8. PROGRAMAREA CONECTIVITĂȚII ÎN INTERNET

8.1. Probleme generale

Definiție:

Un socket este un punct terminal al unui canal de comunicație prin internet care utilizează Internet Protocol (IP).

Spre deosebire de porturile software care facilitează comunicația, în special pe aceeași mașină, socketurile sunt utilizate pentru crearea de fluxuri binare între programe rulând pe mașini diferite.

Comunicația bazată pe socketuri este întâlnită între browserele Web şi servere, la clienții mesageriei instantanee, la clienții şi serverele de email, în aplicațiile de tip peer-to-peer.

În Erlang comunicația se desfășoară între noduri prin transferul de șiruri de octeți. Pentru a avea o seminificație, pachetele de octeți transferate trebuie să fie realizate conform unor convenții cunoscute sub numele de protocoale UDP (User Datagram Protocol) şi TCP (Transmission Control Protocol).

Cele două protocoale, UDP şi TCP, sunt bazate pe modulele Erlang de bibliotecă *gen_udp* şi *gen_tcp*.

Aceste două module permit obținerea unui canal de comunicație bidirecțional prin intermediul protocolului IP.

8.2. Protocolul UDP

UDP este un protocol de transmisie fără verificarea realizării conexiunii. După transmiterea unui pachet UDP, un alt sochet poate recepționa acel pachet, dar nu face nici o confirmare a primirii lui. Protocolul UDP nu este protejat de erorile ce pot apărea în timpul transmisiei pachetelor. Acestea pot sosi la destinație în ordine diferită de cea de la emisie în funcție de lungimea rutei pe care au parcurs-o sau chiar se pot pierde fără posibilitate de recuperare, lipsind din setul recepționat.

Definiție:

Procedeul de emitere a unor date către destinatari, fără precizarea adresei sau numelui, se numeşte difuzare (*broadcast*). Destinatarii trebuie să ia măsurile necesare de recunoaştere a mesajelor sau a pachetelor de date care le sunt destinate.

Protocolul UDP, care funcționează în modul difuzare, este potrivit pentru utilizarea la transmiterea mesajelor scurte între un server şi un număr mare de clienți.

Scrierea de clienți şi servere în Erlang este mai simplă pentru că nu este necesar şi să se stabilească şi să se mențină conexiunea între server şi clienți.

Datorită faptului că UDP este un protocol fără stabilirea conexiunii, nu este posibil ca serverul receptor să blocheze clientul prin refuzul de a primi datele, în situația în care acesta trimite mesaje nedorite.

Anumite protecții trebuie luate pentru protejarea de situațiile în care unele pachete UDP sunt duplicate şi livrate de două ori prin rețea către destinatar. O metodă frecvent utilizată în acest scop este generarea unor referințe unice de către codul client şi verificarea dacă aceste referințe sunt returnate de server.

8.3. Protocolul TCP

Definiție:

Transmission Control Protocol (TCP) este un protocol cu verificarea stabilirii conexiunii între sursă şi destinație, pentru transferul informației sub forma unor pachete binare.

Protocolul TCP asigură recepția garantată a pachetelor de date şi mai mult decât atât, se asigură şi ordonarea acestora la recepție conform ordinii la emitere.

În cazul TCP, conexiunea odată stabilită rămâne deschisă până când una din părți o închide explicit sau încheie cu o eroare.

În practică, de regulă, un proces denumit "ascultător" (*listener*) se află în aşteptarea primirii unor interogări, mesaje de cerere.

În momentul în care cererea soseşte, procesul ascultător confirmă stabilirea conexiunii, devenind proces "acceptor".

Procesul de acceptare se poate desfăşura pe baza a două scenarii:

a. Se generează un nou proces (prin funcția **`spawn`**) care devine procesul acceptor, în timp ce procesul ascultător revine la funcția sa anterioară de aşteptare a unei noi cereri de conexiune.
b. Procesul ascultător devine proces acceptor şi generează un nou proces care devine noul ascultător.

Structura celui mai simplu program server bazat pe TCP execută următoarea secvență de operații:

Ascultă → acceptă cererea → recepționează → decodifică → evaluează → codifică → emite → închide.

Îmbunătățirea acestui model de program server TCP se poate face în mai multe etape.

Prima dezvoltare se referă la capacitatea de a avea mai multe conexiuni deschise în acelaşi timp. Deschiderea şi menținerea mai multor conexiuni simultan se poate face în două moduri:

- printr-un singur socket la care mesajele suplimentare sunt puse într-o coadă de aşteptare până la tratarea celor dinaintea lor;.
- prin generarea de noi socketuri în procese paralele folosind funcția *spawn*.

În primul caz serverul se numeşte secvențial, iar în al doilea caz se numeşte paralel.

Observație:

Se pot crea mii de procese paralele care să gestioneze fiecare un număr de conexiuni pe baza a câte unui socket. Se poate limita numărul de conexiuni pe baza unui contor.

O a doua dezvoltare se referă la controlul fluxului de mesaje recepționate.

Observație:

Un socket TCP poate fi deschis în următoarele variante: activ, activ o dată şi pasiv. Aceste opțiuni pot fi precizate în funcția **`gen_tcp:connect`**. Socketul activ poate primi oricâte mesaje, cel activ o dată poate primi un mesaj după care trebuie revalidat. În cazul socketului pasiv procesul care controlează socketul trebuie să-i trimită un mesaj special prin care să-i precizeze că doreste recepția unui mesaj de la un anumit client.

Observație:

Socketul pasiv poate fi folosit pentru controlul recepției fluxului de date în cazul în care există prea multe mesaje, sau mesajele sunt nedorite. Se spune că recepția poate fi fără blocare, sau cu blocare. Există şi varianta hibridă de blocare parțială.

O a treia dezvoltare a unui program server bazat pe socketuri este dotarea acestuia cu secvențele necesare tratării erorilor, un lucru ce se poate face din Erlang mai simplu decât din alte limbaje.

Limbajul ERLANG

Programarea sistemelor concurente şi distribuite

Eugen Diaconescu

2. ASPECTE APROFUNDATE ALE PROGRAMĂRII SECVENȚIALE

2.1. Tratarea excepțiilor
2.2. Metoda acumulatorului
2.3. Expresiile bloc
2.4. Macrourile
2.5. Preprocesorul
2.6. Numerele
2.7. Masive binare (*binaries*)
2.8. Dicționare
2.9. Referințe

3. PROGRAMAREA CONCURENTĂ

3.1. Elemente de bază
3.1.1. Primitivele concurenței
3.1.2. Comunicarea interproces: emiterea şi recepția mesajelor
3.2. Arhitectura client-server
3.3. Recepția mesajelor întârziate
3.4. Citirea cozii de mesaje
3.5. Înregistrarea proceselor
3.6. Scrierea programelor concurente
3.6.1. Structura standard
3.6.2. Tratarea erorilor în programele concurente
3.6.3. Actualizarea dinamică a codului

4. PROGRAMAREA DISTRIBUITĂ

4.1. Motivare
4.2. Concepte ale arhitecturii distribuite în Erlang
4.2.1. Conceptul "nod"
4.2.2. Înregistrarea proceselor
4.2.3. Conceptul "conexiune la nod"
4.3. Dezvoltarea unei aplicații distribuite
4.3.1. Cel mai simplu sistem distribuit, cu noduri pe acelaşi calculator
4.3.2. Aplicații distribuite pe maşini diferite în aceeaşi rețea sau în Internet
4.3.3. Setarea căilor de căutare implicite în Linux şi MSYS
4.3.4. Exemplu de program comentat

5. INTERFAȚA CU ALTE LIMBAJE

5.1. Principii de bază
5.2. Porturile
5.3. Funcția open_port
5.4. Drivere Linked-in