Számítógéphálózatok (8. rész)

Elemi adatkapcsolati protokollok.

Sorozatunk jelen részében az adatkapcsolati protokollokkal foglalkozunk, azaz megnézzük, miként zajlik az adatkapcsolati rétegen „belüli” kommunikáció. Eddig sok érdekes dolgot hallhattunk az adatkapcsolati réteg feladatairól, most itt az ideje, hogy ezek gyakorlati megvalósításáról is szót ejtsünk. Adott tehát a probléma: A és B gép adatot cserélnének egymással. Mindezt egy olyan világban szeretnék megtenni, ahol a keretek eltünhetnek, megsérülhetnek, illetve a két gép sebessége között jelentős eltérés is előfordulhat. Hogy mindez ne jelenthessen problémát, szükség lesz az adatkapcsolati protokollokra.

Továbbra is az egyszerűből haladunk a bonyolultabb felé, azaz először egy olyan protokollal ismerkedünk meg, ahol csak az A gépnek van módja adatot küldeni a B felé, és élünk azzal a nevetséges feltételezéssel, hogy az A gép végtelen mennyiségű elküldendő adattal rendelkezik, illetve az adatok előállítása semennyi időbe se kerül. ***(A hálózati réteg mindig készenlétben áll).*** A későbbiekben megszabadulunk ezektől a megkötésektől, ezzel eljutva egy gyakorlatban is használható protokollhoz.

Keretek és csomagok

Ahhoz, hogy bemutathassuk az adatkapcsolati protokollok működését, fontos, hogy bizonyos dolgokban előre megállapodjunk. Most sem térünk el az általunk bemutatott hálózati modelltől, azaz a fizikai, az adatkapcsolati és a hálózati réteget továbbra is független, egymással párhuzamosan futó folyamatokként tételezzük. Minden réteg csak a „saját megfelelőjével” kommunikál, azaz az A gép hálózati rétege a B gép hálózati rétegével cserél információt. Emlékezzünk csak az első részben bemutatott filozófusok esetére: mindkét filozófus egymással kommunikál, de az üzenetváltást a titkárnők valósítják meg.

Ebben az esetben a hálózati réteg a filozófus, az adatkapcsolati réteg a titkárnő, a fizikai réteg pedig az email vagy fax, amin a titkárnők az üzeneteket továbbítják. A valóságban ezek a rétegek nem minden esetben vannak ennyire különválasztva. Az adatkapcsolati réteg lehet például a hardver része is. Ehhez a modellhez mi csak azért ragaszkodunk foggal-körömmel, mert így a különböző részfeladatok gyakorlati megvalósítását egymástól függetlenül is be tudjuk mutatni. Sőt mi több: egyik rétegnek sem kell tudnia arról, hogy a másik miként oldja meg a saját feladatát. Ugyanígy a filozófusokat sem foglalkoztatják az élet értelméhez képest olyan, teljesen jelentéktelen kérdések, hogy például a titkárnőjük email-en vagy faxon továbbítja az üzeneteiket.

Mostantól kezdve a hálózati réteg által egyszerre elküldött adatblokkot csomagnak nevezzük, az adatkapcsolati réteg „megfelelője” pedig a keret lesz. Ha a hálózati réteg adatot kíván küldeni, akkor ő azt először egy csomagba szervezi (hogy egy csomagnak milyen részei vannak, arról majd a hálózati réteg tárgyalásakor visszatérünk), ezt követően pedig átadja azt az adatkapcsolati rétegnek. Fontos, hogy míg a hálózati réteg számára a csomag egy jól meghatározott struktúrával bír, az adatkapcsolati rétegnek ez csak bitek sorozata.

Az adatkapcsolati réteg a csomaghoz hozzáragaszt némi plusz információt, ezeket a keret fejlécének fogjuk nevezni. A keret tehát két részből áll: a hálózati réteg számára is érdekes adatmezőből, és a fejlécből, amely további három mezőre oszlik.

Az első a ***kind***, amely egy logikai érték, és azt mondja meg, hogy van-e adat a keretben. Erre nem azért van szükség, hogy haszontalan üres keretekkel terhelhessük kedvünkre a hálózatot, hanem azért, hogy lehetőség legyen vezérlőkeretek küldésére is. Amikor ugyanis érkezik a keret, akkor annak adatmezejét az adatkapcsolati réteg mechanikusan továbbítja. Felmerülhet azonban az igény, hogy a két adatkapcsolati réteg kommunikálhasson egymással.

A keret fejlécéhez tartoznak még az ***ack*** és a ***seq*** mezők, ezekkel valósíthatjuk meg a keretek sorszámozását, illetve a nyugtázást.

Mivel az algoritmusokat a legegyszerűbben valamilyen programnyelvhez hasonló nyelven segítségével lehet ismertetni, mi egy C-hez nagyon hasonló, ám rendkívül magas szintű nyelvet használunk a protokollok bemutatására.

Korlátozás nélküli szimplex protokoll

Ez a lehető legegyszerűbb protokoll, amit csak el tudtunk képzelni. Mint már említettük, először azt a helyzetet vizsgáljuk, amikor csak az

A gép küld, akinek mindig van adata, és egy percet sem kell várni egy-egy keret előállítására. A vevő sincs híján a mesébe illő tulajdonságoknak: végtelen gyorsan képes feldolgozni a hálózaton hozzá érkező kereteket.

Ezt úgyis megfogalmazhatjuk, hogy a vevő végtelen puffer területtel rendelkezik, ahova bármennyi feldolgozásra váró keret befér.

Azon már meg se lepődjünk, hogy a kommunikcáiós csatorna annyira jó minőségű, hogy minden keret sértetlenül megérkezik.
A *2. listán* láthatjuk ennek a protokollnak egy megvalósítását. A `Kuldo()` függvény a küldő, a `Vevo()` pedig a fogadó gépen fut. Mindkét eljárás lelke egy végtelen ciklus. A küldő amilyen gyorsan csak tudja, fogadja az adatokat a hálózati rétegtől, és egyből be is rakja egy keret info mezejébe.
A keret egy összetett típus, ennek definiálását láthatjuk az *1. listán*. Ez a protokoll csak az info mezőt használja, hiszen forgalomszabályozásról, illetve nyugtázásról ő még semmit sem hallott.
A küldő miután összerakta a keretet (ami most még nem volt egy különösebben megterhelő feladat), egyből át is dobja a fizikai rétegnek, amely bitenként továbbítja azt a csatornán keresztül.
A vevő eljárás sem bonyolultabb ennél. Először is vár, hogy történjen valami. Jelen esetben a jövő varázsgömb nélkül is kiszámítható, ugyanis csak egy dolog történhet: egy sértetlen keret érkezik. Az „e" változó értéke tehát biztosan „`keret_erkezett`" lesz, így annak értékének ellenőrzése nélkül egyszerűen csak át kell venni a keretet a fizikai rétegtől, majd továbbítani annak info mezőjét a hálózati réteg felé.

1. lista A keret összetett típus

```
typedef struct
{
  bool kind;
  int seq, ack;
  csomag info;
} keret;
```

Szimplex „megáll és vár" protokoll

A számítógépeink továbbra is varázskábellel vannak összekötve, ahol nem sérülhetnek, illetve veszhetnek el a keretek. A vevő azonban már nem végtelen memóriájú, így előbb vagy utóbb a folyamatosan érkező keretektől megtelik a puffer, és kénytelen lesz a forrásállomáshoz szünetért könyörögni.
A megoldandó probléma tulajdonképpen nem más, mint hogy valamilyen módszerrel visszafogni a küldő állomást abban, hogy keretek sokaságát zúdítsa a vevőre. Általánosan úgy lehetne ezt megfogalmazni, hogy ha mondjuk a vevőnek T időbe telik, míg a fizikai rétegtől a hálózati rétegig eljuttatja az adatot (azaz T a `FizikaiRetegtol` és a `HalozatiRetegnek` utasítások végrehajtásának ideje), akkor a küldő T idő alatt átlagosan kevesebb mint egy keretet küldhessen.
Erre a legkézenfekvőbb megoldás az lehet, ha a forrásba „bedrótozunk" valamiféle késleltetést. A vevőnek azonban nem csak a keretek fogadásával kell foglalkoznia, hanem sok más egyéb feladata is lehet, például más gépekkel is kommunikálhat.
Így a T értéke folyamatosan változhat. Persze előfordulhat, hogy létezik a T-re egyfajta legrosszabb érték, amelynél a T értéke sohasem lehet nagyobb. Ha ezt választjuk a késleltetés időtartamának, akkor biztos, hogy a vevő minden esetben képes lesz feldolgozni az összes keretet, akármennyi más egyéb teendője is akad. Ezzel azonban az a gond, hogy rendkívül pazarló megoldás. Ha ugyanis a vevőnek éppen nincs semmi dolga, akkor sem fogja gyorsabban kapni a kereteket.
Praktikusabb megoldás tehát ha visszacsatolást létesítünk a vevő és a forrás között. Ez annyit tesz, hogy a vevőnek módjában áll visszaszólni a forrásnak.
A „megáll és vár" protokoll esetében például a forrás addig nem küldi el a soron következő keretet, amíg a vevő vissza nem küld egy „álkeretet", amellyel közli, hogy az előző keretet sikeresen feldolgozta, készen áll a következő fogadására *(3. lista)*. Az ábrán láthatjuk, hogy tulajdonképpen teljesen mindegy, hogy mi az álkeret, a forrásnak ugyanis csak a beérkezés a fontos. Rögtön ezt követően a forrás „felébred", és ismét elküld egy keretet. Ezután megint várakozik.
Másik fontos különbség az előző protokollal szemben, hogy itt már az adatok visszafelé is áramolnak. Ez azt jelenti, hogy az összekötő mágikus kábelnek kétirányúnak kell lennie. Az is igaz, hogy az adatok áramlásának iránya szigorúan változik, azaz hol az egyik, hol a másik küld.

Ha zaj van a csatornán...

Bonyolítsuk tovább a helyzetet és cseréljük le a zajmentes mágikus hálózati kábeleinket teljesen hétköznapiakra. Az előző protokoll nyilván használhatatlan, hiszen most már könnyedén előfordulhat, hogy egy-egy keret megsérül, esetleg el is veszik.
A nagy ötlet az, hogy módosítsuk az előző protokollt a következőképp: amikor a forrás elküld egy keretet, elindít egy időzítőt, és vár, hogy a vevő visszaküldjön egy nyugtát a keret sértetlen megérkezéséről.
Ha az időzítő lejártáig nem érkezik meg a nyugta, akkor a kérdéses keret valószínűleg elveszett, ezért a forrás azt ismét elküldi.
Ez jó megoldásnak tűnik, de végig számításba kell venünk azt az esetet is, amikor nem a keret, hanem a nyugta tűnik el. Ilyenkor a forrás nem értesül a keret megérkezéséről és ismét elküldi azt.
Ekkor a vevő ismét nyugtáz, és ha nem veszik el ismét a nyugta, akkor a forrás küldheti a következőt. Ha kicsit utánaszámolunk, láthatjuk, hogy a vevőhöz ugyanaz a keret kétszer is megérkezett. Rendkívül kellemetlen, ha például egy állomány átvitelekor annak bizonyos részei megduplázódnak.
Ezért a vevőnek valahogy fel kell tudnia ismerni azt, ha a beérkezett keret az előző ismétlése. A kereteket legegyszerűbben úgy különböztethetjük meg egymástól, ha sorszámmal látjuk el őket.
A sorszámot nyilván a keret fejlécében tárolnánk, és mivel nem jó, ha a fejléc túl nagy, ezért fontos a kérdés: a keretek sorszámait hány biten ábrázoljuk?
Könnyen belátható, hogy erre elegendő egyetlenegy darab bit is. Ha végig gondoljuk a protokoll működését, akkor megállapíthatjuk, hogy a nem egyértelmű keret csak az n. vagy az n+1. lehet.
Ha ugyanis az n. keret megsérül, akkor nem fog róla nyugta érkezni, ezért a forrás addig küldi, amíg végre hibátlanul át nem ér.

2. lista Korlátozás nélküli szimplex protokoll

```
Küldő:
void Kuldo()
{
  keret k;
  csomag cs;

  while(true) // végtelen ciklus
  {
    HalozatiRetegtol(&cs);
    // megkapjuk a hálózati rétegtől az
    // elküldendő csomagot
    k.info = cs;
    FizikaiRetegnek(&k);
  }
}

Vevő:
void Vevo()
{
  keret k;
  esemeny e;

  while(true)
  {
    EsemenyreVar(&e); // csak a keret_erkezett
                      // a lehetséges esemény
    FizikaiRetegtol(&k);
    HalozatiRetegnek(&k.info);
  }
}
```

3. lista Szimplex megall es var protokoll

```
void Kuldo()
{
  keret k;
  csomag cs;
  esemeny e;

  while(true)
  {
    HalozatiRetegtol(&cs);
    k.info = cs;
    FizikaiRetegnek(&k);
    EsemenyreVar(&e);
    // addig ne folytassuk, amig nincs ra
    // engedely
  }
}

void Vevo()
{
  keret k, s;
  esemeny e;

  while(true)
  {
    EsemenyreVar(&e); // csak a keret_erkezett
                      // a lehetséges esemény
    FizikaiRetegtol(&k);
    HalozatiRetegnek(&k.info);
    FizikaiRetegnek(&s); // alkeret kuldese
  }
}
```


Ha a nyugta veszik el, akkor ismét az n. keret kerül elküldésre. Ha a nyugta is rendben megérkezik, akkor kerül továbbításra csak az n+1. keret.
Folytatva a dolgot, ha az n+2. keretet elküldjük, akkor az az esemény magában hordozza azt is, hogy az m+1. keret nyugtája már sértetlenül megérkezett. Ez pedig azt jelenti, hogy az m. keret, és annak nyugtája is célba ért. Ezért az egy bites sorszám elegendő.
Azokat a protokollokat, ahol a forrás minden keret után a keret megérkezéséről pozitív visszajelzésre vár, ARQ-nak ***(Automatic Repeat reQuest – automatikus ismétléskérés)*** nevezik. Ezek is egyirányú ***(szimplex)*** protokollok, de már tudják kezelni az elveszett kereteket.
Mi történik azonban akkor, ha a forrás időzítője hamarabb jár le, mint ahogy a nyugta célbaérne. Ha már megtörtént a keret újraküldése, és az időzítő nullázása, akkor – ha a kérdéses nyugta mégis célba ér – a forrás azt fogja gondolni, hogy ez az imént elküldött keretre érkezett a nyugta. Ezért fontos, hogy a forrás valamiképpen tudomást szerezzen arról, hogy a beérkezett nyugta valójában melyik keret megérkezését erősíti meg. Persze ezt a problémát könnyen orvosolhatjuk azzal, ha maga a nyugta is tartalmazza a hozzá tartozó keret sorszámát. A *4. listán* láthatunk egy példát a zajos csatornán is alkalmazható szimplex megáll-és-vár protokollról. A forrás minden keret elküldése után elindít egy időzítőt. Ezután várakozik addig, amíg nem történik valami esemény. Ilyen esemény lehet például az, hogy a nyugta visszaérkezik, illetve visszaérkezik valami, de átviteli hibával, vagy esetleg nem érkezik semmi, és lejár az időzítő.
Az első esetben a forrás lekérheti a következő csomagot a hálózati rétegtől, invertálja a sorszámot (ha 0 volt, akkor 1 lehet, vagy pedig fordíva), és továbbadja a fizikai rétegnek. Ha azonban sérült nyugta, vagy éppen semmi sem érkezik, akkor ismét az előző csomagot küldi, és a sorszámot sem írja felül.
A vevő minden beérkezett keretet megvizsgál, hogy egész véletlenül nem-e duplikátum. Erre szolgál a „`vart_keret`" változó. Ha a beérkező keret sorszáma nem egyezik meg ennek a változónak az értékével, akkor biztos, hogy duplikátumról van szó.

Ráültetés (piggybacking)

Eddig olyan helyzeteket vizsgáltunk, amikor csak az egyik fél küld adatokat a másiknak. Igaz, a második és a harmadik protokoll esetében már a kommunikáció valójában kétirányú volt, tehát szükség volt egy olyan csatornára, ahol az adatok mind a két irányban áramolhatnak.

4. lista szimplex megall-es-var protokoll

```
void Kuldo()
{
  sorszam kovetkezo_keret; // a kovetkezo keret
                           // sorszama
  keret k;
  csomag cs;
  esemeny e;

  kovetkezo_keret = 0; // inicializaljuk
  HalozatiRetegtol(&cs);  // lekerjuk az elso
                          // elkuldendo csomagot
  while (true)
  {
    k.info = cs; // bemasoljuk a csomagot
                 // a keretbe
    k.seq = kovetkezo_keret; // beallitjuk a
                             // keret sorszamat
    FizikaiRetegnek(&k); // elkuldjuk a keretet
    IdozitoIndul(k.seq); // elinditjuk az
                         // idozitot
    EsemenyreVar(&e);
    // lehetseges esemenyek: keret erkezes,
    // idotullepes, ill. chesksum hiba
    if ( e == keret_erkezett ) // ha megerke-
                               // zett a nyugta
    {
      FizikaiRetegtol(&k); // megszerezzuk a
                           // nyugtat
      if ( k.ack == kovetkezo_keret )
      {
        HalozatiRetegrol(&cs);  // kovetkezo
                                // csomag ....
        inc(kovetkezo_keret); // invertaljuk a
                              // sorszamot
      }
    }
  }

void Vevo()
{
  sorszam vart_keret;
  keret k, nyugta;
  esemeny e;
  // mivel itt mar megserulhetnek a keretek,
  // ezert a cheksum_hiba is
  // lehetseges esemeny

  vart_keret = 0;
  while(true)
  {
    EsemenyreVar(&e);
    if ( e == keret_erkezett )
    {
      FizikaiRetegtol(&k); // megszerezzuk a
                           // keretet
      if ( k.seq == vart_keret ) // ha erre a
                                 //keretre vartunk
      {
        HalozatiRetegnek(&k.info); // atadjuk a
                             // halozati retegnek
        inc(vart_keret); //legkozelebb mar a
                         //masik sorszamra varunk
      }
      nyugta.ack = 1 - vart_keret; // melyik
                           // keretnek a nyugtaja
      FizikaiRetegnek(&nyugta); // elkuldjuk a
                                // nyugtat
    }
  }
}
```

Ilyen csatornát a legegyszerűbben úgy csinálhatunk, hogy két egyirányú csatornát teszünk egymás mellé, csak végpontjaikat „fordítva” kötjük be. Az egyik dróton tehát A-ból B-be, a másikon pedig B-ből A-ba áramolhatnak az adatok. Ez azonban meglehetősen pazarló megoldás, mivel egyrészt a sávszélesség nagyrésze kihasználatlan marad, másrészt a felhasználó két áramkörért fizet, de kapacitásban annyit kap, mintha egyet használna.

Az ideális megoldás nyilván az, hogy egy áramkört használunk az oda és visszamenő adatok továbbítására.

Erre láttunk példát az előző két protokoll bemutatásakor is. Ilyenkor az A-ból B-be, és a B-ből A-ba tartó adatok egymással összekeveredtek.

Amikor a B állomás is küld adatokat az A-nak, akkor mindkét félnek meg kell tudnia különböztetnie az adat- és a nyugta kereteket. Erre szolgál a keret fejlécében elhelyezett „kind” mező.

A hatékonyságot azonban még tovább is növelhetjük, például úgy, hogy a vevő nem küld minden keret beérkezése után azonnal nyugtát. Csökkenthetnénk a hálózat forgalmát úgy, ha a nyugtát egy kifelé menő adatkerethez „csatoljuk”.

Ha tehát beérkezik egy keret, akkor a vevő a nyugta küldésével addig vár, amíg a hálózati rétegtől nem kap elküldendő adatot. Ha ezt megkapja, akkor a nyugtát ráülteti a kifelé menő adatkeretre. Így a vevő két keret helyet csak egyet küld. Ezzel a megoldással sokat nyerhetünk, hiszen nem csak jobban használhatjuk ki a rendelkezésre álló sávszélességet, hanem a vevő oldalán kevesebb „keret érkezett” megszakítás váltódik ki. Ráadásul, hogy ez megvalósítható legyen, csak egy bitet kell módosítanunk a keret fejlécében.

Ez a módszer azonban csak akkor működik, amikor mindkét fél végtelen mennyiségű elküldendő adattal rendelkezik. Ha ugyanis a vevőhöz megérkezik egy keret, de annak hálózati rétege éppen semmit sem akar küldeni, akkor előfordulhat, hogy a forrás által elindított időzítő lejár.

Ez pedig a kérdéses keret ismétlését fogja eredményezni, tehát ha időközben érkezik is adat a vevő hálózati rétegétől, a nyugta már elvesztette jelentőségét.

Megoldás lehet a jövőbelátás, azaz megjósolni, hogy a hálózati réteg mikor fog adatot átadni, és a forrás időzítőjét ehhez hangolni. Erre azonban még a TV jós sem képes, ezért jobb megoldásnak tűnhet, ha a vevő inkább vár egy előre meghatározott ideig, majd ha addig nem kapott feladatot a hálózati rétegtől, elküldi a nyugtát egy önnálló keretként.

A következő részben innen folytatjuk. Izgalmakban továbbra sem lesz hiány: szó lesz a csúszóablakos protokollokról, végesállapotú automatákról, híres adatkapcsolati protokollokról (például a PPP-ről, amelyet az Internetben is széles körben alkalmaznak).

Garzó András (garzo@interware.hu)
Körülbelül három éve foglalkozik Linux- és más Unix-rendszerekkel. Legjobban az operációs rendszerek lelkivilága érdekli, de nyitott egyéniség. Kedvenc étele a palacsinta, és van egy Richard nevű macskája. Minden észrevételt, megjegyzést, levelet szívesen fogad.