A Linux-csomagszűrő felépítése (1. rész)

A Linux-csomagszűrőről szóló kétrészes sorozatunk első részében Gianluca elmeséli, milyen utat tesz meg egy csomag a rendszermagon keresztül, míg eljut a rendeltetési helyére.

Talán néhányan közületek emlékeznek még „Linux-csomagszűrő: bájtok elfogása a hálózatban" című cikkemre, mely a Linuxvilág júniusi számában jelent meg, és a Linux-rendszermagba épített csomagszűrő működését taglalja. Abban a cikkben a csomagszűrő működését tekintettük át, ezúttal pedig a rendszermag azon részébe ásom be magam, amely a szűrésért felelős, és egyúttal a rendszermag csomagkezelő részébe is betekintést nyújtok.

Az előző cikk témái

Legutóbbi cikkemben felmerült néhány dolog a Linux csomagkezelő rendszerével kapcsolatban. Legjobb lesz, ha ezek közül a legfontosabbakat most újra átvesszük:

- A csomagok átvétele először a hálózati kártya eszközmeghajtójának szintjén zajlik, pontosabban annak megszakításkezelő eljárásában, és az eszközmeghajtó továbbítja egy felsőbb szintre, ahol később feldolgozásra kerül.
- Amennyiben a fogadás és protokollértelmezés művelete során a rendszer túlságosan elfoglalt, néhány csomagot eldob. Az elkövetkezőkben, amint a csomag egyre közelebb ér a felhasználói szinthez, az alacsonyabb szintű protokollok adatai elvesznek.
- Foglalati szinten, közvetlenül mielőtt a csomag elérne a felhasználói szintre, a rendszermag ellenőrzi, hogy az adott csomag számára létezik-e nyitott foglalat – amennyiben nincs, a csomagot eldobja.
- Az ezt követő szinten a Linux-rendszermag `PF_PACKET` nevű szolgáltatása érhető el. Ezzel olyan foglalatokat lehet létrehozni, amelyek közvetlenül a hálózati kártya meghajtójától fogadnak adatokat. Ily módon minden egyéb protokollkezelő eljárás átugorható, és bármilyen csomagot elkaphatunk.
- Az ethernetkártya többnyire csak az olyan csomagokat továbbítja, amelyek magának a rendszermagnak szólnak, minden egyebet eldobnak. Mindazonáltal létezik egy olyan beállítás, amely lehetővé teszi, hogy a hálózati kártyán áthaladó összes csomag MAC-címétől (promiscuous mód) függetlenül továbbítódjon.
- Legvégül pedig a foglalaton egy szűrőt is elhelyezhetsz, hogy csak azok a csomagok továbbítódjanak, amelyek megfelelnek a szűrő beállításainak. Ezt egy `PF_PACKET` foglalattal összekapcsolva remek eszközhöz jutunk hálózatunk forgalmának hatékony elkapására (sniff).

Bár lehallgatónk a `PF_PACKET` foglalatokon alapul, a Linux foglalati szűrője nincs az ilyen eszközök előállítására korlátozva. Valójában ezzel a foglalattal egyszerű TCP- és UDP-csomagokat is elkaphatunk, hogy kiszűrjük a nem kívánt forgalmat – természetesen ez a lehetőség kevésbé szokványos. Az elkövetkezőkben többször hivatkozom e foglalatokra és foglalati felépítésre. Ezen írás keretein belül a két fogalmon ugyanazt értjük, mivel a foglalat valójában a rendszermag belsejében értelmezett foglalati felépítést takarja. A rendszermag számon tart egy foglalati felépítést, továbbá a foglalatok kezeléséhez is egyet, de a kettő közötti különbségtől most eltekintünk.

Egy másik lényeges szerkezet, amely még gyakran szóba kerül, az `sk_buff` (vagy más néven foglalati átmeneti tár), mely a csomagok rendszermagon belüli ábrázolásáért felelős. Ez a szerkezet olyan módon van felépítve, hogy az egyes fejlécadatok hozzáadása vagy elvétele a legkevésbé legyen költséges: nincs szükség adatok másolására, mivel az összes adatváltoztatást mutatók eltolásával oldották meg.

Mielőtt továbblépünk, nem árt, ha megvilágítjuk az eddigieket. A Linux foglalati szűrőnek nevével ellentétben semmi köze a Netfilterhez (lásd még a Linuxvilág 2001 októberi számában). A Netfilter valamikor a rendszermag 2.3-as sorozatának legelején jelent meg, és egészen más a szerepe. Bár a foglalati szűrőkhöz hasonlóan ennek is a csomagok felhasználói szintre juttatása a feladata, ez más célból történik: hálózati címváltásból (NAT), a csomagok felhasználói szintű módosításából, kapcsolatkövetésből, biztonsági megfontolásból és így tovább. Ha mindössze csak a csomagok elkapására (sniff) van szükséged, ennek legkézenfekvőbb eszköze a Linux-csomagszűrő. Most már folytathatjuk a csomagkövetést a számítógépre való belépéstől egészen egy felhasználói programnak történő kézbesítésig. Először egy általános foglalatot (nem `PF_PACKET`) veszünk szemügyre. Kapcsolati szinten egy ethernethálózatból indulunk ki, mivel ez a legelterjedtebb, és egyúttal a legkönnyebben érthető is. Alapvetően minden más kapcsolati szintű protokoll az ethernethez hasonlítható, többnyire nincsenek lényeges különbségek.

Ethernetkártya- és alacsony rendszermagszintű belépés

Mint előző cikkünkben már szó volt róla, az ethernetkártyának van egy hálózati szintű, úgynevezett MAC-címe, és az erre a címre érkező csomagokat emeli ki a hálózatból. Amikor a kártya egy olyan csomaggal találkozik, amely a saját címére szól vagy az úgynevezett Broadcast-címre (FF:FF:FF:FF:FF:FF ethernet esetén), a csomagot a memóriaterületére másolja. Miután az ethernetkártya a csomagot teljes egészében átvette, megszakítási kérelmet hoz létre, amelyet a hálózati kártya meghajtója kezel. Ezalatt a meghajtóprogram az egyéb megszakítások beérkezését letiltja, és többnyire a következő feladatokat hajtja végre:

- Lefoglal egy újabb `sk_buff` szerkezetet, az ***include/linux/skbuff.h***-ban található rendszermagszintű értelmezésnek megfelelően. Ez a szerkezet a rendszermag szemszögéből a beérkező csomagokat ábrázolja.
- A kártya átmeneti tárából az újonnan lefoglalt `sk_buff`

tárterületére másolja a csomagot, mely a DMA felhasználásával is megtörténhet.
- Meghívja a `netif_rx()` függvényt, ez az általános, csomagok beérkezésekor meghívásra kerülő függvény.
- Miután a `netif_rx()` visszatér, újra engedélyezi a megszakításokat, és befejezi a műveletet.

A `netif_rx()` függvény a rendszermagot a beérkezett csomaggal elvégzendő következő műveletre is felkészíti; az `sk_buff`-ot a CPU pillanatnyi beérkező várakozási sorába helyezi, és a `NET_RX` `softirq`-t (erre még visszatérünk) megjelöli, hogy az a `__cpu_raise_softirq()` rendszerhíváson keresztül meghívódjon. Ezzel kapcsolatban meg kell említenünk két fontos dolgot: először is, ha ez a várakozási sor már megtelt, az új csomagot a rendszer eldobja; másodszor: processzoronként csak egy várakozási sorunk van, az új késleltetett feldolgozómodellel (softirq) így több processzor esetén lehetővé válik a csomagok párhuzamos feldolgozása.
Ha egy valódi ethernetmeghajtót működés közben szeretnél látni, vess egy pillantást a NE2000 kártya PCI-meghajtójára, melyet a ***drivers/net/8390.c***-ben találsz; a hálózati kártya megszakítását az `ei_internet()` kapja el, majd meghívja az `ei_receive()` függvényt, ami a következőket hajtja végre:

- A `dev_alloc_skb()` függvényhíváson keresztül helyet foglal egy új `sk_buff` szerkezetnek.
- A csomagot kiolvassa a kártya átmeneti tárából (`ei_block_input()` függvény), és ennek megfelelően állítja be az `skb->protocol`-t.
- Meghívja a `netif_rx()`-et.
- A fentieket legfeljebb tíz egymást követő csomagra vonatkozóan megismétli.

Ennél jóval bonyolultabb megoldást mutat be a ***3c59x.c*** fájlban található 3COM-meghajtó, mely a csomagot a DMA segítségével másolja a kártya tárterületéről az `sk_buff` területére.

Hálózati szintű feldolgozás

Nézzük meg közelebbről a `netif_rx()` függvényt! Mint már említettem, ennek a függvénynek az a feladata, hogy fogadja a csomagot a hálózati kártya meghajtójától, és továbbítsa a magasabb szintű rétegek felé. Ez a függvény szolgál központi gyűjtőhelyül a különböző hálózati kártyameghajtóktól beérkezett csomagoknak, amelyekkel a felsőbb szintű protokollokat látja el.
Mivel ez a függvény megszakítási környezetben fut (azaz a korábban keletkezett megszakítás hatására fut le), működése közben a megszakítások le vannak tiltva, ugyanakkor rövidnek és gyorsnak kell lennie. Nem hajthat végre hosszadalmas ellenőrzést vagy más bonyolult feladatot, mert amíg ezen a függvényen belül vagyunk, csomagok veszhetnek el. Tehát a `netif_rx()` függvény nem tesz egyebet, mint a `softnet_data` tömbből kijelöli a csomag számára megfelelő várakozási sort, és a futtató processzortól függően kiválasztja a tömb indexét. Ezután ellenőrzi a várakozási sor állapotát, majd besorolja az öt lehetséges torlódási szint egyikére: `NET_RX_SUCCESS` (nincs torlódás), `NET_RX_CN_LOW`, `NET_RX_CN_MOD`, `NET_RX_CN_HIGH` (azaz alacsony, közepes vagy magas torlódási szint), illetve `NET_RX_DROP` (a rendszer a nagyon magas torlódási szint miatt a csomagot eldobja).
A `netif_rx()` függvény az esetleges hálózati szakadás elkerülése végett forgalomkorlátozó politikát tart érvényben, amely – ha a torlódás kritikus szintet ér el – a rendszert torlódásmentes állapotba juttatja vissza. Ennek egyik előnye, hogy segítségével a lehetséges DoS-támadások elháríthatók. Átlagos körülmények között a csomag végül a várakozási sorba (`__skb_queue_tail()`) továbbítódik, és a `__cpu_raise_softirq(cpuid, NET_IF_SOFTIRQ)` függvény hívódik meg. Ez utóbbi függvényhívás ütemezi a `softirq` futtatását.
Amikor a `netif_rx()` függvény befejezi munkáját, visszatérési értékével a hívó számára jelzi a pillanatnyi torlódási szintet. Ezen a ponton a megszakítási környezet véget ér, és a felsőbb szintű protokollok készen állnak arra, hogy a csomagot fogadják. A feldolgozás egy későbbi időpontra halasztódik, amikorra a megszakítások ismét engedélyezettek lesznek, és a feldolgozási idő kevésbé kritikus. Ez a késleltetett feldolgozási módszer a 2.2-es és 2.4-es rendszermagok között teljesen eltérő: a 2.2-es rendszermagok az úgynevezett `als felek`-kel dolgoznak, a 2.4-es rendszermagban a működés már a `softirq`-n alapul.

Alsó felek a softirq-k ellenében

Az `als felek`, vagyis az angol rövidítésnek megfelelően a `BH`-k részletes ismertetése meghaladja e cikk kereteit, így egy-két jellemző tulajdonságára csak pontokba szedve térünk ki.
Először is a `BH`-k tervezésekor az alapvető megszakítás környezetbeli politikának megfelelően elsődleges szempont volt, hogy ebben az üzemmódban a rendszermagnak a lehető legkevesebb számítást kelljen elvégeznie. Ezáltal ha bonyolultabb műveletet kellett valamilyen megszakításkérésre elvégezni, a megszakítási környezetben csak az ehhez szükséges `BH` került kijelölésre, anélkül, hogy bármilyen bonyolult művelet foglalta volna le a rendszert. Egy későbbi időpontban a rendszermag ellenőrizte a `BH`-maszkot, hogy eldöntse, valamelyik `BH` futásra vár-e, és amennyiben igen, még az alkalmazásszintű feladat előtt lefuttatta.
A `BH`-k egészen jól működtek, csupán egyetlen hátrányuk akadt: a felépítésük miatt ugyanaz a `BH` egy időben csak egyetlen processzoron futhatott. Ez több processzor között a több CPU-val működő SMP-rendszerekben mindennemű párhuzamosságot és feladatelosztást meggátolt, ugyanakkor a teljesítményt is jelentősen rontotta. A `softirq`-k a `BH`-k 2.4-es rendszermagbeli követőikkel, és az úgynevezett `tasklet`-ekkel együtt a rendszermag szoftveres megszakításcsaládjához tartoznak, azaz olyan kódok, melyeket igény esetén a rendszermag hajt végre, anélkül, hogy a feladatra rendelkezésre álló idő szigorúan meg lenne határozva. A legfontosabb különbség a `BH`-khoz képest, hogy egy `softirq` ugyanabban az időben akár több CPU-n is futhat. Több feladat egyidejű futása esetén a feladatok a rendszermag spinlockjaival hangolhatók össze.

A softirq-k működése

A `softirq`-k feldolgozómagja a `do_softirq()` függvényben található, a ***kernel/softirq.c*** fájlban. A függvény először egy bitmaszkot ellenőriz, majd ha a megfelelő `softirq` bit be van kapcsolva, meghívja a megfelelő kezelőeljárást.
A `NET_RX_SOFTIRQ` esetében a `net_rx_softirq()` függvény érdekel bennünket, ami a ***net/core/dev.c*** fájlban található.
A `do_softirq()` függvény három különböző helyről hívható meg a rendszermagban:

- a ***kernel/irq.c*** fájlban található `do_IRQ()` függvényből (ez az általános megszakításkezelő);
- rendszerhívások kilépésekor a ***kernel/entry.S*** fájlból;

- a `schedule()` függvényből a ***kernel/sched.c*** fájlból, mely a fő feladatütemező függvény.

Más szavakkal a `softirq` meghívódhat, ha a rendszer alkatrészmegszakítást kezelt, ha egy alkalmazásszintű program egy rendszerhívást kér, vagy ha új feladat (process) kerül végrehajtásra. Ily módon a `softirq`-kat olyan gyakran kezeli, hogy közülük egyiknek sem szükséges túl hosszan várakoznia. A kioldórendszer a már elavult `als felek` esetén pontosan ugyanígy működött.

A NET_RX softirq

Mint láthattuk, a csomagok a hálózatról a hálózati kártyán keresztül kerülnek a rendszerbe, majd egy várakozósorba jutnak, ahonnan a rendszer csak később dolgozza fel őket. Megfigyelhettük, miként folytatódik a feldolgozás a `net_rx_action()` függvényen keresztül. Most annak is eljött az ideje, hogy beletekintsünk ebbe a függvénybe.
E függvény feladata alapvetően nagyon egyszerű: a pillanatnyi CPU-hoz tartozó várakozósorról leemeli az első csomagot (`sk_buff`), majd végigfut a két csomagkezelő listán, és ennek alapján meghívja a megfelelő csomagkezelő függvényeket. Érdemes néhány szót ejteni erről a két listáról: vajon miként működnek, és hogyan épülnek fel? A két lista neve ***ptype_all*** és ***ptype_base***, és protokollkezelőket tartalmaznak az általános csomagokhoz, valamint bizonyos csomagtípusokhoz. A protokollkezelők önmagukat jegyzik be: vagy a rendszermag indulásakor, vagy bizonyos foglalati típusok létrejöttekor, megadva, hogy milyen protokolltípusok kezelésére képesek.
Az ehhez kapcsolódó függvény a `dev_add_pack()`, mely a ***net/core/dev.c***-ben található. Ez egy csomagtípus-szerkezetet (lásd az ***include/linux/netdevice.h*** fájlt) hoz létre, amely mutatót tartalmaz arra a függvényre, amelynek meg kell hívódnia, ha ilyen típusú csomag érkezik. A bejegyzés során minden csomagkezelő függvény belekerül a két lista valamelyikébe, azaz vagy a ***ptype_all*** listába (`ETH_P_ALL`-típusok esetén), vagy a ***ptype_base*** listába (minden egyéb `ETH_P_*` típus esetén).
A `NET_RX softirq` mindössze annyit tesz, hogy a csomagok protokolltípusához tartozó protokollkezelőket sorban meghívja. Először az általános protokollkezelők hívódnak meg (azaz a ***ptype_all*** protokollok), ahol az egyes protokollok fajtája nem számít, majd ezt követően az egyedi protokollkezelők következnek. Mint látni fogjuk, a `PF_PACKET` protokoll mindkét listába belekerülhet, attól függően, hogy az alkalmazás milyen foglalati típust választ. Másrészről az alap-IP-kezelő a második listába kerül, az azonosítója pedig `ETH_P_IP` lesz.
Amennyiben a várakozósor több csomagot is tartalmaz, a `net_rx_action()` az összes csomagon – amíg fel nem dolgozza a legnagyobb feldolgozható számú csomagot (`netdev_max_backlog`), vagy míg túl sok időt nem tölt el a csomagok feldolgozásával (a legtöbb rendszermag számára ez a túl sok idő egy pillanatot jelent, azaz körülbelül 10 ms-ot). Ha a `net_rx_action()`-nek nem sikerül befejeznie a feldolgozást, és csomagok maradnak a várakozósorban, akkor újra engedélyezi a `NET_RX_SOFTIRQ`-t, és a lemaradt csomagok egy későbbi alkalommal kerülnek feldolgozásra.

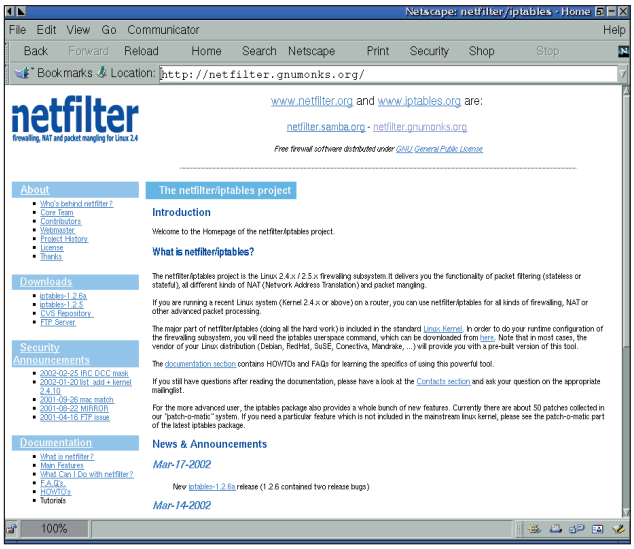
Az IP-csomagkezelő

Az IP fogadófüggvényre, nevezetesen az `ip_rcv()`-re (***net/ipv4/ip_input.c***), a rendszermag indulásakor bejegyzett csomagtípus-szerkezetre mutat az `ip_init()` függvény (a ***net/ipv4/ip_output.c*** forrásfájlban). Az IP bejegyzett protokolltípusa: `ETH_P_IP`. Ennek következtében az `ip_rcv()`-t a `net_rx_action()` függvény hívja meg a `softirq` feldolgozása során, ha egy `ETH_P_IP`-típusú csomag kerül feldolgozásra. Ez a függvény végez el minden kezdeti ellenőrzést az IP-csomagon, mely nagyjából annyiból áll, hogy ellenőrzi az IP-csomag épségét (az IP-ellenőrzőösszegét, az IP-fejlécmezőket, és a legkisebb jelentős csomagméretet). Ha a csomag megfelelőnek tűnik, az `ip_rcv_finish()` függvény hívódik meg. Csak mellékesen jegyzem meg, hogy ennek a függvénynek a meghívása keresztülmegy a Netfilter előútválasztó ellenőrzési pontján, melyet gyakorlatilag az `NF_HOOK` makróval valósítanak meg.
Az `ip_rcv_finish()` – még mindig az ***ip_input.c***-nél maradva – nagyrészt az IP-útválasztó szolgáltatásáért felelős. Ez ellenőrzi, hogy vajon a csomagot továbbítani kell-e egy másik gépre, vagy pedig helyben kell feldolgozni. Az első esetben elvégzi az útválasztást, és a csomagot a megfelelő hálózati eszköz felé irányítja; a második esetben a csomagot helyben kézbesíti. A szemfényvesztést az `ip_route_input()` függvény valósítja meg, mely az `ip_rcv_finish()` legelején hívódik meg, és amely eldönti a csomag következő feldolgozási pontját, a mutatót a megfelelő függvényre állítva be az `skb->dst->input`-ban. A helyben kézbesítendő csomagok esetén ez a mutató az `ip_local_deliver()` függvényre mutat. Az `ip_rcv_finish()` függvény egy `skb->dst->input()` függvényhívással zárul.
Ezen a ponton a csomagok már ténylegesen a felsőbb szintű protokollok felé utaznak. Az irányítás az `ip_local_deliver()`-hez kerül; ez a függvény felelős az IP-töredékek összeállításáért (abban az esetben, ha az IP-adat-

PF_PACKET foglalatok létrehozása

A `PF_PACKET` foglalat története akkor kezd el különbözni az általános foglalatokétól (pl. `PF_INET`), amikor a `socket()` rendszerhívás meghívódik. A `socket()` rendszerhívás (melyet a rendszermagban a `sys_socket()` kezel, a *net/socket.c* forrásfájlban található) a `sock_create()` függvényt használja fel, hogy eldöntse, milyen műveletek tartoznak a megadott protokollcsaládhoz. Ez az adat a `net_families` tömbből származik, melynek tartalma a rendszermag indulásakor dől el a `sock_register()` függvény hatására. Ezen a ponton a rendszermag képes arra, hogy meghívja az adott protokollcsaládhoz tartozó `create()` függvényt, mely létrehoz egy ennek megfelelő foglalati felépítést, és befejezi a létrehozási szakaszt.
`PF_PACKET`-ek esetén a `create()` függvény a *net/packet/af_packet.c* fájlban található *packet_create()* függvénynek felel meg. Egyebek között ennek a feladata, hogy a hivatkozásokat beírja a foglalati felépítésbe, a protokollra jellemző műveletekre. Ez nem egyezik meg a protokollcsaládra jellemző műveletekkel, mivel ez utóbbi csak egy mutatót tartalmaz a `create()` függvényre, míg a protokollra jellemző műveletek egy egész sor függvényhívásra tartalmaznak hivatkozást, amelyeket egy adott foglalaton végrehajthatunk (például `accept`, `connect`, `bind`, `ioctl`, `listen`, `sendmsg`, és így tovább – a teljes listához vess egy pillantást az *include/linux/net.h-ra*.)

gram szét van töredezve). Ezt követően a vezérlés átadódik az `ip_local_deliver_finish()` függvénynek. Mielőtt azonban ez meghívásra kerülne, egy újabb Netfilter-kapocs hajtódik végre (az `ip-local-ip`).
Az `ip_local_deliver_finish()` – mely az IP feldolgozási rétegben található utolsó függvény – hajtja végre a 3. réteg befejezéséhez szükséges még várakozó feladatokat. Az IP-fejléc adatait összerendezi, és továbbítja a 4. szinten lévő protokollnak, valamint megvizsgálja, hogy a csomag nem tartozik-e valamelyik nyers (raw) IP-foglalathoz, és ha igen, továbbítja a megfelelő kezelőnek (`raw_v4_input()`).



A nyers (raw) IP olyan protokoll, melynek segítségével az alkalmazásoknak lehetőségük nyílik arra, hogy olyan IP-csomagokat állítsanak össze, illetve fogadjanak közvetlenül, amelyek a 4. szinten még nem kerültek feldolgozásra. Ennek a protokollnak elsősorban hálózati alkalmazások veszik hasznát, ugyanis feladatuk elvégzéséhez szükségük lehet bizonyos csomagok elküldésére. Néhány jól ismert eszköz, mint a `ping` vagy a `traceroute` úgyszintén nyers IP-t használ a saját csomagjai felépítéséhez, hogy a csomagokhoz egyéni fejléc-adatokat adjon hozzá.
A nyers IP lehetőséget biztosít hálózati szintű protokollok felhasználói szinten történő kezelésére és létrehozására. Egy ilyen, felhasználói szinten létrehozott protokoll például az RSVP is (erőforrás lefoglaló protokoll). A nyers IP valójában nem más, mint a `PF_PACKET` megvalósítása az OSI (szabvány a nyílt rendszerek közti kapcsolattartásra) eggyel magasabb rétegében.
Ezt követően a csomagok tovább utaznak, és egy újabb rétegbe jutnak, ahol a rendszermag részéről egy újabb protokollkezelővel találják szemben magukat. Hogy ez a protokollkezelő melyik lesz, azt az IP-fejlécbe írt ***Protocol*** mező elemzésével dönti el a rendszer. Az ezen a szinten használt módszer, amely azt dönti el, hogy a csomag hogyan folytassa útját, nagyon hasonló ahhoz, mint amit a `net_rx_action()` függvénynél már megismerhettünk. Egy tábla `inet_protos` néven lett megadva, melyben az összes utólagos IP-kezelő protokoll be van jegyezve. A táblázat kulcsait a rendszer természetesen az IP-fejléc ***Protocol*** mezőjéből veszi. Ezt a táblázatot az `inet_init()` függvény tölti fel, mely a rendszermag indulásakor hajtódik végre (***net/ipv4/af_inet.c***). Ez a függvény hívja meg az `inet_add_protocol()`-t külön-külön a TCP-re, az UDP-re, az ICMP-re és az IGMP-re (csak akkor, ha a multicast, azaz a többrétegű címzés engedélyezett). A teljes protokolltáblázat a ***net/ipv4/protocol.c***-ben található.
Minden egyes protokoll számára létezik egy kezelőfüggvény: `tcp_v4_rcv()`, `udp_rcv()`, `icmp_rcv()` és `igmp_rcv()`, amelyek neve egyértelműen utal arra, hogy melyik függvény melyik protokollt kezeli. E függvények akkor kerülnek meghívásra, ha valamilyen feldolgozandó csomag vár rájuk. A visszatérési értékből dől el, hogy vajon egy ICMP- ***destination unreachable*** (Cél nem elérhető el) üzenetet kell-e küldeni válaszul a küldőnek. Ez olyankor fordul elő, amikor egy felsőbb szintű protokoll a beérkező csomagot egyetlen foglalathoz tartozóként sem ismeri el. Talán még emlékszel rá az előző cikkből, hogy az elfogás egyik alapelve az, hogy minden csomaghoz hozzájussunk – tekintet nélkül arra, hogy milyen kapura vagy milyen címre küldték őket. És itt (illetve az előbb említett `_rcv()` függvényeknél) ez az első akadály, mellyel szembetaláljuk magunkat.

Összegzés

Jelen pillanatban a csomag útjának már valamivel több mint felét megtette. Mivel szeretett újságunkban csak korlátozott hely állt rendelkezésünkre, a jövő hónapig a csomagot a 3. rétegben hagyjuk. Hátravan még a 4. réteg megismerése (TCP és UDP), a `PF_PACKET` kezelése és természetesen a csomagszűrő felépítése. Légy türelmes!

Linux Journal 2002. február, 94. szám

Gianluca Insolvibile
már a 0.99pl4-es rendszermag óta Linux-rajongó. Jelenleg hálózat- és digitális videokutatással, valamint -fejlesztéssel foglalkozik. A g.insolvibile@cpr.it címen érhető el.

További érdekességek

A Netfilterrel és az `ipfw`, IP Chains vagy IP Tablesszel kapcsolatban nagy segítségedre lehetnek Rusty „megbízhatatlan ismertetői” (Unreliable guides) a
➲ http://netfilter.gnumonks.org címen.

Biztonsági szempontból a Linuxvilág 2001. 10. számában található egy elemzés a Netfilterről: *David A. Bandel* A Netfilter megszelídítése című írásában.

Kapcsolódó címek

- ➲ http://www.tldp.org/LDP/nag2/
- ➲ http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2817396,00.html
- ➲ http://www.oreilly.com/catalog/linag2/
- ➲ http://www.unixreview.com/documents/urm0106p/
- ➲ http://kernelnewbies.org/documents/
- ➲ http://www.linux.org/books/various/admin2.html


linuxvilág
www.linuxvilag.hu