  char *serializeGameSession(const GameSession *session);
67  int attackedByPawn(ChessBoard *board, int xKing, int yKing);
68  int attackedByBishop(ChessBoard *board, int xKing, int yKing);
69  int attackedByKnight(ChessBoard *board, int xKing, int yKing);
70  int attackedByQueen(ChessBoard *board, int xKing, int yKing);
71  int attackedByRook(ChessBoard *board, int xKing, int yKing);
72  int canPawnBlockCheck(GameSession *gameSession, int xKing, int yKing, int opponentColor);
73  int canQueenBlockCheck(GameSession *gameSession, int xKing, int yKing, int opponentColor);
74  int canKnightBlockCheck(GameSession *gameSession, int xKing, int yKing, int opponentColor)
75  int canRookBlockCheck(GameSession *gameSession, int xKing, int yKing, int opponentColor);
76  int canBishopBlockCheck(GameSession *gameSession, int xKing, int yKing, int opponentColor)
77  int canKingEscapeCheck(GameSession *gameSession, int xKing, int yKing, int opponentColor);
78
```

Figure 7:

4.2 Scenarii de utilizare

Conectarea și începerea jocului:

- Jucătorul 1 se conectează la server și așteaptă un adversar.
- Jucătorul 2 se conectează și este asociat automat cu Jucătorul 1 pentru a începe o nouă sesiune de joc.
- Amândoi jucătorii primesc starea inițială a tablei de șah și pot incepe jocul.

Desfășurarea unei partide de șah:

- Jucătorul 1 trimite o mutare validă.
- Serverul actualizează tabla de șah și trimite starea actualizată ambilor jucători.

- Jucătorul 2 analizează tabla și trimite următoarea mutare.
- Serverul verifică șah, șah-mat sau doar mat, actualizează starea jocului și comunică orice situații speciale jucătorilor.

Gestionarea mutărilor invalide:

- Un jucător trimite o mutare invalidă (de exemplu, mută un pion in forma de L).
- Serverul detectează mutarea invalidă și trimite înapoi un mesaj de eroare, solicitând jucătorului să încerce din nou.

Finalizarea jocului:

- Unul dintre jucători dă șah-mat adversarului.
- Serverul detectează șah-matul și declară câștigătorul.
- Ambii jucători primesc o notificare despre finalul jocului și rezultatul acestuia.

5 Concluzii

Aplicația ChessS reprezintă o implementare a jocului de șah într-un mediu server-client, ilustrând aplicarea noțiunilor avansate de rețea și programare concurentă folosind protocoale TCP. Prin gestionarea conexiunilor multiple și actualizarea dinamică a stării jocului, aplicația oferă un mediu stabil și interactiv pentru desfășurarea partidelor de șah.

6 Bibliografie

1. https://www.andreis.ro/teaching/computer-networks
2. https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php
3. http://cisco.ctcnvk.ro/RS2Romana/course/module2/2.1.2.1/2.1.2.1.html
4. https://ro.wikipedia.org/wiki/Transmission_Control_Protocol
5. https://profs.info.uaic.ro/~eonica/rc/
6. https://ramonnastase.ro/blog/ce-este-o-adresa-ip/
7. https://ro.wikipedia.org/wiki/Port_(re%C8%9Bea)