KGS SİSTEMİNE UZAKTAN ERİŞİM

Erhan ANUK [1] Ozan Eren BİLGEN [2]
Erdal KEMİKLİ [3]

[1]İTÜ Bilişim Enstitüsü, İleri Teknolojiler Anabilim Dalı,
Bilgisayar Bilimleri Bölümü, İTÜ Maslak Kampüsü, 80626, Ayazağa, İstanbul
[2]TÜBİTAK – UEKAE, P.K. 74, 41470 Gebze, Kocaeli
[3]Otokoç İstinye Temsilciliği, İstinye Balabandere Mevkii P.K. 15 Yeniköy, İstanbul

[1] e-posta: erhana@be.itu.edu.tr [2] e-posta: oebilgen@uekae.tubitak.gov.tr
[3] e-posta: erdalk@otokoc.com.tr

Anahtar Sözcükler: kalıcılık, dağıtılmış ortamlar, RPC, IPC, Linux

ABSTRACT

One of the rutine procedures in software engineering is to store variables on exit, in order to reload and reuse them within the next lauch of the application. This temporally objects bring additional workloads and also security problems by. In spite of transient objects, persistent objects are located, loaded and stored automatically by a subsystem, in order to reduce and standartize software development processes. A recent study on this issue is Extensible Persistent System (EPS). EPS provides persistency support through an operating system-based solution, augmented by language level mechanisms. EPS allows data and processes to be viewed as persistent objects, and be handled through a uniform interface. This paper presents the work carried out to provide distributed access to EPS, thus permitting remote users as well to utilize system facilities. This improvement on EPS is achieved by modifying the original communication system of EPS by adding a messaging center interface on top of the Naming and Security Server (NSS). Former SysV Interprocess Communication System (IPC) messages of EPS are also transformed into Remote Procedure Call (RPC) messages form while they pass trough the messaging layer. This replacement is implemented with minimum modification on basic EPS structure.

1. GİRİŞ

Bir uygulamanın kaldığı yerden devam edebilmesi için, işlediği verileri çıkışta seçilen bir medyaya yazması ve çalıştırıldığında tekrar okuması gereklidir. Bu rutin süreç, gerçeklemeye göre çeşitli yöntemlerle icra edilse de, işlemin yazılım süreçlerine getirdiği iş gücü ve güvenlik sıkıntıları bilinmektedir. Bu dağınık yapı, uygulama çeşitliliği arttıkça daha büyük sorunlara yol açmakta, yazılım geliştiricilere her uygulama için konsepte bağlı metodlar geliştirmeye zorlamaktadır.

Bu yöntemin alternatifi olarak, uygulama bitince bellekten temizlenen geçici nesneler yerine, "kalıcı nesneler"'in kullanılması da mümkündür. Konuyla ilgili bir çalışma olan Kalıcı Nesneleri Destekleyen Genişletilebilir Sistem (KGS) aktif/pasif nesne soyutlamasına dayalı modellemesi sayesinde yazılımcıların yeni uygulama ve sistemleri daha hızlı bir şekilde geliştirmesine yardımcı olmak için, kalıcı nesneler ve aktif nesne geliştirme araçlarından faydalanmıştır[1]. Bunun yanı sıra, KGS ek sistem yönetimi ve ayarlama çalışmaları gerektirmez [2]. Sistem soyutlamasına yeni bir yaklaşım getiren yenilikçi bir yazılım geliştirme ve sistem modelleme aracı olmasının yanında, KGS çok yaygın bir ortam olan UNIX'e benzediği için öğrenilmesi kolay bir sistemdir. KGS'de sistemin basit ve varolan sistemlere benzer olması, diğer taraftan da yeni bir programlama modeli ve araçlarla geliştirilmesi amaçlanmıştır. Ayrıca anlaşılır soyutlamalar sayesinde, çeşitli araçlar ve seçenekler ile kontrolün yazılımcıda olduğu bir ortam tanımlanmıştır.

KGS, bilgisayar sistemlerinde önemli bir sorun olan yazılım verimliliğini artırmak amacıyla genişletilebilir sistemler için kalıcı nesne kavramına dayanan bütünleşik bir model önermektedir. Bu yapıda uygulama geliştirici, veri ve uygulamayı kalıcı nesneler olarak değerlendirebilmekte ve tek bir arayüz ile yönetebilmekte; yapıdan nesneler arası haberleşme hizmeti, İstekçi Nesne Yöneticisi (İNY), Aktif Nesne Kütüphanesi (ANS), Adlandırma ve Güvenlik Sunucusu (AGS) ile Nesne Sunucusu (NS) hizmetlerini alabilmektedir.

KGS bir önderleyici ile Linux işletim sistemi üzerinde gerçeklenmiştir.

2. KGS ve PROGRAMLAMA

KGS'te programlama bazı farklılıklar dışında UNIX'te program geliştirmeye oldukça benzemektedir. KGS ile sağlanan yenilikler şunlardır:

- Kalıcı nesneler için sağlanan destek
- Sunucu programlarının (aktif nesnelerin) ihtiyaç duyulduğunda kendiliğinden yüklenmesi
- Kullanımı kolay olmasına rağmen güçlü ve esnek olan nesneler arasında haberleşme
- Sistem programcılığını kolaylaştıran ve hataları azalmasına yardımcı olan bir aktif nesne geliştirme desteği
- Nesnelere erişim yetkileri aracılığı ile denetimli erişim

Bir KGS-C programı KGS-C önişlemcisi ve C derleyicisi ile derlenir. Program İNY ve diğer kütüphaneler ile bağlanır. Tipik bir KGS-C programı kalıcı değişkenlerin tanımlanması ile başlar (Şekil 1). İlk birkaç satırda gerekli nesneleri yükleyerek görev ortamını hazırlayan EpsInit() fonksiyonu çağrılır. Arzu edilen işlemler tamamlandıktan sonra program, kalıcı nesneleri kaydeden ve alınmış kaynakları serbest bırakarak kalıcı görev ortamını kapatan EpsCommit() fonksiyonu çağrılarak sonlandırılır.

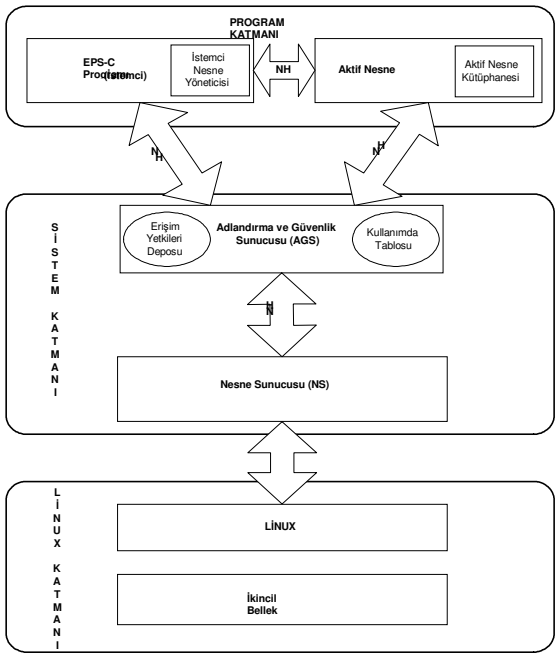
```
/* Kalıcı nesne tanımlaması */
$Adlar *pAdAgac;
CapabilityType caV;
/* Yerel Oturumu baslat */
EpsInit (&G1, "demo1", lCapF);
/* Kalıcı nesneyi yukle */
result = LoadPersistentObject (&G1,
                    (char**) &pAdAgac,
                    caV,
                    ACMRW);
/* … */
/* Yerel oturumu kapat ve kalıcı nesnelerdeki
degisiklikleri kaydet */
EpsClose (&testSession, COMMIT);
```

Şekil-1. Bir KGS programı

KGS-C, C programlama diline bazı eklemeler yapılarak oluşturulmuştur. Bu genişletilmiş sözdizimini standart C programlama dilinden ayırmak üzere KGS-C olarak adlandırılmıştır. Bu gerçekleştirme yöntemi sayesinde programcılar istediklerinde C dilini de kullanabilmektedir. C diline yapılan temel eklenti kalıcı nesneleri temsil eden "$" simgesinin sözdizimine eklenmesidir.

3. KGS MİMARİSİ ve SORUNUN KONUMU

KGS haberleşme altyapısı SysV IPC mesajlarına dayandırılarak gerçeklenmiştir. IPC mesajları sadece yerel bilgisayar üzerinde izlenebilme ve aktarılabilme özelliğini taşırlar. Bu nedenle, KGS sistemi farklı düğümlerden erişilebilir bir sunucu görevi görememekte, ve bu da dağıtılmış bir ortamda yer alan uzak kullanıcıların KGS olanaklarından yararlanmasına engel olmaktadır.



Şekil-2: KGS Yapısı

Bu çalışmanın amacı KGS'ne ağ üzerinden gelecek uzak istekleri de karşılayabilme niteliğini kazandırmaktır. Böylece, merkezi bir KGS sunucusu tasarımıyla, istemci bilgisayarlar sunucu bilgisayarın depolama ve işlem kapasitesinden faydalanabilecek ve tasarımda merkezi pratiklik sağlanabilecektir.

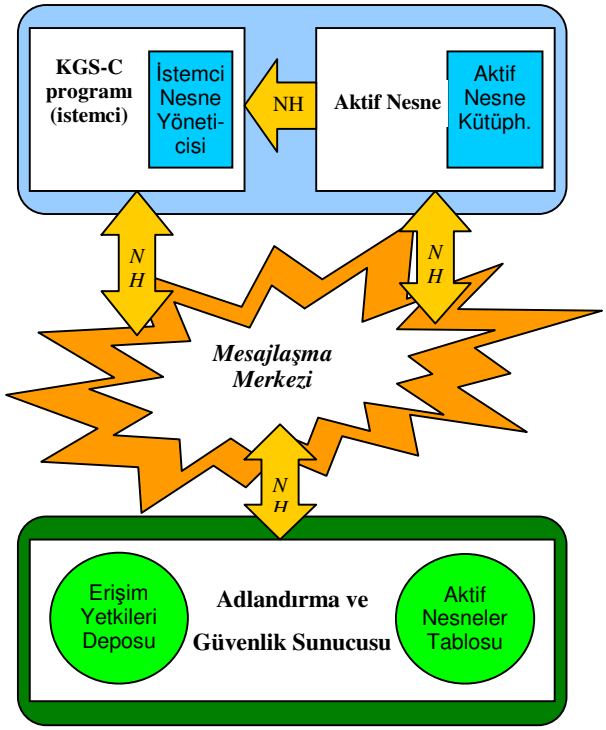
4. MESAJLAŞMA MERKEZİ

Sistemin uzak düğümlere de hizmet verebilmesi için, haberleşme sisteminin dışarıya açılması gereklidir. Ancak, varolan mimarinin kullandığı SysV'nin tasarım olarak buna izin vermemesi ve SysV'yi dışlayan bir değişikliğin de KGS'nin yapısını bozabileceği endişesiyle, mimariye RPC desteği sunacak yeni bir modülün, **Mesajlaşma Merkezi'**nin eklenmesine karar verilmiştir.

Yapılan incelemeler sonucunda, bu modülün Program Katmanı ile Sistem Katmanı arasına yeni bir ara katman olarak yerleştirilmesinin uygun olacağı anlaşılmıştır (bkz. Şekil 3.). Bu sayede, KGS-C programı (istemci) ve AGS arasında geçen haberleşmenin IPC ile değil dağıtık mesaj iletimini

uygulama geliştiriciye saydamlaştıran [3] RPC ile sürdürülmesi mümkün olacaktır.

Bu merkez AGS ile aynı düğümde çalışacak, RPC aracılığıyla istemciden aldığı mesajları IPC biçemine dönüştürecek ve AGS'den aldığı yanıtları da yine RPC biçeminde istemciye iletecektir.



Şekil-3: Önerilen Mesajlaşma Merkezi'nin KGS'deki konumu

5. GERÇEKLEMELER

KGS sisteminde, fonksiyonlar IPC isteklerini direkt olarak çağırmaktaydılar. Değişiklikten sonra bu yapı fonksiyonların RPC isteklerini çağırması şeklinde gerçeklenmiştir. Örnek üzerinde ifade etmek gerekirse:

```
Fonksiyon (parametre1, parametre2) {
        IPC_istegi (parametre1, parametre2);
}
```

şeklindeki bir yapı:

```
Fonksiyon (parametre1, parametre2) {
        RPC_istegi (parametre1, parametre2);
}
/* karşı düğümde */
RPC_istegi_versiyon(parametre1, parametre2) {
        IPC_istegi (parametre1, parametre2);
}
```

haline getirilmiştir.

IPC isteklerinin RPC isteklerine çevrilme aşamsında yapılan ilk gerçekleme IPC fonksiyonlarına geçirilen parametrelerin RPC'ye uyumlu bir veri yapısına dönüştürülmesidir. Bunu için aşağıda gösterilen veri yapısı oluşturulmuştur.

```
struct m_param {
        PIDType objectId;
        PIDType senderObjectId;
        int operation;
        CapabilityTypeRPC capability;
        char parameter[MAX_PAR_SIZE];
        ResultType result;
};
typedef struct m_param m_param;
```

m_param veri yapısı, fonksiyonun beklediği tüm parametreleri içermektedir. Örneğin daha sonra SendMsg fonksiyonunun değişmesinde şöyle kullanılır.

```
m_param *
SendMsg (CLIENT *clnt, m_param *p) {
        m_param  *result;
        result = sendmsg_1 (p, clnt);
        if (result == (m_param *) NULL) {
                client_perror (clnt,
                        "SendMsg failed!");
        }
        return result;
}
```

RPC'ye geçişte ek olarak KGS sistemi devreye sokulurken AGS'nin konumunu belirten bir parametre verilmelidir. Bu parametre düğüm adı veya IP adresi olabilir.

Bunun için, öncelikle EpsInit fonksiyonunu taşıyan epscob.c dosyasını değiştirdik. Eskiden EpsInit başlığı aşağıdaki gibiydi:

```
ResultType EpsInit (SessionType *session,
        char prgName[FILENAMESIZE],
        FILE *lCapFile);
```

Ancak artık sunucuya erişme yoluna da gerek duyulduğundan şu şekilde değiştirildi:

```
ResultType EpsInit (SessionType *session,
        char prgName[FILENAMESIZE],
        FILE *lCapFile,
        char *serverName);
```

Eklenen serverName karakter dizisi, uygulamalarda mecburi olan EpsInit fonksiyonuna eklenen bir parametredir ve bu uygulama tasarlarken ek bir yük getirmemektedir. EpsInit fonksiyonunun başlangıcına aşağıdaki satırlar eklenmiştir:

```
if (!clnt){
        if (!(clnt=rpc_setup(serverName))) {
        fprintf (stderr,
                "EpsInit(): RPC setup failed\n");
        exit (1);
        }
}
```

Burada clnt, RPC istemci tipi olan struct CLIENT veri yapısını gösteren bir işaretçidir ve sistemde global olarak tanımlanması gerekmektedir.
RPC'nin seçilme nedeni soket iletişiminde kolaylık sağlaması ve yaygın olarak kabul görmesidir. Aynı işlem read(2) ve write(2) soket ilkelleriyle de gerçeklenebilirdi; ancak tip boyutları, mesaj bekleme, protokol tanımlama ve benzeri soket programlama maliyetleri nedeniyle, RPC çoklu veri transferinde düşük performans kaybıyla büyük kolaylık sağlamaktadır[4]. Bu sebeplere bulduğu çözümlerden dolayı RPC halen pek çok dağıtık sistemin alyapısını oluşturmaktadır[3].

RPC aynı zamanda erişim, yer, çoğullama, uyuşma ve kalıcılık konularında sunduğu saydamlık özellikleriyle, sistemin dağıtık çalışmasına da olanak tanıdığı için [5], RPC'nin KGS'nin gelecekteki yönlenmesinde belirleyici faktörlerden biri olacağı düşünülmektedir.

6. İLERİ ÇALIŞMALAR

Günümüz itibariyle Linux RPC çokiplikli (multi-thread) çalışmayı desteklememektedir. Bu yapısı KGS'nin RPC mesajlarında bloke olmasına neden olmaktadır. RPC'nin çokiplikli hale uygulama seviyesinde mi geçirileceği yoksa RPC'nin bu özelliğe kavuşmasının mı bekleneceğine henüz karar verilmemiştir [6]. RPC'nin çokiplikli olması durumunda bu ek yazılım seviyesinde de yapılmışsa çift işgücü harcanacağı, yapılmamışsa sistemin bir anda karşılıklı dışlama gerektiren durumlarla karşılaşacağı düşünülmektedir. Belirsizlik sürdüğü için şu anki önerilen çözüm kuyruk sistemiyle belirli bir gecikmenin ardından tekrar sorgulama yapmaktır.

7. SONUÇ

Uygulama geliştirmede toplam maliyeti artıran sorunlardan biri nesnelerin canlandırılması süreçleridir. KGS sistemi bu rutin yazılım faaliyetine aktif/pasif nesne modellemesiyle çözüm getirmiştir. KGS sisteminde nesneleri barındıran Adlandırma ve Güvenlik Sunucusu, istemcilerle IPC üzerinden iletişim kurduğundan, uzak düğümlerden gelen isteklere cevap verememektedir. KGS sisteminin derinlemesine değiştirmeden, AGS'nin üzerine yerleştirilecek Mesajlaşma Merkezi, düğümler arası RPC trafiğini yerel IPC isteklerine çevirebilir. Bu suretle, değişik düğümlerdeki KGS istemcileri merkezi bir bilgisayarı nesne sunucusu olarak kullanabilir. Ağ tasarımlarında veritabanlarıyla aynı kategoride konumlandırılacak KGS/AGS'nin, gerek uygulama geliştiricilerin gerek kullanıcıların verimlilik sorunlarına yeni çözümler getirmesi ümit edilmektedir.

KAYNAKLAR

[1] Kemikli, E., "Design of an Extensible System Supporting Persistent Objects", Doktora Tezi, İstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, 2004.

[2] Kemikli, E., Erdoğan, N., "Kalıcı nesneleri destekleyen genişletilebilir bir sistem", İTÜ Dergisi/D Mühendislik, İstanbul, Aralık 2004.

[3] Tanenbaum. A.S., Steen, M. v., "Distributed Systems – Principles and Paradigms", Prentice Hall, pp. 68-84, 2002.

[4] Marshall, D., "Remote Procedure Calls (RPC)", http://www.cs.cf.ac.uk/Dave/C/node33.html,5/01/1999.

[5] Bilgen, O.E., "Transparency in RPC", http://www.oebilgen.com/himm/okul/transparency_in_rpc.doc, 21/10/2003.

[6] Solomon, M., "Implementing a Multi-Threaded Server", http://www.cs.wisc.edu/shore/1.0/ssmvas/node5.html, 02/08/1996.