DOSYA TAKİP PROGRAMI :DOST

133276

İsmail KURNAZ

YÜKSEK LİSANS TEZİ
(BİLGİSAYAR EĞİTİMİ)

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

133276

T.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ

Ağustos 2003
ANKARA

İsmail KURNAZ tarafından hazırlanan DOSYA TAKİP PROGRAMI (DOST) adlı bu tezin yüksek lisans tezi olarak uygun olduğunu onaylarım.

Prof. Dr. Doğan ÇALIKOĞLU
Tez Yöneticisi

Bu çalışma, jürimiz tarafından BİLGİSAYAR EĞİTİMİ Anabilim Dalında yüksek lisans tezi olarak kabul edilmiştir.

Başkan: : Prof. Dr. Doğan ÇALIKOĞLU

Üye : Doç. Dr. Abdullah ÇAVUŞOĞLU

Üye : Yrd. Doç. Dr. Halil İbrahim BÜLBÜL

Üye : _____

Üye : _____

Bu tez, Gazi Üniversitesi Fen Bilimleri Enstitüsü tez yazım kurallarına uygundur.

İÇİNDEKİLER

DOSYA TAKİP PROGRAMI : DOST

(Yüksek Lisans Tezi)

İsmail KURNAZ

GAZİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

Ağustos 2003

ÖZET

Bu çalışmada, Windows işletim sistemleri altında çalışan uygulamaların listesini kullanıcıya gösteren DOST isimli bir dosya takip programı geliştirilmiştir. Sistemde çalıştırılan uygulamaların listesi bazı özel kütüphane fonksiyonları kullanılarak oluşturulmuştur. DOST, NT sistemlerde PSAPI kütüphane fonksiyonları, 9x/2000/Me/XP sistemlerde Tool Help kütüphane fonksiyonları kullanılarak çalışan uygulamalara ait süreçlerin, modüllerin ve ipliklerin listelenmesini gerçekleştirir. Uygulamalar listelenirken kullanılacak kütüphane fonksiyonları DOST'un çalıştığı işletim sistemini algılama yeteneği sayesinde belirlenir. Kullanıcılar bilgisayarları ile çalışırken, sistemde çalışan bütün uygulamaların listesini DOST'un her zaman üst pencere olarak kalabilme yeteneği sayesinde takip edebilirler.

Bilim Kodu :
Anahtar Kelimeler : Toolhelp fonksiyonları, PSAPI fonksiyonları, Süreçlerin listelenmesi, Modüllerin listelenmesi, İpliklerin listelenmesi, Kanca yordamları
Sayfa Adedi : 81
Tez Yöneticisi : Prof. Dr. Doğan ÇALIKOĞLU

PROGRAM OF FILE MONITOR

(M. Sc. Thesis)

İsmail KURNAZ

GAZI UNIVERSITY

INSTUTE OF SCIENCE AND TECHNOLOGY

August 2003

ABSTRACT

In this study, a file monitor program whose name is called as DOST has been developed. DOST can show the list of the running applications in the Windows operating systems to the users. The list of currently running applications has been implemented by using some of the special library functions. DOST implements to enumerate running applications and their processes, modules and threads by using PSAPI library functions in NT systems and by using Tool Help library functions in 9x/2000/Me/XP systems. Which library's function will be used while the applications are enumerated, determine by the ability of DOST's perceiving operating system. While the users study with their computers, they can monitor the list of all currently running applications by the ability of DOST's always on top window property.

Science Code :
Key Words : Tool Help functions, PSAPI functions, Enumerating processes, Enumerating modules, Enumerating threads, Hook procedures
Page Number: 81
Adviser : Prof. Dr. Doğan ÇALIKOĞLU

TEŞEKKÜR

Çalışmalarım boyunca değerli yardım ve katkılarıyla beni yönlendiren Hocam Prof. Dr. Doğan ÇALIKOĞLU'na, tecrübelerinden faydalandığım hocam Doç. Dr. Abdullah ÇAVUŞOĞLU'na, kütüphane kaynaklarından faydalanmamda emeği geçen arkadaşım Oktay YILDIZ'a ve manevi destekleriyle beni yalnız bırakmayan aileme teşekkür ederim.

ŞEKİLLERİN LİSTESİ

Şekil **Sayfa**

ÇİZELGELERİN LİSTESİ

Çizelge **Sayfa**

EKLERİN LİSTESİ

KISALTMALAR

Bu çalışmada kullanılmış kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

Kısaltmalar	Açıklama
API	Application Programming Interface
BHO	Browser Helper Objects
DLL	Dynamic Library Link
DOST	Dosya Takip Programı
GUI	Graphics User Interface
IE	Internet Explorer
PSAPI	Platform SDK Application Programming Interface
SDK	System Development Kit
TDI	Transport Driver Interface
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VDM	Virtual DOS Machine

1 GİRİŞ

Çalışmanın Tanıtımı

Bu tez çalışmasında, bilgisayar kullanılırken işletilen programların listesinin kullanıcı tarafından izlenebilmesini sağlayan DOST (**DOS**ya **T**akip Programı) olarak isimlendirilebilecek bir program geliştirilmiştir.

Bilgisayar kullanırken, kullanıcının bilerek çalıştırmadığı fakat sistemde çalışan bir çok program vardır. Kullanıcı bu programların çalıştığını sabit diskin veya ağ kartının ışığının yanıp sönmesinden anlamaktadır. Yürütülen bu faaliyetlerin istenen veya istenmeyen bir faaliyet olduğunu belirlemek önem taşımaktadır. Kullanıcı bilgisayarın kendi hakimiyetinde işlem görmesini beklerken, bu tip işlemler "kullanıcısına rağmen" yürütülüyor olabilir. Yapılan işlemlerde bilgisayar sahibinin verileri bir yerlere gönderiliyor ve yahut bilgisayar sahibine ait bazı özel bilgiler (şifreler gibi) başkaları tarafından öğreniliyor olabilir. DOST bu gibi durumları ortaya çıkarmak amacı ile geliştirilmiştir.

DOST'un hazırlanmasının nedenleri şunlardır:

1. Bilgisayarın kontrolünün kullanıcıdan çıkmasını önlemek,
2. Bilgisayara karşı dış tehditleri engellemek,
3. Casus yazılımları engellemek,
4. Windows işletim sisteminde perde arkasında gerçekleştirilen işlemleri öğrenmek.

DOST ile kullanıcının bilgisayarını kullanırken çalıştırdığı programlar ve çalışan fakat kullanıcının çalıştığını bilmediği programlar ve bu programların ayrıntılarını tespit etme imkânı sağlayan bir araç geliştirilmiştir. Tespit edilen olaylarla ilgili yapılması gerekenler kullanıcının insiyatifine bırakılmıştır. İşlem gören her programa ait alt dosyalar, süreçler (process), modüller, iplikler (thread) ortaya konarak disk birimlerindeki konumları gösterilmiştir.

Böylelikle çalışmanın gereklilik şartları sağlanmış ve bunlara çözümler üretilmiştir.

Çalıştırılan Uygulamaları Takip Programları

Sistemde gerçekleştirilen olayları takip etmek için kullanılan bir çok çeşit program vardır. Bu programlar gerçekleştirdikleri faaliyetlere göre dosya takip programları, kayıt düzenleyicisini izleyici programlar, ağ paketlerini izleyici programlar, sabit disk izleyici programlar, port takip programları ve hata ayıklayıcılarını izleyici programlar olarak sınıflandırılabilir.

Genel olarak dosya takip programları (File Monitor), gerçek zamanlı olarak sistemdeki dosya aktivitesini takip eder ve görüntülerler. Sistemde çalıştırılan dosyaların açık, okuma, yazma, silme veya hangi durumda ise durumları ortaya çıkarılır. Arama özelliği kullanılarak istenen herhangi bir uygulama ile ilgili dosyalar görüntülenebilir. Filtreleme özellikleri ile de istenmeyen dosyaların veya uygulamaların görüntülenmesi engellenebilir. NT sistemlerde, dosyaların takibi için bir dosya sistem sürücüsü kullanılır. Sisteme gelen bütün istekler öncelikle bu sürücüye gelir ve böylece sistemde yürütülen her adım izlenebilir. Windows da ise izleme işini sanal aygıt sürücüsüne (filevxd.vxd) eklenen bir kanca[1] (hook) yordamı gerçekleştirir. Bu yordam kendini bütün dosya sistem isteklerinin çağrı zincirine ekler ve sisteme yapılan her istek öncelikle bu yordama gelir. Dosya takip programları Windows'un nasıl çalıştığını anlamak, uygulamaların dosyaları ve dll'leri nasıl kullandığını anlamak ve sistemde bulunan ve sistem veya uygulama dosya ayarlarından kaynaklanan problemleri ortaya çıkarmak için kullanılırlar.

Kayıt düzenleyici izleyici programlar (Registry Monitor), hangi uygulamaların kayıt düzenleyicisine eriştiğini, bu uygulamaların kayıt düzenleyicisindeki hangi anahtarlara eriştiğini, hangi kayıt düzenleyicisi verisini okuyup, yazdığını gerçek zamanlı olarak izlemek için kullanılan araçlardır. Windows 9x sistemlerde kayıt düzenleyicisinin takibi için regvxd.vxd dosyası kullanılır. NT/2000/XP sistemlerde ise sistem çağrılarının arasına girecek bir dosya sürücüsü kullanılır. Kayıt

[1] Kanca yordamları için bkz. Bölüm 3

düzenleyici izleyici programlar kullanıcının, gerçekte hangi programların kayıt düzenleyicisini kullandığını görmesini ve anlamasını sağlarlar.

Ağ paketlerini izleyici programlar, bir yerel sistemde gerçekleşen tcp ve udp aktivitelerini izlemek için kullanılırlar. Ağ üzerindeki paket geçişleri TDI (Transport Driver Interface – Geçiş Sürücüsü Arayüzü) tarafından yönetilir. TDI, API fonksiyonları ile takip edilerek ağ paketleri takip edilir. Ağ paketlerini izleyici programlar ağ ile ilgili konfigürasyon problemlerini ortaya çıkartmada ve uygulamaların ağı nasıl kullandığını analiz etmede kullanılırlar.

Port takip programları (Port Monitor), sistemdeki bütün seri ve paralel port aktivitelerini izleyip, görüntülerler. Port takip programları uygulamaların portları nasıl kullandığını öğrenmek ve sistemdeki veya uygulamadaki ayarlama problemlerini ortaya çıkarmak için kullanılan bir araçtır.

Sabit disk izleyici programları (Disk Monitor), sistemde bulunan disk aktivitelerini izlemek ve görüntülemek için kullanılan araçlardır. Sabit disk izleyici programları disk alanlarının sektör ve uzunluk bilgileri de izlenebilir.

Hata ayıklayıcılarını izleyici programları (Debugger Monitor), bir yerel sistemde veya TCP/IP protokolü ile ulaşılabilen bir ağ üzerindeki herhangi bir bilgisayarda hata ayıklayıcı çıkışlarını takip etmek için geliştirilen araçlardır. Bir hata ayıklayıcı uygulamasına ihtiyaç duymazlar. Hata ayıklayıcısı olarak Win32 veya çekirdek hata ayıklayıcısını kullanabilirler.

Çalıştırılan Uygulamaları Takip Programlarına Örnekler

Hali hazırda çalışan programları, pencereleri, iplikleri, süreçleri takip etmek için kullanılan programlar şunlardır:

- Microsoft Spy++ programı, Microsoft Windows labaratuvarlarında, Jeffrey M. Richter tarafından yazılmıştır. Microsoft Visual Studio paket programında bulunan programlardan biridir.

- Filemon, bir sistemdeki bütün dosya sistem aktivitelerini takip eden ve gösteren bir uygulamadır. Gelişmiş arama ve filtreleme yetenekleri ile Windows uygulamalarının yürütülmesini, uygulamaların dosyaları ve DLL'leri kullanımını, sistemden veya uygulama konfigürasyonlarından kaynaklanan problemleri ortaya koyan güçlü bir araçtır.

Filemon programı aygıt sürücüsü ve GUI (Graphics User Interface - Grafik Kullanıcı Arayüzü) olmak üzere iki bölümdür. NT sürücüsü Windows NT DDK Build ile geliştirilmiştir. GUI, Microsoft Visual C++ 6.0 ile derlenmiştir. VxD, Vireo (şimdiki adı Numega Labs) yazılım şirketinin VtoolsD 2.0 aracı ile geliştirilmiştir. Filemon programı, Mark Russinovich ve Bryce Cogswell tarafından yazılmıştır.

- Process Viewer, sistemde çalıştırılan bütün süreçleri görüntüleyen bir programdır. Görüntülenen süreçler istenirse bu program vasıtası ile sonlandırılabilir.

Çalışmanın Başlıca Özellikleri ve Önemi

DOST görsel ve program kodları olmak üzere iki kısımdan oluşmuştur. Program, Microsoft Visual C++ 6.0 nesne yönelimli programlama dili kullanılarak hazırlanmıştır.

DOST'un diğer dosya takip programlarından farklı olarak getirdiği katkılar şöyle sıralanabilir;

1. Filemon programından kaynaklanan çalışan dosyaların görüntülenmesindeki karmaşıklık giderilmiştir.

2. Programda, işletim sisteminin kullandığı yapılara özdeş yapılar kullanılarak, programın performansı benzer dosya takip programlarına göre arttırılmıştır.

3. Programın kaynak kodları ile işletim sisteminin kaynakları özdeş yapılarda olduklarından dolayı, uygulama çalıştırılırken sistem hasarlarına veya çökmelerine yol açmamaktadır.

4. DOST diğer programlarda bulunmayan çalışan süreçlere ait iplikleri ve bu iplikleri kullanan süreçleri de görüntüleyebilmektedir.

DOST'un kullanım yerlerinden birisi de casus yazılımların tespit edilmesidir. Çalışmanın ikinci bölümünde casus programlar ve bu programların çalışma şekilleri hakkında bilgi verilmiştir.

Çalışmanın üçüncü bölümünde, program yazılırken kullanılan yöntem ortaya konmuş ve diğer yöntemlerle bir karşılaştırma yapılmıştır. Yöntemin daha iyi anlaşılabilmesi için açıklamalı örnek kod parçaları eklenmiştir.

Dördüncü bölümde çalışmanın temel yapısı bir blok diyagram ile ifade edilmiştir. Bu blok diyagramda yer alan öğeler ve görevleri tanımlanmıştır.

Beşinci bölümde program hazırlanırken yararlanılan özellikler ve bu özelliklerden yararlanma biçimleri ortaya konarak, irdelenmiştir.

Altıncı bölümde çalışmanın bir değerlendirilmesi yapılmıştır. Bu çalışmanın gelecek versiyonunda bulunması gereken yeniliklerin neler olacağına dair fikirler ortaya konmuştur.

Çalışmanın sonuna beş kısımdan oluşan Ekler bölümü eklenmiştir. Ek 1'de, yazılımlara eklenti olarak gelen veya kendileri birer casus yazılım olan programların güncel bir listesi verilmiştir. Ek 2'de, dosya takibinde kullanılan özel kütüphane fonksiyonları incelenmiştir. Ek 3'de Windows işletim sistemlerinde yürütülen

iletilerin takibi için kullanılabilecek ileti sistemi fonksiyonları ele alınmıştır. Ek 4'de, giriş birimi olarak kullanılan fare ve klavye aygıtlarına ait sistem iletileri tanımlanmıştır. Ek 5'de Windows işletim sistemine ait kanca tipleri anlatılmıştır.

2 CASUS YAZILIMLAR

Internet'te milyonlarca bedava (freeware) program vardır. İsimleri bedava olmalarına rağmen bu programların gerçekte bedava olup olmadıkları konusu şüphelidir. Zira bu tip bedava programlar çalışmaya başlamalarından itibaren bilgisayar tarafından yürütülen bazı işlemler kullanıcısının kontrolü dışında yürütülmeye başlar. Bunun sebebi bedava olarak isimlendiren programların ya kendilerinin bir **casus yazılım** (spyware) olması veya casus yazılımların bu tip programlara eklenmiş olmasıdır.

Bir casus yazılımın genel olarak yaptığı iş, kullanıcısına fark ettirmeden bazı bilgileri programcılara göndermesi şeklinde özetlenebilir. Internet'te oldukça popüler pek çok yazılımın böyle eklentileri olduğu bilinmektedir [2].

Yazılımın asıl yaptığı iş ile ilgili olmayan ek casus yazılımlar ya asıl yazılım yüklenirken ya da kullanılmaya başladığında bilgisayara kurulmaktadır. Casus yazılımlar, anti virüs programlarının henüz çözüm getiremediği bir tehlikedir. Pek çok casus yazılım arka planda gizlice çalışarak Internet üzerindeki gezinti alışkanlıklarının reklam şirketlerine bildirmek üzere tasarlanmıştır.

Casus yazılımlar tamamen "intruder veya hacker" mantığından farklıdırlar. Amaçları hiçbir zaman sistemlere dahil olarak, sistemlere zarar vermek, çökertmek veya büyük maddi kayıplar (kredi kartı veya banka hesapları gibi) verdirmek değildir. Amaçları, elektronik ileti adresi, gezilen siteler gibi "kişisel" bilgileri elde ederek bedava program geliştiricilerinin de para kazanmasını sağlamaktır. Zira bu tip kişisel bilgiler, reklam firmaları ile program geliştiriciler arasında ciddi bir pazarlık yapma imkânı sağlarlar.

En yaygın casus programlardan bir tanesi, WhenU.com tarafından geliştirilen "SaveNow" programıdır. BearShare, iMesh ve Global DivX gibi programlarla birlikte gelen bu program, kullanıcıların Internet'teki hareketlerini izleyerek,

[2] Bilinen casus programların bir listesi için bkz. Ek 1

kullanıcıların hangi sitelere gittiklerini izlerler ve aniden bir pencerede özel tekliflerde bulunurlar. Bu program pek çok casus program gibi kişisel bilgileri bir adrese göndermez ama sürekli yeni ürün bilgilerini kullanıcısına sunarak, kullanıcısını meşgul eder. SaveNow programı pek çok yazılım hatası içermektedir. Bundan dolayı zaman zaman bilgisayarın kilitlenmesine, ağ bağlantısının kopmasına da sebep olabilmektedir.

Benzer bir program da, Gator firmasınca yazılan ve Audio Galaxy tarafından dağıtılan "Offer Companion" programıdır. Bu yazılım Audio Galaxy yüklendikten sonra yavaş yavaş bilgisayara indirilir ve kullanılan elektronik ileti adresleri, sık sık ziyaret edilen siteler başta olmak üzere pek çok bilgiyi gator.com adresine gönderir. Bazen de popup pencereleri ile bazı ürünlerin tanıtımını yapar.

Casus yazılımların eklendiği bir program türü de dosya indirme programlarıdır. Netscape Smartdownload, Real Download, Netzip gibi programların hepsi aslında aynı programın değişik versiyonlarıdır. Her dosya indirilişinde, bu programlar, indirilen dosyanın adresi, kullanıcının IP adresi, ağ kartının adresi gibi kullanıcıya has bilgileri programcılarına göndermektedirler.

Hızlı dosya indirmek için kullanılan Gozilla, FlashGET ya da JetCAR gibi programlar yüklendiğinde, başka bir casus yazılım olan RADIATE programı da bu programlarla birlikte yüklenir ve kullanım bilgilerini üretici firmaya gönderir.

Casus programları temizlemek için geliştirilen birçok casus yazılım temizleme programı mevcuttur. Fakat casus temizliği yapan programlarda aynen casus programların entegre olduğu programlar gibi bedavadırlar. Yani casus temizliği yapan programlar da casus yazılımlar içerebilirler. Casus yazılımların gönderdikleri bilgiler belki kredi kartı bilgileri, banka hesap bilgileri, vb. bilgiler gibi gerçekten önemli ve tehlikeli kişisel bilgiler değildirler ama bu tip bilgileri de göndermeyecekleri anlamına gelmemektedir.

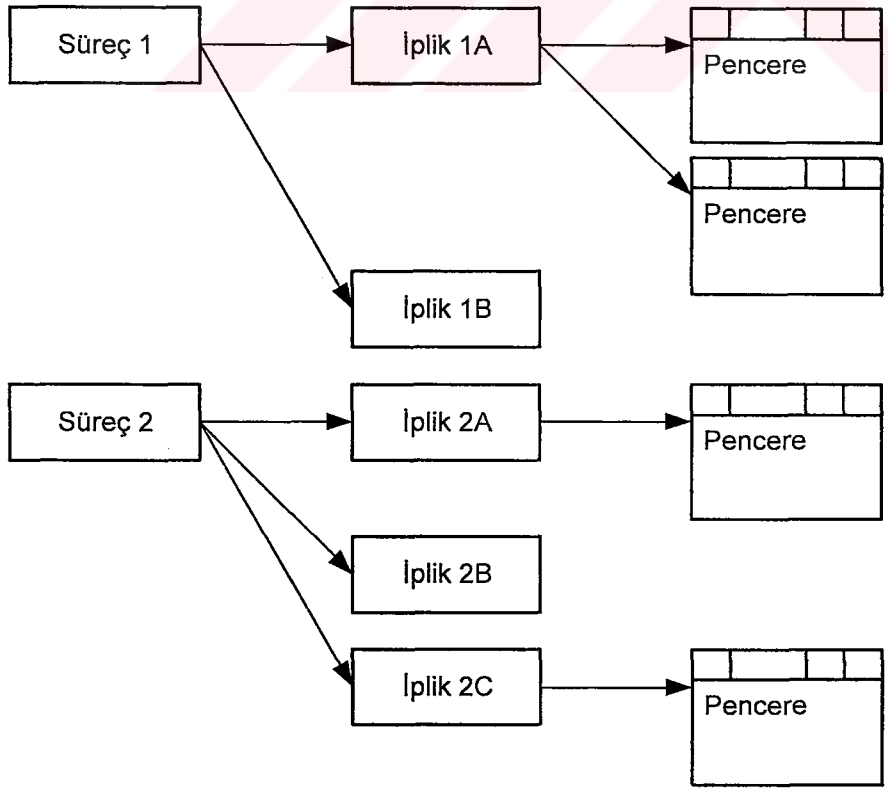
Bedava programların faydası göz önüne alındığında bağlantı hızını düşüren, bilgisayarın düzgün çalışmamasına yol açabilen ve çeşitli tehlikeler içeren bu yan programcıklardan kurtulmak zorunludur. Casus yazılım temizlik programlarının da gerçek bir çözüm üretmedikleri ortadır.

Dosya takip programı casus yazılımların takibinde ve bunların tespiti halinde sistemden silinmesinde kullanılacak yöntemler arasında yer almaktadır.

3 TEORİK DAYANAK

DOS, Windows 3.xx gibi eski nesil işletim sistemleri ve yeni nesil Windows işletim sistemleri (9x/Me/NT/2000/XP) her bir sürece adres alanı ayırmak için karmaşık mekanizmalara sahiptirler. Bu mimari gerçek bir bellek koruması sağlar. Böylece hiçbir uygulama başka bir sürecin adres alanını bozamaz ve işletim sisteminin kendisine zarar vermesini engeller. Bu gerçek sistemi gözetleyen uygulamalar geliştirilmesini de zorlaştırır. (1)

Tipik bir Win32 uygulaması temelde birbirleriyle paralel çalışan bir veya birden daha fazla iplik içerir. Örneğin bir kelime işlemci uygulamasında bir iplik kullanıcı girişlerini işlerken, başka bir iplik dokümanı yazıcıya gönderme işlemini yürütebilir. Bir pencerenin mutlaka bir ipliği vardır. Bir iplik birden fazla pencereye sahip olabilir veya ipliğin hiçbir penceresi de olmayabilir. Bu hiyerarşik yapı Şekil 3.1'de gösterilmiştir.



Şekil 3.1. Süreçler, İplikler ve Pencereler (2)

3.1 Süreç İplik İlişkisi

Bir Win32 işletim sisteminde süreç, çekirdek tarafından kullanılan ve bellekte bulunan program hakkında bilgi veren işlemdir. Süreçler programla ilgili pek çok özelliğe sahiptirler. Bellekte bulundukları yer, kullandıkları iplikler, gönderildikleri bellek sayfaları bu özelliklerden bir kaçıdır. Gerçekte süreçler, iplikleri bulunduran yapılar olarak da düşünülebilir.

Win32 işletim istemlerinde çalıştırma işleminin en önemli birimi ipliktir. İplik, programı çalıştırmak veya çalıştırılmasını beklemek için belleğe yüklenen programa ait bir bölümdür. İpliğin diğer işletim sistemlerindeki karşılığı tasktır. İpliğin tasktan en büyük farkı, diğer iplikler için belleği paylaştırmasıdır. Bu, "çalıştırılan her programa ait birden fazla ipliğin bulunması zorunludur" anlamına gelmez, fakat bir programın çalışması için birden fazla ipliğin çalışması gerekebilir. (3)

İplikler bir programın işlemci tarafından işlenmesini sağlayacak bilgiler içerirler. Örneğin, programın çalıştırılması için bir program sayacı veya komut işaretçisi, bir yığıt işaretçisi ve çeşitli yazmaçlar gereklidir. Bunları bir süreç ile ilişkilendirerek sürecin çalıştırılabilmesi için dinamik bilgiye ihtiyaç vardır. Bu dinamik bilgiler, süreçlerin bünyesinde barındırılan iplikler ile sağlanır. Win32 işletim sisteminin çekirdeği kullandığı mekanizmalar ile ipliklere, işlemciyi kullanma imkânı sağlar. Bir çok-iplikli (multithreaded) uygulamada birden fazla iplik bir süreç ile ilişkilendirilir. Böylece işlemci bir programda pek çok çalıştırma seçeneğinden bir tanesini çalıştırıyor olacaktır. Zira aynı anda bir işlemci ancak tek bir süreci yürütebilir.

3.2 Çalışan Süreç ve İpliklerin Takibi

Bilgisayar kullanırken, kullanıcının bilerek çalıştırmadığı fakat sistemde çalışan bir çok program vardır. Yürütülen bu faaliyetlerin istenen veya istenmeyen bir faaliyet olduğunu belirlemek önem taşımaktadır. Zira bazı programlar güvenlik problemlerine, sistem çökmelerine, donanım uyuşmazlıkları ve çakışmalarına veya

kullanıcıyı boş yere meşgul edecek ekranlar açılmasına neden olabilirler. Bu tip istenmeyen programların varlığının anlaşılabilmesi için sistemde yürütülen faaliyetlerin takip edilmesi gerekir.

Bununla beraber çalışan uygulamaları takip etmenin getirdiği başka avantajlar (1) da vardır. Bunlar:

1. *API fonksiyonlarının takibi:* API fonksiyonlarını kontrol yeteneği, API fonksiyonları çağrılırken meydana gelen görünmeyen özel hareketlerin ortaya konmasında geliştiricilere yardımcı olur. Genellikle perde arkasında kalan problemlerin ortaya çıkarılmasında kullanılırlar. Örneğin, kaynakların hatalı kullanımlarını yakalamak için bellek ile ilişkili API fonksiyonlarının izlenmesi yeterlidir.

2. *Hata ayıklama ve ters mühendislik:* Standart yöntemlerin yanında, sistemlerin gözetlenmesi popüler hata ayıklayıcılar arasında itibarlı bir yere sahiptir. Çoğu program geliştiricisi farklı unsurları tanımlamak ve bunlar arasındaki ilişkileri ortaya koymak için bu tekniği kullanırlar.

3. *İşletim sistemini belgelendirme:* Program geliştiriciler işletim sisteminde derinde kalan olayları öğrenmek için yoğunlaşırlar. Bir hata ayıklayıcı gibi çalışarak gerçekte neler olduğunu anlamaya çalışırlar. Bu amaçların gerçekleştirilmesi için kullanılan bu teknik ile belgelendirilmemiş veya az belgelendirilmiş olayların sırları çözülür.

4. *Fonksiyonellik kazandırma:* Çalışan sistemlere dışarıdan bu teknik kullanılarak eklenen fonksiyonlar ile yürütülen uygulamaların fonksiyonelliği arttırılabilir. Örneğin günümüzde kullanılan uygulamaların çoğunda karşılaşılan güvenlik problemleri, bu tip fonksiyonlar kullanılarak giderilebilir. Böylelikle sistem performansının da olumsuz olarak en az derecede etkilenmesi sağlanır. Zira güvenlik problemi başka bir dış uygulama

ile giderilecek olursa sistem performansı olumsuz yönde daha fazla etkilenecektir.

Çalışan uygulamaları veya bu uygulamaların süreçlerini, ipliklerini, pencerelerini, modüllerini vb. görüntülemek için aşağıdaki yöntemler kullanılır:

1. ToolHelp32 veya PSAPI kütüphane fonksiyonları kullanılır. (4)

2. Windows ileti sistemi takip edilir. Windows da bir uygulamanın çalıştırılabilmesi için iletiler kullanılır. İletiler ile uygulamalar arasına kanca fonksiyonları yerleştirilerek sistemde işlenen tüm iletiler kontrol edilebilir. Kanca fonksiyonlarından dönen veya elde edilen değerler ile sistemde çalıştırılan bütün dosyalar sıralanabilir. Filemon programı kanca fonksiyonları kullanılarak hazırlanmış bir dosya takip programıdır. (5)

Dosya takibinde özel kütüphane fonksiyonlarının kullanılması ile sistem performansında olacak kayıp, kanca yordamlarına nazaran daha azdır. Kütüphane fonksiyonları çalışan uygulamaları, sistemin o andaki durumunun görüntüsünü alarak listeler. Yani sistemi sadece bir anlık meşgul eder. Kanca yordamları ise sistemin ileti akışına dahil olarak her iletiyi değerlendirmek zorundadır. Örneğin farenin hareket ettirilmesi gibi basit bir olayda bile üretilen yüzlerce iletiye müdahil olması gerekir. Her giriş iletisini değerlendirme durumu sisteme fazladan bir yük getirir. Bu da sistem performansını olumsuz yönden etkiler.

Kütüphane fonksiyonlarını kriter alarak program geliştirme, kanca yordamlarına nazaran daha kolaydır. Kütüphane fonksiyonları ile dosya takibi yapılırken bir uygulamanın alt yordamlarını listeleyebilmek için belli fonksiyonları kullanarak belli yapılara erişmek gerekir. Bu fonksiyonlar ve yapılar standart olduğu için program geliştirme daha kolaydır. Kanca yordamları ise her iletiyi değerlendirmek durumunda olduğu için iletinin durumunu, tipini, yaptığı işi, hedeflediği yeri vb. birçok veriyi işlemek zorundadır. Bu da program geliştirmeyi güçleştirir.

Kütüphane fonksiyonları ile sistem fonksiyonları özdeş yapıda olduklarından dolayı kütüphane fonksiyonları çalıştırılırken sistem hasarlarına, çökmelere, uyuşmazlıklara neden olmazlar. Kanca yordamları bazı iletileri işlerken aygıt sürücü dosyalarını kullanırlar. Birbirleri ile eş zamanlı veya farklı zamanda çalışması gereken aygıt sürücüleri bazen birbirleri ile çakışarak sistem hasarlarına, kilitlenmelere ve çökmelere neden olurlar.

Kanca yordamlarının kütüphane fonksiyonlarına göre avantajı ileti sistemine müdahil olarak her iletiyi değerlendirme kabiliyetine sahip olmasıdır. Kütüphane fonksiyonları sadece olan olayları gösterirler, olmadan müdahale etme olanakları yoktur. Kanca yordamları ise olayların gerçekleşmesine izin vermeyebilirler.

3.2.1 Özel kütüphane fonksiyonları

Bir Windows işletim sisteminde cereyan eden olayların listesini oluşturabilmek için Windows işletim sisteminin sürümüne göre iki ayrı özel kütüphane fonksiyonları kullanılır: Windows 9x/2000/Me/XP sistemlerde Tool Help kütüphane fonksiyonları, NT sistemlerde PSAPI kütüphane fonksiyonları.

3.2.1.1 Tool Help fonksiyonları

Tool Help fonksiyonları, program geliştiricilere çalıştırılan Microsoft Win32 tabanlı uygulamalar hakkında daha kolay bilgi vermek için kullanılırlar. Bu fonksiyonlar Windows tabanlı hata ayıklama uygulamalarını temel alarak tasarlanmışlardır.(6)

Tool Help kütüphanesi fonksiyonları Windows 9x/Me/2000/XP işletim sistemlerinde kullanılabilirler. Bu fonksiyonların kullanılabilmesi için tlhelp32.h ve th32.lib dosyalarına ihtiyaç vardır. tlhelp32.h ve th32.lib dosyaları Kernel.dll dosyasını kullanırlar.

3.2.1.1.1 Sistem görüntüsünün yakalanması

Sistem görüntüsü (snapshot), sistem belleğindeki hali hazırda yürütülen süreçlerin, ipliklerin, modüllerin sadece okunabilir bir listesinin kopyasının alınmasıdır. Görüntüler tool help fonksiyonlarının ana öğesidir.

Tool help fonksiyonları görüntüleri kullanarak Win32 süreçlerinin listelerine erişebilirler. Sistem belleğindeki listeler, süreçler başlayıp bittiğinde, iplikler oluşturulup yok edildiklerinde, çalıştırılabilir modüller sistem belleğine yüklenip kaldırıldığında değişirler. Görüntü bilgisinin kullanımı ile uyuşmazlıklar önlenir. Aksi takdirde bir listedeki değişiklikler, bir ipliğin listeye yanlış konulmasına veya bir erişim hatasına neden olabilir. Örneğin, başka iplikler oluşturulurken veya yok edilirken, bir uygulama iplik listesine eklenirse, uygulamayı iplik listesine geçiren bilgi yanlış zamanda gelmiş olabilir ve uygulama iplik listesine eklenirken hataya neden olur. (7)

Sistemden görüntü almak için CreateToolhelpSnapshot() fonksiyonu kullanılır. Bu fonksiyon çağrılırken Çizelge 3.1'deki değerler fonksiyona geçirilerek istenen listelere erişilebilir:

Çizelge 3.1. CreateToolhelpSnapsot() fonksiyonundaki değerler (8)

TH32CS_SNAPPROCESS	Win32 süreçlerin listesi
TH32CS_SNAPTHREAD	Win32 ipliklerin listesi
TH32CS_SNAPMODULE	Win32 süreç modüllerinin listesi
TH32CS_SNAPALL	Bütün görüntülerin listesi

Sistemde çalışan bütün dosyaların takip edilebilmesi için sitemdeki süreçlerin, modüllerin ve ipliklerin takip edilmeleri gerekir.(9, 10)

3.2.1.1.2 Süreç takibi

Win32 süreç listesini içeren bir görüntü, hali hazırda çalıştırılan her bir süreç hakkında bilgi içerir. Listede yer alan birinci süreç hakkında bilgi almak için Process32First fonksiyonu kullanılır. Listedeki birinci sürece eriştikten sonraki diğer süreçlerin bilgilerine de erişmek isteniyorsa Process32Next fonksiyonu kullanılmalıdır. Bu iki fonksiyondan dönen değerler PROCESSENTRY32 yapısında tutulurlar. Bu yapıdan sürecin adı, çalıştırılabilir dosyasının sabit disk üzerindeki yeri, iplik sayısı, modül tanımlayıcısı, tutulduğu bellek alanı, büyüklüğü, bağlı bulunduğu süreç tanımlayıcısı gibi bilgiler elde edilebilir.

3.2.1.1.3 İplik takibi

Win32 iplik listesini içeren bir görüntü, hali hazırda çalıştırılan her bir Win32 sürecinin her bir ipliği hakkında bilgi içerir. Listede yer alan birinci iplik hakkında bilgi almak için Thread32First fonksiyonu kullanılır. Listedeki birinci ipliğe eriştikten sonraki diğer alt ipliklerin bilgilerine de erişmek isteniyorsa Thread32Next fonksiyonu kullanılmalıdır. Bu iki fonksiyondan dönen değerler THREADENTRY32 yapısında tutulurlar. Bu yapıdan ipliğin adı, birlikte çalıştırıldığı süreç adı, numarası, ipliğin öncelikli kullanım bilgisi, tutulduğu bellek alanı büyüklüğü, bağlı bulunduğu süreç tanımlayıcısı gibi bilgiler elde edilebilir.

Belirli bir sürece ait iplikleri listelemek için sistemden görüntü alındıktan sonra, THREADENTRY32 yapısındaki ipliğin birlikte çalıştığı süreç bilgisi kontrol ettirilir. Bu değerin aynı olduğu iplikler bir diziye aktarılarak, bir sürece ait bütün ipliklerin sayısı ve tanımlayıcıları elde edilebilir.

3.2.1.1.4 Modül takibi

Win32 modül listesini içeren bir görüntü, hali hazırda çalıştırılan her bir Win32 sürecinin her bir modülü hakkında bilgi içerir. Listede yer alan birinci modül hakkında bilgi almak için Module32First fonksiyonu kullanılır. Listedeki birinci

modüle eriştikten sonraki modüllerin bilgilerine de erişmek isteniyorsa Module32Next fonksiyonu kullanılmalıdır. Bu iki fonksiyondan dönen değerler MODULEENTRY32 yapısında tutulurlar. Bu yapıdan modülün adı, birlikte çalıştırıldığı süreç adı, numarası, modülün büyüklüğü, modülün sabit disk üzerinde bulunduğu konum, bağlı bulunduğu süreç tanımlayıcısı gibi bilgiler elde edilebilir.

3.2.1.2 PSAPI fonksiyonları

Windows NT sistemlerinde cereyan eden olayların listesini oluşturmak için PSAPI fonksiyonları kullanılırlar. PSAPI fonksiyonları psapi.dll dosyasında bulunurlar. Bu fonksiyonların kullanılabilmesi için psapi.h ve psapi.lib dosyalarının hazırlanan projeye dahil edilmesi gerekir. PSAPI fonksiyonları Windows 9x/Me işletim sistemlerinde çalıştırılamazlar.

PSAPI fonksiyonları kullanılarak sistemde çalıştırılan bütün uygulamaların ve bu uygulamalara ait süreçlerin ve modüllerin listesi oluşturulabilir. Çalışan iplikleri listeleyecek PSAPI fonksiyonu yoktur. Windows 2000/XP gibi NT tabanlı işletim sistemlerinde ipliklerin listesini oluşturmak için Tool Help kütüphane fonksiyonları kullanılabilir.

3.2.1.2.1 Süreçlerin listelenmesi

NT sistemlerde süreçlerin listesini oluşturmak için EnumProcesses() fonksiyonu kullanılır. Bu fonksiyon yürütülen süreçlerin listesini bir diziye aktarır. Süreçlerin tutulacağı dizinin büyüklüğü EnumProcesses() fonksiyonunda belirtilmesi gerekir. EnumProcesses() fonksiyonu ile çalışan süreçlerin tanımlayıcıları döndürülür. Bir süreç tanımlayıcısı on altı tabanlı sayı sistemindeki bir sayıdan ibarettir. Çalışan süreçleri listelerken süreç tanımlayıcıları ile birlikte süreçlerin adları da görüntülenmek isteniyorsa aşağıdaki adımların dizide yer alan her bir sürece uygulanması gerekir.

Dizinin her bir elemanı OpenProcess() fonksiyonu kullanılarak açılır. Süreç açıldıktan sonra GetProcessImageFileName() fonksiyonu kullanılarak sürecin kullandığı dosyanın adı alınabilir. GetProcessImageFileName() fonksiyonu ile döndürülen bu değer süreç listesine eklenir. CloseHandle() fonksiyonu ile açılan süreç kapatılır.

EnumProcesses() fonksiyonu ile oluşturulan süreç dizisinin son elemanı da süreç listesine eklendikten sonra çalışan süreçlerin listesi görüntülenebilecek duruma gelmiştir.

3.2.1.2.2 Modüllerin listelenmesi

NT sistemlerde belirlene bir sürecin modüllerini listelemek için PSAPI kütüphanesindeki EnumProcessModules(), GetModuleBaseName(), GetModuleFileNameEx() ve GetModuleInformation() fonksiyonları kullanılır.

Belirlenen bir sürecin modüllerinin yürütücülerine erişmek için EnumProcessModules() fonksiyonu kullanılır. Bu fonksiyon ile döndürülen modül yürütücü değerleri bir dizide tutulurlar. Dizinin büyüklüğünün tanımlanması gerekir. EnumProcessModules() fonksiyonu modül yürütücü değerleri dışında, bu yürütücülerin tutulduğu alanın büyüklüğünü de döndürür. EnumProcessModules() fonksiyonu ile döndürülen modül yürütücüsü değerleri GetModuleFileNameEx() fonksiyonuna geçirilerek modülün ismi tespit edilebilir. Modülün esas ismini öğrenmek için GetModuleBaseName() fonksiyonu kullanılır. Bu fonksiyona geçirilen modül yürütücü değerleri referans alınarak modüllerin esas isimleri belirlenir ve bir tampon alanına aktarılır.

Modülle ilgili daha fazla bilgi almak için GetModuleInformation() fonksiyonu kullanılır. Hangi modül hakkında bilgi alınacağını, EnumProcessModules() fonksiyonundan döndürülen modül yürütücüsü değeri belirler.

GetModuleInformation() fonksiyonu ile MODULEINFO yapısında tutulan modül yükleme adresi, büyüklüğü ve giriş noktası bilgilerine ulaşılabilir.

Belirlenen sürece ait modüllerle ilgili yukarıdaki fonksiyonlar kullanılarak elde edilen bilgiler bir yapıya aktarılarak, görüntülenmesi sağlanabilir.

3.2.1.3 16-bit uygulamaların listelenmesi

Windows 9x/Me işletim sistemlerinde çalıştırılan 16-bit uygulamaları listelemek için ToolHelp32 fonksiyonları kullanılır. Windows NT/2000/XP işletim sistemlerinde 16 bit uygulamaları listelemek için Tool Help veya PSAPI kütüphane fonksiyonları kullanılamaz. Çünkü bu işletim sistemlerinde DOS işletim sistemi uyumlulukları yoktur. Bu işletim sistemlerinde 16-bit uygulamalar sanal bir DOS makinesi (VDM) ile çalıştırılırlar.

Windows NT/2000/XP işletim sistemlerinde çalıştırılan 16-bit uygulamaları listelemek için VDMEnumTaskWOWEx() fonksiyonu kullanılır. Bu fonksiyonu çalıştırmak için VDMDBG.h ve VDMDBG.lib dosyalarının programa dahil edilmesi gerekir. Bu dosyalar "ntvdm.exe" programı kullanılarak işletilebilir. Çalışan uygulamaları listelemek için VDMEnumTaskWOWEx() fonksiyonu gereklidir fakat yeterli değildir. Uygulamaların listelenmesini gerçekleştirmek için bu fonksiyona, listelemeyi yapacak bir fonksiyonun da parametre olarak geçirilmesi gerekir.

3.2.2 Windows ileti sistemi

Windows işletim sistemi bir GUI uygulamasıdır. GUI uygulamalarında ileti tabanlı bir çalışma biçimi söz konusudur. Klavye, fare vb. girdi birimlerinden alınan bilgilere **ileti** denir. Bir girdi işlemi oluştuğunda bunu önce sistem, kendisi alır ve sonra bu girdi işlemi hangi programa ilişkinse o programa gönderir. Bu yüzden Windows da ki programlama modeli gelen iletinin yorumlanarak işlenmesi temeline dayanır. (11)

İletiler bir Windows sisteminin can damarlarıdır. Bir Windows programını bir insan vücuduna benzetecek olursak GetMessage()/DisPatchMessage() [3] gibi fonksiyonlar programın kalbi, pencere yordamları ve bunlara ilişkin fonksiyonlar da programın arterleri, damarları ve kılcal damarları olurlar. Kan akışı durursa vücut öleceği gibi bir Windows programı da iletileri işlemeyi durdurursa ölür. Sadece program sonlanmakla kalmaz, bu sirkülasyon devam ederse diğer programlarda durur.

Windows da iletiler pek çok amaç için kullanılmaktadırlar. Fare hareketleri veya fare tuşlarına tıklanması, klavye tuşlarına basılması, menu seçimleri gibi olaylar iletiler ile gerçekleştirilir. İletiler sistemde gerçekleştirilen olayları araştırmak için de kullanılır. Örneğin iletiler Windows'un kapatılabilmesi için diğer bütün programların uygun olup olmadığını anlamak için kullanılırlar.

İleti sistemi ile ilgili kodlar USER.EXE uygulaması ile gerçekleştirilir. Ayrıca KERNEL modülündeki zamanlayıcı da sistemin doğru akışını sağlamak için rol oynar.

3.2.2.1 İleti çeşitleri

Beş çeşit ileti vardır. Bu ileti çeşitleri (12), WINDOWS.H dosyasında tanımlanmıştır.

- QS_KEY, QS_MOUSEMOVE, QS_MOUSEBUTTON: GetQueueStatus() fonksiyonu ile alınır. Giriş iletileridirler. Klavye, fare gibi donanım aygıtlarından üretilen giriş iletilerini alırlar. Bu iletiler, paylaşılan sistem ileti kuyruğunda tutulurlar.

[3] Fonksiyonlar için bkz. Ek 3

- QS_POSTMESSAGE: PostMessage() veya PostAppMessage() fonksiyonları ile gönderilen iletiler, ileti kuyruğuna yerleştirilir. Her bir program için tek bir uygulama ileti kuyruğu vardır.
- QS_TIMER: QS_TIMER bayrağı 1 olduğunda ve bir uygulama GetMessage(), PeekMessage() fonksiyonlarını çağırdığında WM_TIMER iletileri üretilir. Bu iletiler uygulamanın veya sistemin ileti kuyruklarında tutulmazlar. Böylece zamanlayıcı (timer) iletileri yedeklenemez ve uygulama veya sistem ileti kuyruklarını dolduramazlar.

- QS_PAINT: QS_TIMER iletisine benzer olarak kuyrukta beklemez, fakat bir uygulama bir ileti talep ettiğinde üretilmesi gerekir. Açılan pencerelerin renklerinin ayarlanmasında etkindir. Bir uygulama yeni bir ileti için sorduğunda, QS_PAINT bayrağı kontrol edilir ve eğer 1 ise WM_PAINT iletisi oluşturulur. Buradaki önemli nokta boyama (paint) iletileri uygulama ileti kuyruğunda ileti kalmayana kadar üretilmezler.

- QS_SENDMESSAGE: SendMessage() API'si, bir iletinin bir pencereye gönderilmesine izin verir ve o pencereden acilen cevap alınmasını garanti eder. Aynı uygulama içinde ileti göndermek zor değildir. Zor olar iki farklı görev arasındaki ileti gönderilmesi işlemidir. Bu zorluğun sebebi her bir pencere yordamı kendi normal görevini yapmak zorunda olmasıdır. Bu yüzden Windows zamanlayıcısında karmaşıklık olur. Bu karmaşıklığın giderilebilmesi için iki görev arasındaki senkronizasyonun iyi sağlanması gerekir.

3.2.2.2 İleti kuyrukları

Windows ileti tabanlı (event driven-olay sürümlü) bir işletim sistemidir. DOS ve UNIX konsol sistemlerinde klavye, fare gibi girdi bilgileri programcının çağırdığı fonksiyonlarla elde edilir. Bu fonksiyonlar olay gerçekleşene kadar bekler ve gereken

bilgiyi alırlar. Oysa Windows sisteminde bir girdi olayı gerçekleştiğinde bu olay ilk elden sistem tarafından alınır.

Gerçekleşen girdi olayı MSG[4] isimli bir yapı biçiminde ifade edilir ve hangi programa ilişkin ise Windows tarafından o programın ileti kuyruğuna yazılır. Windows da her programın (aslında her ipliğin) bir ileti kuyruğu vardır. İleti kuyruğu MSG yapı türünden bir yapı dizisi olarak düşünülebilir. İleti kuyruğu Win3x sistemlerinde 8 eleman uzunluğunda olmasına karşın Win32 sistemlerinde, bağlı liste tekniği ile neredeyse sınırsız biçime dönüştürülmüştür. (11,12)

3.2.2.2.1 Uygulama ileti kuyruğu

Bir program başlatılırken, USER.EXE uygulamanın başlangıç rutini için bir ileti kuyruğu üretir. Uygulama kuyruğu için bellek ayrılır ve iki USER.EXE rutini CreateQueue() ve CreateQueue2() başlatılır. İleti kuyruğu belleğin bir segmentine yerleştirilir. Hali hazırdaki ileti kuyruğu için yürütücü değer GetTaskQueue() fonksiyonundan elde edilebilir.

Sistemdeki her pencere bir uygulama ileti kuyruğu ile ortak çalışır. Örneğin WND yapısında, uygulama ileti kuyruğu için bir handle değeri vardır (18h HANDLE hQueue). İleti gönderildiğinde WND yapısındaki hQueue değeri iletinin hangi kuyruğa ekleneceğini belirler.

İletiler uygulama kuyruğuna PostMessage() fonksiyonu kullanılarak yerleştirilirler. DefWindowProc() gibi bazı dahili Windows fonksiyonları PostMessage() fonksiyonunu kendileri çağırırlar. Eğer bir program kuyruktaki başka bir taska, o taskın ilgili pencere parametresini bilmeden bir ileti göndermek istiyorsa, program PostAppMessage() fonksiyonunu kullanabilir. PostAppMessage() fonksiyonu hTask parametresini alır ve task veri tabanından bu parametreli taskı bularak ilgili kuyruğu tespit eder. Daha sonra ileti hedef kuyruğa konumlandırılır. (12)

---

[4] MSG yapısı için bkz. Ek 5, Kanca Tipleri, WH_GetMessage, lParam parametresi

Bir uygulama ileti kuyruğunun varsayılan ileti büyüklüğü 8 iletidir. Çoğu uygulamalarda bu büyüklük yetmemektedir. İleti kuyruğunun büyüklüğünü artırabilmek için SetMessageQueue() fonksiyonu kullanılabilir. Yalnız bu fonksiyonu kullanabilmek için önceden ileti kuyruğuna en az bir iletinin gönderilmiş olması gerekir. Çizelge 3.2'de bir iletinin, uygulama kuyruğundaki formatı gösterilmiştir.

Çizelge 3.2. Uygulama kuyruğundaki bir iletinin formatı

```
DWORD       ExtraInfo   // Ekstra bilgi
HWND        hwnd        // İlgili pencere
WORD        message     // İleti
WPARAM      wParam      // İletiye bağlı değişkenler
LPARAM      lParam      // İletiye bağlı değişkenler
DWORD       time        // Zaman (msn.)
POINT       pt          // İleti kuyruğundaki adresi
```

3.2.2.2.2 Sistem ileti kuyruğu

Sistem ileti kuyruğu, uygulama ileti kuyruğunun ikiz kardeşi gibidir. Sistem ileti kuyruğu bütün donanım ve giriş iletilerini tutar. Genellikle bunlar fare ve klavye olaylarını kapsar, ama sistem kuyruğu bu tip giriş aygıtları ile sınırlı değildir. Alternatif giriş aygıtları için iletileri sistem kuyruğuna yerleştirecek mekanizmalar vardır (Örneğin digitizing tablets).

Uygulama ileti kuyruğunda olduğu gibi sistem ileti kuyruğu USER.EXE tarafından ayrılır ve başlatılır. Sistem ileti kuyruğunun yapısı tutulan iletilerin farklılığı dışında uygulama ileti kuyruğu ile aynıdır. Uygulama ileti kuyruğundan farkı, bütün pencereler için tek bir sistem ileti kuyruğu vardır.

Sistem ileti kuyruğunun görevi donanım iletilerini saklayarak, bu iletilerin kullanıcı girişine çevrilmesini sağlamaktır. En basit olarak farenin ekran üzerinde hareket

ettirilmesi düzinelerce WM_MOUSEMOVE [5] iletisinin üretilmesine sebep olur. Bu tip olayların kaybolmasını engellemek için sistem kuyruğu bir uygulama kuyruğuna nazaran daha fazla ileti tutabilir. Sistem kuyruğunda tutulan ileti sayısının varsayılan değeri 120'dir. Bu sayı WIN.INI dosyasındaki TypeAhead değeri değiştirilerek azaltılıp çoğaltılabilir.

Sistem kuyruğu bir iplik tarafından kilitlenebilir. Bu durum diğer hiçbir ipliğin, kilitli iplik kendisi için gerekli bütün iletileri okuyana kadar sistem ileti kuyruğundan ileti okuyamamasına neden olur. Örneğin bir çift tıklama iletisi beraberinde birçok iletiyi kuyruğa ekler. Sürecin ortasında herhangi bir ileti görev için çıkarılamayacaktır. Sistem kuyruğunun kilidi bu iplik için başka ileti kalmadığında veya başka bir ipliğin yürütülmesi için bir ileti alındığında kaldırılır. Çizelge 3.3'de sistem kuyruğunda bulunan bir iletinin formatı gösterilmiştir.

Çizelge 3.3. Sistem kuyruğundaki bir iletinin formatı

```
DWORD       // Ekstra bilgi
WORD        // İleti numarasına bağlı değişkenler
WORD        // İleti numarası Örneğin WM_MOUSEMOVE
WORD        // İleti numarasına bağlı değişkenler
DWORD       // Zaman (Sistem açıldığından beri geçen süre (msn.)
```

Bir olay sistem ileti kuyruğuna şöyle eklenir:
USER.EXE dosyasında yer alan EnableInput() rutini fare veya klavye sürücüleri için Enable() fonksiyonlarını çağırır. Enable() fonksiyonundaki klavye veya fare sürücü parametreleri, mouse_event veya keybd_event() USER.EXE fonksiyonlarında adreslenirler. Fareyi hareket ettirmek veya klavye tuşlarından birine basmak bir donanım kesmesi üretilmesine sebep olur. Kesme yürütücüsü fare ve klavye sürücü dosyalarını (mouse.drv, keyboard.drv) çalıştırır. Sürücü dosyaları ilgili mouse_event() veya keyb_event() fonksiyonlarını çağırır. Olay, yürütülen süreçler ile anlaşıldıktan sonra SaveEvent() fonksiyonu ile kaydedilir. SaveEvent() fonksiyonu, WriteSysMsg() fonksiyonunu kullanarak iletiyi sistem ileti kuyruğuna ekler.(13)

[5] Sistem giriş iletiler için bkz. Ek 4

3.2.2.3 Uygulama iletilerinin kaydedilmesi ve cevaplandırılması

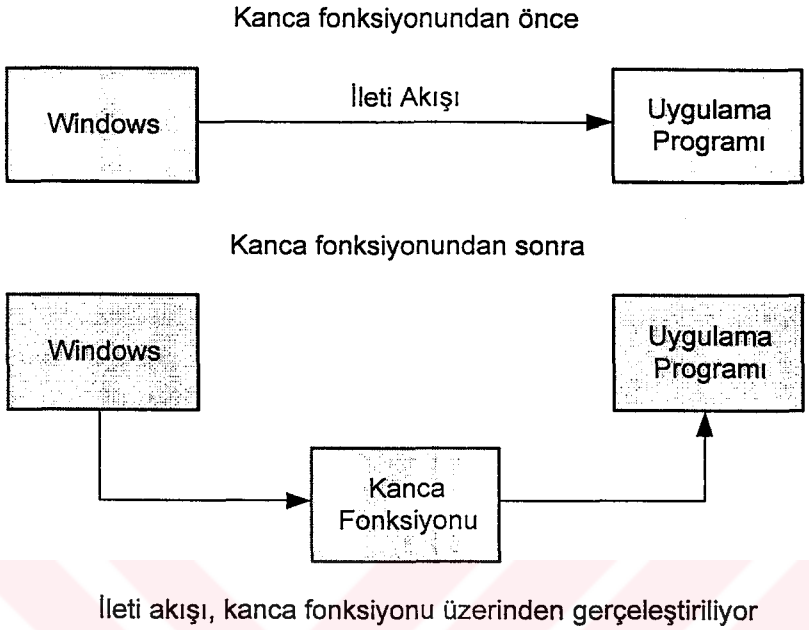
Windows işletim sistemi, programlara iletiler göndererek haberleşir. Bu nedenle programların aldığı iletilerin kaydedilmesi kolay bir görevdir. İletiler, SendMessage() fonksiyonu ile gönderilen iletiler tarafından cevaplandırılabilir. İlk olarak iletilerin bir sıralaması kaydedilir.

Uygulama iletilerini kaydetmenin en kolay yolu programın ileti döngüsüne bir adım eklemektir. Bu ekstra adım programa gönderilen her bir iletiyi kaydedecektir. Bu adım uygulamanın başında bulunacaktır. İleti kaydetme işleminin programın başında yaptırılması kaçınılmazdır, çünkü bu işlem süreçlerin kaydedilmesinin bir bölümüdür.

Alınan her bir ileti depolanmalıdır. Basit uygulamalar için kaydedilmesi gerek çok ileti yok ise her bir ileti bir dizi de depolanabilir. Bu basit dizi yapısı için aşağıdaki kodlar bir örnektir.

```
struct messages{
      UINT        msg;
      WPARAM      wParam;
      LPARAM      lParam;
      HWND        hwnd;
}     MsgArray[MAXMESS];
```

Kapsamlı uygulamalar için bu tip diziler yetersiz kalırlar. Kapsamlı uygulamalarda iletileri depolamak için her bir iletiye dinamik olarak ulaşılabilecek disk alanı ayrılmalıdır ve iletileri tutmak için bir dosya kullanılmalıdır.



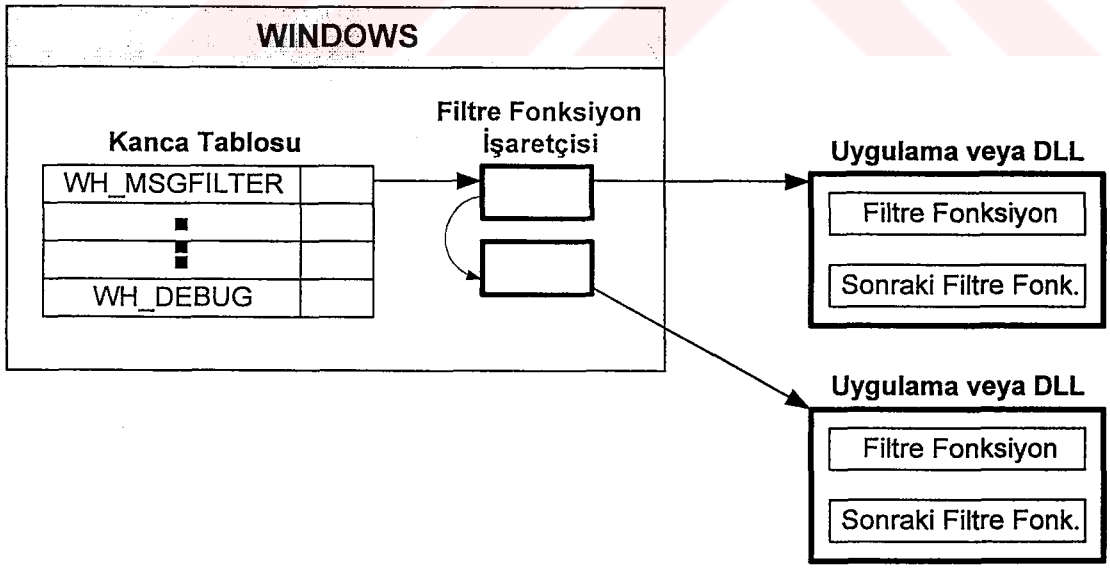
Şekil 3.2. Kanca fonksiyonlarının ileti akışına etkisi (9)

Windows uygulamalarında yapılan herhangi bir basit işlem için dahi birçok ileti üretilmektedir. Bu iletilerin hepsinin depolanması hem fazla disk alanının kullanılmasına yol açacak hem de gereksiz işlemler için dahi sistem meşgul edilecektir. Bu yüzden cevap verilmesi gereken iletiler için bir sıralama oluşturulmalıdır.

3.2.3 Kanca fonksiyonları

Program geliştirilirken bir ileti kaydedicisi kullanmak teknik olarak yanlış olmamakla birlikte genellikle tercih edilmez. İlk olarak, çoğu programcı ileti döngüsünün içinde ekstra kodların bulunmasını rahatsız edici bulurlar. Üstelik bu kodlar potansiyel olarak da etkin değillerdir. Örneğin, sadece iletilerin kaydedilmesi veya takip edilmesi isteniyorsa, bütün iletilerin başına bir işaretçi ekleyerek iletinin kaydedilip kaydedilmediği öğrenilebilir. Neyse ki programlar başka bir yöntem kullanarak ileti akışının içine sızabilmektedirler. Burada kullanılan yöntem Kanca fonksiyonlarıdır. Kanca fonksiyonları ileti akış sistemine dahil olarak, iletilerin izlenmesini gerçekleştirirler (Şekil 3.2).

Kanca yordamları, bir fonksiyon bir uygulamaya ulaşmadan önce olayların (iletilerin) arasına giren bir mekanizmadır. Fonksiyonlar olayları yürütebilirler, hatta onları değiştirebilirler veya bitirebilirler. Bu tip araya giren fonksiyonlar "filtre fonksiyonlar" olarak isimlendirilmiştir (14,15). Filtre fonksiyonlar, arasına girdikleri olayın tipine göre sınıflandırılırlar. Örneğin bir filtre fonksiyonu bütün klavye ve fare olaylarının arasına girmek için yazılmış olabilir. Windows işletim sisteminde, bir filtre fonksiyonu eklendiği kanca ile birlikte çağrılır. Bir kancaya birden fazla filtre fonksiyonu eklenmiş ise Windows bir fonksiyonlar zinciri oluşturur. En son yüklenen fonksiyon zincirin başına konur ve diğer fonksiyonlar yüklenme zamanlarına göre zincirde sıralanırlar. Kancayı tetikleyen bir olay olduğunda zincirdeki ilk fonksiyon çağrılır. Bu olay kanca çağırma olarak adlandırılır. Şekil 3.3'de kanca çağırma olayı gösterilmiştir. Örneğin bir filtre fonksiyon CBT kancasına eklendiğinde ve bir olay bu kancayı tetiklediğinde (örneğin bir pencere açmak), Windows CBT kancasını filtre fonksiyon zincirindeki ilk fonksiyonu çağırmak suretiyle çağırır.



Şekil 3.3. Kanca fonksiyonlarının çağrılması (14)

Windows, filtre fonksiyon zincirini kendi içinde saklar. Sistemde bir kanca fonksiyonu işletilmesi için bir ileti alındığında, kanca tablosundan ilgili fonksiyon

çağrılır. Çağrılan fonksiyona ait filtre fonksiyonlarına, filtre fonksiyon işaretçisi yardımı ile ulaşılır. İşletilecek dosyaya ait (dll veya uygulama dosyaları) filtre fonksiyonu veya filtre fonksiyonları çağrılarak işletilir. (16)

Genel olarak kanca fonksiyonları sistem ve uygulama kancaları olmak üzere ikiye ayrılırlar. Bir sistem, kanca fonksiyonu kullanarak sisteme giren tüm iletileri yakalayabilir. Uygulama kanca fonksiyonları ile sadece o uygulama veya o pencereye ait iletiler yakalanabilir. Daha da ötesi bir uygulama kanca fonksiyonu belirlenmiş bir iplik ile ilgili iletileri da takip edebilir. Bunun anlamı çok-iplikli programlarda izlenmesi gereken her bir iplik, bir pencere oluşturuyorsa; bunlar için ekstra kancaların yüklenmesi gerekebilir.

Bir uygulama kancası kullanırken, kanca fonksiyonunun alabileceği çeşitli ileti kategorilerinden bir tanesini seçmek mümkündür. Örneğin bir kanca ile sadece klavye iletileri alınırken, diğer bir kanca ile de fare iletileri alınabilir. Uygulamanın özelliğine göre sadece istenen iletilere ulaşılması, sisteminde daha iyi kullanılmasını sağlar. Bununla birlikte istenirse tek bir kanca fonksiyonu kullanarak uygulama ile ilgili bütün iletiler de alınabilir.

Bir kanca fonksiyonunu genel formu aşağıdaki gibidir:

```
LRESULT CALLBACK Hook(int code, WPARAM wParam LPARAM lParam);
```

Kanca fonksiyonları sistem tarafından çağrıldıkları için tipleri CALLBACK olarak tanımlanırlar. Fonksiyonda yer alan code ve wParam parametreleri kanca fonksiyonunun tipini belirler. lParam parametresi ise iletiyi tanımlayan bir yapının adresini işaret eder.

3.2.3.1 Kanca fonksiyonlarının sisteme dahil edilme teknikleri

Genellikle bir kanca sistemi, bir servis ve bir sürücüden oluşur. Kanca servisi sürücünün hedeflenen süreçlere, doğru zamanda dahil edilmesinden sorumludur.

Aynı zamanda sürücüyü yöneterek sürücünün, gerçekleştirdiği aktiviteler hakkında bilgi toplar.

Kanca fonksiyonları sisteme dahil etmek için aşağıdaki teknikler (1) kullanılmaktadır:

1. Kayıt Düzenleyicisi (Registry)
2. Sistem Windows Kancaları
3. CreateRemoteThread() API'sini kullanmak
4. Tarayıcı Yardımcı Nesne Eklentileri
5. MS Ofis Eklentileri

3.2.3.1.1 Kayıt düzenleyicisi (Registry)

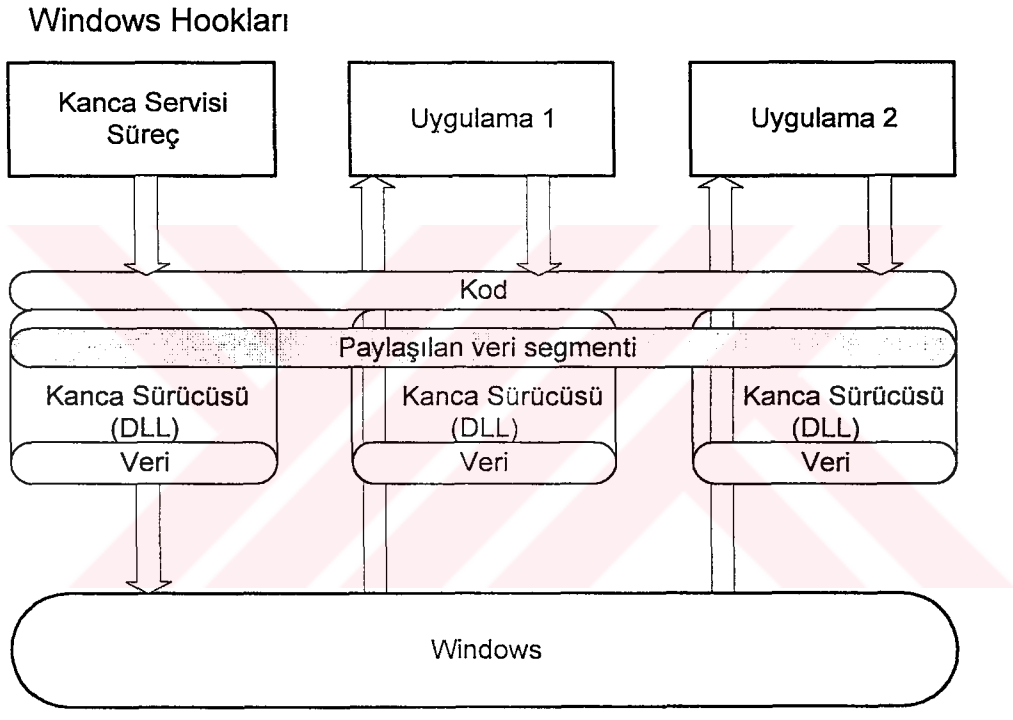
USER32.DLL dosyası ile bağlantı kuracak süreçlere bir dll dosyası eklemek için öncelikle dll isminin belirtilmesi gerekir. Aşağıdaki kayıt düzenleyicisi satırı ile dll ismi kolaylıkla eklenebilir.

```
HKEY_LOCAL_MACHINE\Software\Microsoft\WindowsNT\CurrentVersion\
Windows\AppInit_DLLs
```

Eklenen dll dosyası ile birlikte kayıt düzenleyicisi, her dll dosyası için atadığı gibi bu dosyaya da bir anahtar değeri atar. Bu değer USER32.DLL dosyasına geçirilir. Atanan anahtar değerine bir talep olduğunda, USER32 bu değeri okuyarak, LoadLibrary() fonksiyonunu kullanarak ilgili dll dosyasını çağırır. Böylece önceden belirlenen süreçler yürütülmeden önce eklenen dosya çalıştırılarak süreçlerin gözetlenmesi sağlanır. Yapılan değişikliklerin aktif olabilmesi için Windows'un yeniden başlatılması gereklidir. Bu mekanizma Windows NT/2000 işletim sistemlerince desteklenmektedir.

3.2.3.1.2 Sistem Windows kancaları

Hedeflenen bir süreci gözetlemenin çok popüler başka bir yolu da Windows kancalarının kullanımlarıdır. Bir uygulama sistemdeki ileti trafiğini izlemek için özel bir filtre fonksiyonu yükleyebilir. Bu fonksiyon iletileri, hedef pencere yordamlarına erişmeden önce yakalayarak işleyebilir.



Şekil 3.4. Uygulama 1 ve 2 adres alanlarına bir kanca servisinin dahil edilmesi

Şekil 3.4'de bir sistem kancasının, uygulama 1 ve uygulama 2 isimli adres alanlarının arasına girerek ileti trafiğini nasıl takip ettiği gösterilmiştir. Kanca servisi, ilgili süreç için talep geldiğinde ilgili filtre fonksiyonunu çağırır. Kanca sürücüsü ile bağlantı kurularak Windows, uygulama 1'i veya uygulama 2'yi çalıştırır (çalıştırılacak uygulama fonksiyondan gelen parametre ile saptanır). Çalıştırılan uygulama için filtre fonksiyonu kanca servisine bir değer döndürür.

Bu mekanizma Windows NT/2000/9x işletim sistemleri tarafından desteklenmektedir.

3.2.3.1.2.1 Kanca fonksiyonlarının yüklenmesi

Bir kanca fonksiyonunu yüklemek için SetWindowsHookEx() fonksiyonu kullanılmalıdır. Fonksiyonun prototipi aşağıdaki gibidir:

```
HHOOK SetWindowsHookEx(int type, HOOKPROC HookFunc, HINSTANCE
hInst,DWORD ThreadID);
```

Eğer fonksiyon başarılı olursa fonksiyon kancaya bir yürütücü değeri döndürür. Eğer hata çıkarsa bu değer NULL olur. Prototipte yer alan type parametresine bir kanca tipi[6] girilmelidir. HookFunc, kanca fonksiyonunun yükleneceği işaretçidir. Yüklenecek kanca fonksiyonunun tanımlandığı dll dosyasındaki yürütücü değeri hInst parametresine geçirilir. Bununla birlikte eğer kanca fonksiyonu cereyan eden süreç ile tanımlanmışsa ve süreç tarafından oluşturulan bir ipliği izliyorsa, hInst değeri NULL olmalıdır. ThreadID izlenen ipliğin ID numarasıdır. Global kanca fonksiyonları için bu numara sıfırdır.

3.2.3.1.2.2 Kanca fonksiyonlarının kaldırılması

Kanca fonksiyonlarının sistem performansına olumsuz tesirleri vardır. Bu yüzden kanca fonksiyonlarına gerek kalmadığında mümkün olduğu kadar çabuk sistemden kaldırılmalıdırlar. Bu işlem için UnhookWindowsHookEx() fonksiyonu kullanılır. Fonksiyonun prototipi aşağıdaki gibidir:

```
BOOL UnhookWindowsHookEx(HHOOK HookFunc);
```

HookFunc, kaldırılacak kanca fonksiyonunun yürütücü değeridir. Fonksiyon başarılı olursa sıfır olmayan bir değer, başarısız olursa sıfır değerini döndürür.

3.2.3.1.2.3 Bir sonraki kanca fonksiyonunun çağrılması

Bazı durumlarda bir kanca fonksiyonu bir iletiyi yok sayarak, basitçe bu iletiyi bir sonraki kanca fonksiyonuna geçirir. Böyle durumlarda iletiyi diğer kanca

[6] Kanca tipleri için bkz. Ek 5

fonksiyonuna geçirmek için CallNextHookEx() fonksiyonu kullanılır. Fonksiyonun prototipi aşağıdaki gibidir:

```
LRESULT CallNextHookEx(HHOOK CurHook,int code, WPARAM wParam,
                       LPARAM lParam);
```

CurHook hali hazırda yürütülen kanca fonksiyonunun yürütücü değeridir. code, wParam, lParam değerleri çağrılan kanca fonksiyonuna geçirilmelidirler. Hali hazırda yürütülen kanca fonksiyonu bu parametrelerin değerlerini, çağrılan kanca fonksiyonunun parametrelerine geçirir.

3.2.3.1.3 CreateRemoteThread() API'sini kullanmak

CreateRemoteThread() fonksiyonu sadece NT ve Windows 2000 işletim sistemleri tarafından desteklenmektedir. Win 9x işletim sistemlerinde bu fonksiyon çağrıldığında, hiçbir şey yapmadan her defasında NULL değeri geri döndürülmektedir.

Herhangi bir süreç dinamik olarak, LoadLibrary() API'sini kullanarak bir dll yükleyebilir. Önemli olan herhangi bir sürecin ipliklerine erişim hakkı olmadan LoadLibrary() fonksiyonunu çağırarak bir dış süreç ile söz konusu ipliklere erişmektir. İşte bu işlem için CreateRemoteThread() fonksiyonu kullanılır.(17)

CreateRemoteThread() API'si kullanılırken dikkat edilmesi gereken önemli bir nokta vardır. Eklenen uygulama, hedeflenen sürecin sanal belleğinde işlem görmeden ve CreateRemoteThread() fonksiyonuna bir çağrı yapılmadan önce, her zaman ilk olarak OpenProcess() API fonksiyonu ile süreci açar ve sürece PROCESS_ALL_ACCESS bayrağını parametre olarak geçirir. Bu bayrak bir sürece ait bütün haklara erişilmek istendiğinde kullanılır. Bu durumda OpenProcess() fonksiyonu bazı süreçler için NULL değeri döndürecektir. Bu hata güvenlik şartlarında yer alan gerekli izinlerin olmamasından kaynaklanmaktadır. Bu tip sınırlandırılmış süreçler işletim sisteminin bir bölümüdür ve normal bir uygulama ile bu süreçlerin işletilmesine izin verilmez. Eğer bu tip süreçlerde hata olduğu veya

sisteme zarar verdiği düşünülüyorsa, bu süreçlere fonksiyonlar eklemek için öncelikle bu süreçlere ait erişim hakları gözden geçirilmelidir. Örneğin smss.exe, winlogon.exe, services.exe gibi sistem kaynaklarına erişmek için hata ayıklama hakkının erişime açılmış olması gereklidir.

3.2.3.1.4 Tarayıcı Yardımcı Nesne Eklentileri

Bazen sadece Internet Explorer (IE) programının içine özel kodlar eklemek gerekir. Bu amaç için Microsoft firması kolay ve çok iyi bir ara birim sağlamıştır (Browser Helper Objects-BHO, Tarayıcı Yardımcı Nesneleri). Bu ara birim sayesinde IE her çalıştırıldığında, BHO ile tanımlanmış kodlar çalıştırılarak hedeflenen süreçler gözetlenebilir.

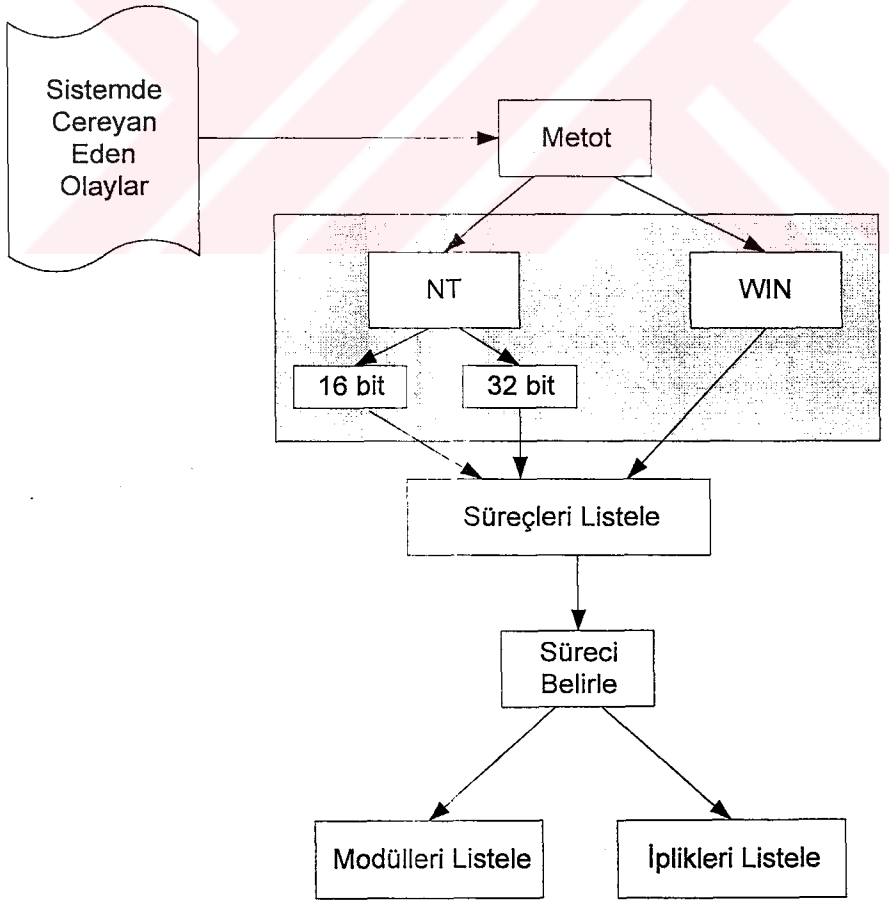
3.2.3.1.5 MS Ofis eklentileri

BHO arabirimine benzer olarak MS Ofis uygulamalarına kodlar eklemek isteniyorsa, MS Ofis'in eklentileri kullanılabilir. Bu eklentilerin nasıl yapılacağı ile ilgili bir çok örnek, MS Ofis uygulama paketlerinin tam sürümlerinde mevcuttur.

4 DOST'UN TEMEL YAPISI

Bir sistemde cereyan eden olayları takip etmek için iki yöntem kullanılabilir. Birinci yöntem de sistem çalışırken hali hazırdaki bütün olayların kopyası alınır. Bu olayların gerçekleşmesi için çalışması gerekli bütün dosyalar NT sistemlerde PSAPI fonksiyonları, diğer Windows sistemlerinde ise Tool Help fonksiyonları ile görüntülenir. DOST, bu yöntem kullanılarak tasarlanmıştır.

İkinci yöntem de ise da cereyan eden olayları izlemek için işletim sisteminin ileti sistemine girilir. İletiler sistemde akarken kanca fonksiyonları, ileti kuyrukları ve iletinin hedeflendiği yer arasına girerek, ileti akışını kontrol eder. Böylelikle sistemdeki bütün faaliyetler takip edilir.



Şekil 4.1. DOST'un Temel Blok Şeması

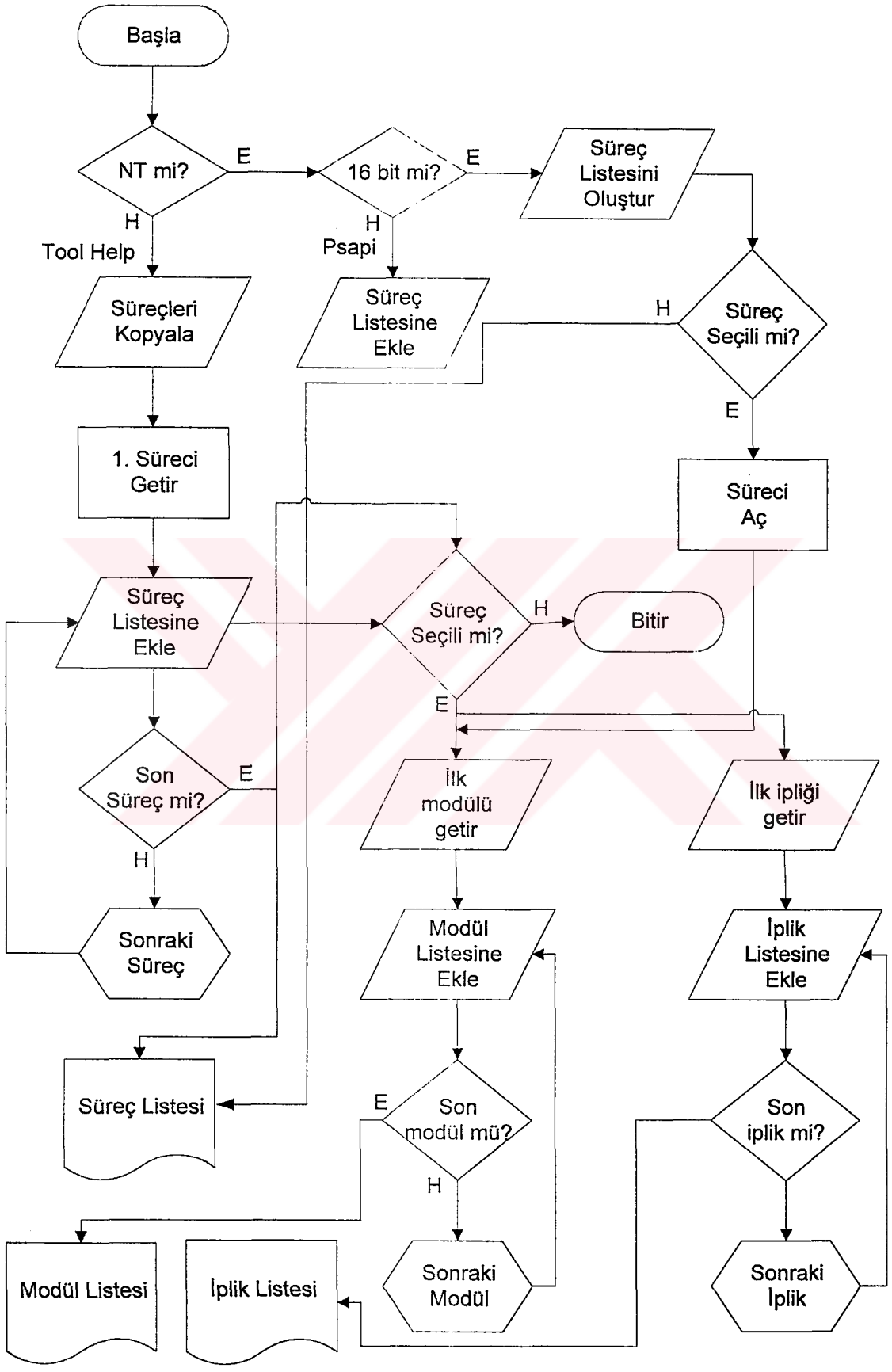
Şekil 4.1'de yapılan çalışmanın blok şeması gösterilmiştir. Sistemde cereyan eden olayları tespit etmek için öncelikle hangi metodun kullanılacağı belirlenir. Windows 9x/Me işletim sistemlerindeki tüm uygulamalar WIN metodu ile tespit edilir. Windows 2000/XP işletim sistemlerinde 32 bit uygulamalar WIN metodu ile tespit edilirler. Windows NT/2000/XP işletim sistemlerindeki 16 bit uygulamalar NT-16 bit metodu ile tespit edilebilir. Windows NT işletim sistemlerinde 32 bit uygulamalar NT-32 bit metodu ile tespit edilirler. 16 bit ve 32 bit uygulamalar için farklı yöntemlerin kullanılmasının sebebi NT/2000/XP işletim sistemlerinde DOS işlemlerinin bir sanal DOS sürücüsü ile gerçekleştirilmesinden kaynaklanmaktadır.(18)

Çalışan bütün süreçler, süreç listesine eklenir. Süreç listesindeki herhangi bir sürecin modülleri ve iplikleri, Windows 9x/Me/2000/XP işletim sisteminde Tool Help fonksiyonları, NT işletim sistemlerinde ise PSAPI fonksiyonları vasıtasıyla listelenmiştir.

DOST programı birçok yordamdan oluşmuştur. Bu yordamlar hali hazırda yürütülen süreçleri tespit etmek, listelemek, süreçlere ait modülleri ve iplikleri meydana çıkarmak ve bu modül ve iplikleri listelemek için kullanılmışlardır.

Şekil 4.2'de DOST'un akış çizelgesi yer almaktadır. Akış çizelgesinde ilk olarak DOST, kullanılan işletim sisteminin NT olup olmadığını kontrol eder. Zira DOST'un blok şemasından da anlaşılacağı üzere sistemde cereyan eden olayların tespit edilebilmesinde, kullanılan işletim sisteminin ve uygulamanın tipinin (16 bit, 32 bit) ne olduğu önem arz etmektedir.

Kullanılan işletim sistemi belirlendikten sonra ilgili fonksiyonlar kullanılarak çalışan süreçlerin listesine erişilir. İlk olarak listedeki ilk süreç getirilir. Bu sürecin listedeki son süreç olma durumu kontrol edilir. Son süreç ise ilk süreç, süreç listesine eklenir ve süreç listesi görüntülenir. Son süreç değilse listedeki sonraki süreç çağrılarak



Şekil 4.2. DOST'un Akış Çizelgesi

süreç listesine eklenir. Bu işlemler listedeki süreçler bitene kadar sürdürülür. Son süreç de süreç listesine eklendikten sonra sistemde çalışan süreçler görüntülenir.

Süreç listesinde yer alan herhangi bir sürecin modüllerini veya ipliklerini listelemek için ilgili süreç seçilmelidir. Bir sürece ait modüller şöyle listelenir:
İlk olarak seçili sürecin ilk modülüne erişilerek, ilk modül, modül listesine eklenir. Erişilen modülün son modül olma durumu kontrol edilir. Son modül ise modül listesi görüntülenir. Son modül değilse, modüller bitene kadar sonraki modüller çağrılarak modül listesine eklenirler. Eklenecek modül kalmadığında modül listesi görüntülenir.

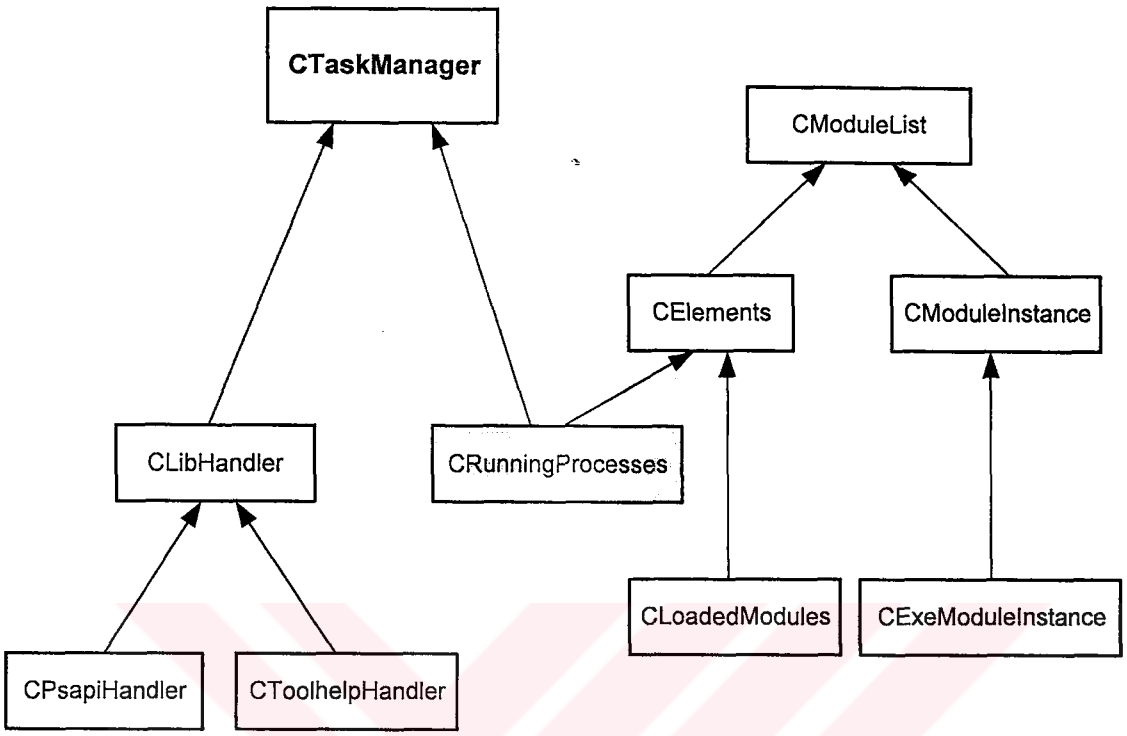
Bir sürece ait iplikleri listelemek için seçili sürecin ilk ipliğine erişilerek, ilk iplik, iplik listesine eklenir. Erişilen ipliğin son iplik olma durumu kontrol edilir. Son iplik ise iplik listesi görüntülenir. Son iplik değilse, iplikler bitene kadar sonraki iplikler çağrılarak iplik listesine eklenirler. Eklenecek iplik kalmadığında iplik listesi görüntülenir.

Süreçler, modüller ve iplikler listelendikten sonra program sonlandırılır.

5 DOST'UN ÇALIŞMASI

Sistemde çalıştırılan olayların izlenmesi için geliştirilen bir program kompleks bir yapıya sahiptir. Bunun en önemli sebebi bu amacın yerine getirilmesi için Win32 işletim sistemindeki birçok farklı API'nin kullanılacak olmasıdır. Windows 95, Windows 98, Windows Millenium Edition, Windows 2000 ve Windows XP işletim sistemlerinde bu amaç için ToolHelp32 kütüphanesindeki API fonksiyonları kullanılabilir. Aynı işlem için Windows NT sistemlerinde ise PSAPI kütüphanesindeki API fonksiyonları kullanılmalıdır.

Şekil 5.1'de sistemde çalışan süreçlerin sıralanmasını sağlayan alt sistemler ve bunların birbirleri arasındaki ilişkiler gösterilmiştir. Burada yer alan CTaskManager öğesi sistemin işlemcisi yerindedir. CTaskManager, kullanılan işletim sistemine göre tercih edilmesi gereken fonksiyon kütüphaneleri için bir yürütücü değer üretmekle sorumludur (CLibHandler). Süreçler hakkında doğru bilgi elde edebilmek için doğru kütüphane ile çalışmak (PSAPI veya ToolHelp32) gereklidir. CTaskManager, CpsapiHandler (veya CToolhelpHandler) parametre değerlerine göre hali hazırdaki aktif süreçlerin bir listesini hazırlar ve bu listeyi bekletir. Oluşturulan listedeki bilgilere başka bir fonksiyon çağrılarak erişilebilir (bütün süreçlerin sıralanması, DLL kütüphane dosyalarına erişilmesi, tutuldukları hiyerarşi bilgileri, ...)



Şekil 5.1. Süreçlerin Sıralandırılması (19)

5.1 Süreçlerin Listelenmesi

DOST, hali hazırda yürütülen süreçleri listelemek için Windows 9x/2000/Me/XP işletim sistemlerinde ToolHelp32 fonksiyonlarını, Windows NT tabanlı işletim sistemlerinde ise PSAPI fonksiyonlarını kullanır. Windows 9x/Me işletim sistemlerinde çalıştırılan 16 bit uygulamaları listelemek için de ToolHelp32 fonksiyonları kullanılır. Windows NT/2000/XP işletim sistemlerinde 16 bit uygulamaları sanal bir DOS sürücüsü ile çalıştırılır. Windows NT/2000/XP işletim sistemlerinde çalıştırılan 16-bit uygulamaları listelemek için VDMEnumTaskWOWEx() fonksiyonu kullanılır. Bu fonksiyonu çalıştırmak için VDMDBG.h ve VDMDBG.lib dosyalarının programa dahil edilmesi gerekir. Bu dosyalar "ntvdm.exe" programı kullanılarak işletilebilir.

ToolHelp32 fonksiyonları kernel.dll dosyasında bulunurlar. Standart API fonksiyonları ile aynı özelliktedirler. Bu fonksiyonlar Windows NT sistemlerde

çalışmazlar. ToolHelp32, süreçleri ve iplikleri listelemek için birçok fonksiyona sahiptir. Süreçleri tespit etmek için bu fonksiyonlardan üç tanesi yeterlidir: CreateToolhelp32Snapshot(), Process32First() ve Process32Next().

Windows NT sistemlerinde süreçlerin listesini oluşturmak için PSAPI fonksiyonları kullanılırlar. PSAPI fonksiyonları psapi.dll dosyasında bulunurlar. Bu fonksiyonların kullanılabilmesi için psapi.h ve psapi.lib dosyalarının hazırlanan projeye dahil edilmesi gerekir. ToolHelp32 fonksiyonlarına benzer olarak PSAPI kütüphanesinde de pek çok yararlı fonksiyon vardır. Süreçleri sıralamak için EnumProcesses() fonksiyonu kullanılır.

Toolhelp fonksiyonlarını kullanmanın ilk adımı sistemden bir görüntü bilgisi almaktır. Bunu yapmak için CreateToolhelp32Snapshot() fonksiyonu kullanılır. Bu fonksiyon ile oluşturulan görüntü bilgisi, o anda çalıştırılan bütün uygulamalar ve bu uygulamaların alt yordamları hakkındaki bilgileri kapsar. Örneğin süreç bilgisine erişmek isteniyorsa, bu fonksiyon TH32CS_SNAPPROCESS bayrağı ile çağrılmalıdır. CreateToolhelp32Snapshot() fonksiyonu, CloseHandle() fonksiyonuna parametre olarak geçirilecek bir yürütücü değer döndürür.

Görüntüdeki süreçlerin listesini almak için öncelikle listedeki ilk sürece erişilmelidir. Listedeki birinci sürece GetProcessFirst fonksiyonu ile erişilir. Aşağıda bu fonksiyonun kodları verilmiştir.

Fonksiyon 5.1.1. GetProcessFirst Fonksiyonu

```
(0) BOOL CEnumProcess::GetProcessFirst(CEnumProcess::CprocessEntry
(1)   *pEntry){
(2)   if (ENUM_METHOD::NONE == m_method) return FALSE;
(3)   if ((ENUM_METHOD::TOOLHELP|m_method) == m_method){
(4)         m_hProcessSnap=FCreateToolhelp32Snapshot(TH32CS_
            SNAPPROCESS ,0);
(5)         INVALID_HANDLE_VALUE == m_hProcessSnap) return FALSE;
(6)         if (!FProcess32First(m_hProcessSnap, &m_pe)) return
            FALSE;
(7)         pEntry->dwPID = m_pe.th32ProcessID;
(8)         strcpy(pEntry->lpFilename, m_pe.szExeFile);
(9)   }
(10)  else{
```

```
(11)        if (m_pProcesses) {delete[] m_pProcesses;}
(12)        m_pProcesses = new DWORD[m_MAX_COUNT];
(13)        m_pCurrentP = m_pProcesses;
(14)        DWORD cbNeeded = 0;
(15)        BOOL OK = FEnumProcesses(m_pProcesses,
            m_MAX_COUNT*sizeof(DWORD), &cbNeeded);
(16)        if (cbNeeded >= m_MAX_COUNT*sizeof(DWORD)){
(17)              m_MAX_COUNT += 256;
(18)              return GetProcessFirst(pEntry);
(19)              }
(20)        if (!OK) return FALSE;
(21)        m_cProcesses = cbNeeded/sizeof(DWORD);
(22)        return FillPStructPSAPI(*m_pProcesses, pEntry);
(23)  }
(24)  return TRUE;
(25) }
```

Windows sistemlerde süreç listesinin görüntüsünü almak için CreateToolhelp32Snapshot() fonksiyonu TH32CS_SNAPPROCESS bayrağı ile çağrılır. Fonksiyon 5.1.1'in 4. satırında ilgili kodlar yer almaktadır. Görüntü alındıktan sonra listedeki ilk sürece Process32First fonksiyonu ile erişilir. 6. satırda Process32First fonksiyonundan bir değer döndürülmemişse GetProcessFirst fonksiyonu FALSE döndürür. Aksi takdirde dönen değerler PROCESSENTRY32 (m_pe) yapısında depo edilir. DOST'da m_pe yapısında sürecin tanımlayıcısı (th32ProcessID) ve konumu (szExeFile) tutulmaktadır.

NT sistemlerde süreçlerin listesini oluşturmak için EnumProcesses fonksiyonu kullanılır. Bu fonksiyon ile yürütülen süreçlerin listesi bir diziye aktarılır. Fonksiyon parametre olarak hali hazırda yürütülen süreçlerin tanımlayıcılarının bulunduğu diziyi gösteren bir işaretçi (m_pProcesses), dizinin büyüklüğü (m_MAX_COUNT) ve süreçlerin listeleneceği ekstra dizi alan (cbNeeded) değerlerini alır. GetProcessFirst fonksiyonunun 11. satırında süreç listesinin aktarılacağı dizi boşaltılmıştır. 12. satırda çalışan süreçlerin büyüklüğü kadar bir dizi açılır ve 14. satırda bu diziye EnumProcesses() fonksiyonu ile süreçlerin listesi yerleştirilir. Dizinin büyüklüğü yeterli değilse cbNeeded dizisi de kullanılabilir. GetProcessFirst fonksiyonundan dönen değerler aşağıda Fonksiyon 5.1.2'de kodları verilmiş FillStructPSAPI fonksiyonundaki yapıya gönderilir.

Sistemde çalıştırılan ilk sürece eriştikten sonra sonraki süreçlere ulaşmak için kodları aşağıda verilen GetProcessNext fonksiyonu kullanılır.

Fonksiyon 5.1.2. GetProcessNext Fonksiyonu

```
(0) BOOL CEnumProcess::GetProcessNext(CEnumProcess::CProcessEntry
*pEntry){
(1)    if (ENUM_METHOD::NONE == m_method) return FALSE;
(2)    pEntry->hTask16 = 0;
(3)    #ifdef USE_VDMDBG
(4)          if (g_Processes16.GetSize() > 0){
(5)                CEnumProcess::CProcessEntry *pTmp =
                   g_Processes16[0];
(6)                g_Processes16.RemoveAt(0);
(7)                strcpy(pEntry->lpFilename, pTmp->lpFilename);
(8)                pEntry->dwPID = pTmp->dwPID;
(9)                pEntry->hTask16 = pTmp->hTask16;
(10)               delete pTmp;
(11)               return TRUE;
(12)         }
(13)   #endif
(14)   if ((ENUM_METHOD::TOOLHELP|m_method) == m_method){
(15)         if (!FProcess32Next(m_hProcessSnap, &m_pe)) return
             FALSE;
(16)         pEntry->dwPID = m_pe.th32ProcessID;
(17)         strcpy(pEntry->lpFilename, m_pe.szExeFile);
(18)   }
(19)   else{
(20)         if (--m_cProcesses <= 0) return FALSE;
(21)         FillPStructPSAPI(*++m_pCurrentP, pEntry);
(22)   }
(23)   #ifdef USE_VDMDBG
(24)         if ((ENUM_METHOD::PROC16|m_method) == m_method)
(25)         if( 0 == _stricmp( pEntry->lpFilename+(strlen(pEntry->
             lpFilename)-9), "NTVDM.EXE")){
(26)               FVDMEnumTaskWOWEx( pEntry->dwPID, (TASKENUMPROCEX)
                   &Enum16Proc, (LPARAM) pEntry->dwPID);
(27)               }
(28)   #endif // USE_VDMDBG
(29)   return TRUE;
(30) }
```

Windows sistemlerde sonraki süreçleri getirmek için tekrarlı olarak Process32Next fonksiyonları çağrılır. GetProcessNext fonksiyonunun 15. satırında sonraki süreç bilgileri getirilerek, 16. ve 17. satırlarda m_pe yapısına yerleştirilmiştir. NULL değeri döndürülüne kadar Process32Next fonksiyonu çağrılmaya devam eder.

NT sistemlerde süreçlerin listesi GetProcessFirst fonksiyonunda oluşturulmuştur. GetProcessNext fonksiyonunda dizinin sonraki süreçleri 21. satırda süreçlerin tutulduğu listeye eklenir. Fonksiyon 5.1.3'de NT sistemlerde çalışan süreçlerin listelenmesine ait program kodları verilmiştir.

Fonksiyon 5.1.3. FillPStructPSAPI Fonksiyonu

```
(0) BOOL CEnumProcess::FillPStructPSAPI(DWORD dwPID,
CEnumProcess::CProcessEntry* pEntry){
(1)    pEntry->dwPID = dwPID;
(2)    HANDLE hProc = OpenProcess(PROCESS_QUERY_INFORMATION |
       PROCESS_VM_READ, FALSE, dwPID);
(3)    if (hProc){
(4)          HMODULE hMod;
(5)          DWORD size;
(6)          if( FEnumProcessModules(hProc, &hMod, sizeof(hMod),
             &size)){
(7)                if( !FGetModuleFileNameEx( hProc, hMod, pEntry-
                   >lpFilename, MAX_FILENAME) ){
(8)                      strcpy(pEntry->lpFilename, "N/A (HATA!)");
(9)                }
(10)         }
(11)         CloseHandle(hProc);
(12)   }
(13)   else strcpy(pEntry->lpFilename, "N/A (Erişim hakkınız yok)");
       return TRUE;
(14)  }
```

Süreçlerin adlarını listelemek için OpenProcess() fonksiyonu ile ilgili sürecin açılması gerekir. FillPStructPSAPI fonksiyonunun 2. satırında, GetProcessNext fonksiyonundan gönderilen süreçler (dwPID değeri ile), süreç adları ve konumları belirlenmek için OpenProcess() fonksiyonu ile oku modunda açılır. 3 ile 11. satırlar arasında sürecin adı ve konumu ilgili yapıya yerleştirilerek (pEntry), süreç CloseHandle() fonksiyonu ile kapatılır.Getirilen sürece, kullanıcının erişim hakkı yok ise pEntry yapısının süreç ismi kısmına "Erişim Hakkınız Yok" mesajı yazdırılır.

NT/2000/XP işletim sistemlerinde 16-bit süreçlerini listelemek için VDMEnumTaskWOWEx() fonksiyonu kullanılır. Bu fonksiyonun çalıştırılması için ntdvm.exe dosyası gereklidir. GetProcessNext fonksiyonu 16-bit süreçlerini listelemek için 25. satırda ntdvm.exe dosyası tanımlanıp, 26. satırda VDMEnumTaskWOWEx() fonksiyonu ile çalışan 16-bit süreçleri Enum16Proc

fonksiyonuna geçirilmiştir. EnumProc16 fonksiyonu gelen süreçleri süreçlerin listesinin tutulduğu yapıya eklemekle görevlidir. Enum16Proc fonksiyonu aşağıda verilmiştir.

Fonksiyon 5.1.4. Enum16Proc Fonksiyonu

```
(0) #ifdef USE_VDMDBG
(1)   BOOL CALLBACK Enum16Proc( DWORD dwThreadId, WORD hMod16, WORD
      hTask16, PSZ pszModName, PSZ pszFileName, LPARAM
      lpUserDefined){
(2)         CEnumProcess::CProcessEntry* pEntry = new
            CEnumProcess::CProcessEntry();
(3)         strcpy(pEntry->lpFilename, pszFileName);
(4)         pEntry->dwPID = (DWORD) lpUserDefined;
(5)         pEntry->hTask16 = hTask16;
(6)         g_Processes16.Add(pEntry);
(7)         return FALSE;
(8)   }
(9) #endif
```

5.2 Modüllerin Listelenmesi

DOST çalışan süreçlerin listesini görüntüledikten sonra süreç listesinde yer alan süreçlerden her birine ait bütün modülleri de listeleyebilir. Bunun için süreç listesinden modülleri listelenmesi istenen sürecin seçilmesi gerekir. DOST çalışan uygulamaların modüllerini listelemek için GetModuleFirst, GetModuleNext, FillMStructPSAPI ve GetModulePreferredBase fonksiyonlarını kullanır.

Aşağıda sürecin ilk modülüne ait bilgilere sahip olmak için kullanılan GetModuleFirst fonksiyonunun kodları yer almaktadır.

Fonksiyon 5.2.1. GetModuleFirst Fonksiyonu

```
(0) BOOL CEnumProcess::GetModuleFirst(DWORD dwPID,
CEnumProcess::CModuleEntry *pEntry){
(1)   if (ENUM_METHOD::NONE == m_method) return FALSE;
(2)   if ((ENUM_METHOD::TOOLHELP|m_method) == m_method){
(3)         if (INVALID_HANDLE_VALUE != m_hModuleSnap)
            ::CloseHandle(m_hModuleSnap);
(4)         m_hModuleSnap =
            FCreateToolhelp32Snapshot(TH32CS_SNAPMODULE, dwPID);
(5)         if(!FModule32First(m_hModuleSnap, &m_me)) return FALSE;
(6)         pEntry->pLoadBase = m_me.modBaseAddr;
```

```
(7)          strcpy(pEntry->lpFilename, m_me.szExePath);
(8)          strcpy(pEntry->lpModul,m_me.szModule);
(9)          pEntry->pPreferredBase = GetModulePreferredBase(dwPID,
             m_me.modBaseAddr);
(10)         return TRUE;
(11)   }
(12)   else{
(13)   if (m_pModules) {delete[] m_pModules;}
(14)         m_pModules = new HMODULE[m_MAX_COUNT];
(15)         m_pCurrentM = m_pModules;
(16)         DWORD cbNeeded = 0;
(17)         HANDLE hProc = OpenProcess(PROCESS_QUERY_INFORMATION |
             PROCESS_VM_READ, FALSE, dwPID);
(18)         if (hProc){
(19)               BOOL OK = FEnumProcessModules(hProc, m_pModules,
                   m_MAX_COUNT*sizeof(HMODULE), &cbNeeded);
(20)               CloseHandle(hProc);
(21)               if (cbNeeded >= m_MAX_COUNT*sizeof(HMODULE)){
(22)                     m_MAX_COUNT += 256;
(23)                     return GetModuleFirst(dwPID, pEntry);
(24)               }
(25)               if (!OK) return FALSE;
(26)               m_cModules = cbNeeded/sizeof(HMODULE);
(27)               return FillMStructPSAPI(dwPID, *m_pCurrentM,
                   pEntry);
(28)         }
(29)         return FALSE;
(30)   }
(31)  }
```

Windows işletim sistemlerinde işlenen sürecin modüllerini listelemek için öncelikle modül verilerini kapsayacak bir sistem görüntüsü almak gerekir. Bu amaçla GetModuleFirst fonksiyonunun 4. satırında CreateToolhelp32Snapshot fonksiyonu TH32CS_SNAPMODULE parametresi ve ilgili sürecin tanımlayıcısı (dwPID) ile birlikte çağrılmıştır. Sürecin ilk modülüne 5. satırda Module32First fonksiyonu çağrılarak ulaşılmıştır. 6 ile 9. satırlar arasında ilk modüle ait bilgiler MODULEENTRY32 yapısında tanımlanan m_me dizisine aktarılmıştır. DOST'da m_me yapısında modülün adresi (modBaseAddress), büyüklüğü (szModule) ve çalıştırılabilir dosyasının konumu (szExePath) tutulmaktadır.

NT sistemlerde bir sürecin modüllerini listelemek için PSAPI kütüphanesindeki EnumProcessModules() ve GetModuleFileNameEx() fonksiyonları kullanılır. Süreçlerin modüllerine erişmek için EnumProcessModules() fonksiyonu kullanılır. Bu fonksiyon hangi sürece ait modülleri listeleyeceğini belirleyen parametreyi GetProcessFirst veya GetProcessNext fonksiyonlarından alır (dwPID).

GetModuleFirst fonksiyonunun 19. satırında ilgili sürece ait ilk modülün bilgisi alınarak bir diziye yerleştirilmiştir. İlk modül bilgisini, modüller listesine eklemek için 27. satırda FillMStructPSAPI fonksiyonuna gönderilmiştir. Aşağıda FillMStructPSAPI fonksiyonunun kodları yer almaktadır.

Fonksiyon 5.2.2. FillMStructPSAPI Fonksiyonu

```
(0) BOOL CEnumProcess::FillMStructPSAPI(DWORD dwPID, HMODULE mMod,
CEnumProcess::CModuleEntry *pEntry){
(1)    HANDLE hProc = OpenProcess(PROCESS_QUERY_INFORMATION |
       PROCESS_VM_READ, FALSE, dwPID);
(2)    if (hProc){
(3)          if( !FGetModuleFileNameEx( hProc, mMod, pEntry-
             >lpFilename, MAX_FILENAME) ){
(4)                strcpy(pEntry->lpFilename, "N/A (HATA!)");
(5)          }
(6)          pEntry->pLoadBase = (PVOID) mMod;
(7)          pEntry->pPreferredBase = GetModulePreferredBase(dwPID,
             (PVOID)mMod);
(8)          CloseHandle(hProc);
(9)          return TRUE;
(10)   }
(11)   return FALSE;
(12) }
```

FillMStructPSAPI fonksiyonunun 3. satırında GetModuleFileNameEx() fonksiyonu çağrılarak modül bilgileri getirilir. Getirilecek modül bilgisi yok ise modül listesine, modül ismi olarak "HATA!" mesajı yazdırılır. Eğer modül var ise 7. satırda görüleceği üzere modüle ait verilere ulaşmak için GetModulePreferredBase fonksiyonu çağrılır. GetModulePreferredBase fonksiyonunun kodları aşağıda gösterilmiştir.

Fonksiyon 5.2.3. GetModulePreferredBase Fonksiyonu

```
(0)PVOID CEnumProcess:: GetModulePreferredBase(DWORD dwPID, PVOID
pModBase){
(1)    if (ENUM_METHOD::NONE == m_method) return NULL;
(2)    HANDLE hProc = OpenProcess(PROCESS_VM_READ, FALSE, dwPID);
(3)    if (hProc){
(4)          IMAGE_DOS_HEADER idh;
(5)          IMAGE_NT_HEADERS inh;
(6)          ReadProcessMemory(hProc, pModBase, &idh, sizeof(idh),
             NULL);
(7)          if (IMAGE_DOS_SIGNATURE == idh.e_magic)
```

```
                ReadProcessMemory(hProc, (PBYTE)pModBase +
                idh.e_lfanew, &inh, sizeof(inh), NULL);
(8)         CloseHandle(hProc);
(9)         if (IMAGE_NT_SIGNATURE == inh.Signature)
                return (PVOID) inh.OptionalHeader.ImageBase;
(10)  }
(11)  return NULL;
(12) }
```

GetModulePreferredBase fonksiyonunun 6. satırında ReadProcessMemory() fonksiyonu ile modülün verileri okunur. Bu fonksiyondan dönen değerler, FillMStructPSAPI fonksiyonunun 6 ve 7. satırlarında modüle ait bilgi listesinin tutulduğu yapıya aktarılır.

İlk modül bilgileri modül listesine eklendikten sonra sürecin Sonraki modülünden başlayarak modül bilgilerinin eklenmesi, eklenecek modül bilgisi kalamayana kadar sürdürülür. Sonraki modül verilerine erişmek için GetModuleNext fonksiyonu kullanılır.

Fonksiyon 5.2.3. GetModuleNext Fonksiyonu

```
(0) BOOL CEnumProcess::GetModuleNext(DWORD dwPID,
CEnumProcess::CModuleEntry *pEntry){
(1)    if (ENUM_METHOD::NONE == m_method) return FALSE;
(2)    if ((ENUM_METHOD::TOOLHELP|m_method) == m_method){
(3)         if(!FModule32Next(m_hModuleSnap, &m_me)) return FALSE;
(4)         pEntry->pLoadBase = m_me.modBaseAddr;
(5)         strcpy(pEntry->lpFilename, m_me.szExePath);
(6)         strcpy(pEntry->lpModul,m_me.szModule);
(7)         pEntry->pPreferredBase = GetModulePreferredBase(dwPID,
            m_me.modBaseAddr);
(8)         return TRUE;
(9)    }
(10)   else{
(11)   if (--m_cModules <= 0) return FALSE;
            return FillMStructPSAPI(dwPID, *++m_pCurrentM, pEntry);
(12)   }
(13) }
```

Windows sistemlerde sürecin sonraki modül bilgisine ulaşmak için Module32Next fonksiyonu kullanılır. Bu fonksiyondan dönen değerler GetModuleNext fonksiyonunun 4 ile 7. satırlarında modül listesinin tutulduğu yapıda (m_me) ilgili alanlarına geçirilir.

NT sistemlerde sonraki modül bilgileri modül listesinin tutulduğu yapıya FillMStructPSAPI fonksiyonu (fonksiyon 5.2.2) kullanılarak aktarılır.

5.3 İpliklerin Listelenmesi

İşlenen sürecin ipliklerini listelemek için öncelikle iplik bilgisini kapsayacak bir sistem görüntüsü almak gerekir. Bu amaçla CreateToolhelp32Snapshot fonksiyonu TH32CS_SNAPTHREAD parametresi ve ilgili sürecin tanımlayıcısı (dwPID) ile birlikte çağrılır. İşlenen sürecin ilk ipliğine erişmek için GetThreadFirst fonksiyonu kullanılır.

Fonksiyon 5.3.1. GetThreadFirst Fonksiyonu

```
(0) BOOL CEnumProcess::GetThreadFirst(DWORD dwPID,
CEnumProcess::CThreadEntry *pEntry){
(1)    if (ENUM_METHOD::NONE == m_method) return FALSE;
(2)    if ((ENUM_METHOD::TOOLHELP|m_method) == m_method){
(3)          if (INVALID_HANDLE_VALUE !=
                   m_hThreadSnap)::CloseHandle(m_hThreadSnap);
(4)                m_hThreadSnap =
                   FCreateToolhelp32Snapshot(TH32CS_SNAPTHREAD,
                   dwPID);
(5)          if(!FThread32First(m_hThreadSnap, &m_te)) return FALSE;
(6)          pEntry->dwSize = m_te.dwSize;
(7)          pEntry->ThreadID = m_te.th32ThreadID;
(8)          pEntry->Owner= m_te.th32OwnerProcessID;
(9)          return TRUE;
(10)   }
(11)  }
```

GetThreadFirst fonksiyonunun 4. satırında ilgili sürece ait görüntü alınarak m_hThreadSnap yapısına aktarılır. Bu yapının ilk ipliğinin bilgilerine ulaşmak için Thread32First fonksiyonu kullanılır. Bu fonksiyondan dönen değerler m_te yapısında depolanır. DOST'da m_te yapısında ipliğin büyüklük (dwSize), tanımlayıcı (ThreadID) ve bağlı bulunduğu sürecin tanımlayıcısı (Owner) değerleri tutulmaktadır.

İlk iplik işlendikten sonra sonraki ipliklerin bilgilerine ulaşmak için GetThreadNext fonksiyonu kullanılmıştır. GetThreadNext fonksiyonunun kodları Fonksiyon 5.3.2'de verilmiştir.

Fonksiyon 5.3.2. GetThreadNext Fonksiyonu

```
(0) BOOL CEnumProcess::GetThreadNext(DWORD dwPID,
CEnumProcess::CThreadEntry *pEntry){
(1)    if (ENUM_METHOD::NONE == m_method) return FALSE;
(2)    if ((ENUM_METHOD::TOOLHELP|m_method) == m_method){
(3)          if(!FThread32Next(m_hThreadSnap, &m_te)) return FALSE;
(4)          pEntry->dwSize = m_te.dwSize;
(5)          pEntry->ThreadID = m_te.th32ThreadID;
(6)          pEntry->Owner= m_te.th32OwnerProcessID;
(7)          return TRUE;
(8)    }
(9) }
```

GetThreadNext fonksiyonunun 3. satırında ilgili sürecin sonraki ipliğine ait verilerin görüntüsü oluşturulup oluşturulmadığı kontrol edilir. Görüntü var ise Thread32Next fonksiyonu ile iplik bilgileri THREADENTRY32 olarak tanımlanan m_te yapısında depolanır. DOST'ta iplik tanımlayıcısı, büyüklük ve bağlı bulunan süreç tanımlayıcısı bilgileri tutulmaktadır.

6 ÇALIŞMANIN DEĞERLENDİRİLMESİ

Bu çalışmanın tamamlanması ile bilgisayar kullanılırken işletilen programların ve bu programlara ait süreçlerin, modüllerin, ipliklerin listelenmesi başarılmıştır. Kullanıcı geliştirilen bu program ile sistemde cereyan eden bütün olayların listesini takip edebilecek bir araca sahip olmuştur.

Bilgisayar ile yapılan işlemlerin nasıl gerçekleştirildiği konusu giderek bir bilinmeyen haline gelmektedir. Yapılan bu çalışma ile bu bilinmeyen olayların önüne geçilerek, gerçekte nelerin olduğu ve nasıl gerçekleştirildiği sorularına cevaplar bulmaya yardım edecek bir araç geliştirilmiştir.

Sistemde çalışan uygulamaları ve bunların alt yordamlarını listelemek için özel kütüphane fonksiyonları kullanılmıştır. Özel kütüphane fonksiyonları (Tool Help veya PSAPI) kullanılarak sistemde çalışan uygulamaların ve bu uygulamaların alt yordamlarının izlenmesi, kanca fonksiyonlarına nazaran daha kolay gerçekleştirilebilir. Dosya takibi için sadece gerekli Tool Help (veya PSAPI) fonksiyonları ve bunların eriştiği yapıları bilmek yeterlidir.

Kütüphane fonksiyonları çalışan uygulamaları, sistemin o andaki durumunun görüntüsünü alarak listelerler. Yani sistemi sadece bir anlık meşgul eder. Bu yüzden sistem performansına olumsuz etkileri yok denecek kadar azdır.

Kütüphane fonksiyonları ile sistem fonksiyonları özdeş yapıda olduklarından dolayı kütüphane fonksiyonları çalıştırılırken sistem hasarlarına, çökmelere, uyuşmazlıklara neden olmazlar.

Dosya takibi, kanca fonksiyonları ile gerçekleştirilmesi sisteme gelen her iletinin öncelikle kanca fonksiyonlarında değerlendirilme zorunluluğunu doğurur. Kanca fonksiyonları, giriş iletileri ile bu iletilerin hedeflediği kaynaklar arasına girerler. Yani kanca fonksiyonları Windows ileti sisteminin bir öğesi gibi davranırlar. Bu da

beraberinde sistem için bazı olumlu ve olumsuz katkılar getirir. Kanca fonksiyonların sisteme getirdiği avantajlar şöyle sıralanabilir:

- Bu mekanizma, Windows işletim sistemi ailesinin 9x, NT, 2000 sürümleri tarafından desteklenmektedir ve gelecek Windows işletim sistemleri tarafından da desteklenmesi sürdürülecektir.

- Dosya takibinde kayıt düzenleyicisi dll dosyaları yerine sistem kancalarının kullanılması daha iyidir. Çünkü bir kanca fonksiyonun çağrılması ile yaptırılan bir çalışma sonlandırıldıktan sonra ilgili sistem kancası kaldırılarak sistemdeki yük azaltılır. Dll dosyaları ise sistem kancaları gibi kaldırılamazlar.

Kanca sisteminin dezavantajları ise şunlardır:

- Windows kancaları sistem performansını düşürürler. Bunun sebebi sistemin her bir iletiyi işleyecek olması sebebi ile sistemdeki işlenecek olayların sayısının artmasıdır.

- Sistem kancaları için hata ayıklama işlemi gerçekleştirmek güçtür. Her bir sistem kancasına ait birçok parametre değeri, fonksiyon ve yordam vardır.

- Kancalar, bütün sistemin çalışmasını etkilerler. Kancaların çalışması sırasında olabilecek bir hata durumunda, sistemi tekrar ele geçirmek için bilgisayarın yeniden başlatılması gerekir.

Kanca yordamlarının kütüphane fonksiyonlarına göre avantajı ileti sistemine müdahil olarak her iletiyi değerlendirme kabiliyetine sahip olmasıdır. Kütüphane fonksiyonları sadece olan olayları gösterirler, olmadan müdahale etme olanakları yoktur. Kanca yordamları ise olayların gerçekleşmesine izin vermeyebilirler.

T.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ

Hazırlanan bu uygulama ile sistemde gerçekleştirilen bütün olaylar takip edilebilir. Sistemde çalıştırılan fakat Ctrl+Alt+Del tuşlarına basıldığında çalıştığı gösterilmeyen uygulamalar da bu çalışma ile görüntülenebilmiştir. Bu sayede çalışan uygulamalar arasında casus yazılımlar gibi istenmeyen programlar tespit edilerek, sistemden kaldırılması sağlanabilir. Sistem bir uygulamayı çalıştırırken yaptıkları takip edilerek işletim sistemlerinin çalışma prensipleri kavranabilir.

Bu çalışma ile sistemde cereyan eden bütün olayların listelenmesi başarılmıştır. Çalışmanın bir sonraki adımında iyi niyetli yazılımlar ile kötü niyetli yazılımlar etiketlendirilebilir. Bu işlem şöyle gerçekleştirilebilir:
Bilinen kötü niyetli veya iyi niyetli yazılımlara eklenti olarak gelen kötü niyetli yazılımlar bir veri tabanında tutulurlar. DOST'un listesini tuttuğu dosya isimleri ile bu veri tabanında bulunan dosya isimleri karşılaştırılarak kötü niyetli olanlar bir işaretle belirtilir. Böylece kullanıcı çalışan uygulamaların hangisinin kötü niyetli olduğunu görebilir. Kötü niyetli uygulamaların sistemden kaldırılmasına da imkân tanınabilir.

Dosya takip programı geliştirmeni amacı hali hazırda yürütülen işlemlerin listesini görüntülemek ise özel kütüphane fonksiyonları kullanılmalıdır. Eğer hedeflenen dosya takibinin yanında bazı işlemlerin gerçekleşmesini engellemek ise kanca yordamları kullanılmalıdır.

KAYNAKLAR

1. Ivonov, I.., "API Hooking Revealed", 2002.
http://www.codeproject.com/system/hooksys.asp (23.07.2003)

2. Toth, V.., "Programming Windows 98/NT Unleashed", ***Sams Publishing,*** 36-60 (1998)

3. Villani, P., "Programming Win32 Under the API", ***CMP Books,*** Kansas, 105-133 (2001)

4. Matt, P., "Under the Hood", ***Microsoft System Journal,*** (11) 11 (1996)

5. Russinovich, M., Cogswell, B., "Examining VxD Service Hooking: Monitoring, altering, or otherwise changing parts of Windows", ***Dr. Dobb's Journal,*** (1996)

6. Redmond, W. A., "Programmer's Guide to Microsoft Windows 95", ***Microsoft Press,*** 459-480 (1995)

7. Petzold, C., "Programming Windows 5th edition", ***Microsoft Press,*** 41-71 (1998)

8. "Windows SDK Reference", www.msdn.com (23.07.2003)

9. Schildt, H., "Schildt's Advanced Windows 95 Programming in C and C++", ***Oracle Pr.,*** Philadelphia, 145-181 (1996)

10. Rector, B. E., Newconer, J. N., "Win32 Programming", ***Addison Wesley,*** Boston, 405-487 (1997)

11. Demirkol, B., "Windows API Programlama", ***Tubaysan,***
http://www.csystem.org/windows_api_programlama.htm (23.07.2003)

12. Pietrek, M., "Windows Internals", ***Addison Wesley,*** Boston, 299-363, 429-462 (1994)

13. Russinovich, M., Cogswell, B., "Examining the Windows 95 Layered File System", ***Dr. Dobb's Journal,*** (1995)

14. Marsh, K., "Win32 Hooks", (1994)
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwui/html/msdn_hooks32.asp (23.07.2003)

15. Hart, J. M., "Win32 System Programming 2nd Ed.", ***Addison Wesley,*** Boston, 173-215 (2000)

16. Dabak, P., Phadke, S., Borate, M., "Hooking System Services", Undocumented Windows NT, ***John Wiley & Song,*** 109-123 (1999)

17. Ritcher, J., "Load Your 32 bit DLL into Another Process's Address Space Using INJLIB", ***Microsoft System Journal,*** (9) 5 (1994)

18. Schreiber, S., "Undocumented Windows 2000 Secrets", ***Addison Wesley,*** Boston 34-43 (2001)

19. Ivonov, I., "Single Interface for Enumerating Processes and Modules under the Win 9x/2K", (2001) http://www.codeguru.com/system/PList.html (23.07.2003)

EKLER

Ek 1. Casus Yazılımların Listesi

7FaSSt	Lop.com
AdBreak	Marketscore(Netsetter)
Adultlinks Quickbar	N
AdvertBar	Ncase
Adware	NetRadar
Alexa	NetSource
Aureate	Netster Searchbar
Aveo Attune	NetworkEssentials
BonziBuddy	Netzany
BrilliantDigital	NewNet
BullaBHO	NowBox
CashBar	OnFlow
ClickTillUWin	OpenMe
CnsMin	Other
CometCursor	Porn Dialer
CommonName	Possible Browser Hijack attempt
Cydoor	Precisiontime/Gator/GAIN
Cytron	PurityScan
Datemanager/Gator	Radiate
DelFin/PromulGate	RapidBlaster
Dialer	SaveNow
Dialers	Scratch And Win
Doubleclick	Searchalot
DownloadWare	SearchExplorerBar
DSSAgent	SearchIt
EarnWithAds	SecretCrush
EverAd	SEXDIALER
EXact Search Bar	Showbehind
ExactSearchBar	Showserver(Telecoin) Dialer
Expedioware	Surf+
EzSearchbar	TGDC
EzuLa	TGDC(md)
FlashGet	Timesink
FlashTrack	TopMoxie
Flyswat	TotemAdverts
FriendGreetings	Tracking Cookie
Gator	Transponder
GoHip	TrustToolBar
Gratisware	TwistedHumor
HomepageWare	Ucmore
HotBar	Unknown
HuntToolBar	Web3000
IgetNet	WebHancer

ImIServer IEPlugin	WhenUSave
ImiServerIEPlugin	WinAD
IPInsight	WurldMedia
iwon	Xupiter
Kontiki	ZapSpot

T.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ

Ek 2. Özel Kütüphane Fonksiyonları

ToolHelp32 Fonksiyonları

CreateToolhelp32Snapshot: Belirtilen parametrelere göre sistemde çalıştırılan uygulamaların kopyasını alır.

```
HANDLE WINAPI CreateToolhelp32Snapshot(DWORD dwFlags,
                                       DWORD th32ProcessID);
```

dwFlags: Görüntüsü alınan sistem bölümüne referans eder. Değerleri;
TH32CS_INHERIT : Görüntü yürütücüsünü kalıtsallaştırır.
TH32CS_SNAPALL: TH32CS_SNAPHEAPLIST, TH32CS_SNAPMODULE, TH32CS_SNAPPROCESS ve TH32CS_SNAPTHREAD değerlerine eşittir.
TH32CS_SNAPHEAPLIST: Görüntüsü alınan sürecin yığıt listesini içerir.
TH32CS_SNAPMODULE: Görüntüsü alınan sürecin modül listesini içerir.
TH32CS_SNAPPROCESS: Görüntüsü alınan Win32 süreç listesini içerir.
TH32CS_SNAPTHREAD: Görüntüsü alınan Win32 iplik listesini içerir.

th32ProcessID: Belirlenen sürecin tanımlayıcısıdır. TH32CS_SNAPHEAPLIST veya TH32CS_SNAPMODULE ile birlikte kullanılır. Diğerlerinde yok sayılır.

Process32First: İlk sürecin bilgilerini getirir.

```
BOOL WINAPI Process32First(HANDLE hSnapshot, LPPROCESSENTRY32 lppe);
```

hSnapshot: CreateToolhelp32Snapshot fonksiyonundan dönen görüntü yürütücüsü.

lppe: PROCESSENTRY32 yapısının adresi.
PROCESSENTRY32, sistemden bir görüntü alındığında, sistem belleğinde bulunan süreçlerle ilgili bilgilerin tutulduğu yapılardır.

```
typedef struct tagPROCESSENTRY32 {
```

```
    DWORD dwSize; // yapının bayt olarak uzunluğu
    DWORD cntUsage; // sürecin referanslarının sayısı.
    DWORD th32ProcessID; // süreç tanımlayıcısı
    DWORD th32DefaultHeapID;//sürecin varsayılan küme tanımlayıcısı
    DWORD th32ModuleID; //sürecin modül tanımlayıcısı
    DWORD cntThreads; //süreçle çalışmaya başlayan iplik sayısı
    DWORD th32ParentProcessID; //bağlı olduğu sürecin tanımlayıcısı
    LONG  pcPriClassBase; //öncelikli iplikler
    DWORD dwFlags; //rezerve
    char szExeFile[MAX_PATH];//çalıştırıldığı dosyanın yolu ve adı
} PROCESSENTRY32;
typedef PROCESSENTRY32 *  PPROCESSENTRY32;
typedef PROCESSENTRY32 *  LPPROCESSENTRY32;
```

Process32Next: Görüntüde kayıtlı sonraki sürecin bilgisini getirir.

```
BOOL WINAPI Process32Next(HANDLE hSnapshot, LPPROCESSENTRY32 lppe);
```

hSnapshot: CreateToolhelp32Snapshot fonksiyonundan dönen görüntü yürütücüsü.

lppe: PROCESSENTRY32 yapısının adresi.

Thread32First: Bir sürecin ilk ipliğinin bilgilerini getirir.

```
BOOL WINAPI Thread32First(HANDLE hSnapshot, LPTHREADENTRY32 lpte);
```

hSnapshot: CreateToolhelp32Snapshot fonksiyonundan dönen görüntü yürütücüsü.

lpte: THREADENTRY32 yapısının adresi.

THREADENTRY32, sistemden bir görüntü alındığında, sistemde çalışan iplikler ile ilgili bilgilerin tutulduğu yapılardır.

```
typedef struct tagTHREADENTRY32{
    DWORD  dwSize;      //yapının bayt olarak uzunluğu
    DWORD  cntUsage;    //ipliğin referanslarının sayısı
    DWORD  th32ThreadID;      //iplik tanımlayıcısı
    DWORD  th32OwnerProcessID; //oluşturulduğu süreç tanımlayıcısı
    LONG   tpBasePri;         //başlatılma önceliği
    LONG   tpDeltaPri;        //öncelik durumundaki değişim
    DWORD  dwFlags;           //rezerve
} THREADENTRY32;
typedef THREADENTRY32 *  PTHREADENTRY32;
typedef THREADENTRY32 *  LTHREADENTRY32;
```

tpBasePri değişkeni şu değerleri alabilir:

THREAD_PRIORITY_IDLE, THREAD_PRIORITY_BELOW_NORMAL, THREAD_PRIORITY_LOWEST, THREAD_PRIORITY_NORMAL, THREAD_PRIORITY_ABOVE_NORMAL, THREAD_PRIORITY_HIGHEST, THREAD_PRIORITY_TIME_CRITICAL

Thread32Next: Bir sürecin sonraki ipliğinin bilgilerini getirir.

```
BOOL WINAPI Thread32Next(HANDLE hSnapshot, LPTHREADENTRY32 lpte);
```

hSnapshot: CreateToolhelp32Snapshot fonksiyonundan dönen görüntü yürütücüsü.
lpte: THREADENTRY32 yapısının adresi.

Module32First: İlgili sürecin ilk modülünün bilgisini getirir.

```
BOOL WINAPI Module32First(HANDLE hSnapshot, LPMODULEENTRY32 lpme);
```

hSnapshot: CreateToolhelp32Snapshot fonksiyonundan dönen görüntü yürütücüsü.

lpme: MODULEENTRY32 yapısının adresi.
MODULEENTRY32, belirlenen sürecin kullandığı modüllerle ilgili bilgilerin tutulduğu yapılardır.

```
typedef struct tagMODULEENTRY32 {
    DWORD   dwSize; //yapının bayt olarak uzunluğu
    DWORD   th32ModuleID; //modül tanımlayıcısı
    DWORD   th32ProcessID; //süreç tanımlayıcısı
    DWORD   GlblcntUsage; //global kullanılış sayısı
    DWORD   ProccntUsage; //sürecin modülü kullanış sayısı
    BYTE  * modBaseAddr; //temel adresi
    DWORD   modBaseSize; //modülün bayt olarak büyüklüğü
    HMODULE hModule; //yürütücü değeri
    char    szModule[MAX_MODULE_NAME32 + 1]; //modülün ismi
    char    szExePath[MAX_PATH]; //modülün yolu
} MODULEENTRY32;
typedef MODULEENTRY32 *  PMODULEENTRY32;
typedef MODULEENTRY32 *  LPMODULEENTRY32;
```

Module32Next: İlgili sürecin sonraki modülünün bilgisini getirir.

```
BOOL WINAPI Module32Next(HANDLE hSnapshot, LPMODULEENTRY32 lpme);
```

hSnapshot: CreateToolhelp32Snapshot fonksiyonundan dönen görüntü yürütücüsü.
lpme: MODULEENTRY32 yapısının adresi.

PSAPI Fonksiyonları

EnumProcesses: Sistemde çalışan her bir sürecin tanımlayıcısını getirir.

```
BOOL EnumProcesses(
  DWORD* lpidProcess,    // Süreç listesini alan dizi
  DWORD cb,              // Dizinin büyüklüğü (bayt)
  DWORD* cbNeeded        // Dizide kullanılan alan miktarı (bayt)
);
```

EnumProcessModules: Belirlenen sürecin her bir modülü için bir yürütücü değer getirir.

```
BOOL EnumProcessModules(
  HANDLE hProcess,       // Süreç yürütücüsü
  HMODULE* lphModule,    // Modül listesini alan dizi
  DWORD cb,              // Dizinin büyüklüğü (bayt)
  LPDWORD lpcbNeeded     // Dizide kullanılan alan miktarı (bayt)
);
```

GetModuleBaseName: Modülün esas adını getirmek için kullanılan fonksiyondur.

```
DWORD GetModuleBaseName(
  HANDLE hProcess,       // Modülü içeren sürecin yürütücüsü
  HMODULE hModule,       // Modül yürütücüsü
  LPTSTR lpBaseName,     // Tampon alanının işaretçisi
  DWORD nSize            // Tampon alanının büyüklüğü (bayt)
);
```

lpBaseName: Bu parametre modülün esas adının tutulacağı tampon alanının işaretçisi değerini alır. Eğer modülün esas ismi, nSize parametresi ile belirtilen tampon alanının alabileceği karakter sayısından daha fazla ise modülün esas ismi kısaltılır.

GetModuleInformation: Belirlenen modülün MODULINFO yapısında tutulan bilgisini getirmek için kullanılan fonksiyondur.

```
BOOL GetModuleInformation(
  HANDLE hProcess,              // Modülü içeren sürecin yürütücüsü
  HMODULE hModule,              // Modülün yürütücüsü
  LPMODULEINFO lpmodinfo,       // MODULINFO yapısının işaretçisi
  DWORD cb                      // MODULINFO yapısının büyüklüğü (bayt)
);
```

MODULINFO yapısı modülün yükleme adresini, büyüklüğünü ve giriş noktasını içerir.

```
typedef struct _MODULEINFO {
  LPVOID lpBaseOfDll;   // Modülün yükleme adresi
  DWORD SizeOfImage;    // Modülün tutulduğu alanın büyüklüğü (bayt)
  LPVOID EntryPoint;    // Modülün giriş noktası
}
MODULEINFO, *LPMODULEINFO;
```

GetModuleFileNameEx: Modülün tam yol ismini veren fonksiyondur.

```
DWORD GetModuleFileNameEx(
  HANDLE hProcess,      // Modülü içeren sürecin yürütücüsü
  HMODULE hModule,      // Modülün yürütücüsü
  LPTSTR lpFilename,    // Tampon işaretçisi
  DWORD nSize           // Tamponun büyüklüğü (bayt)
);
```

lpFilename: Bu parametre modülün tam yol isminin alınacağı tampon alanının işaretçi değerini alır. Modülün ismi nSize parametresi ile belirtilen değerden daha büyükse, modül ismi kısaltılır ve NULL değeri ile sonlandırılır.

GetProcessImageFileName: Belirlenen sürecin çalıştırılabilir dosyasının adını getiren fonksiyondur.

```
DWORD GetProcessImageFileName(
  HANDLE hProcess,              // Süreç yürütücüsü
  LPTSTR lpImageFileName,       // Tampon işaretçisi
  DWORD nSize                   // Tamponun büyüklüğü (bayt)
);
```

lpImageFileName: Bu parametre çalıştırılabilir dosyanın tam yolunun tutulduğu tampon alanının işaretçisini alır.

VDMEnumTaskWOWEx: Windows NT/2000/XP işletim sistemlerinde 16 bit uygulamaları listelemek için kullanılan fonksiyondur.

```
INT WINAPI VDMEnumTaskWOWEx(
      DWORD             dwProcessId,
      TASKENUMPROCEX    fp,
      LPARAM            lp
);
```

dwProcessId: 16-bit sürecin tanımlayıcısı.

fp: Fonksiyonun çağrılış işaretçisi.

lp: Kullanıcının tanımlayacağı fonksiyona geçirilecek listeleme fonksiyonu.

Ek 3. İleti Sistemi Fonksiyonları

CloseHandle: Belirtilen dosyanın yürütücüsünü kapatır. CreateToolhelp32Snapshot fonksiyonu tamamlandıktan sonra, bu fonksiyon ile kapatılmalıdır.

```
BOOL CloseHandle(HANDLE hObject)
```

CreateRemoteThread(): Başka bir sürecin adres alanında çalışan bir iplik oluşturur.

```
HANDLE CreateRemoteThread(
      HANDLE hProcess, LPSECURITY_ATTRIBUTES lpThreadAttributes,
      DWORD dwStackSize, LPİPLİK_START_ROUTINE lpStartAddress,
      LPVOID lpParameter, DWORD dwCreationFlags, LPDWORD lpThreadId
);
```

hProcess: İpliğin içinde oluşturulacağı sürecin yürütücü değeri.

lpThreadAttributes: İpliğin güvenlik özelliklerinin işaretçisi.

dwStackSize: Bayt olarak ipliğin başlangıç yığıt boyutu.

lpStartAddress: İplik fonksiyonunun işaretçisinin başlangıç adresi.

lpParameter: Yeni iplik için gerekli işaretçi.

dwCreationFlags: Oluşturma bayrakları.

lpThreadId: Döndürülen iplik tanımlayıcısının işaretçisi.

DefWindowProc(): Bu fonksiyon, uygulamanın işlemediği bazı pencere iletilerinin işlenmesini sağlamak için pencere yordamını çağırır. Böylece her iletini işlenmesi garantilenir.

```
LRESULT DefWindowProc(
    HWND hWnd,          // pencere yürütücüsü
    UINT Msg,           // ileti tanımlayıcısı
    WPARAM wParam,      // ilk ileti parametresi
    LPARAM lParam       // ikinci ileti parametresi
);
```

DisPatchMessage(): Bir iletiyi bir pencere yordamına gönderir. Genellikle GetMessage() fonksiyonu ile elde dilen bir iletiyi göndermek için kullanılır.

```
LONG DispatchMessage(
    CONST MSG *lpmsg    // MSG yapısının işaretçisi
);
```

MSG yapısı[7] bir iletiyle ilgili bilgileri içerir.

GetMessage(): Çağrılan ipliğin ileti kuyruğundan bir ileti alır ve bu iletiyi MSG yapısına yerleştirir.

```
BOOL GetMessage(
    LPMSG lpMsg,        // MSG yapısının adresi
    HWND hWnd,          // Pencerenin yürütücüsü
    UINT wMsgFilterMin, // ilk ileti
    UINT wMsgFilterMax  // son ileti
);
```

GetQueueStatus(): Çağrılan ipliğin ileti kuyruğunda bulunan iletilerin, tipini gösteren bayraklar döndürür.

```
DWORD GetQueueStatus(
    UINT flags
);
```

flags: İleti kuyruğundan alınan iletinin tipini belirler. Bu parametre aşağıdaki değerleri alabilir.

QS_ALLEVENTS: Bir giriş, WM_TIMER, WM_PAINT, WM_HOTKEY veya gönderilen ileti kuyruktadır.

QS_ALLINPUT: Herhangi bir ileti kuyruktadır.

QS_HOTKEY: Bir WM_HOTKEY iletisi kuyruktadır.

QS_INPUT: Bir giriş iletisi kuyruktadır.

QS_KEY: Bir WM_KEYUP, WM_KEYDOWN, WM_SYSKEYUP veya WM_SYSKEYDOWN iletisi kuyruktadır.

QS_MOUSE: Bir fare iletisi kuyruktadır. (WM_RBUTTONDBLCLK WM_MOUSEMOVE, WM_LBUTTONUP vb gibi)

QS_MOUSEBUTTON: Bir fare düğmesi iletisi kuyruktadır. (WM_LBUTTONUP, WM_RBUTTONDOWN vb gibi)

---

[7] MSG yapısı için bkz. EK E- Kanca tipleri, WH_GETMESSAGE kancası, lParam parametresi

QS_MOUSEMOVE: Bir WM_MOUSEMOVE iletisi kuyruktadır.

QS_PAINT: Bir WM_PAINT iletisi kuyruktadır.

QS_POSTMESSAGE: Listelenenler haricinde gönderilen bir ileti kuyruktadır.

QS_SENDMESSAGE: Başka bir iplik veya uygulamadan gönderilen ileti kuyruktadır.

QS_TIMER: Bir WM_TIMER iletisi kuyruktadır.

OpenProcess(): Belirlenen sürecin açılması için bir yürütücü değeri döndürür.

```
HANDLE OpenProcess(
    DWORD dwDesiredAccess,    // erişim bayrağı
    BOOL bInheritHandle,      // yürütücü bayrağı
    DWORD dwProcessId         // süreç tanımlayıcısı
   );
```

dwDesiredAccess: Sürece erişimle ilgili güvenlik ayarlamalarını belirler. Bu parametre aşağıdaki değerleri alabilir.

PROCESS_ALL_ACCESS: Süreç bütün erişimlere izin verir.

PROCESS_CREATE_PROCESS: Dahili kullanıldı.

PROCESS_CREATE_THREAD: CreateRemoteThread fonksiyonundaki süreç yürütücüsüne süreç de bir iplik oluşturma izni verir.

PROCESS_DUP_HANDLE: Kaynak veya hedef sürecin, süreç yürütücüsünü çiftler.

PROCESS_QUERY_INFORMATION: GetExitCodeProcess ve GetPriorityClass fonksiyonlarının süreç yürütücüsünü kullanarak süreç bilgisini okumasını sağlar.

PROCESS_SET_INFORMATION: SetPriorityClass fonksiyonunun süreç yürütücüsünü kullanarak, sürecin öncelikli sınıfını ayarlamasını sağlar.

PROCESS_TERMINATE: TerminateProcess fonksiyonunun süreç yürütücüsünü kullanarak, süreci sonlandırmasını sağlar.

PROCESS_VM_OPERATION: VirtualProtectEx ve WriteProcessMemory fonksiyonlarının süreç yürütücüsünü kullanarak, sürecin sanal bellek alanını değiştirmesini sağlar.

PROCESS_VM_READ: ReadProcessMemory fonksiyonunun süreç yürütücüsünü kullanarak, sürecin sanal bellekten okunmasını sağlar.

PROCESS_VM_WRITE: WriteProcessMemory fonksiyonunun süreç yürütücüsünü kullanarak, sürecin sanal belleğe yazdırılmasını sağlar.
SYNCHRONIZE (Sadece Windows NT'de): Herhangi bir bekleme fonksiyonunun süreç yürütücüsünü kullanarak, süreci sonlandırmak için beklemesini sağlar.

PostMessage(): Bir pencere tarafından oluşturulan iplik ile ilgili bir iletiyi, ileti kuyruğundan gönderir. İplik iletisinin işlenmesini beklemez. İleti kuyruğundaki ileti, GetMessage() veya PeekMessage() fonksiyonları çağrılarak elde edilir.

```
BOOL PostMessage(
    HWND hWnd,          // hedef pencerenin yürütücüsü
    UINT Msg,           // gönderilen ileti
    WPARAM wParam,      // ilk ileti parametresi
    LPARAM lParam       // ikinci ileti parametresi
);
```

hWnd: İletisi alacak pencere yordamını tanımlar. Bu parametre için kullanılan aşağıdaki iki değerin özel anlamları vardır.
HWND_BROADCAST: İleti sistemdeki bütün üst seviye pencerelere gönderilir. Alt pencerelere gönderilmez.
NULL: Fonksiyon hali hazırdaki ipliğin tanımlayıcısı olarak davranır.

PeekMessage(): Bir ileti için iplik ileti kuyruğunu kontrol eder ve eğer herhangi bir ileti varsa, bu iletiyi belirlenmiş yapıya yerleştirir.

```
BOOL PeekMessage(
    LPMSG lpMsg,        // ileti yapısının işaretçisi
    HWND hWnd,          // pencere yürütücüsü
    UINT wMsgFilterMin, // ilk ileti
    UINT wMsgFilterMax, // son ileti
    UINT wRemoveMsg     // kaldırma bayrakları
);
```

wRemoveMsg: İletinin nasıl yürütüleceğini belirler. Bu parametre aşağıdaki değerlerden birini alabilir.
PM_NOREMOVE: İleti PeekMessage() fonksiyonu ile işlendikten sonra kuyruktan çıkartılmaz..

PM_REMOVE: İleti PeekMessage() fonksiyonu ile işlendikten sonra kuyruktan çıkartılır.

SendMessage(): Belirlenmiş bir iletiyi bir pencereye veya pencerelere gönderir. Fonksiyon belirlenmiş pencere için pencere yordamını çağırır. Pencere yordamı iletiyi işleyene kadar fonksiyon sonlanmaz. Buna zıt olarak PostMessage() fonksiyonu ise bir ipliğin ileti kuyruğundaki bir iletiyi gönderir ve hemen sonlanır.

```
LRESULT SendMessage(
    HWND hWnd,          // hedef pencerenin yürütücüsü
    UINT Msg,           // gönderilen ileti
    WPARAM wParam,      // ilk ileti parametresi
    LPARAM lParam       // ikinci ileti parametresi
   );
```

hWnd: İletiyi alacak pencerenin, pencere yordamını belirler. Parametre değeri HWND_BROADCAST olursa ileti bütün üst seviye pencerelere gönderilir, fakat alt pencerelere gönderilmez.

Ek 4. Giriş İletileri

Fare İletileri

İleti kuyruğuna bir fare iletisi şu parametrelerle birlikte eklenir.

fwKeys: Fare tuşlarına birlikte basılmış tuş olup olmadığını kontrol eden parametredir. Aşağıdaki değerleri alabilir:

MK_CONTROL: CTRL tuşu basılıdır.

MK_LBUTTON: Farenin sol tuş basılıdır.

MK_MBUTTON: Farenin orta tuş basılıdır.

MK_RBUTTON: Farenin sağ tuş basılıdır.

MK_SHIFT: SHIFT tuşu basılıdır.

xPos: İmlecin x düzlemindeki koordinatıdır.

yPos: İmlecin y düzlemindeki koordinatıdır.

Windows işletim sistemlerinde aşağıdaki fare iletileri kullanılır.

WM_LBUTTONDBLCLK: Bir pencerede yer alan bir eleman üzerindeyken kullanıcı farenin sol tuşuna çift tıkladığında, ileti kuyruğuna gönderilir.

WM_LBUTTONDOWN: Bir pencerede yer alan bir eleman üzerindeyken kullanıcı farenin sol tuşuna tıkladığında, ileti kuyruğuna gönderilir.

WM_LBUTTONUP: Bir pencerede yer alan bir eleman üzerindeyken kullanıcı farenin sol tuşunu serbest bıraktığında, ileti kuyruğuna gönderilir.

WM_MBUTTONDBLCLK: Bir pencerede yer alan bir eleman üzerindeyken kullanıcı farenin orta tuşuna çift tıkladığında, ileti kuyruğuna gönderilir.

WM_MBUTTONDOWN: Bir pencerede yer alan bir eleman üzerindeyken kullanıcı farenin orta tuşuna tıkladığında, ileti kuyruğuna gönderilir.

WM_MBUTTONUP: Bir pencerede yer alan bir eleman üzerindeyken kullanıcı farenin orta tuşunu serbest bıraktığında, ileti kuyruğuna gönderilir.

WM_MOUSEACTIVATE: Kullanıcı bir pencereyi aktif yapmak için fare tuşlarına tıkladığında, ileti kuyruğuna gönderilir.

WM_MOUSEMOVE: İmleç hareket ettirildiğinde, ileti kuyruğuna gönderilir.
WM_NCLBUTTONDBLCLK: Pencerede boş bir alandayken kullanıcı farenin sol tuşuna çift tıkladığında, ileti kuyruğuna gönderilir.
WM_NCLBUTTONDOWN: Pencerede boş bir alandayken kullanıcı farenin sol tuşuna tıkladığında, ileti kuyruğuna gönderilir.
WM_NCLBUTTONUP: Pencerede boş bir alandayken kullanıcı farenin sol tuşunu serbest bıraktığında, ileti kuyruğuna gönderilir.
WM_NCMBUTTONDBLCLK: Pencerede boş bir alandayken kullanıcı farenin orta tuşuna çift tıkladığında, ileti kuyruğuna gönderilir.
WM_NCMBUTTONDOWN: Pencerede boş bir alandayken kullanıcı farenin orta tuşuna tıkladığında, ileti kuyruğuna gönderilir.
WM_NCMBUTTONUP: Pencerede boş bir alandayken kullanıcı farenin orta tuşunu serbest bıraktığında, ileti kuyruğuna gönderilir.
WM_NCMOUSEMOVE: İmleç pencerede boş bir alanda hareket ettirildiğinde, ileti kuyruğuna gönderilir.
WM_NCRBUTTONDBLCLK: Pencerede boş bir alandayken kullanıcı farenin sağ tuşuna çift tıkladığında, ileti kuyruğuna gönderilir.
WM_NCRBUTTONDOWN: Pencerede boş bir alandayken kullanıcı farenin sağ tuşuna tıkladığında, ileti kuyruğuna gönderilir.
WM_NCRBUTTONUP: Pencerede boş bir alandayken kullanıcı farenin sağ tuşunu serbest bıraktığında, ileti kuyruğuna gönderilir.
WM_RBUTTONDBLCLK: Bir pencere alanında, kullanıcı farenin sağ tuşuna çift tıkladığında, ileti kuyruğuna gönderilir.
WM_RBUTTONDOWN: Bir pencere alanında, kullanıcı farenin sağ tuşuna tıkladığında, ileti kuyruğuna gönderilir.
WM_RBUTTONUP: Bir pencere alanında, kullanıcı farenin sağ tuşunu serbest bıraktığında, ileti kuyruğuna gönderilir.

Klavye İletileri

İleti kuyruğuna bir klavye iletisi şu parametrelerle birlikte eklenir.

nVirtKey: Sistem dışı tuşlar için bir sanal tuş kodu belirler.

lKeyData: Tuş verisinin belirler. Aşağıdaki değerleri alabilir.
0-15: Tekrar sayısını belirler.
16-23: Tarama kodunu belirler.
24: Tuşun bir fonksiyon tuşu olup olmadığını belirler.
25-28: Rezerve, kullanılmaz.
29: ALT tuşunu kontrol eder.
30: Bir önceki tuş durumunu belirler.
31: Geçiş durumunu belirler.

Windows işletim sistemleri aşağıdaki klavye iletilerini kullanırlar.

WM_IME_KEYDOWN: Bir uygulamada bir tuşa basıldığını, IME (Input Method Editor, Giriş Metot Editörü) kullanarak uygulamaya gönderen iletidir. Genellikle bu ileti, ileti sırasını korumak için IME tarafından üretilir.
WM_IME_KEYUP: Bir uygulamada bir tuşun serbest bırakıldığını IME (Input Method Editor, Giriş Metot Editörü) kullanarak uygulamaya gönderen iletidir. Genellikle bu ileti, ileti sırasını korumak için IME tarafından üretilir.
WM_KEYDOWN: Sistem dışı bir tuşa basıldığında, ileti kuyruğuna eklenir. ALT tuşu ile birlikte basılmayan her türlü tuş veya tuş kombinasyonları sistem dışı tuşu olarak adlandırılır.
WM_KEYUP: Sistem dışı bir tuş serbest bırakıldığında, ileti kuyruğuna eklenir.
WM_SYSKEYDOWN: ALT tuşu ile birlikte herhangi bir klavye tuşuna basıldığında ileti kuyruğuna gönderilir.
WM_SYSKEYUP: ALT tuşu basılıyken diğer tuş serbest bırakıldığında ileti kuyruğuna gönderilir.

Ek 5. Kanca Tipleri

WH_CALLWNDPROC: SendMessage() fonksiyonu ile gönderilen iletileri görüntülemek için kullanılır. CallWndProc() fonksiyonunu kullanır. İleti hedef pencerenin yordamına geçirilmeden önce sistem iletiyi kanca yordamına geçirir. Kanca yordamı iletiyi inceleyebilir fakat değiştiremez.

```
LRESULT CALLBACK CallWndProc(
    int nCode,     // kanca kodu
    WPARAM wParam, // hali hazırdaki süreç bayrağı
    LPARAM lParam  // CWPSTRUCT yapısının adresi
);
```

CWPSTRUCT, WH_CALLWNDPROC kancasına geçirilecek ileti parametrelerinin tutulduğu yapıdır.

```
typedef struct tagCWPSTRUCT {
    LPARAM  lParam;
    WPARAM  wParam;
    UINT    message;
    HWND    hwnd;
} CWPSTRUCT;
```

WH_CBT: Windows bir CBT (computer-based training, bilgisayar tabanlı çevirme) kanca yordamını şu durumlardan önce çağırır.

- Bir pencereyi aktif hale getirmek, oluşturmak, simge durumuna getirmek, ekranı kaplatmak, taşımak, boyutlandırmak.
- Bir sistem komutunu tamamlamak.
- Sistem ileti kuyruğundan bir fare veya klavye olayını çıkarmak.
- Girişi ayarlamak.
- Sistem ileti kuyruğunun senkronizasyonunu sağlamak.

CBTProc fonksiyonunu kullanır.

```
LRESULT CALLBACK CBTProc(
    int nCode,          // kanca kodu
    WPARAM wParam,      // kanca koduna bağlıdır
```

```
    LPARAM lParam        // kanca koduna bağlıdır
   );
```

nCode: Kanca yordamının kullandığı bu kod iletinin nasıl işleneceğini belirler. Aşağıdaki değerleri alabilir.

HCBT_ACTIVATE: Sistem bir pencereyi aktif yapacaktır.

HCBT_CLICKSKIPPED: Sistem, sistem ileti kuyruğundan bir fare iletisini çıkarmıştır.

HCBT_CREATEWND: Bir pencere oluşturulacaktır.

HCBT_DESTROYWND: Bir pencere kapatılacaktır.

HCBT_KEYSKIPPED: Sistem, sistem ileti kuyruğundan bir klavye iletisi çıkarmıştır.

HCBT_MINMAX: Bir pencere simge durumuna getirilecek veya ekranı kaplayacaktır.

HCBT_MOVESIZE Bir pencere taşınacak veya boyutlandırılacaktır.

HCBT_QS: Sistem, sistem ileti kuyruğundan bir WM_QUEUESYNC iletisi almıştır.

HCBT_SETFOCUS: Bir pencere klavye iletisi alacak.

HCBT_SYSCOMMAND: Bir sistem komutu uygulanacak.

WH_DEBUG: Windows bu kanca yordamını, sistemdeki başka kancalarla ilgili kanca yordamları çağrılmadan önce çağırır. Diğer kancalarla ilgili kanca yordamlarının çağrılmasına, sistemin izin verip vermediğini belirlemek için de kullanılabilir. DebugProc fonksiyonunu kullanır.

```
LRESULT CALLBACK DebugProc(
    int nCode,           // kanca kodu
    WPARAM wParam,       // çağrılacak kancanın tipi
    LPARAM lParam        // DEBUGHOOKINFO yapısının adresi
   );
```

wParam: Çağrılacak kancanın tipi aşağıdaki değerler olabilir:

WH_CALLWNDPROC, WH_GETMESSAGE, WH_DEBUG, WH_KEYBOARD, WH_MSGFILTER, WH_JOURNALPLAYBACK, WH_JOURNALRECORD, WH_MOUSE, WH_SHELL, WH_CBT, WH_SYSMSGFILTER

DEBUGHOOKINFO yapısı hata ayıklama bilgisi içerir.

```
typedef struct tagDEBUGHOOKINFO {
    DWORD  idThread;
    DWORD  idThreadInstaller;
    LPARAM lParam;
    WPARAM wParam;
    int    code;
} DEBUGHOOKINFO;
```

WH_FOREGROUNDIDLE: Windows bu kanca yordamını, uygulamanın arka planda çalışan ipliği boşa çıkarılacağı zaman çağırır. ForegroundIdleProc() fonksiyonunu kullanır.

```
DWORD ForegroundIdleProc(
    int code,      // kanca kodu
    DWORD wParam, // kullanılmaz
    LONG lParam    // kullanılmaz
);
```

WH_GETMESSAGE: Bir uygulamanın, GetMessage() veya PeekMessage() fonksiyonlarından döndürülen iletileri izlemesini sağlayan kanca yordamıdır. WH_GETMESSAGE kancası, ileti kuyruğuna gönderilen klavye ve fare girişlerinin veya diğer iletilerin takip edilmesi için de kullanılabilir. GetMsgProc() fonksiyonunu kullanır.

```
LRESULT CALLBACK GetMsgProc(
    int code,           // kanca kodu
    WPARAM wParam,      // kaldırma bayrağı
    LPARAM lParam       // iletinin (veya MSG yapısının) adresi
);
```

wParam: İletinin, kuyruktan kaldırılıp kaldırılmadığını belirler. Bu parametre aşağıdaki değerleri alabilir:

PM_NOREMOVE: İletinin kuyruktan henüz çıkarılmadığını gösterir.

PM_REMOVE: İletinin kuyruktan çıkarıldığını gösterir.

lparam: İletinin tutulduğu yapıyı adresler. MSG, ileti kuyruğundaki iletinin bilgisini içeren yapıdır.

```
typedef struct tagMSG {
    HWND   hwnd;
    UINT   message;
    WPARAM wParam;
    LPARAM lParam;
    DWORD  time;
    POINT  pt;
} MSG;
```

WH_JOURNALPLAYBACK: Bir uygulamanın, sistem ileti kuyruğuna ileti eklemesini sağlayan kanca yordamıdır. Bu kanca bir gecikme zaman değeri döndürür. Bu değer sisteme hali hazırdaki iletinin işlenmeden önce kaç milisaniye beklemesi gerektiğini söyler. JournalPlaybackProc() fonksiyonunu kullanır. Bu fonksiyon sistem ileti kuyruğuna fare ve klavye iletilerini ekler.

```
LRESULT CALLBACK JournalPlaybackProc(
    int code,           // kanca kodu
    WPARAM wParam,      // tanımlanmamış
    LPARAM lParam       // işlenen iletinin (EVENTMSG) adresi
);
```

code: İletinin nasıl işleneceğini belirleyen parametredir. Aşağıdaki değerleri alabilir.
HC_GETNEXT: Kanca yordamı hali hazırdaki fare veya klavye iletisini lParam parametresi ile işaret edilen EVENTMSG yapısına kopyalamalıdır.
HC_NOREMOVE: Bir uygulama PeekMessage() fonksiyonunu wRemoveMsg parametresi PM_NOREMOVE olarak çağırdığında PeekMessage() fonksiyonunun işlenmesinden sonra ileti, ileti kuyruğundan çıkarılmaz.
HC_SKIP: Kanca yordamı lParam ile işaret edilen EVENTMSG yapısına kopyalayacağı bir sonraki fare veya klavye iletisini hazırlamak zorundadır. HC_GETNEXT kodu alınması ile kanca yordamı iletiyi yapıya kopyalamalıdır.
HC_SYSMODALOFF: Bir sistem diyalog kutusu kapatılır. Kanca yordamı iletileri eklemeye devam etmelidir.
HC_SYSMODALONA: Bir sistem diyalog kutusu görüntüleniyordur. Diyalog kutusu kapatılana kadar kanca yordamı iletileri eklemeyi durdurur.

EVENTMSG, sistem ileti kuyruğuna gönderilen bir donanım iletisinin bilgisini içerir.

```
typedef struct tagEVENTMSG {
    UINT  message;
    UINT  paramL;
    UINT  paramH;
    DWORD time;
    HWND  hwnd;
} EVENTMSG;
```

WH_JOURNALRECORD: Giriş olaylarını kaydetmek ve izlemek için kullanılan kanca tipidir. Genellikle klavye ve fare olaylarını kaydederek WH_JOURNALPLAYBACK kancasının bu olayları tekrar ileti kuyruğuna eklemesi için kullanılır. JournalRecordProc() fonksiyonunu kullanır. Bu fonksiyon, sistem ileti kuyruğundan çıkartılan iletileri kaydeder.

```
LRESULT CALLBACK JournalRecordProc(
    int code,           // kanca kodu
    WPARAM wParam,      // tanımlanmamış
    LPARAM lParam       // işlenecek iletinin adresi
);
```

code: İletinin nasıl işleneceğini belirler. Bu parametre aşağıdaki değerleri alabilir.
HC_ACTION: lParam parametresi, sistem ileti kuyruğundan çıkarılan iletinin tutulduğu EVENTMSG yapısını işaret eder. Kanca yordamı bu yapı içindeki verileri kopyalayarak bir tampon alanına veya bir dosyaya kaydetmelidir.
HC_SYSMODALOFF: Bir sistem diyalog kutusu kapatılır. Kanca yordamı iletileri kaydetmeyi sürdürmelidir.
HC_SYSMODALONA: Bir sistem diyalog kutusu görüntüleniyordur. Diyalog kutusu kapatılana kadar kanca yordamı iletileri kaydetmeyi durdurur.

WH_KEYBOARD: İleti kuyruğuna gönderilen klavye giriş iletilerinin izlenmesini sağlar. KeyboardProc() fonksiyonunu kullanır. Bu fonksiyon bir uygulama GetMessage() ve PeekMessage() fonksiyonlarını çağırdığında ve WM_KEYUP veya WM_KEYDOWN gibi bir klavye iletisi işlendiğinde sistem tarafından çağrılır.

```
LRESULT CALLBACK KeyboardProc(
    int code,           // kanca kodu
    WPARAM wParam,      // sanal tuş kodu
    LPARAM lParam       // basılı tuşun ileti bilgisi
);
```

code: Kanca yordamının iletiyi nasıl işleyeceğini belirleyen parametredir. Bu parametre aşağıdaki değerleri alabilir:
HC_ACTION: wParam ve lParam parametreleri bir tuşa basıldığına dair ileti bilgisi içerir.
HC_NOREMOVE: wParam ve lParam parametreleri bir tuşa basıldığına dair ileti bilgisi içerir ve bu ileti henüz ileti kuyruğundan çıkarılmamıştır.

wParam: İleti, basılan tuşa dair bir sanal tuş kodu üretir.

WH_MOUSE: GetMessage() ve PeekMessage fonksiyonlarından dönen fare iletilerini takip eden kanca yordamıdır. Bu kancayı kullanarak ileti kuyruğuna gönderilen fare girişleri izlenebilir. MouseProc() fonksiyonunu kullanır. Bu fonksiyon, bir uygulama GetMessage() ve PeekMessage() fonksiyonlarını çağırdığında ve işlenecek bir fare iletisi olduğunda sistem tarafından çağrılır.

```
LRESULT CALLBACK MouseProc(
    int nCode,          // kanca kodu
    WPARAM wParam,      // ileti tanımlayıcısı
    LPARAM lParam       // fare koordinatları
);
```

code: Kanca yordamının iletiyi nasıl işleyeceğini belirleyen parametredir. Bu parametre aşağıdaki değerleri alabilir:
HC_ACTION: wParam ve lParam parametreleri bir fare iletisi hakkında bilgi içerir.
HC_NOREMOVE: wParam ve lParam parametreleri bir fare iletisi hakkında bilgi içerir ve bu fare iletisi henüz ileti kuyruğundan çıkarılmamıştır.

lParam: MOUSEHOOKSTRUCT yapısını işaret eder. Bu yapı bir fare olayı hakkında bilgi içerir.

```
typedef struct tagMOUSEHOOKSTRUCT {
    POINT pt;           // imlecin x ve y koordinatları
    HWND  hwnd;         // imlecin üzerinde bulunduğu pencere
    UINT  wHitTestCode; // tıklama test kodları
    DWORD dwExtraInfo;  // ileti ile ilgili ekstra bilgi
} MOUSEHOOKSTRUCT;
```

wHitTestCode parametresinin alabileceği değerler şunlardır:

HTBORDER: Pencerenin kenarlığında.

HTBOTTOM: Yatay an alt kenarlıkta.

HTBOTTOMLEFT: Sol alt köşede.

HTBOTTOMRIGHT: Sağ alt köşede.

HTCAPTION : Başlık alanında.

HTCLIENT: Pencere içindeki bir elemanda.

HTERROR: Pencerede boş bir alanda.

HTGROWBOX: Büyültme kutusunda.

HTHSCROLL: Yatay ilerletme çubuğunda.

HTLEFT: Sol kenarlıkta.

HTMENU: Menüde.

HTNOWHERE: Arka planda veya pencereleri ayıran çizgide.

HTREDUCE. Simge durumuna küçültme düğmesinde.

HTRIGHT: Sağ kenarlıkta.

HTSIZE: Büyültme kutusunda (HTGROWBOX ile benzer).

HTSYSMENU: Bir sistem menüsünde veya alt pencerenin kapatma düğmesinde.

HTTOP: Üst yatay kenarlıkta.

HTTOPLEFT: Üst sol köşede.

HTTOPRIGHT: Üst sağ köşede.

HTTRANSPARENT: Başka bir pencereyi kaplayan pencerede.

HTVSCROLL: Dikey ilerletme çubuğunda.

HTZOOM: Ekranı kaplatma düğmesinde.

WH_ MSGFILTER ve WH_ SYSMSGFILTER: Bu kanca yordamları bir menü, ilerletme çubuğu, ileti kutusu veya diyalog kutusu tarafından işlenecek iletileri takip

etmek ve kullanıcının başka bir pencereyi aktif yapmak için ALT+TAB veya ALT+ESC tuş kombinasyonlarını kullandığını denetlemek için kullanılırlar.

WH_MSGFILTER kancası sadece kanca yordamına yüklenen uygulama tarafından oluşturulan menü, ilerletme çubuğu, ileti kutusu veya diyalog kutusuna geçirilen iletilerin izlenmesi için kullanabilir. MessageProc() fonksiyonunu kullanır. Bu fonksiyon bir menü, ilerletme çubuğu, ileti kutusu veya diyalog kutusunda bir giriş olayı gerçekleştikten sonra, giriş olayı ile ilgili ileti üretilmeden önce sistem tarafından çağrılır. Bir kısım uygulamalar veya bütün uygulamalar tarafından oluşturulan menü, ilerletme çubuğu, ileti kutusu veya diyalog kutusu iletilerini takip edebilir.

```
LRESULT CALLBACK MessageProc(
    int code,           // kanca kodu
    WPARAM wParam,      // tanımlanmamış
    LPARAM lParam       // ileti verili yapının adresi
);
```

code: İleti tarafından üretilen giriş olayının tipini belirler. Bu parametre aşağıdaki değerleri alabilir.

MSGF_DDEMGR: Dinamik Veri Değiştirme Yönetim Kütüphanesi (DDEML-Dynamic Data Exchange Management Library) bir senkronize geçişin bitmesini beklerken oluşan giriş olayıdır.

MSGF_DIALOGBOX: İleti kutusu veya diyalog kutusunda oluşan giriş olayıdır.

MSGF_MENU: Menüde oluşan giriş olayıdır.

MSGF_NEXTWINDOW: Kullanıcının ALT+TAB tuşlarına basması ile oluşan giriş olayıdır.

MSGF_SCROLLBAR: İlerletme çubuğunda oluşan giriş olayıdır.

lParam: İleti bilgisini tutan MSG yapısını işaret eder.

WH_SYSMSGFILTER kancası bütün uygulama iletilerini takip eder. SysMsgProc() fonksiyonunu kullanır.

```
LRESULT CALLBACK SysMsgProc(
    int nCode,          // ileti bayrağı
    WPARAM wParam,      // tanımlanmamış
    LPARAM lParam       // ileti verili yapının adresi
);
```

nCode: İleti tarafından üretilen giriş olayının tipini belirler. Bu parametre aşağıdaki değerleri alabilir: [8]

MSGF_DIALOGBOX, MSGF_SCROLLBAR, MSGF_NEXTWINDOW, MSGF_MENU

lParam: İleti bilgisini tutan MSG yapısını işaret eder.

WH_ SHELL: Windows bu kanca yordamını bir shell uygulaması aktif olduğunda ve bir üst seviye pencere kapatıldığında veya oluşturulduğunda çağırır. ShellProc() fonksiyonunu kullanır. Bir shell uygulaması bu fonksiyonu sistemden yararlı notifikasyonlar almak için kullanır.

```
LRESULT CALLBACK ShellProc(
    int nCode,          // kanca kodu
    WPARAM wParam,      // belirtilen olay bilgisi
    LPARAM lParam       // belirtilen olay bilgisi
);
```

nCode: Kanca kodunu belirler. Bu parametre aşağıdaki değerleri alabilir.

HSHELL_ACTIVATESHELLWINDOW: Shell kendi ana penceresini aktif yapmalıdır.

HSHELL_GETMINRECT: Bir pencere küçültülür veya ekranı kaplatılır. Sistem küçültülen pencerenin koordinatlarını bilmelidir. wParam parametresi pencerenin yürütücü değerini, lParam parametresi koordinatların tutulduğu RECT yapısının adres bilgisini içerir. (Windows NT'de yok)

HSHELL_LANGUAGE: Klavye dili değiştirilir veya yeni bir klavye dili yüklenir. (Windows NT'de yok)

HSHELL_REDRAW: Görev çubuğundaki pencere başlığı tekrar resmedilir. wParam parametresi pencerenin yürütücü değerini içerir. (Windows NT'de yok)

---

[8] Açıklamaları için bkz. MessageProc fonksiyonu code parametre değerleri

HSHELL_TASKMAN: Kullanıcı görev listesini seçmiştir. wParam parametresi tanımlanmamış ve yok sayılmalıdır. (Windows NT'de yok)
HSHELL_WINDOWACTIVATED: Aktivite farklı bir üst düzey sahipsiz pencereye yönlendirildi. wParam parametresi pencere yürütücü değerini içerir. (Windows NT'de yok)
HSHELL_WINDOWCREATED: Bir üst düzey sahipsiz pencere oluşturuldu. Sistem ShellProc fonksiyonunu çağırdığında pencere ortaya çıkar.
HSHELL_WINDOWDESTROYED: Bir üst düzey sahipsiz pencere kapatılacaktır. Sistem ShellProc fonksiyonunu çağırdığında pencere hâlâ meydandadır.

wParam: Kanca kodlarına göre farklı değerler alır. Örneğin nCode parametresi HSHELL_ACTIVATESHELLWINDOW ise wParam kullanılmaz.

lParam: Windows sistemlerinde nCode parametresine göre farklı değerler alır. Örneğin eğer nCode HSHELL_GETMINRECT ise lParam LPRECT; nCode HSHELL_WINDOWSACTIVATED ise lParam fFullScreen; nCode HSHELL_REDRAW ise lParam fNewFlash olacaktır..
Windows NT de bu parametre değeri 0 (sıfır) olmalıdır.

ÖZGEÇMİŞ

İsmail KURNAZ, 1978 yılında Ankara'da doğdu. İlk ve orta öğrenimini tamamladıktan sonra 1997 yılında Gazi Üniversitesi Teknik Eğitim Fakültesi Bilgisayar Sistemleri Öğretmenliği bölümüne girdi. 2001 bahar döneminde bu bölümden mezun olduktan sonra MEB'na bağlı okullarda bilgisayar öğretmeni olarak görev yapmaya başladı. 2002 yılının II. yarıyılında Gazi Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Eğitimi Anabilim Dalı'nda tezli-yüksek lisans programına girdi. Halen Yenimahalle Meslek ve Anadolu Meslek Lisesi'nde bilgisayar öğretmeni olarak görev yapmaktadır. İngilizce bilmektedir.