YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DAĞITIK SİSTEMLER İÇİN ÖNERİLEN DOSYA BULUCU BİR SİSTEM TASARIMI VE GERÇEKLENMESİ: DOSYA İSİM SERVİSİ (FILE NAME SERVICE)

Bilgisayar Yük.Müh. Ahmet Tevfik İNAN

F.B.E. Bilgisayar Bilimleri Mühendisliği Anabilim Dalında
Hazırlanan

DOKTORA TEZİ

Tez Savunma Tarihi	: 24 Nisan 1998
Tez Danışmanı	: Prof. Mehmet Yahya KARSLIGİL (Y.T.Ü.)
Jüri Üyeleri	: Prof. Dr. Oğuz TOSUN (B.Ü.)
	Doç. Dr. Bülent Örencik (İ.T.Ü.)

İSTANBUL, 1998

İ Ç İ N D E K İ L E R

Sayfa

ŞEKİL LİSTESİ

TABLO LİSTESİ

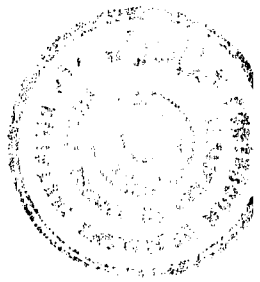
T E Ş E K K Ü R

Doktora çalışmam boyunca benden desteğini esirgemeyen ve sabrı tükenmeyen hocam Sayın Prof. Mehmet Yahya KARSLIGİL'e; tezimi inceleyerek değerli önerileriyle beni yönlendiren jüri üyelerim Sayın Prof. Dr. Oğuz TOSUN ve Sayın Doç. Dr. Bülent ÖRENCİK'e; çalışmam boyunca kaynaklarından yararlandığım Yıldız Teknik Üniversitesi Elektrik Elektronik Fakültesi Bilgisayar Bilimleri ve Mühendisliği Bölümüne ve her türlü yardımları için, bölüm araştırma görevlilerinden sevgili arkadaşlarım Ali Gökhan YAVUZ'a, Mine Elif KARSLIGİL'e, Ahmet HAKTANIR'a; tezimi okumak için değerli vaktini ayıran hocam, Sayın Doç. Kirkor HARUTUNYAN'a; hayatım boyunca desteklerini hep arkamda hissettiğim babam Prof. Dr. Ali Naim İNAN'a, annem Aynur İNAN'a ve parçası olmaktan gurur duyduğum ailemin tüm bireylerine teşekkürü bir borç bilirim.

Ö Z E T

Yarı iletken ve bilgisayar teknolojilerindeki gelişmelere paralel olarak, her geçen gün daha yüksek kapasiteli, daha hızlı işlem yapabilen, çok işlemcili bilgisayarlar tasarlanmakta ve üretilmektedir. Gelişen iletişim imkanları ve protokolleri, bilgisayarlar arası haberleşmenin kolaylaşmasını ve ucuzlamasını sağlamıştır. İletişim alanındaki gelişmenin doğal bir sonucu olarak, işletim sistemleri, mimari yapıları, komut kümeleri çok farklı, bulundukları yerler çok ayrık olan bilgisayar sistemleri, uygun haberleşme protokolleri yardımıyla, ortak amaca varmak için bir bilgisayar ağı dahilinde kullanılmaya başlanmıştır. Kapasite ve hızları ile belirgin bir hesaplama gücü ortaya koyan, yüksek hızlı haberleşme hatları ile birbirlerine bağlı, aynı protokolü kullanan bilgisayar sistemlerinin oluşturduğu küme, fiziksel yerleşimlerinin gösterdiği ayrılıktan ötürü "dağıtık sistemler" olarak adlandırılmaktadır.

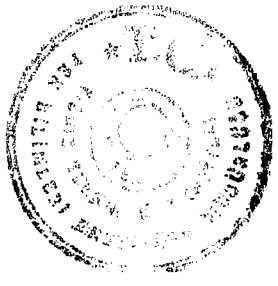
Bu doktora tez çalışması, "dağıtık sistem"ler üzerinde kullanılmak üzere, 'dosya bulucu' bir sistemin tarifi ve bu tariften hareketle geliştirilen uygulamayı içermektedir. Deneysel olarak yerel alan ağları ile sınırlı düşünülen uygulama, istemci/sunucu modeline uygun olarak tasarlanmıştır. 'Dosya İsim Servis'i (File Name Service) olarak adlandırdığımız uygulama, bağlantılı, eş zamanlı, çok işlem parçacıklı, kısmen şeffaf bir yapı üzerine kuruludur. 'Dosya İsim Servisi' üç temel bileşenden oluşmaktadır. Bu bileşenler; 'Dosya İsim Sunucusu' (File Name Server -FNS), 'Dosya İsim Sunucu Ajanı' (File Name Server Agent - FNSA) ve 'Dosya İsim Servis İstemcisi' (File Name Service Client - FNSC) olarak adlandırılmıştır. 'Dosya İsim Servisi', de facto standart olan TCP/IP ağ protokolünü kullanırken, endüstri standardı olan dosya erişim metodu NFS (Network File System)'den destek almaktadır. Geliştirmeye açık bir yapıda tasarlanan ve taşınabilir (portable) olması için, C programlama dilinde kodlanan 'dosya bulucu' sisteminin amacı, kullanıcının içinde bulunduğu etki alanı içindeki 'Dosya İsim Servisi'ne dahil herhangi bir dosyayı, bulmak ve istemciye, dosya erişiminin ayrıntılarından bağımsız olarak sunmaktır.



A B S T R A C T

Recent developments in semiconductor and computer technologies made it possible to design and produce high capacity, high speed computers and multiprocessor systems. Better communications facilities and improved communication protocols made the computer communication easier and cheaper. Consequently, computer systems located at wide apart locations, with different operating systems, architectures and instruction sets, became a part of large computer networks in order to reach a common goal. Collection of the physical dispersed computer systems, communicating with each other via common communication protocols over high bandwidth communication networks, forming a reasonable computing power are called "distributed systems".

In this dissertation, a 'file locating system' and its application on distributed systems are introduced. The application is named 'File Name Service' and based on client/server model and realized on local area network (LAN) for experimental purposes. 'File Name Service' uses connection oriented, concurrent, multithreaded, semi-transparent structure. 'File Name Service' consists of three components. These are called 'File Name Server - FNS', 'File Name Server Agent - FNSA', and 'File Name Service Client - FNSC'. All of the components of the 'File Name Service' run on top of the de facto network protocol TCP/IP and take advantage of the industry standard file access method, NFS (Network Files System). The application is open for any improvements and uses C programming language to be portable. The aim of the 'File Name Service' is to locate any file, served by a 'File Name Service' within the domain, make available for access, without letting the client to deal with the underlying details of the actual file access.



1. GİRİŞ

1.1 Bilgisayar Sistemlerinin Kısa Tarihçesi

14 şubat 1946 tarihinde Pensilvanya Üniversitesinden Profesör J. Eckert ve J. Mauchly'nin ilk gerçek bilgisayar olarak tanımlanan ENIAC'ın üretildiğini duyurmalarını takiben geçen elli yılı aşkın sürede, gerek donanım gerekse yazılım alanında oldukça mesafe kat edildi. 1971'li yıllara gelinceye kadar anabilgisayar (mainframe) sistemleri gelişmelerini sürdürmekle beraber, temelde geniş hacimler kaplayan, çalışması nispeten karmaşık, özellikle de pahalı cihazlar olmaya devam ettiler. Ancak, 1971 tarihinde Intel firmasının ilk 4 bit işlem yapabilme özelliğindeki mikroişlemcisi 4004'ü piyasaya sürmesi ile bilgisayarlar daha küçük hacimlere sığar hale gelebildiler. Fakat bilgisayarlardaki bu küçülmenin fiyatlarına yansıması 1980'li yıllarının ortalarını buldu. Bu tarihlerde artık teknoloji 16, 32 hatta 64 bit işlem kapasitesine sahip işlemcilerin rahatlıkla alınabileceği kadar ucuzlamış, işlemcilerin güçleri yüzlerce MIPS (saniyede milyon işlem sayısı) ile ifade edilmeye başlanmıştı. Grosch'un "bir işlemcinin işlem gücü fiyatının karesi ile doğru orantılıdır" sözü yavaş yavaş geçerliliğini kaybetmeye başladı. Böylelikle anabilgisayarların yanı sıra, daha ucuz, nispeten daha az işlem gücüne sahip mikroişlemcili kişisel bilgisayarların kullanımı da yaygınlaşmaya başladı. Diğer yandan Xerox firması tarafından başlatılan yerel bilgisayar ağları çalışmalarındaki gelişmeler ile kişisel bilgisayarlar hızlı bir altyapı ile birbirlerine bağlanabilir ve bilgi alış-verişinde bulunabilir hal aldılar. Hızla ve paralel olarak gelişen bu iki teknolojinin sonucunda, sistemler* genel çalışma prensiplerine bağlı olarak merkezi (centralized) ve dağıtık (distributed) olarak iki temel sınıfa ayrıldı.

1.2 Bilgisayar Sistemlerinin Geleceği

1946 yılında ENIAC'ın üretilmesini takip eden elli yılı aşkın sürede hızla küçülen ve maliyeti düşen bilgisayarlar, çok çeşitli ve işlevli yazılımların desteğinde, gerek iş, gerekse eğlence aracı olarak günlük yaşamımızın vazgeçilmez bir parçası oldular. Gelişen haberleşme sistemlerinin bilgisayar teknolojisi ile birlikte kullanılmaya başlaması ile

* Bu sınıflamada kullanılan "sistem" kelimesi, temelde bilgisayarın sahip olduğu donanım (hardware) ve yazılım (software) öğeleri ile bir bütün olarak çalışmasını sağlayan temel kavramı ifade etmektedir.

birlikte, dünyanın farklı noktalarındaki bilgisayarlar, önceleri telefon hatlarını kullanarak yaptıkları haberleşmeyi, günümüzde uydu bağlantılı, yüksek kapasiteli, sayısal sistemler ile desteklenen hatlar üzerinden gerçekleştirmeye başladılar. Yapılan her araştırmada, bankacılıktan sağlığa, haberleşmeden sanata kadar hemen hemen her dalda bilgisayar "olmazsa olmaz" bir konuma gelirken, daha hızlı, daha çok işlem yapabilen, paralel çalışabilen bilgisayarlar geliştirilmeye, üretilmeye başlandı. Aileden biri haline gelen bilgisayarların geleceğini ise şimdiden tahmin etmek oldukça zor. Geçen elli yılda yaşananlar ve eldeki veriler ışığında bilgisayarların önümüzdeki elli yılı hakkında bir ipucu elde etmek ne derece mümkündür? Yoksa ortaya atılan fikirler bir temenniden mi ibarettir? ACM*'in ENIAC'ın ellinci yılını kutladığı ve önümüzdeki elli yıl hakkında bilişimcilerin görüşlerine yer verdiği sayısından (ACM, 1997) alıntılar yaparak önümüzde 'nasıl bir elli yıl?' olacağını görmeye çalışalım.

- " Moore kanunu uyarınca, yarı iletken, magnetik hafıza, fiber optik ve benzeri donanım teknolojilerindeki yıllık gelişme faktörü 1.6 seviyesinde olabilirse, 10 milyar kat güçlü, sıfır maliyetli, haberleşebilen (konuşabilen) bilgisayarlar telefonlardan ışık düğmelerine, motorlardan binalara varana kadar her şeyin içinde yer alabilecektir." (Bell, 1997)
- "Gelişmelerin, tariflendiği biçimde sürmesi durumunda 2047 yılında vücut üzerinde taşınabilir, taşıyanın duyduğu, okuduğu ve gördüğü her türlü veriyi algılayıp, saklayabilen ve gerektiğinde kullanabilen, 10 terabyte hafıza ve saniyede 10^{15} (1.000 milyon-milyon - petaops) işlem yapabilme gücüne sahip 'kişisel yardımcılar' hayal edilebilir" (Cochrane**).
- "TrackPoint'in 6 haftada geliştirilebileceğini düşünüyordum, 10 yıl sürdü, sunularda, projeksiyon cihazı üzerinde şeffaf bir kağıt gibi kullanılabilecek bir bilgisayar 6 ayda geliştirilebilir diye düşünüyordum, IBM ThinkPad 755CV'nin geliştirilmesi 3 yıl sürdü, uyarlanabilir bir yardım sisteminin (adaptive help system) yapılması 6 ay sürer diyordum, COACH'in bir ürün olarak ortaya çıkması 14 yıl sürdü" (Selker, 1997)

---

* Association of Computing Machinery

** Cochrane'nin bilgi ve geleceğin teknolojileri konusunda çeşitli makaleleri vardır. (Cochrane, 1992) (Cochrane vd., 1993) (Cochrane, 1994)

Bilgisayar teknolojisindeki gelişmeler, hesaplama güçlerinin yanı sıra iletişim hızlarında da büyük ilerlemelere yol açmıştır. Hesaplama gücünü artırmak için bir yandan, çok işlemcili, sıkı bağlı, yüksek maliyetli sistemler geliştirilirken, diğer yandan yerel alan ağlarında (LAN) 10-100 Mbps, hatta 1Gbps hızlarına, ATM (Asynchronous Transfer Mode) kullanımı ile geniş alan ağlarında (WAN) 155-622 Mbps hızlarına ulaşılması*, gevşek bağlı bilgisayar sistemlerinin daha az maliyet ve belirgin bir hesaplama gücü ile yarışta yerini almasını sağlamıştır. Kısaca, "hızlı" ve "kapsamlı" bir elli yıl bilişimcileri beklemektedir. Ancak, Ted Selker'in (1997) aktardığı deneyimlerinde vurguladığı 'teorik olarak çözülen pek çok problemin, uygulamasının kolay gerçeklenemediği' uyarısı akılda tutmalı, buna göre planlı, sağlam adımlar ile önce küçük, sonra büyük engeller aşılmalıdır.

1.3 Yazılım

Bilgisayarların gelişmesinden söz ederken ister istemez hep hız, kapasite gibi kavramlar ön plana çıkarılmaktadır. Ancak bir gerçek göz ardı edilmemelidir. Yazılım ile donanım ayrılamaz bir bütündür. Hangi unsurun esas olduğunu belirlemek mümkün değildir. Bu unsurlardan herhangi birinin tek başına olması yeterli olmayacağına göre sonuca varabilmek için uygun donanım, uygun yazılımlar ile birlikte çalıştırılmalıdır. Aslında bilgisayarların tarihinden söz ederken, 1954 yılında John Backus'un FORTRAN (FORmula-TRANslator) dilinden, 1964 yılında Tom Kurtz ve John Kemeny'nin BASIC (Beginners All-purpose Symbolic Instruction Language) dilinden, 1969 yılında Niklaus Wirth'in ilk PASCAL derleyicisinden, 1970'li yıllarda ortaya çıkan CP/M, DOS, Xenix, Mac işletim sistemlerinden, 1980'li yıllardan günümüze kadar gelişen Windows, programlama dilleri ve bunların derleyicilerinden, uygulama geliştirme yazılımlarından, grafik tasarım programlarından, çok çeşitli sistem ve uygulama yazılımlarından da söz etmek gerekir. Yazılım, bilgisayarın geçmiş elli yılı içersinde gizli kalmış gibi görünmesine rağmen nasıl etkin bir rol üstlendi ise gelecek elli yıl içinde de bu rolünü koruyacaktır.

---

* Geniş alan ağlarında 2,5 Gbps hızına ulaşmak için çalışmalar yapılmaktadır. (Reizenman, 1994) Hali hazırda sıkı eşlenmiş "Hypercube" ve "Transputer Network" gibi güvenilir bir yol yapısına sahip sistemlerde iletişim hızı, 200 MBps değerine ulaşmıştır. (IEEE Sepctrum, 1994) VME tabanlı sistemler için geliştirilen "Autobahn" projesi 400 MBps hızı hedeflemektedir (Futacı, 1995).

1.4 Tez Çalışması Hakkında

Yazılım, donanımın kullanılabilmesini sağlayan, kullanıcı ile donanım arasındaki ilişkiyi kuran, mevcut donanımdan farklı amaçlar için, etkin bir şekilde yararlanabilmeyi sağlayan bir araçtır. Doktora tez çalışmasına konu olan "Dosya İsim Servisi - File Name Service" bu bağlamda mevcut bilgisayar altyapısından etkin bir şekilde yararlanmayı amaçlayan, istemci/sunucu modelindeki dağıtık sistemler için geliştirilmiş, kullanıcıların aradıkları dosyaya (Dosya İsim Servisine dahil yapıda bulunuyor ise) nerede olduğunu bilmeksizin (location /migration transparent) erişmesini sağlayan, bir yazılımdır.

Sekiz temel bölümden oluşan tez çalışmasının ikinci bölümü, kaynak taraması ve çalışma sırasında karşılaşılan temel tanımları içermektedir. Üçüncü bölüm, ağ tabanlı servis tiplerine, dördüncü bölüm ise taşıma katmanı (transport layer) seviyesinde kullanılan arayüze yer vermektedir. Beşinci Bölüm istemci/sunucu modeli, çok iş parçacıklılığı (çok liflilik - multithreading), eş zamanlılık (concurrency), anuyumluluk (synchronization) ve bunların sağlanması için kullanılan bazı metotlara yer verir. Altıncı bölüm "Dosya İsim Servis"inin genel yapısını ve bu yapıyı oluşturan modülleri tanıtan bölümdür. Yapılan teorik ve pratik çalışmaların sonucunda geliştirilen uygulamalarla elede edilen değerlere, başarım hesaplarına ve uygulanan senaryoların değerlendirmelerine yedinci bölümde yer verilmiştir. Sekizinci bölüm, sonuçlar ve "Dosya İsim Servisi" nin geliştirilmesine yönelik çeşitli önerileri içermektedir.

2. TANIMLAR

2.1 Merkezi (Centralized) Sistemler

Tek ve güçlü bir işlemci, hafıza ve bunlara bağlı diğer yan birimler ile kullanıcıların sisteme erişmesini sağlayacak terminallerden oluşan birimler, merkezi (centralized) sistemler olarak adlandırılmaktadır. Bu sistemlere bu ismin verilmesindeki en büyük etken, sistemi oluşturan en önemli parçanın tek ve merkezde bulunması, üzerinde çalışmakta olan işletim sisteminin tüm kontrolü (monolithic) elinde bulundurmasıdır.

2.2 Dağıtık (Distributed) Sistemler

Dağıtık sistem, en genel anlamıyla "Birbirleri ile bağlantılı ve uyum içinde çalışan işlemcilerin meydana getirdiği sisteme" (Tanenbaum, 1992) verilen isimdir.

2.2.1 Mimari modeller

Literatür incelendiğinde, dağıtık sistemlerin tarifinde bir fikir birliğine varılamamış olduğu gözlenmektedir. (Sloman ve Kramer, 1987) Görülen tek ortak nokta hepsinin birden fazla işlem biriminin varlığından söz etmesidir. (Bal vd., 1989) Bu ayrılık, farklı mimari modellerin varlığından kaynaklanmaktadır. Flynn (Flynn, 1972), bilgisayar mimarilerini komut ve veri akışlarına (instruction and data stream) bakarak genel bir sınıflamada bulunmuştur. Buna göre hesaplama modelleri (computational model); (Akl, 1989,3-20)

1. **SISD** (Single Instruction Single Data - Tek Komut Tek Veri):
   Her komutun tek bir veri üzerinde işletildiği modeldir.
2. **MISD** (Multiple Instruction Single Data - Çok Komut Tek Veri)
   Çok işlemcili bir modeldir. Her veri üzerinde işlemciler farklı birimlerden aldıkları komutları işletirler.

3. **SIMD** (Single Instruction Multiple Data - Tek Komut Çok Veri)

Çok işlemcili bir modeldir. Her seferinde bütün işlemciler farklı veri üzerinde aynı komutu işlerler. Bu modelde işlemciler kendi aralarında, ya paylaşılan hafıza üzerinden (shared memory - SM) ya da bir ara bağlantı (interconnection network) üzerinden konuşarak veri veya ara sonuç alış-verişinde bulunurlar. Bu model, paylaşılan hafıza üzerindeki okuma/yazma işlemlerine göre dört alt sınıfta da (EREW - Exclusive Read Exclusive Write, CREW - Concurrent Read Exclusive Write, ERCW - Exclusive Read Concurrent Write, CRCW - Concurrent Read Concurrent Write) incelenebilir. Ayrıca ara bağlantının organizasyonuna bağlı bir diğer sınıflama (Linear array, 2D array, Tree, Perfect Shuffle, Cube) da mümkündür.

4. **MIMD** (Multiple Instruction Multiple Data - Çok Komut Çok Veri)

Çok işlemcili bu modelde her işlemci farklı veri üzerinde farklı komutlar işletmektedir. SIMD modelinde olduğu gibi paylaşılan hafıza veya bir ara bağlantı üzerinden işlemciler haberleşmektedir.

Bu hesaplama modelleri uyarınca ;

- **Vektör İşlemciler**

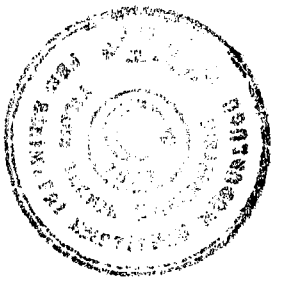
  Farklı veri üzerinde aynı aritmetik işlemi eş anlı (simultaneous) olarak uygulayan işlemcilerin (Russell, 1980, 63-72) oluşturduğu mimari modele verilen isimdir. Bu tür işlemciler SIMD hesaplama modeline bir örnektir.

- **Çoklu İşlemciler (Multiprocessors)**

  Birden fazla bağımsız (autonomous) işlemcinin aynı hafızayı paylaşması (Jones ve Schwarz, 1980) ile oluşan mimari modele verilen isimdir. Bu tür işlemciler MIMD hesaplama modeline bir örnektir.

- **Çoklu Bilgisayarlar (Multicomputers)**

  Çoklu işlemciler ile benzerlik göstermekle birlikte, işlemcilerin ortak bir hafızayı paylaşmadan kendi aralarındaki haberleşme ağı üzerinden mesaj alıp vermek suretiyle haberleştiği mimari modele verilen isimdir. (Athas ve Seitz, 1988, 9-24) Bu tür işlemciler MIMD hesaplama modeline bir örnektir.



- **Dataflow ve Reduction Makineleri (Machines)**

  Farklı data üzerinde farklı işlemler yapmayı sağlayan (Treleaven vd., 1982) mimari modele verilen isimdir. Bu MIMD hesaplama modeline bir örnektir.

2.2.2 Ara bağlantı (Interconnection network)

Dağıtık sistemler, aralarındaki bağlantıyı sağlayan haberleşme ağının yapısına göre de sınıflanabilir;

- **Sıkı Bağlı (Closely Coupled - Tightly Coupled)**

  Dağıtık sistemlerde işlemciler arasındaki haberleşme ağının hızlı ve güvenilir olduğu, işlemcilerin fiziksel olarak birbirlerine yakın bulunduğu yapıları ifade etmek için kullanılan terimdir.

- **Gevşek Bağlı (Loosley Coupled)**

  Dağıtık sistemlerde işlemciler arasındaki haberleşme ağının yavaş ve güvenilemez olduğu, işlemcilerin fiziksel olarak birbirlerinden uzakta bulunduğu yapıları ifade etmek için kullanılan terimdir.

2.3 Dağıtık Sistemin Tarifi

Dağıtık sistemler, bilişimciler tarafından gerek mimari modellere gerekse kullanılan haberleşme tekniğine (hafıza paylaşımı veya haberleşme ağı) bağlı olarak çeşitli şekillerde tanımlansa da, bu tez çalışmasında geçerli olan tanım, dağıtık sistemler ve bunların programlanması üzerine değerli yayınları bulunan Henri E. Bal ve Andrew S. Tanenbaum tarafından yapılan tanımdır. "Dağıtık sistem, ortak bir hafıza paylaşmayan, bağımsız işlemcilerin, bir haberleşme ağı üzerinden mesaj alış-verişi ile işbirliğinde bulunduğu sisteme verilen isimdir." (Bal vd., 1989; Futacı, 1995, 2) Yani dağıtık sistemi oluşturan işlemciler kendi hafızalarında bulunan komutları, yine kendi hafızaları üzerinde bulunan veri üzerinde işletmekte ve bir diğer işlemci ile haberleşme ihtiyacı duyduklarında aralarında bulunan haberleşme ağını kullanarak mesaj alıp vermektedirler. Bu tariften hareketle, dağıtık sistemin, bir ağ üzerinde (ağ, etki alanının büyüklüğüne göre LAN veya WAN olarak nitelendirilebilir) bulunan kişisel bilgisayarlar ya da iş istasyonlarının oluşturduğu gevşek bağlı (loosely coupled) bir yapı olduğu sonucu ortaya çıkmaktadır. Her ne kadar gevşek bağlı yapıların tarifinde bulunan 'yavaş ve güvenilmez' ibaresi günümüz şartlarında yerel alan ağları (LAN) için geçerli olmasa da, mesafelerin arttığı, kaynaktan

hedefe ulaşılana kadar pek çok ara noktanın geçilmesinin gerekli olduğu geniş alan ağları (WAN) için nispeten doğrudur. Bu nedenle, dağıtık sistemlerin temel dayanağını güçlü ve problemsiz bir iletişim altyapısı oluşturmaktadır.

İletişim için kullanılan protokollerin dayandığı iki temel prensip vardır. Bunlar bağlantılı (connection-oriented) ve bağlantısız (connectionless) servislerdir (Tanenbaum, 1989; Black, 1993, 256-259). Bağlantılı servislerin kullanılabilmesi için veri iletişiminden önce alıcı (receiver) ve verici (sender) arasında bir bağlantının kurulması, kullanılacak üst seviye protokolü üzerinde iki tarafın anlaşmaya varması, veri aktarımının yapılması ve veri aktarımının bitmesini takiben bağlantının sonlandırılması gereklidir. Bu servis, temel yapısı yönünden telefon sistemine benzetilebilir. Diğer yandan bağlantısız serviste, önceden taraflar arasında bağlantı kurulması zorunluluğu yoktur. Hazır olan bilgisayar diğerine mesajını yollar. Yollanan mesaj aynı posta sisteminde olduğu gibi alıcıya iletilecektir. İster bağlantılı ister bağlantısız servis olsun, dağıtık sistemlerin gereği olan problemsiz bir iletişim altyapısı için kullanılabilecek temel modeller olarak karşımıza Katmanlı Protokol, İstemci-Sunucu, Uzak Yordam Çağırma (RPC) modelleri çıkmaktadır.

2.3.1 Katmanlı protokol modeli

Dağıtık sistemler tanımı gereği, bağlı bulundukları ağ ortamı üzerinden mesaj alış-verişi ile haberleşirler. Ancak dağıtık sistem içindeki bilgisayar sistemleri arasında, gerek üreticiden kaynaklanan (farklı işlemciler, farklı komut kümeleri vb..), gerekse bilgisayarları kontrol eden işletim sistemlerinden kaynaklanan farklılıklar vardır. Bu durumda, sistemler arasındaki bu farklılıkları ortadan kaldıracak, önceden belirlenmiş kurallara (protokollere) ihtiyaç vardır. 1983 yılında International Standards Organization-ISO tarafından hazırlanan yedi katmalı OSI modeli (Day ve Zimmerman, 1983) her katmanda kullanılan protokolleri tek tek tariflemektedir. Katmanlı modelde her katman yollanacak veriye önbilgi (header) ve/veya art bilgi (trailer) ekleyerek bir sonraki katmana iletmektedir. Bu katmanlar ve görevlerini kısaca özetlemek gerekirse;

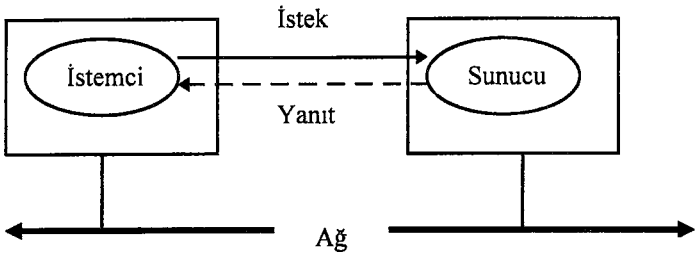
1. **Fiziksel Katman (Physical Layer):** Bu katmanın görevi, 0 ve 1'lerden oluşan verinin belirlenen elektriksel özelliklere ve hıza bağlı olarak mevcut ağ bağlantısını sağlayan ortam (bakır, fiber kablo vb..) üzerinden iletimini sağlamaktır.

2. **Veri Bağlantı Katmanı (Data Link Layer):** Temel görevi veri iletimi sırasında meydana gelen hataları bulmak ve düzeltmektir. Bu görevi yerine getirebilmek için, gönderilen veri belli parçalara (çerçeve-frame) ayrılır ve her parçaya, kendisi için hesaplanan sağlama toplamı (checksum) bilgisi eklenir. Alıcı tarafındaki veri bağlantı katmanı, gelen veri için hesapladığı sağlama toplamı bilgisi ile gelen arasında bir uyuşmazlık olursa o parçayı tekrar ister.
3. **Ağ Katmanı (Network Layer):** Bilgisayar ağları üzerinde bir bilgisayardan çıkan bir bilginin diğer bir bilgisayara ulaştırılması, ulaştırılırken izlenecek yolun belirlenmesi ağ katmanının sorumluluğundadır. Ağ katmanı bilgisayarlar arasındaki veri akışını sağlayabilmek için rotalama (routing) işlemini yerine getirmektedir.
4. **Taşıma Katmanı (Transport Layer):** İletilecek verinin kayıp olmadan bir uçtan diğerine iletilmesini sağlamak üzere taşıma katmanı, bir üst katman olan oturum katmanından (session layer) aldığı veriyi, paket (packet) olarak adlandırılan küçük parçalara ayırıp her birine sıra numarası verir. Alıcı tarafında ise, gelen paketlerin izledikleri farklı rotalardan dolayı yollandıkları sırada gelmemesi durumu göz önünde bulundurularak, gelen paketler sıralanarak bir üst katmana iletilir.
5. **Oturum Katmanı (Session Layer):** Oturum katmanı alıcı ve verici arasındaki iletişimin kontrolünü (dialogue control) ve anuyumluluğu (synchronization) sağlar. Bu katman taşıma katmanının biraz geliştirilmiş bir şekli olarak görülebilir.
6. **Sunu Katmanı (Presentation Layer):** Mimari ve komut kümelerinin farklılığından kaynaklanan gösterim farklılıklarını ortadan kaldırmak için sunu katmanından yararlanılmaktadır.
7. **Uygulama Katmanı (Application Layer):** Tam anlamıyla kullanıcıya hitap eden elektronik posta (e-mail), dosya transferi vb. uygulamaların yer aldığı katmanı ifade etmektedir.

2.3.2 İstemci-Sunucu modeli (Client-Server model)

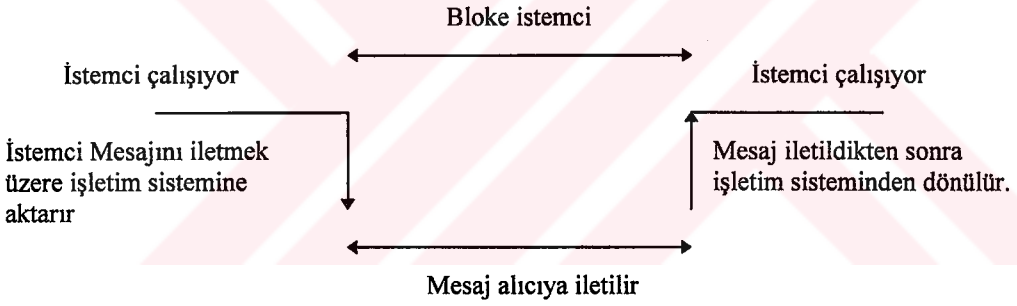
Bağlantılı protokol olan OSI ile ortaya atılan 7 katmanlı modelde her katmanın iletilecek veriye kendi işlevini yerine getirmek üzere önbilgi ve/veya art bilgi eklemesi protokole önemli bir yük getirmektedir (Nutt, 1992). Diğer taraftan istemci-sunucu modelinde, bağlantılı olmayan istek/yanıt (request/reply) protokolü (Sloman ve Kramer, 1987, 88-91; Tanenbaum, 1992, 402-415) kullanılarak bu yükten kurtulmak mümkündür. İstemci,

sunucudan bir istekte bulunur, sunucu isteğin gereğini yerine getirerek elde ettiği bilgiler ile istemciyi yanıtlar (Şekil 2.1). Bu modelin üstünlüğü basit oluşudur.

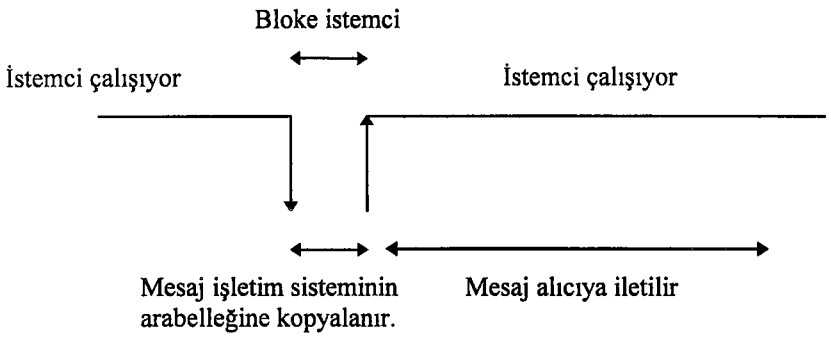


Şekil 2.1 İstemci-Sunucu Modeli

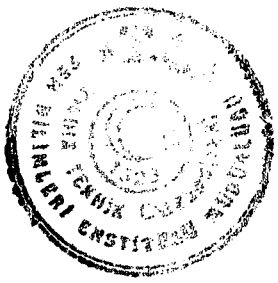
İstemci-Sunucu modelinde işlemin gerçekleşme şekline bağlı olarak bloke (blocking) ve bloke olmayan (non blocking) işlemlerden söz etmek mümkündür. İsminden de anlaşılacağı üzere bloke işlemlerde, istemde bulunan işlem (process) yollamak istediği mesaj yerine varıncaya kadar bloke olacaktır (Şekil 2.2). Diğer yandan bloke olmayan işlemlerde, istemde bulunan işlem yollamak istediği mesajı kendisine hizmet veren işletim sistemine ait bir ara bellek (buffer) alanına kopyaladıktan sonra yapmakta olduğu işe devam edebilecektir (Şekil 2.3).



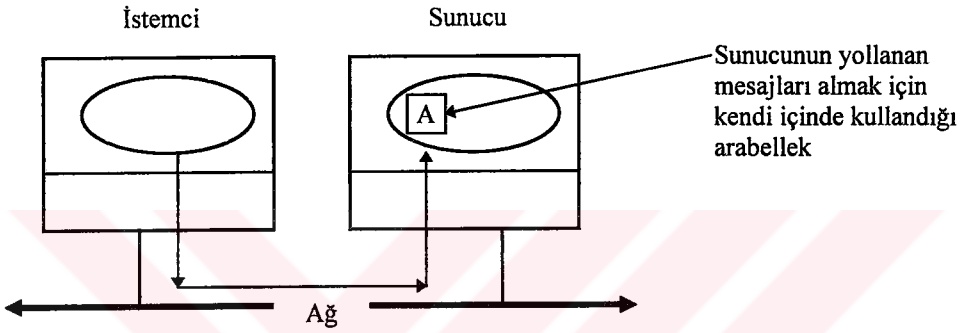
Şekil 2.2 Bloke işlem



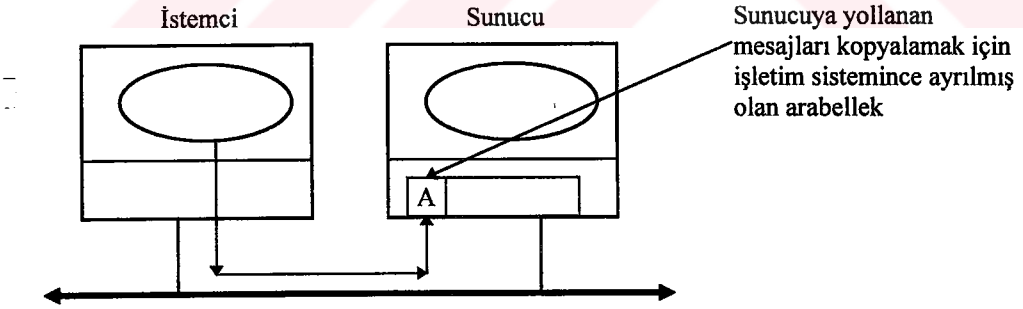
Şekil 2.3 Bloke olmayan işlem



Ayrıca istemci-sunucu sistemlerde ara bellek kullanan (buffered) ve ara bellek kullanmayan (unbuffered) işlemlerden (Ball, 1990, 49) de söz etmek gereklidir. Ara bellek kullanmayan işlemlerde, sunucu tek bir adres üzerinden kendisine gelen veriyi yine kendi içinde bulunan bir ara bellek alanına alarak işlemleri gerçekleştirmektedir (Şekil 2.4). Ancak işlemleri yaparken başka bir istemciden gelen veriyi, ara belleği kullanımda olduğu için alamayacak ve gelen veri kayıp olacaktır. Diğer yanda ara bellek kullanan işlemlerde, işletim sistemi tarafından sağlanan ara bellek kullanılmakta ve sunucu bir kuyruk oluşturan istekleri sırası ile bu ara bellek üzerinden alarak gerekli işlemleri yapmaktadır (Şekil 2.5). Mesaj kaybı ancak kuyruğun dolması durumunda söz konusudur.



Şekil 2.4 Ara belleksiz işlem



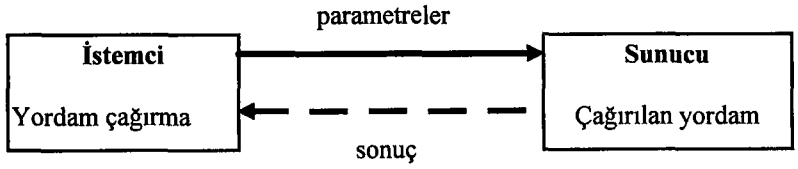
Şekil 2.5 Ara bellek kullanan işlem

İstemci-sunucu modeli işlemlerde deyinilmesi gereken son işlem grubu ise güvenilir (reliable) ve güvenilemez (unreliable) işlemlerdir. Burada temel olarak 3 ayrı yaklaşım söz konusudur. İlk yaklaşımda elden gelenin en iyisi yapılarak mesaj iletilmeye çalışılır. Ancak mesajın yerine varacağı konusunda hiçbir garanti verilmez. İkinci yaklaşımda ise

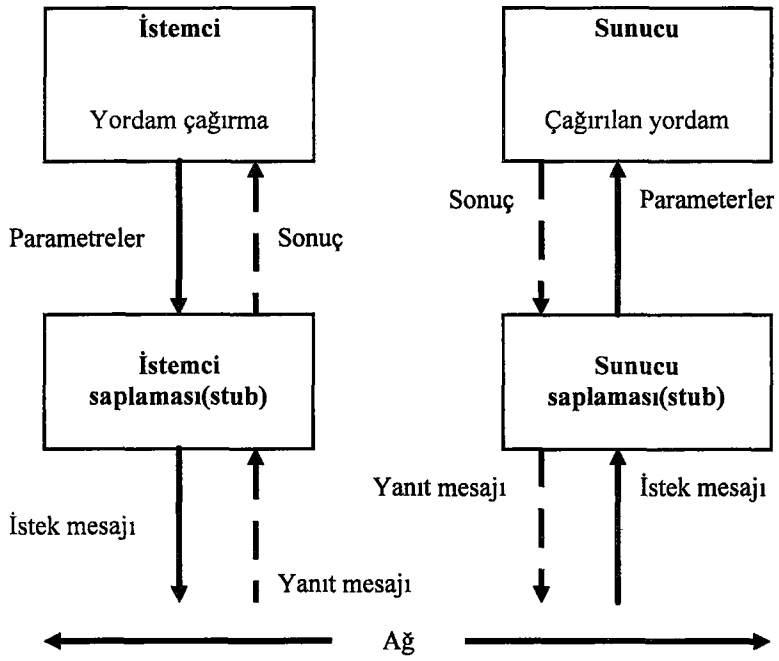
istemcinin yolladığı isteğe karşılık sunucunun işletim sistemi alındı (acknowledge) bilgisini takiben sunucu yanıtını istemciye yollar. Benzer bir metot ile istemci tarafının işletim sistemi de sunucunun yanıtının alınmasını takiben diğer tarafa alındı bilgisi yollar. Burada alındı bilgileri tamamen işletim sistemleri seviyesinde olduğu için ne istemci ne de sunucu bunları değerlendirmek durumunda değildir. Üçüncü ve son yaklaşımda istemci isteğini sunucuya yolladıktan sonra bloke olur. Sunucunun istemciye yanıtı yollamasını takiben istemcinin işletim sistemi sunucununkine alındı bilgisini yollar.

2.3.3 RPC (Remote Procedure Call - Uzak Yordam Çağırma) modeli

Uzak yordam (remote procedure) çağırma (RFC 1057; Bloomer, 1992) ile herhangi bir anda başka bir adres alanına (address space) dahil yordama veri yollayıp sonuç almak mümkündür. RPC ile ağ ortamında, başka bilgisayarların hesaplama güçlerinden de yararlanılarak dağıtık programlar yazılabilir. İstemci uygulaması uzak yordam ile konuşarak gerekli parametreleri bağlı bulunduğu ağ üzerinden aktarmakta, sonucun elde edilip ağ üzerinden istemciye iletilmesine kadar geçen sürede istemci beklemektedir. RPC modelinde yerel yordam (local procedure - Şekil 2.6) ile uzak yordam (remote procedure - Şekil 2.7) çağırma arasındaki farklılığı saklamak temel amaçtır. RPC, istek/yanıt haberleşme modelini kullanmaktadır. İstemci yordamı (client procedure) isteğini sunucu yordamına (server procedure) yollar ve ondan gelecek yanıtı bekler. İstemci ile sunucu arasındaki bu haberleşme her iki tarafta da bulunan saplamalar (stub) üzerinden sağlanır. Saplamalar, istemci veya sunucu tarafından yapılan yerel yordam çağrılarını bilgisayar ağı üzerinden çağrılabilecek şekilde düzenleyerek, bir üst katmandaki yordama şeffaf görünürler. Aşağıdaki şekiller yerel yordam (Şekil 2.6) ve uzak yordam çağırma (Şekil 2.7) işlemlerinin nasıl gerçekleştiğini göstermektedir.



Şekil 2.6 Yerel yordam çağırma

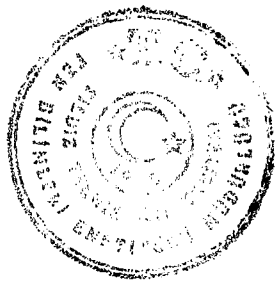


Şekil 2.7 Uzak yordam çağırma

2.4 Dağıtık Sistemin Üç Boyutu

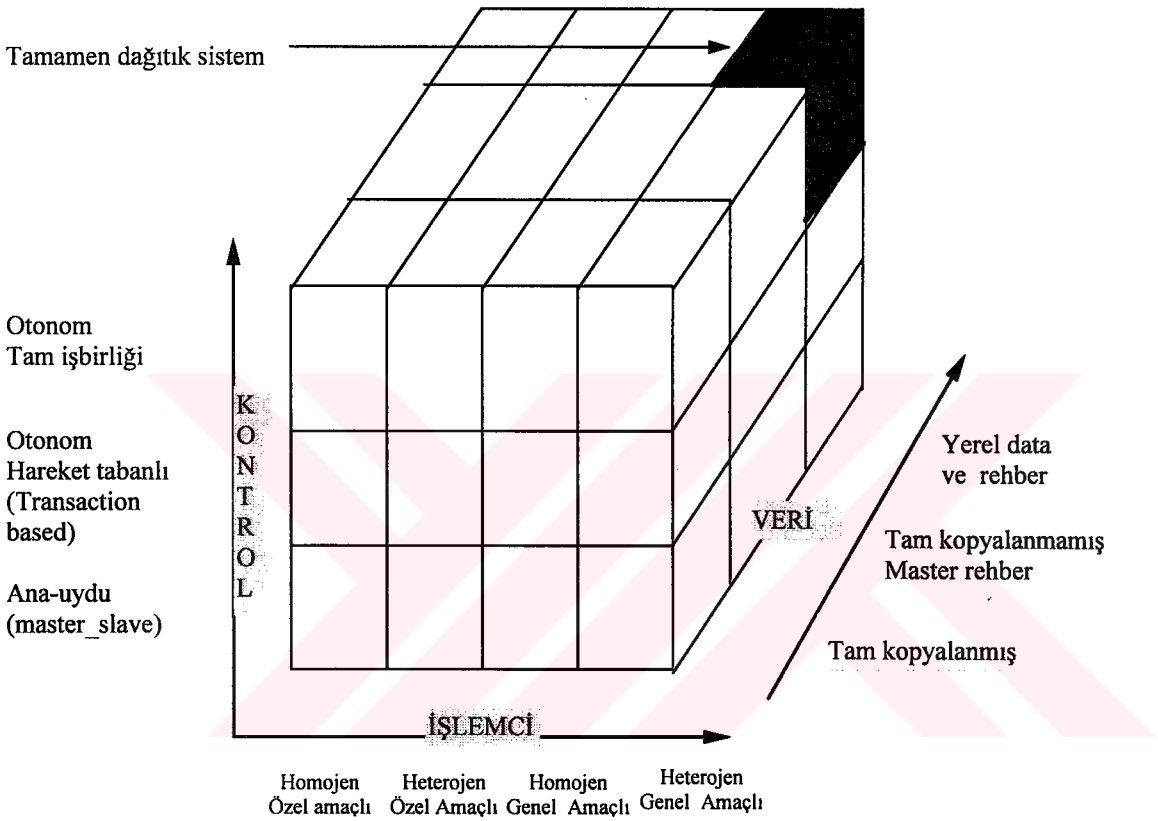
Dağıtık sistem, tarifi gereği, dağıtık donanım, kontrol mekanizması ve veri olmak üzere üç boyutta incelenebilir. (Sloman ve Kramer, 1987, 3)

1. **Donanım :** Kendi hafızası ve işlemcisi olan en az iki veya daha fazla bilgisayarın bulunması ve bunların bir haberleşme ağı ile birbirlerine bağlantılı olması gereklidir.
2. **Kontrol:** Hafıza, işlemci ve diğer çevre birimleri gibi fiziksel kaynakların yanı sıra dosyalar, işlem gibi mantıksal (logical) kaynakların da kontrol edilmesi gerekmektedir. Bunun için her bağımsız işlemci üzerindeki aktiviteleri kontrol edecek bir mekanizmanın hakim kılınması gerekir. Bu kontrol mekanizması merkezi, hiyerarşik olabileceği gibi tamamen işlemcinin kendisine de bırakılabilir.
3. **Veri:** Üzerinde hem sistem hem de uygulama yazılımının kontrolünün olduğu tek kaynaktır. İşlenecek verinin, farklı noktalarda kopyaları olabileceği gibi bazı parçaları bir noktada, diğerleri başka noktalarda da olabilir. (horizontal/vertical distribution)



2.5 Dağıtık Sistem Tipleri (Enslow, 1978)

Dağıtık sistemin üç boyutu üzerindeki desantralizasyon (decentralization) ölçüsü Şekil 2.8 de görüldüğü gibi dağıtıklık derecesini belirlemektedir.



Şekil 2.8 Enslow dağıtık sistem modeli

2.6 Dağıtık Uygulamaların Sınıflandırılması

2.6.1 Geniş taneli (Large grain)

Ortak bir amaca ulaşmak üzere çalışan işlemcilerin zamanlarının büyük kısmını hesaplamayla geçirdikleri ve birbirleri ile seyrek olarak mesaj alış-verişinde bulundukları türdeki uygulamalara verilen isimdir.

2.6.2 İnce taneli (Fine grain)

Ortak bir amaca ulaşmak üzere çalışan işlemcilerin zamanlarının daha az bir kısmını hesaplamayla geçirdikleri ve birbirleri ile daha sık olarak mesaj alış-verişinde bulundukları türdeki uygulamalara verilen isimdir.

2.7 Dağıtık Sistemlerin Avantaj ve Dezavantajları

Dağıtık sistemler, pek çok bilişimci tarafından iyi ve kötü yönleri ile, farklı zamanlarda incelenmiş, neler getirip neler götürdükleri kimi zaman iyimser, kimi zaman kötümser bakış açıları ile ele alınarak, kullanım alanları ve bu alanlarda karşılaşılan başarılar ve başarısızlıklar ile ortaya konmuştur. Literatürün (Sloman ve Kramer, 1987, 5-19; Campine, 1991; Tanenbaum, 1992, 363-365; Goscinski, 1992, 37-42; Nutt, 1992, 193-222; Black, 1993, 320-323) ve yapılan araştırmaların ışığında dağıtık sistemlerin ;

Avantajları:

1. Mikroişlemcili sistemlerin fiyat/performans oranları anabilgisayarlardan daha iyidir.
2. Farklı mekanlarda bulunan bilgisayarlardan oluşan dağıtık yapıdaki bir sistemin toplam işlem gücü, anabilgisayarlardan daha fazladır. Ayrıca toplam işlem gücü sisteme ilave edilecek ek bilgisayarlar/iş istasyonları ile kademe kademe, standart bir donanım ile arttırılabilmektedir (scaleability).
3. İş yükünün (work load) birden fazla işlem birimine (processing unit) dağıtılmasıyla süreklilik ve güvenilirlik sağlanır. İşlem birimlerinden bir veya birkaçının birden devre dışı kaldığı hallerde bile sistem çalışmaya devam eder.
4. Pahalı olan donanım birimleri (renkli yazıcı, çizici, tarayıcı vs..) paylaşılabilir.

Dezavantajları:

1. İşlem birimleri arasındaki iletişimin yarattığı yük, iletişim hatlarının istenmeyen kişiler tarafından dinlenmesinin (tap) yaratabileceği sistem güvenliğine (security) yönelik tehditler ve iletişim hatları üzerinde oluşabilecek hatalar dağıtık sistemlerin en zayıf yönleridir.
2. Dağıtık sistem üzerindeki bazı birimler fazla yüklenirken bazı birimlere yük düşmeyebilir.

3. Bazı yazılımlar ve veri, dağıtık sisteme dahil birden fazla bilgisayar üzerinde bulunabilir*.
4. İlk üç maddenin doğal bir sonucu olarak tüm sistem üzerindeki kontrol azalır.
5. Dağıtık sistemler için geliştirilmiş az sayıda yazılım vardır.
6. Dağıtık sistem parçalarının her birinin önemli bir işleme gücü olmasına rağmen, büyük ve merkezi bir sistemin sahip olduğu işlem gücüne, hiçbir birim tek başına ulaşamamaktadır.
7. Dağıtık sisteme dahil olan bilgisayar sistemleri arasında, gerek üreticiden, kaynaklanan (farklı işlemciler, farklı komut kümeleri vb..) gerekse bilgisayarları kontrol eden işletim sistemlerinin farklılıklarından kaynaklanan uyum problemleri olabilmektedir.

* Aslında bu tam bir dezavantaj olarak görülmemelidir. Programlar veya veri, sistem sürekliliğini sağlamak üzere bilinçli olarak kopyalanmış (replicated) veya işlem başarımını artırmak amacıyla yatay veya dikey olarak dağıtılmış olabilir (Horizontal/vertical distribution).

3. AĞ TABANLI SERVİSLER VE SERVİS TİPLERİ

Bir bilgisayar ağı üzerinde verilecek servisler, bağlı bulunulan ağ üzerinde uygulanacak ağ (network) protokolü ile sunucu tarafından verilecek servisin özelliğine göre farklılıklar göstermektedir. Şekil 3.1 de gösterilen, iki temel bileşen arasındaki ilişki incelendiğinde servisin türünü belirleyecek dört ayrı durum olduğu görülür.

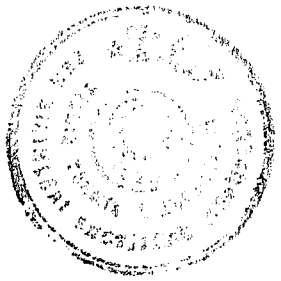
Tekrarlamalı Bağlantılı (Iterative Connection Oriented)	Tekrarlamalı Bağlantısız (Iterative Connectionless)
Eş Zamanlı Bağlantılı (Concurrent Connection Oriented)	Eş Zamanlı Bağlantısız (Concurrent Connectionless)

Şekil 3.1 Servis türü ve ağ protokolü arasındaki ilişki-1 (Comer et al, 1993,101)

Ancak eş zamanlı servisler için bir alt sınıflama da göz önüne alınacak olursa sayı altıya yükselecektir (Şekil 3.2).

			AĞ PROTOKOLÜ	
			Bağlantılı (Connection Oriented)	Bağlantısız (Connectionless)
SERVİS TÜRÜ	Tekrarlamalı (İterative)			
	Eş Zamanlı (Concurrent)	Tek İşlemli (Single Process)		
		Çok İşlemli (Multi Process)		

Şekil 3.2 Servis türü ve ağ protokolü arasındaki ilişki-2 (Bloomer, 1992, 111)



3.1 Bağlantılı Protokoller

Bağlantılı protokoller pek çok kaynakta telefon sistemine olan benzerlikleriyle ele alınmış ve iki uç arasında bilgi alış-verişinin, bir bağlantının sağlanması, kurulan bağlantı üzerinden bilginin aktarımı, aktarımın bitmesini takiben bağlantının iki tarafın da bilgisi dahilinde sonlandırılması aşamalarından oluştuğu vurgulanmıştır. Bağlantı, üzerinde güvenilebilir (reliable) aktarım yapmayı sağlayacak türden olmalıdır. Bağlantılı sistemler, bilginin bir uçtaki göndericiden diğer uçtaki alıcıya iletilmesi esnasında, yollanış sırasını koruması ve kayıpsız olarak iletilmesi esasına dayalı olarak çalışmaktadır. Internet protokolü (IP) (RFC 791) üzerinde kullanılan TCP (RFC 793) bu yapıya en iyi örnektir.

3.2 Bağlantısız Protokoller

Bağlantısız protokoller, bağlantılı protokollerin aksine, adından da anlaşıldığı gibi, veri aktarılmadan önce herhangi bir bağlantının kurulmasının gerekmediği protokollerdir. Arada, önceden kurulmuş doğrudan bir bağlantı olmadığı için bilgi aktarımının güvenilemez bir ortam üzerinden yapıldığı düşünülür. Kaynaklarda, posta servisi ile özdeşleştirilerek açıklanan bu protokolde yollanan bilginin yollandığı sırada yerine varacağı konusunda kesin bir kanıya varılamaz. Internet protokolü (IP) üzerinde kullanılan UDP (RCF 768) buna en iyi örnektir.

3.3 Tekrarlamalı (Iterative) Servisler

Sunucu tarafından kısa süre içinde cevap verilebilecek türdeki servislerde (saat/gün bilgisi gibi) sunucu işlem, sonsuz bir döngü içerisinde kendisine gelen istekleri yanıtlamak suretiyle tekrarlamalı servisi yerine getirir.

3.4 Eş Zamanlı (Concurrent) Servisler

Sunucu, servis verilmesi uzun sürebilecek türde istekler için, kendisinin bir kopyasını yaratarak (*fork()*) gelen isteğin gereğini yerine getirme işlemini ona bırakır. Bu sırada esas sunucu işlem kendisine gelebilecek diğer istekleri karşılamak veya yönlendirmek için beklemeye başlar. Aynı anda farklı işlerin yürütülebildiği bu tür servislere, eş zamanlı (concurrent) servisler adı verilmektedir. Ancak eş zamanlı servisleri de kendi içinde

yaratılan ve etkinleştirilen (activated) kopya sayısına bağlı olarak tek işlemli (single process) ya da çok işlemli (multi process) olarak iki alt gruba ayırmak mümkündür.

1. **Tek İşlemli (Single Process) Servisler:** Eş zamanlı serviste, sunucu işlem, gelen isteği karşılamak üzere sadece bir tek kopya yaratıp onu etkinleştirmiş ise tek işlemli (single process) servis adını alır.
2. **Çok İşlemli (Multi Process) Servisler:** Eş zamanlı serviste, sunucu işlem, gelen istekleri karşılamak üzere birden fazla kopya yaratmış ve bunlardan iki veya daha fazlası aynı anda etkin durumda ise, çok işlemli (multi process) servis adını alır.

4. TAŞIMA KATMANININ (TRANSPORT LAYER) KULLANIMI

Gerçekleştirilen ağ tabanlı uygulamanın üzerinde çalışacağı ağ ortamından yaralanabilmesi için iki temel yol vardır. Bunlar:

1. Tamamen yeni ve kendine has özellikleri olan bir protokol ve bunu destekleyecek olan, mevcut donanım ile uyumlu, sistem ve uygulama yazılımları geliştirmek (kısaca OSI referans modeli göz önüne alınacak olursa taşıma katmanı ve altında kalan katmanlarda yapılacak çalışmaları içerir) veya;
2. Var olan protokoller tarafından desteklenen alışılagelmiş metotları ve sistem çağrılarını (system calls) kullanmaktır.

Birinci seçenek oldukça büyük bir çabayla, standartların, mevcut ve ileride olabilecek teknolojik gelişmelerin göz önüne alındığı, diğer sistemlerle (gerek donanım, gerek sistem ve uygulama yazılımları düzeyinde) uyumlu olacak bir tasarım çalışması olup, kaçınılmaz bir ihtiyacı karşılamak üzere, yürütülecek bir ekip çalışmasını gerektirmesi nedeni ile nadiren tercih edilir. Diğer taraftan varolan protokoller ve sistem çağrılarından yararlanmak, gerek uyumluluk, gerekse ortaya çıkabilecek problemlerin daha rahat çözülebilmesi açısından büyük bir esneklik sağlamaktadır.

Taşıma katmanı (OSI referans modelinde dördüncü seviye katmanı) programcı tarafından bakıldığında, iki nokta arasında (end-to-end), güvenilir ve şeffaf (transparent) veri aktarımını sağlamakla görevlidir. Taşıma katmanında kullanılacak bir arayüz, bir alt katmanda (ağ katmanı-network layer) kullanılan ağ protokollerinin (IP, IPX, SPX vb..) detaylarını kullanıcıya yansıtmaz ve gerçeklenecek uygulamaların geliştirilmesinde kolaylık sağlar. Taşıma katmanında kullanılabilecek, standart iki arayüz mevcuttur. Bunlar, TLI (Transport Layer Interface) (Day vd., 1993; Novell, 1993b) ve Berkeley Soketleri dir.

4.1 Berkeley Soketleri (Berkeley Sockets)

4.1.1 Tarihçesi

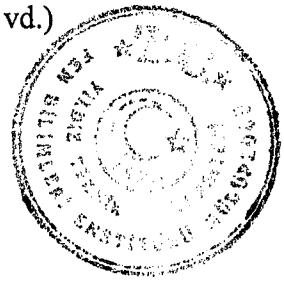
1980'li yılların başında Advanced Research Projects Agency (ARPA), Berkeley'deki Kaliforniya Üniversitesinde oluşturduğu bir grup ile TCP/IP yazılımının UNIX işletim sistemine aktarılmasını ve üretilecek yazılımın diğer sistemlerde de kullanılabilmesini sağlamak amacıyla bir çalışma başlattı. Grup, mevcut UNIX sistem çağrılarını mümkün olduğunca kullanmaya gayret göstererek, gerekli yerlerde TCP/IP çağrılarını destekleyecek yeni sistem çağrılarını (system calls) oluşturdu. Yeni eklenen sistem çağrıları, soket (socket) olarak anılmaya (bazı kaynaklarda Berkeley Socket olarak da geçmektedir) başlandı. Soket destekli yeni UNIX ise Berkeley UNIX ya da BSD UNIX olarak anılmaya başlandı.

4.1.2 Amacı

Soket'in temel kullanım amacı işlemler arası iletişimi (IPC - Interprocess Communication) dosya işlemlerindekine benzer bir giriş/çıkış (I/O) mekanizmasına benzetmektir. Dosyalarda olduğu gibi her aktif soket, soket tanımlayıcısı (socket descriptor) olarak adlandırılan bir tam sayı ile belirlenmektedir. UNIX işletim sistemi dosyalar için kullanılan tanımlayıcı numaralar ile soketler için kullanılanları aynı tabloda tuttuğu için hiçbir zaman bir dosya ile bir soket aynı tanımlayıcı numarasına sahip olamaz.

4.1.3 Soket tipleri

Soketlerin nasıl kullanılacağını onları yaratan uygulama belirler. Verilecek servis tipine göre değerlendirildikleri zaman soketleri bağlantılı veya bağlantısız olarak sınıflandırmak mümkündür. Diğer bir sınıflama ise, ilk yapılandan tamamen bağımsız olan ve yaratılan soketin üstleneceği role göre yapılmış olandır. Bu sınıflamada aktif ve pasif soket olarak iki ayrı tip karşımıza çıkmaktadır. Her ne kadar ilk bakışta dört farklı tip varmış gibi görünse de soket tipini belirleyen temel etken, üstlendiği roldür. Soket yaratıldıktan sonra ister bağlantılı ister bağlantısız olsun, soketi kullanan iki işlem aralarında mesaj alış verişinde bulunur. (Besaw, 1987; Comer ve Stevens, 1993; Sechrest; Leffer vd.)



4.1.3.1 Aktif soket (Active socket)

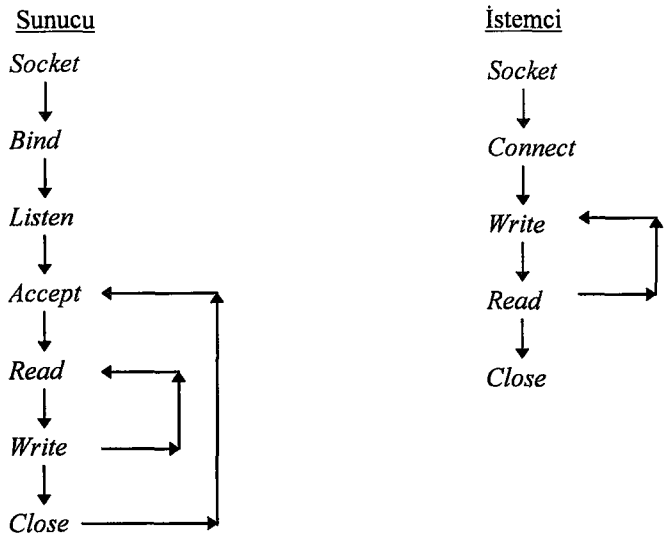
Soket, bir bağlantıyı başlatmak için yaratılmış ise, aktif soket olarak adlandırılır. İstemci/sunucu yapısında aktif soket, istemci tarafından kullanılan soket tipidir.

4.1.3.2 Pasif soket (Passive socket)

Yaratılan soket, kendisine gelen bağlantı isteklerini cevaplamakla görevlendirilmişse pasif soket olarak adlandırılır. İstemci/sunucu yapısında pasif soket sunucu tarafından kullanılan soket tipidir.

4.1.4 Sunucu ve istemcinin soket kullanımında izlemeleri gereken sıra

Soketler, sunucu ve istemci tarafından takip edilmesi gereken bir sıraya göre kullanılır (Şekil 4.3). Bu sıra kullanılan soketin aktif ya da pasif olması ile ilgilidir. Sunucu tarafında yaratılan soket pasif olup gelen isteklere cevap vermekle yükümlü olduğu için temel veri aktarım sırası *Read* (soket üzerinden gelen bilginin alınması) *Write* (soket üzerinden gelen isteğe karşı düşen sonucun yollanması) şeklindedir. Bu bilgi alış-veriş işlemi, aktarılması gereken bilgi tamamlanıncaya kadar tekrarlanır. Sunucu kısmı bu işlemlerin yanı sıra ayrı bir çevrim içinde de gelecek diğer bağlantıları bekler (*Listen* sistem çağrısının bir gereği olarak). Bu nedenle *Accept* ve *Close* çağrıları arasında sonsuz bir döngü vardır. İstemci tarafında yaratılan soket aktif olup, bir bağlantı isteğini herhangi bir sunucuya yönlendirmekle yükümlüdür. Bunun sonucunda, sunucuda karşılaşılan durumun aksine, bilgi alma (*Read*) ve bilgi yollama (*Write*) sistem çağrılarının yerleri değişerek, *Write - Read* şeklinde sıralanacaklardır. Sunucudaki yapıya benzer olarak, aktarılması gereken bilginin miktarına göre, *Write - Read* sistem çağrıları arasında birkaç defa gidip gelinebilir. İstemci tarafından yaratılan soket, aktarımın tamamlanmasını takiben sonlandırılacaktır.



Şekil 4.1 Sunucu ve istemcide soket ile ilgili çağrıların kullanım sırası

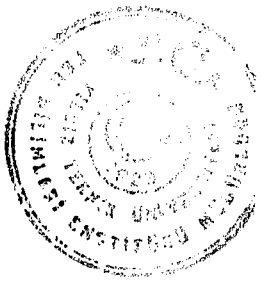
4.1.4.1 Soket çağrılarını kullanabilmek için yapılması gerekenler

C programlama dilinden kullanılacak soket çağrıları, *socket.h* isimli dosyada tariflenmiştir. Yazılan programlara *socket.h* dosyasının ilave edilerek derlenmesi ve üretilen nesne kodun (object code) çalışılan işletim sistemi için daha önceden derlenmiş olan soket kütüphanesi ile link edilmelidir. Soket arabirimi ile ilgili sık kullanılan bazı çağrılar, kullanım amaçları ve şekilleri şöyledir:*

4.1.4.2 Soket'in yaratılması

Soket, kendisini kullanacak olan uygulamanın ne tip bir ağ bağlantısı üzerinden (bağlantılı/bağlantısız) servis vereceğine bağlı olarak yaratılmalıdır. Uygulamanın hangi tür adres etki alanında (address domain) olacağı da önemlidir.

* Ayrıntılar için Berkeley Soketi ile ilgili referanslara bakınız.



sock =socket (int domain, int type, int protocol);

int sock;

domain: Yaratılan soketin hangi adres ailesine bağlı olduğunu belirlemektedir. Alabileceği değerler:

AF_UNIX	(UNIX türünde)
AF_INET	(DARPA Internet türünde)
AF_OSI	(OSI türünde)

type: Yaratılan soketin hangi tip bir ağ bağlantısı üzerinden kullanılacağını belirler. Bağlantılı, bağlantısız olup olmadıkları bu parametre sayesinde belirlenir. Alabileceği değerler:

SOCK_STREAM	(stream - Bağlantılı)
SOCK_DGRAM	(datagram - Bağlantısız)
SOCK_RAW	
SOCK_RDM	(reliable datagram - Güvenilir bağlantısız)
SOCK_SEQPACKET	(sequenced packet stream - Sıralı paket dizisi)

Protocol: Mevcut protokole ilişkin özellikleri kullanması için genelde 0 değeri verilir.

4.1.4.3 Yaratılan soket üzerinden bağlantı kurulması

Soketlerin, kullanım amacına bağlı olarak aktif ya da pasif olarak adlandırıldıklarından daha önce söz etmiştik. Soketlerin yaratılması her iki durumda da aynı olmakla birlikte yaratılan soketler üzerinden bağlantı kurulması aşamasında belli farklılıklar mevcuttur.

4.1.4.3.1 Aktif soket kullanımı

Aktif soket, herhangi bir işlemin bir diğeri ile iletişim başlatmasını sağlamak üzere kullanılmaktadır. Bu tür soketler istemci tarafından kullanılmaktadır.

*int connect (int sock,struct sockaddr *name, int namelen);*

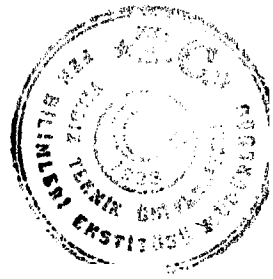
sock: Önceden *socket()* çağrısı ile yaratılan soketi belirlemek için kullanılan numaradır.

name: Bağlı bulunulan adres ailesinin özelliklerini taşıyan bir yapıyı göstermek üzere kullanılır. Eğer socket AF_INET ailesine ait bir şekilde yaratılmış ise *sockaddr_in* yapısı, AF_UNIX ailesine ait şekilde yaratılmış ise *sockaddr_un* yapısı kullanılır.

namelen: *name*'in gösterdiği yapının büyüklüğünü belirlemek için kullanılır.

```
struct sockaddr_un   {      short   sun_family ;
                            char    sun_path [108-4] ;   }
struct sockaddr_in   {      short   sin_family ;
                            u_short             sin_port ;
                            struct  in_addr     sin_addr ;
                            char    sin_zero[8] ;  }
struct in_addr {      union  {
                            struct { u_char s_b1,s_b2,s_b3,s_b4 ;} S_un_b;
                            struct {u_short s_w1,s_w2; } S_un_w;
                            } S-un;
                      #define       s_addr S_un.S_addr }
```

Soket yaratıldığında herhangi bir adrese sahip değildir. Bu durumda başka bir işlem bu soket ile bağlantı kuramaz. Bu nedenle soketin yaratılmasını takiben bir adres ile sisteme bağlanması (tanıtılması) gereklidir. Bu işlem *bind()* çağrısı yardımıyla yapılarak soket sisteme ve dolaylı olarak da bağlantı kuracak diğer işlemlere tanıtılmış olur. Yaratılan soketin sisteme tanıtılması için şu işlemler yapılmalıdır.



*int bind (int sock, struct sockaddr *address, int addrlen);*

sock: Soket numarasıdır.

address: Sokete verilen adres bilgisinin tutulduğu yapıyı gösterir.

addrlen: *address* yapısının uzunluğunu belirleyen değerdir.

```
struct sockaddr {    short   sa_family ;    /* adres ailesini belirler */
                     char    sa_data[14];   /* 14 byte adres belirler */
                }
```

4.1.4.3.2 Pasif soket kullanımı

Sunucu üzerinde, gelecek olan istekleri cevaplamak için yaratılacak soket, pasif sokettir. Pasif soket, bağlı bulunduğu noktaya gelecek olan istekleri karşılamak üzere bekleyecektir. Çağrı bekleyen pasif soketler yaratıldıktan sonra *listen()* çağrısı ile bağlı bulundukları adresten aktif bir soketin bağlantı kurmasını beklerler.

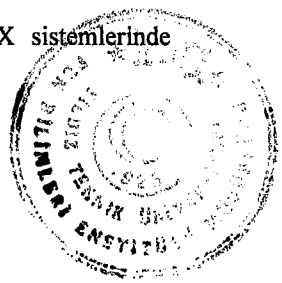
int listen (int sock, int queuelen);

sock: Soket numarasıdır.

queuelen: Soket üzerinden servis almak için bekleyebilecek bağlantı isteklerinin azami sayısını belirleyen değerdir*.

listen() çağrısı ile bağlantı bekleyen soket, gelen bağlantı isteklerini *accept()* çağrısını kullanarak kabul eder. Bu işlem, kuyrukta beklemekte olan ilk istemciye servis vermek için yeni bir soket yaratır. Soketin yaratılması herhangi bir şekilde daha önceden yaratılanların çalışmalarını etkilemez. Eğer kuyrukta bekleyen herhangi bir istemci yok ise *accept()* çağrısı bir istek oluşuncaya kadar bloke olur.

* Bu değer *sys/socket.h* içerisinde *SOMAXCONN* değişkeni ile belirlenmiş olup UNIX sistemlerinde alabileceği en büyük değer 5 olarak verilmiştir.



*new_sock = accept (int sock, struct sockaddr *name,int namelen);*

int new_sock;

new_sock: *accept()* çağrısı *sock* ile aynı özellikte bir soket yaratarak kuyrukta beklemekte olan bir bağlantıya servis vermeye başlar.

name: Bağlantı için kuyrukta bekleyen sokete ait (active socket) bilgiler *name* isimli yapıda bulunmaktadır.

namelen: *name* yapısının büyüklüğünü belirler.

4.1.4.4 Veri transferi

Kullanım amacına göre (aktif/pasif) olarak yaratılan soketin sistem ile bağlantısının (*bind()*) sağlanmasını takiben soket üzerinden veri aktarımı yapmak için (*read(),write()*), (*recv(),send()*), (*recvfrom(),sendto()*), (*recvmsg(),sendmsg()*) ve (*readv(),writev()*) gibi çağrı çiftlerinden yararlanılmaktadır. Bu çağrıların basit tanımları şöyledir:

*int read (int sock, char *buff, int bufflen);*

*int write (int sock, char *buff, int bufflen);*

sock: Soket numarasıdır.

buff: Veri transferi için kullanılacak olan ara belleği (buffer) gösterir.

bufflen: *buff* değişkeninin büyüklüğünü belirler.

*int recv (int sock,char *buff, int bufflen, int flags);*

*int send (int sock, char *buff, int bufflen, int flags);*

sock: Soket numarasıdır.

buff: Veri transferi için kullanılacak olan ara belleği (buffer) gösterir.

bufflen: *buff* değişkeninin büyüklüğünü belirler.

flags: Kontrol bilgisinin bit yapısında tutulduğu değişkendir.

*int recvfrom (int sock, char *buff, int bufflen, int flags,*
*struct sockaddr *from, int fromlen);*
*int sendto (int sock, char *buff, int bufflen, int flags,*
*struct sockaddr * to, int tolen);*

sock: Soket numarasıdır.

buff: Veri transferi için kullanılacak olan ara belleği (buffer) gösterir.

bufflen: *buff* değişkeninin büyüklüğünü belirler.

flags: Kontrol bilgisinin bit yapısında tutulduğu değişkendir.

from: Veriyi gönderenin adres bilgisinin tutulduğu yapıyı gösterir.

fromlen: Veriyi gönderenin adres bilgisinin uzunluğunu belirler.

to: Veriyi alacak olanın adres bilgisinin tutulduğu yapıyı gösterir.

tolen: Veriyi alacak olanın adres bilgisinin uzunluğunu belirler.

*int recvmsg (int sock, struct msghdr *msg, int flags);*
*int sendmsg (int sock,struct msghdr *msg, int flags);*

sock: Soket numarasıdır.

msg: Mesaj yapısını gösterir.

flags: Kontrol bilgisinin bit yapısında tutulduğu değişkendir.

```
struct msghdr  {    caddr_t       msg_name ;
                    int           msg_namelen;
                    struct iovec  *msg_iov;
                    int           msg_iovlen;
                    caddr_t       msg_accrights;
                    int           msg_accrightslen;}
```

4.1.4.5 Soket kullanımının sonlandırılması

Kullanılan soketi sonlandırmak için iki ayrı çağrı mevcuttur. Bunlardan *close()*, veri transferinin tamamlandığı durumlarda, yaratılan soket bağlantısının kurallara uygun bir şekilde sonlandırılmasını sağlarken, *shutdown()* mevcut soketi durdurmakta (abort), eğer mevcut bir veri akışı var ise bilginin kayıp olmasına veya bozulmasına sebep olmaktadır. Bu çağrıların genel kullanım şekilleri şöyledir:

int close (int sock);

int shutdown (int sock,int how);

sock: Soket numarasıdır.

how: Bağlantının nasıl sonlandırılacağını belirler.

0 → soket üzerinden bilgi yollanmayacağını,

1→ soket üzerinden bilgi alınmayacağını,

2→ soketin tam anlamıyla sonlandırılacağını belirler.

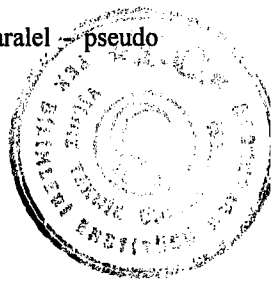
5. İSTEMCİ/SUNUCU MODELİNDE EŞ ZAMANLI İŞLEM (CONCURRENT PROCESSING)

Eş zamanlılık kavramı, aynı veya farklı amaçlara yönelik hesaplamaların aynı anda yapılması veya yapılıyormuş gibi görünmesini ifade etmek için kullanılmaktadır. Çok kullanıcılı bilgisayar sistemlerinde, tek işlemci, farklı birkaç hesaplama arasında hızlı bir şekilde anahtarlanarak (switching), zaman paylaşımı (time-sharing) yardımıyla bu etki* sağlanırken, birden fazla işlemcisi olan bilgisayar sistemlerinde, aynı anda her işlemcide farklı bir hesaplama yapılabilmektedir (paralel processing). Eş zamanlı işleme, dağıtık işlemenin en temel şekli olup, farklı biçimlerde karşımıza çıkmaktadır. Bazen bir bilgisayar ağı üzerindeki farklı bilgisayarların birbirleri ile mesaj alış-verişine dayanan işlemler olarak, bazen de çok kullanıcılı, zaman paylaşımlı bilgisayar sistemlerinde, bir istemci programın herhangi bir sunucudan servis alırken bir başka sunucu programın da başka bir istemciden gelen servis isteğine cevap vermesi olarak karşımıza çıkmaktadır. Kısaca kullanıcının bakış açısından görünen, fiziksel uzaklıklar dikkate alınmaksızın, bütün bu parçacıkların aynı anda çalışmasıdır.

Eş zamanlılık kavramı, istemci/sunucu modelinde iki yönüyle ele alınmalıdır. İstemci programları, bir bilgisayar ağı veya çok kullanıcılı bir sistem üzerindeki kullanıcı tarafından çalıştırılan herhangi bir uygulamadan temelde hiçbir farklılık göstermezler. Dolayısıyla, istemci programlarının, eş zamanlı çalıştırılmasında herhangi bir kısıtlama veya sistem seviyesinde zorluk yoktur. Diğer yandan, sunucu programlarının amacı, farklı noktalardan kendisine gelen istekleri değerlendirmek ve sonuç üreterek en kısa sürede en fazla isteğe cevap vermektir. Sunucu programı tasarlanırken, bağlantı isteklerinin gelme sıklıkları, gelen isteği karşılamak için sunucunun harcaması gereken zaman, istemci ile sunucu arasındaki iletişimin cinsi (bağlantılı/bağlantısız) ve sunucunun çok isteği az zamanda ve doğru cevaplama gerekliliği göz önüne alınmalıdır. Kimi zaman birbiri ile doğrudan bağlantılı, ancak çoğu zaman tamamen bağımsız işlemleri, hesaplamaları yerine getirmesi beklenen sunucu programı, klasik, tek kontrol merkezli (monolithic) bir kod parçası ile bu amaca yeterince ulaşamayacağı için eş zamanlılığın mevcut imkanlar ile sağlanması gereklidir.

---

* Kaynaklarda, tek işlemci üzerinde paralel iş yapılıyormuş hissi veren yapılar, sözde paralel – pseudo parallel olarak da tariflenmektedir.



5.1 Kavramlar

Konu ile ilgili kaynaklarda, tek bir kavramı ifade etmek üzere farklı terimler kullanılmaktadır. Gerek kavramları ortaya koyabilmek ve bunların arasındaki ilişkileri vurgulamak, gerekse kavramı ifade eden terimi belirleyebilmek açısından bu kavramların ve terimlerin açıklanmasında fayda vardır.

5.1.1 Kullanıcı kavramları

Kullanıcı tarafından bakıldığında karşılaşılan kavramlar şunlardır:

5.1.1.1 Program kavramı

Program, hedeflenen amaca varmak için gerçekleştirilecek adımların (algoritma), kullanılacak programlama dilinin özellikleri göz önünde bulundurularak, ardışık bir biçimde yazılmasıyla ortaya çıkan koda verilen addır.

5.1.1.2 Yordam (Procedure) kavramı

Yordamlar, yazılacak programlarda kolaylık sağlamak, anlaşılırlığı artırmak, modüler programlama tekniğine bağlı olarak yeniden kullanılabilirliği (reuse) arttırmak amacıyla, programlama dilleri tarafından desteklenen yapılardır. Program kavramı ile aynı özellikleri göstermekle birlikte, giriş ve çıkış değerleri iyi tariflenmiş, nispeten küçük programlar olarak değerlendirilebilirler.

5.1.2 İşletim sistemi kavramları

İşletim sistemi tarafından bakıldığında karşılaşılan kavramlar şunlardır:

5.1.2.1 İşlem (Process) kavramı

Kaynaklarda işlem (process) kavramı kimi zaman görev (task), kimi zaman iş (job), olarak tanımlanmış olsa da temel hesaplama birimini ifade etmektedir (Comer, 1993,25). İşlem ile ilgili en temel bilgi, işletilmekte olan kodun bulunduğu bellek adresini gösteren, komut göstergesi (instruction pointer) dir. Buna ek olarak her işlemin, kendine ait bir adres alanı, saklayıcıları (registers), yığını (stack) mevcuttur. Kısacası, yapmakla yükümlü olduğu işleri

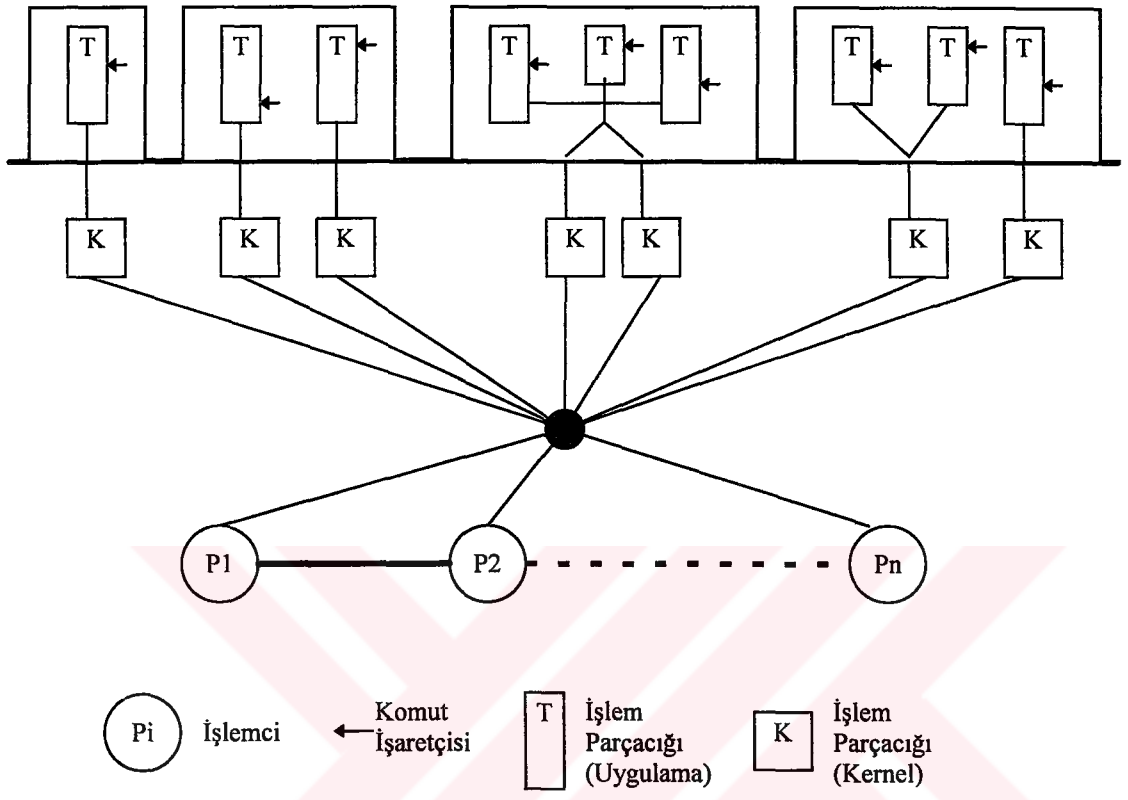
yerine getirebilmek amacıyla ihtiyaç duyduğu unsurlara (komut göstergesi, yığın, saklayıcılar) sahip kod parçacığı işlem olarak adlandırılmaktadır.

Kullanıcı, programını çalıştırmak istediği zaman, işletim sistemi yaratacağı işleme, çalıştırılması istenen program kodunun kontrolünü verir ve program işlem tarafından çalıştırılır. Bu nedenle zaman zaman işlem ve program kavramları karıştırılabilir. Program ile işlem arasındaki temel farklılığı kavramak için UNIX işletim sisteminde dosya ve dizin bilgilerini görmeyi sağlayan *ls* komutunu ele alalım. *ls* komutu iki ayrı noktadan çalıştırıldığında, *processA* ve *processB* olarak adlandıracağımız iki işlem, işletim sistemi tarafından yaratılacaktır. Her işlem *ls* komutunun gereğini yerine getirmek için aynı program parçasını işlemeye başlayacaktır. Farklı zamanlarda çalışmaya başlayan *processA* ve *processB* işlemleri herhangi bir *t* anında aynı kodun farklı noktalarında olmalarına rağmen, işlemekte oldukları kod parçası ile ilgili, birbirlerinden bağımsız değişkenler üzerinde hesaplama yapıyor olacaklardır. Bu durum, mevcut program kodunun tek bir işlem tarafından yürütülmesinden farklıdır. Verdiğimiz örnekte de olduğu gibi eş zamanlı işleme söz konusu olduğunda, bir program birden fazla işlem tarafından ele alınarak, kendilerine ait adres alanı içinde, kendi yığın ve saklayıcı kümeleri kullanılarak işlenebilmektedir.

5.1.2.2 İş/işlem parçacığı (Thread) kavramı

İşlem parçacığı kavramı, sıralı işleme ve bloke sistem çağrıları ile paralelliğin birleştirilmesini sağlamak için ortaya atılmış bir kavramdır. Temelde, sıralı çalışan, kendi komut göstergesi ve yığını olan mini işlemdir (Şekil 5.1). İşlemciyi, aynen işlemlerin kullandığı gibi kullanırlar, bazı sistem çağrılarının oluşmasını beklerken bloke olurlar. Tek bir işlem parçacığının kullanılması durumunda, tek işlem parçacıklı (single-threaded), iki veya daha fazla işlem parçacığının kullanılması durumunda çok işlem parçacıklı (multithreaded) olarak adlandırılırlar. Çok işlem parçacıklı yapı sayesinde, işlem birbirinden bağımsız, birden fazla işlem parçası tarafından yürütülmektedir. İşlem parçacıkları bağlı bulundukları işleme ait adres alanını paylaşırlar. Bu nedenle, birinin yaptığı değişiklikten aynı işleme bağlı bulunan bir diğer işlem parçacığı da anında haberdar olur. İşlem parçacıklarının aynı adres alanını paylaşmasının bir sonucu olarak, bir işlem

parçacığının yığın bilgisi bir diğeri tarafından bozulabilir. Yani işlem parçacıkları arasında bir koruma (protection) yoktur. (Tanenbaum, 1992, 508)



Şekil 5.1 Çok işlem parçalı (Multithread) sistemin yapısı

5.2 Çok İşlem Parçacıklı (Multithread) Yapının Üstünlükleri

Çok işlem parçacıklı yapının üstünlükleri şu şekilde sıralanabilir; (Sunsoft, 1994)

1. ***Uygulamanın cevap verme süresini (response time) iyileştirir:*** Bir uygulama, birbirinden bağımsız olarak çalışabilecek parçacıklardan oluşarak gerçekleştirilirse, sonuç üretme süresi kısalacaktır. Grafik arayüzlü (Graphical User Interface) programlarda açılmış bir pencerenin (window) kontrolünün, yeni açılacak olan pencereninkinden farklı bir işlem parçacığı tarafından yürütülmesi buna en iyi örneği teşkil etmektedir. Her ne kadar her iki pencereyi kontrol etmek için kullanılan kod parçası aynı olsa da, eş zamanlı olarak farklı işlem parçacıkları tarafından kontrol edilmektedir. Bu sayede pencereler üzerinde yapılan işlemlerin kontrol karmaşıklıkları birbirlerini etkilememektedir.

2. ***Çok İşlemcili sistemlerin etkin kullanımını sağlar:*** İşlem parçacıkları yardımıyla eşzamanlılığı sağlayan uygulamalar, donanımın kaç işlemciye sahip olduğundan bağımsız olarak geliştirilmektedir. Uygulamanın başarımı, işlemci sayısının artması ile orantılı olarak artacaktır. Matris çarpımı gibi yüksek paralel işleme düzeyi gerektiren uygulamaların çalıştırıldığı çok işlemcili sistemlerde, işlem parçacıklı olarak geliştirilen uygulamalar daha hızlı çalışabileceklerdir.
3. ***Programın yapısını geliştirir:*** Pek çok program tek ve bölünemez bir kod parçası oluşturacak şekilde yazılmak yerine, birbirinden tamamen veya yarı bağımsız modüller kullanılarak yazılacak olursa ileride doğabilecek kullanıcı istekleri ya da program içerisinde yapılacak değişikliklere o derece kolay uyarlanabilir. İşlem parçalıklı olarak tasarlanıp üretilen programlar bu nedenle programın yapısında olumlu bir değişiklik yaratmaktadır.
4. ***Daha az sistem kaynağı kullanır:*** Her program en az bir yordam, her işlem de en az bir işlem parçacığından oluşur. Aynı veri üzerinde çalışan, iki işleme sahip bir programın her işlemi için bağımsız adres alanı yaratılacak ve bunlar üzerinde işletim sisteminin kontrolünü sağlamak üzere bazı durum tabloları tutulacaktır. Programa dahil olan bu iki işlem parçacığı için harcanan sistem kaynakları (gerek bellek, gerekse durum değişkenleri ile ilgili değişiklikler için harcanacak işlem zamanı olarak), işlem parçacığı yaratmaktan daha fazladır. Ayrıca bu türde bir programın kendi içinde parçalara ayrılması (inherent separation) halinde, programcının farklı işlemlere dahil işlem parçacıkları arasındaki anuyumluluğu sağlamak için yoğun bir emek harcaması gerekir. İşlem parçacıkları, bağlı bulundukları işlem ile aynı adres alanını paylaşırlar. Bu sayede herhangi bir işlem parçacığının adres alanı üzerinde yaptığı bir değişiklikten diğer bir işlem parçacığı anında haberdar olmaktadır. Diğer bir önemli nokta ise işlem parçacıkları arasındaki anahtarlama işleminin adres alanının değiştirilmesini gerektirmemesidir. Oysa işlemler arasında yapılacak anahtarlamalarda adres alanı değişmekte, bu nedenle işlemler işlem parçacıklarına oranla daha fazla kaynak kullanmaktadır.
5. ***Dağıtık uygulamalara imkan sağlaması :*** İşlem parçacıklı yapının herhangi bir şekilde uzak yordam çağırma (RPC) veya benzeri bir arabirim ile bir arada kullanılması, paylaşımlı hafızaya sahip olmayan çok işlemcili sistemlerde (mesela bir bilgisayar ağı

ile birbirlerine bağlı bulunan iş istasyonlarından oluşturulmuş bir yapı içerisinde) dağıtık uygulamalar geliştirilmesine imkan sağlar.

6. ***Başarım artışı (Performance improvement) :*** SPARCStation üzerinde (Sun 4/75) yapılan ölçümlerde işlem yaratmanın, işlem parçacığı yaratmadan 5-30 kat daha uzun zaman aldığı, anuyumluluk zamanının ise 3 kat fazla olduğu gözlenmiştir (Tablo 5.1 ve Tablo5.2).

Tablo 5.1 İşlem-parçacığı yaratma zamanı (Sunsoft, 1994)

Yapılan İş	Süre (µs)
İşlem-parçacığı yaratma (Create unbound* thread)	52
İşlem-parçacığı yaratma (Create bound thread)	350
İşlem yaratma (fork())	1700

Tablo 5.2 İşlem-parçacığı an uyumluluk zamanı (Sunsoft, 1994)

Yapılan İş	Süre (µs)
İşlem-parçacığı (unbound thread)	66
İşlem-parçacığı (bound thread)	390
İşlemler arası	200

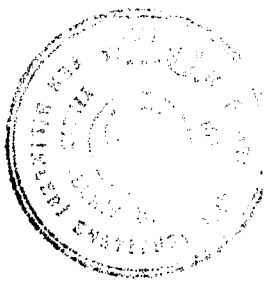
5.3 Çok İşlem Parçacıklı (Multithread) Programlama

Çok işlem parçacıklı program yazmak için UNIX işletim sisteminde *thread* kütüphanesinden yararlanmak gereklidir. Ancak kullanılan işletim sistemine göre kütüphanelerde tanımlanan sistem çağrıları farklılıklar göstermektedir (Sunsoft, 1994; Wagner ve Towsley, 1995). C programlama dili kullanılarak üretilen programlarda *thread.h* isimli dosya programa eklendikten sonra derlenir ve oluşan nesne kodu (object

* İşlem parçacıklarının, daha önceden yaratılmasıyla oluşan havuzdan (pool) kullanılması unbound; istek sonucu yaratılması bound terimleri ile ifade edilmektedir.

code) kullanılan işletim sistemi için oluşturulmuş olan *thread* kütüphanesi ile link edilir. Bu nedenle işlem parçacıklı programlama ile ilgili olarak sık kullanılan çağrıların neler olduklarından çok, ne amaç için kullanıldıkları önem kazanmaktadır. Kullanım amaçlarına göre *thread* işlemleri şunlardır:

- ***İşlem parçacığı yaratma*** : İşlem parçacığını kullanabilmek için öncelikle yaratmak gereklidir. İşlem parçacığı yaratılırken çalıştırmakla sorumlu olduğu yordamın adının ve bu yordama aktarılacak parametrelerin verilmesi gerekmektedir.
- ***İşlem parçacığının işlemci hakkından vazgeçmesi*** : İşlem parçacığı herhangi bir anda işlemciyi kullanma sırası kendisine geldiğinde hakkından vazgeçerek bir başka işlem parçacığının işlemciyi kullanmasını sağlayabilir. Bu sayede işlemcinin boş yere bekletilmesi (busy-waiting) engellenecek, işlemci sırada bekleyen bir diğer işlem parçacığına bırakılacaktır.
- ***İşlem parçacığını askıya alma*** : Bu çağrı yardımıyla herhangi bir işlem parçacığının çalışması askıya alınabilir. Bir işlem parçacığı askıya alınmış ise 'askıya alma' çağrısını takip eden ('çalışmaya devam et' çağrısı haricindeki) çağrının hiçbir etkisi olmayacaktır.
- ***İşlem parçacığının çalışmaya devam etmesi*** : Çalışması askıya alınan herhangi bir işlem parçacığı 'çalışmaya devam et' çağrısı yardımıyla tekrar çalışmaya başlar.
- ***İşlem parçacığına sinyal yollama*** : Aynı işlemin iki işlem parçacığı birbirlerine sinyal yollayarak çalışıp çalışmadıklarını anlayabilirler.
- ***İşlem parçacığını sonlandırma*** : Kullanıcı, işlem parçacığını, herhangi bir anda 'sonlandırma' çağrısı ile sonlandırma imkanına sahiptir. İşlem parçacığının sonlandırılmasını takiben onunla ilişki kesilir.
- ***İşlem parçacığının sona ermesini bekleme*** : Aynı işlemin işlem parçacıklarından biri, diğerinin sona ermesini bekleyebilmek için 'işlem parçacığının sona ermesini bekle' çağrısını kullanmalıdır. Bu çağrıyı kullanan işlem parçacığı, beklediği işlem parçacığı sona erinceye kadar bloke olur.



5.4 İşlemler Arası İletişim (Interprocess Communication)

Ortak amaca erişmek üzere çalışan işlemler, genel yapı içerisinde birbirleri ile belli noktalarda ara sonuç veya sonuç aktarımı için haberleşmek zorundadırlar. Örneğin iş hattı (pipeline) kullanılarak yürütülmesi gereken bir işlemin sonucu, bir diğerine giriş bilgisi olarak aktarılmaktadır. Bu nedenle işlemler arasındaki iletişimin sistem kesmelerini (interrupt) kullanmaksızın, düzgün bir biçimde gerçekleştirilmesi gerekmektedir. İşlemler arası iletişimde, ortak kullanılan kaynaklar en önemli ve sorunlu noktalardır.

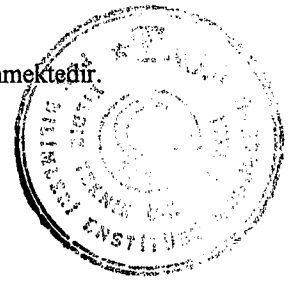
5.4.1 Yarış koşulları (Race Conditions)

Sistem kaynakları, çeşitli işlemler tarafından eşzamanlı olarak erişilip kullanılmaya çalışılır. Ancak burada kaynağa erişilmiş olması o kaynağın kullanılabileceği anlamına gelmemelidir. Şöyle ki:

Ortaklaşa kullanılan yazıcı kuyruğu, *giriş* ve *çıkış* olarak adlandıracağımız iki değişkenin kontrolünde kullanılıyor olsun; *giriş* kuyruğa dahil olacak bilginin yerleştirileceği yeri belirlerken, *çıkış* da basılacak bilginin nereden alınacağını belirlesin. Yazıcı kuyruğuna *t* anında *procA* isimli işlemin eriştiğini düşünelim. *procA* kuyruğa yerleştireceği bilginin konulması gereken yeri *giriş* değişkenine bakarak öğrenir, *procA*'nın öğrendiği değerin *n* olduğunu varsayalım. Ancak tam bu sırada işlemci, işlem sırasını *procA* dan alarak yazıcı kuyruğuna erişmek isteyen *procB* isimli işleme versin. *procB* de ihtiyacı gereği yazıcı kuyruğuna erişir, *giriş* değerine bakar ve *n* olduğunu görür, bilgileri *n* numaralı kuyruk alanına koyar ve daha sonra gelecek isteklerin kuyrukta yer alabilmesi için *giriş* değerini kuyruktaki boş alanı gösterecek biçimde, *n+1* olarak değiştirir. İşlemci, *procB*'nin sonlanmasını takiben, tekrar sıra gelmesini bekleyen *procA*'ya işlem sırasını verir. *procA* daha önceki turda elde ettiği ve *n* olarak bildiği *giriş* değerinin halen geçerli olduğu düşüncesiyle, *n* numaralı alana (daha önceki turda *procB* tarafından doldurulan alana) bilgilerini yerleştirir ve *giriş* değişkenini *n+1* ile değiştirerek sonlanır. Bu işlemler sonucunda, *procB* tarafından yazıcı kuyruğuna yerleştirilen bilgiler hiçbir zaman yazıcıdan alınamayacaktır.

Bu tür problemler literatürde yarış koşulları* (race conditions) olarak tanımlanır. Tamamen işlemcinin, hangi işlemi ne zaman devreye alacağına bağlı olarak oluşan bu problemi

* Yarış koşulları, işletim sistemlerini konu alan kitap ve makalelerde bütün ayrıntılarıyla işlenmektedir.



görmek oldukça zordur. Çoğunlukla (Murphy kuralları uyarınca) yapılan testlerin hiçbirinde fark edilmeyen bu problem, yazılımın gerçekte kullanılacağı yerde ilk çalışması sırasında ortaya çıkabilecektir. (Bach, 1986, 396; Tanenbaum, 1987, 51-53; Tanenbaum, 1992, 33-34)

5.4.2 Kritik bölgeler* (Critical Sections)

Yarış koşulları, ortak erişilen kaynaklara, ki bunlar genel bir bütün çerçevesinde bakıldığında kritik bölgeler (critical sections) olarak adlandırılıp, eşzamanlı erişimler sonucunda oluşmaktadır. Yarış koşullarının oluşmasını engellemek için şu altı koşul uygulanmalıdır.(Milenkovic, 1987, 133; Tanenbaum, 1992, 35)

1. İki işlem aynı anda kritik bölge içinde olamamalıdır.
2. Uygulamalar, kullanılan işletim sisteminden bağımsız çalışabilecek şekilde geliştirilmelidir.
3. İşlemci hızı, adedi üzerinden yapılabilecek çıkarımların hiçbiri doğru değildir.
4. İşlem öncelikleri üzerinden yapılabilecek çıkarımların hiçbiri doğru değildir.
5. Kritik bölgenin dışında çalışan işlemler diğerlerinin çalışmasını engellememelidir.
6. İşlemler kritik bölgeye girmek için sonsuza kadar beklememelidir** .

5.4.3 Yarış koşullarının engellenmesi için yapılması gerekenler

Anuyumluluğu sağlamak, yarış koşullarının engellenmesi açısından oldukça önemlidir. Özellikle paylaşılan verinin (shared data) tutarlılığının (consistency) sağlanması tamamen anuyumluluğa bağlıdır. Örneğin, *ProcA* isimli işlemin bir dosya üzerinde işlem yapmak üzere *thrA* adında bir işlem parçacığına sahip olduğunu düşünelim. *thrA* isimli işlem parçacığı dosya üzerindeki *n* numaralı kaydı değiştirirken *ProcB* isimli işleme ait *thrB* isimli işlem parçacığının *n* numaralı kayıt üzerinde işlem yapması hatalı sonuca neden olabilecektir. Eğer *thrA* kayıt üzerindeki değerleri tam olarak değiştirmeden *thrB* o kaydı okur ise tutarsız bir bilgi almış olacak ve *thrB*'nin bundan sonra yapacağı hesaplamalar

---

* Dijkstra "critical region" ifadesini kullanmıştır. (Dijkstra, 1965)

** Benzer şekilde hiçbir işlem de sonsuza kadar kritik bölge içinde bulunmamalıdır. Böyle bir durum söz konusu olursa, tek işlem kritik bölgeye girmek için beklemekte olan diğer işlemleri bloke edecek, o işlemlerin vermesi gereken servisleri engelleyecek, dolayısıyla da başarımlılığın düşmesine (performance degradation) neden olacaktır.

düzgün sonuç vermeyecektir. Diğer yandan *thrA* kayıt üzerindeki işlemlerini yaparken, *thrB* kayıt üzerindeki bilgileri değiştirmek isterse bu sefer de *thrA* tarafından yapılan işlemler hata verecektir. Bu ve buna benzer problemleri ortadan kaldırmak üzere kullanılacak bir engel mekanizması yardımıyla kritik bölgelere (ortak erişilmesi gereken kaynaklara) erişecek işlem parçacığının, bir diğerinin erişimini engellemesi, kendi hesaplamalarını yapması, işini bitirmesini takiben engeli kaldırması gereklidir. Erişimin engellendiği süre içerisinde ortak erişim alanını kullanmak isteyen diğer işlem parçacıkları bloke olmalı ve ortak alanı işgal eden işlem parçasının işini bitirip, engeli kaldırmasını beklemelidir. Engelin kalkmasını takiben, sırada beklemekte olan işlem parçacıklarından herhangi biri, benzer engel mekanizmasını etkinleştirmek şartıyla ortak erişim bölgesini kullanmalıdır. Bu amaçla kullanılan yöntemler, kullanım amaçlarına bağlı olarak bazı farklılıklar göstermekle birlikte, karşılıklı dışlama (mutual exclusion - mutex), koşul değişkenleri (conditional variables) ve semafor (semaphore) dur. Bu yöntemler de ilk anda daha dar kapsamda da olsa kritik bölge gibi görülebilirler. Oysa bu yöntemlerde, gerek donanım gerekse işletim sistemi tarafından desteklenen "Test And Set Lock - TSL"* benzeri bölünemez (atomic) bir işlemden yararlanılmaktadır. Bu sayede test işlemini takip eden ve testin sonucuna bağlı olarak yapılacak kilitleme işlemi, araya herhangi bir başka işlemin girmesine imkan bırakmaksızın yapılmakta ve kritik bölge oluşması engellenmektedir.

5.4.3.1 Karşılıklı dışlama (Mutual Exclusion - Mutex)

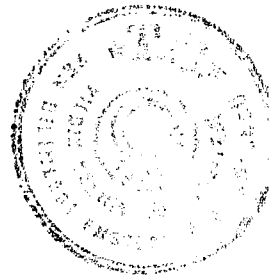
Anuyumluluğun sağlanması için kullanılan en etkin yöntemlerden biri önceki paragrafta da değinildiği gibi karşılıklı dışlama yöntemidir. Bu yöntem, ortaklaşa kullanılan kaynaklara erişimde "bir anda bir işlem" felsefesi ile o ana kadar süregelen eş zamanlı / paralel olan çalışmayı, seri bir yapıya dönüştürmektedir (Bach, 1986, 30; Milenkovic, 1987, 132-144,192-193; Sunsoft, 1994; Tanenbaum, 1997, 53-57,124). Ancak, karşılıklı dışlama ile ilgili olarak kullanılan çağrılar, işletim sistemine ve bu sisteme dahil kütüphanelere göre farklılıklar göstermektedir. Bu nedenle karşılıklı dışlama ile ilgili olarak sık kullanılan çağrıların neler olduklarından çok, ne amaç için kullanıldıkları önem kazanmaktadır. Kullanım amaçlarına göre *mutex* işlemleri şunlardır:

* TSL yapısı, herhangi bir işlemin, işlemciyi meşgul ederek beklemesine (busy-waiting) neden olduğu için sistem kaynaklarının etkin kullanımı açısından bir dezavantaj yaratmaktadır.

- ***Mutex yarat :*** *mutex*, kod içerisinde bir yerde yaratıldıktan sonra tüm işlem parçacıkları tarafından kullanılabilir. *mutex*, hem bağlı bulunulan işlem ile diğer işlemler arasında, hem de bağlı bulunulan işlem içindeki işlem parçacıkları arasındaki anuyumluluğu sağlamak için kullanılır.
- ***Mutex kilitle :*** Paylaşılan kaynağa erişen işlem parçacığı, başka işlem parçacığının erişmesini engellemek üzere yaratılan yapıyı, 'kilitle' çağrısı ile kilitler. Ortak kaynağa erişmek isteyen işlem parçacıkları bu kaynağa sahip olmak istediklerinde 'kilitle' çağrısını kullandıklarında bloke olarak ortak kaynağı kullanan işlem parçasının o alanı boşaltmasını beklerler.
- ***Bloke olmadan mutex kilitle:*** Eğer herhangi bir işlem parçacığı, *mutex*'i kilitlemiş ise, 'bloke olmadan kilitle' çağrısı hata vererek dönecektir. Aksi halde 'bloke olmadan kilitle' çağrısını kullanan işlem parçacığı, *mutex*'i kilitler ve paylaşılan kaynağa sahip olur. Bu çağrı, kendisini kullanan işlem parçacığının bloke olmadan, kaldığı yerden, devam edebilmesini sağlamaktadır.
- ***Mutex kilidini çöz :*** Bu çağrı sayesinde *mutex*'i kilitleyen işlem parçacığı, kullanmakta olduğu ortak kaynaktaki işini bitirerek diğerlerinin de bu alanı kullanmasına izin verir. 'Kilidi çöz' çağrısının kullanılmasını takiben, sırada bekleyen herhangi bir işlem parçacığı serbest kalır ve ortak alana sahip olur.
- ***Mutex'i ortadan kaldır:*** 'Yarat' çağrısı ile yaratılmış olan *mutex*, 'ortadan kaldır' çağrısı yardımıyla ortadan kaldırılır. Ancak *mutex*'in saklandığı alan boşaltılmaz.

5.4.3.2 Koşullu değişkenler (Conditional Variables)

Koşul değişkenleri, belirlenen koşul gerçekleşinceye kadar işlem parçacıklarının bloke olmasını sağlamak üzere kullanılırlar. Koşulların kontrol işlemi mutex'de olduğu gibi "bir anda bir iş parçacığı" felsefesine uygun olarak gerçekleştirilir. Eğer koşul değişkeninin kontrolü sırasında olumlu cevap alınamaz ise işlem parçacığı o koşul değişkeni üzerinde bloke olur. Koşul değişkeninin durumu herhangi bir işlem parçacığı tarafından değiştirilecek olursa, koşul üzerinde beklemekte olan diğer işlem parçacıklarını haberdar edebilmek üzere koşul değişkenine bir sinyal yollanır (Milenkovic, 1987, 177-178;



Tanenbaum, 1997, 60). Sıkça kullanılan koşul değişkeni çağrıları, kullanım amaçları ve şekilleri Sunsoft (1994) 'da yer almaktadır.

5.4.3.3 Semaforlar (Semaphores)*

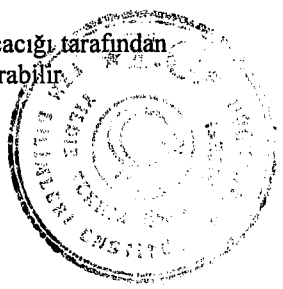
1965 yılında E.W. Dijkstra'nın (Dijkstra, 1965) ortaya attığı semafor kavramı, o dönemde demiryollarında tek hat üzerinden geçiş yapacak trenlerin hattı kullanırken yaptıkları kazaları engellemek üzere geliştirilmiş bir yöntemdi. Bu yöntem uyarınca tren, tek hatlı yola girmeden önce "semaphore" u kontrol edecek, "semaphore" girmeye izin veriyorsa, "semaphore" rengini değiştirip başka trenlerin hatta girmesini engelleyecek, hat üzerinden geçişini yapacak, geçişin tamamlanmasını takiben "semaphore" değerini başka trenlerin hattı kullanabilmesini sağlamak üzere değiştirecekti. Bilgisayara uyarlanan şekli ile semafor (Ben-Ari, 1997, 47-60) tam sayı olarak tanımlı bir değişken şeklinde karşımıza çıkmaktadır. Semaforu kullanmak için değerinin pozitif bir sayı olması gerekmektedir. Semaforu kullanmak isteyen işlem, semafor değişkeninin değerini bir azaltarak (P operation) diğer işlemlerin semaforun kullanımda olduğunu anlamasını sağlar. Semaforu kullanan işlem, işinin bitmesini takiben değeri bir arttırır (V operation)**. Bilgisayar alanında kullanılan semaforlar iki ayrı grup altında toplanabilir. Bunlardan biri, ikili semafor (binary semaphore) olup 0 ve 1 dışında değer alamaz. Temel olarak mutex ile aynı özellikleri göstermektedir. Diğeri ise sayıcı semafor (counting semaphore) olup isteğe bağlı (arbitrary) pozitif değerler alır***. Semafor kullanım amaçları ve özelikleri Bach (1986), Milenkovic (1987) , Sunsoft (1994), Tanenbaum (1997) ve diğer işletim sistemleri ile ilgili kaynaklarda incelenmiştir. Kullanım amaçlarına göre semafor işlemleri şunlardır;

---

* Makinistlerin kullandığı bir işaretleşme mekanizması olan "semaphore" kelimesi için henüz önerilmiş Türkçe bir karşılık bulunmamaktadır.

** P ve V olarak adlandırılan işlemleri sembolize eden harflerin yaygın kullanılan İngilizce kelimeler veya bunların baş harflerinden oluşturulmuş kısaltmalar ile hiçbir ilişkileri yoktur. Bu harflerin seçilmiş olmasının nedeni Dijkstra'nın Hollandalı olmasıdır. Buna göre P, *proberen te verlagen* den türetilmiş *prolagen,* dilimizdeki manasıyla, azaltmaya çalış, V, *verhogen,* dilimizdeki manasıyla, arttır demektir Ancak farklı kaynaklarda (Andrews ve Schneider, 1983) P'nin Hollanda dilinde *passern* (geçmek), V'nin, *vrygenen* (bırakmak) manasında kullanıldığı belirtilmektedir. P ve V işlemleri şu şekilde tariflenmiştir;
P işlemi, semafor değerini kontrol eder . Semafor değeri sıfırdan büyük ise bir azaltır, aksi halde bloke olur. V işlemi ise semafor değerini bir arttırıp, semafor üzerinde beklemekte olan herhangi bir işlem var ise devreye girmesine imkan sağlar.

*** Özel bir kural olmamakla birlikte, mutex üzerindeki kilit, sadece onu kilitleyen işlem parçacığı tarafından çözülmelidir. Semaforda ise isteyen, V işlemini kullanarak semafor değişkeninin değerini arttırabilir.



- ***Semafor yaratılması:*** Semafor, kod içerisinde bir yerde yaratıldıktan sonra tüm işlem parçacıkları tarafından kullanılabilir. Semafor, hem bağlı bulunulan işlem ile diğer işlemler arasında, hem de bağlı bulunulan işlem içindeki işlem parçacıkları arasındaki anuyumluluğu sağlamak için kullanılır.
- ***Semafor değerinin arttırılması :*** Semafor değerini arttırmak için kullanılır. V işlemi olarak da adlandırılan bu çağrı, semafor üzerinde beklemekte olan bir başka işlemin devreye girmesine imkan sağlar.
- ***Semafor değişkenini bekleme :*** Semafor değişkeninin sıfırdan büyük bir değer alması beklenir. Koşul sağlandığı zaman semafor değerini azaltır. Bu çağrı, P işlemi olarak da adlandırılmaktadır.
- ***Semafor değerinin azaltılması :*** 'Semafor değişkenini bekleme' çağrısının bloke olmadan çalışan şeklidir. Semafor değeri (eğer sıfırdan büyükse) azaltır. (P işlemi)
- ***Semafor'un ortadan kaldırılması :*** Belirlenen semaforun ortadan kaldırılmasını sağlar. Ancak semaforun saklandığı alan boşaltılmaz.

6. DAĞITIK SİSTEMLER ÜZERİNDE DOSYALARIN YERİNİ BULMAYI KOLAYLAŞTIRACAK BİR SİSTEM ÖNERİSİ: DOSYA İSİM SERVİSİ*

6.1 Önerilen Sistemin Teorisi

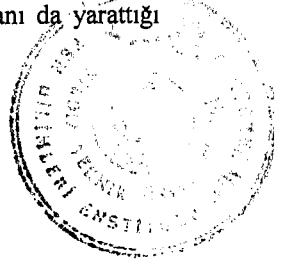
6.1.1 Neden "Dosya İsim Servisi" ?

Bilgisayarların tarihsel gelişimi içerisinde, mikroişlemcili bilgisayar sistemleri oldukça uygun fiyat/başarım oranlarını yakalamıştır. Bilgisayarlar arası iletişimin, özellikle de yerel alan ağlarının (LAN) gelişmesi, ortak bir amacı yerine getirebilmek için çalışan kişisel bilgisayarların veya iş istasyonlarının (workstation) bir bilgisayar ağı üzerinde toplanmasına imkan sağlamıştır. Bunun yanında çalışma gruplarının ihtiyaçlarını karşılamak üzere geliştirilen ağ (Novell, Banyan Vines, Microsoft Network vs..) ve uygulama yazılımları ile dağıtık altyapının desteklenmesi, mümkün olduğunca üreticiden bağımsız, ofis ortamları için yeter düzeyde işlem gücüne sahip, işletilmesi için özel fiziksel şartlar (yükseltilmiş taban, özel ses ve ısı yalıtımı, deneyimli personel, vs..) gerektirmeyen, düşük maliyetle rahat genişleyebilen dağıtık sistemlerin hızla yaygınlaşmasına imkan tanımıştır.

Bilgisayar ağlarında değişik kullanıcı ve kullanıcı gruplarının ihtiyaçlarını karşılamak üzere farklı işletim sistemleri (Unix ve türevleri, Windows ve türevleri, OS/2, vb..) mevcut uygulamaların gereği olarak kullanılır hale gelmiştir. Bunca farklı dosya sunucusu ve işletim sistemi ile desteklenen altyapılarda da dosyaya erişmek önemli bir problemi oluşturmaktadır. Kullanıcılar, ihtiyaç duydukları dosyaya, dağıtık yapı içindeki bilgisayarların işletim sistemlerinin ve dosyaya erişmek için kullanılan metotlardan

---

* "Dosya İsim Servisi", dağıtık sistemler üzerinde ortaya atılmış yeni bir kavram ve bu kavramı gerçekleştirmek üzere geliştirilmiş uygulamalar bütünüdür. Bu yapının ileride bir standard haline gelebilmesi ümidiyle, uluslararası literatürde kolay tanımlanabilmesi ve anlaşılabilmesi için servisin Türkçe isminin baş harfleri olan DİS yerine, servisin İngilizce isminin baş harflerinden oluşan FNS, kısaltma olarak kullanılmıştır. FNS'ye dahil olan modüllerin isimleri de benzer biçimde İngilizce karşılıklarının ilk harfleri ile belirlenmiştir. Ayrıca DOS (Disk Operating System)'in dilimizde DİS (Disk İşletim Sistemi) kısaltması ile kullanıldığı da dikkate alındığında, doğabilecek yanlışlıklara imkan tanımamak amacıyla DİS kullanılmamıştır. Her ne kadar FNS, dosya erişim sistemi olarak kullandığı NFS (Network File System) ile olan yakın ilişkisi ve kullanılan kısaltmaların aynı harflerin basit bir fark ile dizilişinden doğan isim benzerliği yüzünden bazı yazım ve telaffuz problemleri yaratsa da, bunun bir tanıtım imkanı da yarattığı düşüncesindeyim.



bağımsız olarak erişmek ve kullanmak isterler. Bu nedenlerle dağıtık yapılarda mümkün olduğunca kullanıcıya şeffaf* çalışabilecek "dosya bulucu" bir sisteme ihtiyaç vardır. Önerdiğimiz sistem, kullanıcı tarafından belirlenen dosya ismi, dizin ve bilgisayar adresi gibi özelliklerden hareket ile, dosyaların dağıtık sistem içindeki yerini bulacak ve mevcut standartlara uygun bir dosya erişim yöntemiyle kullanıma sunacak bir araştırma çalışmasıdır.

6.1.2 Dosya isim servisi üzerine kurulu olduğu servisler

Dosya isim servisi, istemci-sunucu (client-server) mantığında çalışmak üzere tasarlanmış modüllerden oluşan bir sistemdir. Bu sistemin çalışması için iletişim altyapısında TCP/IP'den, dosya erişim protokolü olarak da NFS'den yararlanılmıştır.

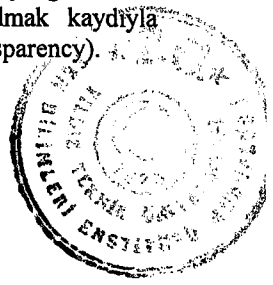
6.1.2.1 İletişim altyapısı (TCP/IP)

Dosya isim servisinin, istemci-sunucu modelinde çalışabilmesi için öncelikle bir iletişim katmanının bulunması gerekmektedir. Bu iletişim katmanı mevcut standartlara ne kadar uygun seçilirse ileride farklı sistemler ile yapılacak entegrasyonlarda haberleşmeden kaynaklanabilecek sorunlar da o derece aza indirilecektir. Bu nedenle çalışmamızda TCP/IP (Transmission Control Program / Internet Protocol) iletişim katmanı olarak kullanılmıştır (RFC 791; RFC 793; RFC 1180; Comer, 1991; Hunt, 1992; Comer ve Stevens, 1993). Taşıma katmanı arayüzü olarak Berkeley Socket'lerinden yararlanılmıştır (Besaw, 1987; Comer ve Stevens, 1993; Leffler vd.; Sechrest).

6.1.2.2 Dosya erişim protokolü (Network File System - NFS)

Mevcut altyapı içerisinde dosya erişim yöntemi olarak diğer sistemler ile uyumluluğu sağlayabilmek ve "Dosya İsim Servisi - FNS" ile ortaya atılan 'dosya bulucu' sistemin uygulanacak farklı bir dosya erişim yöntemi arkasında kayıp olmasını engellemek için Sun Microsystems Inc. tarafından ortaya atılan ve geliştirilen NFS (RFC 1094; RFC 1813) kullanılmıştır. NFS, çevrim içi (on-line), paylaşılabilir ve şeffaf bir dosya erişim sistemi olması, dosyayı bir noktadan diğerine tamamen aktarmak yerine bilgileri gerektikçe aktarması, TCP/IP iletişim katmanı ile uyum içinde çalışması nedeniyle seçilmiştir.

* Dağıtık işletim sisteminin önemli özelliklerinden biri olan şeffaflık (transparency) kavramı önerdiğimiz 'dosya bulucu' sistem kapsamında iki değişik şekliyle karşımıza çıkmaktadır. Bunlar; istenen kaynağın nerede olduğunun kullanıcı tarafından bilinmemesi (Location Transparency), isimleri aynı kalmak kaydıyla kaynakların bulundukları yerlerin kullanıcının bilgisi dışında değiştirilmesidir (Migration Transparency).



Dataquest'in yaptığı araştırmalardan, NFS'nin, çok farklı işletim sistemi ve mimari üzerinde kullanılabilme özelliği sayesinde, 1994 yılında 8.5 milyon bilgisayar sistemi üzerinde kullanılan, 1997 yılında ise 12 milyon bilgisayar sistemi üzerinde kullanılması beklenen, yaygın bir doysa erişim metodu olduğu da görülmektedir.

6.1.3 Dosya isim servisini oluşturan kavramlar ve modüller

İstemci-sunucu (client-server) mantığında çalışmak üzere tasarlanmış modüllerden oluşan Dosya İsim Servisi, Dosya İsim Servisi Etki Alanı (FNSD) kapsamında hizmet verir. Dosya İsim Servis Etki Alanı kavramının tanımı ve Dosya İsim Servisini oluşturan modüllerin görevleri şu şekildedir:

6.1.3.1 Dosya isim servisi etki alanı (File Name Service Domain - FNSD)

Dağıtık sistemlerde çalışan sunucu servislerin, servis vermekle sorumlu olduğu bir etki alanı (domain) vardır. Bu etki alanı her ne kadar iletişim katmanı ile sınırlı olsa da istemci-sunucu türündeki bu sistemin, haberleşme hızı göz önünde bulundurularak etki alanının (deneysel olarak) yerel alan ağ (LAN) ortamı ile sınırlı olması düşünülmüştür. Gerekli istemci ve sunucu modüllerinin kullanılması ile sistem, teorik olarak geniş alan ağlarına (WAN) da hizmet verebilecektir. Ancak geniş alan ağlarındaki uygulamalarda oluşabilecek iletişim problemleri (hat hızları, çeşitli güvenlik problemleri vb.) göz ardı edilmemelidir. FNSD'ye dahil olabilmek için dosya isim sunucusu (FNS) ile etkileşimli çalışacak dosya isim sunucu ajanına (FNSA) sahip olunması gerekmektedir. Etki alanında bulunan bilgisayarların adreslenmesi için, sistemin üzerine kurulduğu TCP/IP ağ altyapısının bir gereği olarak Internet adresleri kullanılmıştır. Sisteme dahil bilgisayarlara Internet numaraları yerine, kolay hatırlanabilir isimlerle erişilebilmesi için, altyapı içinde DNS (Domain Name System) sisteminin düzgün çalışması son derece önemlidir (RFC 882; RFC 883; Comer, 1991, 311-333).

6.1.3.2 Dosya isim sunucusu (File Name Server - FNS)

Dosya isim sunucusunun temel görevleri:

1. Bulunduğu etki alanı-FNSD içerisindeki çeşitli dosya sunucu veya işletim sistemleri üzerinde çalışmakta olan dosya isim sunucu ajanı-FNSA ile mevcut TCP/IP katmanı üzerinden, soket arayüzü yardımıyla haberleşerek, o sistemlerde bulunan dizin

(directory) ve dosya (file) bilgilerinin bir ana (master) indeksini, dinamik olarak genişleyen, linkli liste özelliğindeki tablolar kullanarak oluşturmak,

2. Kendisinden, dosya isim servis istemcisi-FNSC yardımıyla dosya bilgisi sorgulayan istemcilere, istenilen dosya ile ilgili, sahip olduğu bilgileri göndermek,
3. İkincil dosya isim servisi-SFNS'den düzgün aralıklarla gelen yenileme (refresh) isteklerine cevap vererek, SFNS'nin indeks bilgisinin mümkün olduğunca doğru ve güncel olmasını sağlamak,
4. Belli zaman aralıklarında, tablolar yardımıyla indeksini tuttuğu bilgilerin ne derece güncel olduğunu belirleyen değişkenleri azaltarak, öncelikle belirlenen süre zarfında FNSA tarafından güncellenmeyen dizin bilgilerini, daha sonra bilgisayar (host) bilgilerini mevcut linkli liste yapısından çıkartmak.

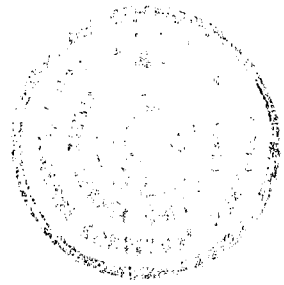
6.1.3.3 İkincil dosya isim sunucusu (Secondary File Name Server - SFNS)

Birincil dosya isim sunucusunun-FNS çalışmadığı durumlarda onun yerini alabilecek, her türlü indeks bilgisine sahip olan bir sistemdir. Bu FNS ile eşzamanlı olarak çalışacak ve belirli aralıklarla birincil FNS'e yollayacağı tazeleme istekleriyle kendisindeki mevcut indeks bilgilerini, birincil FNS'den gelen güncelleme bilgileriyle yenileyecektir. İkinci bir dosya isim sunucusu kullanmanın temel nedenleri:

1. Dosya isim servisi istemcileri-FNSC tarafından birincil dosya isim sunucusu-FNS üzerinde yaratılacak işlem yükünü mümkün olduğunca düşürmek,
2. Bilgisayar ağı üzerinde tek noktada oluşacak (birincil FNS) trafik yoğunluğunu mümkün olduğunca başka sunuculara yönlendirmek,
3. Özellikle dağıtık sistemlerde sık karşılaşılan tek nokta hatası (single point of failure) problemini kısmen de olsa ortadan kaldırarak, birincil FNS'nin devre dışı kaldığı durumlarda, kendi indeks bilgileri geçerli olduğu sürece, FNSC'lerden gelecek isteklere cevap verilmesini sağlamaktır.

6.1.3.4 Dosya isim sunucu ajanı (File Name Server Agent - FNSA)

Bu modül, FNSD'ye dahil her türlü dosya sunucusu ve/veya işletim sistemi üzerinde çalışması gereken temel parçadır. Bu parça kendi dosya sistemi üzerinde (file system) olan değişiklikleri bağlı bulunduğu FNS'e belli aralıklarla bildirmekle yükümlüdür. Bu sayede

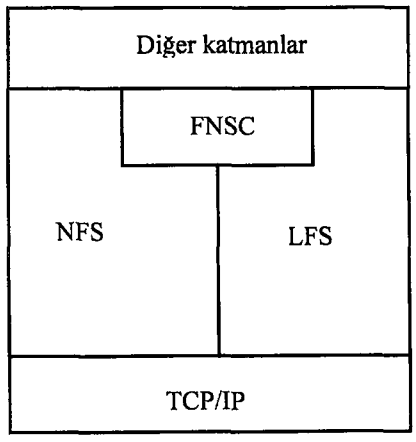


FNS, kendi etki alanı içinde bulunan dosya sunucularından ve/veya işletim sistemlerinden bilgi toplayarak kendi indeksini oluşturabilmektedir.

6.1.3.5 Dosya isim servis istemcisi (File Name Service Client - FNSC)

Kullanıcıların, programlarında FNS'den yararlanmalarını sağlayacak modülün ismidir. Bu modül mümkün olduğunca kullanıcıya şeffaf olacak ve kullanıcının verdiği bilgiler doğrultusunda (dosya, dizin ve bilgisayar (node) isimleri), aradığı dosyanın 'Dosya İsim Servisi' dahilinde nerede olduğunu bulmasını sağlayacaktır. Bu işlemi gerçekleştirirken, kullanıcının bilgisayarında bulunan yerel dosya sistemi (local file system-LFS) ile bağlantılı çalışacak (Şekil 6.1) ve kullanıcıya, ister yerel diskinde, isterse FNSD'ye dahil diğer sunucular üzerinde olsun, aradığı dosyayı, bilgisayar (host), dizin, büyüklük (byte cinsinden) ve tarih bilgileri ile getirecektir. Aynı isimde birden fazla dosyanın, gerek farklı dizinler, gerekse farklı bilgisayarlar üzerinde olabileceği düşünüldüğünde kullanıcının bunlar arasından birini seçmesi gerekmektedir. Her ne kadar bu durum, "kullanıcıya şeffaf görünmek" kavramıyla bir çelişki oluşturmaktaysa da, teknik olarak bu problemin çözümü olanaklı görünmemektedir.

FNSC, kullanıcının erişmek istediği dosyayı, yerel dosya sisteminin yanı sıra FNS'e yollayacağı sorgulama ile bulmaya çalışacak, uzak dosya sistemleri üzerindeki bir dosyaya erişmek istenmesi durumunda da NFS'den yararlanacaktır. NFS'nin gereği olarak, erişilmek istenen dosyanın bulunduğu dosya sisteminin, kullanıcının çalışmakta olduğu bilgisayar sistemi ile bağlantılı olması (mount) gereklidir.



Şekil 6.1 Dosya İsim Servis İstemcisi-FNSC ve diğer katmanların ilişkisi

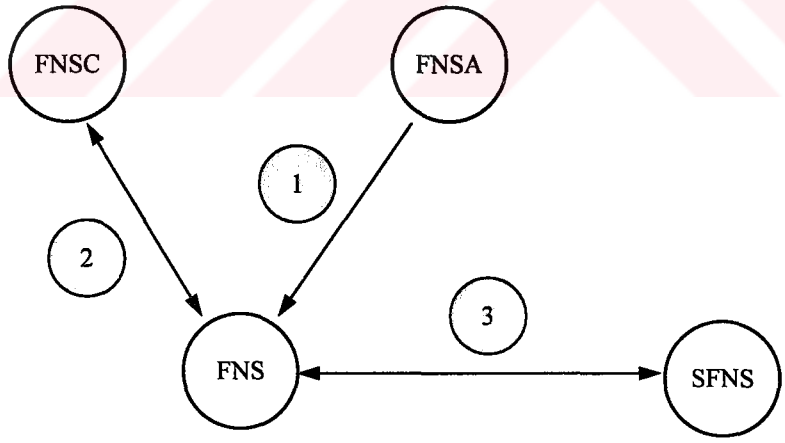
6.1.3.6 Dosya isim servisi modülleri arasındaki bilgi akışı

Şekil 6.2 de verilen durum diyagramı uyarınca FNS modülleri arasındaki bilgi akışı şu şekilde tariflenmiştir:

1) Dosya İsim Servis Ajanı (FNSA) dan Dosya İsim Sunucusuna (FNS) doğru (tek yönlü) bir bilgi akışı tariflidir. FNSA belli aralıklarla kendisine kayıt ettirilen dizin/dosya bilgilerini FNS'ye yollamakla görevlidir.

2) Dosya İsim Servis İstemcisi (FNSC) ile Dosya İsim Sunucusu (FNS) arasında iki yönlü bir bilgi akışı tariflidir. FNSC, kullanıcının erişmek istediği dosyanın isteniş şekline bağlı olarak FNS'ye bir sorgulama yollamalı ve FNS'nin kendi tabloları üzerinde yaptığı aramanın sonucu FNSC'ye yollanmalıdır.

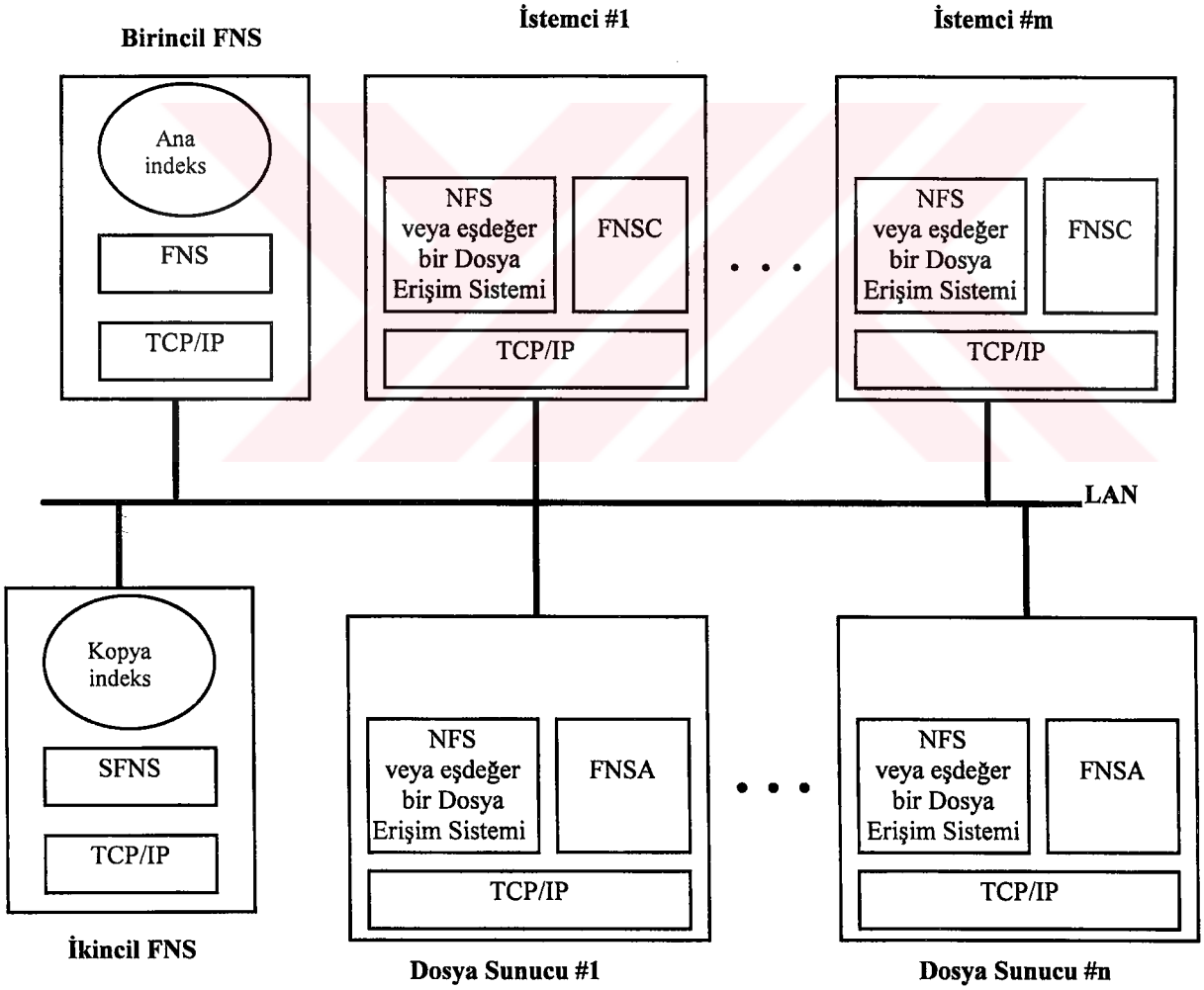
3) Dosya İsim Sunucusu (FNS) ile İkincil Dosya İsim Sunucusu (SFNS) arasında iki yönlü bir bilgi akışı tariflidir. SFNS, belli aralıklar ile FNS'ye güncel dizin dosya bilgilerini yollaması için istekte bulunmalı ve FNS'den gelen bilgileri alarak kendi tablolarını güncelleştirmelidir.



Şekil 6.2 Dosya İsim Servisi modülleri arasındaki ilişkiyi gösteren durum diyagramı

6.2 Önerilen Sistemin Uygulaması

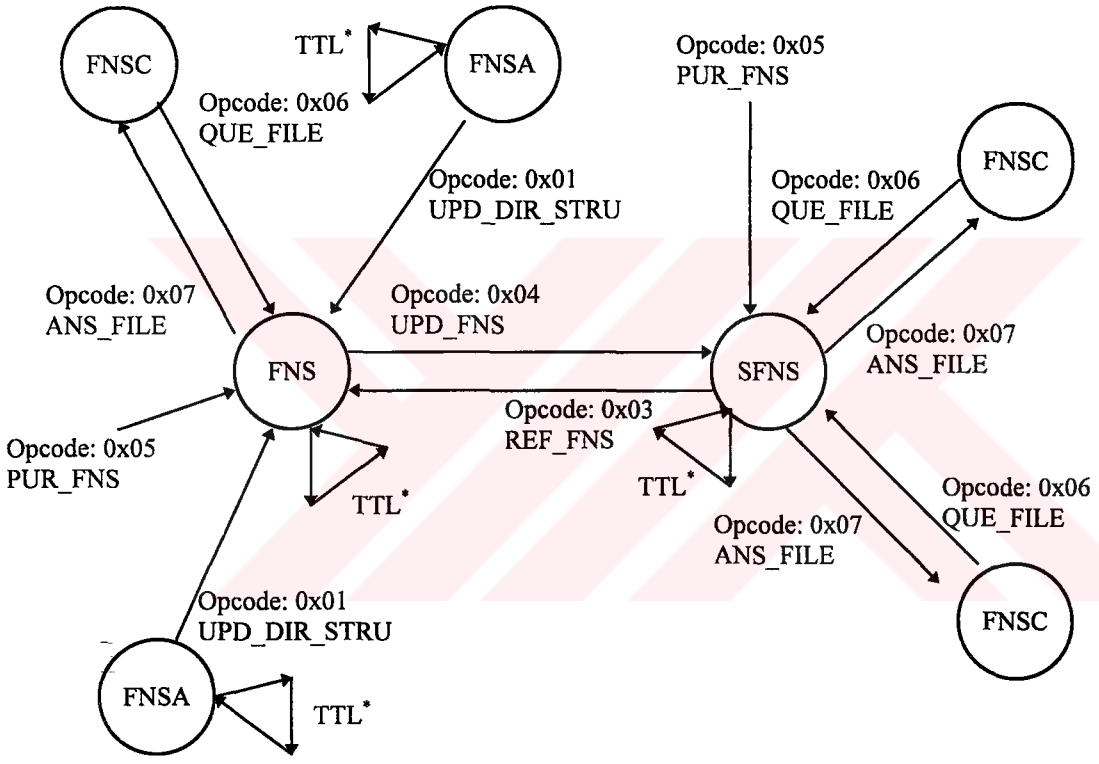
Dosya İsim Servisi, TCP/IP tabanlı bilgisayar ağları üzerinde kullanılmak üzere tasarlanmıştır. Ağ üzerinde seçilecek herhangi bir bilgisayar sistemi sunucu modülünü (FNS) çalıştıracak, dosya sunucu görevini yerine getiren bilgisayarlar ise ajan modülünü (FNSA) çalıştırarak sunucuya kendilerine kayıt ettirilen dizin/dosya bilgilerini ileteceklerdir. Diğer bilgisayarlar ise kullanıcıların dosyalara erişmesini sağlayacak istemci modüllerini (FNSC) çalıştıracaklardır. Dosya erişiminde kullanılan NFS, FNSA modülünüm çalıştığı bilgisayarda sunucu, FNSC'nin çalıştığı bilgisayarda ise istemci modülü ile yer alacaktır (Şekil 6.3). Mevcut bilgisayar sistemlerin kullanım amaçlarına göre FNS modülleri ayrı ayrı bilgisayarlarda çalıştırılabilecekleri gibi, gerekli olması durumunda tüm modüller aynı bilgisayar sistemi üzerinde de çalıştırılabilir.



Şekil 6.3 FNS'iyi kullanan yerel alan ağı

6.2.1 FNS'ye dahil modüller arası ilişkiler

Dosya isim servisine dahil modüller, aralarında soket arayüzü yardımıyla yollanan paketleri kullanarak bilgi alış verişinde bulunurlar. Bu haberleşme sırasında, isteklerini bir diğer modüle iletirken çeşitli işlem kodlarını (op-code) kullanırlar. Bu kodlar bilgi aktarımında kullanılan paket(ler)in kimden, ne amaçla geldiğini, yani isteği karşılamak için yapılması gereken işlemi belirlerken, kullanılan bir diğer alt kod paket içindeki bilginin türünü belirlemektedir. Modüller arası ilişki ve işlem kodları Şekil 6.4'de gösterilmiştir .



Şekil 6.4 Dosya İsim Servisi modülleri arasındaki ilişki ve kullanılan işlem kodları*

* TTL (Time To Live) değeri gerek FNS gerekse FNSA modülünde farklı amaçlar için kullanılan bir zamanlama mekanizmasını ifade etmektedir. FNS modülünde kullanılan TTL değerleri için Bölüm 6.2.3.3'e, FNSA modülünde kullanılan TTL değeri için Bölüm 6.2.4.2'e bakınız.

6.2.1.1 FNS ile SFNS arasında kullanılan işlem kodları

Bu işlem kodlarının isimleri ve işlevleri şu şekilde tanımlanmıştır:

REF_FNS (0x03) SFNS, koşullama dosyasında (*fnsconf.dat*) belirlenen süre sonunda, birincil FNS ile bağlantı kurarak, güncel indeks bilgilerini almak ister. Bu bilgileri alabilmek için FNS'ye *REF_FNS* işlem kodunu yollar.

UPD_FNS (0x04) FNS, *REF_FNS* işlem kodunu aldığı zaman kendisine bu kodu gönderen ikincil FNS'ye *UPD_FNS* işlem kodunu takiben elindeki indeks bilgilerini yollar.

6.2.1.2 FNS ile FNSA arasında kullanılan işlem kodu

FNS ile FNSA arasında tek yönlü bir haberleşme vardır. Bu haberleşmede kullanılan işlemin kodu ve işlevi şu şekilde tanımlıdır:

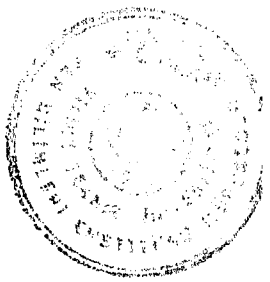
UPD_DIR_STRU (0x01) Bu kodu takiben FNSA tarafından (*fnsaconf.dat* dosyasında belirlenen) düzenli aralıklar ile oluşturulan dizin/dosya bilgisi FNS'ye yollanmaktadır. *UPD_DIR_STRU* işlem kodunu alan FNS, kendi tablolarındaki bilgileri FNSA'dan gelen yeni bilgiler ile güncelleştirir.

6.2.1.3 FNS/SFNS ile FNSC arasında kullanılan işlem kodları

Kullanıcıların, FNS'den yararlanmasını sağlamak için geliştirilen FNSC modülünün kullandığı işlem kodları ve işlevleri şu şekilde tanımlıdır:

QUE_FILE (0x06) FNSC, FNS'ye erişmek istediği dosyaya ait bazı bilgileri (bilgisayar (host) adı ve/veya dizin adı ve/veya dosya adı) *QUE_FILE* işlem kodu ile gönderir.

ANS_FILE (0x07) FNS, kendisine gelen *QUE_FILE* işlem koduna cevap olarak, tabloları üzerinde yaptığı aramanın sonucunu *ANS_FILE* işlem kodu ile FNSC'ye yollar.



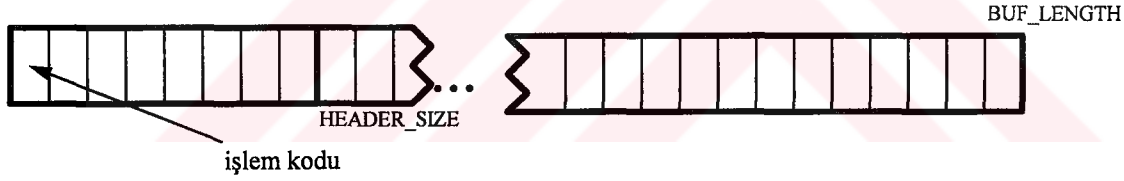
6.2.1.4 Diğer işlem kodları

EOT (0x00) — FNS'nin diğer modüller ile haberleşmesi sırasında ne kadar veri aktarılacağı ve bu aktarımın ne kadar süreceği belli değildir. Bu nedenle, aktarımın sonunu belirlemek üzere *EOT - 0x00* işlem kodu kullanılmıştır. Bu komutu alan modül, aktarımın sona erdiğini anlar. *EOT* işlem kodu, aktarımı başlatan modül tarafından kullanılır.

PUR_FNS (0x05) — FNS'lerin mevcut tablolarını silerek yeniden çalışmaya başlamalarını sağlayan işlem kodudur. Bu komut sistem sorumlusunun müdahalesi için düşünülmüştür.

6.2.2 İletişimde kullanılan paketin yapısı

Bilgi alış-verisi sırasında, iki uç arasında gidip gelen paketler genel bir formata göre düzenlenmiştir. Buna göre paketler, *BUF_LENGTH*[*] büyüklüğünde olacaktır. Paketin başından *HEADER_SIZE*[**] kadar kısmı, yollanacak komut bilgilerinin tutulması için ayrılmıştır (Şekil 6.5).



Şekil 6.5 İletişimde kullanılan paket yapısı (genel hal)

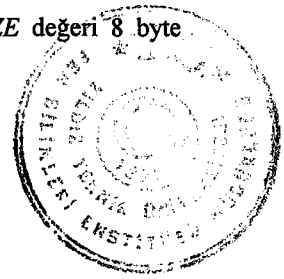
---

[*] *BUF_LENGTH* büyüklüğü kullanıcı tarafından ayarlanabilir bir değer olup, *glbdef.h* isimli tanım dosyasında tariflenmektedir. Ancak mevcut sabit tanımlar göz önüne alındığında, teorik olarak alabileceği en küçük değer;

BUF_LENGTH = [*HEADER_SIZE*]+[sizeof(long int)+1]+[*_D_NAME_MAX*+1]+
[*_D_NAME_MAX*+1+*_OWNER_NAME_MAX*+1+2*sizeof(long int)+1] = 329 byte

olacaktır. Kullanılacak paketin büyüklüğü, yollanacak bilgi miktarına göre ayarlanabilmektedir. Bu değerin küçük seçilmesi, paket sayısını artırıp hat üzerinde trafik yaratmanın dışında, işlemlerin uzamasına neden olurken, büyük seçilmesi (paket tam olarak bilgi ile doldurulamayacağı için) kaynak israfına neden olacaktır. Bu nedenle, *BUF_LENGTH* değerinin, FNS'nin tüm modülleri ile birlikte uygulanacağı altyapıya göre seçilmesi gereklidir.

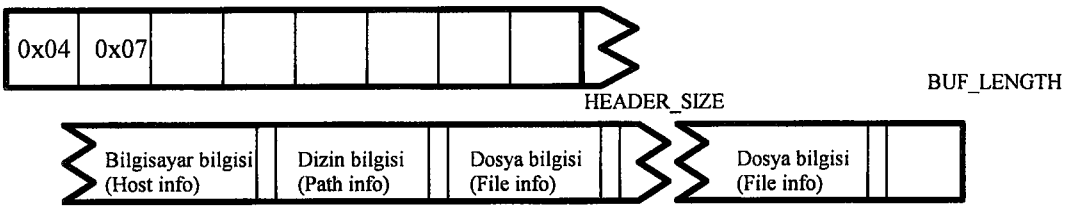
[**] *HEADER_SIZE* değeri kullanıcı tarafından belirlenebilir. Bu çalışmada *HEADER_SIZE* değeri 8 byte olarak seçilmiş ancak 2 byte'lık bölümü kullanılmıştır.



6.2.2.1 FNS ile SFNS iletişiminde paket yapısı

FNS, SFNS'ye *UPD_FNS (0x04)* işlem kodu ile indeks bilgisini gönderirken paket içinde sırası ile:

1. Bilgisayar (host), dizin ve dosyalara ait bilgiler olabilir: Bu durumda önbilgi (header) alanında *UPD_FNS* işlem kodunun yanı sıra paket içeriğini belirleyen 0x07 kodu kullanılır (Şekil 6.6);

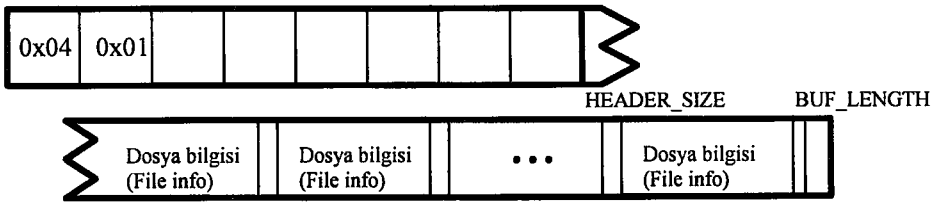


Şekil 6.6 FNS'den SFNS'ye yollanan paketin yapısı, 0x07 tipi

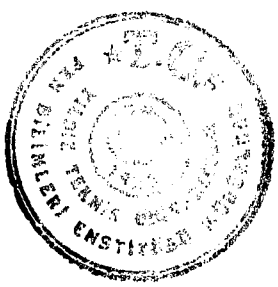
2. Dizin ve dosyalara ait bilgiler olabilir: Bu durumda, paket içeriğini belirleyen 0x03 kodu kullanılır (Şekil 6.7);

Şekil 6.7 FNS'den SFNS'ye yollanan paketin yapısı, 0x03 tipi

3. Sadece dosyalara ait bilgiler olabilir: Bu durumda, paket içeriğini belirleyen 0x01 kodu kullanılır (Şekil 6.8).



Şekil 6.8 FNS'den SFNS'ye yollanan paketin yapısı, 0x01 tipi



6.2.2.2 FNS ile FNSA iletişiminde paket yapısı

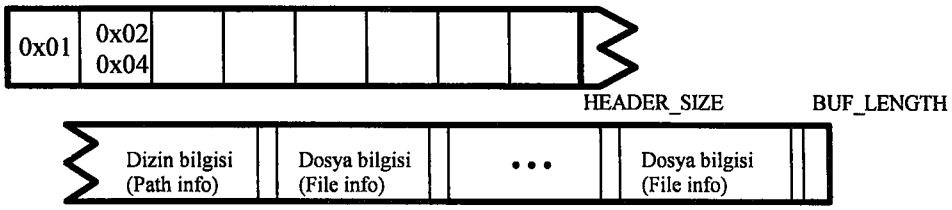
FNSA, belirlenen aralıklarla, kendi dizin ve dosya yapısındaki değişiklikleri, FNS'ye yollamaktadır. Yollanan bilginin miktarı, yollanacak paketin türünü de belirlemektedir. Bu nedenle, FNS ile SFNS arasındaki paket yapısına benzer bir yapı FNSA ile FNS arasında da kullanılmaktadır. Buna göre:

1. Yollanacak paket dizisinin ilk paketi, 0x01kodu,
2. Yollanacak paket dizisinin ara paketi, 0x02 kodu,
3. Yollanacak paket dizisinin son paketi, 0x04 kodu ,
4. Özel bir durum olarak ilk ve son paket, 0x05 kodu ile belirlenmektedir.

Yollanan ilk paket (0x01) veya ilk ve son paket (0x05) içinde, dizin ve sırasıyla dosyalara ait bilgiler yer almaktadır (Şekil 6.9).

Şekil 6.9 FNSA'dan FNS'ye yollanan paketin yapısı, 0x01/0x05 tipi

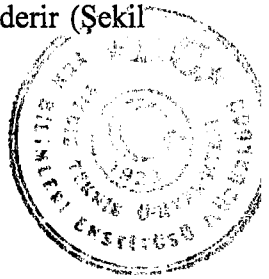
Ancak, ara (0x02) ve son (0x04) paketler sadece dosya bilgilerini içermektedir (Şekil 6.10).



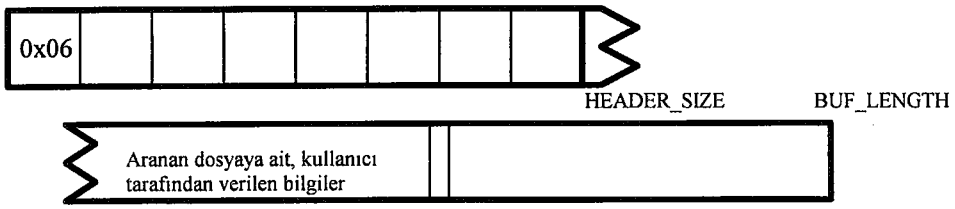
Şekil 6.10 FNSA'dan FNS'ye yollana paketin yapısı, 0x02/0x04 tipi

6.2.2.3 FNS ile FNSC iletişiminde paket yapısı

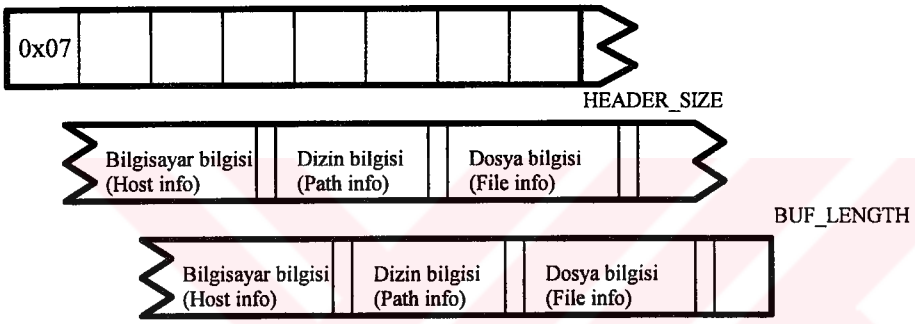
FNSC, FNS'ye yolladığı pakette, ilgilendiği dosyaya ait bilgileri (bilgisayar (host) ve/veya dizin ve/veya dosya), *QUE_FILE - 0x06* işlem koduna sahip bir paket ile gönderir (Şekil



6.11). Diğer taraftan FNS, kendi tabloları üzerinde, verilen koşullara uyan bilgileri tarayarak oluşturduğu listeyi, *ANS_FILE - 0x07* işlem koduna sahip paket(ler) içerisinde FNSC'ye iletir (Şekil 6.12).



Şekil 6.11 FNSC'den FNS'ye yollanan paketin yapısı

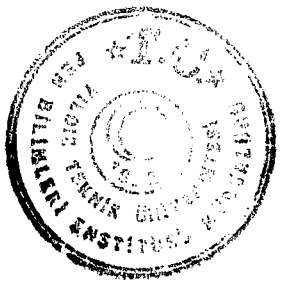


Şekil 6.12 FNS'den FNSC'ye yollanan paketin yapısı

6.2.3 Dosya isim sunucusunun (FNS) çalışma ilkesi

FNS için üretilen kod aynı zamanda ikincil FNS (SFNS) tarafından da kullanılmaktadır. Kodun, birincil ya da ikincil sunucu olarak çalışıp çalışmayacağı, *fnsconf.dat* isimli koşullama dosyası yardımıyla belirlenmektedir. FNS, bağlantılı* , eş zamanlı, çok işlem parçacıklı çalışmak üzere tasarlanmıştır. FNS, indeksini tuttuğu bilgilerin ne derece geçerli olduğunu belirleyen *TTL* değerlerini, *fnsconf.dat* isimli koşullama dosyasında belirlenen aralıklarla azaltabilmek için bir zamanlama mekanizmasıyla donatılmıştır. Zamanlama mekanizması yardımıyla tetiklenen bir yordam, dinamik olarak yaratılıp birbirleriyle ilintilendirilen (linkli liste şeklinde) bilgisayar (*host*), dizin tablolarını tarayarak, öncelikle süresi geçen dosya, daha sonra dizin bilgilerinin tutulduğu tabloları linkli liste yapısından ayıracaktır. FNS, bağlantı isteklerine, *glbdef.h* isimli tanım dosyasında tariflenen,

* bakınız Bölüm 3. Ağ Tabanlı Servisler ve Servis Tipleri

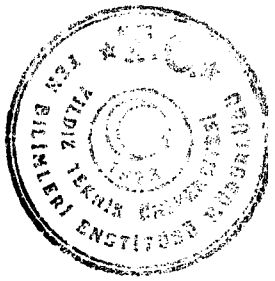


FNS_PORT ile belirlenen pasif soket üzerinden servis verir. FNS'ye gelen istekler, gelen paketin işlem koduna bakılarak değerlendirilmekte ve isteği karşılamak üzere yazılmış yordamlardan uygun olanı, yaratılan işlem parçacığı tarafından çalıştırılmaktadır. Bu sırada FNS, gelebilecek diğer istekleri karşılamak amacıyla, yarattığı soketi dinlediği sonsuz bir çevrim içinde dönmektedir. Diğer yandan yaratılan işlem parçacıkları, eş zamanlı çalışmalarının doğal bir sonucu olarak, zaman zaman aynı tablolara erişebilmektedirler. Bu kritik bölge problemi* karşılıklı dışlama (mutual exclusion - mutex) kullanılarak, indeks tablolarına kontrollü erişimle çözülmüştür. Kritik bölgede aynı zamanda bulunulmasına neden olabilecek, çalıştırdıkları alt yordamlar nedeniyle birbirlerinin yaptıklarını etkileyebilecek işlem kodları şunlardır;

1. *UPD_DIR_STRU - 0x01* işlem kodu ile FNSA'nın gönderdiği güncelleme bilgilerini tablolara yerleştirmek için kullanılan *fill_ptab()* alt yordamı,
2. İkincil FNS tarafından, *REF_FNS - 0x03* işlem kodu ile gelen güncelleme isteğine, *UPD_FNS - 0x04* işlem kodu ile indeks bilgileri (bölüm 6.2.2.1.'de tariflenen paket yapısına sadık kalarak) yollanırken kullanılan *send_tab()* alt yordamı,
3. İkincil FNS'nin, FNS den gelen bilgileri alıp kendi tablolarını güncelleştirmek için kullandığı *rcv_tab()* alt yordamı,
4. FNS'nin, *QUE_FILE - 0x06* işlem kodu ile gelen sorgulamaların sonucunu *ANS_FILE - 0x07* işlem kodu ile yollarken kullandığı *send_ans()* alt yordamı,
5. Tablolardaki bilgilerin geçerlilik sürelerini belirli aralıklarla azaltmak için kullanılan *dec_TTL()* alt yordamı,
6. İkincil FNS'nin belirli aralıklarla, FNS'ye, *REF_FNS - 0x03* işlem kodunu yollaması için kullanılan *dec_REF_TTL()* alt yordamı,
7. Belirlenen sürenin dolması ile geçerliliklerini yitiren dizin ve dosya tablolarını, bağlı bulundukları linkli liste yapısından çıkarmak için kullanılan *clean_all()* alt yordamı.

FNS, kendisine *UPD_DIR_STRU - 0x01* işlem kodu ile gelen paketlerin kimden geldiğini, soket arayüzünden öğrenir. Öğrenilen değer, mevcut bilgisayar (host) tablolarındaki bilgiler ile karşılaştırılarak benzer bir değer olup olmadığı kontrol edilir. Eğer aynı bilgisayardan daha önce de bilgi alınmışsa linkli liste içinde o bilgisayar ile ilintili bulunan tüm dizin ve

* bakınız Bölüm 5. İstemci/Sunucu Modelinde Eş Zamanlı İşlem



dosya tablolarının değişmiş olacağı düşünülerek, geçerlilik süreleri dolmamış olsa bile, mevcut dizin ve dosya tabloları linkli liste yapısından çıkartılır. Yeni gelen paketlerde bulunan bilgilerle yeni dizin ve dosya yapıları oluşturulur ve bunlar bilgisayar (host) tablosundaki kayıt ile ilintilendirilir.

6.2.3.1 FNS tarafından yaratılan ve kullanılan linkli liste yapısı

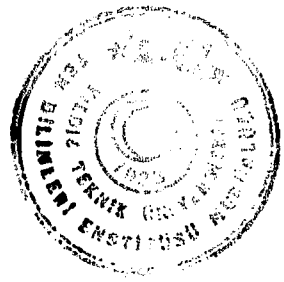
FNS, FNSA dan düzenli aralıklarla gelen bilgileri bilgisayar (host), dizin ve dosyalar için dinamik olarak yaratılan ayrı ayrı, ancak birbirleri ile ilintili tablolar üzerinde tutmaktadır. Bu tabloların yapıları *fns.h* isimli dosyada tariflenmiştir. FNS çalışmaya başladığı zaman ilk bilgisayar (host) adres tablosunu kendisi yaratır. Bu tablonun ve yaratılacak diğer tabloların büyüklükleri *glbdef.h* isimli dosyada *ALLOC_SIZE* ile belirlenmiştir. *ALLOC_SIZE* değeri, kullanıcı tarafından değiştirilebilir. Bu değerin çok küçük seçilmesi, mevcut bilgileri tutmak için daha çok tablonun yaratılmasına ve tablolar arasındaki geçişler için daha fazla zaman harcanmasına neden olacaktır. Diğer yandan, *ALLOC_SIZE* değeri çok büyük seçilirse, tablolar tam olarak dolmayacağı için, dinamik olarak ayrılan hafıza boş kalmış, kaynaklar israf edilmiş olacaktır. Yaratılan tablonun ne kadarının dolu olduğu *count* olarak adlandırılan tablo gözünde saklıdır. Tablolardan herhangi birine ekleme yapmak isteyen yordam, öncelikle *count* değerinin *ALLOC_SIZE*'dan küçük olup olmadığını kontrol etmelidir. Eğer değer küçük ise mevcut tablo kullanılabilir. Aksi halde dinamik olarak yeni bir tablo yaratılarak bir önceki ile *next_ght/prev_ght (next-gpt/prev_gpt, next_gft/prev_gft*) alanları kullanılarak ilintilendirilmelidir. Dosya ve dizin tablolarını bir üst seviye tablo (sırasıyla, dizin ve bilgisayar (host) tablosu) ile ilintilendirmek için *back_ptr* ve *back_off* isimli tablo gözleri kullanılmaktadır. Tablolar arası ilişkiler EK-1 de sunulmuştur.

6.2.3.2 FNS'de iş akışına ait blok diyagramları

FNS'de iş akışını belirleyen blok diyagramı ve kullanılan yordamlara ilişkin blok akış diyagramları EK -2 de sunulmuştur.

6.2.3.3 FNS koşullama dosyası (fnsconf.dat)

FNS, çalışmaya *fnsconf.dat* isimli koşullama dosyası içinde bulunan, kullanıcı tarafından belirlenebilen bazı değerleri alarak başlar. Bu değerler ve açıklamaları şu şekildedir;



primary FNS sunucusunun birincil olduğunu belirlemek üzere kullanılır. Bu tanımı takiben FNS'nin hangi alanda etkili olacağı belirtilir.

secondary FNS sunucusunun ikincil sunucu olduğunu belirler. Bu ifadeyi takiben birincil FNS'nin IP ismi/adresi'nin verilmesi gereklidir.

htabttl Bilgisayar (host) tablosunda bulunan bilgilerin geçerlilik süresini belirlemek üzere kullanılan parametredir. *chkttl* ile belirlenen sürenin dolmasını takiben bir azaltılmaktadır.

ptabttl Dizin tablosunda bulunan bilgilerin geçerlilik süresini belirlemek üzere kullanılan parametredir. *chkttl* ile belirlenen sürenin dolmasını takiben bir azaltılmaktadır.

chkttl Bilgisayar (host) ve dizin tablolarının kaç saniyede bir kontrol edileceğini belirleyen parametredir. Buna göre bir dizin tablosu bilgisi *ptabttl*chkttl* saniye sonunda güncellenmemiş ise ona bağlı dosya tabloları silinecektir. Benzer durum *htabttl*chkttl* saniye sonunda güncellenmeyen bilgisayar (host) tabloları için gerçekleşirse, onunla ilgili bulunan dizin ve dosya tabloları silinecektir.

refttl İkincil FNS için geçerli bir değerdir. Bu değer ikincil FNS'nin birincil FNS'den kaç saniye sonra güncelleme bilgisi isteyeceğini belirlemek üzere kullanılan parametredir. Bu değerin, bilgisayar (host) ve dizin silme sürelerinden daha küçük olmasında fayda vardır.*

Koşullama dosyasında uygun değerlerin verilip verilmediği, FNS içindeki basit bir kontrol yordamı tarafından incelenmekte, uygun olmayan koşullama dosyası FNS tarafından kabul edilmemekte, olası hata noktaları kullanıcıya bildirilerek düzeltilmesi istenmektedir.

6.2.4 Dosya isim servis ajanının (FNSA) çalışma ilkesi

FNSA, çalışmakta olduğu bilgisayar üzerinde, *fnsaconf.dat* isimli koşullama dosyasında belirtilen dizinler ve bunların alt dizin ve dosya bilgilerini, kendi içindeki zamanlama

---

* *refttl* değeri şu şekilde formüle edilebilir;
refttl < (min (*htabttl*, *ptabttl*) * *chkttl*)

mekanizması yardımıyla, kullanıcı tarafından (*fnsaconf.dat* dosyasında) belirlenmiş aralıklarla FNS'ya yollamaktadır. FNSA, topladığı bilgileri, dinamik olarak yarattığı, linkli liste yapısındaki dizin ve dosya tablolarında tutulmaktadır. Bu tablolar, zamanlama mekanizması yardımıyla tetiklenen *getdir()* isimli yordam tarafından doldurulmaktadır. Koşullama dosyasında belirlenen dizinlerin hepsinin taranmasını takiben elde edilen sonuç FNS'ye *UPD_DIR_STU - 0x01* işlem kodu ile, bölüm 6.2.2.2. de tariflenen paket yapısına sadık kalınarak *wrt_dir()* yordamı ile yollanır. Koşullama dosyasında belirlenen her dizin, yeni bir paket dizisi ile FNS'ye ulaştırılmaktadır. FNSA'dan FNS'ye yapılan aktarımın sonlanmasını takiben *getdir()* yordamı ile dinamik olarak yaratılan dizin ve dosya tabloları, sistem kaynaklarını meşgul etmemek için ortadan kaldırılır.

6.2.4.1 FNSA'da iş akışına ait blok diyagramları

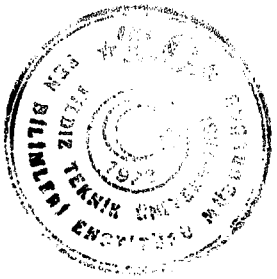
FNSA'da iş akışını belirleyen blok diyagramı ve kullanılan yordamlara ilişkin blok akış diyagramları EK -3 de sunulmuştur.

6.2.4.2 FNSA koşullama dosyası (fnsaconf.dat)

FNSA, *fnsaconf.dat* isimli koşullama dosyasında, kullanıcı tarafından belirlenen değerleri kullanarak FNS ile bağlantı kurar. Bu değerler şunlardır:

primary	FNSA'nin bilgileri yollayacağı FNS'inin adresini belirler.
refttl	FNSA'nın, kendi dosya sisteminden bilgileri alıp, bunları FNS'ye kaç saniyede bir yollayacağını belirlemek üzere kullanılan parametredir.
port	FNSA'nın, FNS ile kaç numaralı soket aracılığıyla haberleşeceğini belirler.
path	FNS'ye dahil olması istenen dizin isminden oluşan bir listedir. Bu listede verilen dizin isimleri, dosya sistemi üzerinde mevcut olmalı ve her dizin '\' işareti ile sonlandırılmalıdır.

FNSA, koşullama dosyasında, *primary*, *refttl* ve *port* değişkenlerine değer verilmemesi FNSA'nın çalışmasını engeller. Oluşan hata kullanıcıya iletilerek düzeltilmesi istenir. Benzer şekilde FNSA'nın düzgün çalışabilmesi için *path* ile FNS'ye dahil olması istenen



dizin isimlerinin birer boşluk bırakılarak ve '\' işareti ile sonlandırılarak yazılması gereklidir.

6.2.5 Dosya isim servis istemcisinin (FNSC) çalışma ilkesi

FNSC, kullanıcı programlarında kullanılan *fopen()* çağrısının yerini almak üzere hazırlanmıştır. Buna göre kullanıcı herhangi bir dosyayı açmak istediğinde şu notasyonlardan birini kullanabilecektir:

1. fopen('<host:><full_path><file>', '<attrib>')
2. fopen('<full_path><file>', '<attrib>')
3. fopen('<file>', '<attrib>')
4. fopen('<host:><file>', '<attrib>')

Bu notasyonlarda;

<host:> Bilgisayar (host) adresini ,

<full_path> Dizin adını,

<file> Dosya adını,

<attrib> Dosyaya nasıl erişileceğini belirlemek için kullanılmaktadır.

Dosyalara, yaratmak (create, w, wb, w+, wb+), okumak (read, r, rb, r+, rb+) ya da eklemek (append, a, ab, a+, ab+) için erişilir. Kullanıcı, herhangi bir *<host:>* ifadesi kullanılmaksızın, dosya yaratmak isterse, yerel dosya sisteminde (local file system) *<full_path><file>* ile belirlenen şekilde yaratılacaktır. *<full_path>* yapısı '/' özel işareti ile başlamalı ve sonlandırılmalıdır. Dosya, okuma veya ekleme amaçlı olarak açılacak ise, yerel dosya sistemi taranacak, daha sonra FNS'ye yollanacak sorguya alınacak cevaptan oluşturulacak sonuç listesi (eğer birden fazla seçenek oluşmuş ise) kullanıcıya sunularak bir tanesini seçmesi istenecektir. Eğer kullanıcı, uzak bir sistem üzerindeki dosyaya, bilinçli olarak *<host:><full_path><file>* veya *<host:><file>* ile referans vermiş veya kendisine sunulan liste üzerindeki dosya bilgilerinden yararlanarak, uzak dosya sistemi üzerinde bulunan bir dosyayı seçmiş ise, FNSC, *<host:>* ile belirlenen bilgisayarın dosya sisteminin herhangi bir şekilde kendisine bağlı (mount) dosya sistemleri arasında olup olmadığını kontrol edecektir. Kullanıcının seçtiği dosyanın bulunduğu bilgisayarın

(<*host:*>), dosya sisteminin herhangi bir bağı yok ise* hata kodu dönecektir. Bağ mevcut ise <*full_path*> ile belirlenen dizinin, bağlantılı olan dosya sistemine ait dizin tarafından kapsanıp kapsanmadığı kontrol edilecektir. Eğer tüm koşullar sağlanmış ise FNSC, dosya erişim sistemi olarak kullandığı NFS'den yararlanarak, kullanıcıya, aradığı dosyaya NFS üzerinden erişebilmesi için gerekli bağıl (relative) adres bilgisini döndürecektir. Bu sayede kullanıcı, kullanmakta olduğu dosyanın bulunduğu bilgisayar sisteminden bağımsız olarak, kendisine tanınan haklar çerçevesinde dosya üzerinde işlemlerini yapabilecektir.

6.2.5.1 FNSC'de iş akışına ait blok diyagramları

FNSC'de iş akışını belirleyen blok diyagramı ve kullanılan yordamlara ilişkin blok akış diyagramları EK -4 de sunulmuştur.

6.2.5.2 FNSC koşullama dosyası (fnscconf.dat)

FNSC, fnscconf.dat isimli koşullama dosyasından aldığı, kullanıcı tarafından değiştirilebilen değerleri kullanarak görevini yerine getirmektedir. Bu değerler:

primary FNSC'nin temas kuracağı birincil FNS'nin adresidir.

secondary FNSC eğer birincil FNS'den belirlenen süre içerisinde cevap alamazsa isteğini secondary değerini takip eden ikincil FNS'lere yollayacaktır. Sistem en fazla sekiz ikincil sunucuyu desteklemektedir.

port FNS (veya SFNS) ile kaç numaralı soket aracılığıyla bağlantı kuracağını belirleyen değerdir.

ttl FNSC'nin cevap almak için kaç saniye beklemesi gerektiğini belirler. Eğer bu süre içinde *primary* ile belirlenen FNS'den cevap alınamamış ise FNSC, *secondary* ile belirlenen SFNS'leri sırası ile deneyecektir. Her denemede *ttl* değeri sonuç almak için saniye cinsinden beklenmesi gereken süreyi belirler.

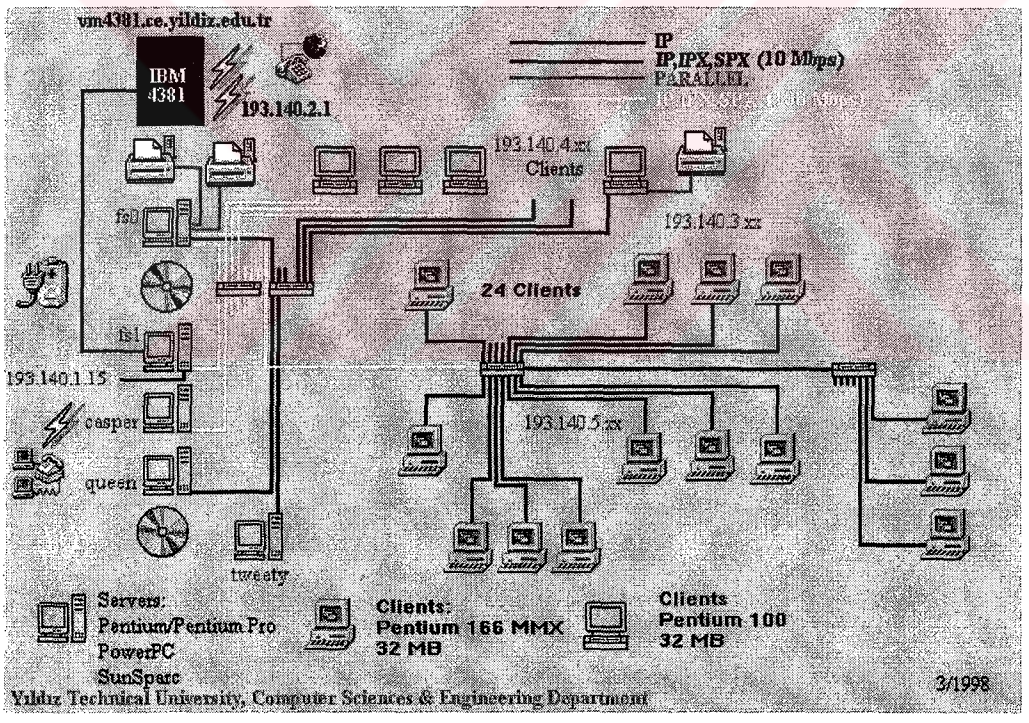
---

* *mount*, yetki gerektiren bir komuttur. Bu nedenle yetkili bir kullanıcı tarafından çalıştırılmalıdır. Bu komutun kontrolsüz bir biçimde kullanılmasının yaratacağı güvenlik problemleri göz önüne alınarak, bağlı olmayan bir dosya sisteminin otomatik bir şekilde bağlanması (mount) doğru olmayacaktır.

7. DENEYSEL ÇALIŞMALAR

7.1 Deneysel Çalışmalarda Kullanılan Bilgisayar Ağının Altyapısı

Deneysel çalışmalar boyunca Yıldız Teknik Üniversitesi Bilgisayar Bilimleri ve Mühendisliği Bölümünce kurulan ve desteklenen bilgisayar ağ altyapısı kullanılmıştır (Şekil 7-1). Bu altyapı, farklı hızlarda (10Mbps / 100Mbps) ethernet segmentlerinden oluşan, çeşitli işletim sistemlerine sahip sunucu bilgisayarları içermektedir. Bilgisayar ağı, IEEE 802.2 ve IEEE 802.3 desteklidir. Ağ üzerinde iletişim protokolü olarak TCP/IP kullanılmaktadır. Alt yapıya bağlı bilgisayarlar, DNS kontrolündeki IP numaralarına ve karşı düşen isimlere sahiptir. Ağ üzerindeki bilgisayar sistemlerinin bağlantıları Şekil 7.1'de gösterilmiştir.



Şekil 7.1 YTÜ Bilgisayar Bilimleri ve Mühendisliği Bölümü yerel alan ağ yapısı

7.2 Kullanılan Yazılımlar

Tez çalışması sırasında kullanılan yazılımları iki temel grup altında toplamak mümkündür. Bunlar, destek yazılımları ve tez çalışması boyunca geliştirilen sistem/uygulama yazılımlardır. Destek yazılımları, mevcut altyapıya dahil olarak çalışmakta olan bilgisayar sistemleri üzerinde temel olarak kullanılan yazılımlardır. Bunlar;

1. İşletim sistemi (Unixware, AIX, SunOS vb..),
2. Bilgisayar ağı ile etkileşimi sağlayan TCP/IP yazılımları,
3. NFS (Network File System) yazılımı

7.3 Dosya İsim Servisi'nin Deneysel Sonuçları

7.3.1 Dosya isim servisinin uygulandığı bilgisayarlar ve özellikleri

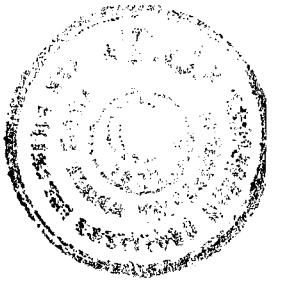
Deneysel çalışmalarda Tablo 7.1 de işlemci ve işletim sistemi özellikleri verilen bilgisayar sistemleri kullanılmıştır.

Tablo 7.1Kullanılan Bilgisayar Sistemleri ve Özellikleri

Bilgisayar	İşlemci	RAM	İşletim Sistemi	Ağ Bağlantı Hızı
casper	Intel MMX - 166 Mhz	32 MB	UnixWare 2.1	100 Mbps
queen	PowerPC 604 - 100 Mhz	32 MB	Aix 4.1.4	10 Mbps
tweety	SunSparc - 75 Mhz	64 MB	SunOS 4.1.3	10 Mbps

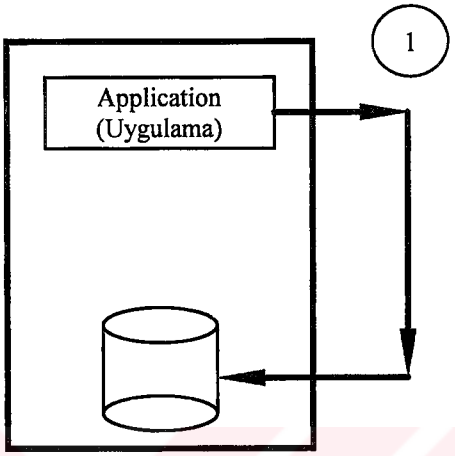
7.3.2 Dosya isim servisinde kullanılan senaryolar

Dosya İsim Servisinin deneysel sonuçlarını elde edebilmek üzere değişik senaryolar kullanılmıştır. Senaryolarda elde edilen sonuçlar, değişik aralıklarla 50 defa tekrarlanan 1000 adetlik okuma/yazma/ekleme işlemlerinin ortalamasıdır. Kullanılan senaryolar, amaçları ve uygulanmaları ile elde edilen sonuçlar şöyledir:

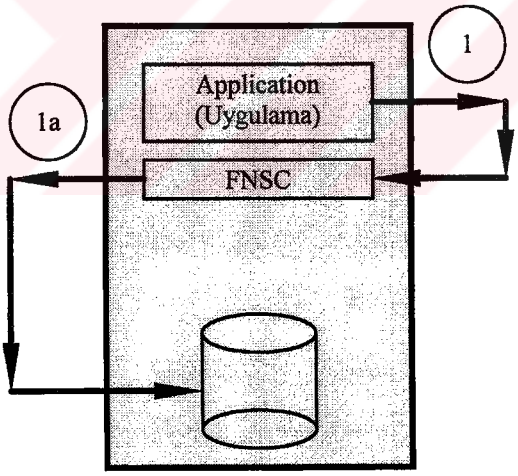


7.3.2.1 Senaryo 1

Bu senaryo, yerel disk üzerinde yapılan dosya erişimlerinde, normal fopen() çağrısı (Şekil 7.2) ile FNSC-fopen() çağrısının (Şekil 7.3) kullanılması durumunu incelemektedir. Senaryonun sonuçları karşılaştırmalı olarak, okuma (r+) Tablo 7.1'de, yazma (w+) Tablo 7.2'de ve ekleme (a+) Tablo 7.3'de sunulmuştur.

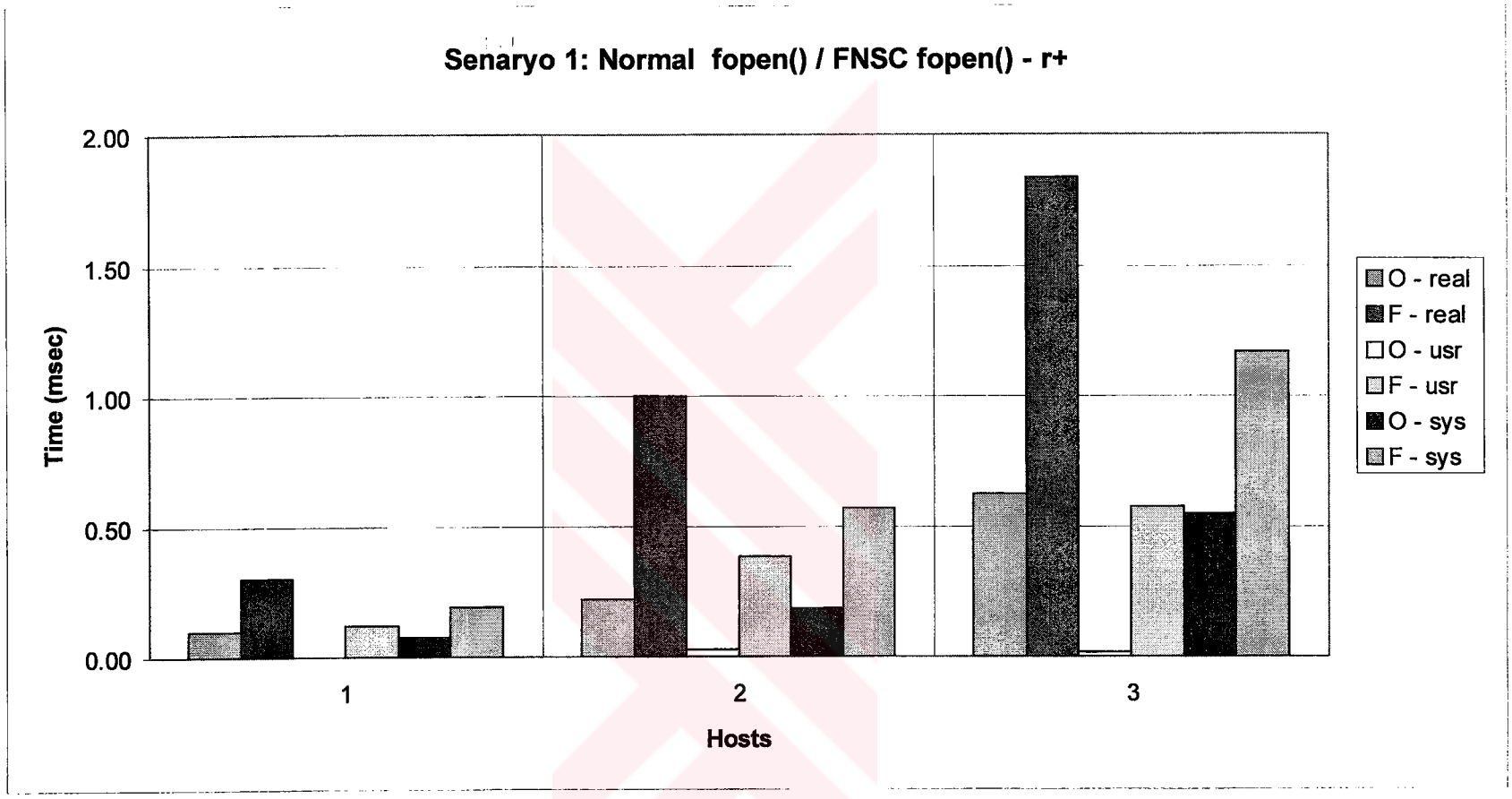


Şekil 7.2 Senaryo 1: Yerel disk üzerinde normal fopen() çağrısının kullanılması



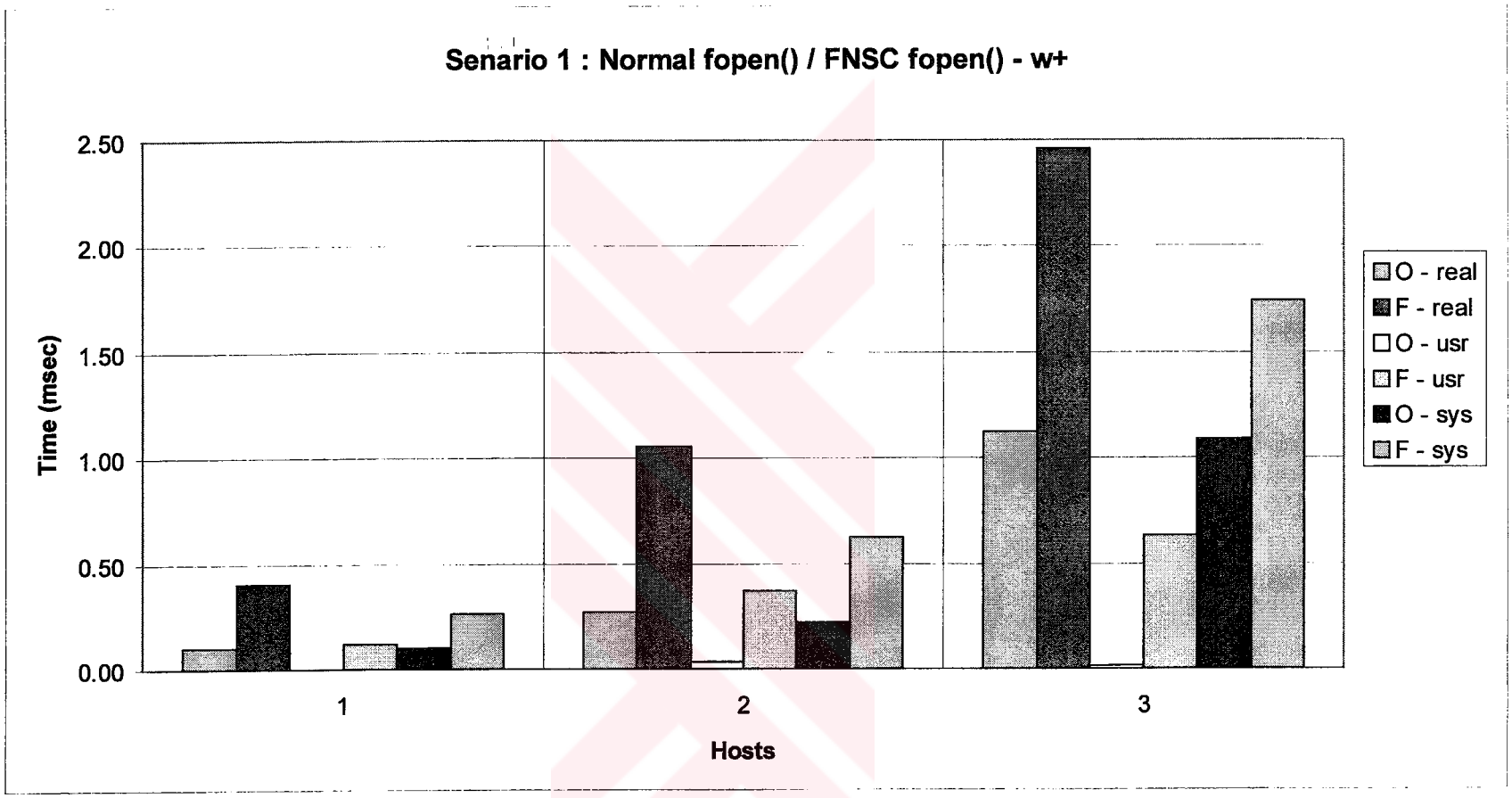
Şekil 7.3 Senaryo 1: Yerel disk üzerinde FNSC-fopen() çağrısının kullanılması

Tablo 7.2 Senaryo 1: Normal fopen() / FNSC-fopen() (r+) sonuçları



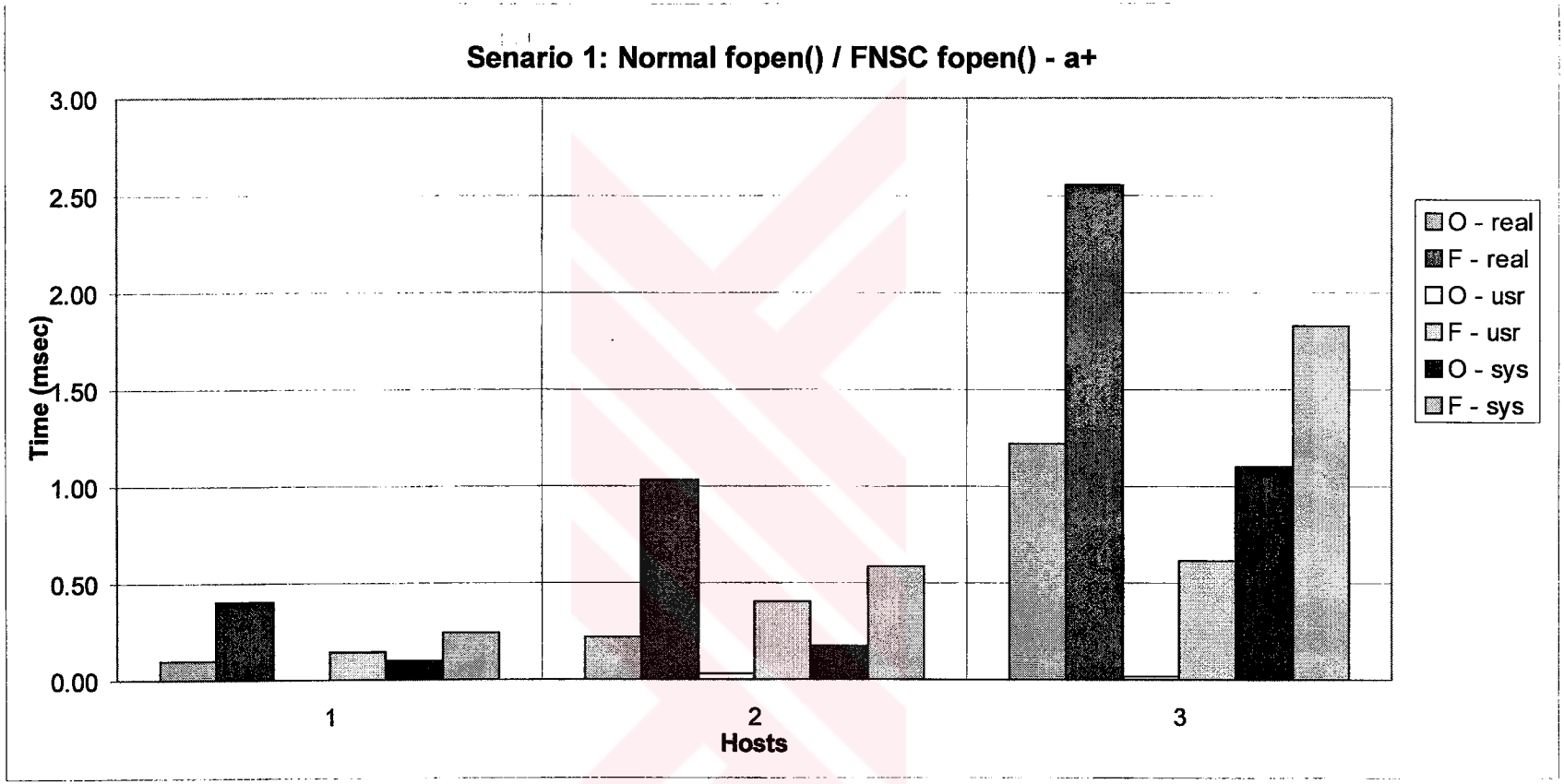
(r+) ortalama	Normal fopen()			FNSC fopen()		
	casper	queen	tweety		casper	queen	tweety
real	0.10	0.22	0.62	real	0.30	1.00	1.84
usr	0.00	0.03	0.02	usr	0.12	0.38	0.57
sys	0.07	0.18	0.54	sys	0.19	0.57	1.17

Tablo 7.3 Senaryo 1: Normal fopen() / FNSC-fopen() (w+) sonuçları



(w+) ortalama	Normal fopen()			FNSC fopen()		
	casper	queen	tweety	casper	queen	tweety
real	0.10	0.27	1.12	0.40	1.06	2.46
usr	0.00	0.03	0.01	0.12	0.37	0.63
sys	0.10	0.22	1.09	0.26	0.63	1.74

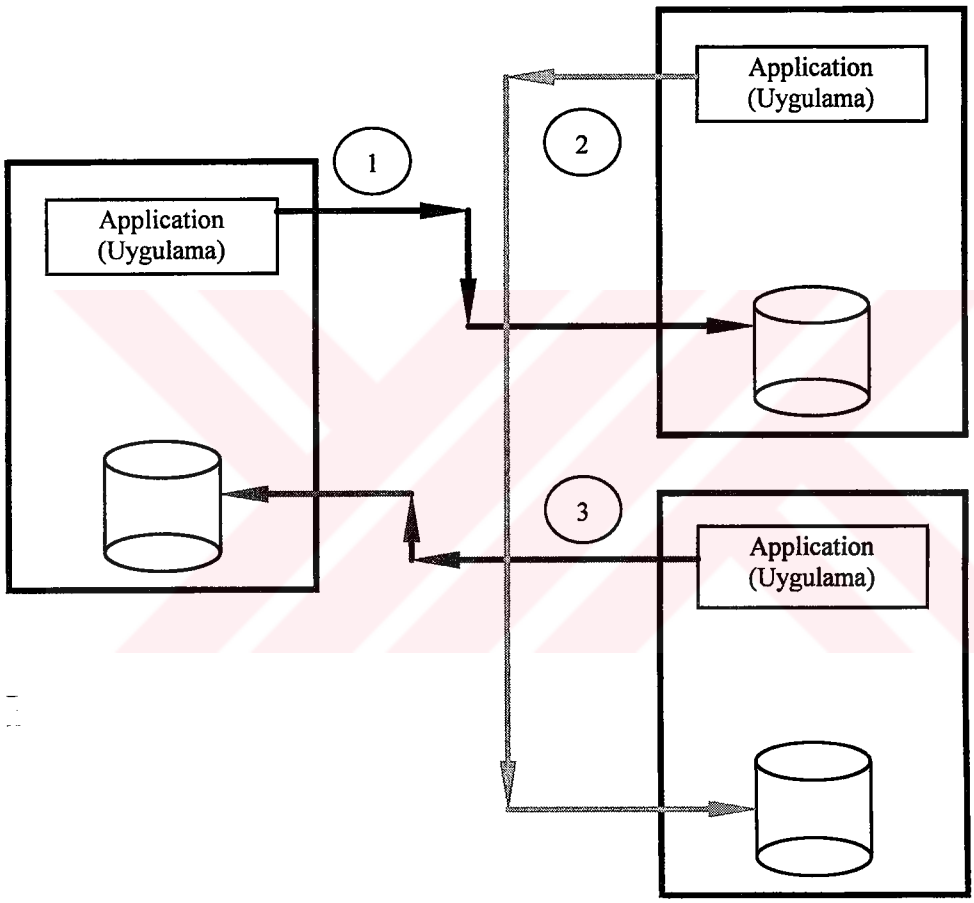
Tablo 7.4 Senaryo 1: Normal fopen() / FNSC-fopen() (a+) sonuçları



(a+) ortalama	Normal fopen()			FNSC fopen()		
	casper	queen	tweety		casper	queen	tweety
real	0.10	0.22	1.22	real	0.40	1.03	2.56
usr	0.00	0.03	0.01	usr	0.15	0.40	0.61
sys	0.10	0.18	1.10	sys	0.24	0.58	1.82

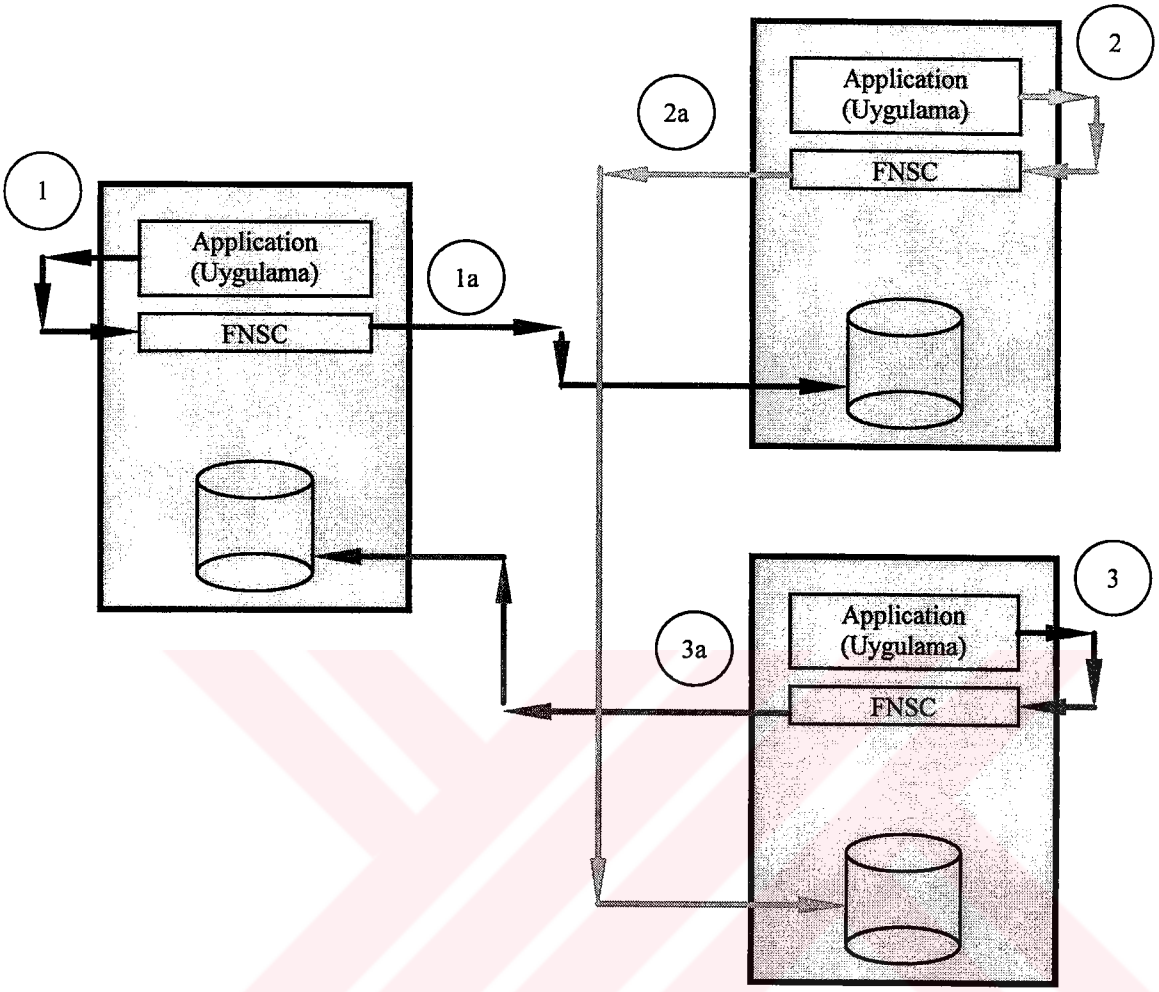
7.3.2.2 Senaryo 2

Bu senaryo, NFS üzerinden yapılan dosya erişimlerinde, normal fopen() çağrısı (Şekil 7.4) ile FNSC-fopen()* çağrısının (Şekil 7.5) kullanılması durumunu incelemektedir. FNSC-fopen() çağrısının kullanılırken, sunucu (FNS) ve ajan (FNSA) çalıştırılmamıştır. Senaryonun uygulanması ile, farklı zamanlarda, farklı bilgisayarların birbirlerinin disklerindeki dosyalara erişim süreleri ölçülerek elde edilen değerler, okuma (r+) Tablo 7.5'de, yazma (w+) Tablo 7.6'de ve ekleme (a+) Tablo 7.7'da sunulmuştur.



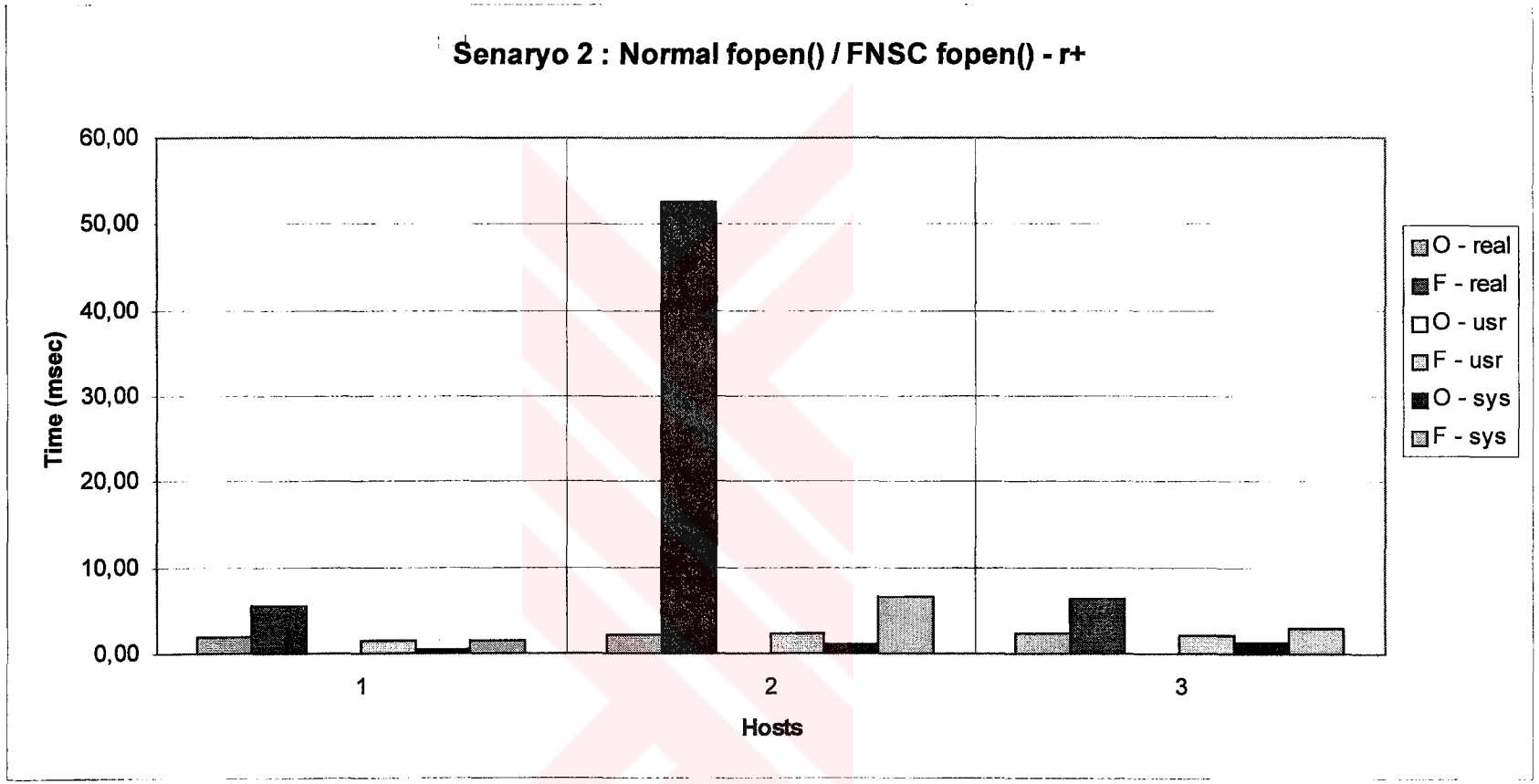
Şekil 7.4 Senaryo 2: NFS üzerinden normal fopen() çağrısının kullanılması (ayrık zamanlı istemci)

* FNSC modülünde kullanılan fopen() çağrısı, diğer bir sisteme erişmesi sırasında Internet isimlerinin adreslere dönüştürülmesini sağlayan DNS (Domain Name Server/Service) den yararlanmaktadır. Bu nedenle kullanıcıya yansıyan süreler isim-adres dönüşümü için harcanan zamanı da içermektedir.



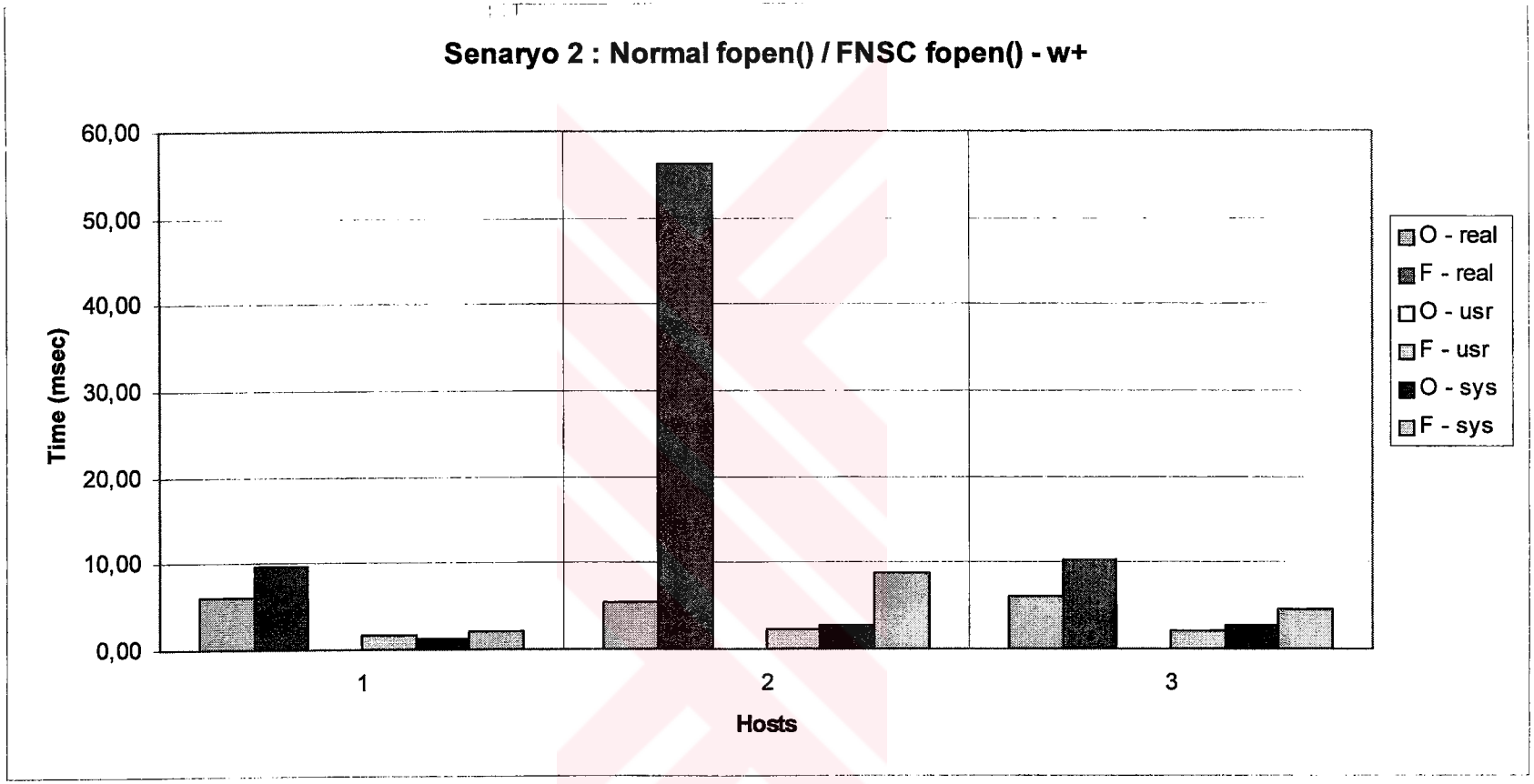
Şekil 7.5 Senaryo 2: NFS üzerinden FNSC-fopen() çağrısının kullanılması (ayrık zamanlı istemci)

Tablo 7.5 Senaryo 2 : Normal fopen() / FNSC-fopen() (r+) sonuçları



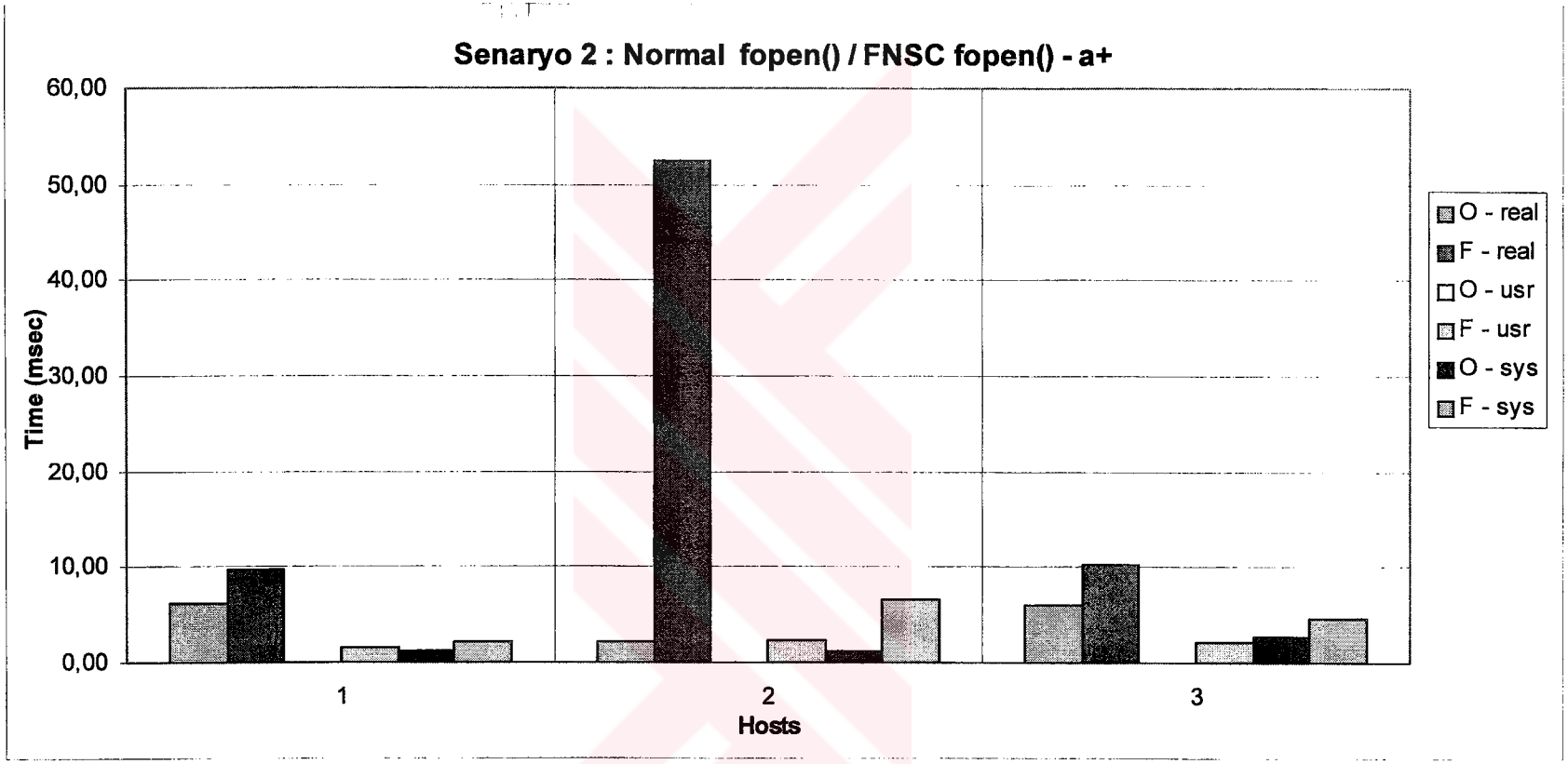
(r+) ortalama	Normal fopen()			FNSC fopen()		
	casper	queen	tweety	casper	queen	tweety
real	1,85	2,19	2,29	5,54	52,66	6,38
usr	0,00	0,06	0,01	1,57	2,22	2,17
sys	0,47	1,14	1,19	1,53	6,58	2,89

Tablo 7.6 Senaryo 2 : Normal fopen() / FNSC-fopen() (w+) sonuçları



(w+) ortalama	Normal fopen()				FNSC fopen()		
	casper	queen	tweety		casper	queen	tweety
real	6,14	5,55	6,11	real	9,58	56,32	10,24
usr	0,00	0,07	0,03	usr	1,65	2,32	2,15
sys	1,16	2,68	2,64	sys	2,16	8,88	4,58

Tablo 7.7 Senaryo 2 : Normal fopen() / FNSC-fopen() (a+) sonuçları



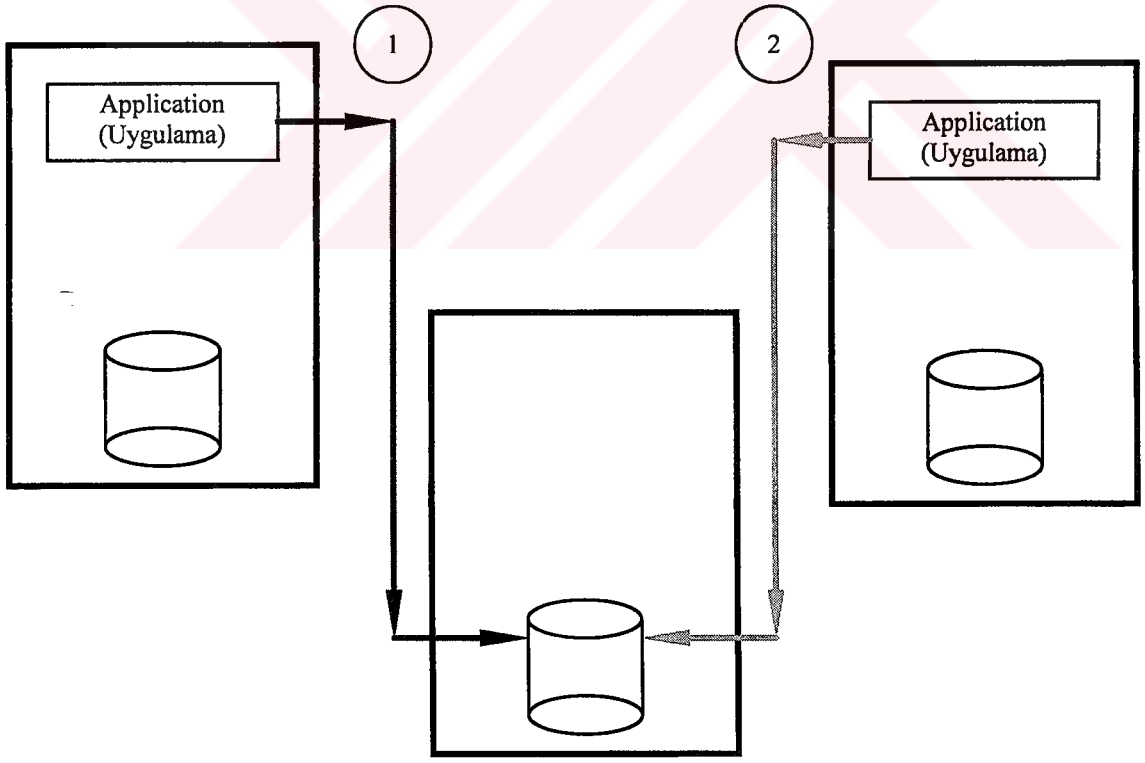
(a+) ortalama	Normal fopen()			FNSC fopen()		
	casper	queen	tweety	casper	queen	tweety
real	6,11	2,16	6,07	9,68	52,47	10,26
usr	0,01	0,04	0,05	1,62	2,25	2,22
sys	1,16	1,10	2,72	2,20	6,67	4,59

7.3.2.3 Senaryo 3

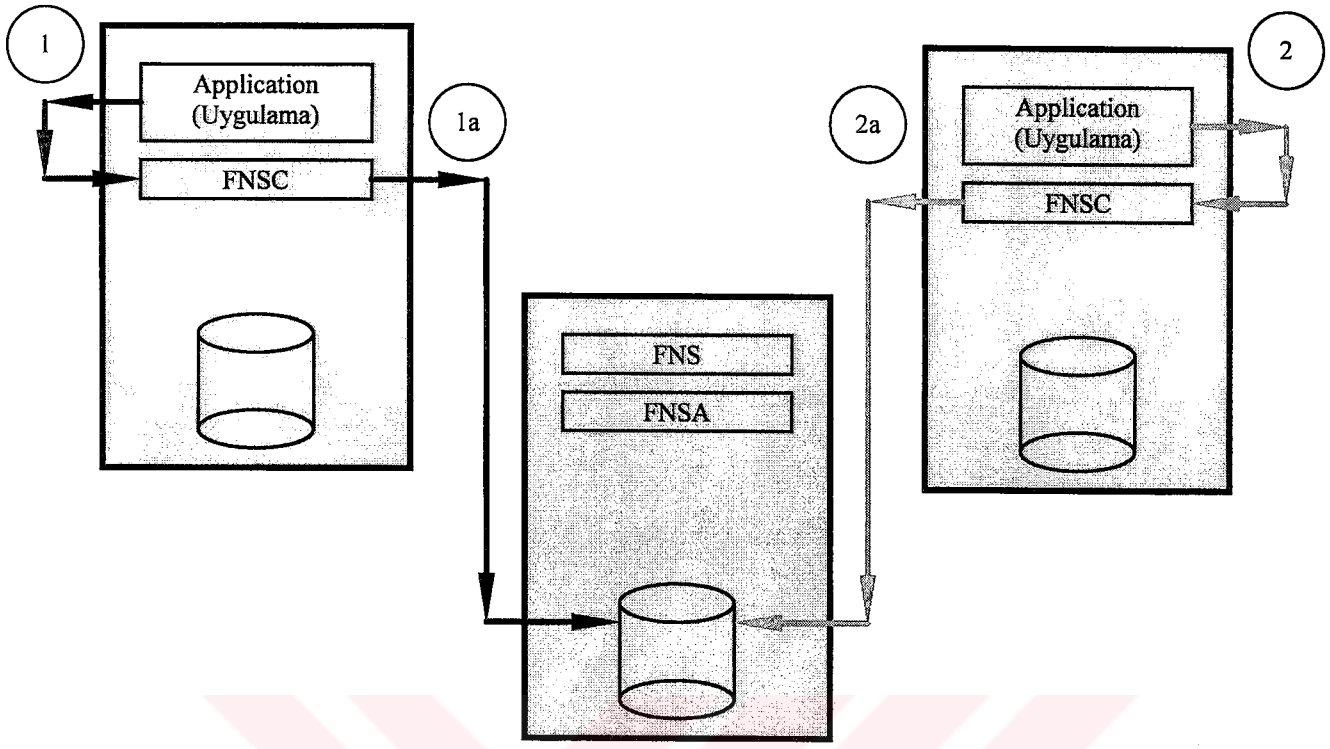
Senaryo 3, sunucu (FNS) ve ajan (FNSA) modülleri aynı bilgisayar üzerinde çalışırken FNSC-fopen()* çağrısının, sunucu (FNS) üzerinden yaptığı sorgulama ile dosyaya eriştiği durumu incelemektedir. FNS ve FNSA modüllerinin aynı bilgisayar üzerinde çalışması nedeniyle işlem yükü tek bilgisayar üzerinde odaklanmıştır. Senaryoda, sunucu (FNS) ve ajan (FNSA) modülleri *casper* isimli bilgisayarda çalışırken, istemci (FNSC) modülleri *queen* ve *tweety* isimli bilgisayarlarda çalıştırılmıştır. Senaryo, normal fopen() çağrısı ile FNSC-fopen() çağrısının kullanımını karşılaştıran üç ayrı testi içermektedir;

7.3.2.3.1 Senaryo 3 Test A

queen ve *tweety* isimli bilgisayarlardan farklı zamanlarda (1 istemci), *casper* isimli bilgisayar üzerindeki dosyalara NFS üzerinden normal fopen() çağrısı (Şekil 7.6) ile FNSC-fopen() çağrısı (Şekil 7.7) ile yapılan erişim sonuçlarını inceler. Dosya erişim süreleri ölçülerek elde edilen değerler, okuma (r+) Tablo 7.8'de, yazma (w+) Tablo 7.9'da ve ekleme (a+) Tablo 7.10'da sunulmuştur.

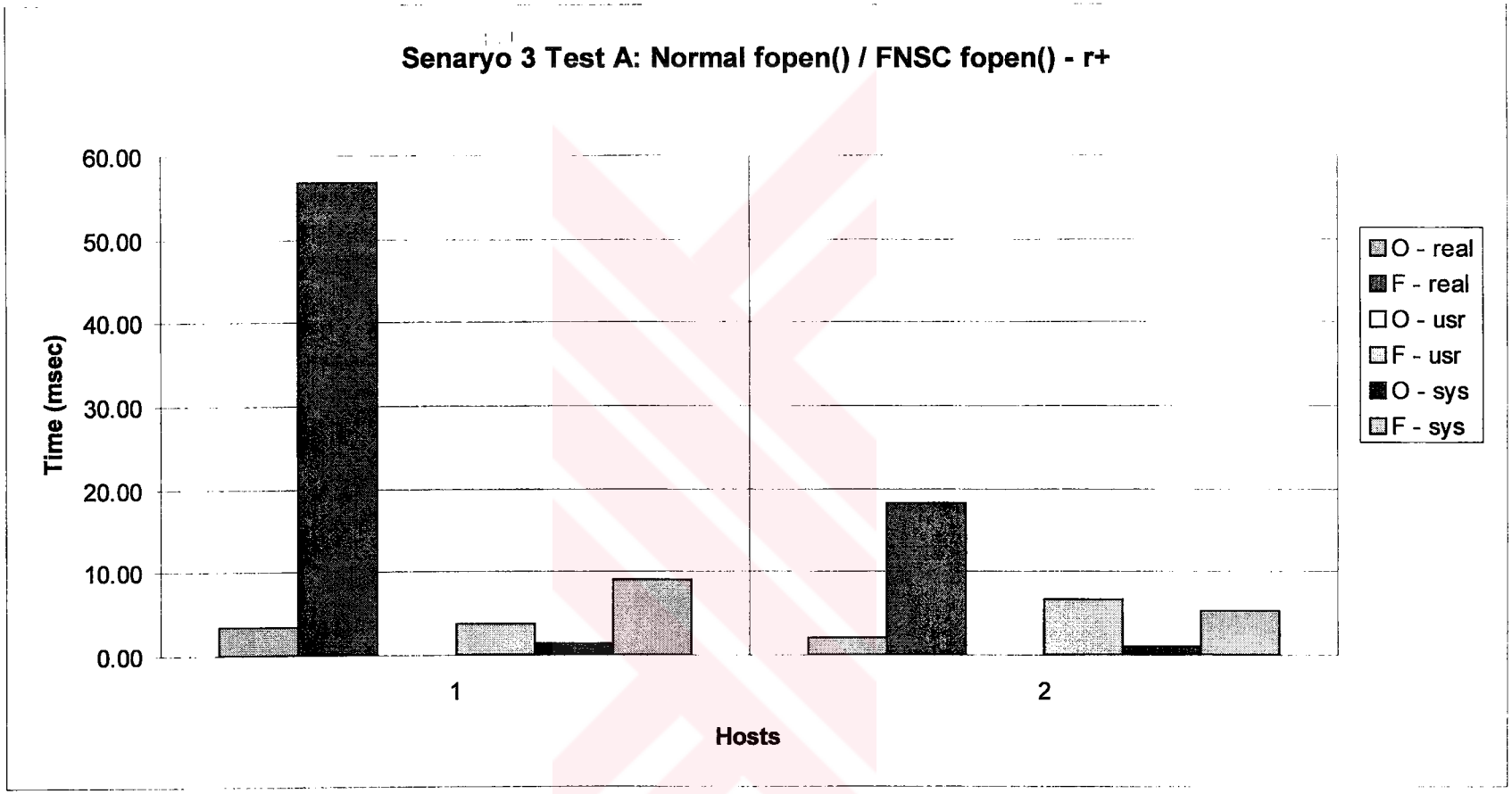


Şekil 7.6 Senaryo 3 Test A: NFS üzerinden normal fopen() çağrısının kullanılması (ayrık zamanlı istemci)



Şekil 7.7 Senaryo 3 Test A: NFS üzerinden FNSC-fopen() çağrısının kullanılması (ayrık zamanlı istemcinin sunucu üzerinden yaptığı sorgulama ile erişimi)

Tablo 7.8 Senaryo 3 Test A : Normal fopen() / FNSC-fopen() (r+) sonuçları

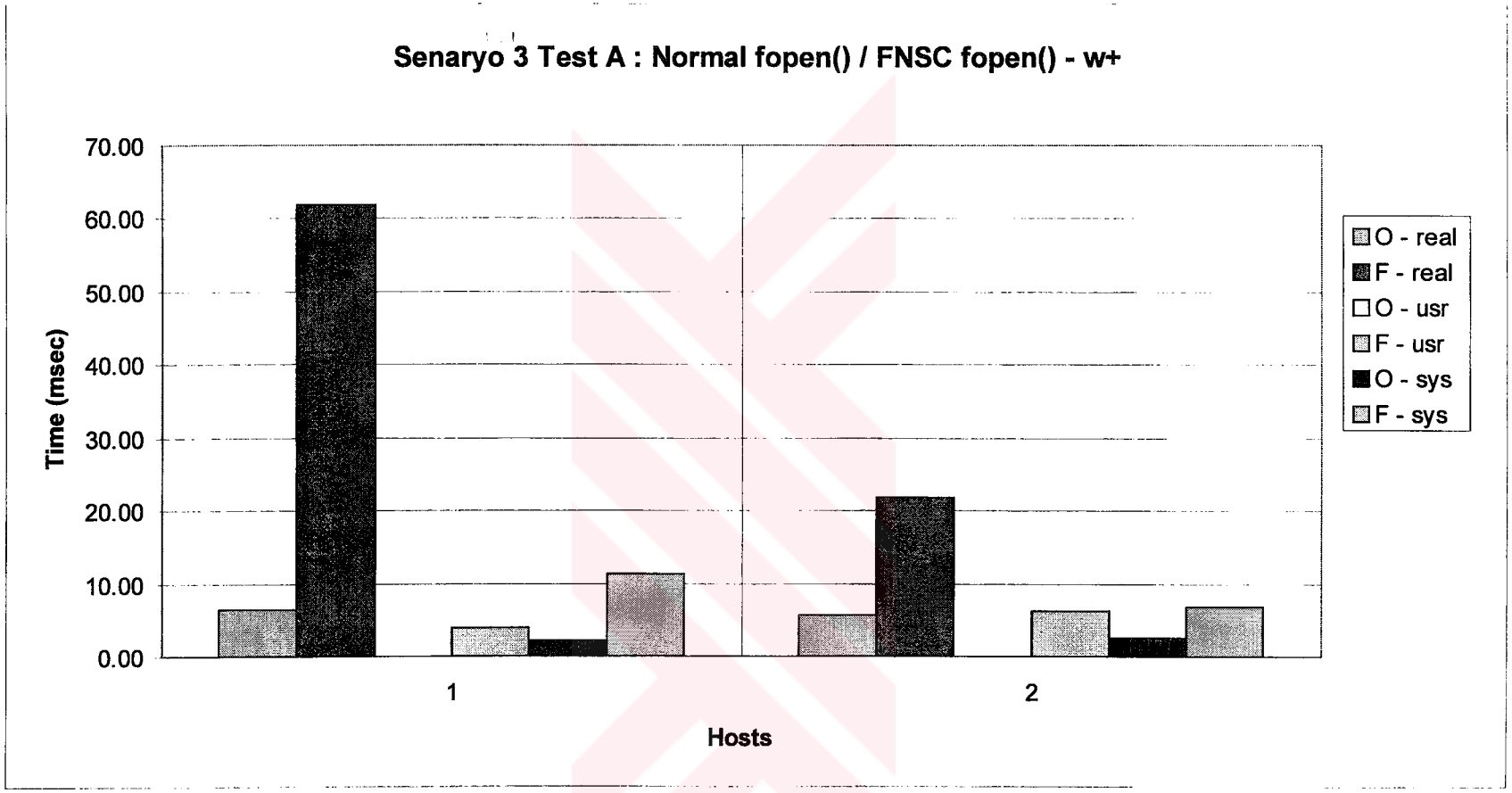


(r+) ortalama

	Normal fopen() queen	Normal fopen() tweety
real	3.47	2.11
usr	0.06	0.01
sys	1.38	1.06

	FNSC fopen() queen	FNSC fopen() tweety
real	56.75	18.26
usr	3.72	6.64
sys	8.97	5.35

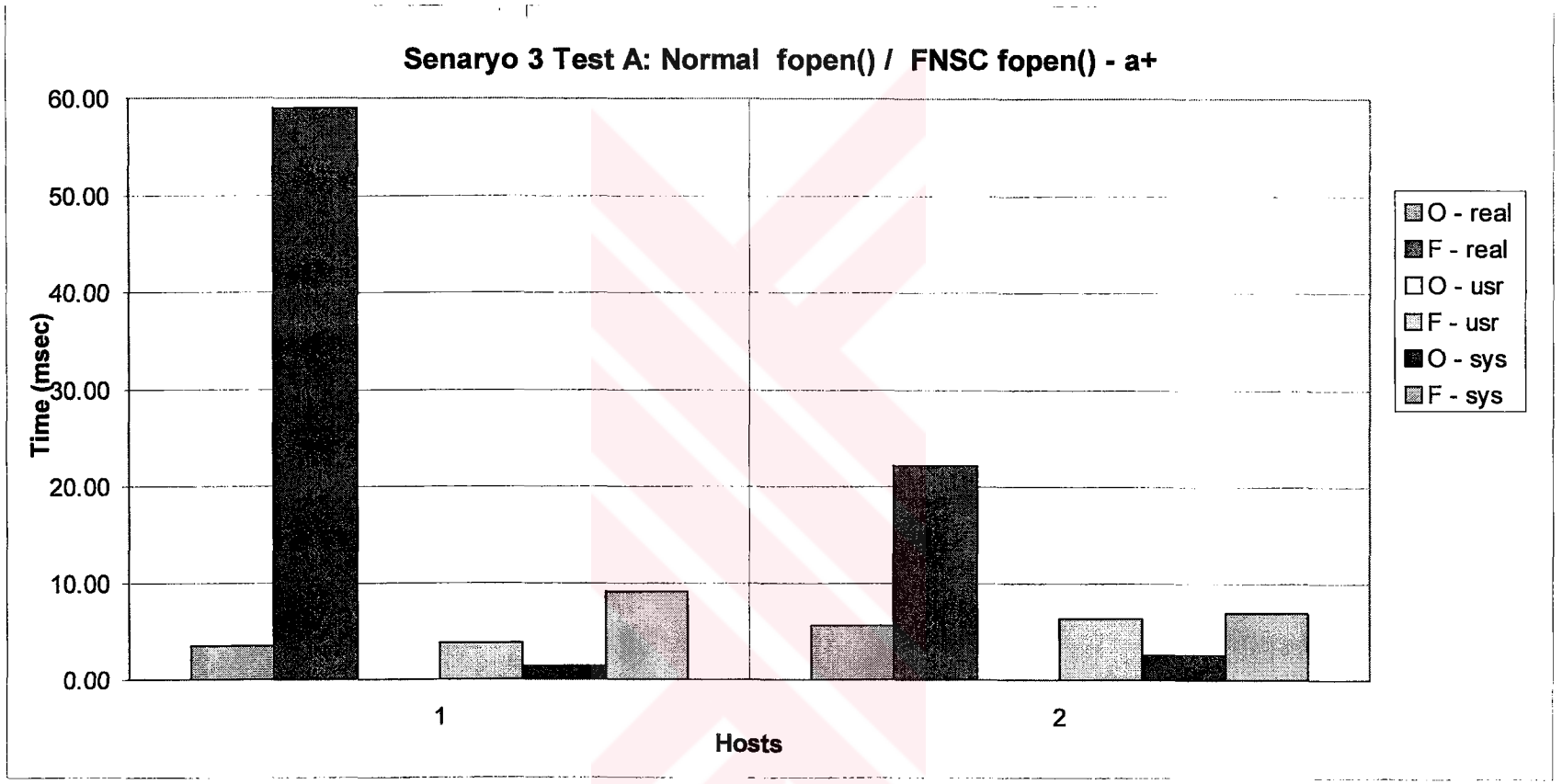
Tablo 7.9 Senaryo 3 Test A : Normal fopen() / FNSC-fopen() (w+) sonuçları



(w+) ortalama	Normal fopen()	
	queen	tweety
real	6.38	5.58
usr	0.05	0.01
sys	2.16	2.49

	FNSC fopen()	
	queen	tweety
real	61.75	21.79
usr	3.82	6.31
sys	11.21	6.83

Tablo 7.10 Senaryo 3 Test A : Normal fopen() / FNSC-fopen() (a+) sonuçları

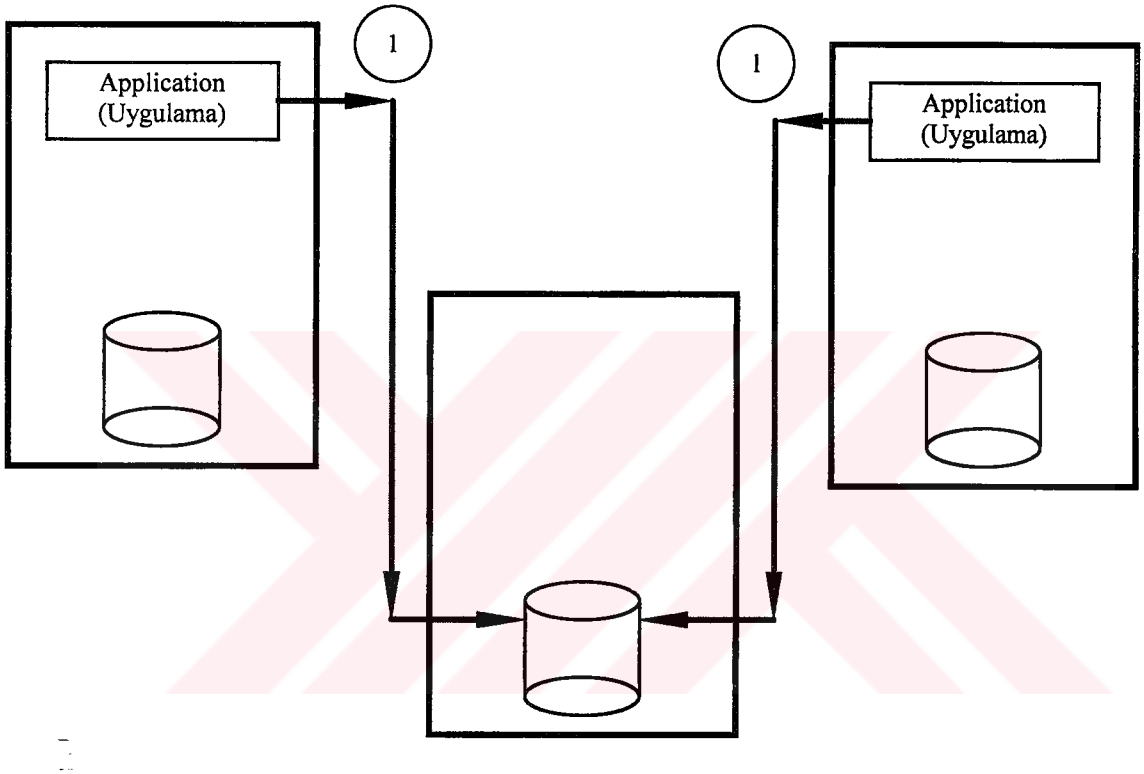


(a+) ortalama

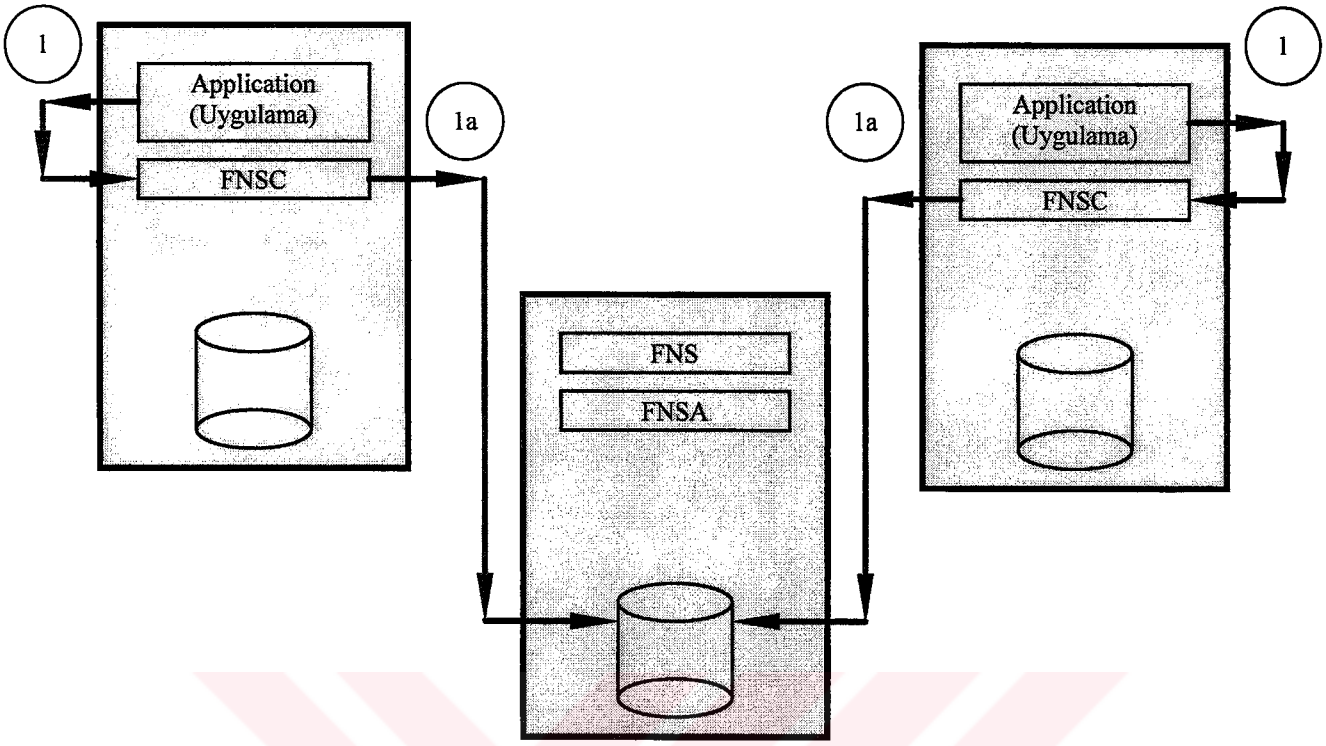
	Normal fopen() queen	Normal fopen() tweety	FNSC fopen() queen	FNSC fopen() tweety
real	**3.46**	**5.64**	**58.98**	**22.04**
usr	**0.06**	**0.03**	**3.74**	**6.33**
sys	**1.37**	**2.58**	**9.07**	**6.96**

7.3.2.3.2 Senaryo 3 Test B

queen ve *tweety* isimli bilgisayarlardan aynı anda birer istemci ile (2x1 istemci), *casper* isimli bilgisayar üzerindeki dosyalara NFS üzerinden normal fopen() çağrısı (Şekil 7.8) ile FNSC-fopen() çağrısı (Şekil 7.9) ile yapılan erişim sonuçlarını inceler. Dosya erişim süreleri ölçülerek elde edilen değerler, okuma (r+) Tablo 7.11'de, yazma (w+) Tablo 7.12'de ve ekleme (a+) Tablo 7.13'de sunulmuştur.

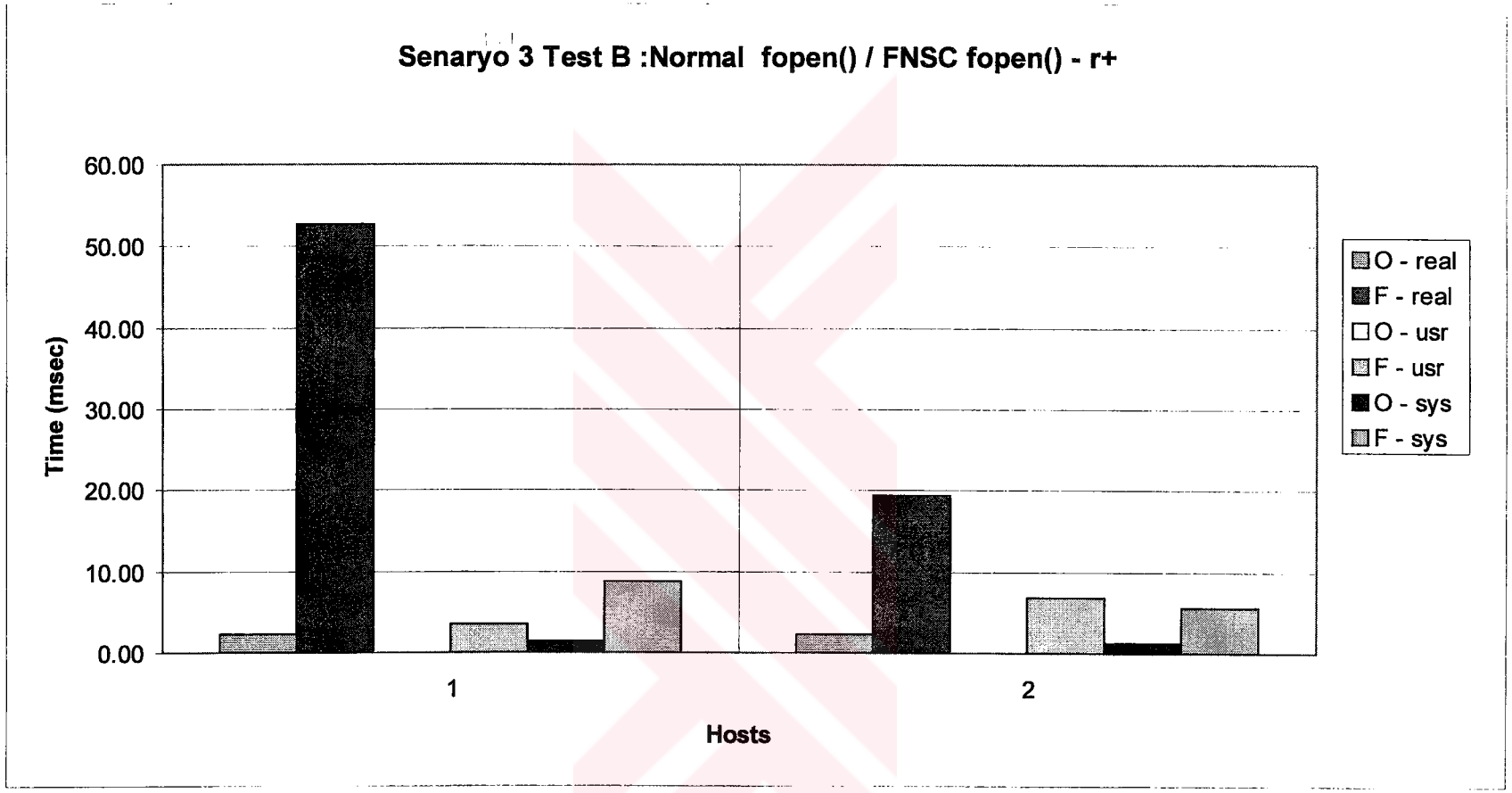


Şekil 7.8 Senaryo 3 Test B : NFS üzerinden normal fopen() çağrısı kullanılması (eş zamanlı 2 istemci)



Şekil 7.9 Senaryo 3 Test B : NFS üzerinden FNSC-fopen() çağrısının kullanılması (eş zamanlı 2 istemcinin sunucu üzerinden yaptıkları sorgulama ile erişimleri)

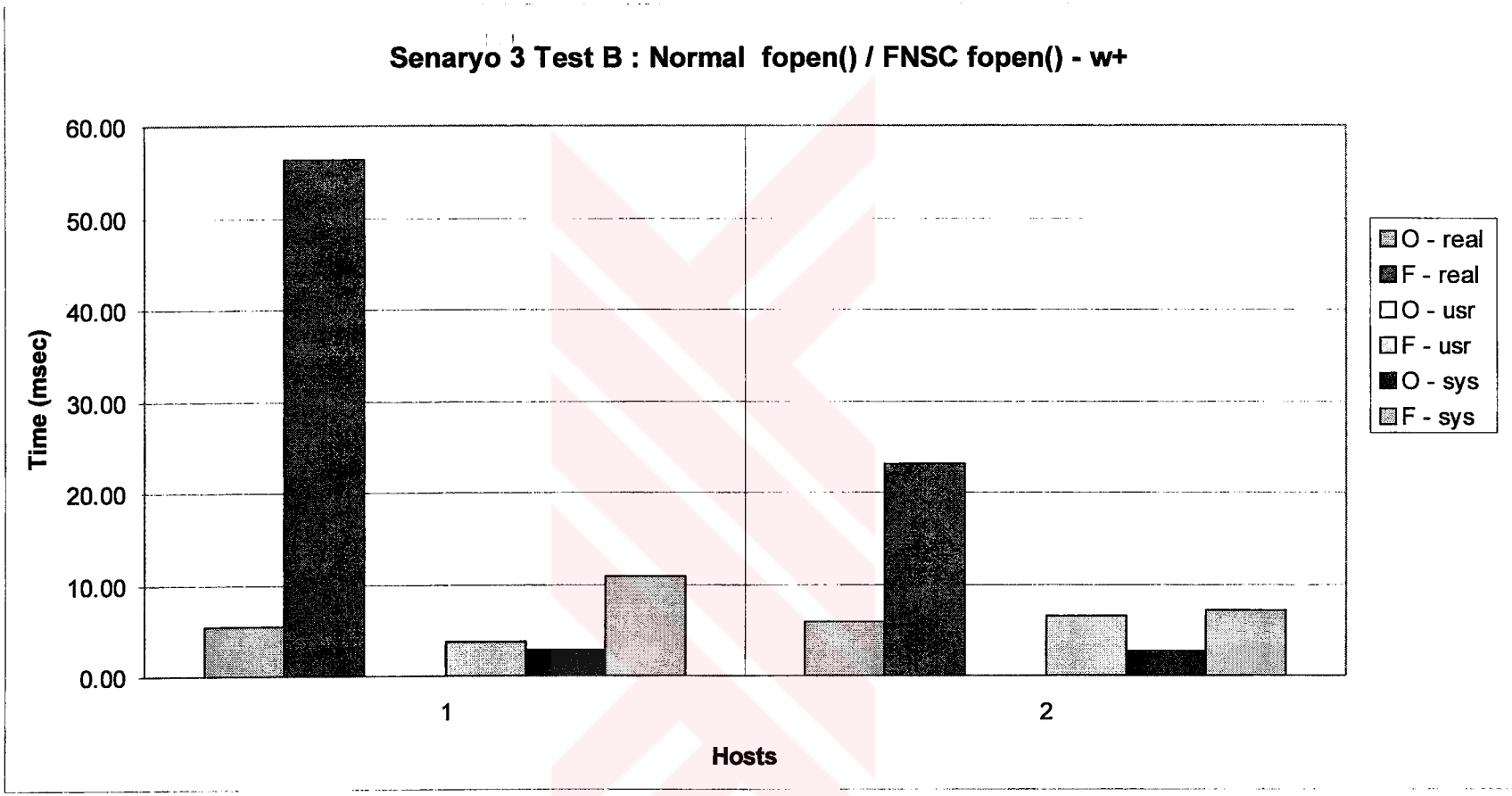
Tablo 7.11 Senaryo 3 Test B : Normal fopen() / FNSC-fopen() (r+) sonuçları



(r+) ortalama	Normal fopen()	
	queen	tweety
O - real	2.33	2.30
O - usr	0.06	0.01
O - sys	1.37	1.20

	FNSC fopen()	
	queen	tweety
F- real	52.61	19.39
F - usr	3.48	6.72
F - sys	8.75	5.58

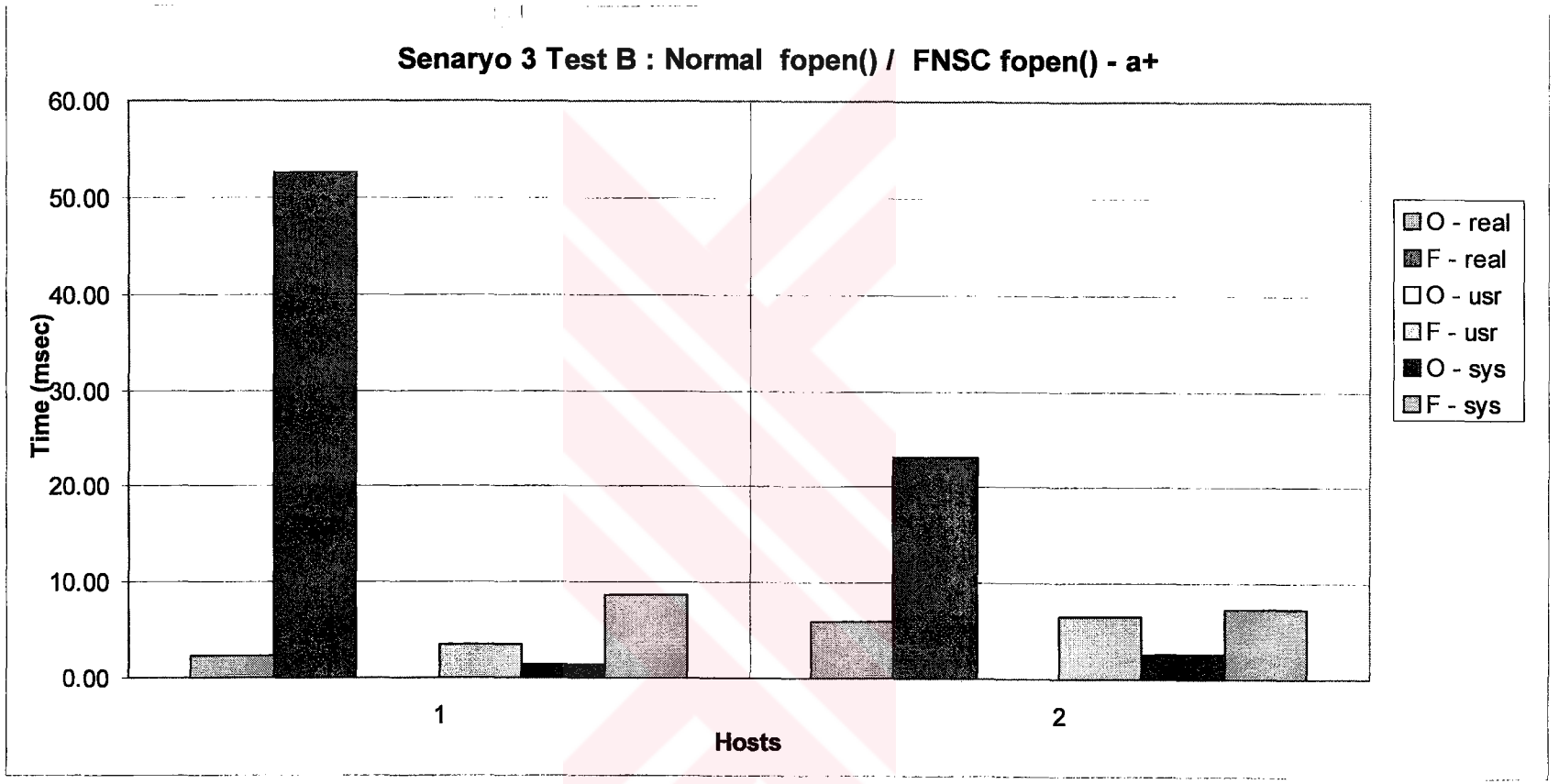
Tablo 7.12 Senaryo 3 Test B : Normal fopen() / FNSC-fopen() (w+) sonuçları



(w+) ortalama	Normal fopen()	
	queen	tweety
O - real	5.40	5.93
O - usr	0.07	0.01
O - sys	2.72	2.62

	FNSC fopen()	
	queen	tweety
F- real	56.35	23.15
F - usr	3.70	6.47
F - sys	10.78	7.09

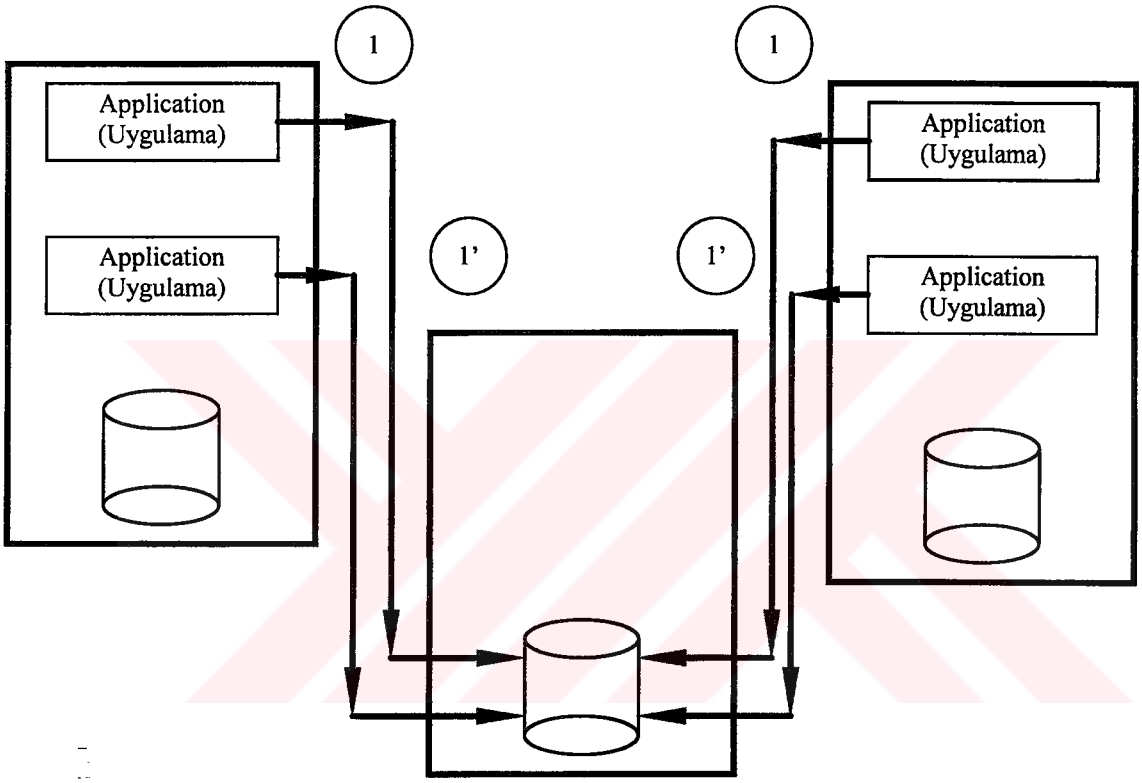
Tablo 7.13 Senaryo 3 Test B : Normal fopen() / FNSC-fopen() (a+) sonuçları



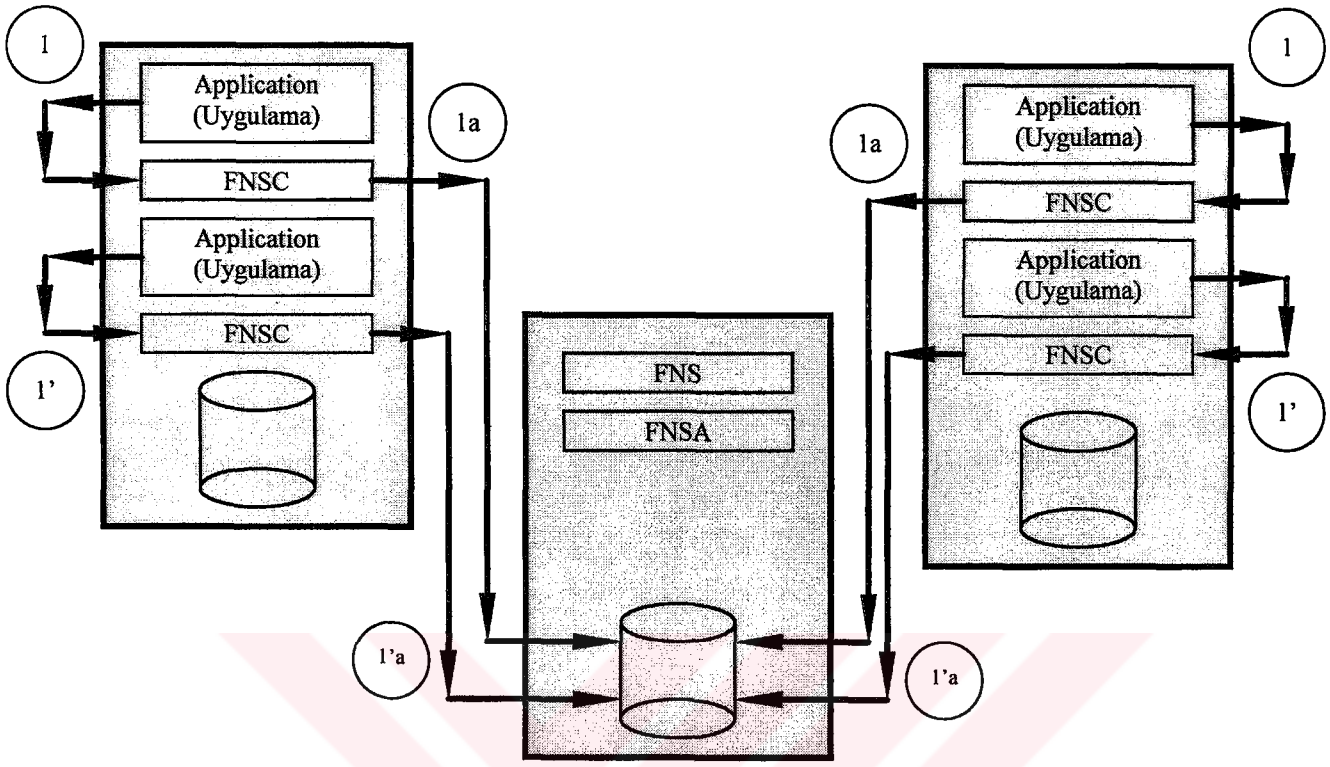
(a+) ortalama	Normal fopen() queen	tweety		FNSC fopen() queen	tweety
O - real	2.29	5.92	F- real	52.54	23.15
O - usr	0.05	0.03	F - usr	3.46	6.48
O - sys	1.35	2.65	F - sys	8.71	7.25

7.3.2.3.3 Senaryo 3 Test C

queen ve *tweety* isimli bilgisayarlardan aynı anda ikişer istemci ile (2x2 istemci), *casper* isimli bilgisayar üzerindeki dosyalara NFS üzerinden normal fopen() çağrısı (Şekil 7.10) ile FNSC-fopen() çağrısı (Şekil 7.11) ile yapılan erişim sonuçlarını inceler. Dosya erişim süreleri ölçülerek elde edilen değerler, okuma (r+) Tablo 7.14'de, yazma (w+) Tablo 7.15'de ve ekleme (a+) Tablo 7.16'da sunulmuştur.

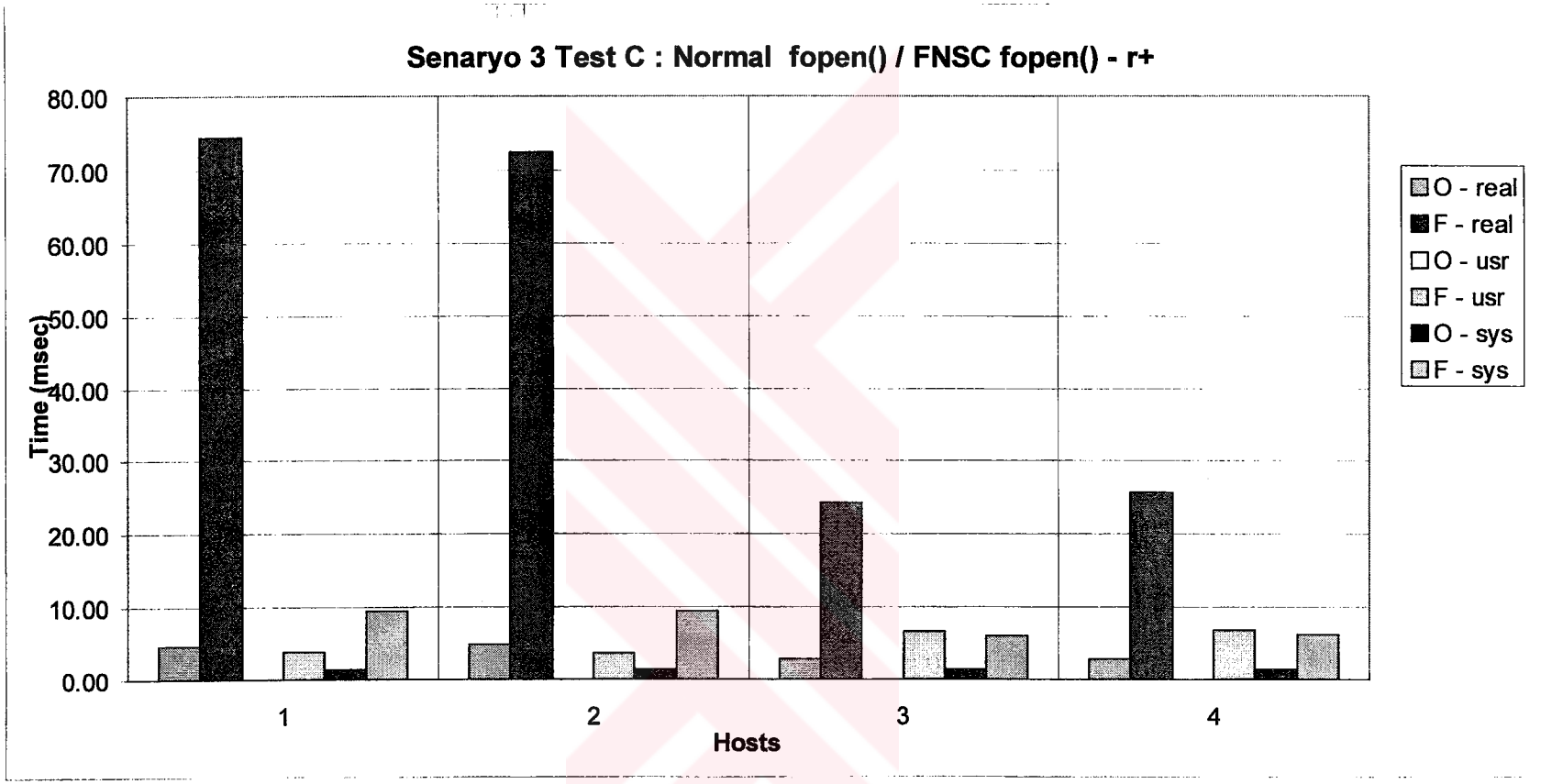


Şekil 7.10 Senaryo 3 Test C : NFS üzerinden normal fopen() çağrısı kullanılması (eş zamanlı 4 istemci)



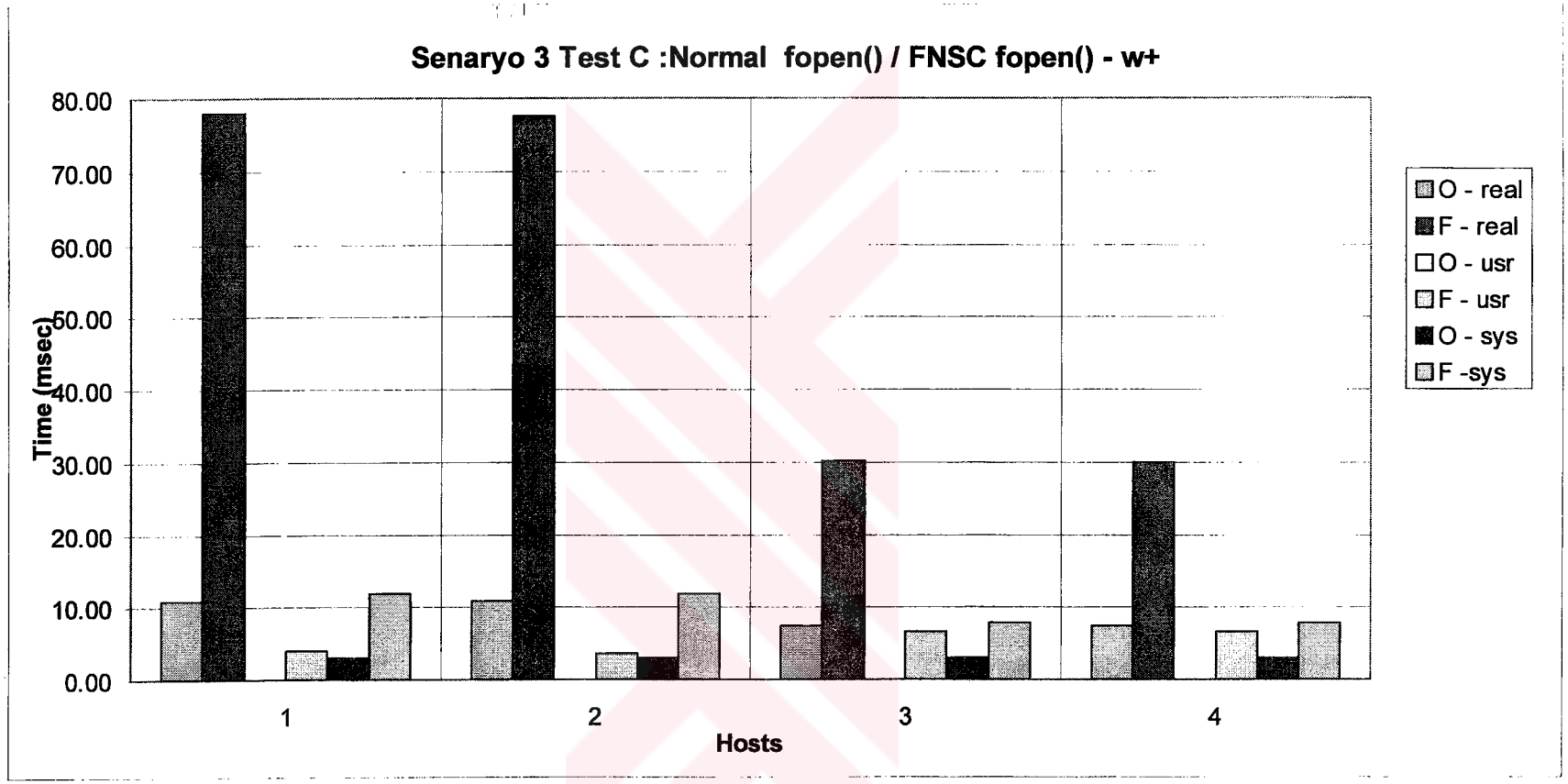
Şekil 7.11 Senaryo 3 Test C : NFS üzerinden FNSC-fopen() çağrısı kullanılması (eş zamanlı 4 istemcinin sunucu üzerinden yaptıkları sorgulama ile erişimleri)

Tablo 7.14 Senaryo 3 Test C : Normal fopen() / FNSC-fopen() (r+) sonuçları



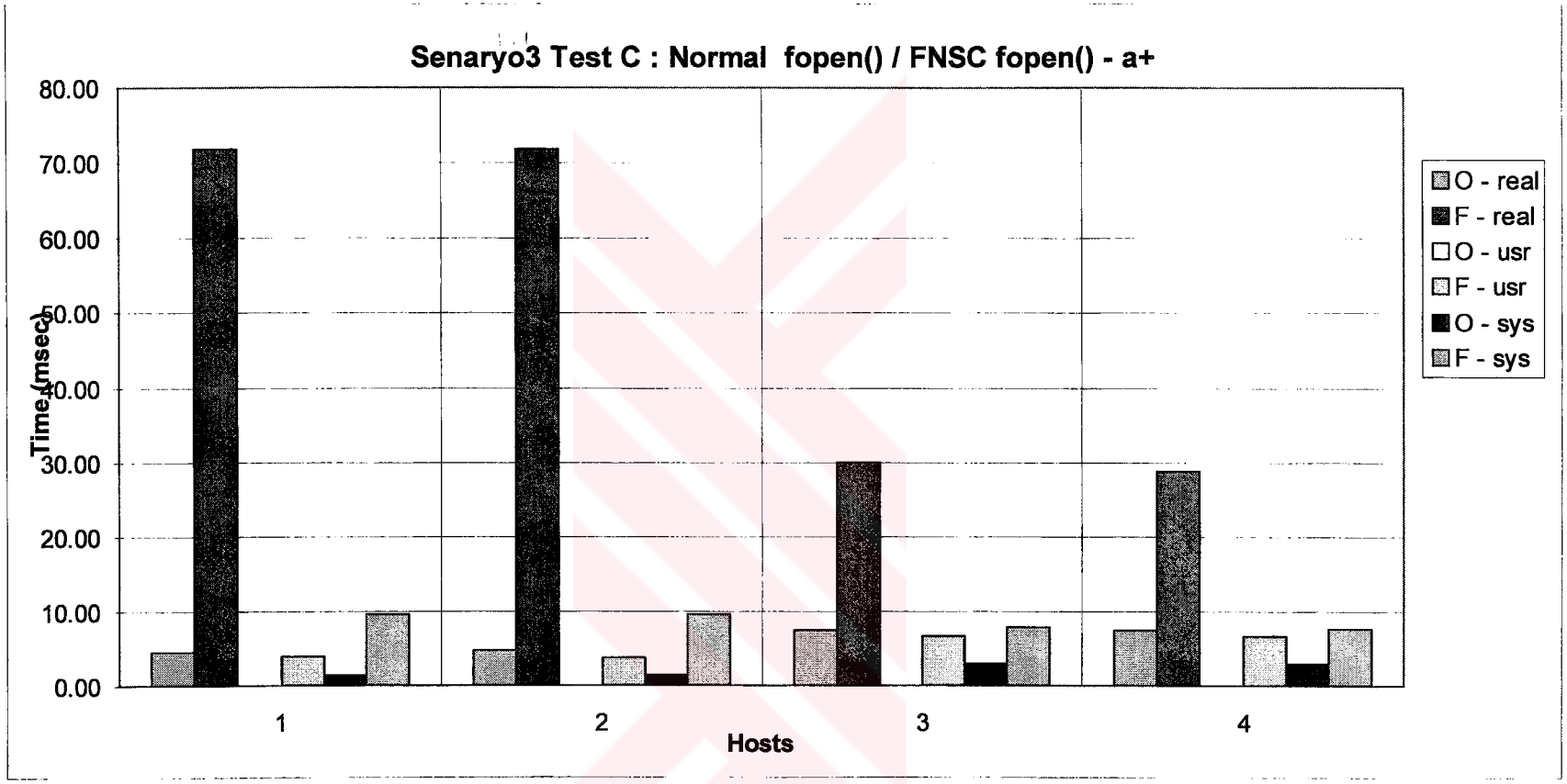
(r+) ortalama	Normal fopen()					FNSC fopen()			
	queen	queen1	tweety	tweety1		queen	queen1	tweety	tweety1
O - real	4.62	4.72	2.88	2.86	F- real	74.40	72.41	24.31	25.57
O - usr	0.07	0.07	0.03	0.02	F - usr	3.88	3.67	6.70	6.79
O - sys	1.41	1.44	1.37	1.36	F - sys	9.45	9.43	5.99	6.29

Tablo 7.15 Senaryo 3 Test C : Normal fopen() / FNSC-fopen() (w+) sonuçları



(w+) ortalama	Normal fopen()					FNSC fopen()			
	queen	queen1	tweety	tweety1		queen	queen1	tweety	tweety1
O - real	10.87	10.84	7.43	7.43	F- real	77.99	77.67	30.17	29.91
O - usr	0.07	0.07	0.04	0.04	F - usr	3.96	3.70	6.64	6.69
O - sys	2.99	2.98	2.97	3.07	F - sys	11.84	11.85	7.80	7.85

Tablo 7.16 Senaryo 3 Test C : Normal fopen() / FNSC-fopen() (a+) sonuçları



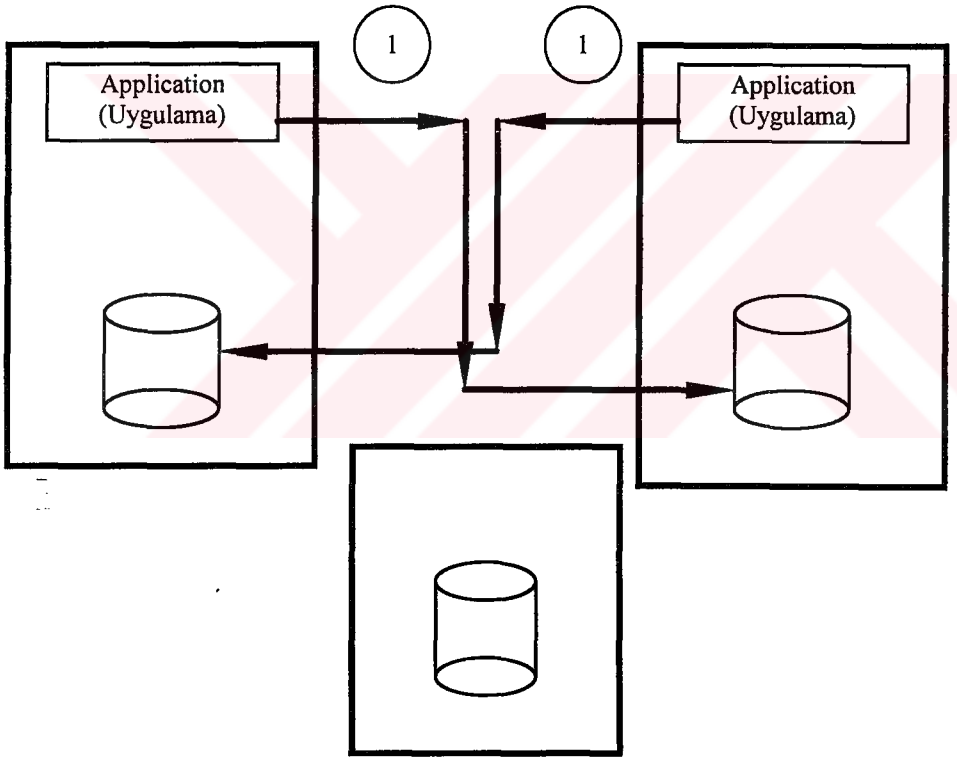
(a+) ortalama	Normal fopen()					FNSC fopen()			
	queen	queen1	tweety	tweety1		queen	queen1	tweety	tweety1
O - real	4.56	4.62	7.49	7.52	F- real	71.69	71.85	29.86	28.83
O - usr	0.07	0.07	0.05	0.04	F - usr	3.86	3.64	6.67	6.56
O - sys	1.41	1.42	3.02	3.02	F - sys	9.56	9.52	7.91	7.55

7.3.2.4 Senaryo 4

Bu senaryo, Senaryo 3 den farklı olarak FNS ve FNSA modüllerinin yarattıkları yüklerin farklı bilgisayarlara dağıtıldığı durumu incelemektedir. Senaryoda sunucu (FNS) modüllü *casper* isimli bilgisayarda çalışırken, ajan (FNSA) ve istemci* (FNSC) modülleri *queen* ve *tweety* isimli bilgisayarlarda çalıştırılmıştır. Senaryo iki ayrı testi içermektedir;

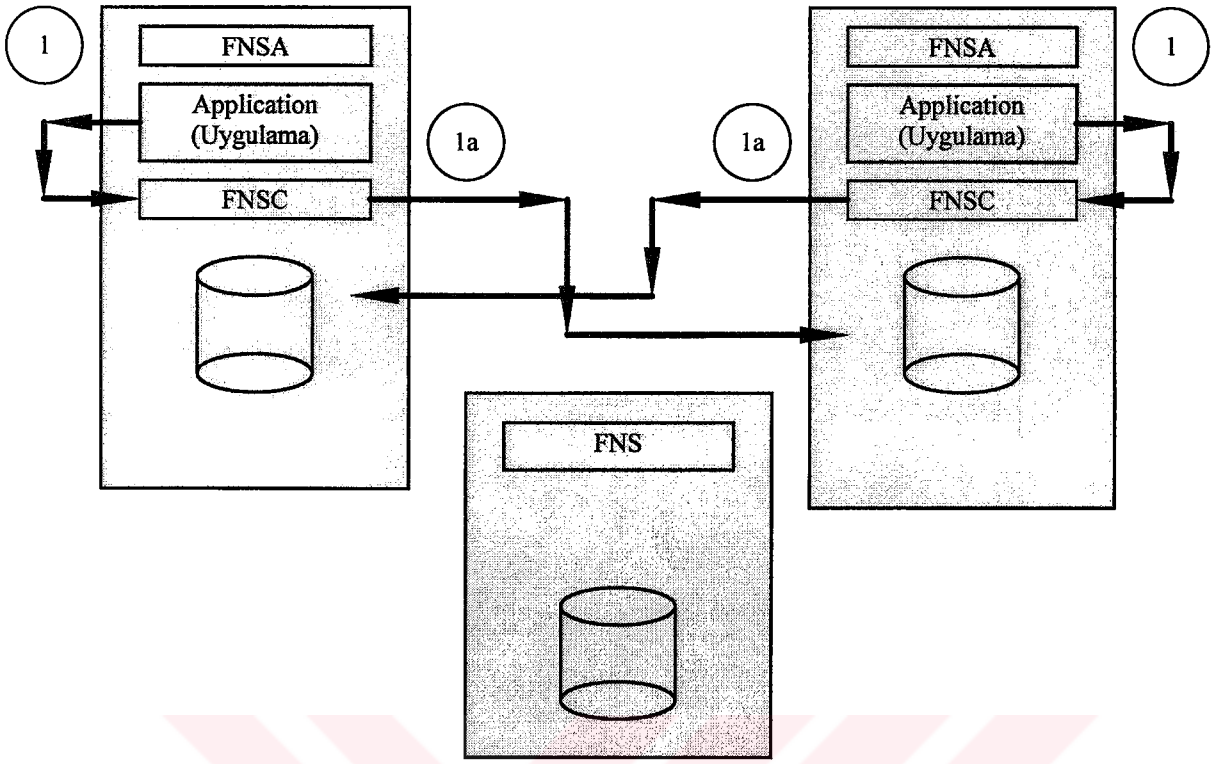
7.3.2.4.1 Senaryo 4 Test A

queen ve *tweety* isimli bilgisayarların aynı anda birer istemci ile (2x1 istemci), birbirlerine ait dosyalara NFS üzerinden normal fopen() çağrısı (Şekil 7.12) ile FNSC-fopen() çağrısı (Şekil 7.13) ile yapılan erişim sonuçlarını inceler. Dosya erişim süreleri ölçülerek elde edilen değerler, okuma (r+) Tablo 7.17'de, yazma (w+) Tablo 7.18'de ve ekleme (a+) Tablo 7.19'da sunulmuştur



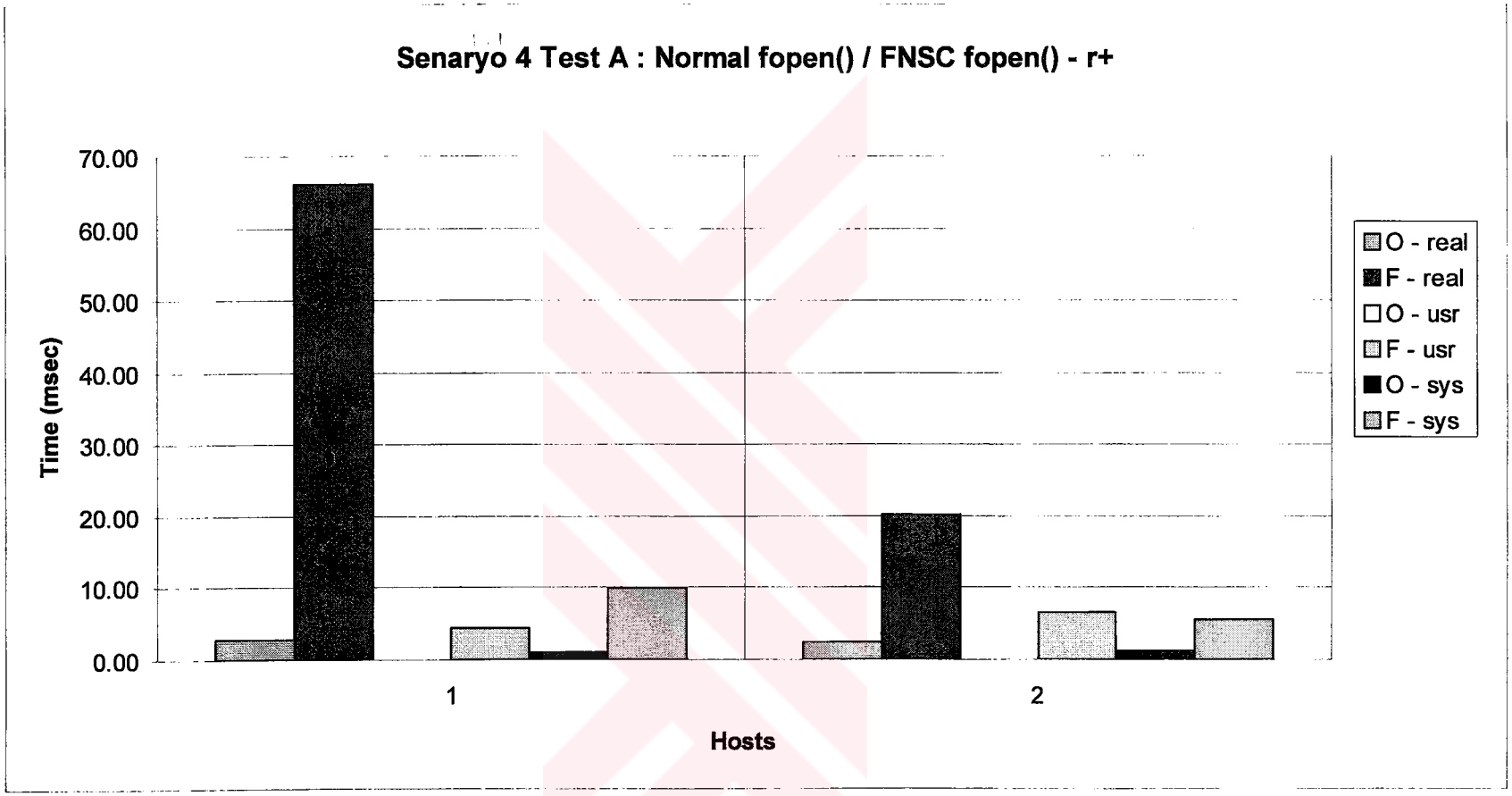
Şekil 7.12 Senaryo 4 Test A : NFS üzerinden normal fopen() çağrısı kullanılması (eş zamanlı 2 istemci))

* FNSC modülünde kullanılan fopen() çağrısı, diğer bir sisteme erişmesi sırasında Internet isimlerinin adreslere dönüştürülmesini sağlayan DNS (Domain Name Server/Service) den yararlanmaktadır. Bu nedenle kullanıcıya yansıyan süreler isim-adres dönüşümü için harcanan zamanı da içermektedir.



Şekil 7.13 Senaryo 4 Test A : NFS üzerinden FNSC-fopen() çağrısı kullanılması (eş zamanlı 2 istemcinin sunucu üzerinden yaptıkları sorgulama ile erişimleri)

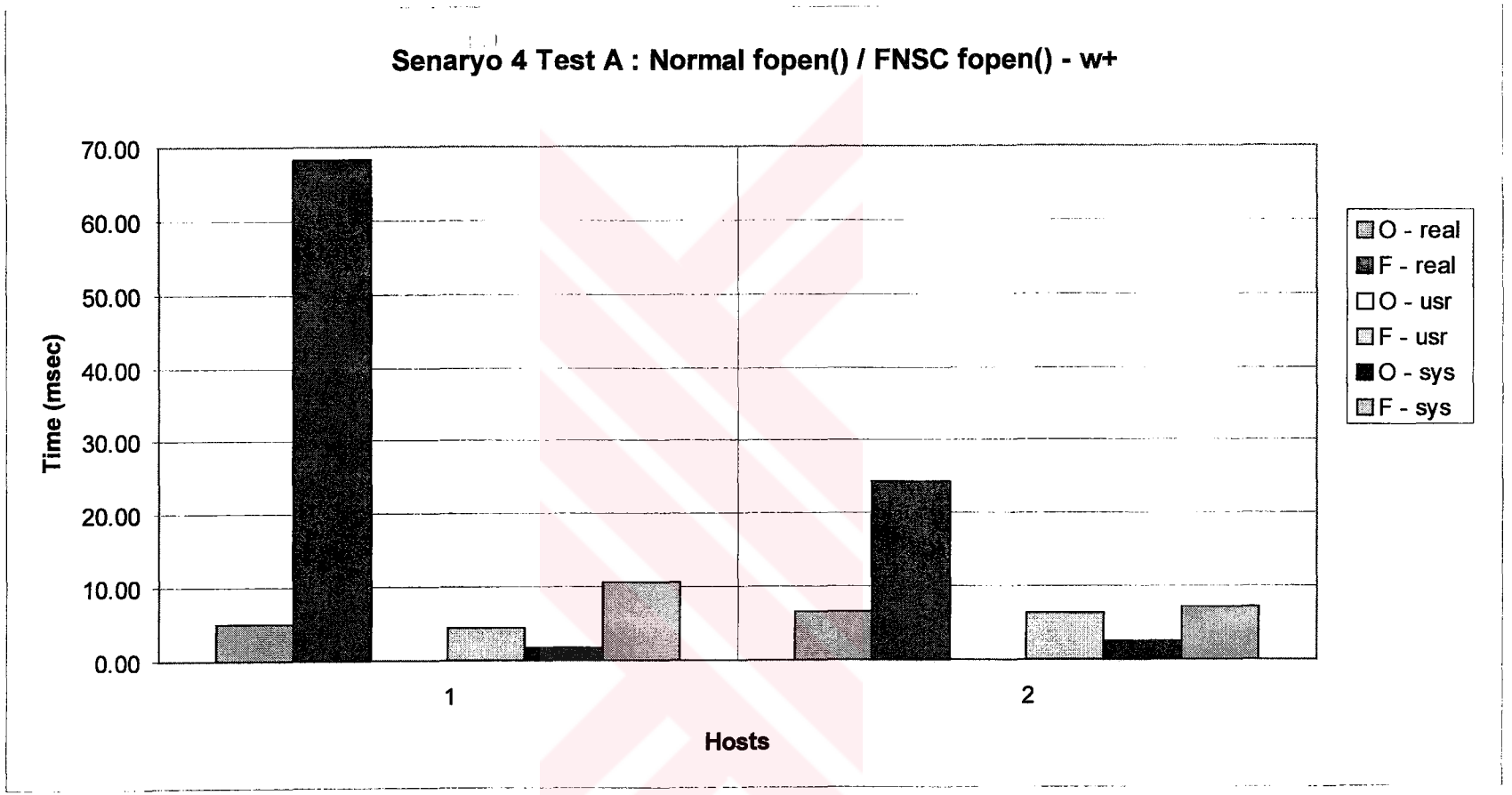
Tablo 7.17 Senaryo 4 Test A : Normal fopen() / FNSC-fopen() (r+) sonuçları



(r+) ortalama

	Normal fopen() queen	Normal fopen() tweety		FNSC fopen() queen	FNSC fopen() tweety
O - real	**2.73**	**2.46**	**F- real**	**66.06**	**20.15**
O - usr	**0.06**	**0.02**	**F - usr**	**4.39**	**6.53**
O - sys	**1.02**	**1.22**	**F - sys**	**9.98**	**5.63**

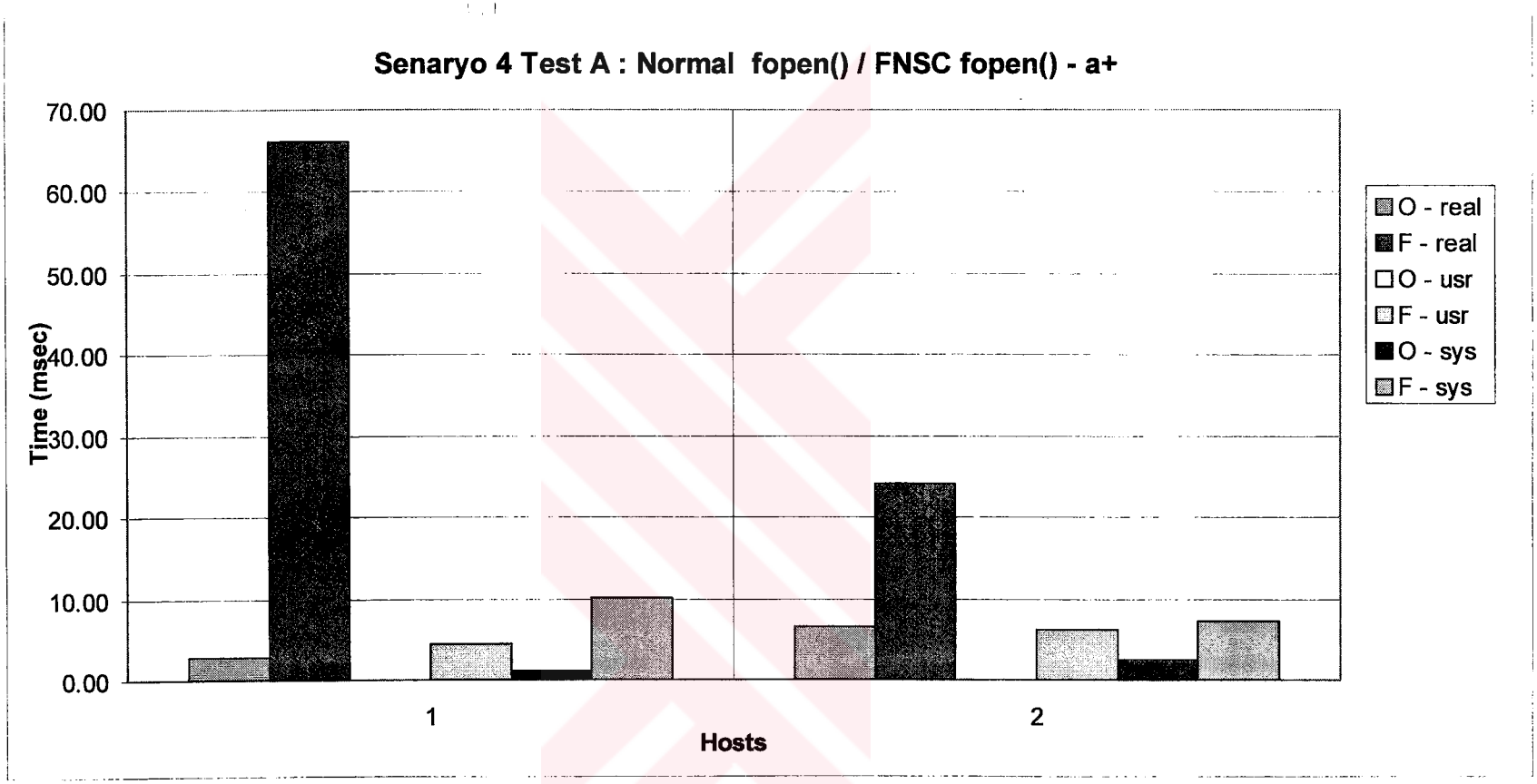
Tablo 7.18 Senaryo 4 Test A : Normal fopen() / FNSC-fopen() (w+) sonuçları



(w+) ortalama	Normal fopen()	
	queen	tweety
O - real	4.96	6.50
O - usr	0.06	0.03
O - sys	1.81	2.49

	FNSC fopen()	
	queen	tweety
F- real	68.28	24.26
F - usr	4.36	6.28
F - sys	10.67	7.20

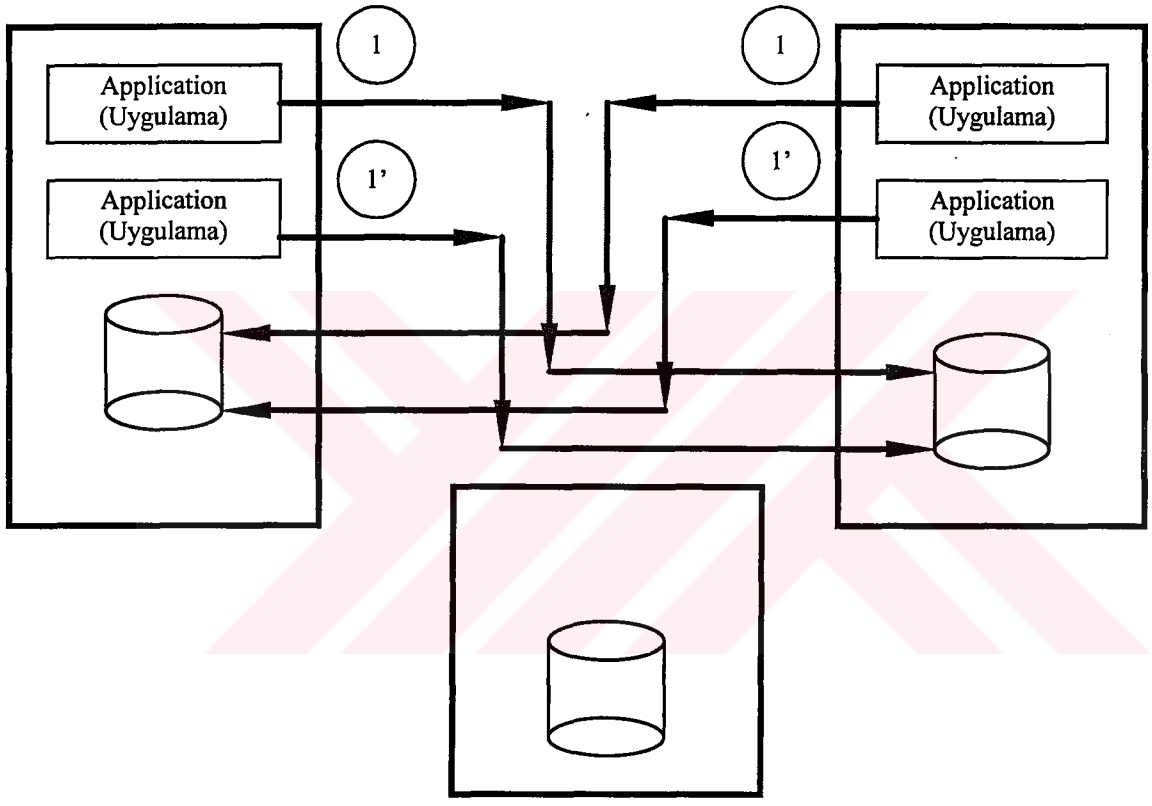
Tablo 7.19 Senaryo 4 Test A : Normal fopen() / FNSC-fopen() (a+) sonuçları



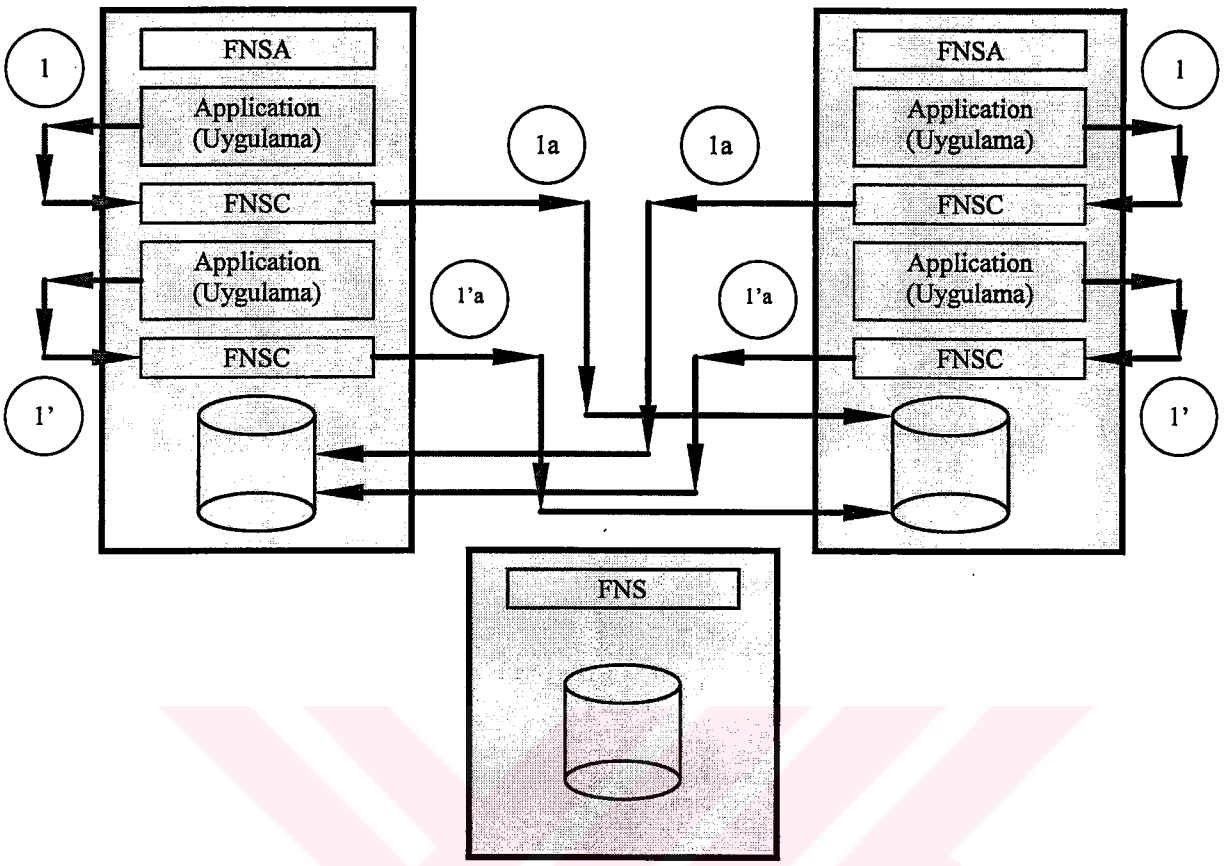
(a+) ortalama	Normal fopen() queen	Normal fopen() tweety		FNSC fopen() queen	FNSC fopen() tweety
O - real	**2.76**	**6.48**	**F- real**	**66.07**	**24.07**
O - usr	**0.05**	**0.04**	**F - usr**	**4.37**	**6.25**
O - sys	**1.00**	**2.56**	**F - sys**	**10.03**	**7.29**

7.3.2.4.2 Senaryo 4 Test B

queen ve *tweety* isimli bilgisayarların aynı anda birbirlerine ait dosyalara ikişer istemci ile (2x2 istemci) NFS üzerinden normal fopen() çağrısı (Şekil 7.14) ile FNSC-fopen() çağrısı (Şekil 7.15) ile yapılan erişim sonuçlarını inceler. Dosya erişim süreleri ölçülerek elde edilen değerler, okuma (r+) Tablo 7.20'de, yazma (w+) Tablo 7.21'de ve ekleme (a+) Tablo 7.22'de sunulmuştur

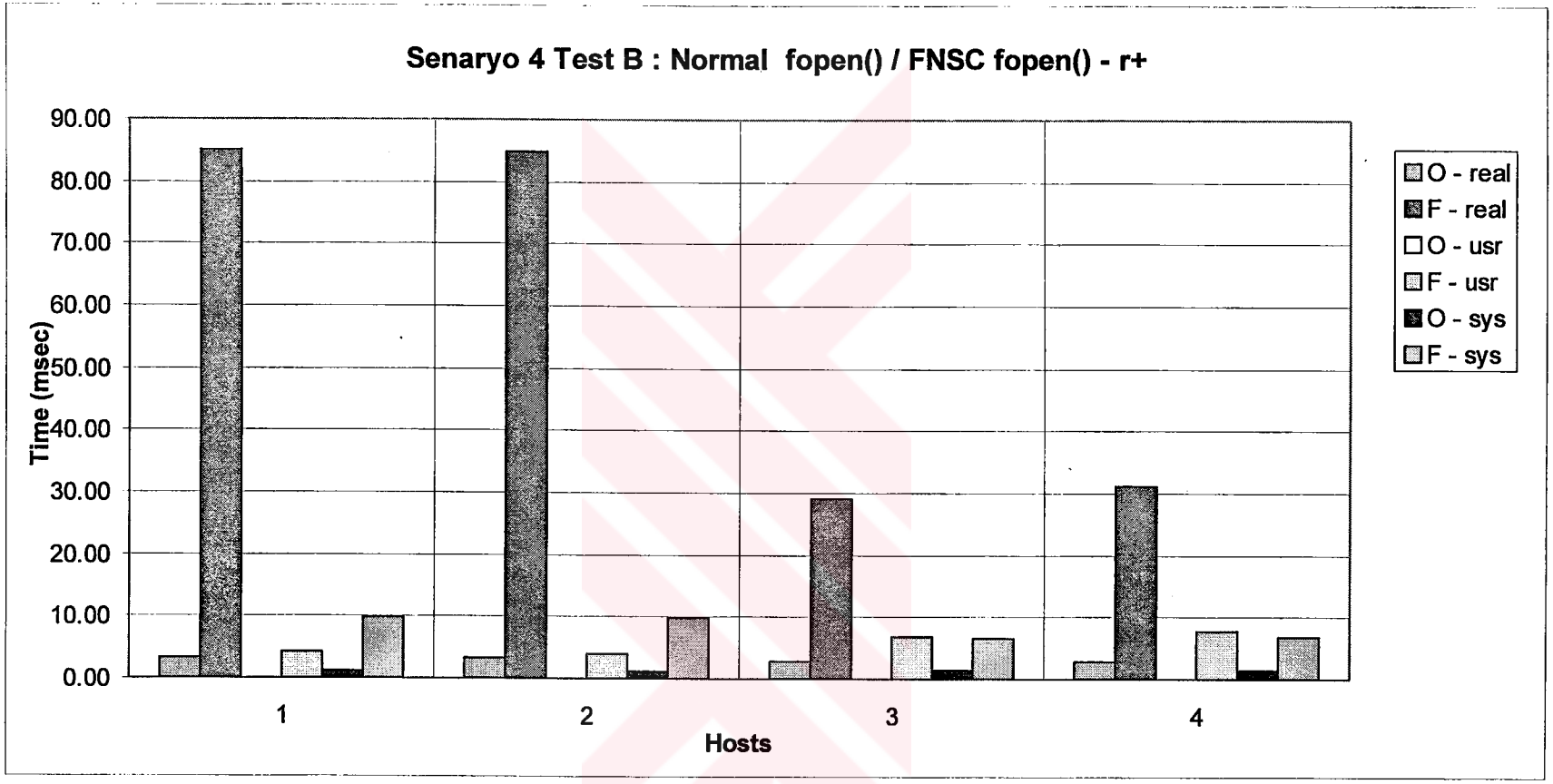


Şekil 7.14 Senaryo 4 Test B : NFS üzerinden normal fopen() çağrısı kullanılması (eş zamanlı 4 istemci)



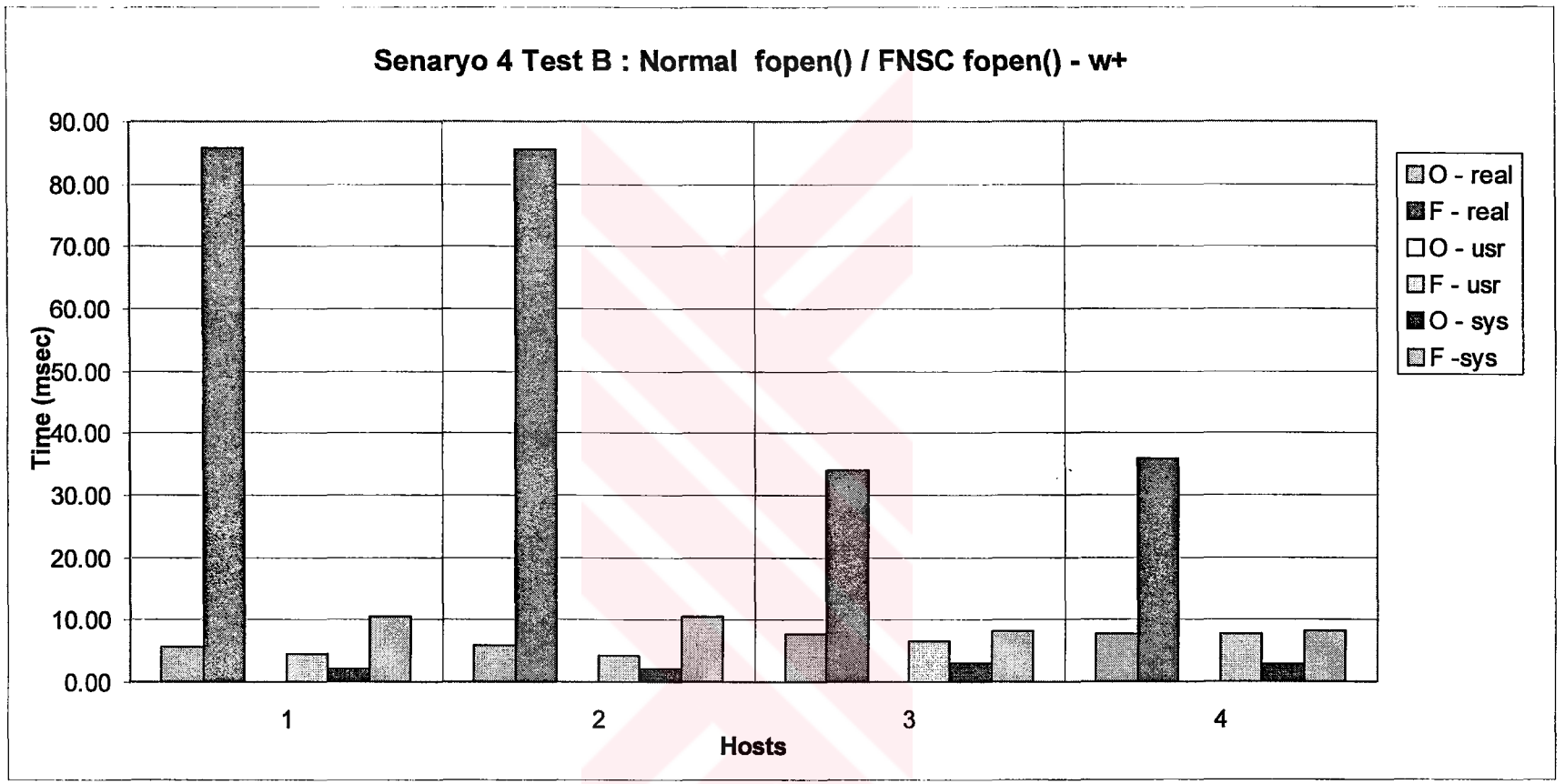
Şekil 7.15 Senaryo 4 Test B : NFS üzerinden FNSC-fopen() çağrısı kullanılması (eş zamanlı 4 istemcinin sunucu üzerinden yaptıkları sorgulama ile erişimleri)

Tablo 7.20 Senaryo 4 Test B : Normal fopen() / FNSC-fopen() (r+) sonuçları

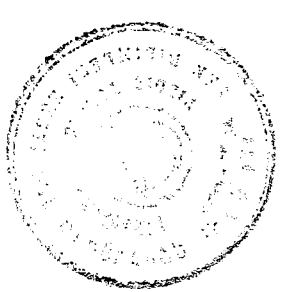


(r+) ortalama	Normal fopen()					FNSC fopen()			
	queen	queen1	tweety	tweety1		queen	queen1	tweety	tweety1
O - real	3.28	3.35	2.93	2.93	F- real	85.10	84.94	28.96	31.14
O - usr	0.07	0.06	0.02	0.02	F - usr	4.24	4.04	6.71	7.66
O - sys	1.28	1.26	1.32	1.32	F - sys	9.77	9.83	6.58	6.72

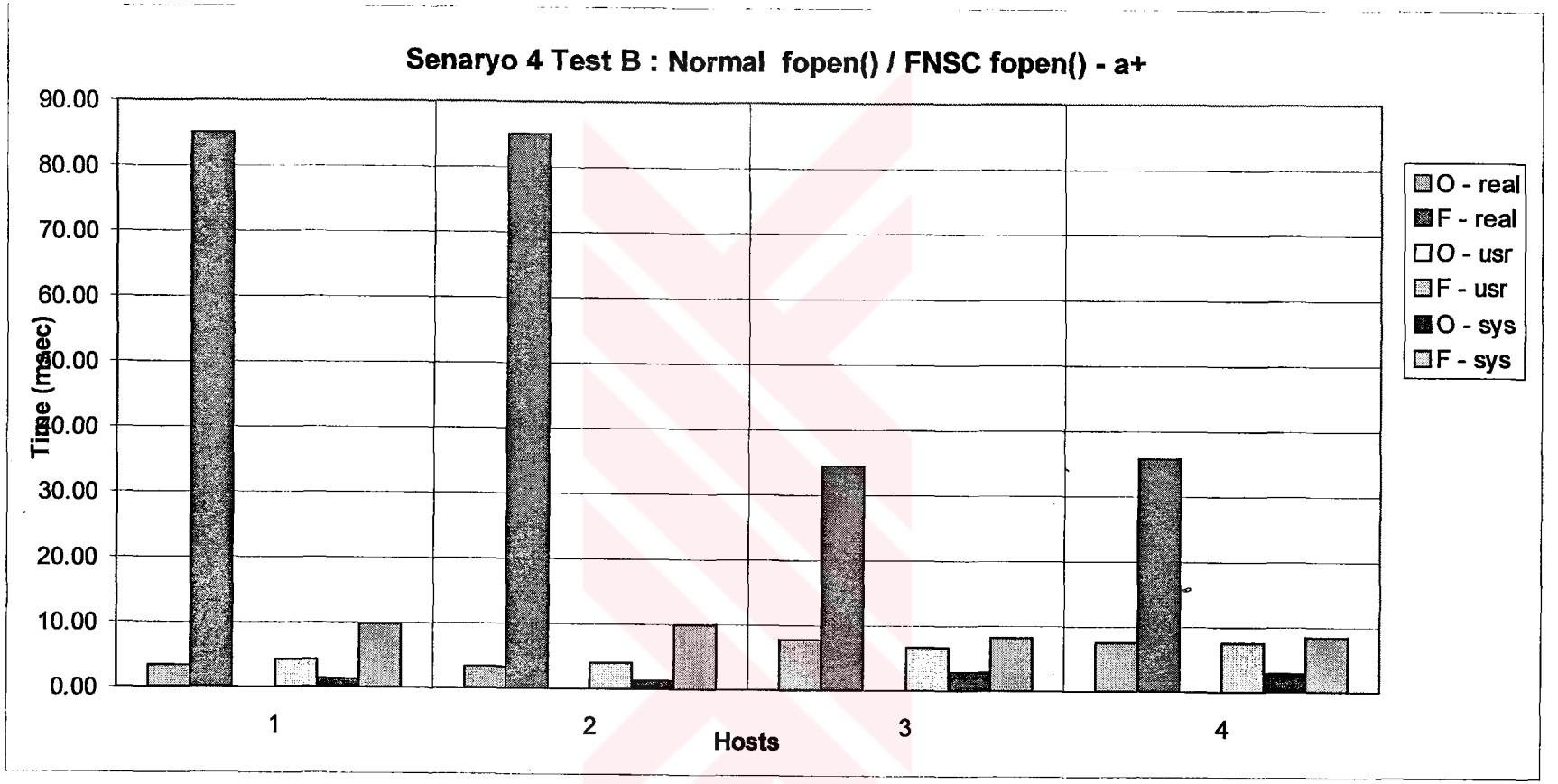
Tablo 7.21 Senaryo 4 Test B : Normal fopen() / FNSC-fopen() (w+) sonuçları



(w+) ortalama	Normal fopen()					FNSC fopen()			
	queen	queen1	tweety	tweety1		queen	queen1	tweety	tweety1
O - real	5.73	5.77	7.70	7.68	F- real	85.78	85.46	34.04	35.80
O - usr	0.06	0.07	0.03	0.04	F - usr	4.37	4.17	6.58	7.67
O - sys	2.07	2.11	2.99	2.92	F - sys	10.50	10.51	8.19	8.29



Tablo 7.22 Senaryo 4 Test B : Normal fopen() / FNSC-fopen() (a+) sonuçları



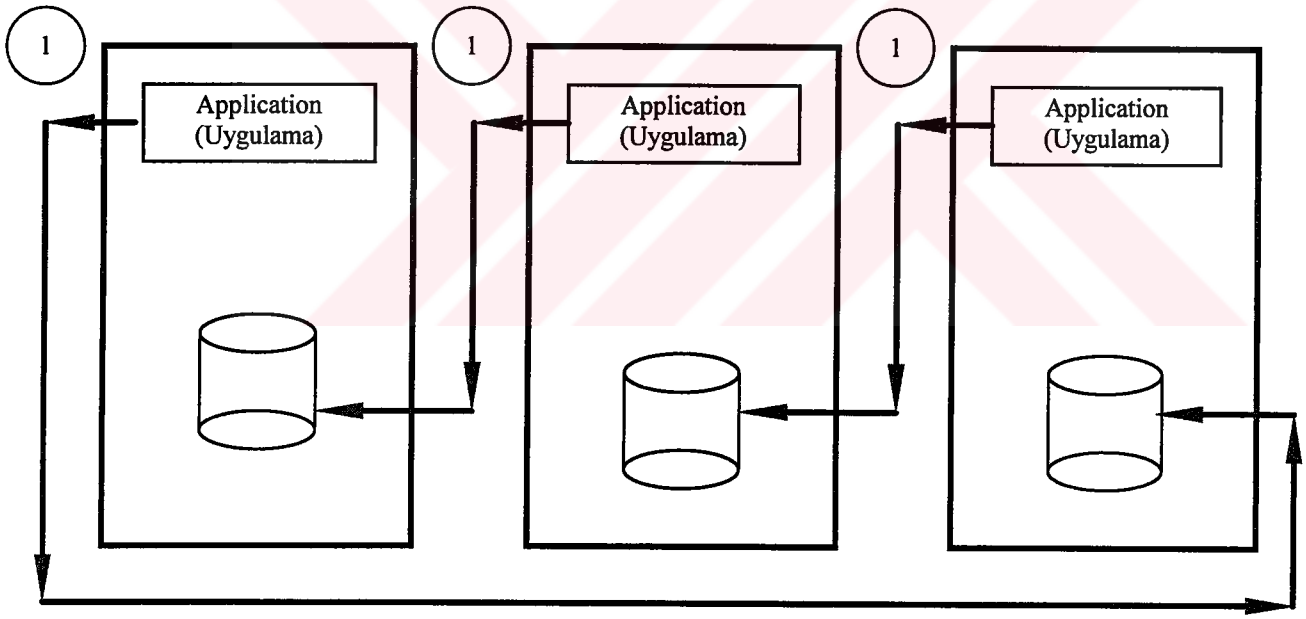
(a+) ortalama	Normal fopen()					FNSC fopen()			
	queen	queen1	tweety	tweety1		queen	queen1	tweety	tweety1
O - real	3.26	3.33	7.67	7.53	F- real	85.07	85.01	34.34	35.76
O - usr	0.06	0.06	0.05	0.05	F - usr	4.23	4.05	6.63	7.66
O - sys	1.25	1.26	2.92	2.95	F - sys	9.88	10.03	8.32	8.40

7.3.2.5 Senaryo 5

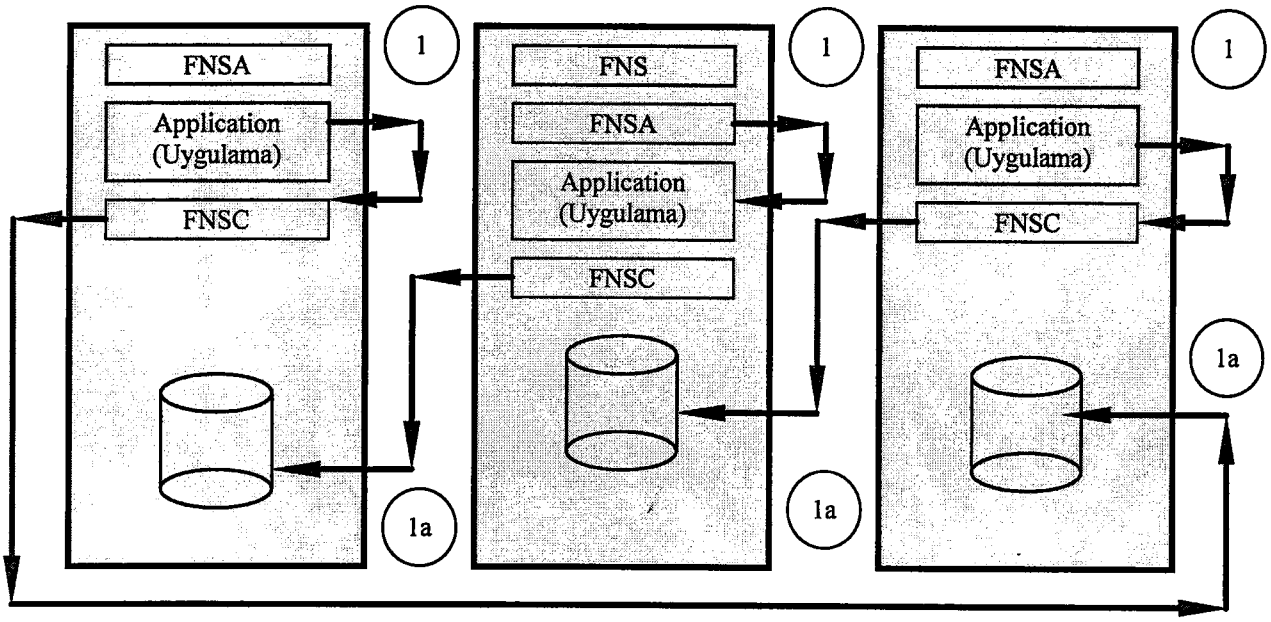
Senaryo 5'de, her bilgisayar üzerinde ajan (FNSA) ve istemci* (FNSC) modüllerine yer verilmiş ve istemci (FNSC) modülleri aynı anda çalıştırılmak suretiyle bir diğer bilgisayar üzerindeki dosyaya erişimleri sağlanmıştır. Senaryoda sunucu (FNS) modüllü *casper* isimli bilgisayarda çalışırken, ajan (FNSA) ve istemci (FNSC) modülleri *queen, tweety* ve *casper* isimli bilgisayarlarda çalışmaktadır. Senaryo iki ayrı testi içermektedir;

7.3.2.5.1 Senaryo 5 Test A

queen , *tweety* ve *casper* isimli bilgisayarların aynı anda birer istemci ile (3x1 istemci), diğerindeki dosyalara NFS üzerinden normal fopen() çağrısı (Şekil 7.16) ile FNSC-fopen() çağrısı (Şekil 7.17) ile yapılan erişim sonuçlarını inceler. Dosya erişim süreleri ölçülerek elde edilen değerler, okuma (r+) Tablo 7.23'de, yazma (w+) Tablo 7.24'de ve ekleme (a+) Tablo 7.25'de sunulmuştur

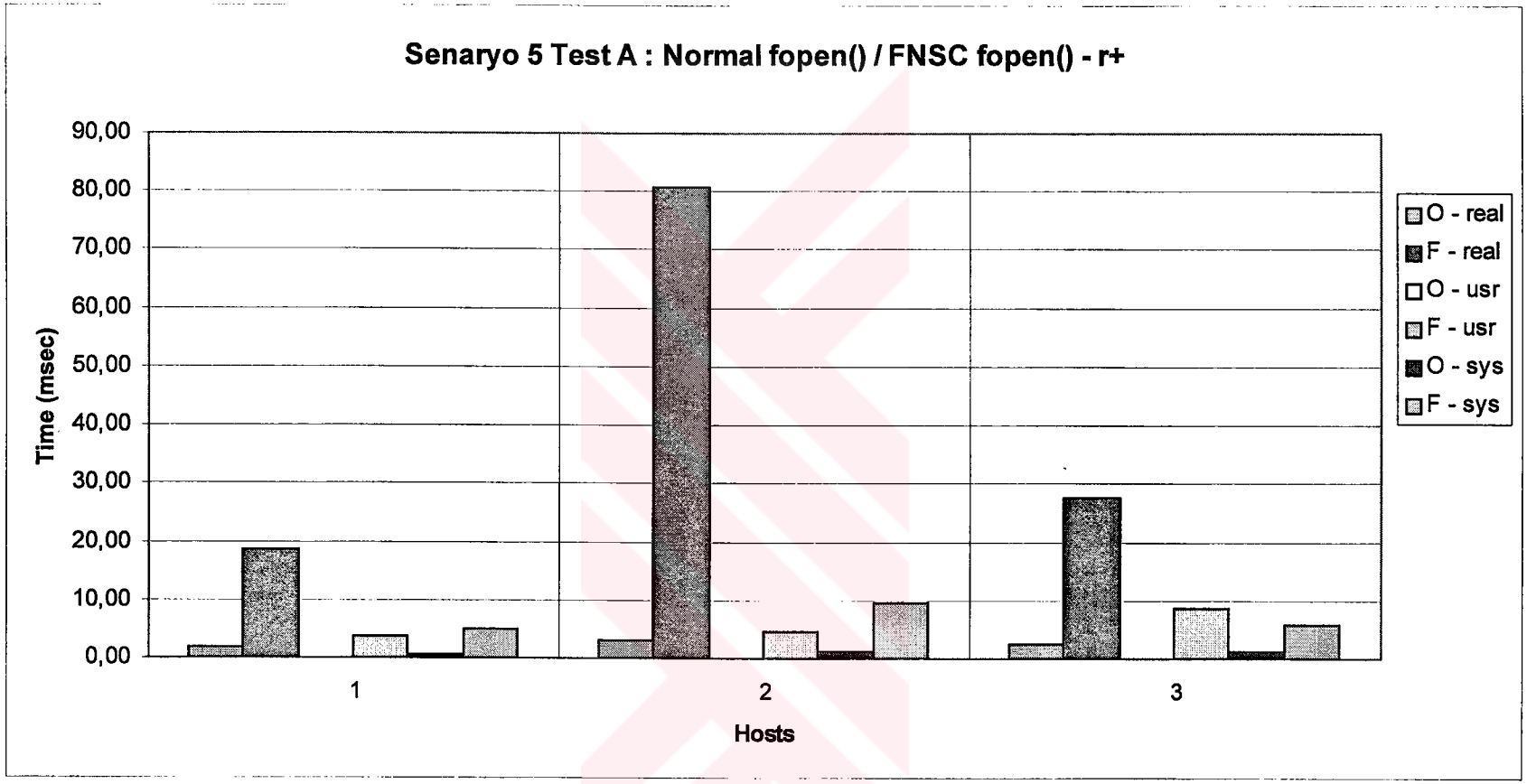


Şekil 7.16 Senaryo 5 Test A : NFS üzerinden normal fopen() çağrısı kullanılması (eş zamanlı 3 istemci)



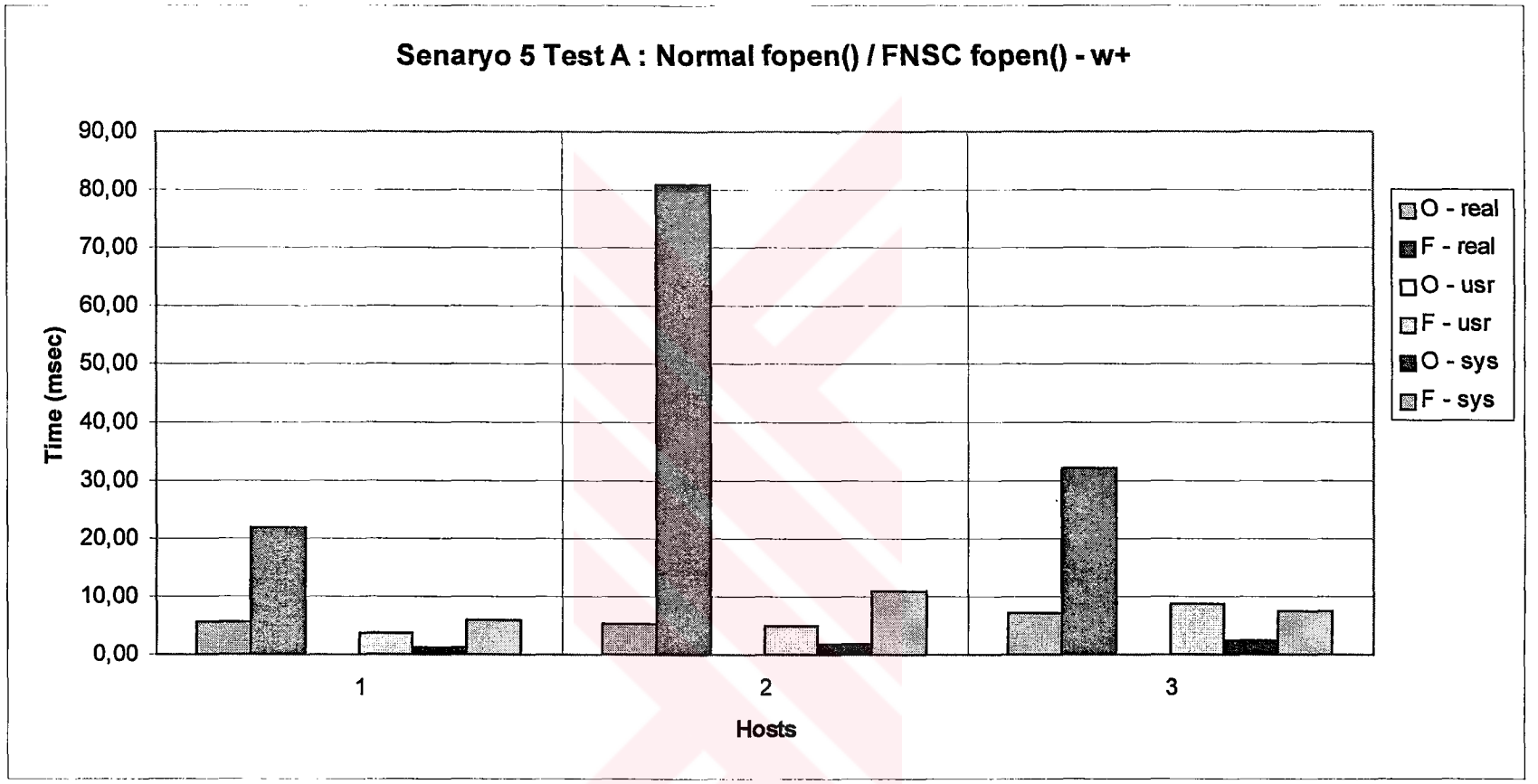
Şekil 7.17 Senaryo 5 Test A : NFS üzerinden FNSC-fopen() çağrısı kullanılması (eş zamanlı 3 istemcinin sunucu üzerinden yaptıkları sorgulama ile erişimleri)

Tablo 7.23 Senaryo 5 Test A : Normal fopen() / FNSC-fopen() (r+) sonuçları



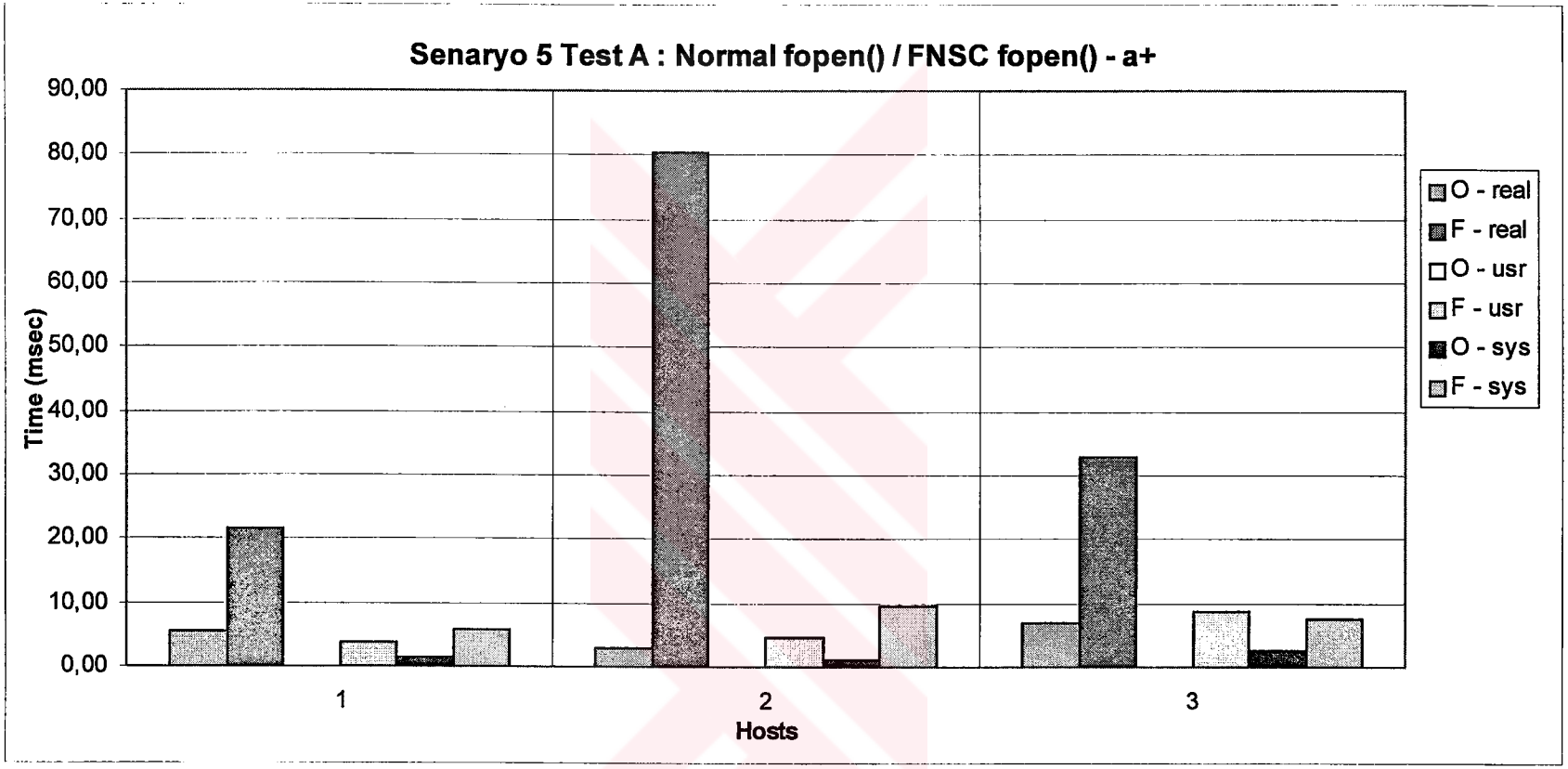
(r+) ortalama	Normal fopen()				FNSC fopen()		
	casper	queen	tweety		casper	queen	tweety
O - real	1,81	3,03	2,60	F- real	18,63	80,54	27,74
O - usr	0,00	0,06	0,02	F - usr	3,69	4,55	8,82
O - sys	0,56	1,09	1,19	F - sys	5,12	9,51	6,02

Tablo 7.24 Senaryo 5 Test A : Normal fopen() / FNSC-fopen() (w+) sonuçları

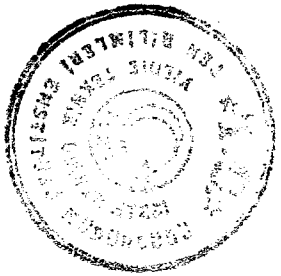


(w+) ortalama	Normal fopen()				FNSC fopen()		
	casper	queen	tweety		casper	queen	tweety
O - real	5,59	5,46	7,06	F- real	21,92	80,90	32,18
O - usr	0,01	0,07	0,01	F - usr	3,69	4,96	8,86
O - sys	1,38	1,84	2,62	F - sys	5,96	11,09	7,51

Tablo 7.25 Senaryo 5 Test A : Normal fopen() / FNSC-fopen() (a+) sonuçları

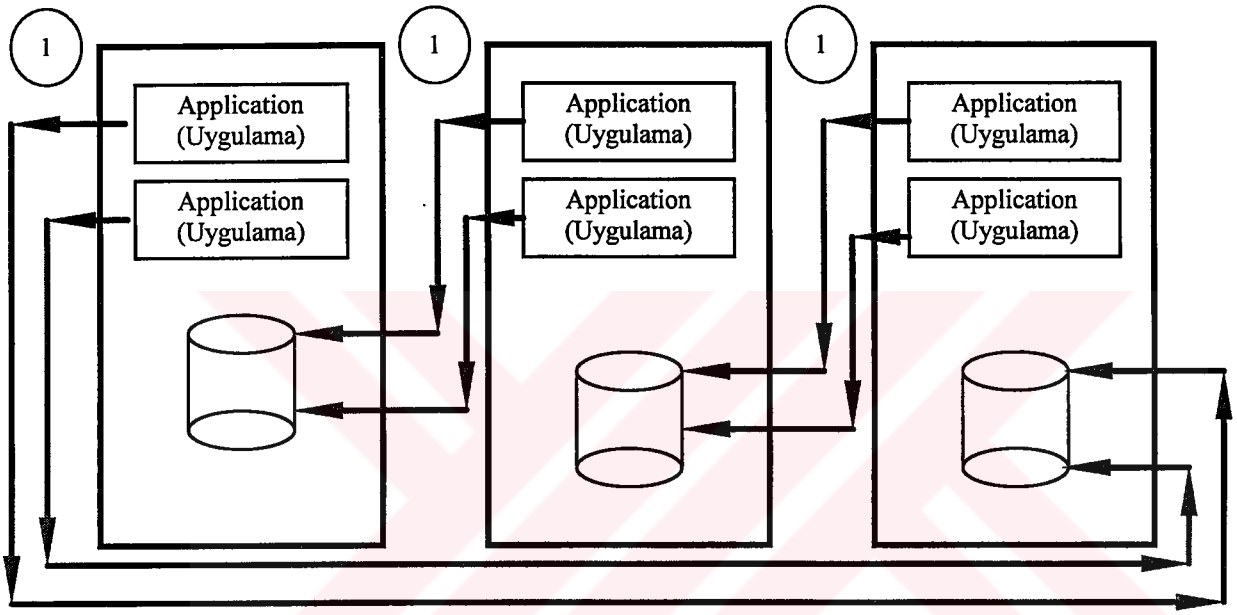


(a+) ortalama	Normal fopen()				FNSC fopen()		
	casper	queen	tweety		casper	queen	tweety
O - real	5,57	3,01	6,91	F- real	21,62	80,35	32,81
O - usr	0,00	0,06	0,04	F - usr	3,75	4,63	8,84
O - sys	1,40	1,09	2,66	F - sys	5,92	9,61	7,59

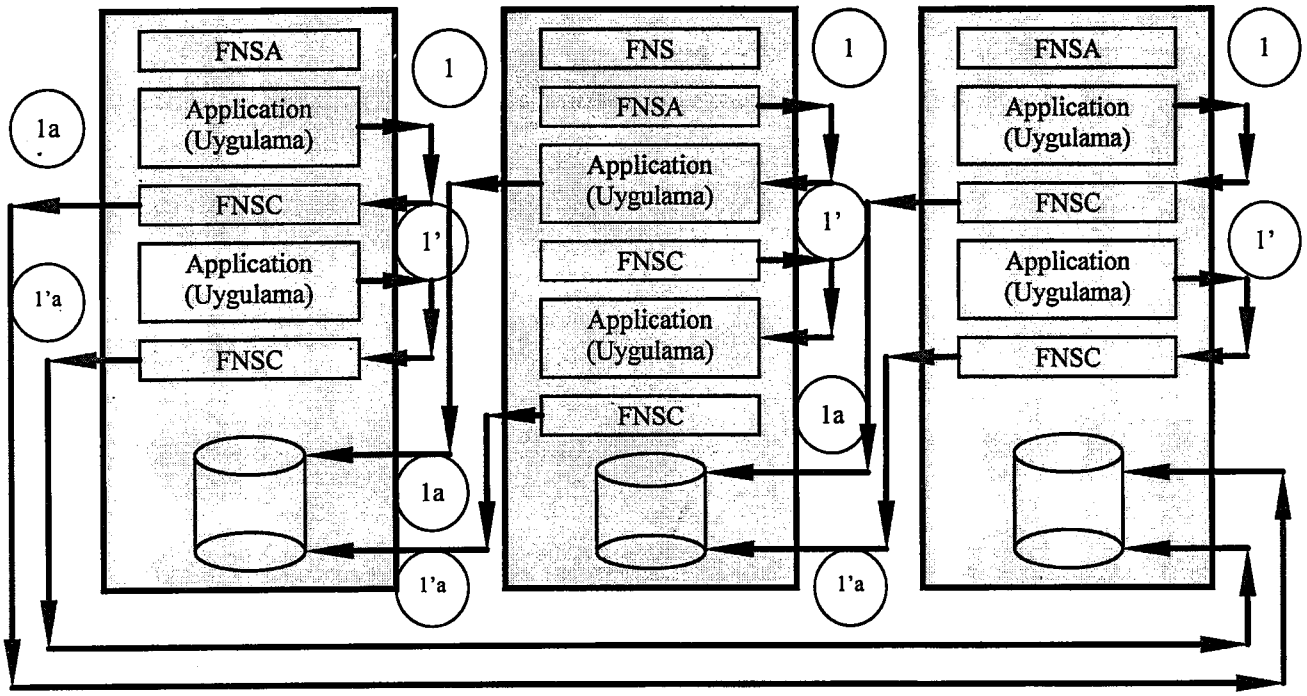


7.3.2.5.2 Senaryo 5 Test B

queen , *tweety* ve *casper* isimli bilgisayarların aynı anda bir diğerindeki dosyalara, ikişer istemciyle (3x2 istemci) ile NFS üzerinden normal fopen() çağrısı (Şekil 7.18) ile FNSC-fopen() çağrısı (Şekil 7.19) ile yapılan erişim sonuçlarını inceler. Dosya erişim süreleri ölçülerek elde edilen değerler, okuma (r+) Tablo 7.26'da, yazma (w+) Tablo 7.27'de ve ekleme (a+) Tablo 7.28'de sunulmuştur

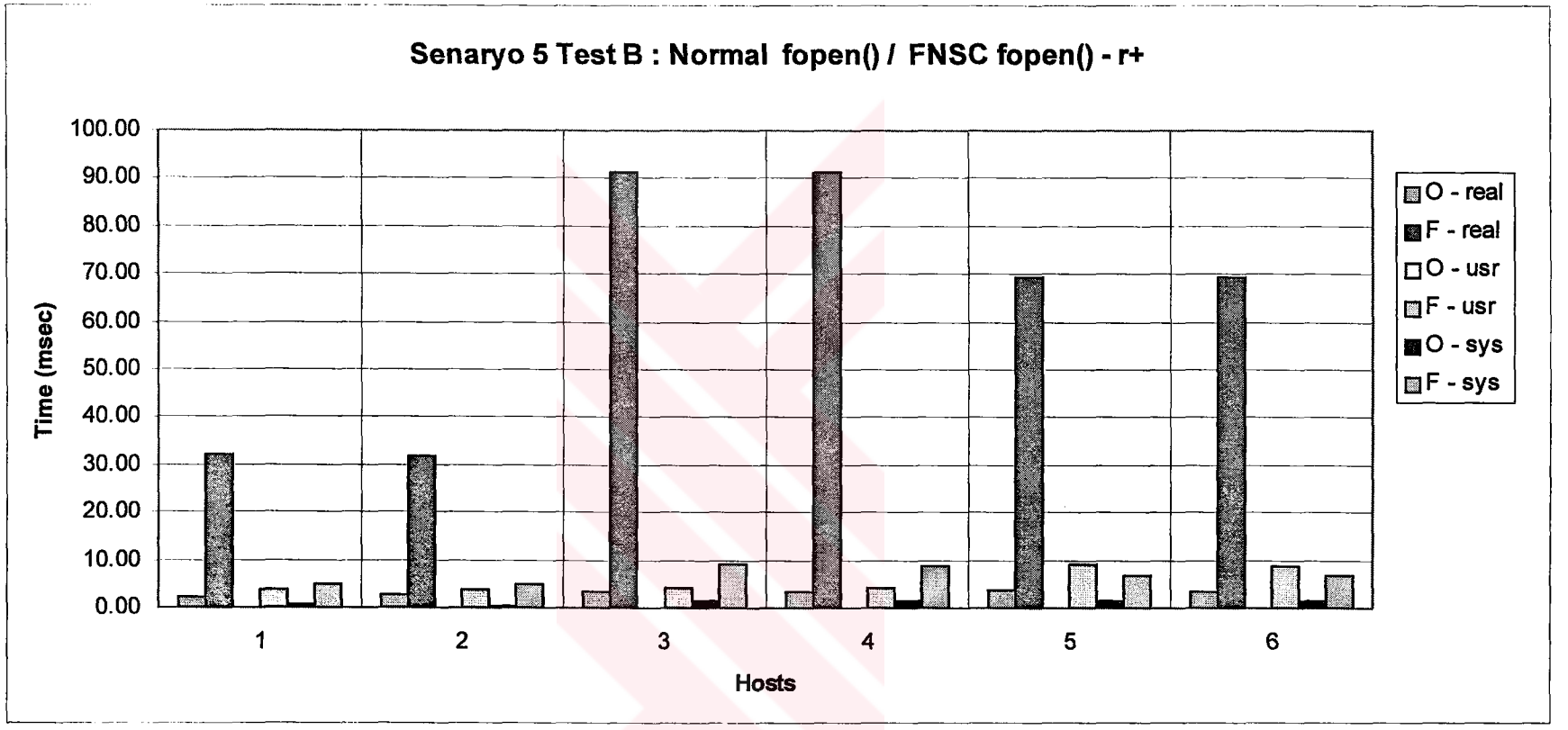


Şekil 7.18 Senaryo 5 Test B : NFS üzerinden normal fopen() çağrısı kullanılması (eş zamanlı 6 istemci)

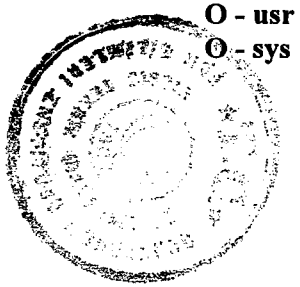


Şekil 7.19 Senaryo 5 Test B : NFS üzerinden FNSC-fopen() çağrısı kullanılması (eş zamanlı 6 istemcinin sunucu üzerinden yaptıkları sorgulama ile erişimleri)

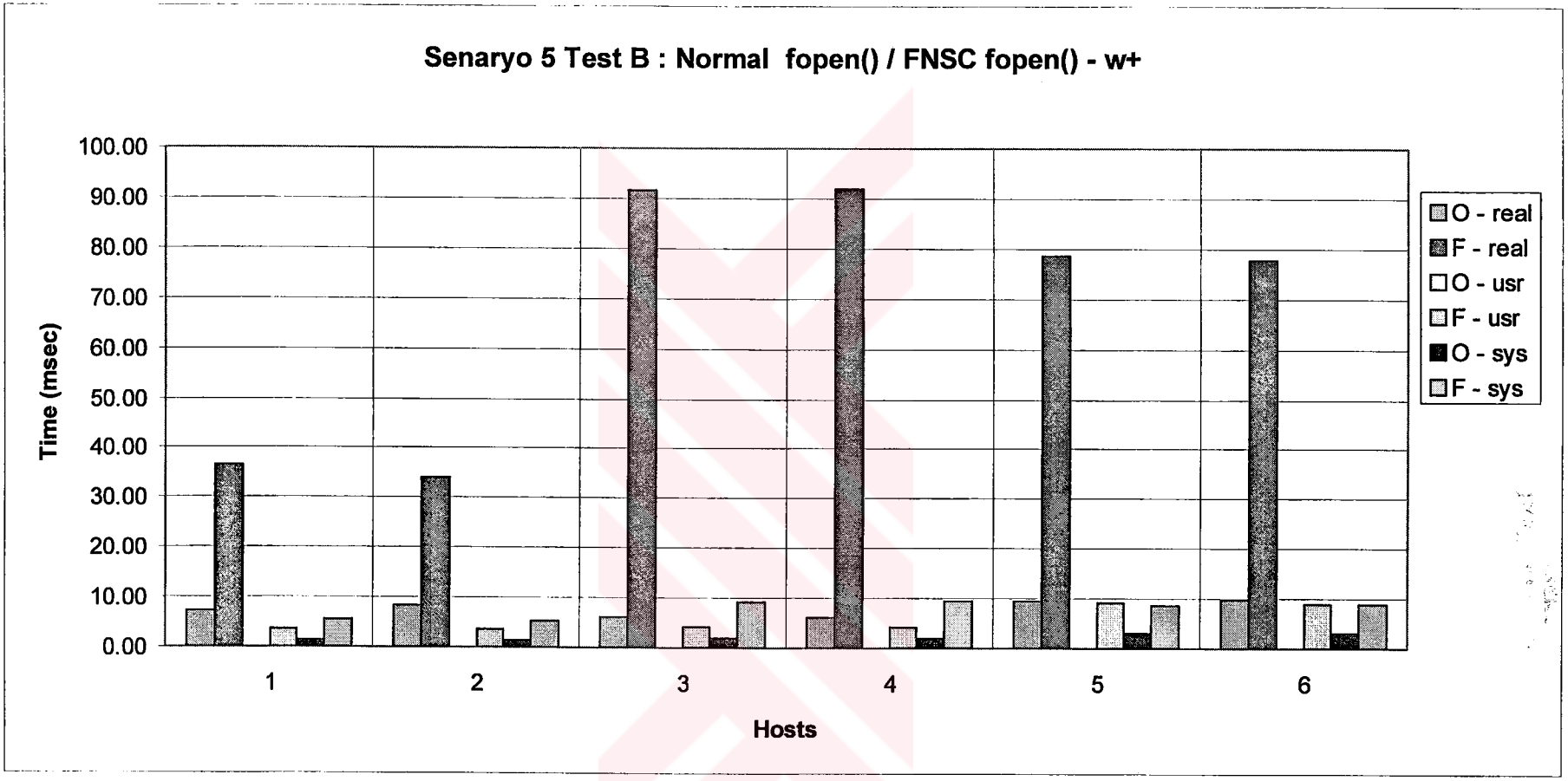
Tablo 7.26 Senaryo 5 Test B : Normal fopen() / FNSC-fopen() (r+) sonuçları



(r+)	ortalama		Normal fopen()					FNSC fopen()					
	casper	casper1	queen	queen1	tweety	tweety1		casper	casper1	queen	queen1	tweety	tweety1
O - real	2.19	2.62	3.53	3.40	3.63	3.23	F- real	32.12	31.75	91.14	91.15	69.45	69.22
O - usr	0.00	0.00	0.06	0.06	0.03	0.03	F - usr	3.71	3.75	4.16	4.31	9.09	8.73
O - sys	0.58	0.55	1.36	1.35	1.34	1.33	F - sys	4.76	4.94	8.91	8.88	6.97	6.81

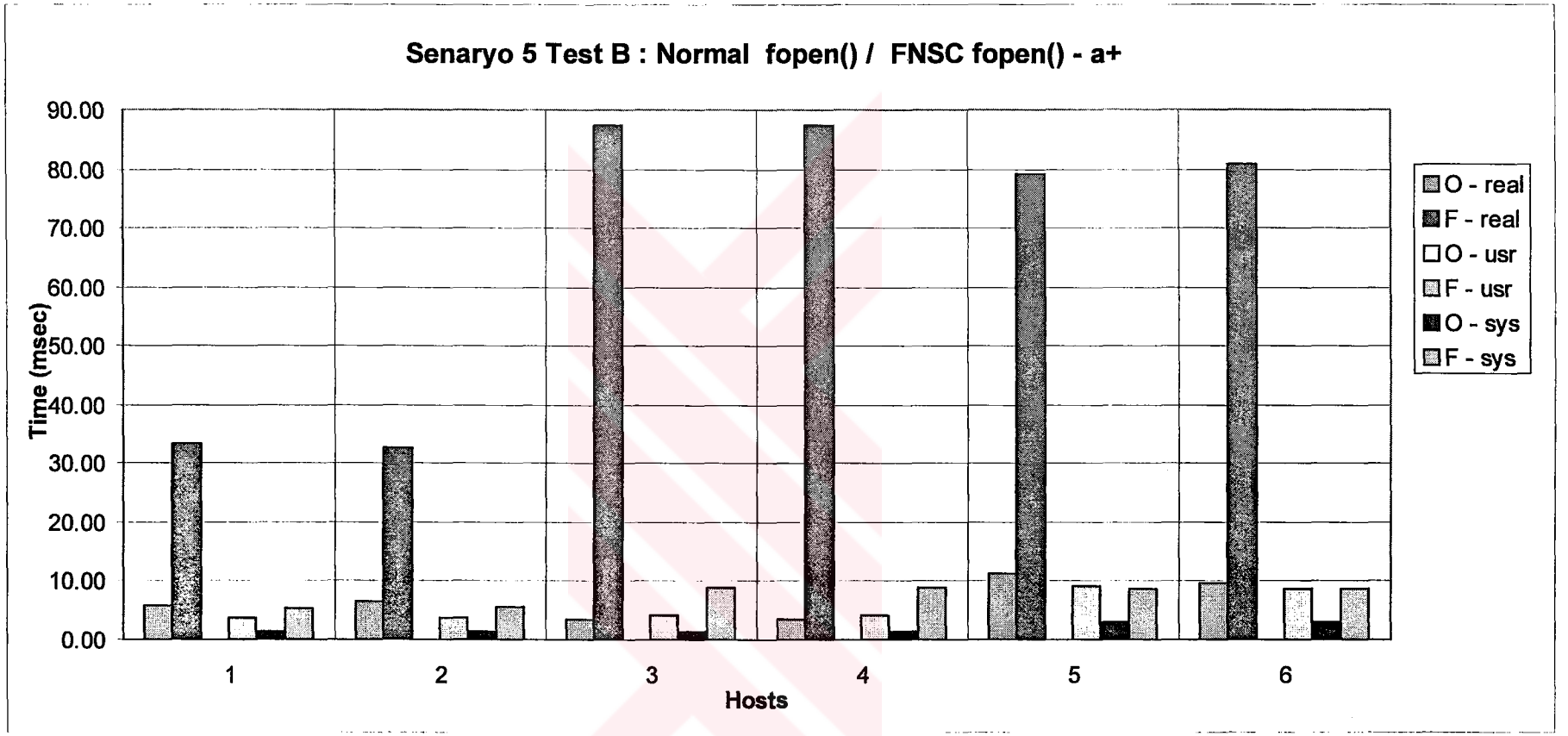


Tablo 7.27 Senaryo 5 Test B : Normal fopen() / FNSC-fopen() (w+) sonuçları



(w+)	ortalama		Normal fopen()					FNSC fopen()					
	casper	casper1	queen	queen1	tweety	tweety1		casper	casper1	queen	queen1	tweety	tweety1
O - real	7.24	8.44	6.14	6.06	9.55	9.65	F- real	36.38	33.90	91.60	91.95	78.63	77.83
O - usr	0.00	0.00	0.07	0.07	0.06	0.04	F - usr	3.69	3.62	4.14	4.13	9.08	8.98
O - sys	1.40	1.40	1.96	1.98	3.03	3.11	F - sys	5.49	5.33	9.33	9.37	8.74	8.80

Tablo 7.28 Senaryo 5 Test B : Normal fopen() / FNSC-fopen() (a+) sonuçları



(a+)	ortalama		Normal fopen()						FNSC fopen()					
	casper	casper1	queen	queen1	tweety	tweety1		casper	casper1	queen	queen1	tweety	tweety1
O - real	5.83	6.55	3.62	3.52	11.19	9.53	F- real	33.42	32.59	87.41	87.30	79.09	80.77
O - usr	0.00	0.01	0.07	0.07	0.06	0.06	F - usr	3.66	3.74	4.21	4.19	9.08	8.79
O - sys	1.37	1.31	1.36	1.36	3.03	3.09	F - sys	5.51	5.67	8.95	8.87	8.77	8.67

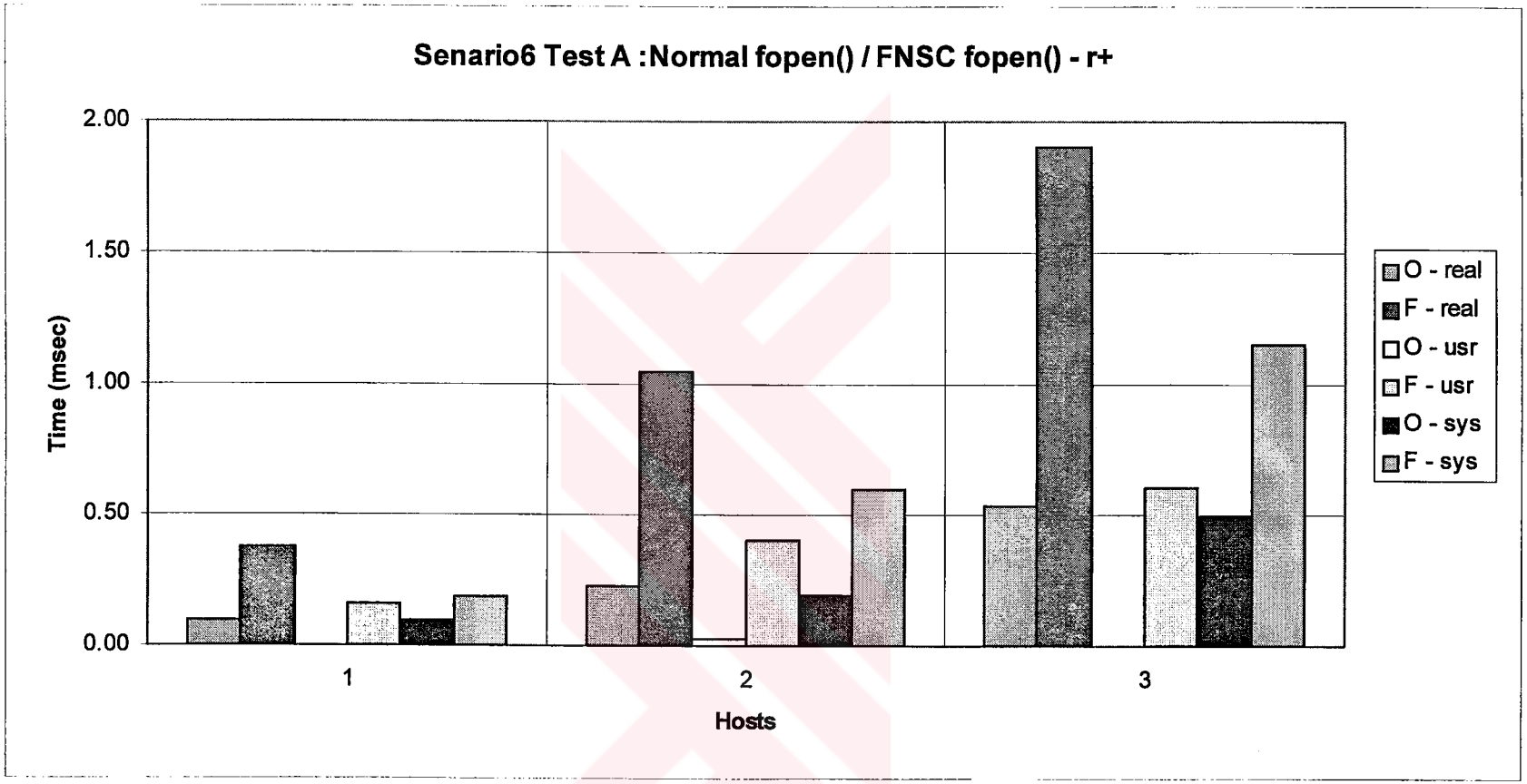
7.3.2.6 Senaryo 6

Senaryo 6 aranılan dosyanın bulunamaması durumunda sistem davranışını incelemek üzere gerçekleştirilmiştir. Senaryo üç ayrı testi içermektedir.

7.3.2.6.1 Senaryo 6 Test A

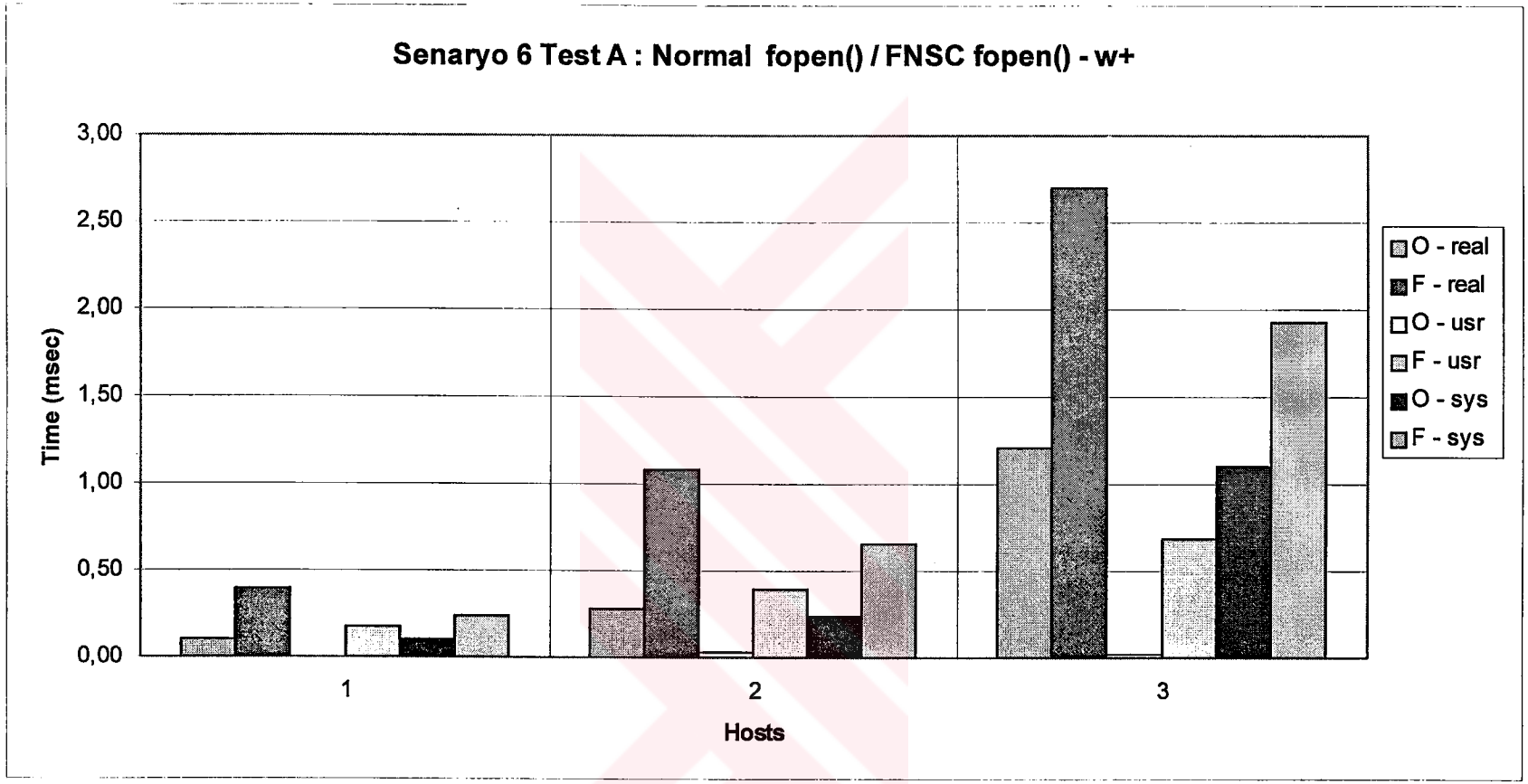
Senaryo 1'in aynısıdır. Ancak aranan dosyanın yerel disk üzerinde olmadığı durumda NFS üzerinden normal fopen() çağrısı (Şekil 7.2) ile FNSC-fopen() çağrısı (Şekil 7.3) ile yapılan erişim sonuçlarını inceler. Dosya erişim süreleri ölçülerek elde edilen değerler, okuma (r+) Tablo 7.29'da, yazma (w+) Tablo 7.30'da ve ekleme (a+) Tablo 7.31'de sunulmuştur

Tablo 7.29 Senaryo 6 Test A : Normal fopen() / FNSC-fopen() (r+) sonuçları



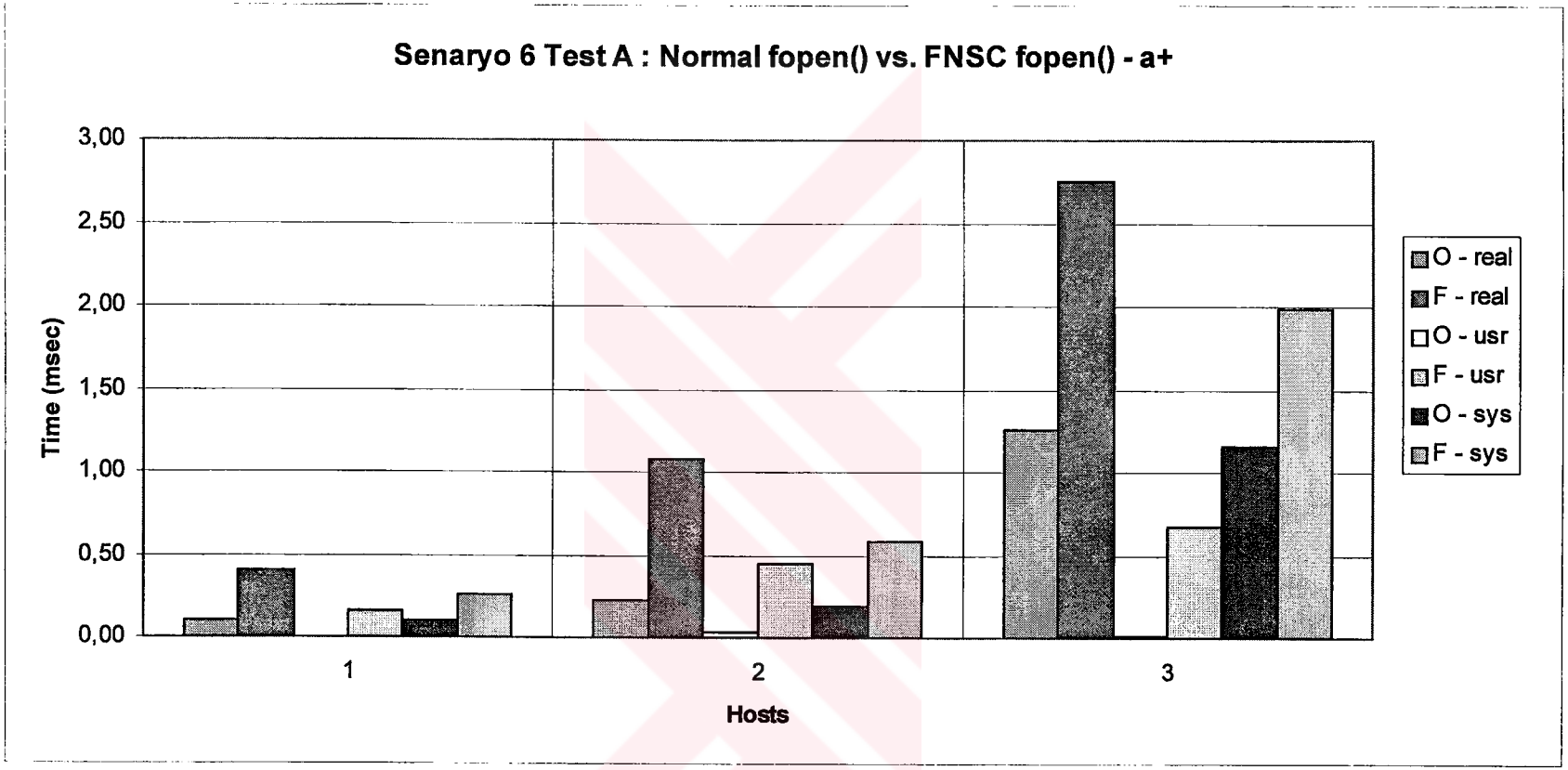
(r+) ortalama	Normal fopen()			FNSC fopen()			
	casper	queen	tweety		casper	queen	tweety
O - real	0.10	0.23	0.54	F - real	0.38	1.05	1.90
O - usr	0.00	0.03	0.00	F - usr	0.16	0.40	0.61
O - sys	0.10	0.19	0.49	F - sys	0.19	0.59	1.15

Tablo 7.30 Senaryo 6 Test A : Normal fopen() / FNSC-fopen() (w+) sonuçları



(w+) ortalama	Normal fopen()				FNSC fopen()		
	casper	queen	tweety		casper	queen	tweety
O - real	0,10	0,28	1,20	F - real	0,40	1,08	2,69
O - usr	0,00	0,04	0,02	F - usr	0,18	0,40	0,69
O - sys	0,10	0,24	1,11	F - sys	0,24	0,65	1,92

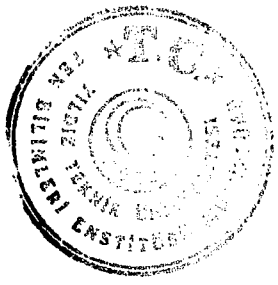
Tablo 7.31 Senaryo 6 Test A : Normal fopen() / FNSC-fopen() (a+) sonuçları



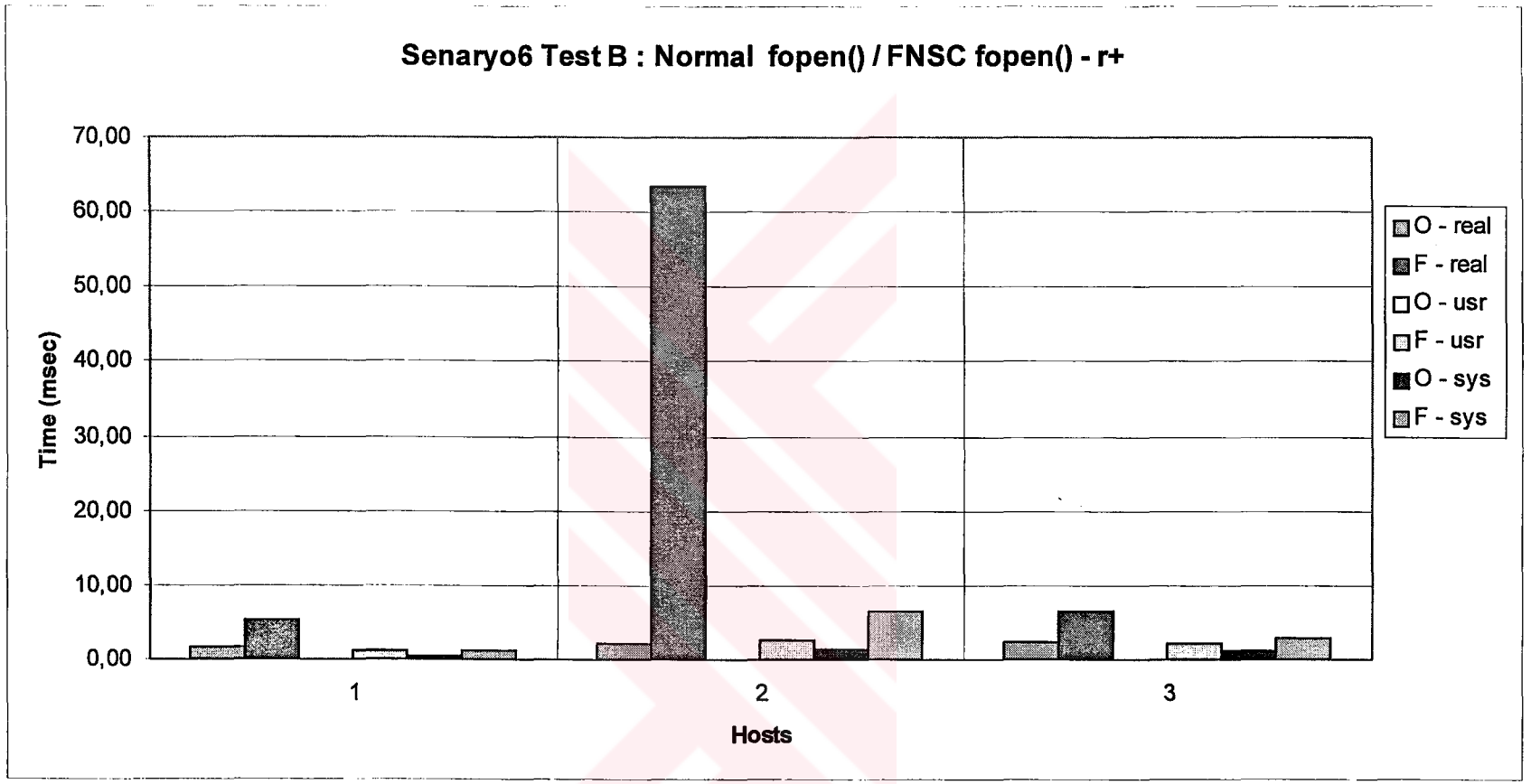
(a+) ortalama	Normal fopen()				FNSC fopen()		
	casper	queen	tweety		casper	queen	tweety
O - real	0,10	0,23	1,26	F - real	0,40	1,07	2,76
O - usr	0,00	0,03	0,02	F - usr	0,16	0,45	0,67
O - sys	0,10	0,19	1,16	F - sys	0,26	0,58	1,99

7.3.2.6.2 Senaryo 6 Test B

Senaryo 2'de olduğu gibi farklı zamanlarda *queen, casper* ve *tweety* isimli bilgisayarların birbirlerinin disklerine NFS üzerinden normal fopen() çağrısı (Şekil 7.4) ile FNSC-fopen() çağrısı (Şekil 7.5) ile yapılan erişim sonuçlarını inceler. Bu senaryoda diğerinden farklı olarak aranılan dosyanın olmaması durumu incelenmektedir. Dosya erişim süreleri ölçülerek elde edilen değerler, okuma (r+) Tablo 7.32'de, yazma (w+) Tablo 7.33'de ve ekleme (a+) Tablo 7.34'de sunulmuştur.

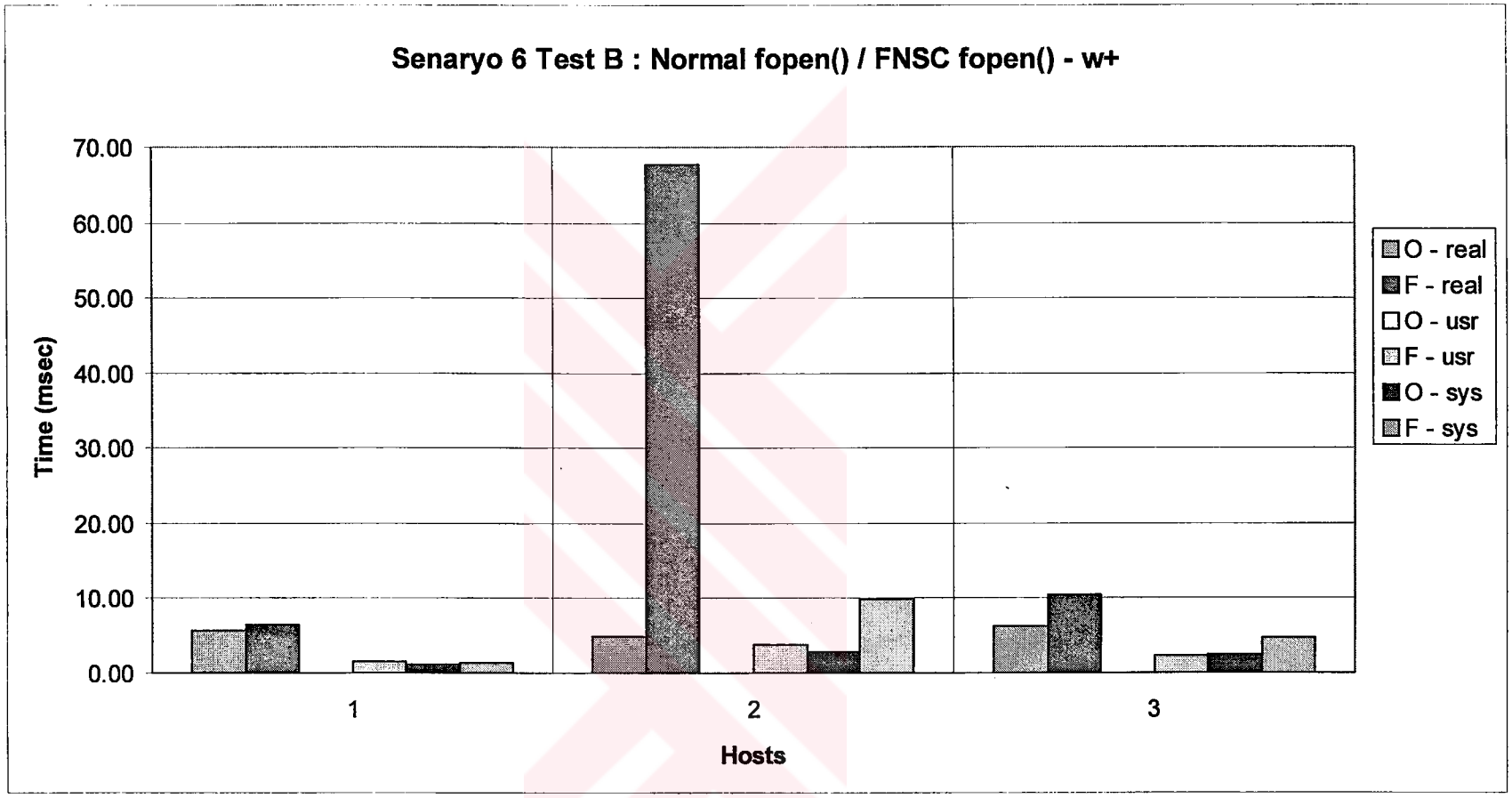


Tablo 7.32 Senaryo 6 Test B : Normal fopen() / FNSC-fopen() (r+) sonuçları



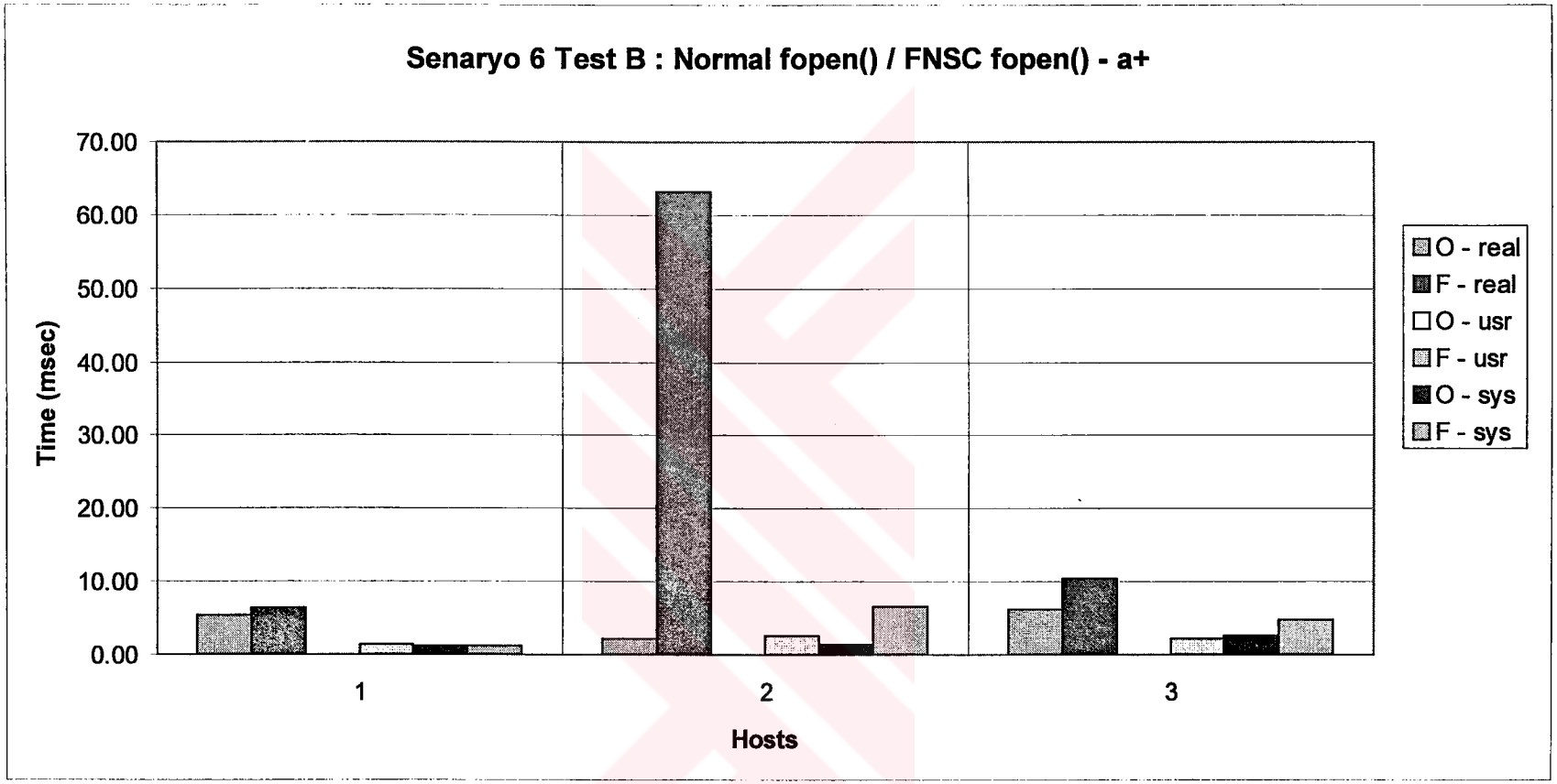
(r+) ortalama	Normal fopen()				FNSC fopen()		
	casper	queen	tweety		casper	queen	tweety
O - real	1,75	2,25	2,35	F - real	5,45	63,49	6,59
O - usr	0,00	0,04	0,03	F - usr	1,31	2,63	2,19
O - sys	0,44	1,39	1,22	F - sys	1,17	6,62	2,96

Tablo 7.33 Senaryo 6 Test B : Normal fopen() / FNSC-fopen() (w+) sonuçları



(w+) ortalama	Normal fopen()				FNSC fopen()		
	casper	queen	tweety		casper	queen	tweety
O - real	5,80	4,93	6,28	F - real	6,47	67,79	10,48
O - usr	0,00	0,07	0,05	F - usr	1,45	3,76	2,23
O - sys	1,08	2,83	2,53	F - sys	1,29	9,93	4,68

Tablo 7.34 Senaryo 6 Test B : Normal fopen() / FNSC-fopen() (a+) sonuçları

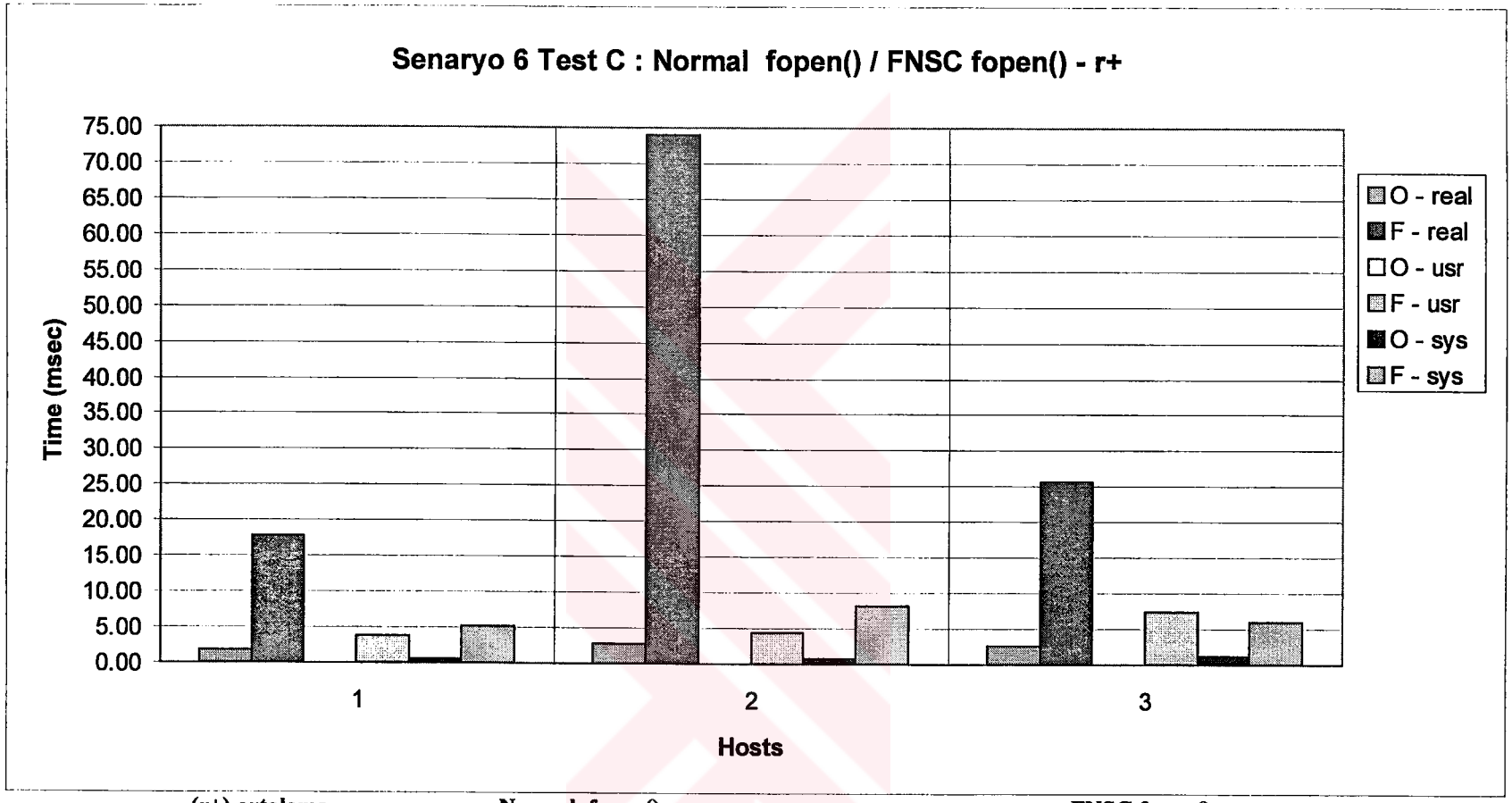


(a+) ortalama	Normal fopen()				FNSC fopen()		
	casper	queen	tweety		casper	queen	tweety
O - real	5,48	2,23	6,25	F - real	6,44	63,12	10,46
O - usr	0,01	0,06	0,06	F - usr	1,38	2,68	2,17
O - sys	1,10	1,35	2,54	F - sys	1,20	6,63	4,77

7.3.2.6.3 Senaryo 6 Test C

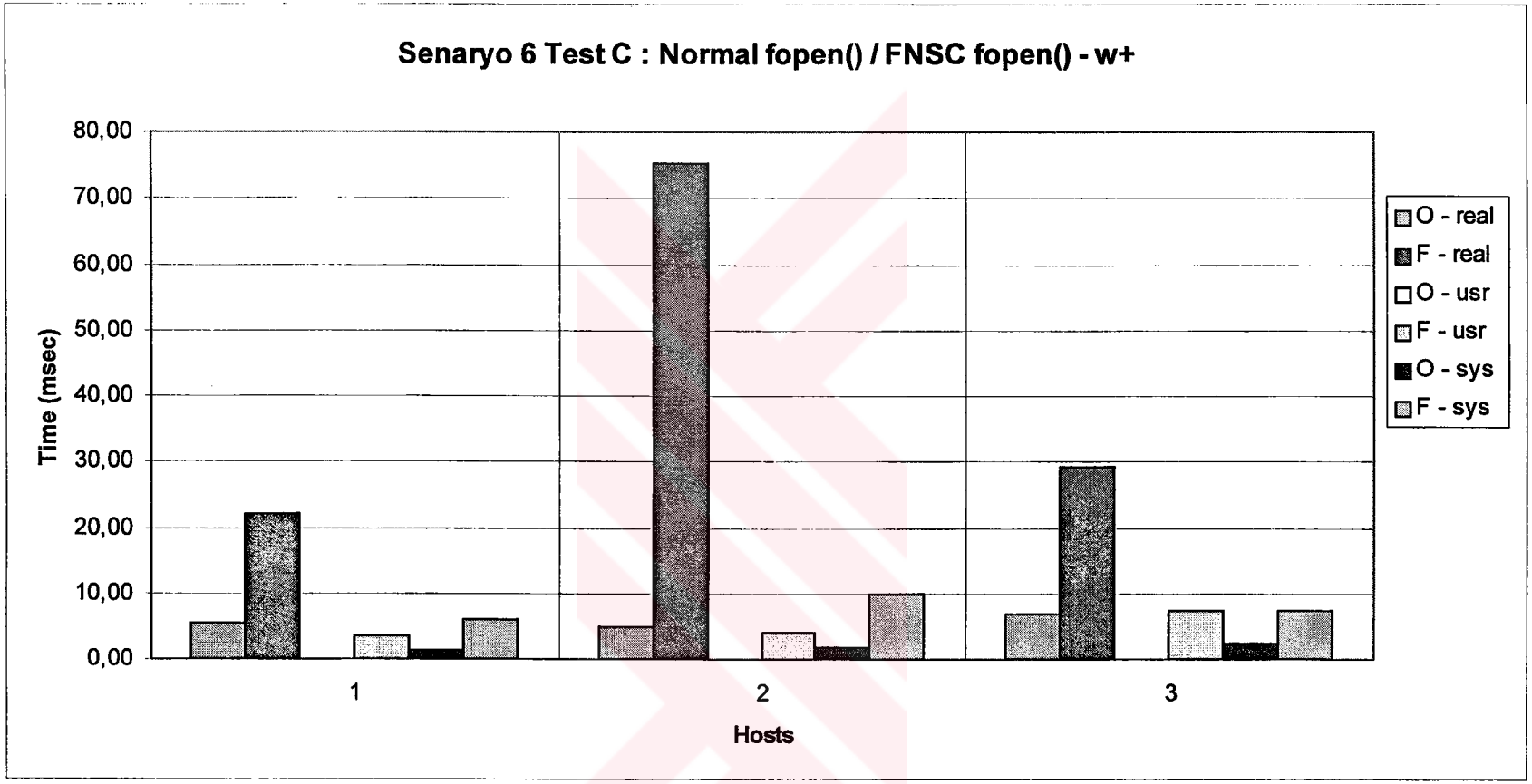
Senaryo 5 Test A olduğu gibi aynı anda *queen, casper* ve *tweety* isimli bilgisayarların birbirlerinin disklerine NFS üzerinden normal fopen() çağrısı (Şekil 7.16) ile FNSC-fopen() çağrısı (Şekil 7.17) ile yapılan erişim sonuçlarını inceler. Bu senaryoda diğerinden farklı olarak aranılan dosyanın olmaması durumu incelenmektedir. Dosya erişim süreleri ölçülerek elde edilen değerler, okuma (r+) Tablo 7.35'de, yazma (w+) Tablo 7.36'da ve ekleme (a+) Tablo 7.37'de sunulmuştur.

Tablo 7.35 Senaryo 6 Test C : Normal fopen() / FNSC-fopen() (r+) sonuçları



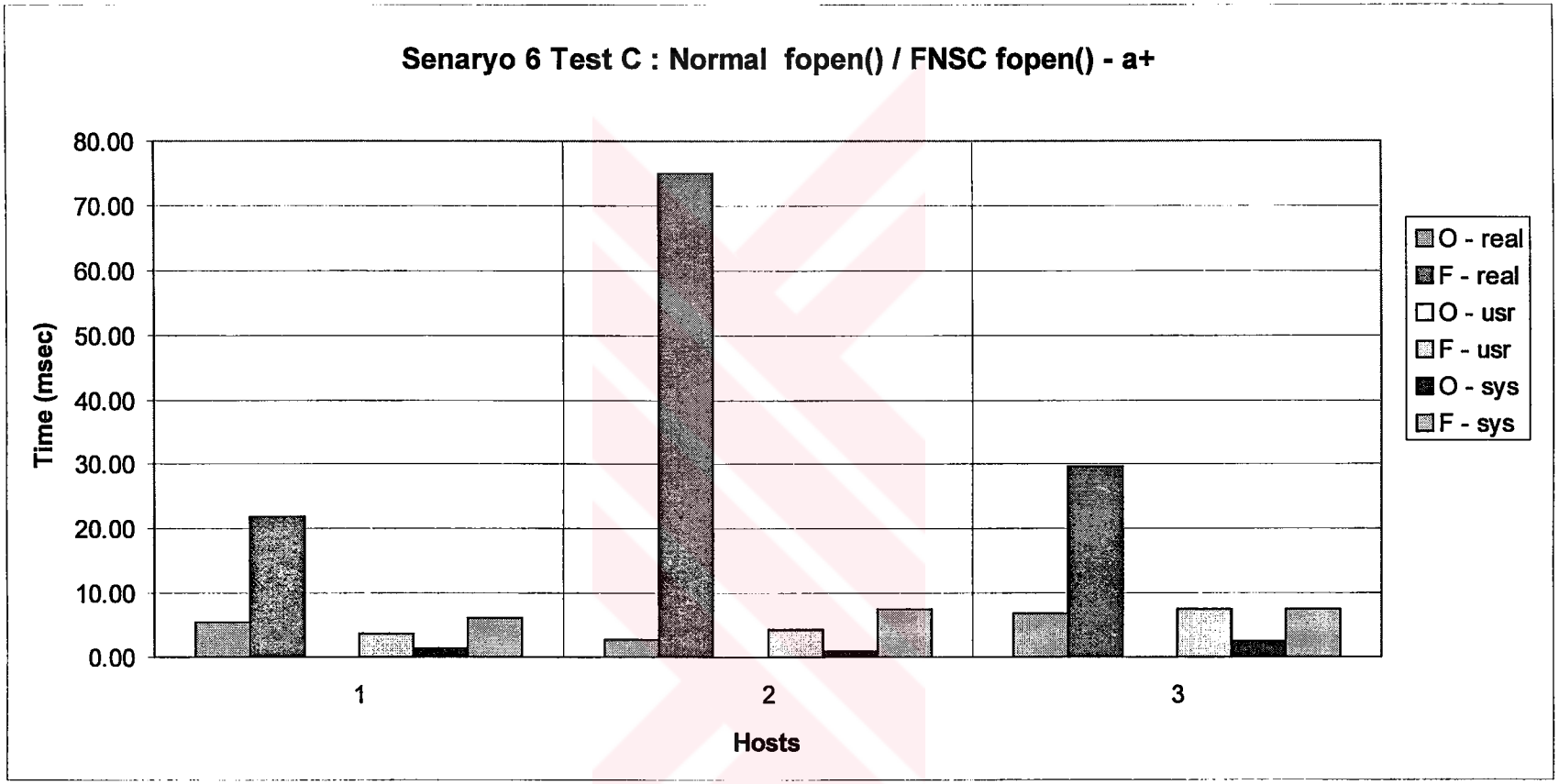
(r+) ortalama	Normal fopen()			FNSC fopen()			
	casper	queen	tweety		casper	queen	tweety
O - real	1,85	2,84	2,64	F - real	17,75	73,99	25,59
O - usr	0,00	0,05	0,03	F - usr	3,72	4,43	7,48
O - sys	0,56	0,88	1,18	F - sys	5,19	8,17	6,03

Tablo 7.36 Senaryo 6 Test C : Normal fopen() / FNSC-fopen() (w+) sonuçları



(w+) ortalama	Normal fopen()				FNSC fopen()		
	casper	queen	tweety		casper	queen	tweety
O - real	5,64	5,01	6,85	F - real	22,16	75,42	29,22
O - usr	0,00	0,07	0,03	F - usr	3,72	4,26	7,54
O - sys	1,34	1,82	2,59	F - sys	5,97	9,96	7,42

Tablo 7.37 Senaryo 6 Test C : Normal fopen() / FNSC-fopen() (a+) sonuçları



(a+) ortalama	Normal fopen()				FNSC fopen()		
	casper	queen	tweety		casper	queen	tweety
O - real	5,55	2,78	6,86	F - real	21,83	74,92	29,71
O - usr	0,01	0,05	0,03	F - usr	3,73	4,25	7,58
O - sys	1,39	0,86	2,61	F - sys	6,08	7,48	7,49

7.4 Senaryoların Değerlendirilmesinde Dikkat Edilmesi Gereken Noktalar

Örnek senaryolar ve bunların uygulaması ile elde edilen değerlerin grafikleri incelenirken şunlara dikkat edilmelidir:

- Uygulanan senaryolarda verilen değerler, bilgisayarların ve bilgisayar ağının anlık yüklenmesinden kaynaklanan gecikmelerin yaratacağı yüksek erişim değerlerinden mümkün olduğunca az etkilenebilmek amacıyla, ayrı zamanlarda, her biri 1000 adet okuma/yazma/ekleme işlemini gerçekleştiren 50 test sonucunun ortalamasını yansıtmaktadır. Buna rağmen her senaryonun uygulandığı zaman aralığında bilgisayarların genel işlem ve iletişim yükleri bazı ufak farklılıkların doğmasını engelleyememiştir.
- Her senaryo FNS ile ortaya atılan sistemin farklı bir temel özelliğini ortaya çıkartmak üzere özel olarak belirlenmiş ve uygulanmıştır. Bunlara ek olarak her zaman farklı bir özelliği ortaya çıkartabilecek yeni bir senaryo üretilebilir.
- Senaryo 1 ile 2 karşılaştırıldığında NFS'nin devreye girmesi ile yerel disk ile NFS yardımıyla erişilen uzak disk üzerinde normal fopen() çağrısı kullanılarak yapılan okuma/yazma/ekleme işlemlerinde 15-50 kat arasında zaman farkları olduğu görülmektedir.
- Senaryo 2,3,4,5 ve 6'da görüldüğü gibi *queen* isimli bilgisayar üzerinde çalışan Aix işletim sisteminde, normal fopen() çağrısı ile yapılan (w+) işlemi, (r+) ve (a+) işlemlerine göre yaklaşık olarak 2 kat daha fazla zaman almaktadır. Bu fark FNSC-fopen() çağrısında bu derece bariz görünmemekle birlikte (r+) ve (a+) işlemlerinin (w+) işlemine göre daha az zaman aldığı gözlenmiştir.
- Aix işletim sistemini çalıştıran *queen* isimli bilgisayar üzerinde uygulanan Senaryo 2,3,4,5 ve 6'da kullanıcıya yansıyan dosya açma sürelerinin diğer bilgisayarlardakilerden daha fazla olduğu dikkat çekmektedir. Bunun nedeni Aix işletim sisteminin Internet isim-adres dönüşümlerini DNS den sorgulama ile çözmesidir. Oysa *tweety* isimli bilgisayar üzerinde çalışan SunOS, NIS (Network Information Service) üzerinden isim-adres dönüşümlerini daha hızlı sonuçlandırabilmektedir.

- Senaryo 2,3,4 ve 5'de kullanılan normal fopen() çağrısına erişilmek istenen dosyanın tam dizin bilgisinin verilmesi gerekmektedir. Bu ise dosyanın kullanıcı tarafından nerede olduğunun bilinmesi demektir. Oysa, FNSC-fopen() çağrısı, dosyaya kısmi bilgi (dosya adı veya kısmi dizin bilgisini takip eden dosya adı) ile erişilebilmesini sağladığı için yeri tam olarak bilinmeyen bir dosyaya erişimde kullanıcının dosyanın tam yerini bulmak için harcayacağı zaman FNSC-fopen() çağrısının kullanımı ile harcanan zamana göre karşılaştırılamayacak kadar fazla olacaktır. Buna göre FNSC-fopen() çağrısı ile elde edilen değerleri salt büyüklük olarak değerlendirmek yanıltıcı sonuçlar çıkarılmasına sebep olur.
- Senaryo 1 ile Senaryo 6 Test A, Senaryo 2 ile Senaryo 6 Test B ve Senaryo 5 Test A ile Senaryo 6 Test C karşılaştırıldığı zaman FNS'ye dahil bir dosyaya erişmek ile FNS'de bulunmayan bir dosyaya* erişmek istemek arasında kullanıcıya yansıyan süreler açısından belirgin bir fark görülmemektedir.

7.5 Dosya İsim Servisinin başarımı

Dosya İsim Servisinin başarımı;

- İstemci ve sunucu modüllerinin,
- İstemci ve sunucu modüllerinin üzerinde çalıştıkları işletim sistemlerinin,
- Dosya erişiminde yararlanılan NFS'in,
- Dosyaların üzerinde yer aldığı disk sistemlerinin,
- TCP/IP tabanlı iletişim protokolünün dayandığı ve Internet isimlerinin adreslere dönüştürülmesini sağlayan DNS'in,
- Kullanılan bilgisayar ağının

başarımı ile ilgilidir. Bu değerler, zamanın yanı sıra birbirlerinin de fonksiyonları olarak karşımıza çıkmaktadır. Bu nedenle Dosya İsim Servisinin başarımını tek ve basit bir formül ile ifade etmek oldukça güçtür.

---

* FNS'ye dahil bir dosyaya erişmek demek, dosyaya ait bir işaretçinin (file pointer) kullanıcıya döndürülmesi demektir. Kullanıcı daha sonra yapacağı dosya erişimlerinde bu işaretçiden yararlanarak dosya üzerindeki işlemlerini gerçekleştirecektir. Oysa FNS'ye ait indeksler içerisinde bulunmayan bir dosya erişilmek istenmesi durumunda kullanıcıya NULL işaretçi döndürülecektir. Döndürülen NULL değeri kullanıcıya böyle bir dosyanın olmadığını veya erişilemeyeceğini bildirmektedir.

Genel başarım hakkında doğru fikir edinmek ve hangi değerlere bağlı olarak değişiklik gösterdiğini ifade edebilmek için örnek senaryolar üzerinde yapılan ölçümlerin kullanılması uygun olacaktır. Örnek senaryoların işleyişleri sırasında izlenen iş akışlarının farklılıklarından dolayı her senaryo için değişik değerler elde edilmektedir. Bu sebeple Dosya İsim Servisinin başarımını, izlenen iş akışına bağlı olarak ifade eden farklı denklemler Bölüm 7.5.1 ve Bölüm 7.5.2 de sunulmuştur. Bu denklemlerde kullanılan değişkenler ve açıklamaları Tablo 7.38 de verilmiştir.

Tablo 7.38 Denklemlerde kullanılan değişkenler ve açıklamaları

t_{OS}	İşletim sisteminin cevap verme süresini belirler.
t_{FNS}	Dosya İsim Sunucusu'nun (FNS) cevap verme süresini belirler.
t_{FNSC}	Dosya İsim Servis İstemcisi'nin (FNSC) cevap verme süresini belirler.
t_{NFS}	Network File System (NFS) cevap verme süresini belirler.
$t_{I/O}$	Disk erişim (I/O) süresini belirler.
t_{NET}	Bilgisayar ağı üzerindeki erişim süresini belirler.
t_{USER}	Kullanıcının seçim yapma süresini belirler.
t_{DNS}	Internet isimlerinin, adreslere dönüştürülmesi işlemini yapan Domain Name Server (DNS) cevap verme süresini belirler.
T	Kullanıcıya yansıyan dosya açma süresi

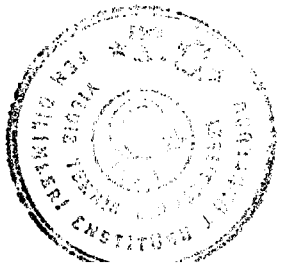
7.5.1 Normal fopen() kullanılması durumu

Kullanıcının, C programlama dilindeki fopen()* çağrısı ile yerel disk üzerinde dosya açma süresi denklem 7.1 de verilmiştir.

$$T = f(t_{OS}, t_{I/O}) \qquad (7.1)$$

Kullanıcının, NFS üzerinden erişilen (mounted) disk üzerinde dosya açma süresi denklem 7.2 de verilmiştir.

* fopen("[[/] path/] file", "r | r+ | rb | rb+ | w | w+ | wb | wb+ | a | a+ | ab | ab+")



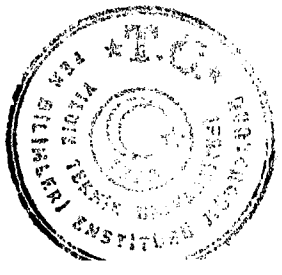
$$t_{NFS} = f(t_{OS}, 2 * t_{NET}, t_{I/O})$$
$$T = f(t_{OS}, t_{NFS}) \quad (7.2)$$

7.5.2 FNSC - fopen() kullanılması durumu

FNSC-fopen() çağrısının kullanılması durumunda elde edilen sonuçlar, aktarılan parametrelere bağlı olarak değişiklik göstermektedir. Bu parametreler Tablo 7.39 görüldüğü gibi kendi içlerinde sınıflandırılmıştır.

Tablo 7.39 FNSC-fopen() sınıfları

Sınıf 1A	fopen("file", "w | w+ | wb | wb+ | a | a+ | ab | ab+") fopen("path/file", "w | w+ | wb | wb+ | a | a+ | ab | ab+") fopen("/path/file", "w | w+ | wb | wb+ | a | a+ | ab | ab+") fopen("path/file", "r | r+ | rb | rb+") fopen("/path/file", "r | r+ | rb | rb+")
Sınıf 1B	fopen("localhost:file", "w | w+ | wb | wb+ | a | a+ | ab | ab+") fopen("localhost:path/file", "w | w+ | wb | wb+ | a | a+ | ab | ab+") fopen("localhost:/path/file", "w | w+ | wb | wb+ | a | a+ | ab | ab+") fopen("localhost:/path/file", "r | r+ | rb | rb+")
Sınıf 2	fopen("remotehost:file", "w | w+ | wb | wb+ | a | a+ | ab | ab+") fopen("remotehost:path/file", "w | w+ | wb | wb+ | a | a+ | ab | ab+") fopen("remotehost:file", "r | r+ | rb | rb+") fopen("remotehost:path/file", "r | r+ | rb | rb+")
Sınıf 3	fopen("remotehost:/path/file", "w | w+ | wb | wb+ | a | a+ | ab | ab+") fopen("remotehost:/path/file", "r | r+ | rb | rb+")
Sınıf 4	fopen("localhost:file", "r | r+ | rb | rb+") fopen("localhost:path/file", "r | r+ | rb | rb+")
Sınıf 5	fopen("file", "r | r+ | rb | rb+")



Kullanıcının, Tablo 7.39 de verilen sınıflara göre dosya açma sürelerini belirleyen denklemler aşağıda verilmiştir.

$$t_{FNS} = f(t_{OS}, t_{NET}) \quad (7.3)$$

$$t_{NFS} = f(t_{OS}, 2 * t_{NET}, t_{I/O}) \quad (7.4)$$

$$0 < t_{USER} < \infty \quad (7.5)$$

$$t_{DNS} = f(t_{OS}, 2 * t_{NET}, t_{I/O}) \quad (7.6)$$

olmak üzere;

Sınıf 1A $T = f(t_{OS}, t_{FNSC}, t_{I/O})$ (7.7)

Sınıf 1B $T = f(t_{OS}, t_{FNSC}, t_{DNS}, t_{I/O})$ (7.8)

Sınıf 2 $T = f(t_{OS}, t_{FNSC}, t_{NET}, t_{FNS}, t_{USER}, t_{DNS}, t_{NFS})$ (7.9)

Sınıf 3 $T = f(t_{OS}, t_{FNSC}, t_{DNS}, t_{NFS})$ (7.10)

Sınıf 4 $T = f(t_{OS}, t_{FNSC}, t_{NET}, t_{FNS}, t_{USER}, t_{DNS}, t_{I/O})$ (7.11)

Sınıf 5 $T = f(t_{OS}, t_{FNSC}, t_{NET}, t_{FNS}, t_{USER}, t_{DNS}, t_{I/O} \mid t_{NFS})$ (7.12)

8. SONUÇ VE ÖNERİLER

8.1 Sonuç

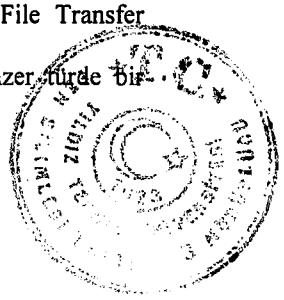
Hızlı iletişim altyapısının yaygın ve paralel işlemcili sistemlere göre düşük maliyetli olması, ağ protokollerindeki ilerlemelerin sonucu olarak bağımsız bilgisayarların birbirleriyle iletişim kurup, ortak amaca ulaşmak için güç birliği yapması fikrini doğurdu. Bunun sonucunda istemci/sunucu yapısındaki dağıtık uygulamalar yerel ve geniş alan ağlarında büyük önem kazanmaya başladı. Kazanılan bu itici güç, dağıtık sistemler üzerinde gerek ticari, gerek akademik alanda çok farklı uygulamaların geliştirilmesine imkan sağladı. Internet'in yaygınlaşması ile de dünya çapında istemci/sunucu tipinde, oldukça farklı donanım ve yazılım platformlarında çalışabilen çeşitli servisler* verilmeye başlandı. 'Dosya İsim Servisi - FNS' olarak adlandırdığım doktora tez çalışması bu yönde yapılan yeni** bir uygulamayı içermektedir. 'Dosya İsim Servisi'nin özellikleri ve yerine getirdiği temel işlevleriyle şu şekilde özetlenebilir:

1. **Özellikler**
    - De facto ağ protokolü olan TCP/IP üzerinden, bağlantılı tipte servis vermektedir.
    - Dağıtık sistemler üzerinde, istemci/sunucu modelini kullanmaktadır.
    - Sunucu modülü, birden fazla istemciye cevap verebilmek için, eşzamanlı olarak çalışmaktadır.
    - Gelen farklı istekleri, mümkün olduğunca kısa zamanda karşılamak için 'çok işlem parçacıklı' bir yapıda tasarlanmıştır.
    - TCP/IP ağ protokolü üzerinde, standart bir dosya erişim yöntemi olan NFS'yi kullanmaktadır.
    - İstenen dosyanın dağıtık yapı içindeki yerinin bulunması ve erişimin sağlanması işlemleri kullanıcıya mümkün olduğunca şeffaftır.

---

* Bu servislerin en yaygın kullanılanlarından bazıları WWW (World Wide Web), FTP (File Transfer Protocol), Gopher, Wais, Whois, IRC (Internet Relay Chat) dir.

** Doktora tezinin yazıldığı Ekim-1997 tarihine kadar yapılan kaynak araştırmalarında benzer türde bir uygulamaya rastlanmamıştır.



- Farklı donanım ve yazılım platformlarında istemci ve/veya sunucu modülleri çalışabilir. Örnek olarak, PowerPC işlemcisinde çalışan AIX işletim sisteminde: FNSA ve FNSC; Intel işlemcisinde çalışan UnixWare işletim sisteminde: FNS,FNSA ve FNSC; Sparc işlemcisinde çalışan SunOS işletim sisteminde: FNSA ve FNSC modülleri gerçekleştirilmiştir.
- Sunucu modülleri (FNS ve FNSA) dinamik hafıza ayrımı sayesinde sistem kaynaklarını etkin biçimde kullanır.
- Modüllerin geliştirmeye açık ve taşınabilir olması için C programlama dili kullanılmıştır.
- FNSC modülü, kullanılacak *fnsdef.h* tanım dosyası ile hedef seçilen işletim istemi için önceden derlenerek oluşturulmuş bir nesne kod (object code) yardımıyla, C programlama dilinde kullanılan *fopen()* çağrısının yerini alabilmektedir.
- İkincil sunucu (SFNS) kullanımı ile; servisin sürekliliğine, FNSC'lerin yaratacağı haberleşme ve işlem yükünün sunucular arasında dengeli dağılımına ve sistemin genel cevap verme süresinin iyileşmesine katkıda bulunmaktadır.
- FNSC modülünün kullanıcı arayüzü, kullanıcının kendi amacına uygun olarak hazırlayacağı bir başka arayüz modülü ile değiştirilebilecek şekilde tasarlanmıştır.

2. **İşlevler**

- Etki alanındaki dizin ve dosya bilgileri sunucular üzerinde tutulur.
- Sunucu ve ikincil sunucular arasında bilgi akışı ve tutarlılığı (consistency) sağlanır.
- Etki alanı içindeki ajanlar, kendi dosya sistemlerine ait bilgileri sunuculara gönderirler.
- İstemcilerden sunuculara gelen sorgulama istekleri, eldeki bilgiler taranarak cevaplandırılır.
- İstemciler, sunucunun etki alanında bulunmak kaydıyla erişmek istedikleri dosyaya yarı şeffaf* bir şekilde ulaşır.

* Aranan özelliğe uygun birden fazla dosya bulunması durumunda kullanıcıya dosyanın bulunduğu yerler, büyüklük ve tarih bilgileri gösterilerek seçim yapması istenmektedir. Kullanıcı müdahalesini gerektiren bu durum nedeniyle erişime tam anlamıyla şeffaf demek mümkün değildir. Bu nedenle yarı şeffaf tanımlaması kullanılmıştır.

FNSC-fopen() çağrısı normal fopen() çağrısına bir alternatif olarak sunulmuştur. Aranan dosyanın farklı bir dizin hatta Dosya İsim Servisinin etki alanı (FNSD) içinde başka bir bilgisayar sistemi üzerinde bile olsa erişilmesine (location, migration transparency) imkan sağlamaktadır. Bu özellik dikkate alındığında, FNSC-fopen() çağrısı kullanılarak oluşturulan programlar, kullandıkları dosyalar yer değiştirse bile (location/migration) yeniden derlenmeye ihtiyaç duymadan çalışmaya devam edecek, ayrıca normal fopen() çağrısının FNSC-fopen() çağrısı ile sistem seviyesinde değiştirilmesi sonucunda, sistemde var olan diğer programlar da FNSC-fopen() çağrısının getirisinden yararlanacaklardır.

8.2 Öneriler

Araştırmacılar "ne, neden, nasıl?" sorularına cevap ararlar. Gerçekleştirdiğim tez çalışmasına mümkün olduğunca objektif bir gözle bakarak 'Eksikleri nedir? Nasıl daha iyi olurdu?' sorularını sorduğumda aldığım cevapları bir özeleştiri, bu servisi geliştirmek için bir sebep, bu yönde çalışma yapacaklara bir uyarı olarak görüyorum. Buna göre:

- Aynı etki alanı içinde çalışması düşünülen FNS ve SFNS'ler arasında tanımlanacak bir protokol ile FNSA'ların sadece birincil FNS'ye bilgi aktarmasının yaratacağı trafik ve iş yükü, etki alanı içinde bulunan diğer SFNS'lere dağıtılabilecektir. FNS ile SFNS'ler arasında uygulanacak bu protokol uyarınca, FNS'den düzenli aralıklarla güncelleme bilgilerini isteyen SFNS, gelen bilgileri kendi tablolarını silmeden, sadece değişiklikleri işleyerek alacak, böylece sistem kaynaklarından daha etkin yararlanabilecektir. Ayrıca FNSA'lardan SFNS'ye gelen bilgiler de, SFNS tarafından FNS'ye yollanmalıdır. SFNS'lerden bilgi toplayan FNS, her SFNS'nin güncelleme isteğine elindeki bilgilerin tamamını yollayacağı için etki alanı içindeki tüm SFNS'ler birbirlerinin bilgilerine sahip olabilecektir.
- FNS ve SFNS'ler arasındaki bilgi akışı, son güncellemeden sonraki değişikliklerin gönderilmesiyle azalacaktır.
- FNS modülünde kullanılan işlem parçacıklarının, ihtiyaç duyuldukça yaratılması yerine, işlem parçacıklarından oluşan bir havuzdan (thread pool) kullanılması* hem

* Bu özellik kullanılan işletim sistemi ile ilgilidir.

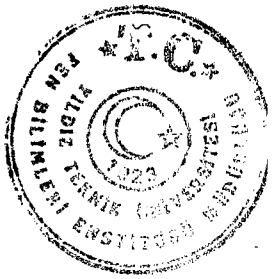
sistem kaynaklarından daha etkin yararlanılmasını hem de servis verme süresinin kısaltılmasını sağlayacaktır.

- FNSC'nin birden fazla aynı isimde dosya bulması durumunda, kullanıcıya gösterilecek liste, seçme kolaylığı sağlamak amacıyla, belirlenecek bazı özelliklere göre (dosyanın yaratılış tarihi, kullanılma sıklığı vb.) ön sıralama işleminden geçirilebilir.

8.3 Dosya İsim Servisi Konusunda İleride Yapılabilecek Çalışmalar

Dosya İsim Servisi (FNS), kullanılacak altyapının özelliklerine göre sistem sorumlularınca ayarlanabilecek koşullama dosyaları yardımıyla oldukça esnek bir yapıya sahiptir. Ancak kullanılan altyapı içerisindeki bazı parametrelere (kullanılan iletişim altyapısının hızı, mevcut trafiği, dosya erişim sıklıkları, dosya büyüklükleri, dosya sunucuların iş yükleri ve dosya erişim zamanları vb.) bağlı olarak koşullama dosyasında kullanılan değişkenler arasındaki denklem, yoğun deneysel ve istatistiksel çalışma ile modellenebilecektir.

Dosya İsim Servisi akademik ve deneysel bir çalışma olması nedeniyle güvenlik doğrudan dikkate alınmamıştır. Ancak Dosya İsim Servisi'nin güvenlik konusunda oluşturduğu veya oluşturabileceği açıkların tespit edilmesi ve bu yönde yeniliklerin servise kazandırılması da faydalı olacaktır.

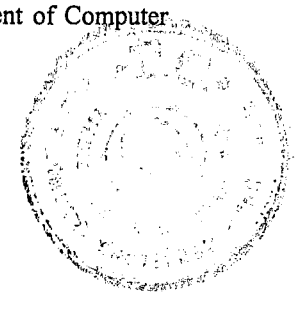


KAYNAKLAR

- ACM, (1997), Communications of ACM, Feb.
- Akl, S.G., (1989), The Design and Analysis of Parallel Algorithms, Prentice-Hall International Editions, New Jersey
- Andrews, G.R.ve Schneider, F.B., (1983), "Concepts and Notations for Concurrent Programming", ACM Computing Surveys:15, 3-44,
- Athas, W.C ve Seitz, C.L., (1988), "Multicomputers: Message-Passing Concurrent Computers", Computer, Aug.
- Bach, M.J., (1986), The Design of the Unix Operating System, Prentice-Hall International Editions, New Jersey
- Bal, E.H., (1990), Programming Distributed Systems, Silicon Press
- Bal, E.H., Steiner, G.J.ve Tanenbaum, A.S., (1989), "Programming Languages for Distributed Computing Systems", ACM Computing Surveys Vol.21, No.3, Sep.:261-322, New Jersey
- Bell, Gordon, (1997), "The Body Electric", Communication of the ACM, Feb.: 31-32
- Ben-Ari, M., (1990), Principles of Concurrent and Distributed Programming, Prentice-Hall International Series in Computer Science, Hertfordshire
- Besaw, L., (1987), "Berkeley UNIX System Calls and Interprocess Communication", Revised, Solomon, M. Sep.1987, Jan.1991
- Bic, L. ve Shaw,A.C., (1988), The Logical Design of Operating Systems, Prentice-Hall International Editions, New Jersey
- Bilişim Terimleri Sözlüğü, (1996), Türkiye Bilişim Derneği
- Black, U.D., (1993), Data Communications And Distributed Networks 3e, Prentice-Hall International Editions, New Jersey
- Bloomer, J., (1992), Power Programming with RPC, O'Reilly & Associates Inc., CA
- Champine, A.G., (1991), MIT Project Athena - A Model for Distributed Campus Computing, Digital Press, MA
- Cochrane, P., (1994), "Information Technology and the Future", BT Laboratories, Martlesham Heath, Ipswich, UK (http://www.labs.bt/com/people/cochrap/papers/it_expon.html)
- Cochrane, P., "Technology of Tomorrow", BT Laboratories, Martlesham Heath, Ipswich, UK (http://www.labs.bt/com/people/cochrap/papers/technolo.html)
- Cochrane, P., (1992), "A Glimpse of The Future", BT Laboratories, Martlesham Heath, Ipswich, UK (http://www.labs.bt/com/people/cochrap/papers/technolo.html)
- Cochrane, P., Fisher, K. ve Taylor, R., (1993), "The Office You Wish You Had", BT Laboratories, Martlesham Heath, Ipswich (http://www.labs.bt/com/people/cochrap/papers/officeyo.html)
- Comer, D.E, (1991), Internetworking with TCP/IP - Principles; Protocols, Architecture Volume I, Prentice-Hall International Edition, New Jersey

- Comer, D.,E.ve Stevens, D.,L., (1993), Internetworking With TCP/IP Client-Server Programming and Applications BSD Socket Version Volume III, Prentice-Hall International Editions, New Jersey
- Darnel, P.A. ve Margolis, P.E., (1988), Software Engineering in C, Springer Books on Professional Computing
- Day, J.D. ve Zimmerman, H., (1983), "OSI Reference Model", Proceedings of IEEE, Vol 71, Dec.: 1334-1340
- Day, M., Koontz, M. ve Marshall, D., (1993), Novell's Guide to NetWare 4.0 NLM Programming, Novell Press, San Jose
- Dijkstra, W.E., (1965), Co-Operating Sequential Processes in Programming Languages, Genuys, F. (ed.), Academic Press
- Enslow, P.H., (1978), "What is a Distributed System?", Computer: 13-21
- Flynn, M.J., (1972), "Some Computer Organizations and Their Effectiveness", IEEE Transactions on Computers, Sep.: 948-960
- Futacı, S. (1995), "RELP Paralel Programlama Dili Tasarımı ve Uygulaması", YTÜ, Fen Bilimleri Enstitüsü, Doktora Tezi
- Goscinski, A., (1992), Distributed Operating Systems: The Logical Design
- Hunt, C., (1992), Help For System Administrators, TCP/IP Network Administration, O'Reilly & Associates Inc. , CA
- IBM, (1991), IBM Bilgisayar Terimleri Sözlüğü
- IEEE, (1994), "Kit Helps With Design of Fast VMEbus Systems", IEEE Spectrum
- Jones, A.K. ve Schwarz, P., (1980), "Experience Using Multiprocessor Systems - A Status Report", ACM Computing Surveys, Jun.: 121-165
- Leffler,. S.J., Fabry, R.S., Joy, W.N. ve Lapsey, P., Computer Systems Research Group Dept. of Electrical Eng. And Computer Science University of California at Berkeley, Miller, S. ve Torek, C., Heterogeneous Systems Laboratory Dept. of Computer Science University of Maryland College Park, "An Advanced 4.4BSD Interprocess Communication Tutorial"
- McGlynn, D.R., (1978), Distributed Processing And Data Communications, Wiley-Interscience Publication
- Milenkovic, M. (1987), Operating Systems Concepts and Design, McGraw-Hill Book Company
- Novell INC., (1993a), Netware NLM Library Reference Volume II Edition 1.1 Software Developer's Kit, Utah
- Novell INC., (1993b), Using Netware Services for NLMs Edition 1.1 Software Developer's Kit, Utah
- Novell INC., (1993c), Netware NLM Library Reference Volume I Edition 1.1 Software Developer's Kit, Utah
- Nutt, G.J., (1992), Open Systems, Prentice Hall Series in Innovative Technology, New Jersey
- Nutt, G.J., (1992b), Centralized And Distributed Operating Systems, Prentice Hall, New Jersey

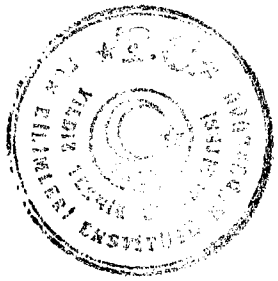
- RFC 768, (1980), Postel, S.J., "User Datagram Protocol"
- RCF 791, (1981), "Internet Protocol", Darpa Internet Program Protocol Specifications
- RFC 793, (1981), "Tansmission Control Protocol", Darpa Internet Program Protocol Specifications
- RFC 882, (1983), Mockapetris, P., "Domain Names: Concepts and Facilities"
- RFC 883, (1983), Mockapetris,P., "Domain Names: Implementation and Specification"
- RFC 1057, (1988), "Remote Procedure Call Specifications Version 2", Sun Microsystems Inc.
- RFC 1094, (1989), "Network File System Specifications" , Sun Microsystems Inc.
- RFC 1180, (1991), Socolofsky, T. ve Kale, C., "A TCP/IP Tutorial", Spider Systems Limited
- RFC 1813, (1995), "NSF Version 3", Sun Microsystems Inc.
- Riezenman, M.J., (1994), "Special Report", IEEE Spectrum
- Russell, R.M., (1978), "The CRAY-1 Computer System", Communication of ACM, Jan.
- Schildt, H., (1985), C Made Easy, Osborn Mc-Graw Hill, California
- Schildt, H., (1987), The Complete Reference C, Osborn Mc-Graw Hill, California
- Schildt, H., (1989), Born to Code in C, Osborn Mc-Graw Hill, California
- Sechrest, S., "An Introductory 4.4BSD Interprocess Communication Tutorial", Computer Sciences Research Group - Computer Science Division Dept. of Electrical Eng. and Computer Sciences University of California at Berkeley
- Selker, T., (1997), "What Will Happen in the Next 50 Years?", Communication of the ACM, Feb.: 88-89
- Sloman, M. ve Kramer, J., (1987), Distributed Systems and Computer Networks, Prentice-Hall International Series in Computer Science, Hertfordshire
- Sunsoft, (1994), Multithreaded Programming Guide, Sun Microsystems Inc., California
- Tanenbaum, A.S., (1987), Operating Systems, Design and Implementation, Prentice-Hall International Editions, New Jersey
- Tanenbaum, A.S., (1989), Computer Networks, Prentice-Hall International Editions, New Jersey
- Tanenbaum, A.S., (1992), Modern Operating Systems, Prentice-Hall International Editions, New Jersey
- Tanenbaum, A.S., (1995), Distributed Operating Systems, Prentice Hall, New Jersey
- Treleaven, P.C., Brownbridge, D.R. ve Hopkins, R.P., (1982), "Data-Driven And Demand-Driven Computer Architectures", ACM Computing Surveys, Mar.: 93-143
- Wagner, T. ve Towsley, D. (1995), "Getting Started With POSIX Threads", Department of Computer Science, University of Massachusetts at Ahherst



SÖZLÜK

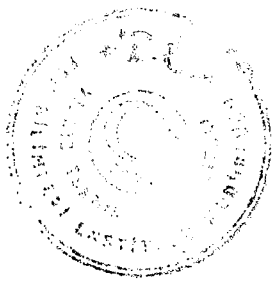
Abort	Terk Etmek
Acknowledge	Alındı
Address Space	Adres Alanı
Atomic	Bölünemez
Autonomous	Bağımsız
Blocking	Bloke
Buffer	Ara bellek
Centralized	Merkezi
Checksum	Sağlama Toplamı
Client	İstemci
Communication	İletişim
Computational Model	Hesaplama Modeli
Concurrent	Eşzamanlı
Conditional Variables	Koşul Değişkenleri
Connection Oriented	Bağlantılı
Connectionless	Bağlantısız
Consistency	Tutarlılık
Cooperative	Müşterek
Critical Sections	Kritik Bölgeler
Data	Veri
Distributed	Dağıtık
DNS (Domain Name System)	Etki Alanı İsim Sistemi
E-mail	Elektronik Posta
File	Dosya
Fine Grain	İnce Taneli
Flow	Akış
FNS (File Name Service / File Name Server)	Dosya İsim Servisi/Dosya İsim Sunucusu
FNSA (File Name Service Agent)	Dosya İsim Servis Ajanı
FNSC (File Name Service Client)	Dosya İsim Servis İstemcisi
FNSD (File Name Service Domain)	Dosya İsim Servisi Etki Alanı
Frame	Çerçeve
Function	Fonksiyon / Çağrı
Gateway	Ağ Geçişi
Gracefully	Nazikçe , Kurallara Uygun Biçimde
GUI (Graphical User Interface)	Grafik Kullanıcı Arayüzü
Hardware	Donanım
Header	Önbilgi
Heterogenious	Heterojen
Homogenious	Homojen
Host	Internet Adresi Olan Bilgisayar
Information	Bilgi
Instruction	Komut
Interconnection Network	Arabağlantı
Interface	Arayüz
Interrupt	Kesme
IP (Internet Protocol)	Internet Protokolü
IPC (Inter Process Communication)	İşlemler Arası İletişim
IPX (Internetwork Packet Exchange)	Ağlar Arası Paket Değişimi
ISO (International Standards Organization)	Uluslararası standart Organizasyonu
IT (Information Technology)	Bilgi Teknolojisi
Iterative	Tekrarlamalı
Kernel	İşletim Sistemi Çekirdeği
LAN (Local Area Network)	Yerel Alan Ağı
Large Grain	Geniş Taneli
Layer	Katman

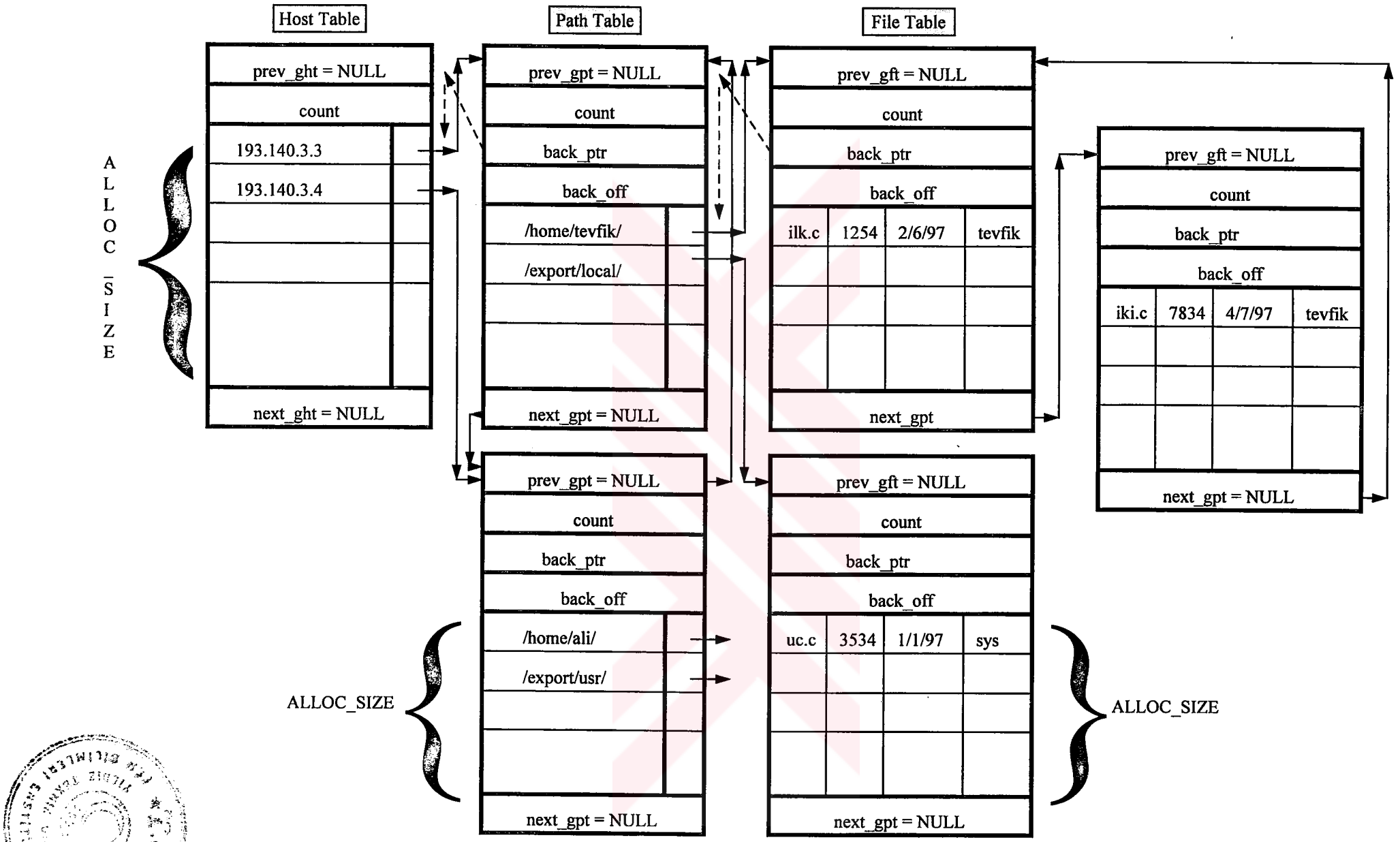
LFS (Local File System)	Yerel Dosya Sistemi
Lock	Kilitleme
Logical	Mantıksal
Loosly Coupled	Gevşek Bağlı
LWP (Light Weight Process)	Hafif Sıklet İşlem
Mainframe	Anabilgisayar
Master	Ana
MIMD(Multiple Instruction Multiple Data)	Çok Komut Çok Veri
MIPS (Million Instruction Per Second)	Saniyede yapılan milyon işlem sayısı
MISD (Multiple Instruction Single Data)	Çok Komut Tek Veri
Monolithic	Tek Kontrol Merkezli
Mounted	Bağlantılı
Multi Process	Çok İşlemli
Multicomputer	Çok Bilgisayarlı
Multiprocessor	Çok İşlemcili
MultiTasking	Çok İşli
MultiThread	Çok İşlem Parçacıklılığı / Çok Liflilik
Mutex (Mutual Exclusion)	Karşılıklı Dışlama
Network	Ağ
NFS (Network File System)	Ağ Dosya Sistemi
Nonblocking	Bloke Olmayan
Object Code	Nesne Kodu
On-Line	Çevrim İçi
OSI (Open Systems Interconnection)	Açık Sistem Bağlantısı
Path	Dizin
PC (Personal Computer)	Kişisel Bilgisayar
Performance	Başarım
Pipeline	İş Hattı
Primary	Birincil
Primary Storage	Birincil Hafıza
Procedure	Yordam
Process	İşlem
Program	Program
Protection	Koruma
Race Conditions	Yarış Koşulları
Receiver	Alıcı
Refresh	Tazeleme
Relative	Bağıl
Reliability	Güvenilirlik
Reliable	Güvenilir
Replicated	Kopyalanmış
Reply	Yanıt
Request	İstek
Response Time	Cevap Süresi
RFS (Remote File System)	Uzak Dosya Sistemi
Routing	Rotalama
RPC (Remote Procedure Call)	Uzak Yordam Çağırma
Scaleability	Ölçeklenebilirlik
Secondary Storage	İkincil Hafıza
Security	Güvenlik
Semaphore	Semafor
Sender	Verici / Yollayıcı
Sequenced	Sıralı
Server	Sunucu
SFNS (Secondary File Name Server)	İkincil Dosya İsim Sunucusu
SIMD(Single Instruction Multiple Data)	Tek Komut Çok Veri
Simultaneous	Anında
Single Process	Tek işlemli



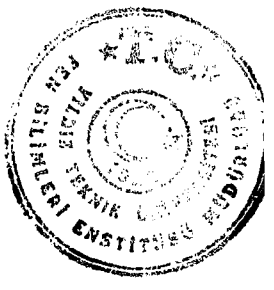
SISD (Single Instruction Single Data)	Tek Komut Tek Veri
Slave	Uydu
SM (Shared Memory)	Paylaşılan Hafıza
SNA(Systems Network Architecture)	Sistem Ağ Mimarisi
Socket	Soket
Software	Yazılım
SPX (Sequenced Packet Exchange)	Sıralı Paket Değişimi
State	Durum
Stub	Saplama
Synchronization	an uyumluluk
System Call	Sistem Çağrısı
TCP (Transmission Control Program)	İletim Kontrol Programı
Temporary	Geçici
Thread	İş/işlem Parçacığı
Tightly Coupled	Sıkı Bağlı
TLI(Transport Layer Interface)	Taşıma Katmanı Arayüzü
Traffic	Trafik
Trailer	art bilgi
Transaction	Hareket
Transparent	Şeffaf
Transport Address	Taşıma Adresi
Transport End-Point	Taşıma Noktası
Transporter	Taşıyıcı
TSL (Test And Set Lock)	Kontrol Et ve Kilitle
TTL (Time To Live)	Yaşam Süresi
UDP (User Datagram Protocol)	Kullanıcı Datagram Protokolü
Unbuffered	Ara Bellek Kullanmayan
Unlock	Kilidi Açma
Unmounted	Bağlantısı kesilmiş
Unreliable	Güvenilmez
Up to Date	Güncel
WAN (Wide Area Network)	Geniş Alan Ağı
Window	Pencere
Work Flow	İş Akışı
Work Load	İş yükü
Workstation	İş İstasyuonu

EK - 1 Tablolar arası ilişkiler

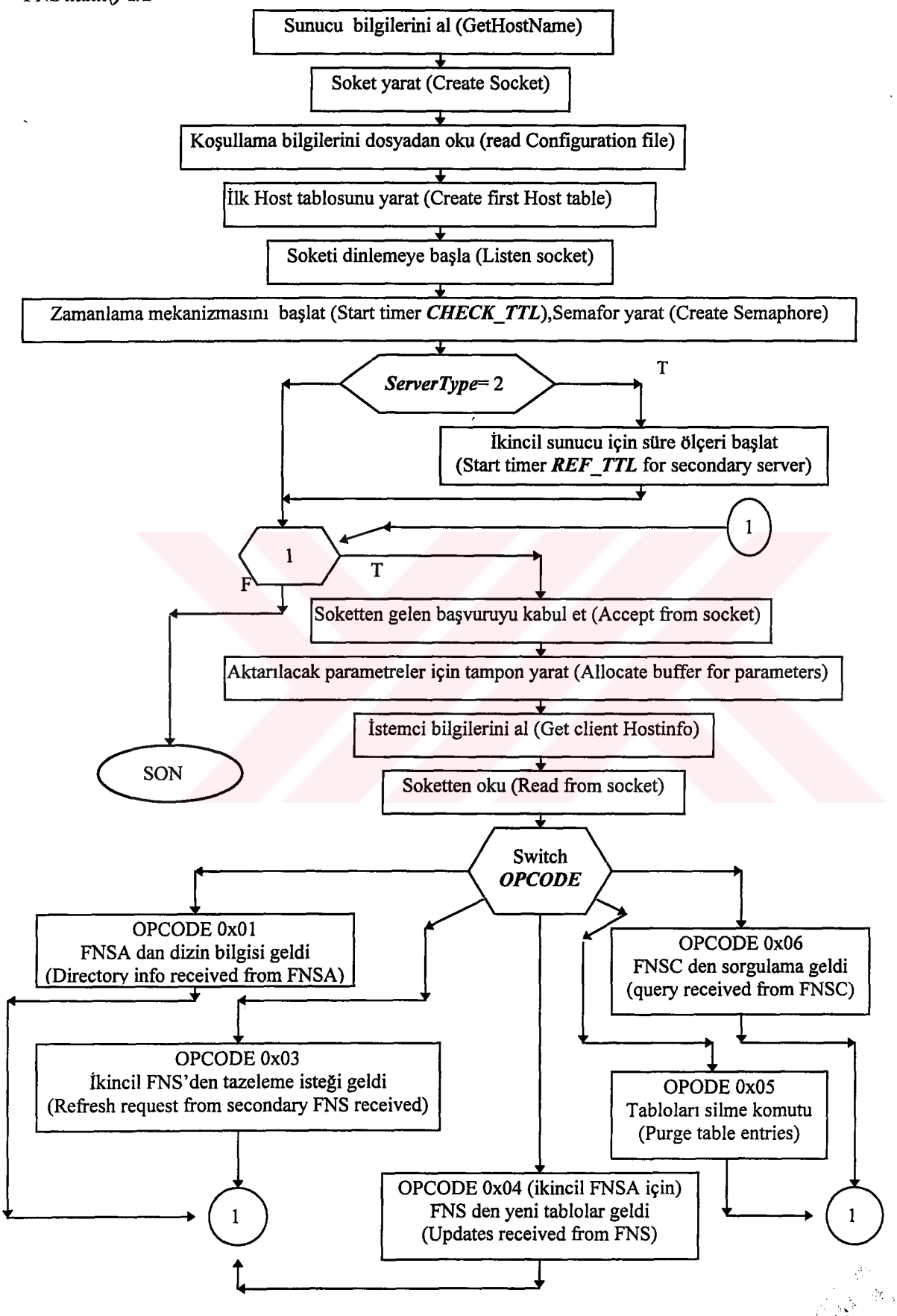



Host Table
Path Table
File Table
prev_ght = NULL
count
193.140.3.3
193.140.3.4
next_ght = NULL
ALLOC_SIZE
prev_gpt = NULL
count
back_ptr
back_off
/home/tevfik/
/export/local/
next_gpt = NULL
prev_gft = NULL
count
back_ptr
back_off
ilk.c
1254
2/6/97
tevfik
next_gpt
prev_gft = NULL
count
back_ptr
back_off
iki.c
7834
4/7/97
tevfik
next_gpt = NULL
prev_gpt = NULL
count
back_ptr
back_off
/home/ali/
/export/usr/
ALLOC_SIZE
next_gpt = NULL
prev_gft = NULL
count
back_ptr
back_off
uc.c
3534
1/1/97
sys
ALLOC_SIZE
next_gpt = NULL

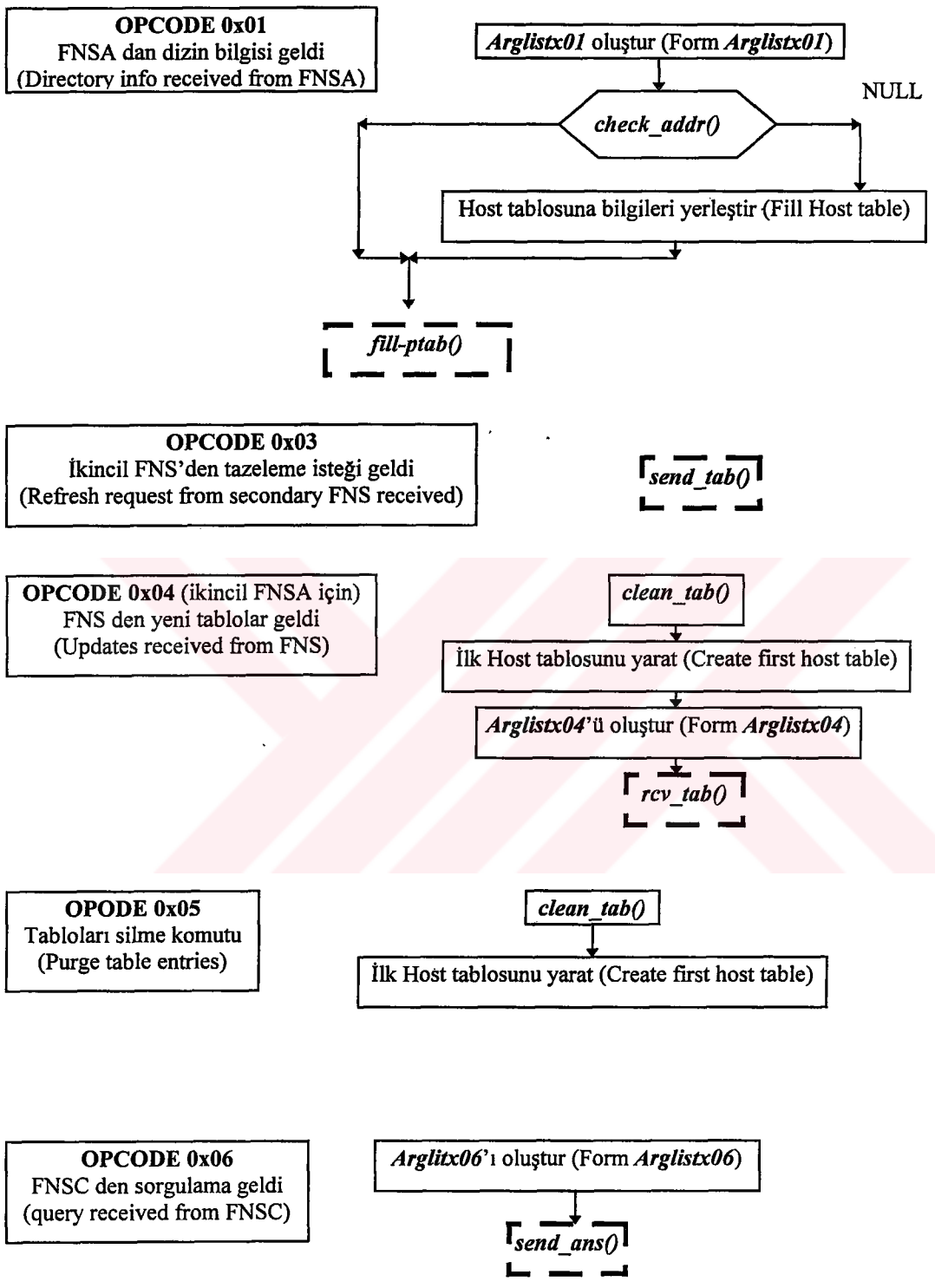
EK - 2 FNS ile ilgili yordamların blok akış diyagramları



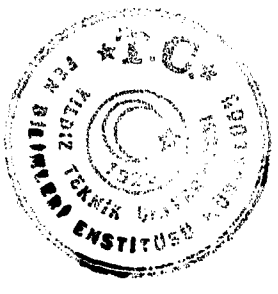
FNS *main()* 1/2


Sunucu bilgilerini al (GetHostName)
Soket yarat (Create Socket)
Koşullama bilgilerini dosyadan oku (read Configuration file)
İlk Host tablosunu yarat (Create first Host table)
Soketi dinlemeye başla (Listen socket)
Zamanlama mekanizmasını başlat (Start timer CHECK_TTL),Semafor yarat (Create Semaphore)
ServerType= 2
T
İkincil sunucu için süre ölçeri başlat (Start timer REF_TTL for secondary server)
1
1
T
F
Soketten gelen başvuruyu kabul et (Accept from socket)
Aktarılacak parametreler için tampon yarat (Allocate buffer for parameters)
İstemci bilgilerini al (Get client Hostinfo)
SON
Soketten oku (Read from socket)
Switch OPCODE
OPCODE 0x01
FNSA dan dizin bilgisi geldi
(Directory info received from FNSA)
OPCODE 0x06
FNSC den sorgulama geldi
(query received from FNSC)
OPCODE 0x03
İkincil FNS'den tazeleme isteği geldi
(Refresh request from secondary FNS received)
OPODE 0x05
Tabloları silme komutu
(Purge table entries)
1
OPCODE 0x04 (ikincil FNSA için)
FNS den yeni tablolar geldi
(Updates received from FNS)
1

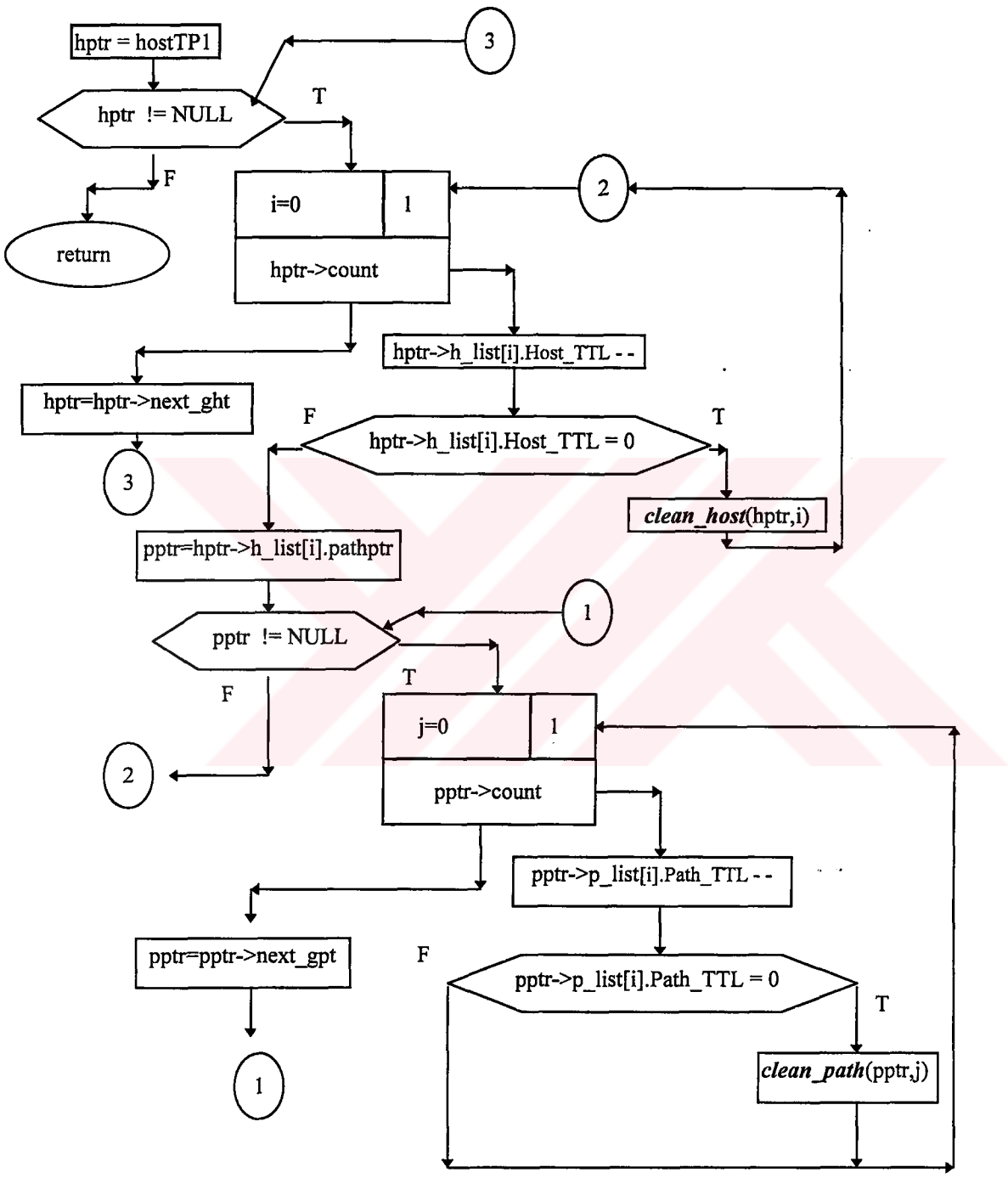
FNS *main()* 2/2



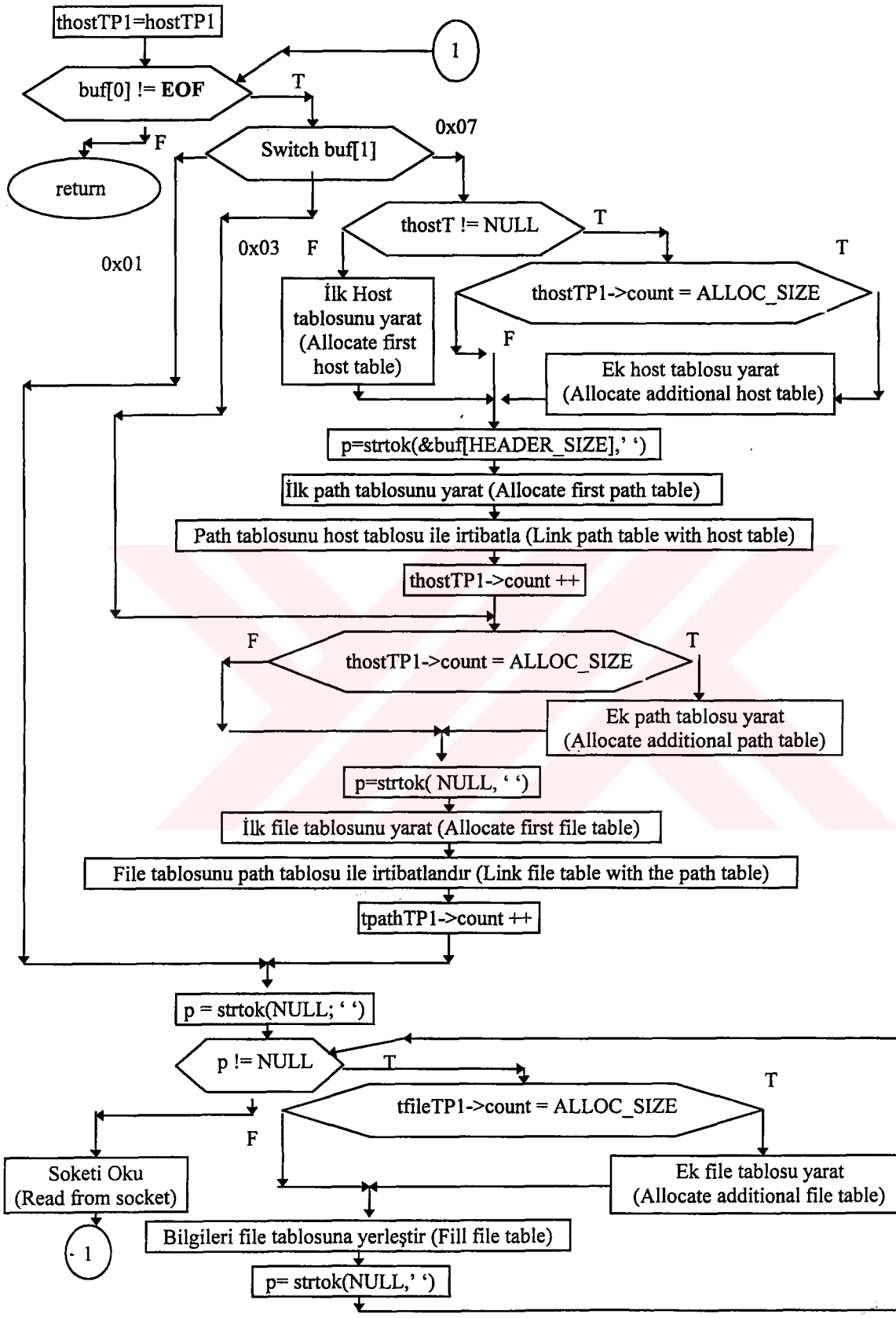
— — — Kritik alanlara erişen parçaları belirler.



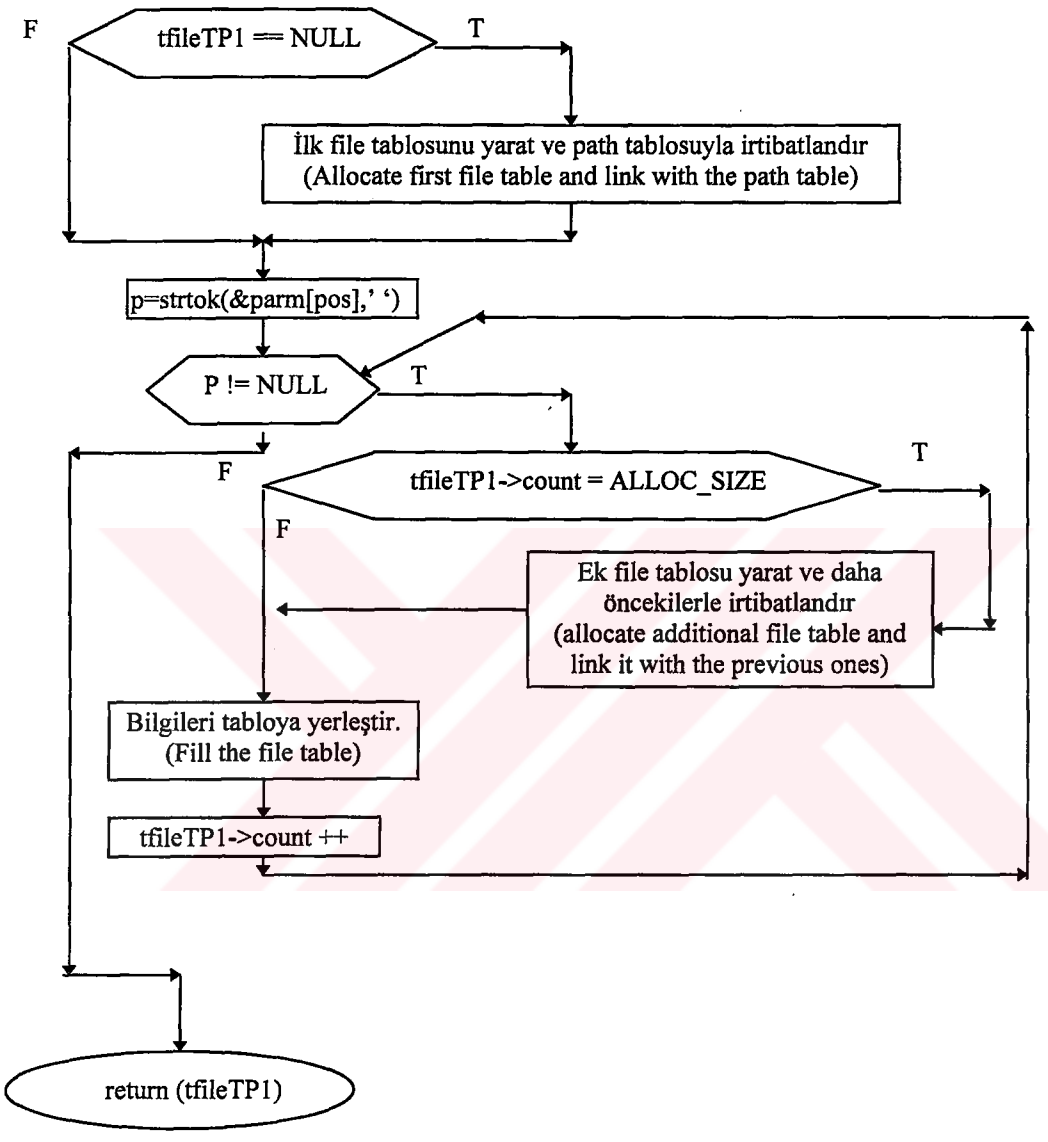
FNS *dec_TTL()*


hptr = hostTP1
3
hptr != NULL
T
F
return
i=0
1
2
hptr->count
hptr->h_list[i].Host_TTL - -
hptr=hptr->next_ght
F
hptr->h_list[i].Host_TTL = 0
T
3
clean_host(hptr,i)
pptr=hptr->h_list[i].pathptr
1
pptr != NULL
T
F
j=0
1
2
pptr->count
pptr->p_list[i].Path_TTL - -
pptr=pptr->next_gpt
F
pptr->p_list[i].Path_TTL = 0
T
1
clean_path(pptr,j)

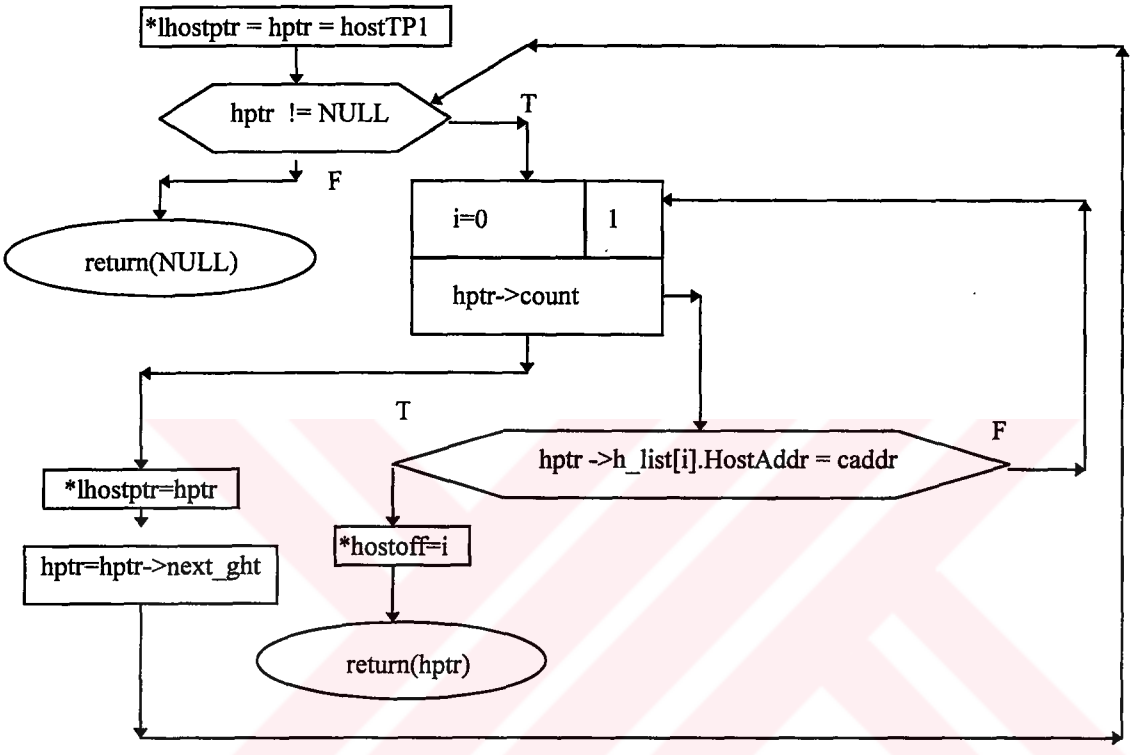
FNS ***rcv_tab(ChildBufx04 *parm)***



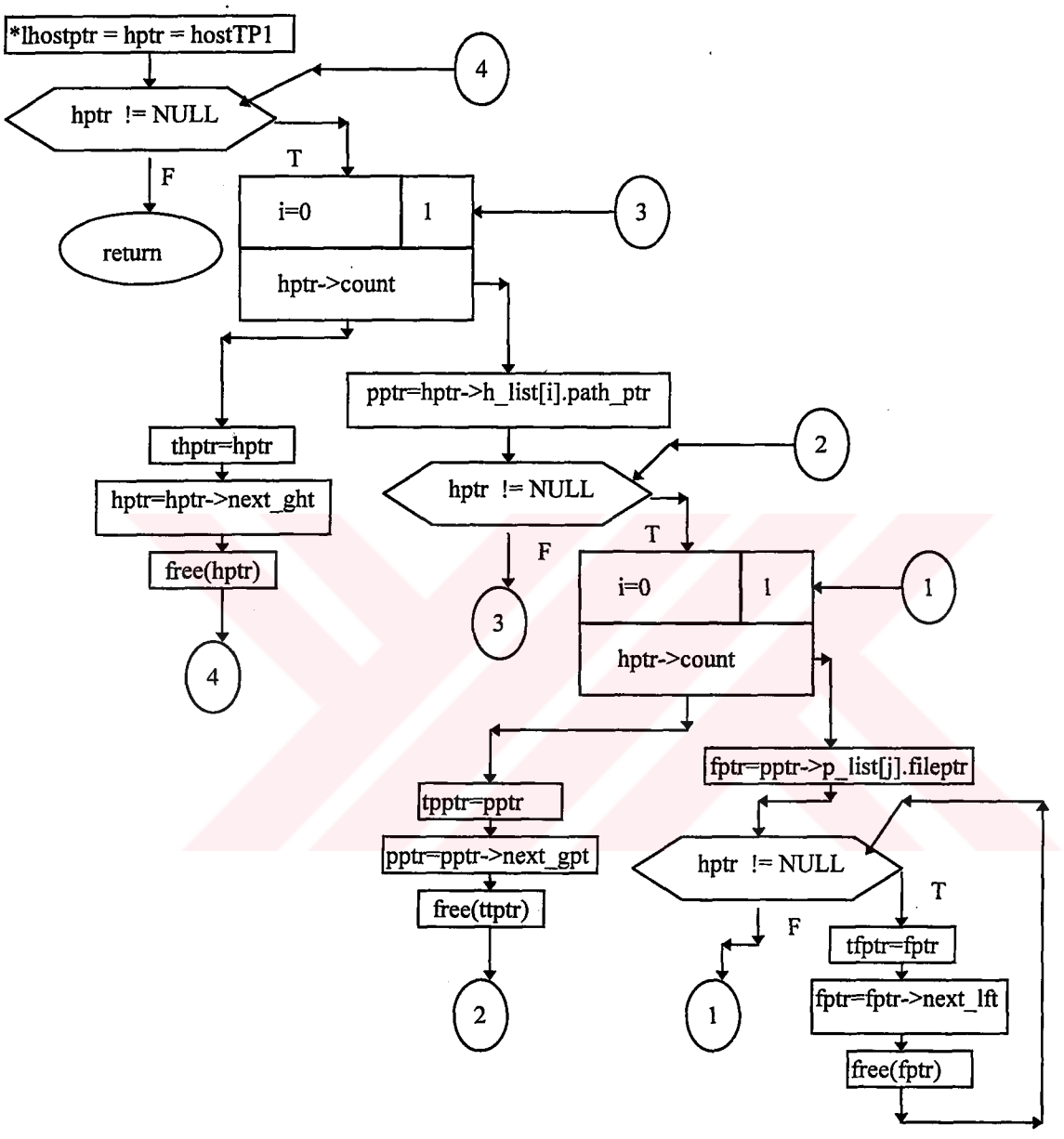
FNS ***Ftable *fill_ftab(int pos, char *parm, PTable *tpathTP1, int tpof, Ftable *tfileTP1*** **)**



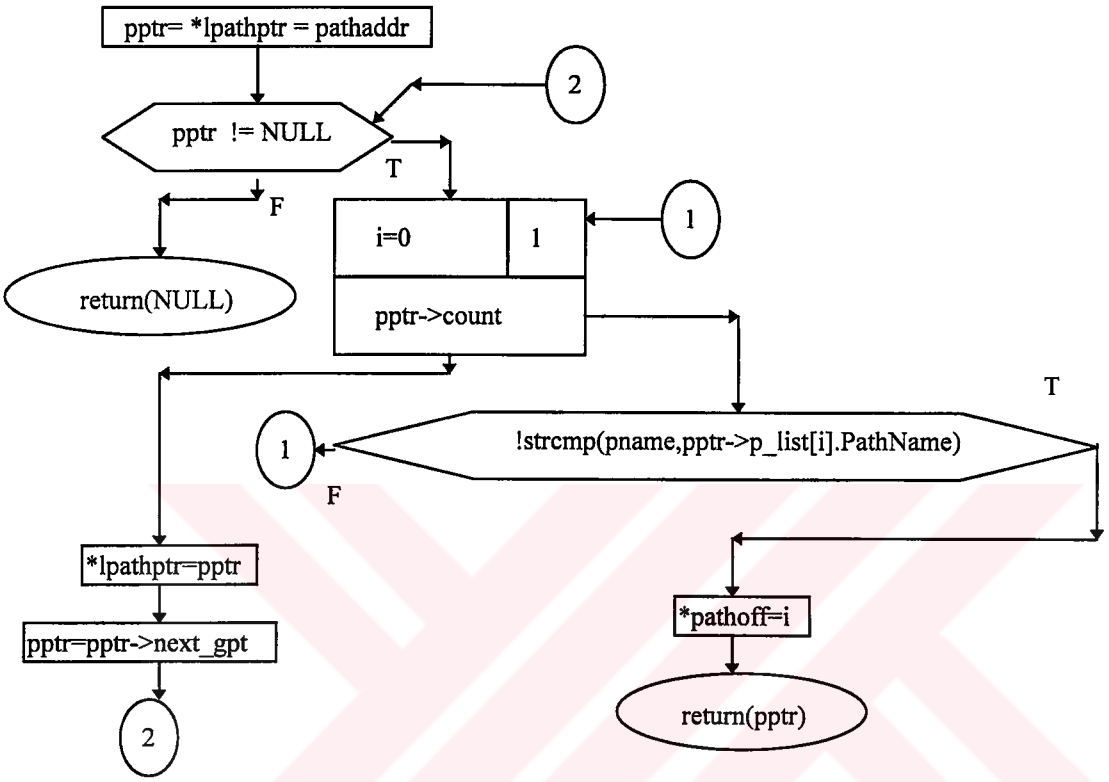
FNS ***Htable *check_haddr(unsigned long caddr, int *hostoff, HTable **lhostptr)***



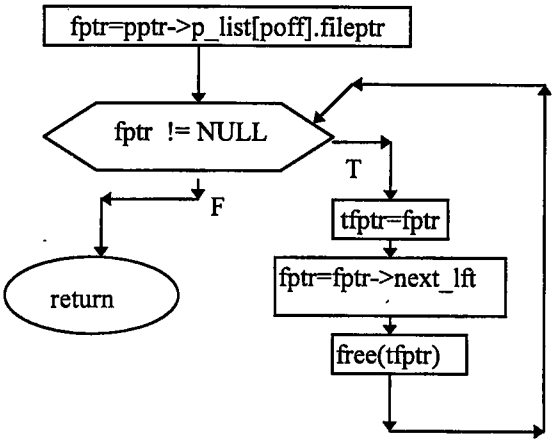
FNS *clean_all()*



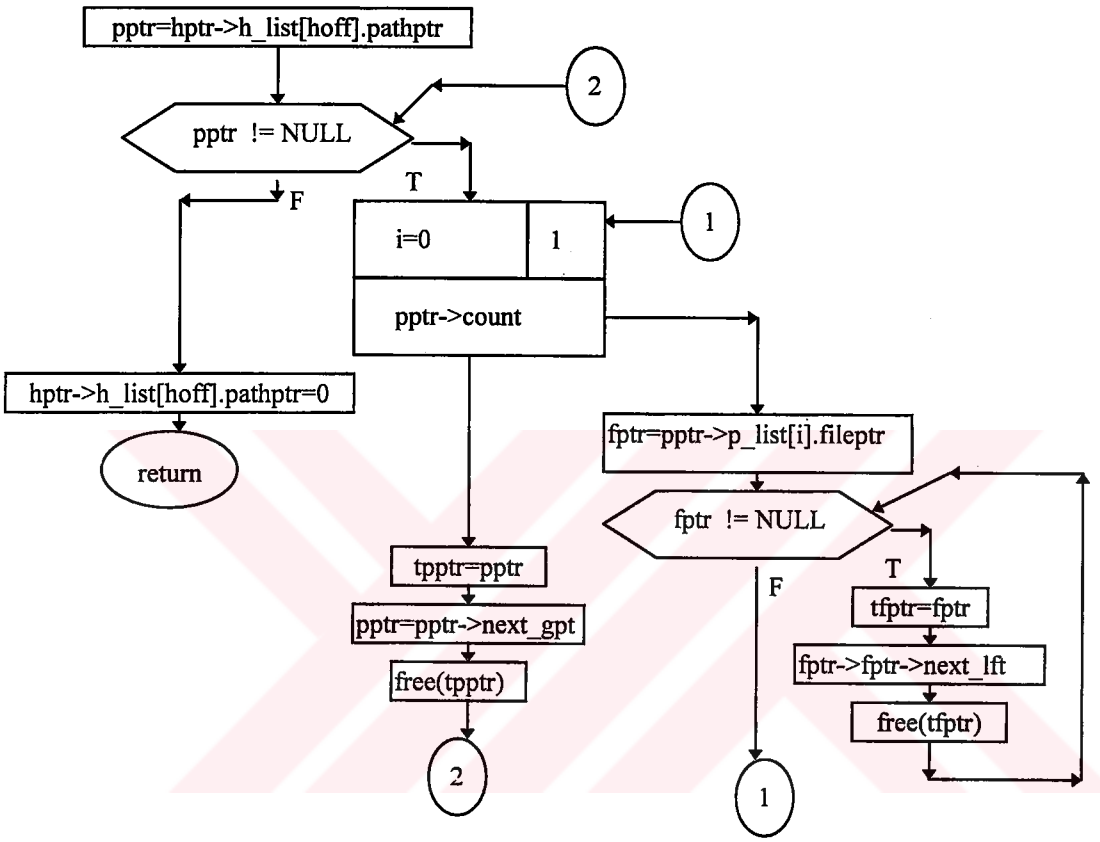
FNS ***Ptable *check_paddr(PTable *pathaddr, char *pname, int *pathoff, Ptable **lpathptr)***



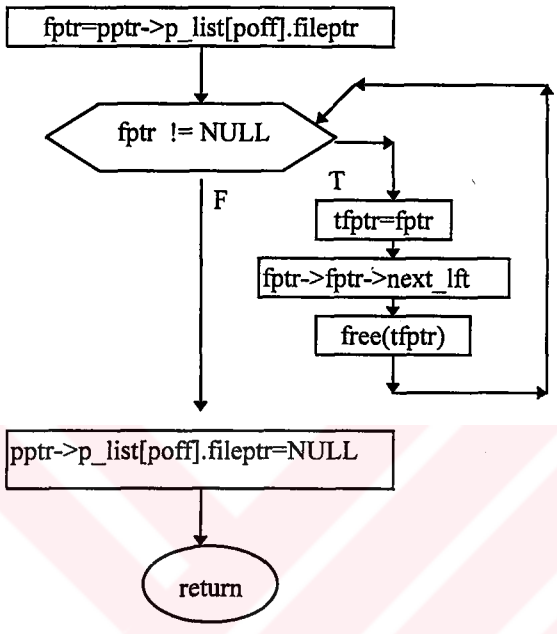
FNS ***clean_ftab(PTable *pptr, inf poff)***



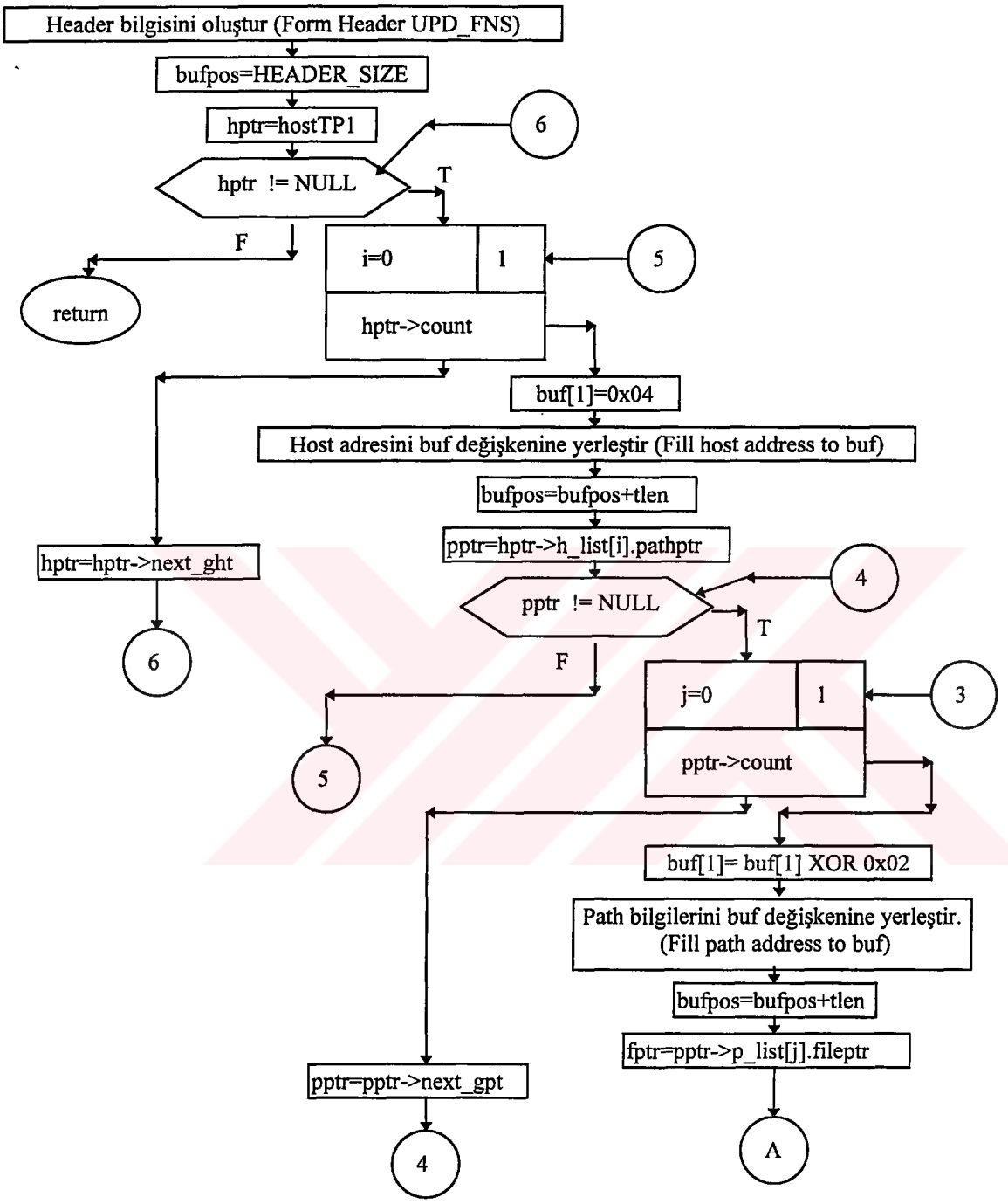
FNS ***clean_host(HTable *hptr, int hoff)***



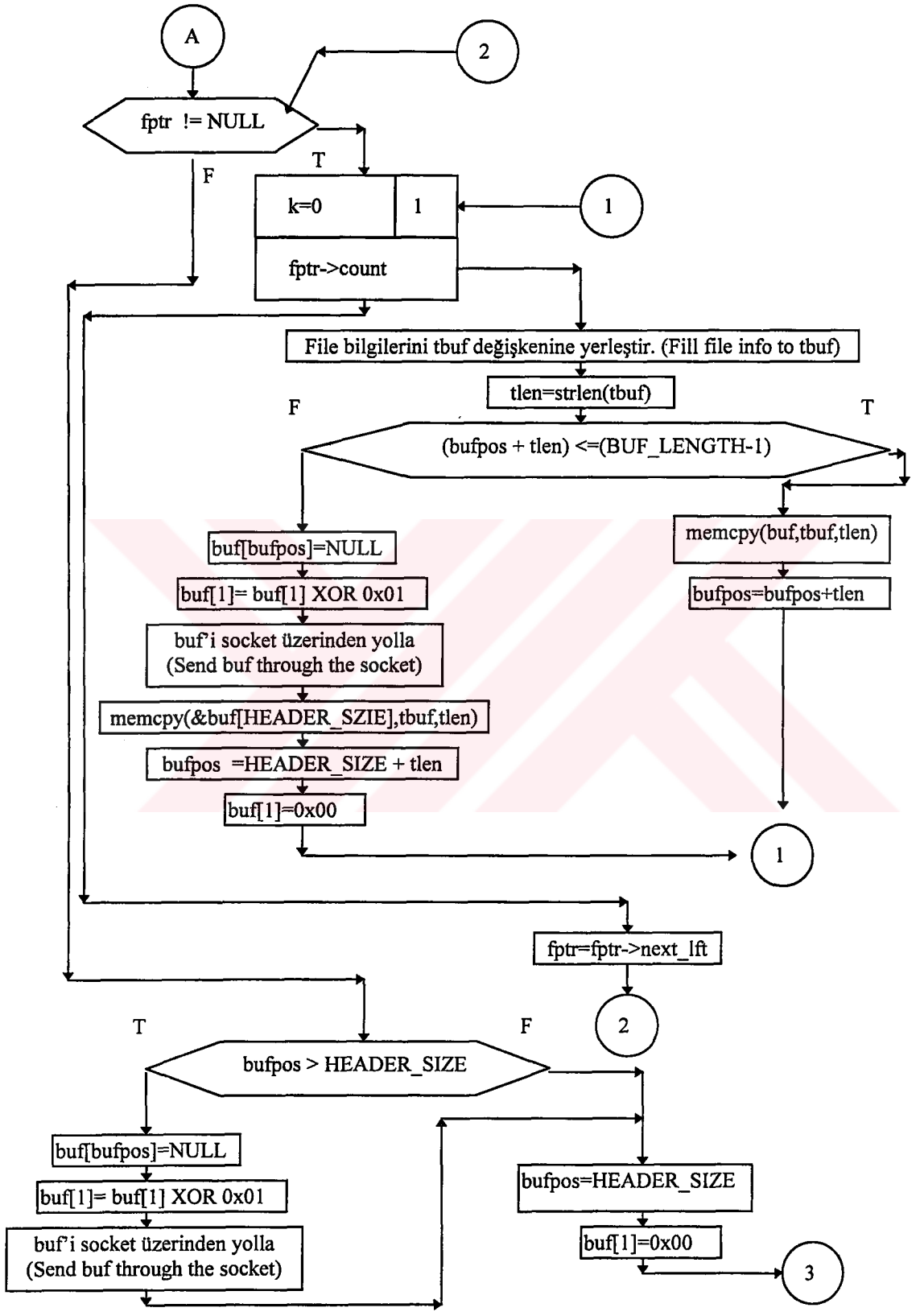
FNS ***clean_path(PTable *pptr, int poff)***



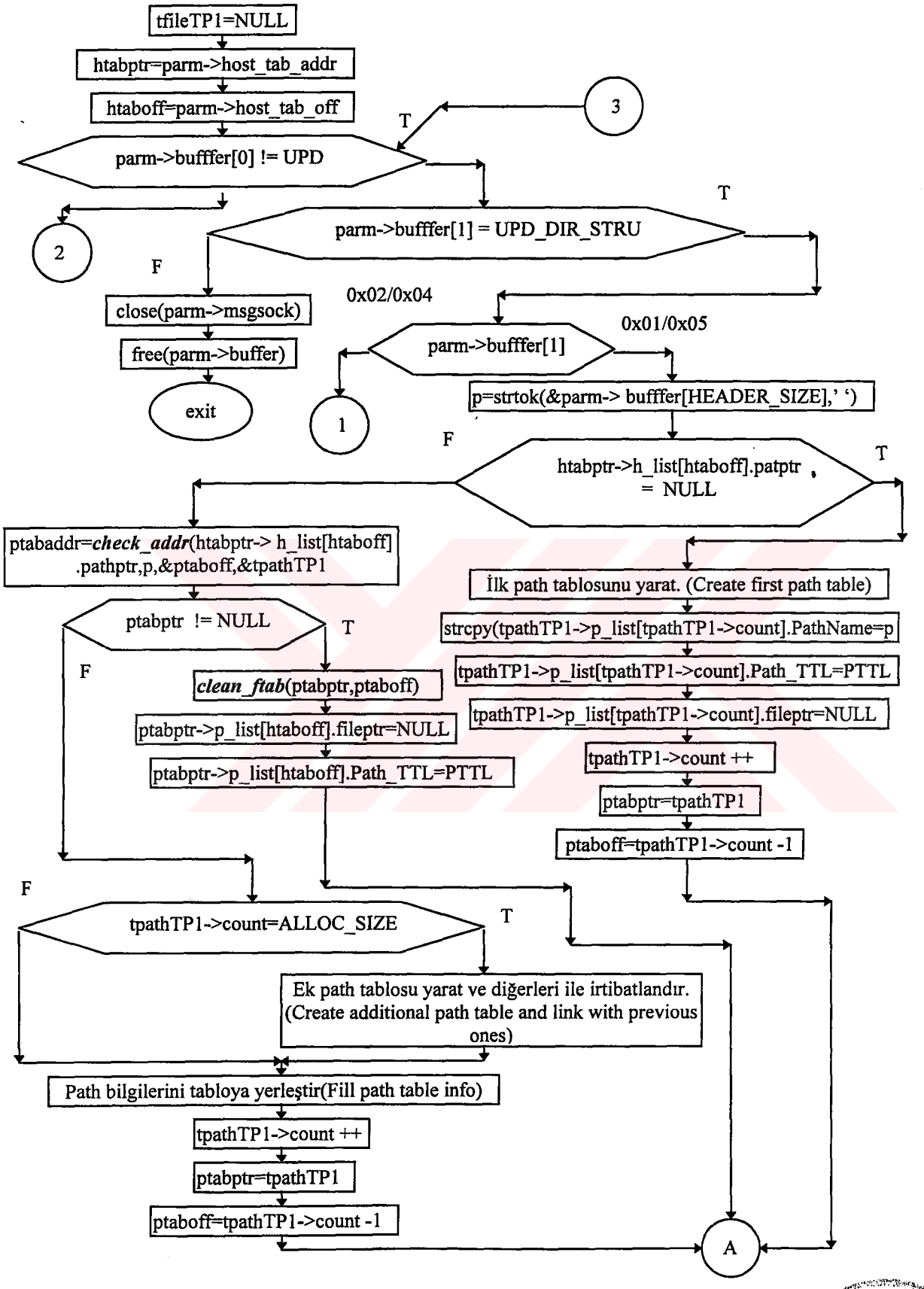
FNS *send_tab(void *msock)* 1/ 2



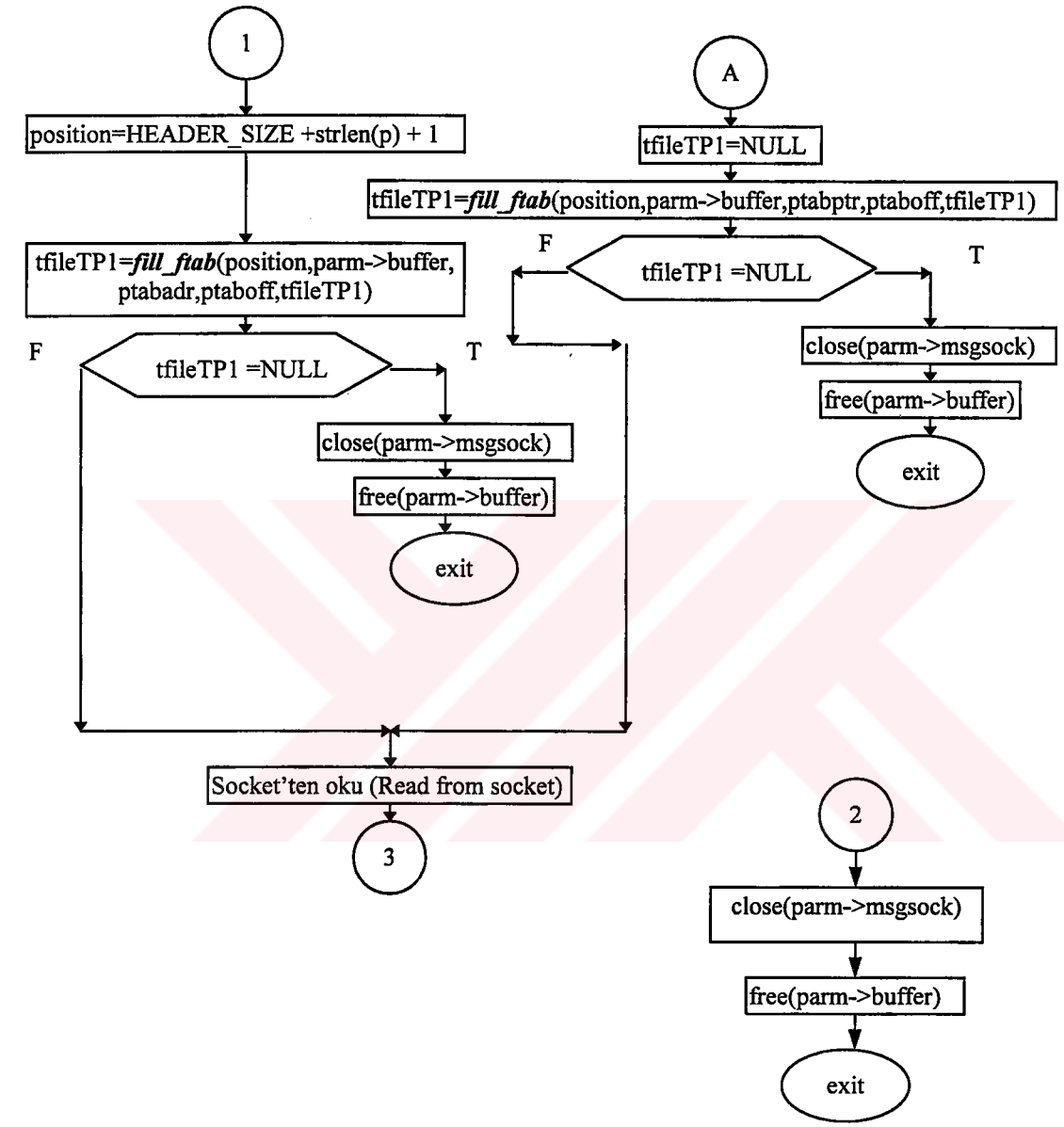
FNS *send_tab(void *msock)* 2/ 2

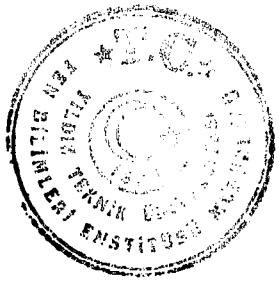


FNS *fill_ptab(ChildBufx01 *parm)* 1/2



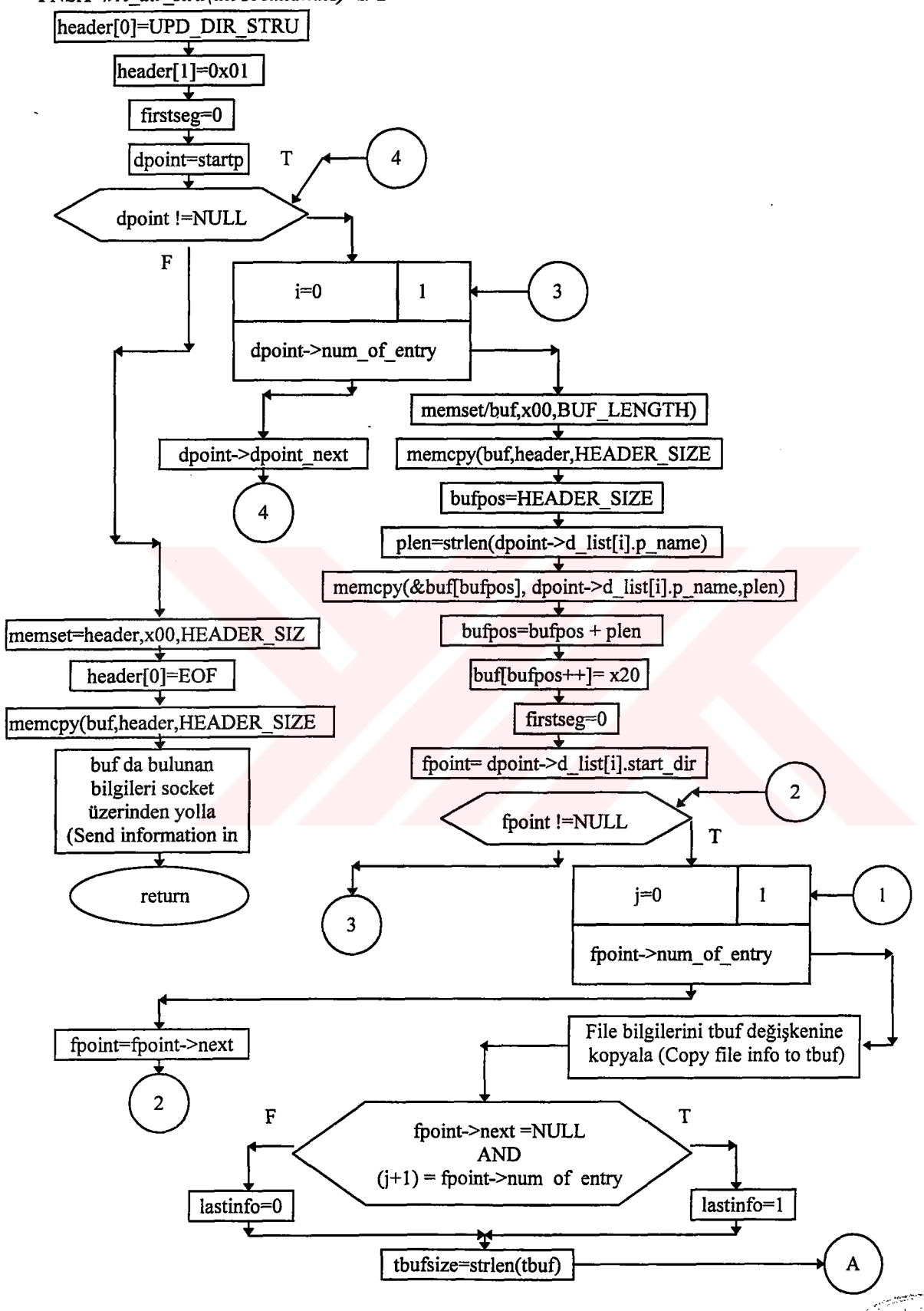
FNS *fill_ptab(ChildBufx01 *parm)* 2 /2



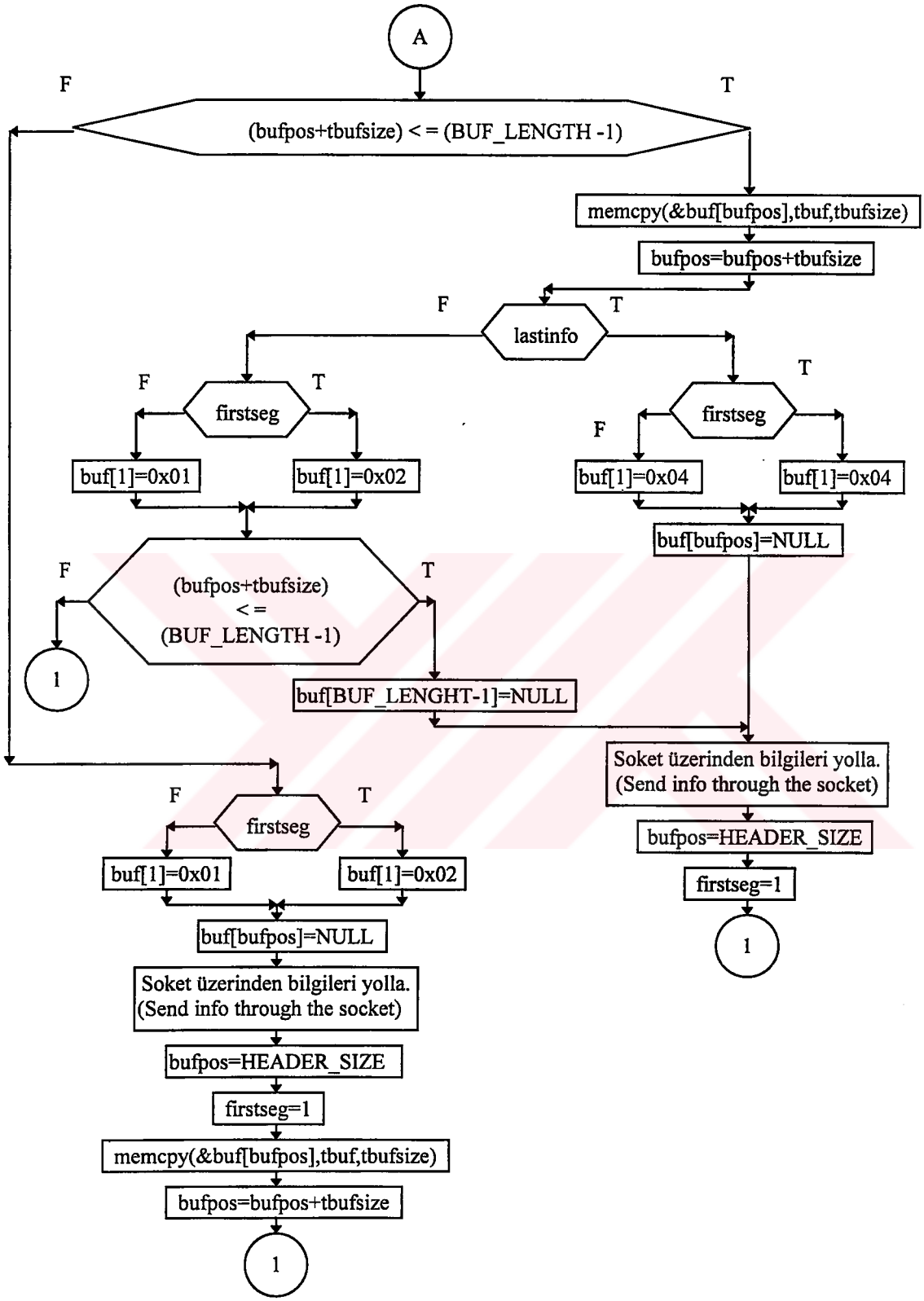


EK - 3 FNSA ile ilgili yordamların blok akış diyagramları

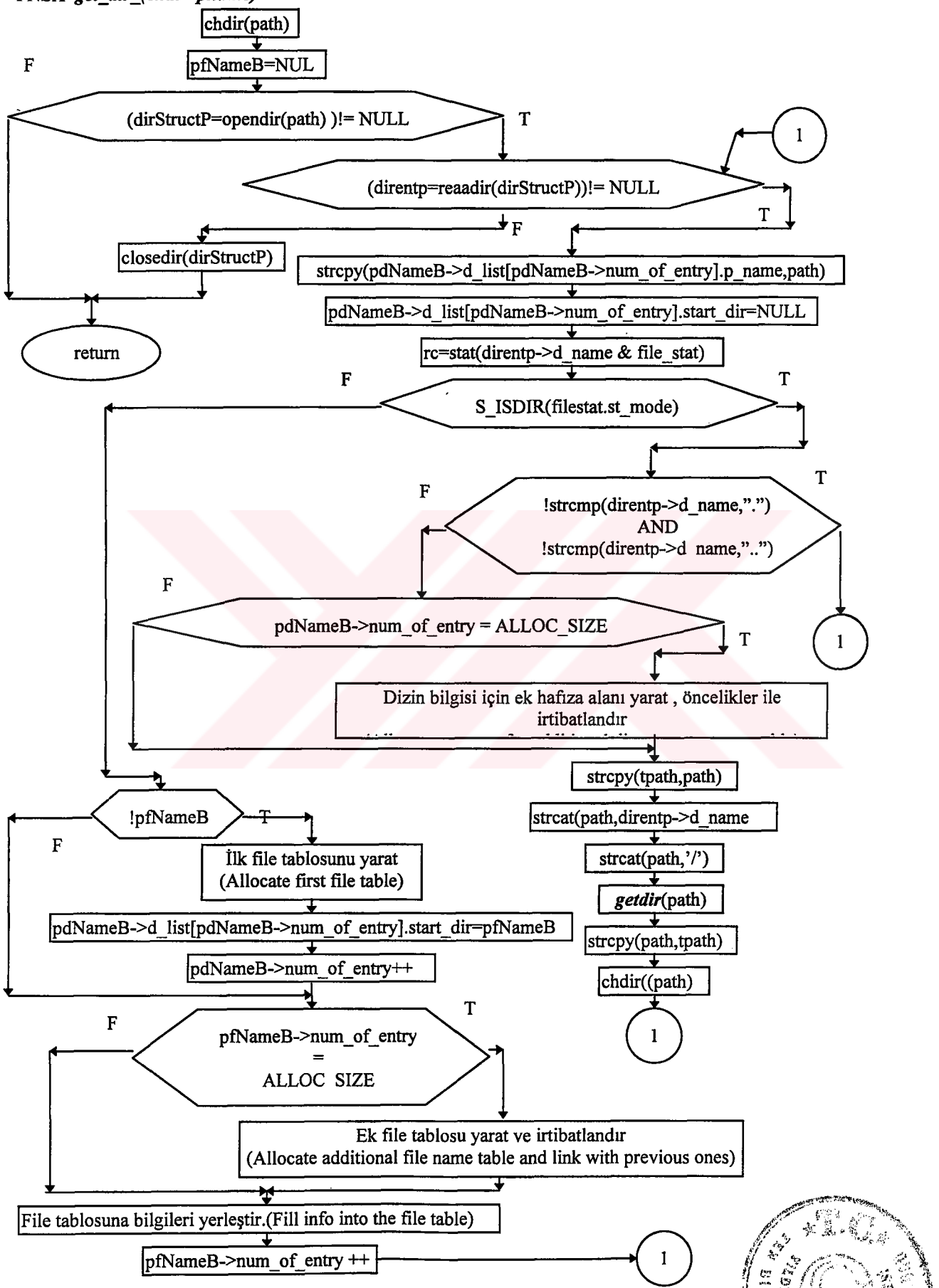
FNSA ***wrt_dir_stru(int sockhandle)*** **1/ 2**



FNSA ***wrt_dir_stru(int sockhandle)*** **2 /2**

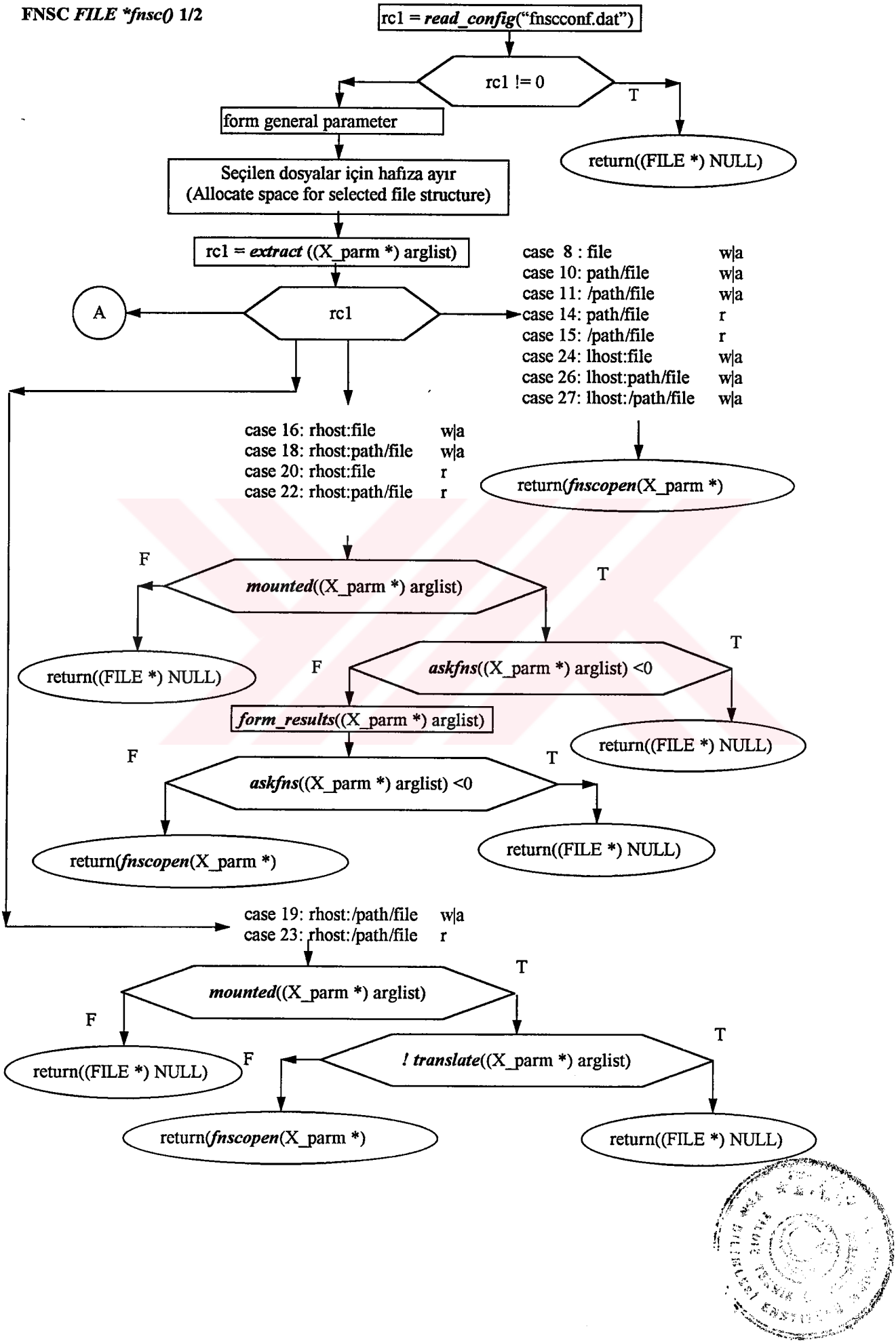


FNSA ***get_dir_(char *pname)***

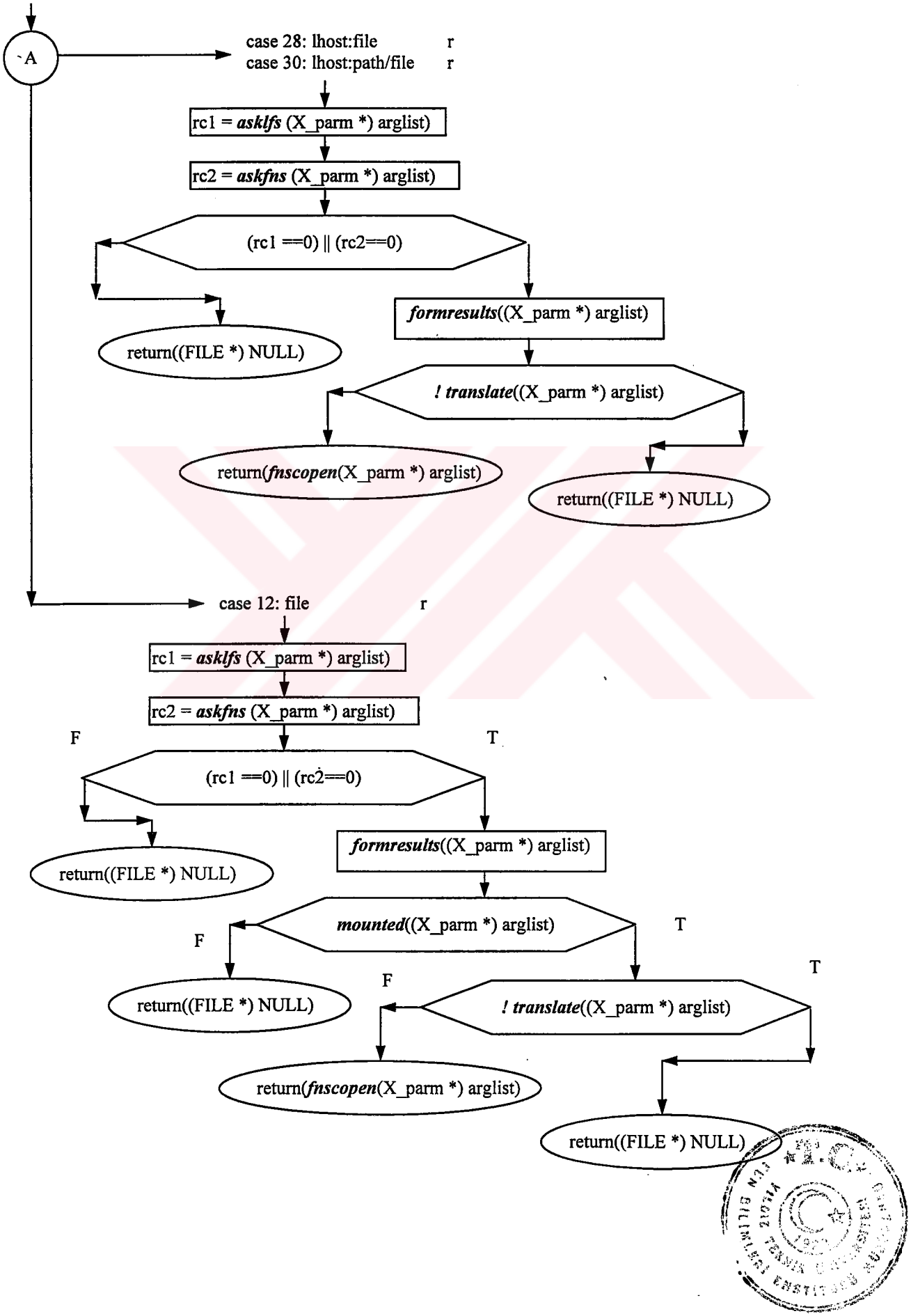


EK - 4 FNSC ile ilgili yordamın blok akış diyagramı

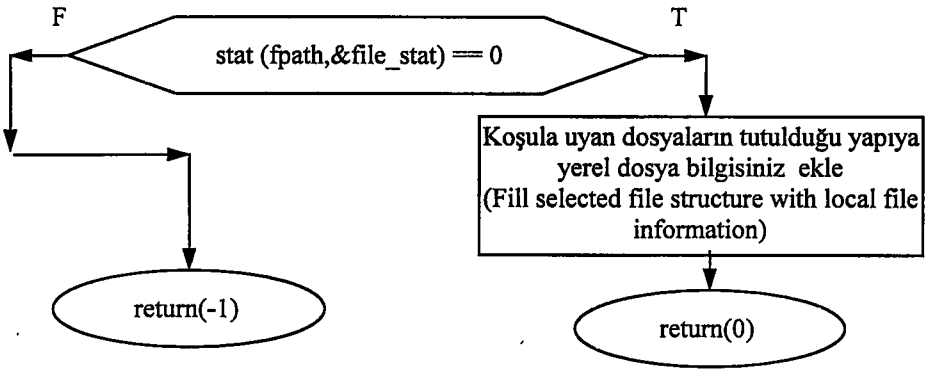
FNSC *FILE *fnsc()* 1/2



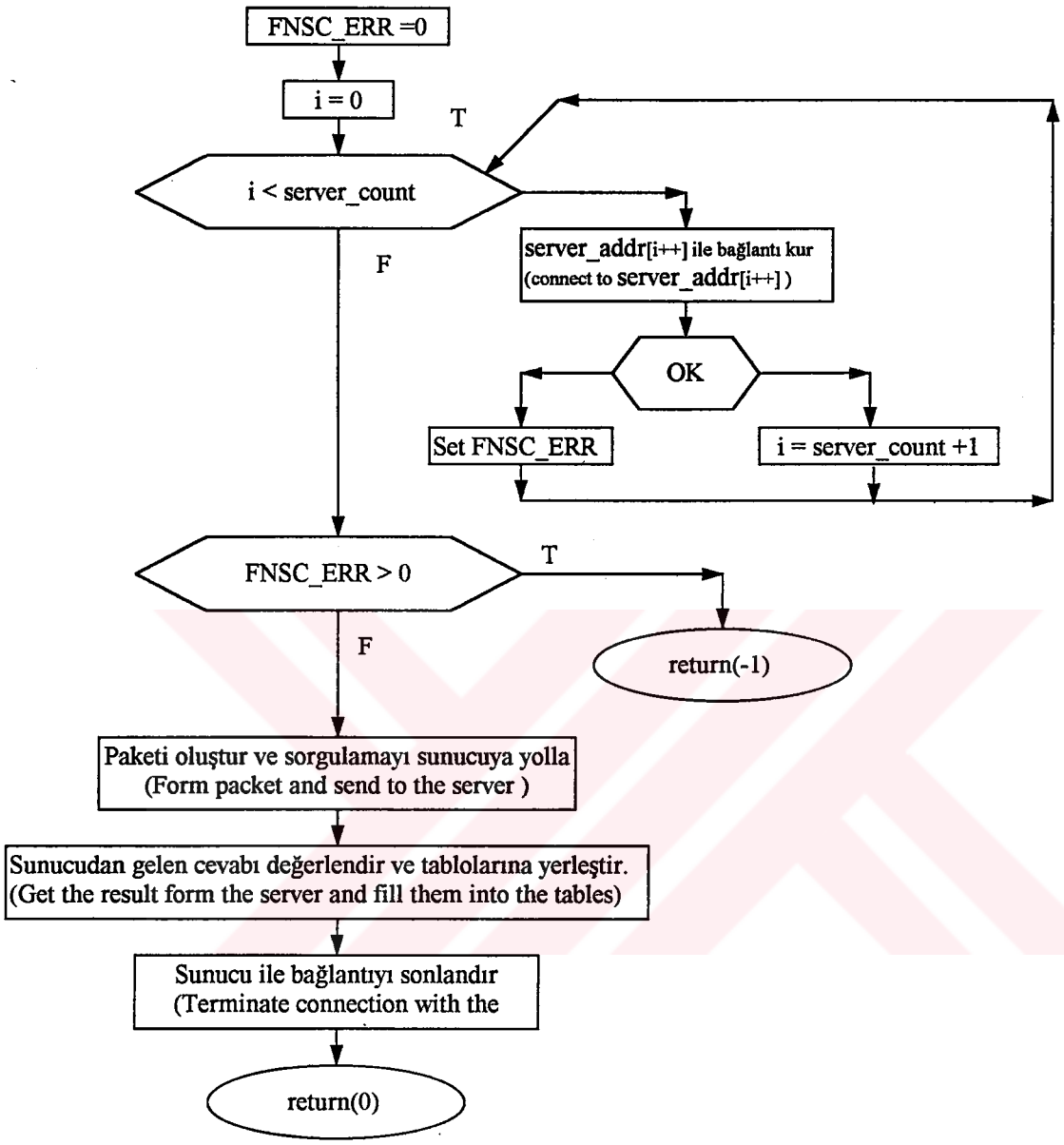
FNSC ***FILE *fnsc() 2/2***



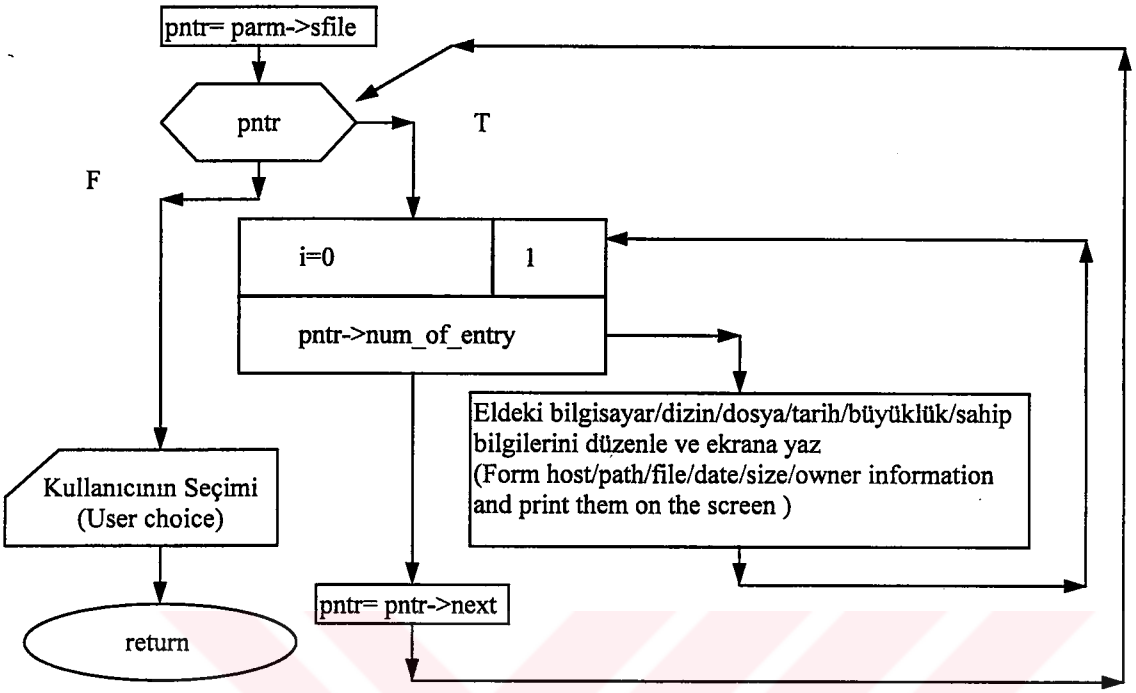
FNSC ***int asklfs ((X_parm * parm)***



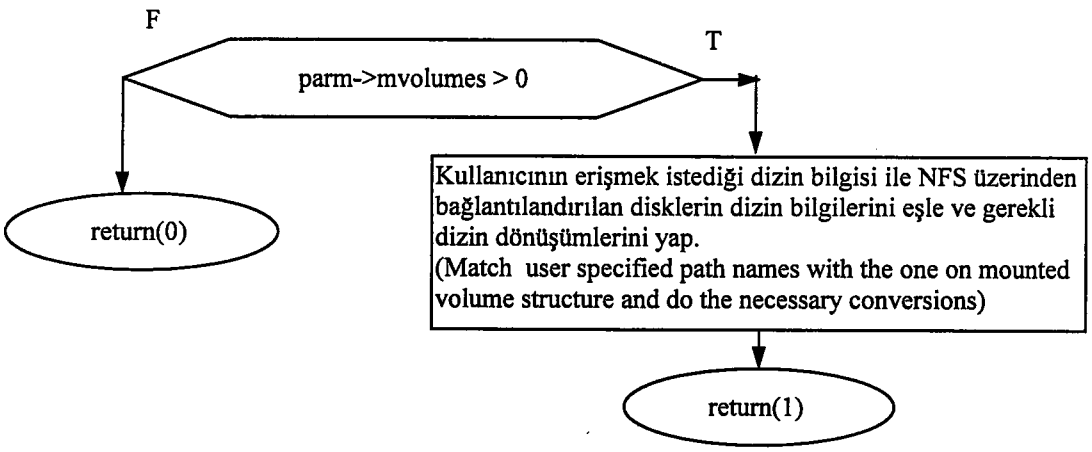
FNSC ***int askfns (X_parm *parm)***



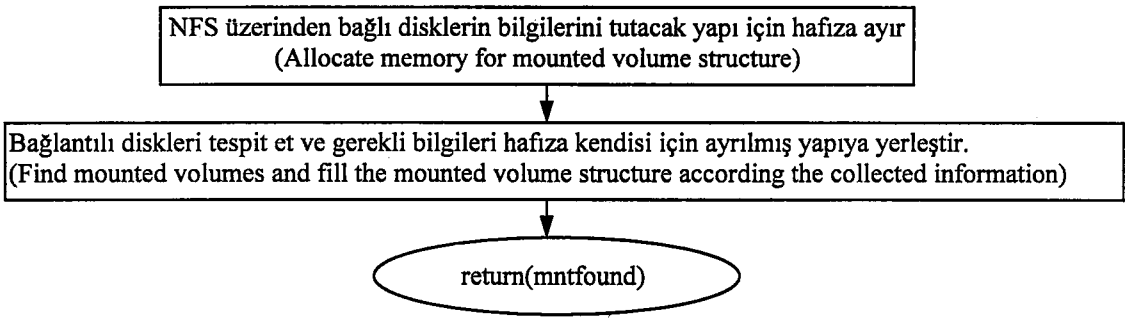
FNSC ***form_results (X_parm *parm)***



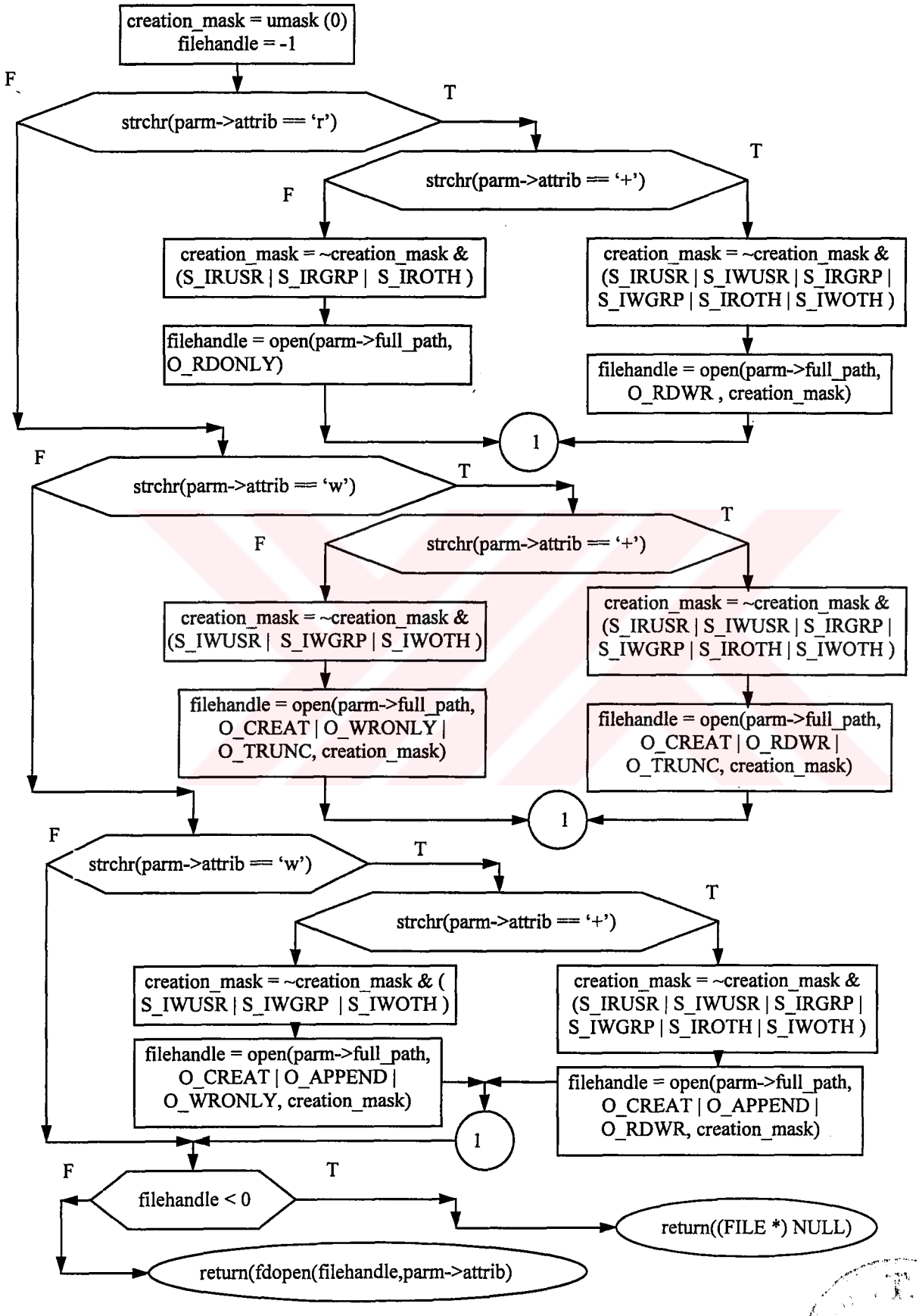
FNSC ***int translate (X_parm *parm)***



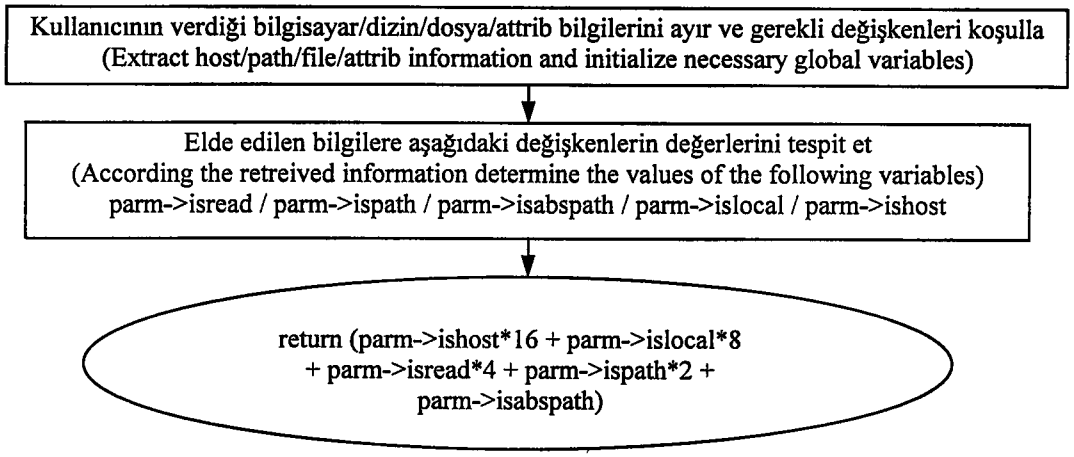
FNSC ***int mounted (X_parm *parm)***



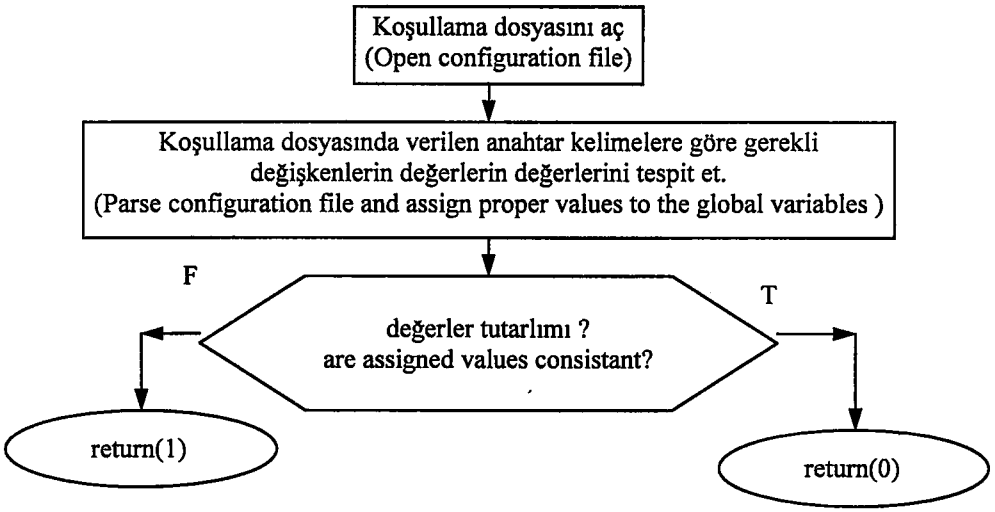
FNSC *FILE *fnscopen (X_parm *parm)*



FNSC ***extract (X_parm *parm)***



FNSC ***int read_config (char * conffile)***



ÖZGEÇMİŞ

Soyad	İNAN
Ad	Ahmet Tevfik
Doğum Tarihi	25 Temmuz 1967
Doğum Yeri	Ankara

Orta Okul	1979-1981	A.Ö.D. Tevfik Fikret Lisesi (Orta Kısım) - Ankara
Lise	1981-1984	Ankara Lisesi - Ankara
Lisans	1984-1988	Yıldız Üniversitesi Mühendislik Fakültesi Bilgisayar Bilimleri ve Mühendisliği Bölümü
Yüksek Lisans	1988-1991	Yıldız Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Bilimleri Mühendisliği Ana bilim Dalı
Doktora	1991-1997	Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Bilimleri Mühendisliği Ana bilim Dalı

Çalıştığı Kurumlar

1986 (6 hafta)	T.C.İş Bankası G.Md. Ankara -Stajyer
1987 (9 hafta)	Scicom International HW / SW Dept. Colchester Essex (İngiltere) - Stajyer
1990 (2 ay)	Northern Telecom Inc.DMS-10 Patch Div.Raleigh NC (A.B.D.) -Stajyer
1988-Devam ediyor	Yıldız Teknik Üniversitesi Elektrik Elektronik Fakültesi Bilgisayar Bilimleri ve Mühendisliği Bölümü - Araştırma Görevlisi