Squid alapú forgalomszabályozó és felügyeleti rendszer

Amikor a hálózatunkon folyó forgalom túlnyomó része webes forgalom lett, egyetemünk a nyílt forrású Squid gyorsítótárazó programra építve üzembe helyezett egy a hozzáférések követésére és korlátozására alkalmas rendszert.

Bármilyen szervezet számítógépes hálózatáról is legyen szó, az internetelérés az egyik legfontosabb és leginkább igényelt szolgáltatás. ***Olifer*** és ***Olifer*** a ***Computer Networks: Principles, Technologies and Protocols*** című könyvükben azt írják, hogy az elmúlt 10-15 évben a 80/20-as arány a belső és a kimenő forgalom között megfordult, és jelenleg a forgalom 80 százaléka kifelé irányul (lásd az internetes forrásokat). A hozzáférés sebessége, a szolgáltatások száma és a rendelkezésünkre álló tartalom mennyisége folyamatosan növekedik. Az internetelériéshez fűződő felhasználói hozzáférés-vezérlés kérdése ezzel párhuzamosan egyre fontosabbá válik. Maga a probléma természetesen nem új, ám bizonyos vonatkozásai az idők során megváltoztak. Írásunkban az elérhető korszerű megoldásokat tekintjük át, példaként a ***Bashkir State Pedagogical University (Baskíri Állami Pedagógiai Egyetem, BSPU)*** hálózatát véve.
Elsőként határozzuk meg az internet elérését szabályozó és felügyelő rendszerrel szemben támasztott elvárásokat:

- Felhasználói fiókok támogatása és kezelése
- Felhasználói forgalom számlázása és szabályozása
- Háromféle mód a felhasználói forgalom szabályozására: havi, heti és napi
- A mozgó felhasználók támogatása (ők az egyes alkalmakkor különböző számítógépeket használnak az internet elérésére; ilyenek például a hallgatók)
- Napi és heti kimutatások a rendszer állapotáról, jelentések a webes és elektronikus levélforgalomról
- Web alapú kimutatások és rendszerfelügyelet

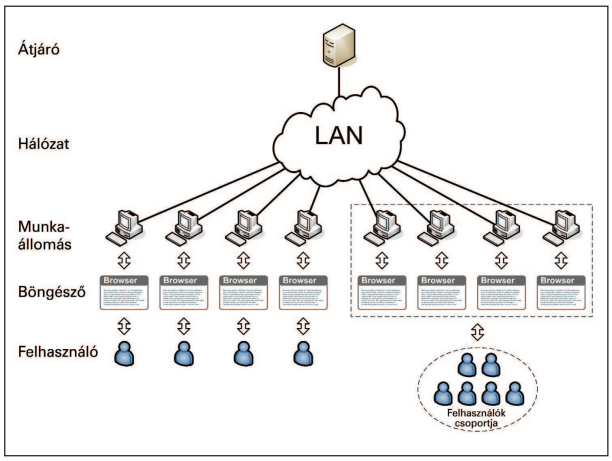
Láthatóan mindezek az elvárások semmilyen módon nem hatnak ki a rendszer megvalósításának fázisára, vagyis képzeletünknek sem jelentenek korlátokat. A problémát tehát általános jelleggel szemléltük, majd kidolgoztuk a megoldását. Írásunk további részeiben a végső döntésünk felé vezető ötleteket és megfontolásokat ismertetjük.

A kérdéskör általános elemzése

A legnépszerűbb szolgáltatás, a ***World Wide Web (WWW)*** példáján röviden tekintsük át, hogy az internetelérés folyamata milyen lépésekből áll:

1. A felhasználó elindítja a böngészőprogramot, majd beírja a kívánt ***URL***-t.
2. A böngésző vagy közvetlen kapcsolatot létesít a ***WWW***-kiszolgálóval az átjárón keresztül, utóbbi akár címfordítást vagy más csomagmanipulációs műveletet is végezhet, vagy proxykiszolgálón keresztül csatlakozik, mely gondosan elemzi az ügyfél kérését és ellenőrzi, hogy a kért adatok szerepelnek-e a gyorsítótárában. Ha nem rendelkezik a kért adatokkal, esetleg azok elavultak, akkor a proxykiszolgáló a saját nevében kapcsolódik a ***WWW***-kiszolgálóhoz.
3. A kapott adatokat megkapja az ügyfél.
4. A böngészőprogram lezárja vagy ébrenléti állapotba állítja a kapcsolatot.

Az *1. ábrán* az internetelérés folyamatát szemléltettük. A folyamat fő résztvevői a felhasználó, az ügyfélprogram (böngésző és operációs rendszer), a munkaállomás vagy egyéb ügyfélhardver, a hálózati eszközök és az átjáró vagy proxykiszolgáló. A hálózatra további, például felhasználóhitelesítést végző kiszolgálók is csatlakozhatnak, például ***Microsoft Windows*** tartományvezérlők, ***OpenLDAP***- vagy ***NIS***-kiszolgálók.



1. ábra A felhasználók internetelérésének folyamata

Mint az 1. ábra is szemlélteti, a felhasználók és a munkaállomások között egy-egy és több-több megfeleltetés egyaránt lehet. Az egyetem munkatársai például a legtöbb esetben saját számítógéppel rendelkeznek.
Elsődleges céljaink a felhasználói forgalom számlázása, a felhasználók hitelesítése, a felhasználók hozzáférésének ellenőrzése, a felügyelet és jelentések készítése.
Ezek a célok egymástól nagymértékben függetlenek, és mindegyiket többféle módon is el lehet érni. A hitelesítés, a forgalomszámlázás és a hozzáférés-vezérlés feladata a fenti séma elemeinek bármelyikéhez hozzárendelhető, a legjobb megoldás azonban kétségtelenül az lenne, ha az összes feladat elvégzése egyetlen helyen összpontosulna.
A hozzáférés-vezérlés ügyfél- és kiszolgálóoldalon egyaránt megvalósítható. Az ügyféloldali hozzáférésvezérlés különleges, a felhasználók hitelesítésére is képes ügyfélprogram használatát teszi szükségessé.
A hozzáférés-vezérlés kiszolgálóoldali megvalósításának kétféle módja van: tűzfal vagy proxykiszolgáló alkalmazása. A tűzfal alapú megoldás nem alkalmas a felhasználók hitelesítésére, hiszen a hálózati csomagok csak *IP*-címeket tartalmaznak, amelyek semmilyen módon nem kötődnek a felhasználónevekhez. Ha tűzfalat akarunk használni, akkor erre kétféle megoldás kínálkozik: saját felhasználóhitelesítést alkalmazó *VPN* használata vagy dinamikus felhasználó/IP-cím hozzárendelések megadása. Ezeket a megoldásokat külső eszközökkel valósíthatjuk meg.
A legegyszerűbb megoldás mégis egy proxykiszolgáló használata, mely képes a böngészőn keresztül végzett felhasználóhitelesítésre. A böngésző alapú hitelesítésnek háromféle módszere létezik:

- Alapszintű hitelesítés – Egyszerű és széles körben alkalmazott megoldás, melyet az internetes böngészők és proxykiszolgálók túlnyomó része támogat. Fő hátránya, hogy a felhasználó jelszava a hálózaton keresztül titkosítatlanul kerül továbbításra.

- A kivonat alapú hitelesítés egy megbízhatóbb megoldás, hátránya a szoftveres támogatás hiánya.

- Az *NTLM* alapú hitelesítés csak a *Microsoft* termékekre épülő hálózati rendszerekben alkalmazható. A windowsos munkaállomásokat tartalmazó hálózatokban ugyanakkor nemcsak elfogadható, de kifejezetten előnyös, márpedig tudomásunk szerint *Oroszországban* ezek a rendszerek terjedtek el leginkább. A fő előny itt az, hogy lehetőség nyílik a proxyhitelesítés összevonására a windowsos és a *Samba* alapú tartományvezérlőkre szóló hitelesítéssel.

A körülmények elemzésével és néhány ötlet nyomán kétféle rendszert dolgoztunk ki:

1. *VPN*, *PPTP* használatával, a tűzfal beépített szolgáltatásaira építve. A *VPN*-kiszolgáló hagyományosan *FreeBSD* volt, ezért az *ipfw* tűzfalfelületet és az *mpd PPTP*-kiszolgálót alkalmaztuk. A forgalomszabályozást az ingyenesen elérhető *NetAMS* rendszer végezte.

2. *Squid*-alapú hozzáférés-vezérlési és felügyeleti rendszer.

Az első rendszert *Vladimir Kozlov* fejlesztette ki, feladata az egyetem az internet elérésére állandó gépet használó munkatársai által használt kapcsolatok biztosítása. Legnagyobb hátránya az, hogy ügyféloldali *VPN*-telepítést igényel. Ez elég komoly gondot jelent, ha a hálózat elemei szétszórtan helyezkednek el, és a felhasználók kevésbé jártasak a számítógép használatában.
A második megoldás *Tagir Bakirovhoz* kötődik, segítségével történik az egyetemi felhasználók túlnyomó részének csatlakoztatása – ők azok, akik nem rendelkeznek állandó géppel. Ennek kapcsán a fő problémát a fejlesztés bonyolultsága jelentette; a továbbiakban ennek részleteit tárgyaljuk.

Squid-alapú internetes hozzáférés-vezérlési és felügyeleti rendszer

Mindenek előtt szeretnénk felhívni rá a figyelmet, hogy az itt szereplő elérési utak mindig a *Squid* forrásának alapkönyvtárához viszonyítva értendők, ami a mi esetünkben a */usr/local/src/squid-2.5STABLE7/* volt. A *Squid* beszerzésével, lefordításával és használatával kapcsolatosan a *Squid* webhelyén lehet részletes útmutatást találni. Fussuk át a *Squid* néhány jellemzőjét, ezek a *Squid Programming Guide*-ból származnak.
A *Squid* egy egyfolyamatos proxykiszolgáló. Minden ügyfél *HTTP*-kéréseit a főfolyamat kezeli. Futása lényegében visszahívó függvények sorozata. A visszahívó függvény végrehajtására akkor kerül sor, amikor a be/kivitel elvégezhető vagy valamilyen más esemény történt. Amikor a visszahívó függvény végzett, bejegyzi a következő be/kiviteli művelet visszahívó függvényét.
A *Squid* alapját a *select(2)* és a *poll(2)* rendszerhívás adja, melyek be/kiviteli eseményekre várakoznak fájlleírók egy csoportján. A *Squid* ezekkel végzi el a megnyitott fájlleírók be/kiviteli műveleteit. A `select()` rendszerhívást a `comm_select()` függvény adja ki, mely a teljes `fd_table[]` tömböt letapogatja, kezelő-

függvényeket keresve. Az eljárás minden kész leíróhoz meghívja a megfelelő kezelőt. A kezelőfüggvények bejegyzése a `commSetSelect()`, a lezáró kezelők meghívása pedig a `comm_close()` függvénnyel történik. A lezáró kezelők feladata a fájlleírókhoz rendelt adatszerkezetek felszabadítása. Ez az oka annak, hogy minden hívássorozatot a `comm_close()` meghívásának kell zárnia.
Érdekes ***Squid***-szolgáltatás az ügyfél/IP-cím adatbázis támogatása. Kódja a ***src/client_db.c*** fájlban található. Alapeleme egy kivonat alapján indexelt tábla (`client_table`), mely `ClientInfo` adatszerkezetekre vezető mutatókat tartalmaz. Ezek az adatszerkezetek különböző adatokat tartalmaznak a ***HTTP***-ügyfélről és az ***ICCP*** proxykapcsolatokról (mint például kérés-, forgalom- és időszámlálók). A ***src/structs.h*** fájl vonatkozó kódrészlete:

```
struct _ClientInfo {
    /* elsőnek kell lennie */
    hash_link hash;
    struct in_addr addr;
    struct {
        int result_hist[LOG_TYPE_MAX];
        int n_requests;
        kb_t kbytes_in;
        kb_t kbytes_out;
        kb_t hit_kbytes_out;
    } Http, Icp;
    struct {
        time_t time;
        int n_req;
        int n_denied;
    } cutoff;
    /* a jelenleg létrejött kapcsolatok
    ➥száma */
    int n_established;
    time_t last_seen;
};
```

Néhány fontos átfogó és helyi függvény az ügyféltábla kezeléséhez:

- `clientdbInit()` – Átfogó, az ügyféltábla kezdeti értékadását végző függvény.
- `clientdbUpdate()` – Átfogó függvény, szükség szerint a táblában lévő rekordot frissíti vagy új rekordot hoz létre.
- `clientdbFreeMemory()` – Átfogó függvény, törli a táblát és felszabadítja a hozzárendelt memóriát.
- `clientdbAdd()` – Helyi függvény, a `clientdbUpdate()` hívja meg; hozzáadja a rekordot a táblához, illetve ütemezi a szemétrekordokat összegyűjtő eljárást.
- `clientdbFreeItem()` – Helyi függvény, a `clientdbFreeMemory()` függvény hívja meg, és egyetlen rekordot távolít el a táblából.
- `clientdbSheduledGC()`, `clientdbGC()` és `clientdbStartGC()` – Helyi, a szemétrekordokat összegyűjtő eljárást megvalósító függvények.

Párhuzamba állítva a rendszerrel szemben támasztott követelményeket és a meglévő ügyféladatbázis által nyújtott lehetőségeket, kijelenthetjük, hogy az alapszolgáltatások egy része máris megvalósult, kivéve az ügyfelek felhasználóneve szerinti indexelést.
A meglévő ügyfélstatisztika-adatbázis további súlyos hiányossága, hogy az adatok frissítése csak azt követően történik meg, hogy az ügyfél a teljes kért tartalmat megkapta.
Munkánk során egy másik párhuzamos és független, ügyfél/felhasználó adatbázist készítettünk, kisebb módosításokkal a ***src/client_db.c*** fájlban található kódot felhasználva. A felhasználói statisztikák a `ClientInfo_sb` adatszerkezetbe kerülnek. Az ***src/structs.h*** fájl vonatkozó része:

```
#ifdef SB_INCLUDE
#define SB_CLIENT_NAME_MAX_LENGTH 16
struct _ClientInfo_sb {
    /* elsőnek kell lennie */
    hash_link hash;
    char *name;
    unsigned int GID;
    struct {
        long value;
        char type;
        long cur;
        time_t lu;
    } lmt;
    /* HTTP-kérések számlálója */
    int Counter;
};
#endif
```

Az ügyféladatbázis kezelését a következő – az előbbiekhez roppant hasonló – átfogó és helyi függvények végzik:

- `clientdbInit_sb()` – Átfogó, az ügyféltábla kezdeti értékadását végző függvény.
- `clientdbUpdate_sb()` – Átfogó függvény, mely frissíti a táblában lévő rekordot, a korlát elérésekor leválasztja az ügyfelet, illetve szükség esetén a `clientdbAdd_sb()` meghívásával hozzáadja az új ügyfelet.
- `clientdbEstablished_sb()` – Átfogó függvény, mely megszámlálja az ügyfélkéréseket, és rendszeres időközönként beírja a megfelelő rekordot a fájlba, a korlát elérésekor leválasztja az ügyfelet, illetve szükség esetén a `clientdbAdd_sb()` függvény meghívásával hozzáadja az új rekordot.
- `clientdbFreeMemory_sb()` – Átfogó függvény, törli a táblát és felszabadítja a hozzárendelt memóriát.

1. kódrészlet A clientWriteComplete() és a clientReadRequest() függvény egyes részei az src/client_side.c fájlból

```
static void
clientWriteComplete(int fd,
                    char *bufnotused,
                    size_t size,
                    int errflag,
                    void *data)
{
    clientHttpRequest *http = data;
...
    if (size > 0)
    {

kb_incr(&statCounter.client_http.kbytes_out,
                size);
/*-Itt kezdődik az SB rész---------------------*/
#ifdef SB_INCLUDE
    if (http->request->auth_user_request)
    {
        if ( authenticateUserRequestUsername(
                http->request-
>auth_user_request) )
            if (!clientdbUpdate_sb(
      authenticateUserRequestUsername(
          http->request->auth_user_request),
          size) )
            {
                comm_close(fd);
                return;
            }
    }
#endif
/*---------------------------------------------*/
        if (isTcpHit(http->log_type))
            kb_incr(
      &statCounter.client_http.hit_kbytes_out,
      size);
    }
...
}
...
static void
clientReadRequest(int fd, void *data)
{
    ConnStateData *conn = data;
    int parser_return_code = 0;
    request_t *request = NULL;
    int size;
    void *p;
    method_t method;
    clientHttpRequest *http = NULL;
    clientHttpRequest **H = NULL;
    char *prefix = NULL;
    ErrorState *err = NULL;
    fde *F = &fd_table[fd];
    int len = conn->in.size - conn->in.offset - 1;
...
    /* A kérés törzsének feldolgozása, ha van
    ➥ilyen */
    if (conn->in.offset > 0 &&
        conn->body.callback != NULL)
    {
        clientProcessBody(conn);
    }
    /* A következő kérés feldolgozása */
    while (conn->in.offset > 0 &&
           conn->body.size_left == 0)
    {
        int nrequests;
        size_t req_line_sz;
...
        /* Kérés feldolgozása */
        http = parseHttpRequest(conn,
            &method,
            &parser_return_code,
            &prefix,
            &req_line_sz);
        if (!http)
            safe_free(prefix);
        if (http) {
...
            if (request->method ==
            ➥METHOD_CONNECT)
            {
                /* Kérések beolvasásának
                ➥leállítása... */
                commSetSelect(fd,

COMM_SELECT_READ,
                                    NULL,
                                    NULL,
                                    0);
                clientAccessCheck(http);
/*-Itt kezdődik az SB rész---------------------*/
#ifdef SB_INCLUDE
                if(http->request
                ➥->auth_user_request)
                {
                    if (
      authenticateUserRequestUsername(
          http->request->auth_user_request
      )!=NULL)
                    {
                        if(!clientdbCount_sb(
          authenticateUserRequestUsername(
              http->request
              ➥->auth_user_request)))
```

1. kódrészlet (folytatás)

```
                    {
                        comm_close(fd);
                        return;
                    }
                }
            }
#endif
/*-------------------------------------------*/
                break;
            } else {
                clientAccessCheck(http);
/*-Itt kezdődik az SB rész--------------------*/
#ifdef SB_INCLUDE
                if(http->request
                ➥->auth_user_request)
                {
                    if (
      authenticateUserRequestUsername(
           http->request->auth_user_request
      )!=NULL)
                    {
                        if(!clientdbCount_sb(
      authenticateUserRequestUsername(
           http->request
           ➥->auth_user_request)))
                    {
                        comm_close(fd);
                        return;
                    }
                }
            }
#endif
/*-------------------------------------------*/
/* while offset > 0 && body.size_left
➥== 0 */
                continue;
            }
        } else if (parser_return_code == 0) {
...
/* while offset > 0 && conn->body.size_left
➥== 0 */
    }
...
}
```

- `clientdbAdd_sb()` – Helyi függvény, a `clientdbUpdate_sb()` hívja meg; hozzáadja a rekordot a táblához, illetve ütemezi a szemétrekordokat összegyűjtő eljárást.

- `clientdbFlushItem_sb()` – Helyi függvény, a `clientdbEstablished_sb()` és a `clientdbFreeItem_sb()` hívja meg; a megadott rekordot írja ki a fájlba.

- `clientdbFreeItem_sb()` – Helyi függvény, a `clientdbFreeMemory_sb()` függvény hívja meg, és egyetlen rekordot távolít el a táblából.

- `clientdbSheduledGC_sb()`, `clientdbGC_sb()` és `clientdbStartGC_sb()` – Helyi függvények, a szemétrekordokat összegyűjtő eljárást valósítják meg.

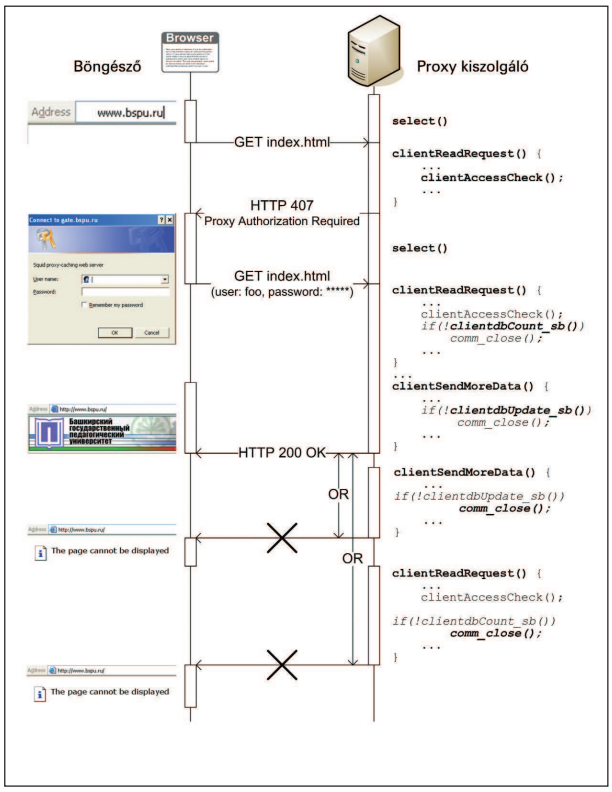
Az ügyféladatbázis kezdeti értékadásának és elengedésének megvalósítása az eredeti táblához hasonlóan az ***src/main.c*** fájlban található. Kódunk legfontosabb jellegzetessége a `clientdbUpdate_sb()` és a `clientdbEstablished_sb()` függvény meghívása a ***src/client_side.c*** fájl ügyféloldali eljárásaiban:

- A `clientdbUpdate_sb()` függvény meghívása a külső `clientWriteComplete()` függvényből, mely az adatok ügyfél felé történő elküldéséért felelős.

- A `clientdbEstablished_sb()` függvény meghívása az ügyfél kérését feldolgozó `clientReadRequest()` függvényből.

Az *1. kódrészlet* az ***src/client_side.c*** fájlból a `clientWriteComplete()` és a `clientReadRequest()` függvény megfelelő részeit tartalmazza.
A kód működése rendkívül egyszerű. A *2. ábra* az ügyfelektől érkező kérések feldolgozásának folyamatát szemlélteti a rendszer szemszögéből. Minden az ügyfelektől érkező kérés tartalmazza a hitelesítési adatokat, köztük a felhasználónevet is. A `clientdbUpdate_sb()` függvény megkeresi a kérésben szereplő felhasználónévhez tartozó `ClientInfo_sb` rekordot. Ha nincs ilyen rekord, akkor a hitelesítési fájlok tartalma alapján új `ClientInfo_sb` rekordot hoz létre. Ha a felhasználók túllépik korlátjukat, akkor a `comm_close()` függvény azonnal megszakítja a kapcsolatukat.
A `clientdbEstablished_sb()` függvény meghívása az ügyfélkérések számának szabályozását szolgálja, illetve az aktuális felhasználói információk elmentését a hitelesítési fájlokba, minden `SB_MAX_COUNT` kérés után. A hitelesítési fájlok neve – a unixos világban megszokott módon – `passwd` és `group`. A `passwd` fájl tartalmazza a felhasználói, a `group` pedig a csoportokkal kapcsolatos adatokat. Például:

```
`passwd':
#<név>:<teljes név>:<csoportazonosító>:
#<jelenlegi korlát>:<korlát utolsó frissítésének
➥időpontja>
```



2. ábra Ügyféltől beérkezett kérés feldolgozásának folyamata

```
tagir:Tagir Bakirov:1:6567561:12346237467
`group':
#<név>:<teljes név>:<csoportazonosító>:
#<csoport korlátja>:<csoportkorlát típusa>
users:BSPU users:1:10000000:D
```

Háromféle korlát létezik: `D` (***daily, napi***), `W` (***weekly, heti***) és `M` (***monthly, havi***). A `passwd` és a `group` fájl nevét és elérési útját a *Squid* beállító fájljában, a *squid.conf*-ban lehet megadni, ez a *squid.conf* sablonfájl és a *Squid* beállításait tartalmazó adatszerkezet módosításával vált lehetővé.
Néhány további apróbb, a *Squid* forráskódján végrehajtott módosítás:

- Átfogó függvények meghatározása az *src/protos.h* fájlban
- A `ClientInfo_sb` adatszerkezet típusmeghatározása az *src/typedefs.h* fájlban
- A `ClientInfo_sb` adatszerkezet azonosítójának megadása az *src/enums.h* fájl adatszerkezet-listájában
- A `ClientInfo_sb` adatszerkezet kezdeti értékadása a `memInit()` memóriafoglaló eljárásban, az *src/mem.c* fájlban

Minden módosítást hasonlóan végeztünk el a kódban is, megőrizve az eredeti ügyfél/IP-cím adatbázist. Reméljük, semmit nem rontottunk el.

Áttekintve a módosításainkat, bizonyára feltűnt, hogy minden kódunkat feltételes fordítást előíró `#ifdef SB_INCLUDE ... #endif` szakaszokba helyeztük. Az `SB_INCLUDE` változó akkor kerül megadásra, ha az `--enable-sbclientdb` átadott érték szerepel a *Squid* beállító parancsfájljának parancssorában. Ehhez az *autoconf* segítségével újra kellett fordítanunk a *configure.in* parancsfájlt – előtte elvégeztünk rajta néhány apróbb változtatást.

Összefoglalás

Írásunkban áttekintettük az internetelérés szabályozásának problémáját. Több megoldást is javasoltunk rá, majd részletesebben is ismertettük a *Squid* proxy-kiszolgálóra épülőt, amelyet egyetemünk helyi hálózatában is megvalósítottunk. Tisztában vagyunk vele, hogy megoldásunk nem hozza el a megváltást, valamint nyilván számos hátránya is van, ám egyszerű, rugalmas és teljesen nyílt.
Webes programozónk, *Elmir Mirdiev* mostanában fejez be egy kisebb, *PHP* alapú webhelyet, amelyen keresztül mód nyílik majd a rendszer felügyeletére és a felhasználói statisztikák lekérdezésére. Utóbbiakat a *Squid* naplóiból a *Sarg* rendszerrel állítjuk elő.
További tudnivalók a rendszer forráskódjában találhatók. Webhelyünkről a *Squid 2.5STABLE7* kiadásának teljes módosított változata és a megfelelő foltok külön is letölthetők. Bárki kérdéseit örömmel fogadjuk elektronikus levélben.

Linux Journal 2005. június, 134. szám

Tagir K. Bakirov (batk@mail.ru) rendszergazda a BSPU-nál és első éves doktoranduszhallgató az Ufai Állami Repüléstechnikai Egyetemen. Főként az adatbiztonság, a többügynökös rendszerek és egyéb informatikai témák érdeklik. Szabadidejében sportol, olvas, zenét hallgat és nyelveket tanul.

Vladimir G. Kozlov (admin@bspu.ru) a pedagógiai tudományok doktora, docens, a BSPU vezető rendszergazdája, ahol több informatikai tárgyat is oktat. Elsősorban a *NIX alapú hálózatok, az informatika és az elektronika érdekli. Hobbija a rádiózás (UA9WBZ) a sport és a családja.

KAPCSOLÓDÓ CÍMEK

- www.bspu.ru
- www.netams.com
- www.squid-cache.org
- www.squid-cache.org/Doc/FAQ/FAQ-2.html
- www.squid-cache.org/Doc/Prog-Guide/prog-guide.html
- sarg.sourceforge.net
- www.bspu.ru/squid/squid-2.5STABLE7.sb.tar.gz
- www.bspu.ru/squid/squid-2.5STABLE7.sb.patch