Eseménykezelő alrendszer Linuxhoz

A távközlési alkalmazások rendkívül alacsony késleltetést követelnek meg, és a bonyolultságuk szintén nem mindennapi. Egy aszinkron eseménykezelő rendszerre építkezve teljesíthetők az elvárások.

Ez az írás egy sorozat első része, amely egy új linuxos eseménykezelő alrendszert ismertet. Ez alkalommal a motivációt és az elvárásokat tárgyalom, illetve egy ilyen távközlési szintű Linux megvalósításának az előnyeit ecsetelem. Egy natív eseményalapú rendszer Linux-rendszermagon belüli megvalósítási lehetőségének tanulmányozása kutatási tervezetként 2001-ben kezdődött meg az Open Systems Labnél (az Ericsson Research egyik egysége), Montrealban. A cél az volt, hogy a Linuxot eseményvezérelt környezettel bővítsék, amely távközlési alkalmazásokban a meglévő megoldásoknál jobb teljesítmény tudna nyújtani.

A Linux operációs rendszer távközlési szintűvé történt fejlesztése iránt egyre nagyobb érdeklődés mutatkozott. Például az OSDL Carrier-Grade Linux munkacsoport (➲ http://www.osdl.org/projects/cgl) jelenleg is azon dolgozik, hogy kidolgozza mindazokat a követelményeket, amelyeket teljesítve a Linux a következő nemzedékbeli hálózatok megbízható, távközlési szintű kiszolgálórendszerévé válhat.

A távközlési alkalmazásoknál az operációs rendszernek rendkívül alacsony válaszidőt kell garantálnia, miközben a leállási ideje éves szinten legfeljebb öt perc lehet – ez 99,999 százalékos rendelkezésre állásnak felel meg –, és ebbe a vas és az operációs rendszer meghibásodásai és a programfrissítések egyaránt beletartoznak. Egy távközlési rendszer esetében ezek mellett további elvárásoknak is meg kell felelni, például a méretezhetőség, az elérhetőség és a teljesítmény tekintetében.

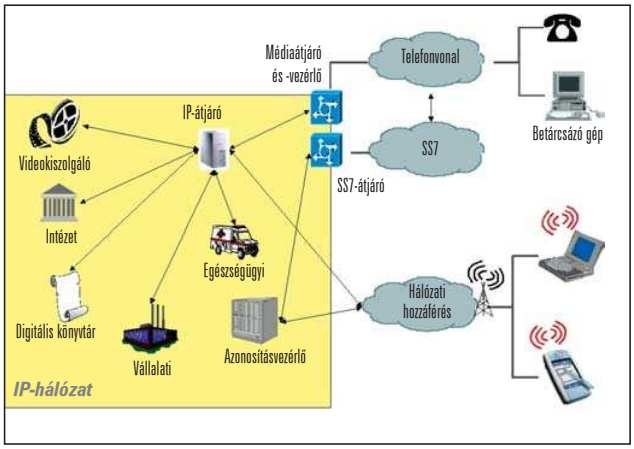
Az ilyen rendszereknek egyszerre több ezer kérést kell kezelniük olyan módon, hogy a teljesítményük még erős terhelés mellett sem csökkenhet. Az előfizetők egy-egy kérés kiadásakor elviselnek ugyan némi késleltetést, ám a féktelenül növekedő válaszidők számukra egyértelműen elfogadhatatlanok. Bizonyos tranzakciók kezelése több okból sem történik meg azonnal, a válaszra néhány milliszekundumnyi ideig vagy néhány másodpercig várni kell. Amíg azonban egy alkalmazás valamilyen válaszra vár, kevésbé tud foglalkozni a többi tranzakció feldolgozásával.

Számos, a Linux képességeinek javítását ígérő megoldás látott napvilágot, köztük többszálas megvalósításokat is találni lehetett, amelyeknek hatékony Posix-felületek készítésétől vagy a meglévő rendszermageljárások méretezhetőségének javításától remélték az áttörést. Mi úgy gondoljuk, hogy e megoldások egyike sem felel meg kellő mértékben a távközlési szintű kiszolgálók igényeinek.

Ahhoz, hogy eme álláspontunk megfelelő alátámasztást nyerjen, ebben a cikkben át szeretném tekinteni a távközlési hálózatok működését. Célom az, hogy ilyen módon tisztázzam a távközlési szintű operációs rendszerekkel szemben támasztott elvárásokat. Ennyi bevezetés után elmagyarázom egy natív aszinkron eseménykezelő rendszer előnyeit – ezek kivétel nélkül a távközlési iparág igényeinek magasabb szintű kielégítését szolgálják.

A távközlési ipar elvárásai

Távközlés, telekommunikáció – esetünkben ez azt jelenti, hogy két eszköz között létrejön egy telefonhívás, és vezetékes összeköttetésen hangot továbbítunk. Az *1. ábrán* a hagyományos



1. ábra A PSTN és az IP alapú hálózatok felépítése és együttműködése

nyilvános kapcsolt telefonhálózatok (Public Switched Telephone Network, PSTN) vázlata látható. A szolgáltatók eszközei a jelzéskezelésnek nevezett művelethalmazzal kezelik a két pont között felépülő telefonhívásokat. A PSTN-hálózatokban a jelzéskezelés a Signaling System 7 (SS7) protokollkészlettel történik. Az SS7 végzi a hívások útválasztását, ennek során – a megfelelő áramkörökön keresztül – egy útvonal jön létre a célkészülék felé. Az útvonal létrejötte után a két telefonkészülék összekapcsolódik, és megkezdődhet a hangtovábbítás. Az SS7 protokoll képes a hívások útválasztására és átirányítására, illetve bizonyos hibákat is kezel.

Az internet rugalmassága, költséghatékonysága és folyamatos növekedése számos távközlési céget arra késztet, hogy internetes alapokra helyezze át szolgáltatásait. Ez is hozzájárul ahhoz, hogy az IP alapú megoldások lassan a távközlési szolgáltatások új szabványaivá válnak. A két különböző típusú hálózat eltérő műszaki megoldásokat alkalmaz, együttműködésükhöz jelzéskezelésre és médiaátjárókra van szükség.

Az átjárók végzik az adatok fordítását a két eltérő típusú hálózat között. Egy SS7 átjáró például a jelzéseket a Stream Control Transmission Protocol (SCTP) segítségével ágyazza be, így teszi lehetővé IP alapú továbbításukat. A médiaátjáró feladata a PSTN-hálózat és az IP alapú hálózat között átadott hangok kódolása és dekódolása.

A jelzéskezelés és a médiaátjáró együttesen képes arra, hogy a PSTN-hálózatból érkező hívások számára elérhetővé tegye az IP alapú hálózatot. A jelzéskezelő átjáróknak beágyazást kell végezniük, megőrizve az SS7 üzenetek írásmódját és szemantikáját,

miközben azok továbbítása internetes protokollok segítségével, például IP felett SCTP vagy IP felett UDP megoldással történik. Egy jelzéskezelő kiszolgálónak az egy időben érkező kérések számának növekedésekor is változatlan teljesítményt kell nyújtania. Egy médiaátjáró segítségével a szolgáltatók megoldhatják az adatfolyamok átléptetését a PSTN és az IP alapú hálózatok között. Az, hogy hány kapcsolatot képesek fogadni, az adott eszköztől függ, a kapcsolatok száma felületenként egy és több ezer között változhat. A médiaátjáróknak nagyszámú kapcsolatot kell kezelniük, méghozzá valós időben.

A hitelesítő, engedélyező és számlázó (authentication, authorization, accounting – AAA) kiszolgálók felhasználói profilokból tartanak fenn adatbázisokat. Általában egy vagy két AAA kiszolgáló egy-egy szolgáltató sok millió előfizetőjének adatait kezeli. Nem szokatlan, hogy csúcsidőben másodpercenként több ezer hitelesítési kérés is érkezik. A kapcsolatok számának szeszélyes változása miatt meglehetősen nehéz előre tervezni. Az AAA kiszolgálók kritikus szerepet játszanak az IP alapú hálózathoz való hozzáférés vezérlésében, így esetükben a meghibásodások nem engedhetők meg. Nem szigorú valós idejű képességekkel kell rendelkezniük, válaszidejük a kérések túlnyomó részében nem haladhatja meg a néhány msec időtartamot.

A médiakiszolgálók különleges erőforrásokat és szolgáltatásokat nyújtanak a felhasználók számára, például videokonferencia és elektronikus levelezési lehetőséget, videokiszolgálókat és egyéb alkalmazásokat. Az ilyen távközlési rendszereknél fontos szempont a méretezhetőség, továbbá a processzorok vagy a felületek számának növelésével, illetve a sávszélesség bővítésével a kezelhető tranzakciók számának is lineárisan kell növekednie. A távközlési szolgáltatók mindig lineáris méretezhetőségben gondolkodnak, vagyis az egy tranzakcióra eső költség a kiszolgáló bővítésekor nem növekedhet.

Meghibásodás vagy terven kívüli leállás esetén a távközlési szintű berendezések önműködően helyreállítják magukat, illetve a redundáns hálózati erőforrások felhasználásával más kiszolgálóknak adják át a feladataikat. Az üzem közben végzett programfrissítések és az üzem közben cserélhető alkatrészek szintén a 99,999 százalékos rendelkezésre állást szolgálják.

A Linux már bizonyítottan üzembiztos, kiegyensúlyozott teljesítményre képes, így a távközlési szolgáltatók számára is figyelemre érdemes megoldást jelent. Ahhoz azonban, hogy a távközlési rendszerek alapelemévé válhasson, olyan összetevőkkel kell bővíteni, amelyek révén képes teljesíteni az iparág rendkívül szigorú kívánalmait.

Teljesítmény és módosuló felépítés: párban járnak

A hagyományos programozási modelleknél a programok összetevői explicit módon végzik egymással az összehangolást. Ha sok adatcserét kell végezni, ez a megszokott módszer. A fájlleírók változásainak figyelésére például a `select()` vagy a `poll()` hívást szokták használni. A `select` általános megvalósítása a leírók egész tömbjét végigolvassa. Méretezhetőségről ez esetben nem is beszélhetünk, hiszen egy-egy leíró módosulásának felismerése a tömb méretével arányosan növekvő ideig tart. Ez az alkalmazások késleltetéseinek növekedését okozza, illetve a rendszer általános teljesítményének romlását idézi elő. Ha átvizsgálunk egy leírótömböt, vagy valamilyen adatra várakozunk, akkor fogyasztjuk a processzor erőforrásait. A hatékony algoritmusok tervezésekor általánosan elterjedt ötlet a rendszeresemények aszinkron kezelése. A felhasználói térben futó alkalmazások eseményekről való értesítésére alkalmas megoldás többek közt a Posix AIO, az epoll vagy a BSD kqueue (lásd a *Kapcsolódó címeket*).

Amikor az ilyen módszerek hatékonyságát vizsgálják, sokszor azt az átlagos időt mérik, amelyik az eseménynek a rendszermag által történő észlelése és az alkalmazás által való tényleges kezelése között telik el. Az az oka, hogy ilyen jellegű vizsgálatokat végeznek, hogy a mikro-teljesítménypróbák ezekben az esetekben hamis eredményt adnának. Ezek a megoldások önmagukban lehetnek ugyan hatékonyak, ám más az alkalmazási területnek kevésbé megfelelő, például többszálú eljárásokkal társítva már kevésbé teljesítenek jól. Számos webkiszolgáló például egy szálkészletet (thread) alkalmaz, amely a kiszolgáló indításakor jön létre. Általában egy szál kezeli a bejövő kapcsolatokat, és mellette minden tranzakcióhoz létrejön egy-egy szál. Alacsonyabb számú bejövő kapcsolatnál ez a megközelítés megfelelő, ám amikor a terhelés magasabbra szökik, a hatékonysága jelentősen leromlik.

Többszálú alkalmazásokat akkor kell használni, ha magas fokon egyidejű futtatást kell megvalósítani a processzorért versengő objektumok között. Jól ismert példát szolgáltatnak a nagyteljesítményű, számításokat végző alkalmazások, amelyeknél fontos, hogy minden szál gyorsan fusson, ám viszonylag kisszámú szállal kell megbirkózni.

A szálak soros és szinkronműködést valósítanak meg, az egyidejű futtatást igénylő alkalmazásoknál gyakorlatilag szabványos megoldássá váltak. Csakhogy az alkalmazások tervezésekor vagy az összehangolások kezelésekor elkövetett hibák miatt könnyen torlódások keletkezhetnek a rendszerben, amelynek a teljesítménye látványosan csökkenhet. ***J. Ousterhout*** „Why Threads Are a Bad Idea" (Miért rossz ötlet a szálak használata?) című írásában megállapítja, hogy a szálak programozása nehéz, és a használatuk általában olyan alkalmazások létrejöttéhez vezet, amelyek nagyobb terhelés alatt meglehetősen rosszul teljesítenek.

A távközlési alkalmazásokban nincs versengés a szálak között. Egyidejűség akkor fordul elő, ha közös objektumokat, például osztott adatszerkezeteket kell kezelni. Ezeknél az alkalmazásoknál a szálak az osztott adatokhoz való egyidejű hozzáférés lehetőségének biztosításához szükségesek.

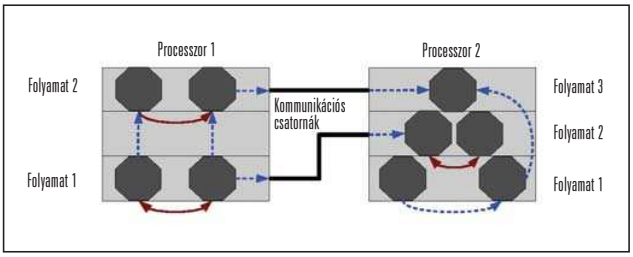
A távközlési alkalmazásoknál másodpercenként több ezer tranzakciót és több száz egyidejű kapcsolatot kell kezelni ugyanazzal a processzorral. Ezek mellett rendszereseményeket, például adatbázis-hozzáféréseket, alkalmazáshibákat, túlterhelésjelzéseket, riasztásokat, a rendszerösszetevők állapotának változásait is kezelni kell. Egy alkalmazás futását egy rendszeren több ezer esemény létrejötte kísérheti, vagyis az eseményeknek szálakkal való kezelése nem volna hatékony.

A hagyományos aszinkron módszerek úgy próbálják kezelni ezt a méretezhetőségbeli hiányosságot, hogy megkísérlik megakadályozni az alkalmazásokat abban, hogy feleslegesen várakozzanak, illetve a linuxos epollhoz hasonlóan segítik a működő leírók hatékony felismerését. Ezeknek a módszereknek a hatóköre sajnos a fájlleírókra korlátozódik, így a figyelembe veendő eseményeknek csak a töredékét képesek kezelni. Nagyszámú szálat indítva – mint az események kezelése miatt az a webkiszolgálók esetében is történik – torlódás alakulhat ki a rendszerben, a helyzet pedig tovább súlyosbodhat.

Eseményalapú rendszerek

Az összetett, osztott programrendszerek fejlesztéséhez olyan eljárásokat kell megvalósítani, amelyek képesek a rendszer rendelkezésre álló erőforrásainak maradéktalan kiaknázására. Ígéretes ötletnek tűnik, ha a kérdés megoldására egy eseményalapú alrendszert próbálunk meg Linux alá készíteni. Egy ilyen bővítménnyel valódi együttműködést lehetne megvalósítani az

operációs rendszer és az alkalmazások között. Bizonyos összetevők segítségével jelentkezni lehetne a különféle eseményekre, amelyekről később aszinkron módon, a megfelelő kezelők végrehajtása révén lehetne értesülni.
Ha összehasonlítjuk a jelzés- és eseménykezelőket, az utóbbiak sokkal hasznosabbaknak bizonyulnak, hiszen az adatokat azonnal át is adják az alkalmazásnak. Egy aszinkron eseménykezelő rendszert alapvetően általános felhasználói kezelők – amelyeket a rendszer hív meg –, illetve periodikus figyelő összetevők, például időmérők készítéséhez lehet felhasználni.
Az első eset különösen érdekes, ha egy alkalmazás nem tudja, hogy mikor történik meg egy esemény. Ha az eseményeket aszinkron módon fogadja, az alkalmazás olyan módon is elvégezheti a szükséges műveleteket, hogy nem foglalkozik az adatok beszerzésével – ugyanis átadott értékek formájában kapja meg őket.
Folytattak már vizsgálatokat arra vonatkozólag, hogyan lehetne létrehozni egy gyors üzenetátadó rendszert, amely az aszinkron eseményeknél megismert alapelvek alapján működne. Például az aktív üzenetek (lásd a *Kapcsolódó címeket*) aszinkron módon kerülnek végrehajtásra, a fogadó folyamat vermében. A felbukkanó szálaknál minden kezelőhöz külön futtatási szál jön létre, az egyszálú felhívásoknál pedig minden egyes processzoron kinevezett szál keletkezik. Az AEM olyan rohamosan fejlődő megoldás, amely natív környezetet ad a ténylegesen aszinkron működést megkövetelő alkalmazások fejlesztéséhez. Az AEM felhasználásával például natív aszinkron foglalatfelületet készítettünk TCP-protokollhoz. AEM környezetben a bejegyzés végrehajtásakor kell kiválasztani a kezelőt, ennek futtatása vagy a pillanatnyi folyamaton belül, vagy új szálként történik meg. Léteznek más kutatási tervezetek is, amelyek hasonló elgondolások alapján próbálják a nagy terhelésnek kitett webkiszolgálók viselkedését javítani (a *Kapcsolódó címek* között lásd az „A Scalable and Explicit Event Delivery Mechanism for UNIX” című írást).
Az eseményközpontú szemlélet legfontosabb előnye az, hogy az események és a szálak kezelését egyetlen alrendszerrel oldja meg. Ebből következik, hogy teljes ellenőrzést enged az erőforrások felhasználása felett.
Az eseményalapú rendszereknél mindig a teljesítmény javítása a cél. Ha az eseménykezelést leválasztjuk az alkalmazásról, az a különféle memóriafoglalási minták alkalmazása vagy az ütemező döntéseinek befolyásolása révén magasabb fokú lokalitást tesz lehetővé. A nem szigorúan valós idejű válaszadásokat például olyan módon lehet megvalósítani, hogy a folyamatok fontosságát (priority) a várakozó események száma határozza meg.
Az egyre nagyobb teret nyerő szemlélet a többszálú rendszerek bonyolultságához képest sokkal egyszerűbb és természetesebb programozási stílust teremt. Hatékonysága látványosan megmutatkozik a többrétegű programrendszerek fejlesztésekor, amelyeknél az egyes rétegek a felsőbb szintűek számára nyújtanak szolgáltatásokat. Az ilyen jellegű rendszerek az osztott alkalmazások területén teljesen megszokottak.
A 2. *ábrán* egy jellegzetes elosztott, eseményvezérelt modellre épülő alkalmazás működése látható. Több programösszetevőből áll, és az alkalmazás minden rétegét egy folyamat valósítja meg. Az osztott alkalmazásoknál azonos és eltérő szintek között, távolra és helyben egyaránt jelentős adatforgalommal kell számolni.
Ezeknek az alkalmazásoknak sokszor olyan szolgáltatásokat kell nyújtaniuk, amelyek a világ bármely pontján kiváló teljesítménnyel üzemeltethetők. Alapvető, hogy az alkalmazások kihasználják a rendelkezésre álló eszközerőforrásokat, hogy a rendszer



2. ábra Egy többrétegű, osztott alkalmazás, amit két processzoron futó, eseményalapú modell feldolgozásával terveztek. Minden réteg egyetlen szálból áll, az alkalmazás összetevői közötti adatcsere szinkron (folytonos vonalak) és aszinkron (szaggatott vonalak) lehet

képességeit tekintve lineáris méreteződésre legyenek képesek. Már a programok tervezésekor gondoskodni kell arról, hogy holtpontok vagy versenyhelyzetek ne alakulhassanak ki az összetevők között. Az ilyen jellegű tervezési hibák végzetes hatást gyakorolnának a rendszerre. Többszálas megközelítést alkalmazva nehéz az ilyen helyzeteket elhárítani, a nagyszámú lehetséges élethelyzet miatt ugyanis túlságosan bajos felismerni és feloldani őket. Eseményalapú megközelítést alkalmazva kevesebb meghibásodási pont kerül a rendszerbe, mivel az aszinkron módon indított szálak számát folyamatosan kézben tartjuk. Könnyebben biztosítható a kezelők futtatásának osztatlansága is, hiszen az alrendszer a rendszermag részét képezi.
A gépek erőforrásai korlátozottak, az elindítható folyamatok száma ugyancsak véges. Bejegyzéskor mód nyílik a futtatni kívánt kezelő kiválasztására. Így olyan alkalmazások készíthetők, amelyek a terhelés növekedésekor is üzembiztosak maradnak. Az alkalmazások szempontjából elsősorban a soros és az aszinkron kód keverésének lehetőségét érdemes kiemelni.
Ha ezt sikerül elérni, akkor olyan alkalmazásokat lehet tervezni, amelyek képesek mindkét szemlélet jó tulajdonságainak a kihasználására.

KAPCSOLÓDÓ CÍMEK

G. Banga és mások „A Scalable and Explicit Event Delivery Mechanism for UNIX”, USENIX Annual Technical Conference, 1999. június, 253–265. oldal
R. Bhoedjang és mások „Friendly and Efficient Message Handling”, 29. Annual Hawaii International Conference on System Sciences, Hawaii, USA, 1996. január, 121–130. oldal
J. Lemon „Kqueue: A Generic and Scalable Event Notification Facility”, FreeBSD Project.
D. Libenzi „/dev/epoll” ➲ http://www.xmailserver.org/linux-patches/nio-improve.html
J. Ousterhout „Why Threads Are a Bad Idea”, USENIX Technical Conference, 1996. január 25.
T. von Eicken és mások „Active Messages: A Mechanism for Integrated Communication and Computation” 19. International Symposium on Computer Architecture, Ausztrália, 1992. május, 256–266. oldal
Linux AEM honlap ➲ http://aem.sourceforge.net
The Open Group, „Posix Asynchronous I/O”, The Single UNIX Specification ➲ http://www.opengroup.org
OSDL CGL (Open Source Development Lab – Carrier-Grade Linux) ➲ http://www.osdl.org/projects/cgl

Eseményalapú keretrendszert alkalmazva a szolgáltatók a szolgáltatást csak a legkisebb mértékben zavarva végezhetnek dinamikus erőforrás-átcsoportosítást. Az eszközök cseréjét és a programok frissítését üzem közben, a rendszer újraindítása nélkül kell elvégezni. Az osztott alkalmazások nagyszámú, egymással párbeszédet folytató összetevőből épülnek fel, az ilyen programok frissítése pedig roppant kényes művelet. A távközlési rendszereknél az összes szolgáltatás rendelkezésre állásának el kell érnie a 99,999 százalékot. A szolgáltatások nem állíthatók le a karbantartások idejére, hiszen ez más szolgáltatói rendszerek működését is befolyásolná, az előfizetőkről nem is beszélve. A programfrissítéseket lépcsőzetesen kell elvégezni. Az eseményalapú alrendszerek lehetővé teszik, hogy az osztott alkalmazásokat ilyen képességekkel ruházzuk fel. Mint a 2. ábrán is látható, a programrétegek között nincs közvetlen függőség, ha az adatcserék aszinkron módon folynak, tehát az alkalmazás egyes részeinek cseréje a teljes rendszer megzavarása nélkül is végrehajtható.

Összegzés

Egy eseményalapú alrendszer alapján újfajta programozási modellt állíthatunk fel, amelynek a segítségével a fejlesztők egyedülálló, nagy teljesítményű környezetet teremthetnek a folyamatok aszinkron végrehajtásához. Természetesen a soros programozási stílustól gyökeresen eltérő szemléletet kell elsajátítani, ám a programfejlesztés szempontjából sokkal jobban áttekinthető keretrendszer bőségesen kárpótol mindezért. A bonyolult programösszetevők egybeépítése és együttműködése is egyszerűbbé válik.

Egy ilyen alrendszer legfontosabb erénye az, hogy képes az aszinkron és a szinkron programkód egyesítésére ugyanabban az alkalmazásban – sőt a kétféle típus akár ugyanazon az eljáráson belül is elegyíthető. A kevert modellt alkalmazva – a tényleges környezettől függően – mindkét szemlélet előnyei kihasználhatók. Az újfajta modellt elsősorban biztonságos alkalmazások fejlesztésekor, illetve küldetéskritikus alkalmazások hosszú távú támogatásakor érdemes használni. Következő írásomban bemutatom, hogy az AEM-nek a Linux-rendszermagban történt megvalósításával hogyan vált elérhetővé az új modell támogatása, illetve mi módon használhatjuk fel alkalmazások fejlesztésekor.

Köszönetnyilvánítás

Köszönettel tartozom az Open Systems Lab munkatársainak, amiért átnézték az írásomat és segítették a megjelenését; ***Laurent Marchand***-nak az Ericsson Research Canadától hasznos megjegyzéseiért, illetve ***Philippe Meloche***-nak, a Sherbrooke University hallgatójának.

Linux Journal 2003. július, 111. szám

Frederic Rossi

(frederic.rossi@ericsson.ca) Az Ericsson Research Open Systems Lab laboratóriumának kutatója a kanadai Montrealban. Részt vesz azokban a kutatásokban, amelyeknek eredményeképpen olyan rendszermag-összetevőket készítenek, amelyek lehetővé teszik egy távközlési szintű operációs kifejlesztését.