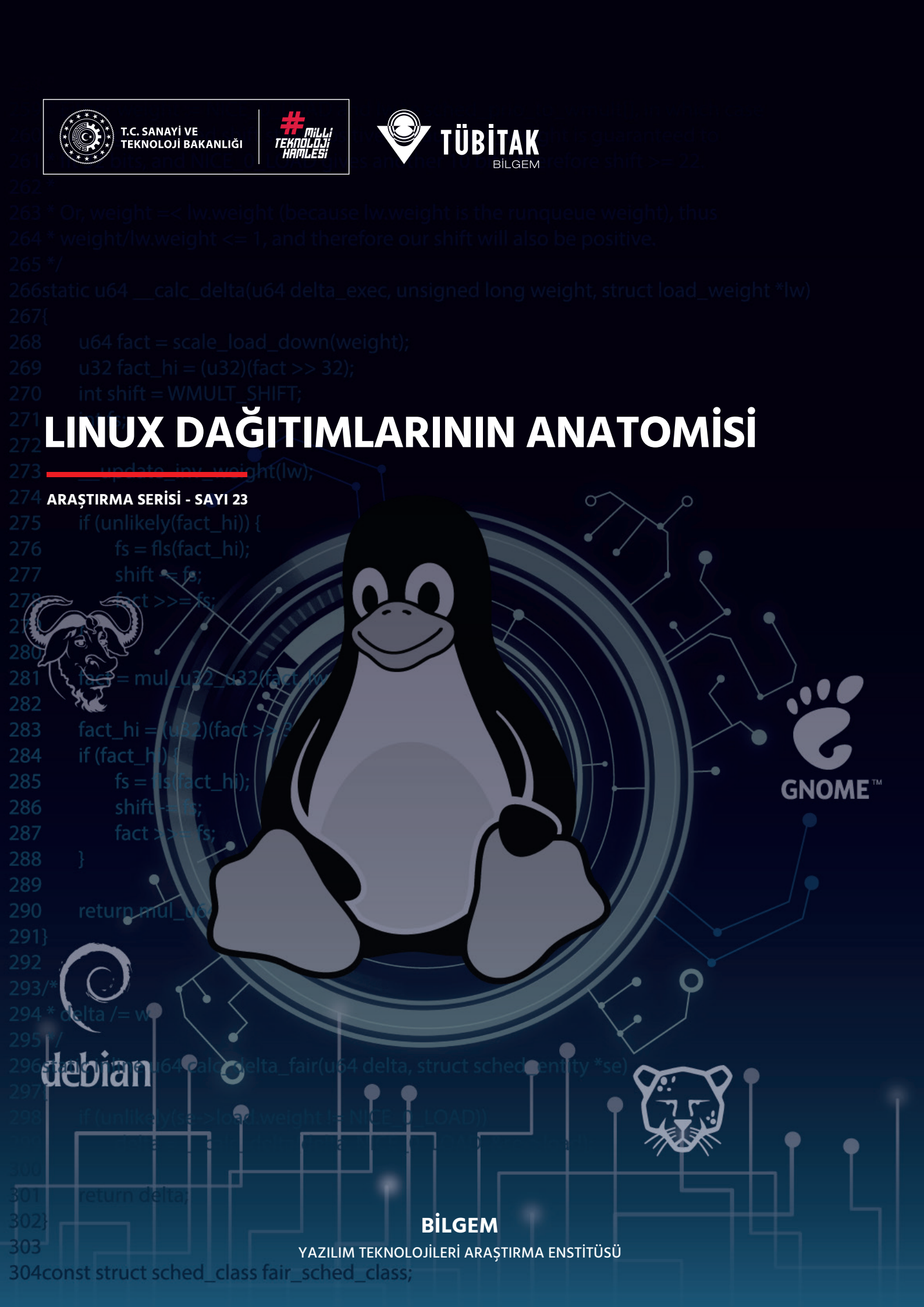
T.C. SANAYİ VE TEKNOLOJİ BAKANLIĞI

#MİLLİ TEKNOLOJİ HAMLESİ

TÜBİTAK BİLGEM

LINUX DAĞITIMLARININ ANATOMİSİ

ARAŞTIRMA SERİSİ - SAYI 23



BİLGEM

YAZILIM TEKNOLOJİLERİ ARAŞTIRMA ENSTİTÜSÜ

Simge ve Kısaltmalar

Kısaltmalar	Açıklama
TÜBİTAK	Türkiye Bilimsel ve Teknolojik Araştırma Kurumu
BİLGEM	Bilişim ve Bilgi Güvenliği İleri Teknolojiler Araştırma Merkezi
YTE	Yazılım Teknolojileri Araştırma Enstitüsü
APIC	Advanced Programmable Interrupt Controller (Gelişmiş Programlanabilir Kesme Denetleyicisi)
AT-SPI	Assistive Technology Service Provider Interface (Destekleyici Teknoloji Hizmet Sağlayıcı Arayüzü)
BSD	Berkeley Software Distribution (Berkeley Yazılım Dağıtımı)
CFS	Completely Fair Scheduler (Tamamen Adil Zamanlayıcı)
CIFS	Common Internet File System (Ortak İnternet Dosya Sistemi)
COW	Copy-on-Write (Yazarken Kopyala)
CPU	Central Processing Unit (Merkezi İşlem Birimi)
DE	Desktop Environment (Masaüstü Ortamı)
DHCP	Dynamic Host Configuration Protocol (Dinamik Ana Bilgisayar Yapılandırma Protokolü)
DNS	Domain Name System (Alan Adı Sistemi)
EDF	Earliest Deadline First (En Erken Son Tarih İlk)
FS	File System (Dosya Sistemi)
FSF	Free Software Foundation (Özgür Yazılım Vakfı)
FTP	File Transfer Protocol (Dosya Transfer Protokolü)
FVWM	F Virtual Window Manager (Zayıf Sanal Pencere Yöneticisi)
GCC	GNU Compiler Collection (GNU Derleyici Koleksiyonu)
GIMP	GNU Image Manipulation Program (GNU Görüntü İşleme Programı)
GNOME	GNU Network Object Model Environment (GNU Ağ Nesne Modeli Ortamı)
GNU	GNU's Not UNIX (GNU UNIX Değildir)
GPG	GNU Privacy Guard (GNU Gizlilik Koruması)
GPL	GNU General Public License (GNU Genel Kamu Lisansı)
GRUB	Grand Unified Bootloader (Büyük Birleşik Önyükleyici)
GTK	Gimp ToolKit (old)/Toolkit (Gimp Araç Takımı / Araç Takımı)
GUI	Graphical User Interface (Grafiksel Kullanıcı Arayüzü)
HTTP	Hyper Text Transfer Protocol (Hiper Metin Transfer Protokolü)
HTTPS	HTTP Secure (Güvenli Hiper Metin Transfer Protokolü)
IBM	International Business Machines Corporation (Uluslararası İş Makineleri Şirketi)
IDT	Interrupt Descriptor Table (Kesme Tanımlayıcı Tablosu)
IOT	Internet of Things (Nesnelerin İnterneti)
IP	Internet Protocol (Genel Ağ Protokolü)

Simge ve Kısaltmalar

Kısaltmalar	Açıklama
IRQ	Interrupt Request (Kesme İsteği)
ISO	International Organization for Standardization - Disk Image (Uluslararası Standardizasyon Kuruluşu - Disk İmajı)
KDE	K Desktop Environment (K Masaüstü Ortamı)
LAPIC	Local Advanced Programmable Interrupt Controller (Yerel Gelişmiş Programlanabilir Kesme Denetleyicisi)
LXDE	Lightweight X11 Desktop Environment (Hafif X11 Masaüstü Ortamı)
LXQt	Lightweight X11 Qt Desktop Environment (Hafif X11 Qt Masaüstü Ortamı)
MAC Address	Media Access Control Address (Ağ aygıtı fiziksel adresi)
MIT	Massachusetts Institute of Technology (Massachusetts Teknoloji Enstitüsü)
Multics	Multiplexed Information and Computing Service (Çoklama Bilgi ve Bilgi İşlem Hizmeti)
NFS	Network File System (Ağ Dosya Sistemi)
OpenSSL	Open Secure Sockets Layer (Açık Güvenli Yuva Katmanı)
OS	Operating System (İşletim Sistemi)
OSI	Open Source Initiative (Açık Kaynak Girişimi)
PGD	Page Global Directory (Sayfa Küresel Dizini)
PID	Process ID (Proses Kimlik Numarası)
POSIX	Portable Operating System Interface (Taşınabilir İşletim Sistemi Arayüzü)
procfs	Process File System (Proses Dosya Sistemi)
PTE	Page Table Entry (Sayfa Tablosu Girdisi)
Qemu	Quick Emulator (Hızlı Emulator)
Qt	Q Toolkit (Q Araç takımı)
RAM	Random Access Memory (Rastgele Erişimli Geçici Hafıza)
TCP	Transmission Control Protocol (Aktarım Denetim Protokolü)
TCP/IP	Transmission Control Protocol/Internet Protocol (İletim Kontrol Protokolü/İnternet Protokolü)
TID	Thread Identifier (İş Parçacığı Kimliği)
TWM	Tom's/Tab Window Manager (Tom'un Pencere Yöneticisi)
VFS	Virtual File System (Sanal Dosya Sistemi)
VMAs	Virtual Memory Areas (Sanal Bellek Alanları)
VM	Virtual Machine (Sanal Makine)
X11	X Window System Version 11 (X Pencere Sistemi Sürüm 11)
XFCE	X Forms Common Enviroment (X Formları Ortak Ortamı)
XFS	X File System (X Dosya Sistemi)
XML	Extensible Markup Language (Genişletilebilir İşaretleme Dili)

Yazarlar
Cihangir AKTÜRK
Şenol ALDIBAŞ
Fatih ALTUN
Ali Rıza KESKİN

Yayın Koordinatörü
Tuğçe YILMAZ

Editörler
Beyza ŞENEL
Kübra ERTÜRK
Nurhan ÖNER

Tasarım
Özge DOĞAN

İletişim: 0(312) 289 92 22 - yte.bilgi@tubitak.gov.tr

https://bilgem.tubitak.gov.tr/yte/

İçindekiler

Bölüm 2

Ön Söz

TÜBİTAK BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü (YTE), 2012 yılından bu yana yazılım teknolojilerinde Ar-Ge faaliyetleri yürüten bir araştırma kuruluşudur. Araştırma faaliyetlerinde elde ettiği birikimini stratejik, hassas ve kritik projeler yürüterek kamu adına hayata geçirmekte; kurumlarımıza dijital dönüşüm, yazılım geliştirme teknolojileri ve kalite süreçleri konusunda danışmanlık vermektedir.

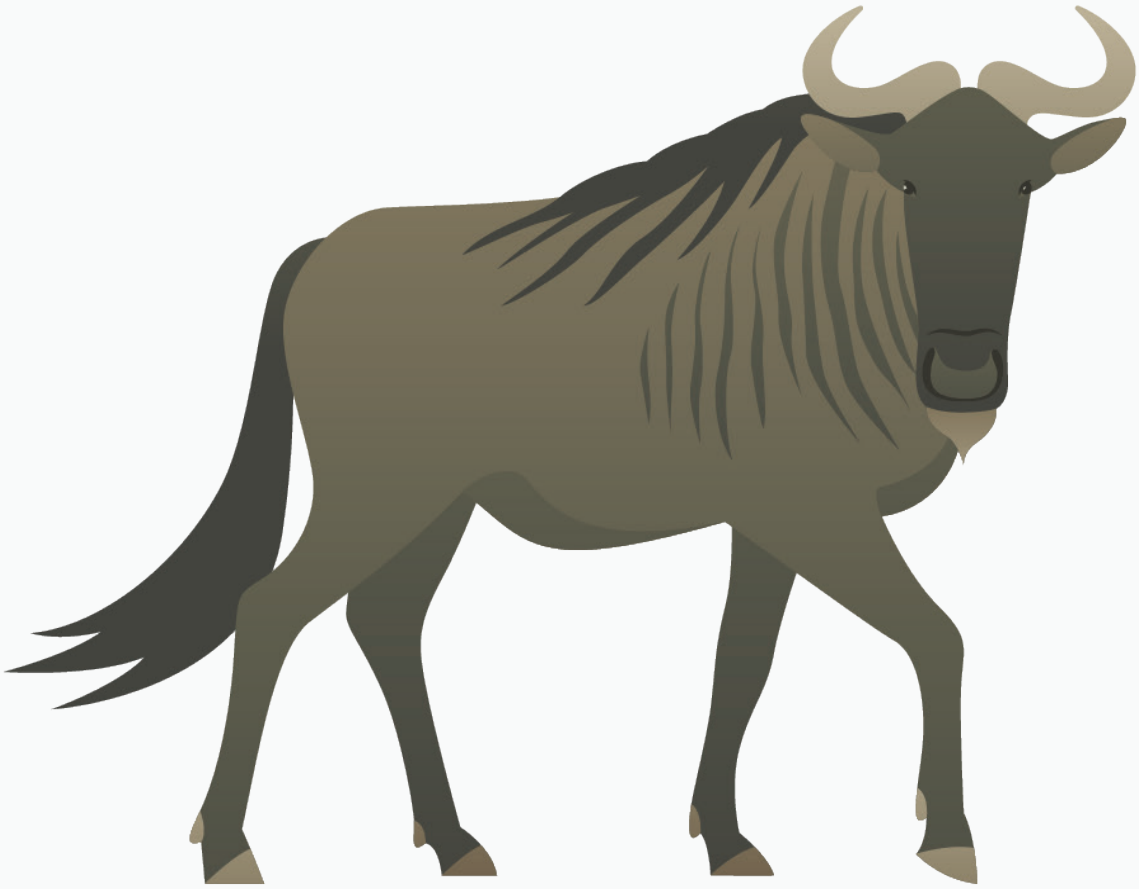
TÜBİTAK BİLGEM YTE tarafından hazırlanan Araştırma Serisi ile kurum içi içerik üretme çalışmalarının yaygınlaştırılması ve hazırlanan içeriklerin sektörün erişimine açılması amaçlanmaktadır. Araştırma Serisi'nde yayınlanan çalışmalar TÜBİTAK BİLGEM YTE çalışanlarının projelerde elde ettiği bilgi birikimini paylaşmak adına derlenmiştir. Bu çalışmalar ile ülkemizin yazılım sektörüne katkı sağlanılması hedeflenmektedir.

Giriş

Günümüzde UNIX ve Linux tabanlı işletim sistemleri, açık kaynaklı yapıları ve esneklikleriyle modern bilişim dünyasında önemli bir yer tutmaktadır. Bu çalışmada, UNIX'in tarihsel gelişiminden Linux'un evrimine kadar uzanan süreç ele alınarak, bu sistemlerin bilişim dünyasına etkileri değerlendirilmektedir. Ayrıca, güncel uygulamalar ve yenilikler analiz edilerek, gelecekte bilişim teknolojileri ve açık kaynak ekosisteminin nasıl şekillenebileceğine dair öngörüler sunulmaktadır.

Araştırma serisinin bu sayısında, UNIX benzeri işletim sistemleri beş ana başlık altında incelenmektedir. İlk olarak, UNIX'in temellerini atan zaman paylaşımlı sistemleri ve erken dönem platformları ele alınarak tarihsel bir çerçeve sunulmakta, ardından UNIX'in gelişimi ve temel ilkelerini şekillendiren yenilikler üzerinde durulmaktadır. Bu bağlamda, Linux'un ortaya çıkışı, büyümesini yönlendiren etkenler ve UNIX ilkelerini nasıl benimsediği ile genişlettiği detaylandırılmaktadır.

İkinci bölümde, Linux çekirdeğinin kritik alt sistemleri incelenerek proses ve bellek yönetimi, kesme işleme, proses zamanlaması ve sanal dosya sistemi gibi temel bileşenler ele alınmaktadır. Üçüncü bölüm, Buildroot kullanarak minimal bir Linux dağıtımı oluşturma sürecini ve Debian tabanlı bir dağıtım geliştirme aşamalarını açıklamaktadır. Dördüncü bölüm, UNIX/Linux masaüstü ortamlarının evrimini ve grafik arayüz uygulamalarını analiz etmektedir. Ek olarak yaygın masaüstü ortamları incelenerek, tasarım felsefeleri ve kullanıcı deneyimleri arasındaki etkileşim ortaya konmaktadır. Bu bölümleri takip eden okuyucular, UNIX ve Linux sistemlerine hem tarihsel hem de teknik açıdan kapsamlı bir bakış kazanacaklardır. Çalışmanın eklerinde, **Ek A: Tanımlar** ve **Ek B: Fonksiyon, Kütüphane ile İsimler** tablolarına yer verilerek açıklamalar sunulmuştur.



Bölüm 1
UNIX'in Öncülleri ve Tarihi Gelişimi

Linux'u anlamak ve anlatmak, UNIX'ten bahsetmeden mümkün değildir. Modern işletim sistemleri ailesinin önemli bir üyesi olan Linux, köklerini ve temel ilkelerini UNIX'ten almıştır. Bu nedenle, Linux'un ortaya çıkışını anlamak için öncelikle UNIX'in tarihine ve gelişim sürecine göz atmak önemlidir. UNIX, bilgisayar biliminin mihenk taşlarından biri olarak, modern işletim sistemlerinin temellerini atan bir sistemdir. Ancak bilgisayar teknolojisi ve işletim sistemlerinin gelişimi, UNIX'in öncesinde, onun ortaya çıkmasını sağlayan bir dizi atılıma ve öncüle dayanmaktadır. UNIX'in ortaya çıkışından itibaren tarihsel gelişimi ve bu süreçte etkili olan faktörler incelenecektir.

Bilgisayar Teknolojisinin İlk Adımları

UNIX öncesinde bilgisayar bilimindeki gelişimin kökenleri, 1950'li yıllarda bu alanda temel ilkeleri oluşturan ve aşağıda belirtilen önemli çalışmalara dayanmaktadır.

Batch (Toplu) Sistemler: İşlem süreçlerinin manuel olarak yürütüldüğü erken dönem bilgisayarlarında, süreçlerin verimli hale getirilmesi için programların toplu şekilde işlenmesini sağlayan ilk işletim sistemleridir.

Zaman Paylaşımlı Sistemler: 1960'larda geliştirilen zaman paylaşımlı sistemler, birden fazla kullanıcının bir bilgisayardan aynı anda yararlanabilmesine imkân tanımıştır.

Multics: UNIX'in İlham Kaynağı

UNIX'in doğuşunu anlamak için Multics'i anlamak çok önemlidir. 1960'lı yıllarda MIT, Bell Labs ve General Electric tarafından ortak geliştirilen Multics (Multiplexed Information and Computing Service), modern işletim sistemlerinin çok kullanıcılı ve çok görevli yapısının ilk örneğidir. Ancak, Multics'in karmaşıklığı ve ağır yapısı, geliştiriciler arasında memnuniyetsizliğe yol açmıştır. 1969 yılında Bell Labs, artık kısa vadede çalışan bir sistem elde edilemeyeceği anlaşılan Multics projesinden çekilme kararı almıştır [1]. Böylece Ken Thompson ve Dennis Ritchie gibi araştırmacılar, alıştıkları gelişmiş zaman paylaşım (time-sharing) ortamından yoksun kalmıştır. Bu dönemde, Multics üzerinde çalışan "Space Travel" oyununu oynamaya devam etmek isteyen Ken Thompson, atıl durumda olan bir PDP-7 mini bilgisayar bularak eşi ve çocuğunun akrabalarını ziyaret ettiği sırada birkaç hafta içerisinde UNIX'in ilk sürümünü geliştirmiştir ([2], [3]). Dennis Ritchie ise kısa süre sonra bu çabaya katılarak sistemi daha da geliştirmiş ve önce B programlama dilini, ardından da UNIX'in taşınabilirliğinin (portability) temelini oluşturan C dilini ortaya koymuştur.

Bu yeni işletim sisteminin en önemli özelliği, tıpkı Multics'de olduğu gibi, düz bir dosya sistemi yapısı (flat file system) yerine dizinler içinde dizinlere izin veren hiyerarşik dosya sistemiydi (hierarchical file system) ([2], [4]). Basitlik ve modülerliği merkeze alan bu tasarım, UNIX'i daha iddialı ancak fazla karmaşık kılan Multics'ten ayırmıştır. Sonuç olarak yalın, çok kullanıcılı ve çok görevli (multi-tasking)

bir ortam ortaya çıkmıştır. Bu gelişme, gelecekte neredeyse tüm internetin üzerinde çalışacağı bir işletim sistemi ailesi olarak kabul edilecek UNIX'in doğuşuna yol açmıştır.

Multics'ten UNIX'e

1960'lı yılların sonları, bilgisayar biliminin birçok açıdan evrim geçirdiği heyecan verici bir dönemdir. Bu dönemde, çok kullanıcılı ve işbirlikçi sistemlere olan ilgi artarken, ilgili alanda cesur denemeler yapılmıştır. Dönemin en dikkat çekici projelerinden biri, Multics'tir. Multics, o zamanların teknoloji anlayışına göre inanılmaz özelliklere sahip bir işletim sistemi oluşturmayı hedeflemiştir. Ancak bu cesur proje, planlandığı kadar başarıya ulaşamadan sona ermiştir. Bununla birlikte, bu proje UNIX'in ortaya çıkmasına zemin hazırlamıştır.

Multics: Süreç ve Zorluklar

1964 yılında MIT, General Electric ve Bell Labs bir araya gelerek Multics adını verdikleri işletim sistemi projesine başlamıştır. Bu sistemin hedefi, modern işletim sistemlerinin özelliklerinden birçoğunu barındıran, çok kullanıcılı, güçlü bir sistem tasarlamaktı. Özellikle dosya sistemi yapısı, bellek yönetimi ve çoklu görev özellikleriyle dikkat çekiyordu.

Ancak, bu ölçekte bir sistem tasarlamak, beklenenden çok daha zor olmuştur. Multics'in gelişim süreci hem maliyet hem de zaman açısından önemli bir yük haline gelmiştir. Sonunda, 1969 yılında Bell Labs, bu projeden çekilmeye karar vermiş, ancak bu geri çekiliş, Bell Labs'taki bir grup yazılımcının kendi başlarına çalışması için fırsat oluşturmuştur.

UNIX'in Doğuşu

Bell Labs'tan Ken Thompson, Dennis Ritchie ve Doug McIlroy gibi geliştiriciler, Multics'ten çıkan derslerle daha basit, kullanılabilir ve etkili bir sistem oluşturma fikrini benimsemişlerdir. Thompson, o dönemde PDP-7 adı verilen basit bir bilgisayar sistemi üzerinde "Space Travel" adlı bir oyun geliştirmekteydi, ancak bu oyunun mevcut sistemlerde çok yavaş çalışması nedeniyle kendi işletim sistemini yazmaya karar vermişti.

Bu yeni işletim sistemi, basitlik üzerine kurulu olup geliştiricileri korkutabilecek karmaşıklıklardan kaçınacak şekilde tasarlanmıştır. 1970 yılında bu sisteme esprili bir yaklaşımla "UNIX" adı verilmiştir. Bu isim, Multics'in daha sade ("uni" – tek kullanıcılı) versiyonunu temsil etmektedir.

UNIX'in geliştirilmesi aşağıdaki temel özelliklerle desteklenmiştir.

- **Basitlik ve Modülerlik:** UNIX, çekirdek yapısının yalın ve modüler yapıya dayalı olmasıyla öne çıkmıştır.
- **Dil Devrimi:** Dennis Ritchie tarafından geliştirilen C programlama dili, UNIX'in taşınabilir olmasını sağlamıştır. Bu özellik, UNIX'in farklı donanım platformlarında çalışabilmesine olanak tanımıştır.

UNIX'in Yaygınlaşması

1970'lerin ortalarından itibaren UNIX, akademik dünyada ve ticari alanda yayılmaya başlamıştır. Berkeley Üniversitesi'nde geliştirilen BSD (Berkeley Software Distribution), UNIX'in popülerliğini artıran bir diğer önemli faktördür. AT&T, UNIX'in kaynak kodunu akademik kurumlara açık hale getirdiği için, öğrenciler ve akademisyenler sistem üzerinde deneyler yapma ve geliştirme fırsatı bulmuştur. Bu esnek ve açık yapı, UNIX'in sadece bir işletim sistemi değil, yazılım dünyasında bir hareket haline gelmesini sağlamıştır. Bugün Linux, macOS ve FreeBSD gibi sistemlerin temelinde UNIX yatmaktadır.

Özellikle FreeBSD işletim sistemi, doğrudan Kaliforniya Üniversitesi Berkeley'de geliştirilen Berkeley Yazılım Dağıtımı (BSD) üzerinden Bell Labs'ın orijinal UNIX sistemine dayanmaktadır[4]. İlk BSD sürümleri AT&T'nin Araştırma (Research UNIX) UNIX'inden kaynak kodunu dahil ederek bir varyant oluşturmuştur. Ancak 1990'ların başında, genellikle USL ve BSD'i davası olarak bilinen yasal bir anlaşmazlık, AT&T'den türetilen kodun BSD tabanlı sistemlerde dağıtılmasıyla ilgili soruları gündeme getirmiştir. Bu anlaşmazlık, tartışmalı dosyaları kaldıran veya değiştiren 4.4BSD-Lite'ın piyasaya sürülmesine sebep olmuş ve FreeBSD gibi modern türevlerin AT&T köklerinden tamamen bağımsız olarak gelişmesinin önünü açmıştır. Özellikle BSD, VAX platformunda en erken talep sayfalı sanal bellek (demand-paging virtual memory) uygulamalarından birini tanıtarak bellek yönetiminde devrim niteliğinde değişikliklere yol açmıştır.

UNIX'in yaygınlaşmasını destekleyen önemli faktörler:

- **AT&T'nin Rolü:** Bell Labs'ın ana şirketi olan AT&T, UNIX'in ticari lisanslamasını yapmış ve akademik dünyada yayılmasına katkıda bulunmuştur.
- **BSD (Berkeley Yazılım Dağıtımı):** UNIX'in akademik kullanımını yaygınlaştıran çok önemli bir versiyonudur ve açık kaynak kodlu geliştirme hareketinin temel taşlarından biri olmuştur.

Teknolojik Yenilikler

UNIX'in getirdiği yenilikler, sistem tasarımında devrim yaratmıştır.

Dosya Sistemi: Kullanışlı ve hiyerarşik bir yapıya sahip dosya sistemi tasarımlanmış olup bu sistem, bugünkü dosya yönetim şemalarının temelini oluşturmaktadır.

Basitlik: Kod yapısı son derece sade tutulmuştur. Karmaşıklığı azaltılan sistem, geliştiriciler için çok daha kolay yönetilebilir hale gelmiştir.

C Programlama Dili: Dennis Ritchie tarafından geliştirilen C dili, UNIX'in taşınabilir olmasını sağlayarak farklı donanım platformlarına uyarlanmasını kolaylaştırmıştır.

Araçlar ve Borular (Pipes): UNIX'in modüler yapısı, küçük araçları birleştirerek karmaşık işlevlerin yerine getirilmesini sağlamıştır.

Projenin Hayata Geçmesine Katkı Sağlayan Önemli Kişiler ve Katkıları

Ken Thompson: UNIX'in ilk çekirdeğini tasarlayan kişidir. Sadeliğe olan inancı, sistemin tasarımına yön vermiştir.

Dennis Ritchie: UNIX'in çekirdeğini yazmak için C dilini geliştirerek, yazılım geliştirme dünyasında büyük bir yenilik yaratmıştır.

Brian Kernighan: UNIX adını bulan ve sistemin yayılmasına katkı sağlayan kişidir.

Doug McIlroy: Sistemin esnekliğini çok büyük oranda artıran UNIX boruları (pipes) fikrini ortaya atan kişidir.

Açık Kaynak ve Özgür Yazılım Hareketi: Bir Devrim

Açık Kaynak ve Özgür Yazılım Hareketi teknoloji dünyasında dönüştürücü bir değişimi tetiklemiştir. Temelinde yazılımın yalnızca bir ürün değil, aynı zamanda bilgi alışverişi için bir araç olduğu anlayışı yer almaktadır. Bu hareket, kullanıcının yazılımı özgürce kullanma, yeniden şekillendirme, paylaşma ve dolaştırma hakkını savunmaktadır.

Özgür Yazılımın Doğuşu

Özgür yazılım, MIT'nin Yapay Zeka Laboratuvarı gibi kurumların toplumsal paylaşım kültüründe geliştiği 1950'lerde ve 60'larda ortaya çıkmıştır. O zamanlar yazılım donanıma açıkça eşlik ediyor, mühendisler arasında elden ele dolaşıyor ve bir iş birliği ruhu besliyordu. MIT'nin laboratuvarları, kodun kilit altında olmadığı, paylaşılan bir kaynak olarak geliştiği bu açık ruh için verimli bir zemin haline gelmiştir.

Richard Stallman ve GNU'nun Doğuşu

1980'lerde özel mülkiyetli yazılımların özgürlüğü kısıtlamasıyla Richard Stallman, 1983'te tamamen özgür bir işletim sistemi öngörerek GNU Projesi'ni başlatmıştır. Stallman, bu vizyonu korumak için GNU Genel Kamu Lisansı'nı (GPL) tasarlamıştır. Bu öncü lisans, yazılım özgürlüğünü koruyup, geliştiricilerin kaynak kodunu erişilebilir ve türevlerini eşit derecede açık tutmasını zorunlu hale getirmiştir.

Linus Torvalds ve Açık Kaynak Dönemi

1991'de Linus Torvalds, GNU'nun eksik parçasını tamamlayarak Linux çekirdeğini serbest bırakmıştır. Bu bileşenler bir araya gelerek GNU/Linux işletim sistemini oluşturarak özgür yazılımı küresel bir fenomene dönüştürmüştür.

1998'de, özgür yazılımı geniş bir şekilde benimsenmesi için yeniden çerçevelemek üzere Açık Kaynak Girişimi (OSI) ortaya çıkmıştır. Eric Raymond'un The Cathedral and the Bazaar adlı makalesi, açık geliştirmenin yenilikçiliği ve esnekliği ateşleme gücünü övmektedir.

Dönüm Noktaları ve Vizyonerler

1985 : Stallman, Özgür Yazılım Vakfı'nı (FSF) kurmuştur.

1990'lar : Apache Web Server gibi projeler interneti yeniden tanımlamıştır.

2004 : Kullanıcı dostu bir Linux dağıtımı olan Ubuntu piyasaya sürülmüş ve internet erişimini genişletmiştir.

Günümüzde açık kaynak teknolojiler, teknolojiyi giderek güçlendirmekte, Google ve Microsoft gibi teknoloji devleri bu yaklaşımı desteklemektedir. Kubernetes ve Git gibi projeler ekosistemde önemli bir yer edinirken, açık kaynak yazılımın gelecekte de teknolojik gelişimin temel unsurlarından biri olmaya devam etmesi beklenmektedir.

Açık Kaynak ve Özgür Yazılım Hareketi kapsayıcı bir yazılım kültürü oluşturarak, şeffaflığı, iş birliğini ve gelişimi ön plana çıkarmıştır. İleriye bakıldığında, yazılımın etkisinin yazılımdan çok daha öteye geçmesi ve dokunduğu her alanda yenilikler doğurması muhtemel görünmektedir.

Linux Çekirdeğinin Doğuşu: Linus Torvalds ve Bir Dünya Değişimi

1991 yılında Finlandiya'nın Helsinki şehrinde, bir üniversite öğrencisi olan Linus Torvalds, bilgisayar bilimi tarihini en fazla etkileyecek projelerinden birini başlatmıştır. Henüz 21 yaşında olan Torvalds, kısa süre içerisinde teknoloji dünyasını kökten değiştirecek olan Linux çekirdeğini (Linux kernel) geliştirmeye başlamıştır. Bu çalışmada, Linux çekirdeğinin oluşturulma süreci, Torvalds'ın motivasyonu ve bu ürünün modern bilişim dünyasına etkileri ele alınacaktır [6].

Yeni Bir Çekirdeğin Doğuşu

Linus Torvalds, Helsinki Üniversitesi'nde bilgisayar bilimi bölümünde lisans öğrencisi olarak eğitim gördüğü dönemde, çok sayıda öğrenci ve geliştirici, UNIX benzeri işletim sistemlerini öğrenmek ve incelemek istemekteydi. Ancak, ticari UNIX sistemleri (AT&T'nin orijinal UNIX'i gibi) çok pahalı ve kaynak koduna erişim son derece kısıtlıydı. Torvalds, bu eksiklikleri gidermek için Andrew S. Tanenbaum tarafından eğitim amaçlı geliştirilmiş, UNIX tabanlı bir mikro-cekirdek işletim sistemi olan MINIX adlı bir işletim sistemini kullanmaya başlamıştır. Bununla birlikte, MINIX'ın özelliklerinin sınırlı olması Torvalds'ın kendi çözümünü geliştirme motivasyonunu tetiklemiştir ([6], [8]).

1991 yılında Torvalds, MINIX'ın yerine geçecek, daha özgür ve esnek bir sistem yaratmak amacıyla kendi işletim sistemi çekirdeğini geliştirmeye başlamıştır. Torvalds bu yeni çekirdeği, kendi sahip olduğu bilgisayar donanımında çalışacak şekilde tasarlamış ve GNU Projesi'nden büyük ölçüde faydalanmıştır. Bu dönemde Torvalds, Intel 80386 işlemcinin çoklu iş parçacığı (multitasking) işlemlerini ve process switching mekanizmalarını destekleyen talimat setlerini (örneğin, "task switching" ile ilgili özel komutları) incelemiştir. Torvalds, bu bilgi birikimini Linux çekirdeğinin temel tasarımına entegre ederek, işlemcinin sunduğu donanımsal avantajlardan faydalanmıştır [10].

Topluluğun Katkıları ve Linux Adının Doğuşu

25 Ağustos 1991 tarihinde, Torvalds, yeni projesini tanıtmak için Usenet'ın comp.os.minix haber grubunda şu mesajı yayımlamıştır;

“Merhaba herkese. Minik bir proje üzerinde çalışıyorum (bir hobi projesi), MINIX'e benzer bir işletim sistemi. Bu sadece şahsi bir projedir ve ticari bir amacı yoktur.” [9]

Torvalds, bu açıklamasında, geliştirdiği çekirdeğin tamamen özgür olacağını vurgulamıştır. İlk sürümü, 17 Eylül 1991 tarihinde ücretsiz olarak yayımlanmış olup sürecin devamında, Torvalds'ın pro-

jesi geliştiricilerin ilgisini hızla çekmeye başlamıştır. GNU Projesi'nin özgür yazılım hedefi ve Richard Stallman'ın lisanslama sistemi (GPL) sayesinde Linux, hızlı bir şekilde açık kaynak kodlu topluluk tarafından benimsenerek desteklenmeye başlamıştır [7].

Linux adı, aslında Torvalds tarafından öncelikli olarak kullanılmamıştır. Çekirdeğini ilk başta "Freax" olarak adlandırmış, ancak FTP sunucusunu yöneten arkadaşı Ari Lemmke, bu adın uygun olmadığını düşünerek, projeyi "Linux" adıyla yayımlamıştır.

Linux'un Evrimi ve İş birliği

Linux'un gelişiminde açık kaynak kodlu topluluğun çok büyük bir etkisi olmuştur. Torvalds, bir lider olarak, topluluğun katkılarını etkili bir şekilde yönetmiş ve bu sayede Linux çok farklı donanım platformlarında çalışabilir hale gelmiştir.

1990'ların sonunda Linux hem bireysel geliştiriciler hem de şirketler tarafından benimsenmiştir. IBM, Red Hat ve Novell gibi şirketler, Linux tabanlı projelere yüksek miktarda yatırım yapmış ve bu da çekirdeğin endüstriyel uygulamalarda da kullanılmasını sağlamıştır.

Çekirdek Mimarisi

Linux çekirdeği, proses yönetimi, bellek yönetimi, aygıt sürücüleri ve dosya sistemleri gibi tüm temel bileşenleri tek bir büyük çalıştırılabilir dosyaya entegre eden monolitik bir çekirdek mimarisi kullanmaktadır. Bu tasarım bileşenler arasında hızlı iletişim sağlar, ancak kararlılık ve güvenliği sağlamak için dikkatli bakım gerektirir.

Monolitik Çekirdek ve Mikro Çekirdek

Monolitik Çekirdek: Sürücüler ve bellek yönetimi gibi tüm bileşenler çekirdek uzayında çalışır, bu da yüksek performans sağlar, ancak bir hata meydana geldiğinde sistem genelinde çökme riski vardır.

Mikro Çekirdek: Yalnızca temel bileşenler (örneğin, bellek ve proses yönetimi) çekirdek uzayında çalışırken diğer hizmetler kullanıcı uzayında çalışır. Bu, daha iyi modülerlik ve hata toleransı sunar, ancak performans ek yüküne neden olabilir.

Bölüm Özeti

Linus Torvalds'ın hobi olarak başlattığı Linux projesi, bugün dünyanın en önemli açık kaynak kodlu projelerinden biri haline gelmiştir. Günümüzde çekirdek, Android cihazlardan süper bilgisayarlara kadar çok geniş bir yelpazede kullanılmaktadır. Linux'un doğuşu, bireylerin tutkusunun ve topluluk desteğinin teknolojiye nasıl büyük etkiler yapabileceğini gösteren en iyi örneklerden biridir.

Linux çekirdeği, Linux işletim sisteminin kullanıcı uygulamaları ile donanım kaynakları arasında aracılık eden kalbidir. Çalışan tüm prosesler için verimli iletişim ve kaynak tahsisi sağlar, güvenlik, çoklu görev ve çeşitli donanım platformları için destek sunar.

Bölüm 2
Linux Çekirdeğinin Temel Alt Sistemlerine Kapsamlı Bir Bakış

Linux çekirdeği, 1991 yılında Linus Torvalds tarafından başlatılmış monolitik bir işletim sistemi çekirdeğidir. Son otuz yılda, karmaşıklığı ve benimsenme oranı önemli ölçüde artmış, pek çok Linux dağıtımının ve gömülü sistemin yapı taşı haline gelmiştir. Mimarisi, küçük gömülü cihazlardan büyük ölçekli sunuculara ve ana bilgisayarlara kadar geniş bir donanım yelpazesinde esnek, yüksek performanslı ve taşınabilir olacak şekilde tasarlanmıştır.

Linux'u anlamanın kritik noktalarından biri, proses management (proses yönetimi), memory management (bellek yönetimi), proses scheduling (proses zamanlama), interrupt (kesme) ve interrupt handling, proses address spaces ve Virtual File System (VFS) katmanı gibi çekirdeğin temel alt sistemlerini gözden geçirmektir. Bu alt sistemler birlikte çalışarak kullanıcı uygulamalarının verimli ve kararlı bir şekilde görev yürüttüğü istikrarlı bir ortam sağlamaktadır. Linux'un açık kaynak doğası nedeniyle, çekirdek dünya çapındaki geliştiriciler tarafından yeni donanımlara ve yazılım gereksinimlerine uyum sağlamak amacıyla sürekli iyileştirilmekte hata düzeltmeleri eklenmekte ve yenilikçi özellikler kazandırılmaktadır.

Bu bölümde, söz konusu her bir bileşenin mimari motivasyonlarını, veri yapılarını, algoritmalarını ve çekirdeğin geri kalanıyla olan etkileşimini açıklanarak alt sistemlere derinlemesine bir bakış sunulmaktadır. Ayrıca güncel ve doğru bakış açıları sunabilmek için resmi belgelere atıflar yapılarak farklı çekirdek sürümlerinde nasıl evrildiklerine dair bilgiler verilmektedir.

Proses Yönetimi

Linux'ta Proseslerin Rolü

Linux'ta bir proses, çalışan bir programın örneğini ifade etmekle birlikte, Kod (text), veri (data), yığın (stack) ve çeşitli çekirdek taraflı veri yapılarını (örneğin, dosya tanımlayıcılar, bellek eşlemeleri) kapsamaktadır. Prosesler temeldir, çünkü kullanıcı-uzay uygulamalarının yürütme bağlamlarını oluşturmaktadırlar. Çekirdek, bu prosesleri aşağıdaki işlevler aracılığıyla yönetmektedir;

1. Yeni prosesler oluşturma (kullanıcı alanındaki fork(), clone() sistem çağrıları ve pthread_create() gibi üst seviye sarmalayıcılar).
2. Prosesleri fiziksel veya sanal CPU'larda çalıştırma (scheduling).
3. Bellek koruması ile bir proses başka bir prosesin belleğine müdahale etmesini engelleme.
4. Pipe, soket, paylaşımlı bellek vb. yöntemlerle prosesler arası iletişime olanak sağlama.

Kullanıcı herhangi bir komut çalıştırdığında veya bir uygulama başlattığında, çekirdek en az bir proses yaratır (çoklu iş parçacıklı bir uygulamada daha fazla olabilir). Geleneksel olarak proses tek bir yürütme akışını ifade etse de Linux'ta iş parçacıkları (thread) da aynı kavram üzerinden tanımlanır. Aslında Linux'ta thread'ler, bellek alanları, dosya tanımlayıcılar ve sinyaller gibi belirli kaynakları paylaşan proseslerdir [17].

Proses Yönetimi için Veri Yapıları

Proses yönetiminin temel çekirdek veri yapısı task_struct'da ve include/linux/sched.h başlık dosyasında tanımlanır. Bu yapı şu bilgileri içerir:

- **Proses Durumu (Process State):**

TASK_RUNNING, TASK_INTERRUPTIBLE, TASK_UNINTERRUPTIBLE, TASK_STOPPED veya TASK_TRACED olabilir.

- **Proses Tanımlayıcıları:** PID (Proses ID), TID (Thread ID), ebeveyn PID vb.
- **Scheduling Bilgisi:** Scheduling sınıfı, öncelik, kullanılan CPU süresi ve benzeri verilerdir.
- **Bellek Yönetimi (Memory Management) Bilgisi:** Proses adres alanını betimleyen mm_struct göstericisidir.
- **Açık Dosyalar:** Dosya tanımlayıcı tablosu (files_struct).
- **Sinyal İşleyicileri:** SIGINT, SIGKILL vb. sinyallerin nasıl ele alınacağını saklayan bir yapıdır.
- **İş Parçacığına Özgü Veriler:** Çekirdek yığını, CPU'ya özgü kayıtlar vb.

Çoklu iş parçacığı desteğiyle birlikte, bir proses birden çok task_struct'a bölünebilir, ancak hepsi ortak bir mm_struct'ı paylaşabilir. Bu tasarım, iş parçacıklarının (thread) aynı adres alanlarını paylaşmasına olanak tanırken, her biri için ayrı yürütme bağlamını korur ([11], [12]).

Proses Oluşturma ve Sonlandırma

Linux'ta proses oluşturma mekanizması geleneksel olarak fork() sistem çağrısıyla tetiklenir ve bu, çağıran prosesi kopyalar. Ancak modern Linux sistemlerinde, çekirdek içinde daha genel clone() çağrısı kullanılmaktadır. clone()'a verilen bayraklar (flags), child prosesin (alt sürecin) ebeveyniyle hangi kaynakları paylaşacağını belirler (örneğin dosya tanımlayıcılar, adres alanı veya sinyal işleyiciler) [15].

1. **fork():** Child proses, ebeveynin neredeyse birebir kopyasıdır, ancak PID gibi bazı farklılıklar vardır. Adres alanı, Copy-on-Write (COW) mekanizması sayesinde birbirinden bağımsız hale gelir [15].
2. **vfork():** Child prosesi hızlıca exec() çağrısı yapacaksa gereksiz kopyaları önlemek için ebeveynin adres alanını geçici olarak paylaşan bir varyanttır.
3. **clone():** İnce ayar yapılmasını sağlar. CLONE_VM gibi bayraklarla adres alanı paylaşılan thread'ler oluşturulabilir.

Yeni oluşturulan child proses, exec() ailesinden [örn. execve()] bir fonksiyon çağırarak kendi

adres alanına yeni bir program yükleyebilir. Bir proses tamamlandığında exit() çağrısı (ya da kullanıcı alanında yürütmenin bitmesi) ile sonlanır; çekirdek kaynaklarını serbest bırakır. Ebeveyn proses, wait() veya benzeri fonksiyonlarla child'ın çıkış durumunu alabilir.

Bağlam Değişimi (Context Switch)

Context switch, CPU'nun yürütmeyi bir proses (veya thread) üzerinden alıp başka birine aktarmasıdır. Bu işlem, mevcut prosesin CPU durumunun (kayıtlar, program sayacı, yığın işaretçisi vb.) saklanmasını ve sonraki prosesin durumunun yüklenmesini içerir. Context switch, mimariye özgü assembly dili rutinleriyle yapılırken, üst düzeyde zamanlayıcı (scheduler) tarafından yönetilir [14].

Linux çekirdeği, context switch yükünü olabildiğince hafifletmeye çalışır, çünkü çok sık gerçekleşmesi performans düşüklüğüne neden olabilir. Öte yandan, çoklu görev için de vazgeçilmezdir. Verimli context switch aşağıdaki faktörlere dayanır:

- Kaydedilecek ve geri yüklenecek işlemci durumunu düzenleyen ve uygun duruma getiren assembly kodu.
- Önbellek davranışı (örneğin, thread'ler aynı adres alanını paylaşıyorsa TLB flush'larının en aza indirilmesi).
- Çekirdeğin scheduling algoritması (**Bölüm 2**'de ayrıntılı olarak incelenecektir).

Bellek Yönetimi (Memory Management)

Linux Memory Management'a Genel Bakış

Linux'ta Memory Management, düşük bellekli gömülü sistemlerden terabaytlarca RAM'e sahip büyük sunuculara kadar geniş kullanım senaryolarını kapsayacak şekilde tasarlanmıştır. Sistem, belleği verimli şekilde tahsis edip yönetmeli, sanal bellek adreslerini idare etmeli, gerektiğinde swap (takas) yapmalı ve prosesler arasındaki güvenlik sınırlarını korumalıdır. Temel amaçlar şunlardır:

1. **Koruma (Protection):** Proseslerin birbirlerinin belleğine zarar vermesini engellemek.
2. **Sanal Bellek (Virtualization):** Her prosese kesintisiz bir adres alanı sunmak ve bu alanı fiziksel RAM'e (veya swap alanına) gerektiği kadar eşlemek.
3. **Talep Üzerine Paging (Demand Paging):** Diskten RAM'e sayfaları sadece ilk erişim anında yüklemek.

Kavramsal olarak, Linux bellek yönetimi alt katmanlardan oluşur:

- **Virtual Memory katmanı:** Her prosese fiziksel detayları soyutlanmış bir sanal adres alanı sunar.
- **Fiziksel Bellek katmanı:** Gerçek RAM çerçevelerini (page frames) yönetir.
- **Swap:** Paging ve önbellekleme (caching) mekanizmaları, bellek kullanımının dinamik yönlerini ele alır.

Fiziksel ve Sanal Bellek (Virtual Memory) Yerleşimi

Modern 64-bit Linux sistemlerinde (x86_64), virtual memory adres alanı kullanıcı bölümü ile çekirdek bölümü arasında bölünmektedir. Örneğin, çekirdek üst adres alanında (birçok dağıtımda ffff800000000000 gibi) haritalanırken, alt adresler kullanıcı prosesler için ayrılmıştır.

- **Kullanıcı Uzayı (User-space):** Genelde 0 adresinden belirli bir üst sınıra kadar olan bölümdür.
- **Kernel Uzayı (Kernel-space):** Çekirdeğin kodunu, statik verilerini, modüllerini ve bazen fiziksel belleğin doğrudan haritalamasını içerir.

Kernel uzayı her proseste aynıdır (global olarak haritalanır), ancak kullanıcı uzayı her proseste farklıdır. Bu düzen, çekirdek geliştirmeyi basitleştirir çünkü çekirdek adreslerine sabit şekilde erişilebilir.

Paging ve Swapping

Linux, sayfa (page) temelli bir virtual memory modelini kullanmaktadır. Bellek, sabit boyutlu sayfalara (genellikle x86'da 4 KB) bölünmüştür. Daha büyük sayfalarda (2 MB, 1 GB huge pages) destek mevcuttur.

- **Paging:** Her sanal adres, bir sayfa tablosu hiyerarşisindeki kayda (örn. x86_64'te dört veya beş seviyeli) karşılık gelir. Bu tablolar, sanal sayfaları fiziksel çerçevelere eşler.
- **Swapping:** Fiziksel bellek yetersiz kalırsa sayfalar disk (swap alanı) üzerine taşınabilir. Linux, bellek baskısını hafifletmek için swap yapar ancak bunu mümkün olduğunca minimumda tutmayı amaçlar. swappiness parametresi, çekirdeğin swap yapma eğilimini ayarlar.

Bir proses, RAM'de bulunmayan bir sayfaya erişmeye çalıştığında, page fault (sayfalama arızası) oluşur. Çekirdek, hatalı adresin geçerli olup olmadığını kontrol eder; geçerliyse ya swap alanından sayfayı RAM'e yükler ya da ilk defa erişiliyorsa sıfırla doldurur. Geçersizse prosese SIGSEGV sinyali gönderilir.

Memory Bölgeleri (Zones) ve Tahsis

Fiziksel bellek, donanım ve çekirdek kısıtlarını karşılamak için farklı zones birimlerine bölünmüştür:

- **ZONE_DMA:** Eski sistemlerde 16 MB altında DMA tahsisleri için.
- **ZONE_DMA32:** 64-bit sistemlerde 4 GB altındaki 32-bit DMA için.
- **ZONE_NORMAL:** Normal 32-bit doğrudan haritalama (direct mapping) bölgesi.
- **ZONE_HIGHMEM (32-bit sistemlerde):** 32-bit haritalamanın üstündeki bellek.
- **ZONE_MOVABLE:** Bellek hotplug veya düzenleme (defragmentation) sırasında taşınabilir sayfalar için.

Çekirdek, bellek tahsisinde çeşitli stratejiler kullanabilir:

- **Buddy Sistemi:** Bitişik sayfa çerçevelerini tahsis etmek için kullanılan birincil mekanizmadır.

- **Slab, SLOB ve SLUB Allocator'lar:** Küçük nesneler için uzmanlaşmış çekirdek bellek tahsisçileridir. Slab allocator, inode, dentry gibi sık kullanılan nesneleri organize eden önbellekler oluşturur. SLUB, daha modern ve basit bir alternatif iken, SLOB çok küçük sistemlerde kullanılır.

Slab, SLOB ve SLUB Allocator'lar

- **Slab Allocator:** Çekirdek veri nesnelerinin sıklıkla kullanıldığı durumlardaki performans ve parçalanma (fragmentation) sorununu iyileştirmek için tasarlanmıştır. Her önbellek bir veya daha fazla slab'a bölünür ve her slab içinde birden çok önceden tahsisli nesne bulunur.
- **SLOB (Simple List Of Blocks):** Gömülü veya çok küçük sistemlerde kullanılan, serbest bloklardan oluşan basit bir bağlantılı listeyle çalışan bir tahsisçidir.
- **SLUB (Unqueued Slab Allocator):** Daha modern, özellikle çok çekirdekli sistemlerde ölçeklenebilirlik ve hız için tasarlanmış bir slab alternatifidir.

Linux, çekirdek yapılandırması sırasında hangi allocator'ın kullanılacağını seçme esnekliği sunar ve böylece farklı ortamlara uyum sağlar.

Proses Scheduling

Scheduler Kavramları ve Amaçları

Scheduling, hangi prosesin ne zaman çalışacağına karar veren mekanizmadır. Temel amaçları:

1. **Adil Dağıtım (Fairness):** Tüm proseslere makul düzeyde CPU süresi sağlamak.
2. **Verimlilik (Efficiency):** Yüksek CPU kullanımı, etkileşimli işler için hızlı tepki süresi ve düşük ek yük sağlamak.
3. **Ölçeklenebilirlik (Scalability):** Yüzlerce veya binlerce prosesi birçok çekirdekli sistemde idare edebilmek.
4. **Gerçek Zamanlı Tepkisellik (Real-time Responsiveness):** Zaman kritik görevlerin süre sınırlarına uyabilmesidir.

Tarihsel olarak Linux, uzun süre O(N) zaman karmaşıklığına sahip (sched.c) bir scheduler kullanmıştır. 2.6 çekirdeğiyle birlikte O(1) scheduler'a geçilmiş ve ardından 2.6.23 civarında Completely Fair Scheduler (CFS) benimsenmiştir.

Completely Fair Scheduler (CFS)

CFS, önceliğine göre her çalışabilir prosese, "adil" CPU süresi dağıtma fikri üzerine kuruludur. Başlıca tasarım ögeleri:

- **Virtual Runtime (vruntime):** Bir görevin (task) ne kadar CPU süresi aldığının bir ölçüsüdür. Çalışabilir her task, vruntime değerine göre sıralanan kırmızı-siyah (red-black) bir ağaç üzerinde tutulur. Ağacın en solundaki düğüm, CPU süresini en az almış görevi (en düşük vruntime) temsil

eder ve bir sonraki çalışacak işlem olarak seçilir.

- **Targeted Latency:** CFS, belirli bir "hedef gecikme" süresi içinde her görevin en az bir kez çalışmasını amaçlar. Örneğin N görev varsa, bu gecikme süresi N'e bölünür.
- **Öncelikler:** Ağırlıklar (weights) ile uygulanır. Daha yüksek öncelik, daha büyük ağırlık ve dolayısıyla vruntime ın daha yavaş artışı anlamına gelir; bu da görevin daha sık çalışmasına yol açar.

Bu tasarım sayesinde CFS, çok çekirdekli sistemlerde bile CPU süresini daha öngörülebilir ve adil şekilde dağıtabilir.

Real-Time Scheduling Politikaları

Linux, POSIX gerçek zamanlı (real-time) scheduling politikalarını da desteklemektedir. Zamanlama açısından sıkı gereksinimleri olan uygulamalar için kritik öneme sahiptir:

1. **SCHED_FIFO:** İlk giren ilk çıkar (first-in, first-out) ilkesine dayalı gerçek zamanlı politikadır. SCHED_FIFO politikası ile çalışan bir görev, daha yüksek öncelikli bir başka gerçek zamanlı görev tarafından kesilinceye, gönüllü olarak CPU'yu serbest bırakıncaya veya bloke olana kadar çalışmaya devam eder.
2. **SCHED_RR:** Round-robin gerçek zamanlı politikadır. FIFO'ya benzer ancak her öncelik seviyesi için belirli zaman dilimleri kullanır.
3. **SCHED_DEADLINE:** Task'ların son tarihlerini (deadline) ve CPU rezervasyon parametrelerini kullanır ve çekirdek içinde Earliest Deadline First (EDF) ilkesini uygular.

Gerçek zamanlı prosesler, normal (CFS) proseslere göre her zaman önceliklidir, bu da süre sınırlarını karşılamalarını mümkün kılar.

Scheduling Sınıfları ve Dağıtım

Modern çekirdeklerde scheduling, scheduling sınıfları etrafında örgütlenmiştir ve her sınıf belirli bir politika veya politika ailesini uygular:

- **Stop Class:** Özel durumlarda (örneğin CPU'yu kapatmak gibi) CPU'ları durdurmak için kullanılır.
- **Deadline Class:** SCHED_DEADLINE'ı uygular.
- **RT Class:** Gerçek zamanlı politikaları (SCHED_FIFO, SCHED_RR) uygular.
- **CFS Class:** Normal görevler için varsayılan sınıftır.

Scheduler, en yüksek öncelik sınıfından en düşük öncelik sınıfına doğru giderek hangi görevin çalışacağına karar verir. Bu tasarım, çekirdeğin temel scheduling mantığını yeniden yazmaya gerek kalmadan yeni scheduling stratejilerinin entegre edilmesini sağlamaktadır.

Interrupt ve Interrupt Handling

Interrupt'lara Giriş

Interrupt, donanım veya yazılım (hata yakalama ya da tuzak/trap) tarafından işlemciye gönderilen ve acil dikkat gerektiren bir sinyaldir. Sistemlerin olay odaklı (event-driven) ve hızlı tepki verebilmesi için kritik öneme sahiptir. Örneğin, bir tuşa basıldığında, klavye denetleyicisi CPU'ya bir interrupt gönderir ve işletim sistemi klavyeden gelen verileri okuyarak işleyebilir. Interrupt'lar olmasaydı, sürekli donanımı sorgulamak gerekecek, bu da CPU zamanını boşa harcayacak ve sistem tepki süresini düşürecekti.

Interrupt türleri:

- **Donanım kesmeleri (Hardware interrupts):** Klavye, ağ arayüzü, zamanlayıcı gibi dış cihazlardan üretilir.
- **Yazılım kesmeleri (Software interrupts):** Yazılım talimatlarıyla üretilir (örneğin x86'da eski ABI'lerdeki sistem çağrıları için int 0x80, hata ayıklama tuzakları vb.) [20].
- **İstisnalar (Exceptions):** İşlemcide oluşan sayfa hatası (page fault), sıfıra bölme (divide by zero), geçersiz komut (invalid opcode) gibi özel durumlar.

Interrupt Descriptor Table (IDT)

x86 mimarisinde Interrupt Descriptor Table (IDT), işlemcinin belirli bir interrupt vektörüne nasıl tepki vereceğini belirlemek için başvurduğu bir tablo görevi görür. IDT'nin her girişi, ilgili interrupt işleyicisinin adresini içerir. Bir interrupt gerçekleştiğinde işlemci, vektör numarasını (bir indeks) kullanarak doğru işleyiciye atlar. Linux çekirdeği, IDT'yi önyükleme (boot) esnasında başlatır ve donanım IRQ'ları, yazılım interrupt'ları ve istisnalar için gerekli rutinleri buraya yazar.

Üst Yarı (Top Half) ve Alt Yarı (Bottom Half)

Bir interrupt geldiğinde, CPU önce top half olarak adlandırılan interrupt işleyici kodunu, interrupt context'inde ve genellikle diğer interrupt'lar kapalıyken çalıştırır. Bu kod mümkün olduğu kadar kısa tutulmalıdır. Top half genelde şu işlemleri yapar:

1. Interrupt'ı, ilgili cihaz ve interrupt denetleyiciye (controller) onaylar.
2. Gerekli verileri okur (örneğin cihaz kaydındaki durum bilgilerinin okunması).
3. Daha fazla işlemin ileride yapılması için (bottom half) planlama yapar ve dönüş yapar.

Bottom half ise interrupt context'i dışında, interrupt'lar açık durumdayken çalışır. Linux bu amaçla çeşitli mekanizmalar sunar:

- **Softirqs:** Çekirdek içinde statik olarak tanımlanmış yazılım interrupt'larıdır; özel bir context'te çalışırlar.

- **Tasklets:** Softirqs üzerine inşa edilmiş, daha basitleştirilmiş bir arayüzdür. Fonksiyonların ileride çalıştırılmak üzere planlanmasına izin verir.
- **Workqueues:** Proses context'inde çalışırlar, bu sayede gerektiğinde uyku (sleep) durumuna geçilebilir.

Donanım Soyutlaması (APIC, I/O APIC)

Çoklu işlemci (multiprocessor) sistemlerde genellikle Advanced Programmable Interrupt Controller (APIC) bulunur. Her çekirdekte local APIC (LAPIC) bulunur ve dış cihazlardan gelen interrupt'ları yönetmek için I/O APIC devreye girer. Bu yapı:

- Interrupt'ların birden fazla CPU çekirdeğine dağıtılmasına olanak tanıyarak performansı artırır.
- Interrupt yeniden eşleme (remapping) ve CPU yakınlığı (affinity) gibi gelişmiş özellikler sunar.

Softirqs ve Tasklets

- **Softirqs:** Örneğin NET_RX_SOFTIRQ (ağ alım işlemleri) gibi belirli sayıda sabit softirq vardır. Her softirq birçok kez planlanabilir, ancak çekirdek ilgili softirq'yi çalıştırdığı sırada toplu işleme yapılır.
- **Tasklets:** Softirqs üzerine kurulmuş dinamik geri çağırma (callback) mekanizmalarıdır. Aynı CPU üzerinde aynı tasklet aynı anda çalışamaz, ancak farklı CPU'larda paralel çalışabilirler.

Bu mekanizmalar, interrupt işleyicilerinin hızlı olması (top half) ve daha karmaşık veya uzun süren işlemlerin ertelenmesi (bottom half) yaklaşımını destekler, düşük interrupt gecikmesi ve dengeli sistem performansı sağlar.

Proses Adres Alanları (Process Address Spaces)

Sanal Bellek Alanları (Virtual Memory Areas - VMAs)

Her prosesin adres alanı, çekirdek içinde mm_struct isimli bir yapı ile tanımlanır. Bu yapı, Virtual Memory Areas (VMAs) denilen ve birbirleriyle benzer koruma (okuma, yazma, çalıştırma) ve benzer arka plan depolama (backing store) özelliklerini paylaşan kesintisiz sanal adres bloklarından oluşan bir bağlantılı liste veya kırmızı-siyah ağaç içerir. Örneğin, bir proseste şu VMAlar olabilir:

1. Programın text segmenti (salt okunur, çalıştırılabilir).
2. Veri (data) ve BSS segmentleri (okunur/yazılır).
3. Heap (dinamik olarak büyüyen bölüm).
4. Dosya tabanlı bellek eşlemeleri (mmap() ile).
5. Yığın (stack) segmenti.

Çekirdek, benzer koruma ve arka plan belleği paylaşan bitişik VMAları birleştirir; bazen de küçük değişiklikler olduğunda VMAları böler.

Sayfa Tabloları (Page Tables) ve Hiyerarşi

Tipik bir x86_64 sistemde, her proses kendi page tables takımına sahiptir. Bu tablolar sanal adresleri fiziksel çerçevelere eşler. Hiyerarşi beş seviyeye kadar uzayabilir:

- Page Global Directory Pointer (PGD)
- Page Upper Directory (PUD)
- Page Middle Directory (PMD)
- Page Table (PTE)

(5-seviyeli paging uzantısıyla: PGD ile PUD arasında P4D eklenir.)

Bir proses sanal bir adrese eriştiğinde, donanım bu tabloları yürüyerek (walk) fiziksel adrese ulaşır. Çekirdek, proses oluşturma sırasında bu tabloları yapılandırır, ebeveynin tablolarını kopyalar veya paylaşır; Copy-on-Write (COW) mekanizması, yazma anına kadar sayfaların fiilen kopyalanmasını geciktirir.

Copy-on-Write (COW) Mekanizması

Copy-on-Write, bir sayfanın gerçekten kopyalanmasını ilk yazma anına kadar erteleyen bir optimizasyon tekniğidir. fork() sırasında, child ve ebeveyn aynı fiziksel sayfaları paylaşır ve sayfalar salt okunur olarak işaretlenir. Ebeveyn veya child bu sayfalara yazmaya kalkışırsa, bir page fault oluşur; çekirdek, yeni bir sayfa çerçevesi ayırır, orijinal içeriği kopyalar ve yazma işlemini yapan prosesin tablosunu günceller. Bu yöntem, özellikle büyük prosesler çatalandıktan (fork) hemen sonra exec() çağırıyorsa, gereksiz kopyalamaları ciddi ölçüde azaltır.

Paylaşımlı Bellek (Shared Memory) ve Haritalamalar

Linux, proseslerin belleği paylaşması için çeşitli yöntemler destekler:

- **Anonim Paylaşımlı Bellek:** Genellikle /dev/zero gibi bir kaynakla mmap(MAP_SHARED) kullanılarak oluşturulur.
- **POSIX Paylaşımlı Bellek:** shm_open() ve mmap() ile sağlanır.
- **System V Paylaşımlı Bellek:** shmget(), shmat() vb. eski arayüzü kullanır.

Aynı fiziksel sayfaları birden fazla adres alanına eşleyerek, prosesler veriyi kopyalamadan hızlıca iletişim kurabilir.

Sanal Dosya Sistemi (Virtual File System - VFS)

VFS Soyutlaması ve Mimarisi

Virtual File System (VFS) katmanı, ext4, xfs, btrfs, NFS, CIFS gibi çeşitli dosya sistemlerine ve depolama altyapılarına tek tip bir arayüz sağlayan kritik bir soyutlamadır. Amaç, kullanıcı alanı uygulamalarının open, read, write, close gibi dosya işlemlerini, altta hangi dosya sistemi olursa olsun aynı şekilde yapabilmesidir. VFS, bunu ortak veri yapıları ve operasyonlarla başarır:

- **struct inode:** Bir dosyanın sahiplik, izinler, zaman damgaları gibi meta verisini temsil eder.
- **struct dentry:** Bir dizin girdisini ifade eder; bir dizin ile inode arasındaki bağlantıyı gösterir.
- **struct file:** Çekirdek içinde açık bir dosya tanımlayıcıyı temsil eder.

Bir proses open() çağırdığında, VFS ilgili dizin girişini (dentry) bulur, ilgili inode'u belirler ve yeni oluşturulan struct file ile bu inode'u ilişkilendirir. Bu sayede farklı dosya sistemlerinde aynı üst seviye fonksiyonlar (vfs_read(), vfs_write()) kullanılabilir.

VFS Veri Yapıları

- **inode:** Dosyanın meta verisini içerir. Her dosya sistemi kendi inode yapısını uygular; ancak VFS, struct inode içinde dosya sistemine özgü fonksiyonlara işaretçiler barındıran genel bir yapı sunar.
- **dentry:** Yol adlarını yönetir ve hız için önbelleklenir (dentry cache). Çekirdek, klasörleri sürekli parse etmek zorunda kalmadan hızlıca yol çözümleme yapabilir.
- **super_block:** Bir mount edilmiş dosya sistemini temsil eder. Bloğun cihaz bilgisini, dosya sistemi türünü ve dosya sistemi operasyonları için işaretçileri barındırır.
- **file:** Bir proses tarafından açık olan her dosya için bir struct file mevcuttur. Bu yapı dosyanın mevcut ofsetini ve f_op gibi dosya operasyonlarını tanımlayan işaretçiyi tutar.

Filesystem Kaydı (Registration) ve Mount Etme

Linux'ta her dosya sistemi türü, çekirdek başlatma sırasında veya yüklenebilir bir modül olarak VFS'ye kaydolur. Bu kayıt işleminde, dosya sisteminin:

- Superblock okuma,
- Dosya/dizin oluşturma/silme,
- Inode okuma/yazma,
- Dosya işlemleri (truncate, rename vb.)

gibi fonksiyonları tanımlanır. Kullanıcı mount komutuyla (veya eş değer sistem çağrısıyla) bir dosya sistemini mount ettiğinde, çekirdek dosya sisteminin mount fonksiyonunu çağırır ve bu fonksiyon

bir super_block döndürür. VFS, bu super_block'u küresel mount ağacına ekleyerek dosyaları belirtilen mount noktasında erişilebilir hale getirir [16].

Gerçek Dosya Sistemler (Filesystem) ile Etkileşim

VFS soyutlamasının altında birçok gerçek dosya sistemi vardır:

- **Yerel Dosya Sistemleri:** ext4, xfs, btrfs, ReiserFS vb.
- **Ağ Dosya Sistemleri:** Uzaktaki depolamaya yönelik NFS, CIFS vb.
- **Özel Amaçlı Dosya Sistemleri:** procfs (proses bilgisi), sysfs (sistem bilgisi), tmpfs (bellek tabanlı) vb.

Kullanıcı uygulamaları hangi dosya sisteminin kullanıldığını bilmeden aynı POSIX sistem çağrılarını (open, read, write vb.) kullanır. VFS ve dosya sistemi sürücüsü (driver), bu çağrıları ilgili disk veya ağ işlemlerine dönüştürür [13].

Linux Çekirdeğine Yeni Bir Sistem Çağrısı Nasıl Eklenir?

Linux çekirdeğine yeni bir sistem çağrısı eklemek, çekirdek geliştiricileri ve bu alana ilgi duyanlar için yaygın bir görevdir. Çekirdeğin işlevselliğini genişletmek için bu yöntem kullanılır. Bu başlıkta, Linux çekirdeğine nasıl yeni bir sistem çağrısı ekleneceği adım adım anlatılmıştır. Ayrıca Linux çekirdeği, C programlama dili ve çekirdeği derleyip test etme konusunda temel bilgilere sahip olunduğu varsayılmıştır.

Gereksinimler

Devam etmeden önce aşağıdakilerin hazır olduğundan emin olunmalıdır.

- **Çekirdek Kaynak Kodu:** Linux çekirdek kaynak kodu edinilmelidir. kernel.org adresinden indirebilir.
- **Geliştirme Ortamı:** gcc, make, ncurses, openssl

Adımlar

Adım 1: Sistem Çağrısı Tanımı

Öncelikle, yeni sistem çağrısının hangi işlevselliği sağlayacağına karar verilmelidir. Bu başlıkta, **"Hello, World!"** mesajını çekirdek günlüğüne yazdıran basit bir sistem çağrısı olan sys_hello gerçekleştirilecektir.

Adım 2: Çekirdek Kaynak Kodunu Değiştirilmesi

1. Sistem Çağrısı Fonksiyonunu Tanımlama

Aşağıda Linux kerneli için yazılmış dummy bir syscall kodu görülmektedir. Bu syscall'u derleyip yeni modifiye edilmiş kernele boot ettikten sonra userspace'den ilgili sistem cağrısına erişebilecek

duruma gelecektir. Burada, SYSCALL_DEFINE0, sıfır argümanlı sistem çağrılarını tanımlamak için kullanılan bir makrodur. Gerekirse argüman sayısı değiştirilir (örneğin, SYSCALL_DEFINE1, SYSCALL_DEFINE2, vb.).

```
arch/x86/syscalls/syscall_32.tbl    | 1 +
arch/x86/syscalls/syscall_64.tbl    | 1 +
include/linux/syscalls.h            | 1 +
include/uapi/asm-generic/unistd.h   | 4 +++-
kernel/Makefile                     | 3 ++-
kernel/foo_core.c                   | 18 ++++++++++++++++++
kernel/foo_module.c                 | 38 ++++++++++++++++++++++++++++++++++++++
7 files changed, 64 insertions(+), 2 deletions(-)
create mode 100644 kernel/foo_core.c
create mode 100644 kernel/foo_module.c

diff --git a/arch/x86/syscalls/syscall_32.tbl b/arch/x86/syscalls/syscall_32.tbl
index 028b781..3509e3d 100644
--- a/arch/x86/syscalls/syscall_32.tbl
+++ b/arch/x86/syscalls/syscall_32.tbl
@@ -363,3 +363,4 @@
354  i386    seccomp         sys_seccomp
355  i386    getrandom       sys_getrandom
356  i386    memfd_create    sys_memfd_create
+357  i386    foo             sys_foo

diff --git a/arch/x86/syscalls/syscall_64.tbl b/arch/x86/syscalls/syscall_64.tbl
index 35dd922..ce1ecb8 100644
--- a/arch/x86/syscalls/syscall_64.tbl
+++ b/arch/x86/syscalls/syscall_64.tbl
@@ -327,6 +327,7 @@
318     common  getrandom       sys_getrandom
319     common  memfd_create    sys_memfd_create
320     common  kexec_file_load sys_kexec_file_load
+321     common  foo             sys_foo
#
# x32-specific system call numbers start at 512 to avoid cache impact
diff --git a/include/linux/syscalls.h b/include/linux/syscalls.h
index 0f86d85..ae8a160 100644
--- a/include/linux/syscalls.h
+++ b/include/linux/syscalls.h
@@ -875,5 +875,6 @@ asmlinkage long sys_seccomp(unsigned int op, unsigned int flags,
```

```
const char __user *uargs);
asmlinkage long sys_getrandom(char __user *buf, size_t count,
unsigned int flags);
+asmlinkage long sys_foo(int high_id, int low_id);

#endif
diff --git a/include/uapi/asm-generic/unistd.h b/include/uapi/asm-generic/unistd.h
index 11d11bc..8f947c1 100644
--- a/include/uapi/asm-generic/unistd.h
+++ b/include/uapi/asm-generic/unistd.h
@@ -705,9 +705,11 @@ __SYSCALL(__NR_seccomp, sys_seccomp)
__SYSCALL(__NR_getrandom, sys_getrandom)
#define __NR_memfd_create 279
__SYSCALL(__NR_memfd_create, sys_memfd_create)
+#define __NR_foo 280
+__SYSCALL(__NR_foo, sys_foo)

#undef __NR_syscalls
-#define __NR_syscalls 280
+#define __NR_syscalls 281

/*
 * All syscalls below here should go away really,
diff --git a/kernel/Makefile b/kernel/Makefile
index dc5c775..c399241 100644
--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ -9,7 +9,7 @@ obj-y = fork.o exec_domain.o panic.o \
extable.o params.o \
kthread.o sys_ni.o nsproxy.o \
notifier.o ksysfs.o cred.o reboot.o \
-       async.o range.o groups.o smpboot.o
+       async.o range.o groups.o smpboot.o foo_core.o

ifdef CONFIG_FUNCTION_TRACER
# Do not trace debug files and internal ftrace files
@@ -27,6 +27,7 @@ obj-y += printk/
obj-y += irq/
obj-y += rcu/
+obj-m += foo_module.o
obj-$(CONFIG_CHECKPOINT_RESTORE) += kcmp.o
obj-$(CONFIG_FREEZER) += freezer.o
obj-$(CONFIG_PROFILING) += profile.o
```

```
diff --git a/kernel/foo_core.c b/kernel/foo_core.c
new file mode 100644
index 0000000..aaf4033
--- /dev/null
+++ b/kernel/foo_core.c
@@ -0,0 +1,18 @@
+#include <linux/syscalls.h>
+#include <linux/export.h>
+
+typedef long (*foo_func_t)(int high_id, int low_id);
+
+long foo_dummy_syscall(int high_id, int low_id)
+{
+   return -ENOSYS;
+}
+EXPORT_SYMBOL_GPL(foo_dummy_syscall);
+
+foo_func_t foo_syscall = &foo_dummy_syscall;
+EXPORT_SYMBOL_GPL(foo_syscall);
+
+SYSCALL_DEFINE2(foo, int, high_id, int, low_id)
+{
+   return foo_syscall(high_id, low_id);
+}
diff --git a/kernel/foo_module.c b/kernel/foo_module.c
new file mode 100644
index 0000000..97f3bed
--- /dev/null
+++ b/kernel/foo_module.c
@@ -0,0 +1,38 @@
+#include <linux/module.h>
+#include <linux/init.h>
+#include <linux/types.h>
+
+#define foo_ID(0x13fff248d7be)
+
+typedef long (*foo_func_t)(int high_id, int low_id);
+
+extern long foo_dummy_syscall(int high_id, int low_id);
+extern foo_func_t foo_syscall;
+
+static long foo_syscall_stub(int high_id, int low_id)
+{
```

```
+u64 id;
+
+   id = (unsigned int)low_id; /* prevent sign extension */
+   id |= ((u64)high_id << 32);
+
+   return id == foo_ID ? 0 : -EINVAL;
+}
+
+static int __init foo_init(void)
+{
+   foo_syscall = &foo_syscall_stub;
+   return 0;
+}
+
+static void __exit foo_exit(void)
+{
+   foo_syscall = &foo_dummy_syscall;
+}
+
+module_init(foo_init);
+module_exit(foo_exit);
+
+MODULE_AUTHOR("Cihangir Akturk");
+MODULE_LICENSE("GPL");
+MODULE_DESCRIPTION("foo syscall implementation");
```

2. Sistem Çağrı Numarası Atama

Sistem çağrısına benzersiz bir numara atamak için arch/x86/entry/syscalls/syscall_64.tbl dosyası (veya mimari için eşdeğeri) değiştirilir:

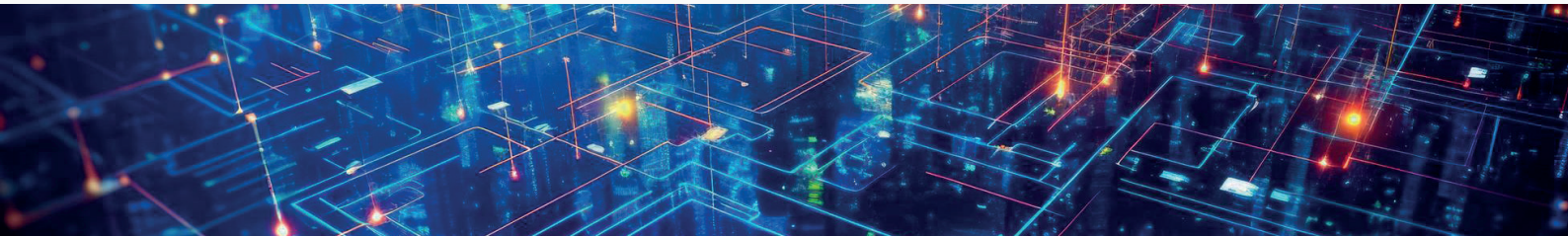
```
335   common   hello   sys_hello
```

Burada, 335 yeni sistem çağrısı numarasıdır (benzersiz olduğundan emin olun), common mimariler arası uyumluluğu belirtir, hello sistem çağrısının adı ve sys_hello ise tanımlanan fonksiyondur.

3. Sistem Çağrısını Bildirme

Sistem çağrısı include/linux/syscalls.h dosyasına aşağıdaki gibi bildirilir:

```
asmlinkage   long sys_hello(void);
```



Adım 3: Çekirdeği Derleme ve Kurma

1. Çekirdeği Yapılandırma

Çekirdeği yapılandırmak için aşağıdaki komut çalıştırılır:

```
make menuconfig
```

Sisteminize uygun tüm gerekli seçeneklerin etkinleştirildiğinden emin olun.

2. Çekirdeği Derleyin

Çekirdeği ve modülleri aşağıdaki gibi derlenir:

```
make -j$(nproc) make modules_install make install
```

3. Yeni Çekirdekle Yeniden Başlatma

Sistemi yeniden başlatılır ve önyükleyiciden yeni çekirdek seçilir.

Adım 4: Yeni Sistem Çağrısını Test Etme

Yeni çekirdekle başlattıktan sonra, sistem çağrısı test edilir. Sistem çağrısını çağırmak için küçük bir kullanıcı alanı programı oluşturulur. Örneğin:

```
#include <unistd.h> #include <sys/syscall.h> #include <stdio.h>

#define SYS_hello 335 // Sistem çağrı numaranızı buraya yazın

int main(void)
{
long res = syscall(SYS_foo);
if (res)
perror("failed call foo:\n");
printf("foo returned %ld\n", res);
}
```

Bu programı derleme ve çalıştırma:

```
gcc -o test_hello test_hello.c ./test_hello
```

Çekirdek günlüğünde **"Hello, World!"** mesajı kontrol edilir:

```
dmesg | tail
```

Adım 5: Hata Ayıklama ve Temizlik

Sistem çağrısı beklenilen gibi çalışmazsa, aşağıdaki hata ayıklama teknikleri kullanılır:

Günlükler Kontrol Edilir: Hatalar veya uyarılar için **"dmesg"** komutu kullanılarak çekirdek günlükleri incelenir [18], [19].

Hata Ayıklama Araçları: Çekirdek hatalarını ayıklamak için **"gdb"** veya **"kgdb"** gibi araçlar kullanılır.

Linux çekirdeğine yeni bir sistem çağrısı eklemek, çekirdek kaynak kodunu değiştirmeyi, çekirdeği derlemeyi ve değişiklikleri test etmeyi gerektirir. Buradaki örnek basit olsa da daha karmaşık sistem çağrıları için benzer adımlar izlenebilir. Kodlama standartlarına uyma ve değişiklikleri kapsamlı bir şekilde test etme unutulmamalıdır.

Bölüm Özeti

Bu bölümde, Linux çekirdeğinin temel alt sistemlerine odaklanıldı. Her alt sistem, Linux'u modern bir işletim sistemi yapan işlevselliği, performansı ve genişletilebilirliği sağlamak adına kritik öneme sahiptir. Bir prosesin fork() ile başlayıp bellek yerleşimi ve scheduling aşamalarına, interrupt yönetiminden dosya sistemleri soyutlamasına kadar olan yolculuğu, farklı iş yükleri ve donanımlar için titizlikle tasarlanmış mekanizmalar içerir.

Proses yönetimi, çoklu görevlilik ve eş zamanlı çalışmayı mümkün kılar; proses oluşturma, iş parçacığı yönetimi gibi temel bileşenleri kapsar. Memory management, RAM kullanımını paging, swapping ve gelişmiş tahsis stratejileriyle optimize eder. Scheduling (CFS ve gerçek zamanlı politikalar dâhil), CPU zamanını adil ve öngörülebilir şekilde dağıtır. Interrupt handling, donanım ve yazılım sinyallerine hızlı tepki verebilmeyi sağlar. Top half ve bottom half konseptleri, hız ve esneklik dengesini kurar. Proses adres alanları ve VMAlar, fiziksel belleğin nasıl düzenlendiğini ve paylaşıldığını ince bir şekilde yönetir. VFS katmanı, dosya işlemlerini tek bir arayüzde birleştirerek farklı dosya sistemlerini kullanıcıya şeffaf hale getirir.

Bu alt sistemleri anlamak, yalnızca çekirdek geliştiricileri için değil, aynı zamanda sistem programcıları, performans mühendisleri ve düşük seviyeli sistem davranışlarını teşhis etmeye çalışanlar için de çok önemlidir. Açık kaynak dünyasında Linux çekirdeği, yeni donanım yeteneklerini kucaklamaya ve iç mimarisini her geçen gün rafine etmeye devam ederek güçlü ve modern bir platform olarak varlığını sürdürmektedir.

Bölüm 3
Buildroot ile Minimal Linux Dağıtımı Oluşturma

Aşağıdaki buildroot ile Linux dağıtımı oluşturma adımları Pardus üzerinde gerçekleştirilmiştir.

Hazırlık Aşaması

Öncelikle gerekli paketleri aşağıdaki gibi yüklenir:

```
sudo apt install bc binutils bison dwarves flex gcc \
        git gnupg2 gzip libelf-dev libncurses5-dev gawk \
        libssl-dev make openssl perl-base tar xz-utils \
        python3 bzip2 findutils cpio
```

Şimdi buildroot kodu git kullanarak indirelir:

```
# --depth=1 sadece son commiti çekmek için kullanılır.
git clone https://git.buildroot.net/buildroot.git --depth=1
```

Kodun Yapılandırılması

Buildroot yapılandırmak için make defconfig komutu kullanılır. Bu komut varsayılan ayarlara göre yapılandırma yapacaktır.

```
make defconfig
```

Ardından make menuconfig komutunu kullanarak özelleştirmeler yapılır veya ".config" dosyası elle düzenlenir.

```
# Terminal arayüzü ile yapılandırma
make menuconfig
```

```
# Elle yapılandırma
nano .config
```

Derleme Aşaması

Buildroot yapılandırması tamamladıktan sonra make komutu ile derleme işlemi başlatılır.

```
make -j`nproc`
```

Derleme işlemi tamamlandıktan sonra “output/images” içerisinde derlenmiş dosyalar bulunur.

Paket Eklenmesi

Buildroot içerisine yeni bir paket eklemek için öncelikle packages/Config.in içerisine istenilen bir menünün altına aşağıdaki gibi bir ekleme yapılır:

```
...
menu “Misc”
source “package/hello/Config.in”
endmenu
```

Ardından aşağıdaki içeriğe sahip “packages/hello/Config.in” dosyası oluşturulur.

```
config BR2_PACKAGE_HELLOWORLD
bool “helloworld”
default y
help
Hello world component.
```

Sonraki adımda “buildroot” için derleme talimatı dosyası oluşturulur.

```
HELLO_VERSION = 1.0
HELLO_SITE = ./package/hello
HELLO_SITE_METHOD = local

define HELLO_BUILD_CMDS
$(MAKE) CC=”$(TARGET_CC)” LD=”$(TARGET_LD)” -C $(@D)
endef

define HELLO_INSTALL_TARGET_CMDS
$(MAKE) install DESTDIR=”$(TARGET_DIR)” -C $(@D)
endef

$(eval $(generic-package))
```

Yukarıdaki dosyalar oluşturulduktan sonra “makefile” dosyası ve “main.c” dosyaları oluşturulur.

“Helloworld.mk” dosyası:

```
build:
$(CC) $(CFLAGS) main.c -o main

install:
install -Dm755 main $(DESTDIR)/usr/bin/hello
```

“main.c dosyası”:

```
#include <stdio.h>
int main(int argc, char** argv){
printf(“Hello World\n”);
return 0;
}
```

Bu işlemler ile paketler hazırlanmış olur. Buildroot menüsü içerisinde “misc/helloworld” üzerinden ya da ilgili konfigürasyon (config) dosyasından **“BR2_PACKAGE_HELLOWORLD=y”** satırı ayarlanarak etkinleştirme sağlanır.

Sıfırdan Minimal Dağıtım Hazırlama

Bu başlıkta kaynak koddan derleyerek sıfırdan basit dağıtım oluşturma anlatılmıştır. Anlatımı Pardus üzerinden yapılacaktır.

Hazırlık Aşaması

Öncelikle gerekli olan paketler aşağıdaki gibi yüklenmelidir:

```
sudo apt install bc binutils bison dwarves flex gcc \
        git gnupg2 gzip libelf-dev libncurses5-dev gawk \
        libssl-dev make openssl perl-base tar xz-utils \
        python3 bzip2 findutils cpio
```

Test için sanal makina olarak qemu kullanılacaktır.

```
sudo apt install qemu-kvm
```

Ardından gerekli kaynak kodların temin edilmesi gerekmektedir.

- Glibc: https://www.gnu.org/software/libc/
- Busybox: https://www.busybox.net/

- Linux: https://kernel.org

Derleme Aşaması

Öncelikle gerekli olan paketler aşağıdaki gibi yüklenmelidir:

İndirilen kaynak kodun tarball dosyalarının açıp derlenmesi gerekmektedir. Bu işlem sonucunda dağıtımın ana şablonu oluşturulmuş olacaktır.

Bu başlık boyunca kök dizin şablonu olarak **/home/pardus/rootfs** dizini varsayılmıştır.

Glibc

Öncelikle arşiv bir dizine çıkarılır:

```
tar -xf glibc-2.40.tar.xz
```

Ardından derleme işlemini başlatılır. Bu işlem işlemci gücüne bağlı olarak tahmini 30dk civarı sürebilir.

```
make -j`nproc` -C build
```

Son olarak derleme işleminden sonra sistemi oluşturacak şablon dizinine kurulum yapılır.

```
make -j`nproc` install DESTDIR=/home/pardus/rootfs -C build
```

Kök dizin şablonu aşağıdakine benzer olmalıdır.

```
$ tree -L1 /home/pardus/rootfs

/home/pardus/rootfs/
|-- bin
|-- etc
|-- include
|-- lib
|-- libexec
|-- sbin
|-- share
`-- var
```

Bizim sistemimiz tarafından derlenen dosyalar /lib64/ld-linux-x86-64.so.2 dosyalı ile çalışmaktadır. Fakat görüldüğü üzere kök dizin şablonunda /lib64 bulunmuyor. Bu durum ldd komutu ile kontrol edilebilir.

```
$ ldd /bin/bash
        linux-vdso.so.1 (0x00007f6261a60000)
        libtinfo.so.6 => /lib/x86_64-linux-gnu/libtinfo.so.6 (0x00007f62618e2000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f6261701000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f6261a62000)
```

Bunun için /lib dizinini /lib64 olarak sembolik link atılabilir.

```
ln -s lib /home/pardus/rootfs/lib64
```

Busybox

Öncelikle arşiv bir dizine çıkarılır:

```
tar -xf busybox-1.37.0.tar.bz2
```

Ardından kaynak kodun olduğu dizine gidilerek kod yapılandırılır ve derlenir.

```
cd busybox-1.37.0
make defconfig
```

Ardından derleme işlemini başlatılır. Bu işlem işlemci gücüne bağlı olarak 5dk civarı sürebilir.

```
make -j`nproc`
```

Son olarak derlenen busybox dosyası kök dizin şablonuna kopyalanır.

```
install -m755 busybox /home/pardus/rootfs/bin/busybox
```

Daha sonra busybox komutlarının kök dizin şablonuna kurulması sağlanır.

```
sudo chroot /home/pardus/rootfs /bin/busybox --install -s /bin
```

Linux

Öncelikle arşivi bir dizine çıkarılır:

```
tar -xf linux-6.13.tar.xz
```

Ardından kaynak kodun olduğu dizine gelinir ve kod yapılandırılır.

```
cd linux-6.13
```

varsayılan config ayarlamak için

```
make defconfig
```

elle ayarlamak için

```
make menuconfig
```

Ardından kod derlenir.

```
make bzImage -j`nproc`
```

Bu işlem işlemci gücüne ve yapılandırmaya bağlı olarak 2-4 saat civarı sürebilir.

Son olarak çekirdeğin vmlinuz dosyası kopyalanır.

```
install arch/x86_64/boot/bzImage /home/pardus/linux
```

Init Dosyası

Linux çekirdeği açılışta /init dosyasını çalıştırır. Bu dosya geri kalan tüm herşeyin başlatılmasını sağlar.

Bu dosyayı derlenmiş olan busybox yardımı ile çalıştırarak basit bir kabuk ortamı oluşturulması gerekmektedir. Bunun için kök dizin şablonuna aşağıdaki gibi bir dosya oluşturulur.

#!/bin/sh

Gerekli dizinleri oluşturmak için

```
mkdir -p /dev /proc /sys
```

Sistem dizinlerini bağlamak için

```
mount -t devtmpfs none /dev
mount -t proc none /proc
mount -t sysfs none /sys
```

Bu aşamada istenilen komutlar çalıştırılabilir.

Ekranı temizleyip bir mesaj yazmak için

```
clear
echo “Hello World”
```

Kabuk başlatmak için

```
exec /bin/sh
```

Not: Eğer init dosyası sonlanırsa çekirdek çalışacak başka bir şey bulamadığı için hata verir. Bu sebeple init dosyasının kapatılmaması gerekmektedir.

Son olarak, init dosyasının çalıştırılabilir hale getirmesi gerekmektedir.

```
chmod 755 /home/pardus/rootfs/init
```

İsteğe Bağlı Aşamalar

Bu aşamada yapılan minimal dağıtım için bazı isteğe bağlı aşamalardan söz edilmiştir.

Ağ Erişiminin Sağlanması

Oluşturulacak olan minimal sistemin ağ erişimi busybox yardımı ile sağlanabilir. Bunun için öncelikle udhcpc betiği hazırlanması gerekmektedir. Aşağıdaki betiği kök dizin şablonunda "/etc/udhcpc.script" dosyası içine yazmamız gerekmektedir:

#!/bin/sh

Ip adresi ayarla

```
ip addr add $ip/$mask dev $interface
```

Varsa route ekle

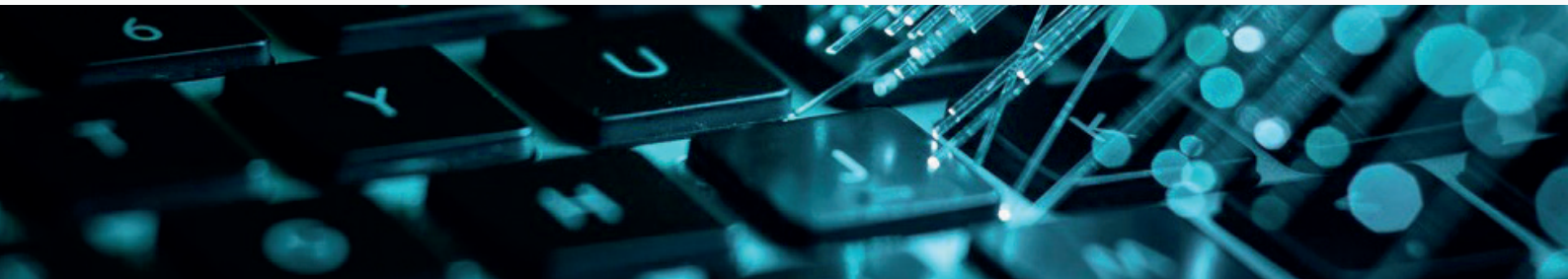
```
if [ "$router" ]; then
ip route add default via $router dev $interface
fi
```

Dns ayarla

```
echo "nameserver 8.8.8.8" > /etc/resolv.conf
echo "nameserver 8.8.4.4" >> /etc/resolv.conf
```

Hazırlanan dosya çalıştırılabilir hale getirilmelidir:

```
chmod 755 /home/pardus/rootfs/etc/udhcpc.script
```



Ardından init içerisine aşağıdaki komutları eklemeniz gerekmektedir:

Aygıtı başlat

```
ip link set up eth0
```

Dhcp client başlat

```
udhcpc -s /etc/udhcpc.script -i eth0
```

Kullanılmayan Dosyaların Silinmesi

Boyutu küçültmek için sistemin çalışması için doğrudan gerekli olmayan ".a" uzantılı static kütüphane dosyaları ve "header" dosyaları silinebilir.

```
find /home/pardus/rootfs -iname "*.a" -exec rm -f {} \;
```

İlave olarak, aşağıdaki şekilde "Glibc" tarafından gelen fakat sistemi çalışması için doğrudan gerek duyulmayan dosyalar silinir.

header dosyaları

```
rm -rf /home/pardus/rootfs/include
```

dil dosyaları

```
rm -rf /home/pardus/rootfs/share
```

Initramfs Dosyasının Oluşturulması

Sistemin açılabilmesi için kök dizin şablonunun belleğe yüklenmesi ve çalıştırılması gerekmektedir. Bu sebeple, "initramfs" (başlangıç ramdiski) oluşturulur.

Kök dizin şablonunun olduğu dizine gelinir ve aşağıdaki gibi dosyalar oluşturulur.

```
cd /home/pardus/rootfs
find . | cpio -R root:root -H newc -o | gzip > ../initramfs
```

Test Aşaması

Qemu kullanarak çekirdeği ve initramfs dosyasını test etmek için aşağıdaki komut kullanılır:

```
qemu-system-x86_64 \
        --enable-kvm \
        -m 512M \
        -kernel linux \
        -initrd initramfs \
        -append “quiet console=ttyS0” \
        -nographic
```

Not: Sistemi kapatak için “poweroff -f” komutu kullanılabilir.

Iso Dosyası Haline Getirme

Sistemin iso haline getirilip paketlenmesi için grub kullanılır. Öncelikle aşağıdaki gibi gereken paketler kurulur.

```
sudo apt install grub-pc-bin grub-efi-amd64-bin xorriso mtools
```

Ardından bir çalışma dizini oluşturulur ve çekirdek ile initramfs dosyası içine kopyalanır.

```
mkdir -p /home/pardus/iso
cp -f /home/pardus/linux /home/pardus/iso/
cp -f /home/pardus/initramfs /home/pardus/iso/
```

Grub yapılandırması boot/grub/grub.cfg içine aşağıdaki gibi yazılır:

```
insmod all_video
menuentry “My Linux Distro” {
        linux /linux quiet
        initrd /initramfs
}
```

Iso taslağı aşağıdaki gibi gözükmelidir:

```
$ tree /home/pardus/iso
/home/pardus/iso/
|-- boot
|   `-- grub
|       `-- grub.cfg
|-- initramfs
`-- linux
```

Son olarak iso dosyası haline getirilir.

```
grub-mkrescue -o mylinux.iso /home/pardus/iso/
```

Debian Tabanlı Dağıtım Hazırlama

Aşağıdaki Debian tabanlı basit bir dağıtım oluşturma adımları Pardus 23 sunucu sürümü üzerinde ele alınmıştır.

Gerekli Paketlerin Kurulması

Gerekli olan paketler aşağıdaki gibi yüklenir:

```
sudo apt install aptly live-build
```

Depoların İmzalanması İçin Anahtar Oluşturulması

Yansıladığımız ve oluşturduğumuz yerel deponun imzalanması için aşağıdaki komut çalıştırılıp yönergeler tamamlanarak GPG anahtarı oluşturulmalıdır.

```
gpg --full-generate-key
```

Oluşturulan bu imzanın dağıtımda kullanılabilmesi için ilgili sunucuda aşağıdaki komut ile GPG açık anahtarı alınmaktadır:

```
gpg --armor --export key-id > yereldepo-publickey.asc
```

Elde edilen dosya, kullanılacak sistemin **/etc/apt/trusted.gpg.d/** klasörüne eklenmektedir. Bu sayede dağıtım, ilgili imzaya güvenmekte ve bu depodan gerçekleştirilen paket yöneticisi işlemleri güvenli ve sorunsuz bir şekilde yürütülmektedir.

Depo Yansılanması ve Yerel Depo Oluşturulması

Debian deposu yansılanarak bu yansı depo üzerinde yönetim sağlanmaktadır. Bu sayede, Debian deposuna güncelleme geldiğinde, söz konusu güncellemeler kontrollü bir şekilde yansı depoya entegre edilebilmektedir.

Oluşturulan yansı depoya ek olarak, bir yerel depo oluşturulmakta ve bu yerel depo üzerinden istenilen ek paketler sunulabilmektedir.

Depo işlemleri için aptly aracı kullanılmıştır. Aptly; depo yansılamak, yerel depo oluşturmak ve

yönetmek için kullanılan açık kaynak kodlu bir araçtır.

Debian deposunu yansılamaya başlamadan önce, Debian anahtarının güvenilir anahtarlara eklenmiş olması gerekir. Bunu yapmak için aşağıdaki komut çalıştırılır.

```
gpg --no-default-keyring --keyring /usr/share/keyrings/debian-archive-keyring.gpg --export | gpg --no-default-keyring --keyring trustedkeys.gpg --import
```

Debian bookworm deposunu yansılamak için aşağıdaki yansı ekleme komutları çalıştırılır. Bu komutta amd64, i386 ve arm64 mimarisindeki paketlerin kaynak kodlarıyla yansılanmak istendiği belirtilmiştir.

```
aptly mirror create -architectures="amd64,i386,arm64" -with-sources=true -with-udebs=true debian-bookworm-main http://deb.debian.org/debian bookworm main

aptly mirror create -architectures="amd64,i386,arm64" -with-sources=true debian-bookworm-contrib http://deb.debian.org/debian bookworm contrib

aptly mirror create -architectures="amd64,i386,arm64" -with-sources=true debian-bookworm-non-free http://deb.debian.org/debian bookworm non-free

aptly mirror create -architectures="amd64,i386,arm64" -with-sources=true debian-bookworm-non-free-firmware http://deb.debian.org/debian bookworm non-free-firmware

aptly mirror create -architectures="amd64,i386,arm64" -with-sources=true debian-bookworm-security-main http://deb.debian.org/debian-security bookworm-security/updates main

aptly mirror create -architectures="amd64,i386,arm64" -with-sources=true debian-bookworm-security-contrib http://deb.debian.org/debian-security bookworm-security/updates contrib

aptly mirror create -architectures="amd64,i386,arm64" -with-sources=true debian-bookworm-security-non-free http://deb.debian.org/debian-security bookworm-security/updates non-free

aptly mirror create -architectures="amd64,i386,arm64" -with-sources=true debian-bookworm-security-non-free-firmware http://deb.debian.org/debian-security bookworm-security/updates non-free-firmware
```

Oluşturulan yansıların tümünü güncellemek için aşağıdaki komut kullanılabilir.

```
for i in $(aptly mirror list -raw); do aptly mirror update $i; done
```

Yansılanılan depoların güncelleme işlemleri tamamlandıktan sonra aşağıdaki komutla snapshot oluşturulabilir.

```
aptly snapshot create snapshot1-debian-bookworm-main from mirror debian-bookworm-main

aptly snapshot create snapshot1-debian-bookworm-contrib from mirror debian-bookworm-contrib

aptly snapshot create snapshot1-debian-bookworm-non-free from mirror debian-bookworm-non-free

aptly snapshot create snapshot1-debian-bookworm-non-free-firmware from mirror debian-bookworm-non-free-firmware
```

Oluşturulan snapshotlar aşağıdaki komutla yayına alınabilir.

```
aptly publish snapshot -distribution=bookworm -architectures="amd64,arm64,i386,source" -component=main,contrib,non-free,non-free-firmware -origin="Debian" -label="Bookworm" snapshot1-debian-bookworm-main snapshot1-debian-bookworm-contrib snapshot1-debian-bookworm-non-free snapshot1-debian-bookworm-non-free-firmware debian
```

İlk yayına alma işleminden sonra oluşturulan yeni snapshotlara aşağıda örnek olarak verilen komut ile geçiş yapılabilir.

```
aptly publish switch -component=main,contrib,non-free,non-free-firmware bookworm debian snapshot2-debian-bookworm-main snapshot2-debian-bookworm-contrib snapshot2-debian-bookworm-non-free snapshot2-debian-bookworm-non-free-firmware
```

Yansı depo işlemi tamamlandıktan sonra yerel depo oluşturmak için aşağıdaki komut kullanılabilir.

```
aptly repo create -distribution="yirmiuc" -architectures="amd64,i386,source,arm64" -component="-main" -comment="Pardus 23 Main" pardus-yirmiuc-main

aptly repo create -distribution="yirmiuc" -architectures="amd64,i386,source,arm64" -component="-contrib" -comment="Pardus 23 Contrib" pardus-yirmiuc-contrib

aptly repo create -distribution="yirmiuc" -architectures="amd64,i386,source,arm64" -component="-non-free" -comment="Pardus 23 Non-Free" pardus-yirmiuc-non-free

aptly repo create -distribution="yirmiuc" -architectures="amd64,i386,source,arm64" -component="-non-free-firmware" -comment="Pardus 23 Non-Free-Firmware" pardus-yirmiuc-non-free-firmware
```

Bu komutlarla main, contrib, non-free, non-free-firmware bileşenlerine ve belirtilen mimarilere sahip yirmiuc yerel deposu oluşturulmuştur.

Oluşturulan bu yerel depoya paket ekleme yaparken hangi bileşene hangi paketin ekleneceği aşağıdaki gibi kısaca özetlenebilir.

- main bileşen, tamamen özgür ve bağımsız paketleri içermektedir.
- contrib bileşen, özgür ama özgür olmayan yazılımlara bağımlı paketleri içermektedir.
- non-free bileşen, kapalı kaynak veya kısıtlı lisanslı yazılım paketlerini içermektedir.
- non-free-firmware bileşen, özgür olmayan donanım paketlerini içermektedir.

Örneğin elimizde yansı debiandan çatallanıp dağıtımımıza göre özelleştirilmiş base-files paketi ve harici google-chrome paketi olsun. Bu paketler aşağıdaki komutta belirtildiği gibi yerel depomuza eklenebilir.

```
aptly repo add pardus-yirmiuc-main /dizin-yolu/base-files/
aptly repo add pardus-yirmiuc-non-free /dizin-yolu/google-chrome/
```

Bu paketleri ekledikten sonra yerel depomuzu yayına almak için aşağıdaki komut kullanılır.

```
aptly publish repo -component=main,contrib,non-free,non-free-firmware -architectures="am-
d64,i386,source" -origin="Pardus" -label="Yirmiuc" -distribution=yirmiuc pardus-yirmiuc-main
pardus-yirmiuc-contrib pardus-yirmiuc-non-free pardus-yirmiuc-non-free-firmware pardus
```

İlk yayına alma işleminden sonra yerel depoda yaptığımız değişiklikler aşağıdaki komut ile tekrardan yayına alınabilir.

```
aptly publish update yirmiuc pardus
```

Depoların Sunulması

Depoları sunmak için apache, nginx gibi hizmetler kullanılabilir. Aptly üzerinden kolayca yayına almak için aptly serve komutu yeterlidir. Bu komut çalıştırıldığında depo adresleri aşağıdaki gibi verilmektedir.

```
deb http://ip-adresiniz:8080/debian bookworm contrib main non-free non-free-firmware
deb-src http://ip-adresiniz:8080/debian bookworm contrib main non-free non-free-firmware
deb http://ip-adresiniz:8080/pardus yirmiuc contrib main non-free non-free-firmware
deb-src http://ip-adresiniz:8080/pardus yirmiuc contrib main non-free non-free-firmware
```

Bu adresler dağıtımın temel adresleridir.

ISO Oluşturulması

ISO oluşturmak için live-build aracı kullanıldı. live-build; debian tabanlı özelleştirilmiş canlı ISO dosyaları oluşturmak için kullanılan bir araçtır. Debian'ın resmi canlı sürümleri de bu araç ile üretilir.

Aşağıdaki gibi çalışılacak bir klasör oluşturulup burada live-build özelleştirmeleri isteğimize göre yapılabilir.

```
mkdir ~/live-build
cd ~/live-build
lb config
```

Oluşturulan taslak üzerinde live-build dokümantasyonuna göre istenilen özelleştirmeler yapıldıktan sonra lb build komutu ile ISO build işlemi gerçekleştirilebilir.

Bölüm 4
UNIX & GNU/Linux Masaüstü Ortamları ve Gelişim Süreci

UNIX türevi sistemlerde grafiksel kullanıcı arayüzlerinin (GUI) ortaya çıkışı, kullanıcı deneyiminin iyileştirilmesi, sistem kaynaklarının etkin kullanımı ve çoklu kullanıcı/çoklu-görev kavramlarının desteklenmesi gibi etkenlerle şekillenmiştir. Özellikle 1984 yılı civarında MIT Laboratuvarlarında geliştirilmeye başlanan X Window System, ilk kez dağıtık (network transparent) pencereleme sistemi yaklaşımını benimseyerek, UNIX benzeri sistemlerde GUI kavramının temellerini atmıştır. Böylece X Window ile başlamış olan bu süreç günümüzde kullanılan modern masaüstü ortamlarına evrilmiştir.

X Window System Nedir?

X Window System (kısaca X veya X11), UNIX türevi işletim sistemlerinde grafiksel kullanıcı arayüzlerini (GUI) sağlamak için kullanılan bir sistemdir. Başlangıçta "X" olarak adlandırılan bu sistem, X10 sürümünden evrilerek 1987'de piyasaya sürülen X11 ile modern UNIX sistemlerinde temel pencereleme sistemi haline gelmiştir. X11'in tanıtımı, GUI'nin hem yerel hem de uzak sistemlerde kullanılmasına olanak sağlayarak akademik kurumlar ve araştırma merkezlerinde yoğun şekilde benimsenmiştir. Zaman içinde, birçok akademik makale, teknik rapor ve resmi dokümantasyon, X'in gelişim sürecini ve evrimini belgelemeye başlamış; bu durum X'in, UNIX-türevi sistemlerde grafiksel arayüz tasarımında mihenk taşı olarak kabul edilmesine yol açmıştır.

Teknik olarak, X'in geliştirilme felsefesi; "basitlik, genişletilebilirlik ve ağ üzerinden şeffaf iletişim" prensipleri üzerine kurulmuştur. Bu özellikler, X'in ilk geliştirildiği ortamın ötesinde farklı donanım ve yazılım mimarileri üzerinde çalışabilmesini mümkün kılmıştır. X'in gelişiminde MIT'nin yanı sıra, Stanford Üniversitesi, UC Berkeley ve diğer önde gelen araştırma kurumları önemli katkılarda bulunmuştur. Böylece, sistem zamanla hem kurumsal hem de açık kaynak topluluklarının desteğiyle evrimleşerek, günümüzün modern masaüstü ortamlarının temelini oluşturmuştur.

X, doğası gereği sadece bir pencereleme sistemi sunar. Menüler, görev çubukları veya tam masaüstü ortamı gibi kullanıcı deneyimine yönelik unsurlar sağlamaz. Bu tür özellikler, masaüstü ortamları tarafından eklenmiştir.

Temel Bilgiler:	
Yazılım Lisansı	MIT Lisansı
Destekleyen Topluluk	X.Org Vakfı (X.Org Foundation)
Dağıtım Tarihleri	İlk sürüm 1984, X11R1: 1987
Desteklediği Platformlar	UNIX, Linux, BSD, macOS ve diğer POSIX uyumlu sistemler
Yazılım Mimarisi	İstemci-sunucu modeli (X sunucusu donanım kaynaklarını yönetirken, istemciler ağ üzerinden iletişim kurar.)
Önemli Özellikler	Ağ şeffaflığı, X protokolü, pencere yöneticisi bağımsızlığı.
Geliştirildiği Diller	C, protokol tanımları için XML

X Window System'in Çıkışı, Gelişimi ve Etkileri

X Window System, Robert Scheifler1 ve Jim Gettys2 tarafından, ağ üzerinden çalışabilen ve platform bağımsız bir pencere sistemi sağlamak amacıyla başlatılmıştır. X11'in 1987'deki sürümü (X11R1), istemci-sunucu modelini standartlaştırmış ve UNIX dünyasında GUI'nin temel altyapısını oluşturmuştur ([21], [22], [23]). Bu temel üzerine inşa edilen ilk pencere yöneticiler ve sonrasında geliştirilen tam donanımlı masaüstü ortamları, kullanıcı dostu arayüzler ve gelişmiş etkileşim modelleri sunarak hem bireysel kullanıcıların hem de kurumsal sistemlerin kullanım ihtiyaçlarını karşılamıştır [24].

Bu evrimsel süreçte donanımın ve yazılım teknolojilerinin hızla gelişmesi, masaüstü ortamlarının mimari yapılarında ve sundukları özelliklerde sürekli yeniliklere gidilmesine neden olmuştur. Örneğin, grafik hızlandırıcı teknolojilerinin, pencere kompozitörlerinin ve modern API'lerin entegrasyonu, kullanıcı deneyimini önemli ölçüde değiştirmiştir. Özellikle iş istasyonları ve araştırma merkezlerinde önemli bir ihtiyaç haline gelen grafiksel kullanıcı arayüzleri (GUI), X Window System'in dağıtık yapılandırması sayesinde farklı cihazlar arasında kolay entegrasyon sağlamış ve bu da akademik, bilimsel ve endüstriyel alanlarda geniş bir kullanım bulmasını sağlamıştır. Örneğin, büyük veri görselleştirme sistemleri ve yüksek çözünürlüklü grafik işleme uygulamaları, X'in avantajlarından yararlanmıştır. Ayrıca, X protokolünün uzaktan erişim özellikleri, sunucu tabanlı uygulamaların yaygınlaşmasında da önemli bir rol oynamıştır.

Öne Çıkan Tarihsel Gelişmeler	
1984	MIT Laboratuvarlarında X prototiplerinin geliştirilmesi başlatıldı.
1987	X11'in piyasaya sürülmesi ile X'in bugünkü mimarisinin temel taşları atıldı.
1990'lar	Açık kaynak modelinin benimsenmesiyle, topluluk katkıları artarak sistemin geniş bir kullanıcı kitlesine ulaşması sağlandı.
2000 sonrası	X.Org Foundation çatısı altında, sürekli güncellemeler ve modern donanım desteğiyle, sistem evrimini sürdürdü.

Etkileri:

- Grafiksel arayüzlerin standartlaşmasını teşvik etti.
- Çoklu platform desteği sunarak yazılım geliştirme süreçlerini kolaylaştırdı.
- UNIX sistemlerinin iş istasyonlarında tercih edilmesinde önemli bir faktör oldu.

X Protokolü, İstemci-Sunucu Modeli ve Teknik Özellikler

X Window Sistemi, temel olarak istemci-sunucu mimarisine dayanmaktadır. Bu yapı, grafik işlemlerini gerçekleştiren X sunucusu ile bu işlemleri istemci uygulamalardan gelen komutlarla yöneten X istemcileri arasında net bir ayrım getirir. Böylece, uygulamalar (istemciler) doğrudan donanım kaynaklarına erişmek yerine, sunucu aracılığıyla iletişim kurar. Bu durum, sistem güvenliği ve esnekliği açısından avantaj sağlamaktadır.

Temel Teknik Bileşenler:

- **Ağ Şeffaflığı:** X protokolü, TCP/IP üzerinden çalışarak, uygulamaların farklı makinelerde çalışmasına ve sunucunun yerel donanım kaynaklarını kullanmasına olanak tanır. Böylelikle, uzak masaüstü uygulamalarının yerel ağlarda dahi kesintisiz çalışması mümkün hale gelir.
- **Pencere Yöneticisi:** X, pencere yöneticileri aracılığıyla kullanıcı arayüzü elemanlarını yönetir. Pencere yöneticileri, X sunucusu ile etkileşime girerek pencere çerçevelerini, menüleri, ikonları ve diğer grafiksel ögeleri düzenler. Ayrıca, X'in genişletilebilir yapısı sayesinde, XInput, XRender ve diğer eklentilerle grafiksel işlemler ve etkileşimler optimize edilmiştir.
- **Olay ve İstemci-Komut İletişimi:** X, kullanıcı girişleri (fare hareketi, klavye tuşları) ve pencere olaylarını (yenileme, yeniden boyutlandırma) sunucu aracılığıyla istemcilere iletir. Bu iletişim, protokol düzeyinde belirlenmiş standartlar çerçevesinde gerçekleşir; böylece, farklı uygulamaların uyumlu çalışması sağlanır.
- **Grafik Çizim ve Render İşlemleri:** X, temel çizim işlevleri (çizgi, dikdörtgen, metin çizimi) sunarken, gelişmiş grafik işlemler için harici kütüphaneler ve eklentilerle entegre çalışabilmektedir. Bu durum hem 2B hem de 3B grafik işlemlerinin gerçekleştirilmesini destekler.

Mimari Özellikler:

- **Modülerlik:** X'in modüler yapısı, sistemin farklı bileşenlerinin (örneğin; sunucu, istemciler, pencere yöneticileri) bağımsız olarak geliştirilmesine ve güncellenmesine olanak tanır. Bu yapı, sistemin geniş bir yelpazede uygulama alanı bulmasını sağlar.
- **Dil ve Protokol Bağımsızlığı:** X protokolü, donanım bağımsız olarak tasarlanmış olup farklı programlama dilleriyle yazılmış istemcilerin sunucuyla etkileşime girmesine imkân tanır. Temel protokol, genellikle C dilinde yazılmış olsa da üst katman uygulamaları çeşitli dillerde geliştirilebilir.
- **Genişletilebilirlik:** X'in protokolü ve mimarisi, ek özelliklerin entegre edilmesine olanak tanır. Örneğin, XInput2, çoklu dokunmatik ve gelişmiş giriş cihazı desteği sunarken; XComposite, pencere içeriklerinin yeniden düzenlenmesi için altyapı sağlar.

Bu teknik özellikler, X Window Sistemi'nin esnek, ölçeklenebilir ve farklı donanım ortamlarına uyarlanabilir olmasını sağlamıştır ([24], [25]).

X11'in Sınırlamaları ve Alternatifler

X11, zaman içinde ağ üzerinden uzaktan erişim, esneklik ve ölçeklenebilirlik gibi avantajlar sunmuş olsa da bazı önemli sınırlamaları da bulunmaktadır:

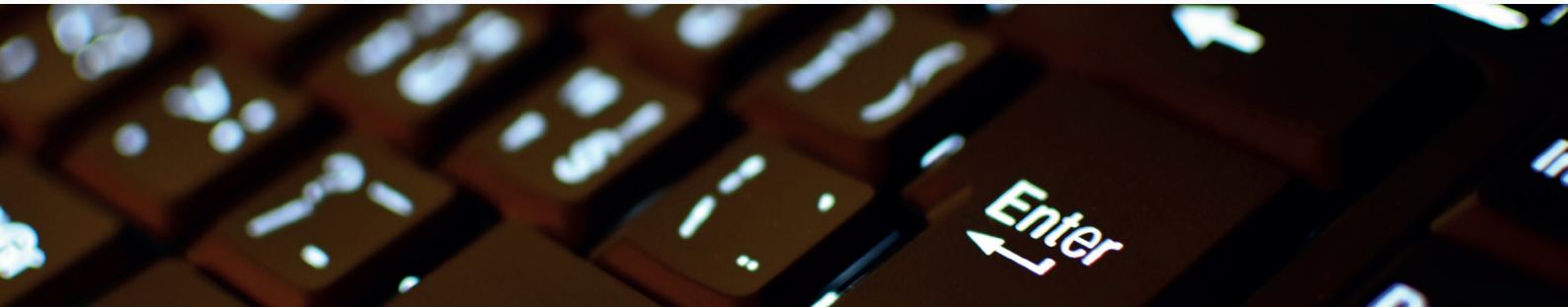
- **Güvenlik Zafiyetleri:** Veri iletiminin şifrelenmemesi, MIT-MAGIC-COOKIE gibi eski kimlik doğrulama yöntemlerinin modern sistemlerde yetersiz kalması.
- **Performans Sorunları:** Karmaşık istemci-sunucu modeli nedeniyle gereksiz ağ trafiği oluşması ve gecikmelere neden olması.
- **Wayland'in Ortaya Çıkışı:** X11'in sınırlamalarını gidermek amacıyla geliştirilen Wayland, daha basit ve güvenli bir mimari sunarak modern Linux masaüstü ortamlarına entegrasyonu hızlandırmıştır.

Sektörlere Kattıkları İyileştirme ve Yeni Kullanım Alanları

Eğitim: Açık kaynaklı masaüstü ortamları, eğitim kurumlarına maliyet tasarrufu sağladı ve öğrencilere erişilebilir teknolojiler sundu.

Bilim ve Araştırma: X Window System'in dağıtık yapısı, araştırma ekiplerinin uzak sunucular üzerinde çalışmasına olanak tanıdı.

Endüstri: Hafif masaüstü ortamları, endüstriyel otomasyon sistemlerinde verimliliği artırdı.



X.Org Foundation ve Topluluk Katkıları

X11'in gelişimi, X.Org Foundation tarafından yönetilmektedir. X.Org, açık kaynak geliştirme modelini benimseyerek topluluk katkılarını koordine eder ve sistemin modern donanım ve yazılım gereksinimlerine uyum sağlamasını güvence altına alır [26].

Yönetim ve Standartlar: X.Org Foundation, X11'in gelişimini yönlendirir ve güncellemeleri koordine eder.

Topluluk Katılımı: Dünya çapındaki geliştiriciler ve akademisyenler, X.Org etkinlikleri aracılığıyla bilgi paylaşımında bulunur.

Teknik olarak, X'in geliştirilme felsefesi; "basitlik, genişletilebilirlik ve ağ üzerinden şeffaf iletişim" prensipleri üzerine kurulmuştur. Bu özellikler, X'in ilk geliştirildiği ortamın ötesinde farklı donanım ve yazılım mimarileri üzerinde çalışabilmesini mümkün kılmıştır. X'in gelişiminde MIT'nin yanı sıra, Stanford Üniversitesi, UC Berkeley ve diğer önde gelen araştırma kurumları önemli katkılarda bulunmuştur. Böylece, sistem zamanla hem kurumsal hem de açık kaynak topluluklarının desteğiyle evrimleşerek, kullanıcıların, uygulamaları daha kolay yönetebildikleri kullanıcı dostu KDE, GNOME gibi modern masaüstü ortamlarının temelini oluşturmuştur.

Masaüstü Ortamlarının Gelişimi: X11'den Günümüze

İlk Pencere Yöneticilerden Masaüstü Ortamlarına Geçiş

X Window Sistemi'nin ilk dönemlerinde, UNIX-türevi sistemlerde grafiksel kullanıcı arayüzü yalnızca temel pencere yöneticileriyle sınırlı kalmaktaydı. Örneğin, TWM (Tab Window Manager) ve FVWM (F Virtual Window Manager) gibi erken dönem pencere yöneticileri, sadece pencere açma, kapama, yeniden boyutlandırma gibi temel işlevleri yerine getiriyordu. Bu yapı, sistem kaynaklarını minimal düzeyde kullanırken, kullanıcılara sadece basit pencere kontrolü sunuyordu.

1990'lı yılların ortalarına doğru artan kullanıcı beklentileri ve teknolojik gelişmeler, daha bütüncül ve entegre çözümlerin geliştirilmesini gerektirmiştir. Bu dönemde, Masaüstü Ortamları (Desktop Environments – DE) kavramı ortaya çıkmış; pencere yöneticileri, dosya yöneticileri, panel uygulamaları, sistem ayarları ve diğer temel bileşenleri tek çatı altında toplayarak, kullanıcılara bütünsel bir deneyim sunmaya başlamıştır.

Özellikle KDE ve GNOME projeleri, bu evrimin öncüsü olarak öne çıkmıştır. KDE, C++ dili ve Qt kütüphanesi kullanılarak modüler bir yapı sunarken; GNOME, C dili ve GTK+ kütüphanesi temelinde daha sade, kullanıcı dostu bir arayüz geliştirme hedefiyle hareket etmiştir. Bu entegrasyon süreci, masaüstü ortamlarının yalnızca estetik bir sunum değil, aynı zamanda sistem verimliliğini ve kullanıcı etkileşimini artıran kapsamlı platformlar haline gelmesine zemin hazırlamıştır.

Geçiş Süreçlerinin Gerekçeleri ve Sunulan Yenilikler

Masaüstü ortamlarının evriminde, geçiş süreçlerini tetikleyen temel gereksinimler şunlardır:

- **Entegre Kullanıcı Deneyimi:**

Basit pencere yöneticilerinin sunduğu sınırlı etkileşim, modern kullanıcıların dosya yönetimi, uygulama başlatma, sistem ayarları ve bildirim yönetimi gibi geniş kapsamlı ihtiyaçlarını karşılamaktan uzaktı. Bu nedenle, masaüstü ortamları tüm bu işlevleri entegre eden kapsamlı sistemler olarak geliştirilmiştir.

- **Görsel ve İşlevsel Yenilikler:**

Artan donanım gücü ve grafik teknolojilerinin evrimi, masaüstü ortamlarının daha dinamik, çekici ve etkileşimli hale gelmesine olanak tanımıştır. GNOME'un Mutter ve KDE'nin KWin gibi pencere yöneticileri, 3B kompozisyon, animasyonlar ve görsel efektler gibi özelliklerle kullanıcılara daha modern bir deneyim sunmaya başlamıştır.

- **Modülerlik ve Özelleştirilebilirlik:**

Geliştirilen yeni mimariler, kullanıcıların sistem üzerinde daha fazla kontrol sahibi olmasını sağlarken, aynı zamanda geliştiricilere ortamı ihtiyaçlara göre genişletme ve özelleştirme imkânı tanımıştır. Böylece, masaüstü ortamları hem performans hem de kişisel tercihler açısından esnek çözümler sunmuştur.

- **Sistem Performansı ve Kaynak Yönetimi:**

Modern masaüstü ortamları, eski pencere yöneticilerinin minimal kaynak kullanımı prensibini, kapsamlı işlevsellik ile dengelemeyi başarmıştır. Geliştirilen optimizasyon teknikleri, bellek ve CPU yönetiminde önemli iyileştirmeler sağlayarak hem eski hem de yeni donanım konfigürasyonlarında verimli çalışma imkânı sunmuştur.

Bu gelişmeler, UNIX-türevi sistemlerin kullanıcı deneyimini ve verimliliğini artırma hedefleri doğrultusunda, pencere yöneticilerinin ötesine geçerek tam teşekküllü masaüstü ortamlarına evrilmesine zemin hazırlamıştır ([27], [28]).

Teknolojik Dönüşümler ve Donanım/Yazılım Etkileşimleri

Son on yılda, teknolojik dönüşümler masaüstü ortamlarının gelişiminde belirleyici bir rol oynamıştır. Bu dönüşümler, donanım ve yazılım arasındaki etkileşimin optimize edilmesiyle, günümüzün modern grafiksel arayüzlerine evrilmeyi mümkün kılmıştır:

- **Donanım Hızlandırması ve Kompozit Pencere Yöneticileri:**

Grafik işlemciler (GPU) ve gelişmiş grafik teknolojileri, pencere yöneticilerinin donanım hızlandırması kullanarak 3B efektler, animasyonlar ve kompozit arayüzler sunmasına olanak tanımıştır. Bu sayede, masaüstü ortamları yalnızca statik bir arayüz yerine, akıcı ve görsel açıdan zengin deneyimler sağlamaya başlamıştır.

- **X'ten Wayland'e Geçiş Çabaları:**

X Window Sistemi uzun yıllar temel pencereleme altyapısı olarak kullanılsa da, bazı sınırlamaları nedeniyle modern ihtiyaçları tam olarak karşılayamamıştır. Bu kapsamda, daha basit, güvenli ve doğrudan GPU erişimi sunan Wayland gibi alternatif protokoller geliştirilmiştir. Her ne kadar bu geçiş süreci mevcut X tabanlı ekosistemle entegrasyon konusunda zorluklar içerse de, gelecekte masaüstü ortamlarının daha verimli çalışmasına yönelik önemli adımlar olarak görülmektedir.

- **Yazılım Katmanları ve API Gelişmeleri:**

Modern masaüstü ortamları, GTK+, Qt ve benzeri grafik kütüphaneleri sayesinde geliştiricilere hem performans hem de esneklik açısından büyük avantajlar sunmuştur. Bu API'ler, masaüstü ortamlarının modülerliğini artırırken donanım ile daha etkin bir etkileşim sağlamış; böylece sistemler, kullanıcı deneyimini iyileştiren yeni işlevselliklerle sürekli güncellenebilmiştir.

Bu teknolojik dönüşümler, UNIX-türevi sistemlerde GUI'nin evrimine yön verirken, donanım ve yazılım arasındaki sinerjinin optimizasyonu ile daha güvenli, esnek ve verimli masaüstü ortamlarının geliştirilmesine olanak tanımıştır ([29], [30]). Geleceğe yönelik araştırmalar, mevcut mimarilerin iyileştirilmesi ve yeni nesil protokollerin entegrasyonu ile bu evrimin daha da derinleşeceğini öngörmektedir.

Masaüstü Ortamları ve Teknik Özellikleri

UNIX ve UNIX benzeri sistemlerde masaüstü ortamları, kullanıcı deneyimini şekillendiren temel bileşenlerdir. UNIX ve GNU/Linux sistemlerde, günümüzde yaygın olarak kullanılan masaüstü ortamları olarak GNOME, KDE Plasma, XFCE, Cinnamon ve LXDE (LXQt) gösterilebilir.

GNOME

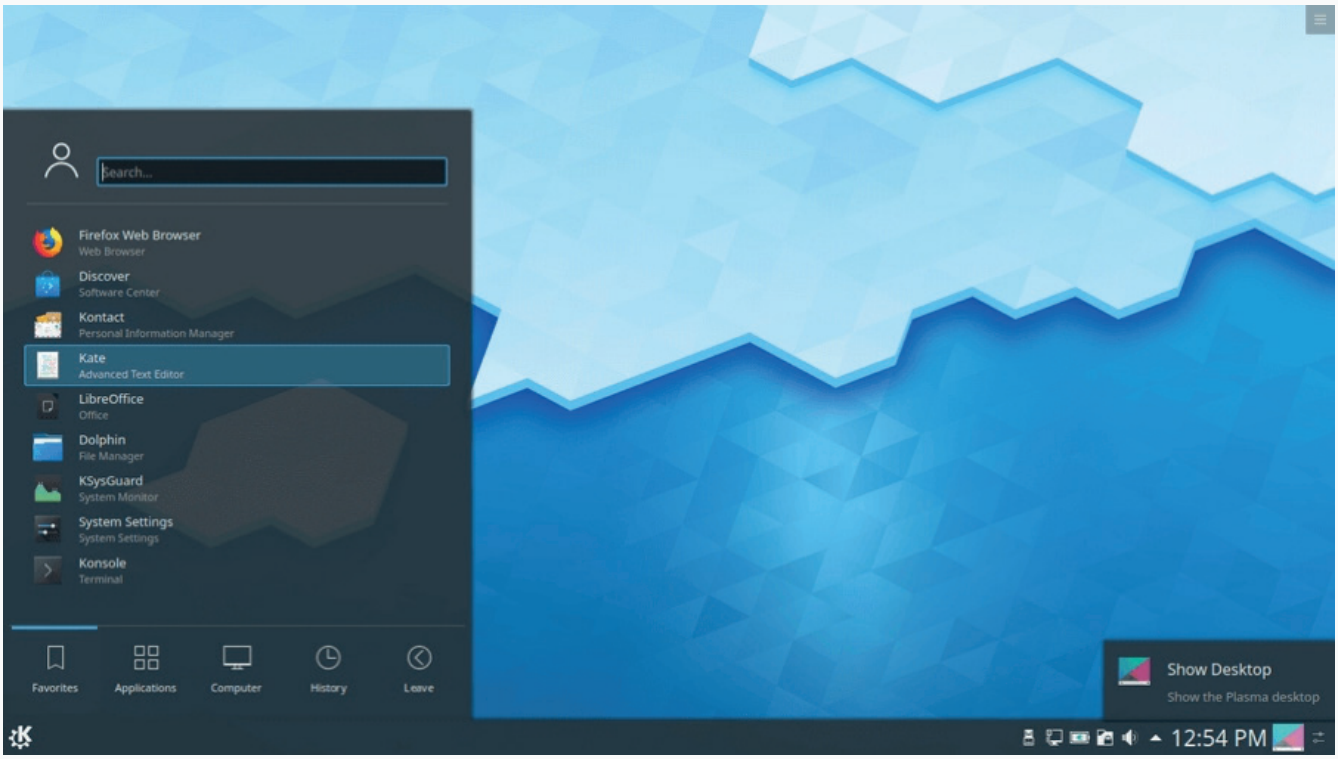
GNOME, UNIX benzeri sistemlerde minimalist tasarımı ve evrensel erişilebilirlik özellikleriyle tanınan bir masaüstü ortamıdır. GNOME Shell, dinamik iş akışı yönetimi ve uygulama organizasyonu için merkezi bir rol oynar. 1999'da başlayan proje, 2011'de GNOME 3 ile birlikte geleneksel masaüstü metaforlarını terk ederek aktivite odaklı bir modele geçmiştir. GTK 4 tabanlı mimarisi ve Wayland desteği, modern donanımlarda performans ve güvenlik hedefler. Özellikle engelli kullanıcılar için tasarlanan AT-SPI ve Orca entegrasyonu, erişilebilirlik alanında öncü kabul edilir.

Şekil 1. Gnome Masaüstü Ortamı

Temel Bilgiler:	
Yazılım Lisansı	GNU GPL v2+ ve GNU LGPL v.2.1
İlk Dağıtım Tarihi	1999 (GNOME 2 serisi: 2002, GNOME 3 serisi: 2011)
Destekleyen Topluluk	GNOME Vakfı (GNOME Foundation)
Desteklediği Platformlar	Linux, BSD
Yazılım Mimarisi	GTK (GIMP Toolkit [eski]) tabanlı
Geliştirildiği Diller	C, JavaScript (GNOME Shell)

KDE Plasma

KDE Plasma, UNIX benzeri sistemlerde Qt çerçevesini kullanarak yüksek derecede özelleştirilebilir ve görsel olarak tutarlı bir masaüstü deneyimi sunar. 1996'da başlayan KDE projesinin evrimi, 2008'de Plasma adıyla mimari bir yeniden yapılanmaya gidilmesini sağlamıştır. Güncel sürümleri, Wayland desteği ve modüler tasarımıyla hem eski hem de modern donanımlarda performans hedefler. Dolphin (dosya yöneticisi) ve KWin (pencere yöneticisi) gibi bileşenler, KDE Frameworks ile derin entegrasyon sağlar. KDE Plasma, kullanıcıların masaüstü deneyimlerini kapsamlı şekilde özelleştirmelerine olanak tanıyan dinamik bir yapıya sahiptir.

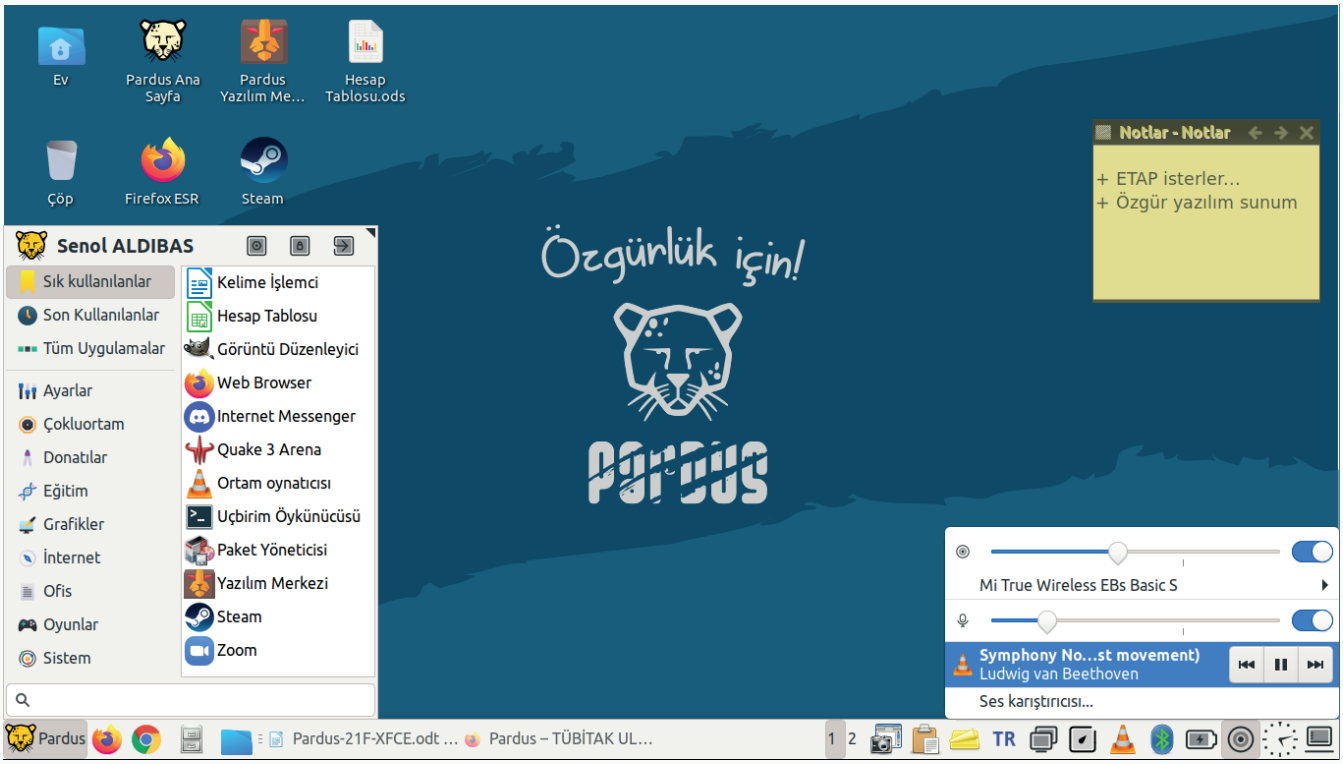


Şekil 2. KDE Plasma Masaüstü Ortamı

Temel Bilgiler:	
Yazılım Lisansı	GNU LGPL ve GPL
İlk Dağıtım Tarihi	12 Temmuz 1998, Plasma 5: 15 Temmuz 2014
Destekleyen Topluluk	KDE Topluluğu (KDE e.V [vakıf])
Desteklediği Platformlar	Linux, BSD, Windows/macOS (sınırlı kullanım)
Yazılım Mimarisi	Qt tabanlı (Plasma 6: Qt 6)
Geliştirildiği Diller	C++ (Çekirdek), QML (Kullanıcı arayüz tasarımı)

XFCE

XFCE, özellikle düşük sistem gereksinimleri ve hafif tasarımıyla öne çıkan bir masaüstü ortamıdır. 1996'da XForms araç setiyle başlayan proje, 2001'de GTK'ya geçiş yaparak modern UNIX/Linux sistemlerinde yaygın kabul görmüştür. Temel tasarım felsefesi, modülerlik, sadelik ve UNIX'in "basit araçlarla çalışma" ilkesine dayanır. XFCE, özellikle eski donanımlarda bile modern ve işlevsel bir kullanıcı deneyimi sunacak şekilde tasarlanmıştır.



Şekil 3. Xfce Masaüstü Ortamı

Temel Bilgiler:	
Yazılım Lisansı	GNU LGPL ve GPL
İlk Dağıtım Tarihi	1996
Destekleyen Topluluk	XFCE Topluluğu
Desteklediği Platformlar	Linux, BSD, Windows/macOS (sınırlı kullanım)
Yazılım Mimarisi	GTK
Geliştirildiği Diller	C

Cinnamon

Cinnamon, GNOME Shell ile birlikte yayınlanan GNOME 3 sürümünün geleneksel masaüstü düzeninden farklı bir deneyim sunması ve bu değişikliğin memnuniyetsizlik yaratması sonucunda, Ocak 2012'de GNOME masaüstü ortamının kullanıcı arayüzü bileşeni olan GNOME Shell'in çatallanmasıyla ortaya çıkmış; Ekim 2013'te yayınlanan Cinnamon 2.0 sürümüyle birlikte ise tamamen bağımsız bir masaüstü ortamı haline gelmiştir [31].

Temel Bilgiler:	
Yazılım Lisansı	GNU GPL 2 veya sonrası (GPLv2+) [32]
İlk Dağıtım Tarihi	Ocak 2012
Destekleyen Topluluk	Linux Mint Topluluğu tarafından geliştirilmektedir.
Desteklediği Platformlar	Linux ve UNIX benzeri işletim sistemleri
Yazılım Mimarisi	GNOME 3 temelli mimariden türemiştir.
Geliştirildiği Diller	C, C++, JavaScript

LXDE

LXDE, düşük donanım kaynaklarına sahip cihazlarda kullanım için optimize edilmiş hafif bir masaüstü ortamıdır. Modern sürümleri olan LXQt, Qt tabanlı bir yapıya sahiptir ve görsel olarak daha zengin özellikler sunar.

Temel Bilgiler:	
Yazılım Lisansı	GNU LGPL
İlk Dağıtım Tarihi	2006
Destekleyen Topluluk	LXDE Topluluğu
Desteklediği Platformlar	Linux, BSD
Yazılım Mimarisi	GTK+ tabanlı (ilk sürümler), daha sonra Qt'ye geçiş yapılarak LXQt olarak yeniden markalanmıştır.
Geliştirildiği Diller	C, C++ (LXQt için)

Kaynak Kullanımı ve Performansa Göre Masaüstü Ortamlarının Karşılaştırılması

Modern masaüstü ortamları, tasarım felsefeleri ve mimari tercihleri doğrultusunda farklı performans ve kaynak kullanımı özellikleri sergilemektedir.

- **GNOME ve KDE Plasma:**

Geniş fonksiyonellik, modern görsel efektler ve zengin entegrasyon sunmaları sebebiyle, özellikle bellek ve CPU kullanımında daha yüksek maliyetlere sahiptir. Bu durum, güçlü donanım gereksinimi ve daha fazla işlem gücü isteyen kullanıcılar için uygun olabilir ([29], [30]).

- **XFCE ve LXDE:**

Minimal ve hafif yapıları sayesinde eski donanımlarda dahi düşük kaynak kullanımı ile stabil performans sunarlar. Kullanılabilirlik açısından, hızlı yanıt süreleri ve az hata ile öne çıkmaktadırlar.

- **Cinnamon:**

GNOME tabanlı modern unsurlarla geleneksel masaüstü düzeni arasında denge kuran Cinnamon, orta seviye kaynak kullanımı ile hem estetik hem de işlevsel bir deneyim sunar.

Bu farklılıklar, kullanıcıların sistem konfigürasyonlarına ve ihtiyaçlarına göre DE seçimini belirleyici unsurlar olarak karşımıza çıkmaktadır.

Bölüm Özeti

Bu bölümde, X Window Sistemi'nin ortaya çıkışından başlayarak UNIX-türevi sistemlerde kullanılan masaüstü ortamlarının evrimsel süreci, teknik mimarileri, topluluk dinamikleri ve geleceğe yönelik trendleri kapsamlı biçimde incelemiştir. Araştırmada, X Window Sistemi'nin modüler, ağ şeffaflığına dayalı mimarisinin modern grafiksel arayüzlerin temelini oluşturduğu; ancak teknolojik gereksinimler doğrultusunda, Wayland gibi yeni protokollerle alternatif çözümlerin geliştirilmesinin evrimin devamını sağladığı ortaya konulmuştur ([21], [22], [30], [35]).

Modern masaüstü ortamları açısından yapılan analizde, GNOME ve KDE Plasma gibi geniş işlevsellik sunan masaüstü ortamlarının, görsel efektler ve zengin entegrasyon imkanları sayesinde kullanıcı deneyimini üst düzeye çıkardığı; ancak yüksek kaynak kullanımı gibi dezavantajlarının bulunduğu gözlemlenmiştir. Buna karşın, XFCE ve LXDE'nin düşük kaynak tüketimi ve stabilite odaklı yapıları, özellikle eski donanımlar ve performansın kritik olduğu senaryolarda tercih edilebilirliği artırmaktadır. Cinnamon ise, modern tasarım unsurlarını geleneksel masaüstü düzeniyle harmanlayarak orta seviye kaynak kullanımı ve estetik denge sunmaktadır ([22], [35]).

Karşılaştırmalı analiz, her masaüstü ortamının geliştirme felsefesi ve topluluk dinamiklerinin, sürdürülebilirlik ve yenilikçilik açısından belirleyici olduğunu göstermiştir. GNOME'un merkezi yönetişim modeli, KDE'nin modüler ve esnek yaklaşımı ile XFCE, LXDE ve Cinnamon'ın topluluk odaklı, minimal yapılarına dair gözlemler, masaüstü ortamlarının kullanım amaçları ve hedef kitleleri arasında

önemli farklılıklar ortaya koymuştur ([34], [36]).

Geleceğe yönelik değerlendirmelerde ise, enerji verimliliği, mobil uyumluluk, bulut entegrasyonu ve mikroservis tabanlı mimari yaklaşımlarının, masaüstü ortamlarının gelişiminde belirleyici rol oynayacağı öngörülmektedir. Özellikle Wayland gibi yeni nesil grafik protokollerinin, doğrudan GPU erişimi ve geliştirilmiş güvenlik özellikleri sayesinde mevcut X tabanlı sistemlerin yerini kademeli olarak alabileceği düşünülmektedir ([35], [36]). Bu dönüşüm, UNIX-türevi sistemlerde masaüstü ortamlarının daha esnek, modüler ve performans odaklı hale gelmesine zemin hazırlamaktadır.

Sonuç olarak bu bölüm; UNIX sistemlerinde masaüstü ortamlarının tarihsel evrimi, teknik altyapıları ve topluluk etkileşimleri arasındaki karmaşık ilişkiyi ortaya koymuş, gelecekte yapılacak iyileştirmeler ve inovasyonlar için kapsamlı bir analiz sunmuştur. Elde edilen bulgular, hem geçmişteki teknolojik dönüşümlerin anlaşılmasına hem de ilerleyen dönemlerde yapılacak araştırmaların ve uygulama geliştirmelerinin yönlendirilmesine katkı sağlayacak niteliktedir.

Sonuç ve Öneriler

Bu çalışma, UNIX felsefesinin tarihsel temellerinden başlayarak özgür yazılım yaklaşımının gelişimine ve modern Linux dağıtımlarının oluşum sürecine ışık tutmak amacıyla hazırlanmıştır. Çekirdek mimarisinin temel bileşenleri (Proses yönetimi, bellek yönetimi, zamanlama politikaları, kesme yapısı vb.) ayrıntılı şekilde ele alınmış, ardından minimal dağıtım oluşturma konularında, Buildroot ile sıfırdan sistem oluşturma ve Debian temelli yansı yönetimi gibi uygulamalar örneklendirilmiştir. Son olarak, "X Window System'den Wayland'e uzanan masaüstü ortamlarının" teknolojik dönüşüm süreci ve GNOME, KDE gibi platformların kullanıcılara sunduğu olanaklar incelenmiştir. Bu bütüncül bakış, farklı gereksinimlere yönelik özelleştirilmiş çözümler geliştirilmesinde rehber niteliğinde olup, gelecek çalışmalar için bir temel oluşturmaktadır.

Özgür Yazılım modelinin esnek yapısının, kamu kurumları ve şirketlerin özel çalışmaları ve çözümlerine izin vermesi sayesinde Linux, hem Özgür Yazılım hem de Açık Kaynak ekosistemlerinin itici gücü haline gelmiştir. Günümüzde süper bilgisayarlar, sunucular ve milyarlarca IoT cihazında Linux çekirdeği ve GNU araçları kullanılmaktadır. Vakıf, topluluk, büyük şirket ve kurumların desteği ile Açık Kaynak/Özgür projelerin yükselişi artarak devam edecektir. Bu farkındalık ile, kamu kurumları ve şirketlerin çözümlerini Açık Kaynak/Özgür projelere entegre ederek çözümlerini sürdürmesi; maliyet, kalite, zaman yönetimi ve güvenlik açısından çok önemli avantajlar sağlayacaktır. Linux ve bulut tabanlı çözümler, teknolojik özgürlüğün ve inovasyonun sembolü olarak, insanlığın dijital dönüşümündeki kilit rolünü sürdürecektir.

Kaynakça

[1] D. M. Ritchie, “The Evolution of the UNIX Time-sharing System.”, Bell System Technical Journal, 57(6), 1905–1929, Temmuz-Ağustos 1978.

[2] K. Thompson, “UNIX Implementation.” Bell System Technical Journal, 57(6), 1931–1946, Temmuz-Ağustos 1978.

[3] Ken Thompson interviewed by Brian Kernighan at VCF East 2019. Erişim Adresi: https://www.youtube.com/watch?v=EY6q5dv_B-o, Erişim Zamanı: 14.02.2025.

[4] S. J. Leffler, M. K. McKusick, M. J. Karels & J. S. Quarterman, “The Design and Implementation of the 4.3BSD UNIX Operating System”, Addison-Wesley, 1989.

[5] M. K. McKusick, “Twenty Years of Berkeley UNIX: From AT&T-Owned to Freely Redistributable.” In Proceedings of the 1999, BSDCon, 1999.

[6] Torvalds, Linus ve Diamond, David. Just for Fun: The Story of an Accidental Revolutionary. Harper Business, 2001.

[7] GNU Genel Kamu Lisansı (GPL). Erişim Adresi: https://www.gnu.org/licenses/gpl-3.0.html, Erişim Zamanı: 20.02.2025.

[8] Tanenbaum, Andrew S. “MINIX ve Linux: Bir Karşılaştırma”.

[9] Usenet arşivi: Torvalds'ın orijinal duyuru mesajı. Erişim Adresi: https://www.cs.cmu.edu/~awb/linux.history.html / https://www.usenetarchives.com, Erişim Zamanı: 28.12.2024.

[10] Linux Çekirdeği: Tanım ve İşlevler: Torvalds'ın Intel 80386 işlemcinin mimarisiyle ilgili çalışmaları, Erişim Adresi: https://lwn.net/Articles/292143/ , Erişim Zamanı: 25.12.2024.

[11] D. P. Bovet & M. Cesati, “Understanding the Linux Kernel (3rd ed.).” O’Reilly Media, 2005.

[12] R. Love, “Linux Kernel Development (3rd ed.)”, Addison-Wesley Professional, 2010.

[13] J. Corbet, A. Rubini & G. Kroah-Hartman, “Linux Device Drivers (3rd ed.)”, O’Reilly Media, 2005.

[17] The Linux Kernel Archives. Erişim Adresi: https://www.kernel.org , Erişim Zamanı: 05.01.2025.

[18] Documentation in the Linux source tree. Erişim Adresi: https://docs.kernel.org , Erişim Zamanı: 22.01.2025.

[19] Torvalds, L., & the Linux community. Linux source code. Erişim Adresi: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git, Erişim Zamanı: 22.12.2024.

[20] Intel 64 and IA-32 Architectures Software Developer's Manual, Erişim Adresi: https://cdrdv2-public.intel.com/825743/325462-sdm-vol-1-2abcd-3abcd-4.pdf, Erişim Zamanı: 15.01.2025.

[21] X.Org Foundation, "X Window System – Architecture and Implementation", X.Org Documentation, 2020.

[22] R. Love, "X Window System: A Technical Overview", Journal of Computer Graphics, 2005.

[23] J. Gettys, R. W. Scheifler, "The X Window System", Temmuz 1986.

[24] D. Steinman, "From Window Managers to Desktop Environments: An Evolutionary Study", Proceedings of the UNIX Symposium, 2010.

[25] J. Newman, "Network Transparency in X11: Design and Implementation," ACM Transactions on Graphics, 2012.

[26] X.Org Foundation, "Community and Governance in Open Source Graphics Systems," Whitepaper, 2018.

[27] J. Doe, "The Evolution of Window Managers in UNIX Systems," Journal of Open Source Development, 2011.

[28] A. Smith, "From Simple Windows to Integrated Desktop Environments: A Technical Perspective," Proceedings of the Linux Symposium, 2013.

[29] L. Brown, "Hardware Acceleration and the Modern Desktop Environment," ACM Computing Surveys, 2015.

[30] M. White, "Wayland and the Future of UNIX-based Graphical Systems," IEEE Transactions on Software Engineering, 2017.

[31] A. Krause, "Linux Mint Essentials", Packt Publishing, 2013.

[32] Linux Mint Cinnamon Kod Deposu, Erişim Adresi: https://github.com/linuxmint/cinnamon, Erişim Zamanı: 03.01.2025.

[33] E. Roberts, "Resource Management in Modern Desktop Environments," Linux Performance Journal, 2018.

[34] K. Harris, "User-Centered Development in Desktop Environments," International Journal of Human-Computer Interaction, 2018.

[35] P. O'Neil, "Transitioning to Wayland: Opportunities and Challenges," ACM SIGGRAPH Conference Proceedings, 2020.

[36] T. Murphy, "The Future of Desktop Virtualization and Cloud Integration," IEEE Transactions on Cloud Computing, 2021.

Resimler:

Sayfa 22: https://www.freepik.com/free-vector/neon-circuit-board-background_6072271.htm

Sayfa 31: https://www.freepik.com/free-ai-image/global-business-internet-network-connection-iot-internet-things-business-intelligence-concept-busines-global-network-futuristic-technology-background-ai-generative_49395764.htm

Sayfa 33: https://www.freepik.com/free-vector/blue-futuristic-networking-technology_13311397.htm#fromView=keyword&page=1&position=4&uuid=9ce9c332-cacc-467b-806a-c7d94ca39156&query=It

Sayfa 40: https://www.freepik.com/free-photo/laptop-with-blue-optical-fiber_11382733.htm

Sayfa 50: https://www.freepik.com/free-photo/keyboard-close-up_1234599.htm

Sayfa 53: https://www.freepik.com/free-photo/application-programming-interface-hologram_18098423.htm#fromView=keyword&page=1&position=25&uuid=41421b6f-4b00-4970-a34d-f7a6542cd89f&query=Api

EK-A: Tanımlar

Tanım	Açıklama
amd64	Intel ve AMD işlemcileri için 64 bit x86 mimarisi.
arm64	Mobil ve düşük güç tüketimli cihazlar için 64 bit ARM mimarisi.
AT&T	UNIX'in ilk ticari geliştiricisi olan telekomünikasyon şirketidir.
Bash	Bourne Again Shell
Batch Sistemler	Kullanıcı etkileşimi olmadan işleri sırayla işleyen sistemler.
Bell Labs	Telekomünikasyon ve yazılım alanında araştırmalar yapan laboratuvarlardır.
Boot loader	Donanım tarafından tetiklenen sistemin nasıl çalıştırılacağını kontrol eden yazılımdır.
Borular (Pipes)	İşlemler arası veri iletişimini sağlayan mekanizmadır.
Bottom Half (Alt Yarı)	Kesme sonrası ertelenen işlemler (softirq, tasklet, workqueue).
Buddy System	Boş bellek bloklarını birleştirip bölerek parçalanmayı önleyen bellek yönetim algoritmasıdır.
C Dili	Dennis Ritchie tarafından UNIX için geliştirilen programlama dilidir.
Cinnamon	X Pencere Sistemi için ücretsiz ve açık kaynaklı bir masaüstü ortamıdır.
Çok Görevli (Multitasking)	Aynı anda birden fazla işlemin eş zamanlı yürütülmesidir.
Çok Kullanıcılı	Aynı anda birden fazla kullanıcının sistemi kullanabilmesidir.
Dağıtık (Network Transparent)	Ağ üzerindeki kaynakların tek bir sistem gibi çalışabilmesidir.
Debian	"Özgür Yazılım" lisanslarına sahip işletim sistemi dağıtımı.
Demand Paging	Talep Üzerine Sayfalama.
Disk Belleği (Takas Alanı)	Fiziksel bellek dolduğunda verilerin geçici olarak saklandığı disk bölümü.
Dosya Sistemleri	Verilerin depolama cihazlarında nasıl organize edildiğini ve erişildiğini düzenleyen yapılardır.
DualBoot	Bir bilgisayara yan yana iki işletim sistemi yüklenmesi durumunda, seçilen sistemin açılabilmesi işlemidir.
Free Software (Özgür Yazılım)	Bir kısıtlamaya bağlı olmaksızın herkesçe erişilebilen, kullanılabilen, değiştirilebilen ve paylaşılabilen kısaca kullanıcıya ve topluma saygı duyan yazılımlardır.
GRUB	Bilgisayar açılışında işletim sistemlerini yüklemeye yarayan özgür bir önyükleme yazılımdır.
GTK (Gimp ToolKit (eski)/ Toolkit)	Grafiksel kullanıcı arayüzü geliştirme araç takımı.
Huge Pages (Büyük Sayfalar)	Standart 4 KB'dan daha büyük bellek sayfaları.
ISO	Optik disklerin birebir kopyasını içeren dosya formatı.
İstemci-Sunucu Modeli	İstemcilerin sunucu üzerinden kaynaklara eriştiği iletişim mimarisi.
Kaynak Kod	Yazılımın programlama dilindeki orijinal hali.
KDE (K Desktop Environment)	-K Masaüstü Ortamı- "KDE Topluluğu" tarafından geliştirilen, X Pencere Sistemi ve Wayland protokolünü destekleyen masaüstü ortamı (Plasma).
Kernel	İşletim sisteminin donanım ve yazılım arasındaki temel iletişim katmanı (Çekirdek).
Kernel Uzayı	Çekirdek kodunun ve verilerinin çalıştığı bellek bölgesi.
Linus Torvalds	Linux çekirdeğinin yaratıcısı.
Linux	UNIX türevi işletim sistemi çekirdeği.
LXDE (Lightweight X11 Desktop Environment)	Eski donanımlar için optimize edilmiş basit masaüstü ortamı.

Tanım	Açıklama
LXQt (Qt port of LXDE)	X Pencere Sistemi için ücretsiz ve açık kaynaklı hafif bir masaüstü ortamıdır. LXDE'nin Qt kütüphanesi kullanılarak geliştirilmiş halidir.
Memory Management (Bellek Yönetimi)	Fiziksel ve sanal bellek tahsisi, koruma ve optimizasyon işlemleri.
Memory Zones (Bellek Bölgeleri)	Fiziksel belleğin farklı amaçlara ayrılmış bölümleridir.
MINIX (Mini UNIX)	Eğitim amaçlı geliştirilmiş basit bir UNIX türevi işletim sistemidir.
MIT (Massachusetts Institute of Technology)	Teknoloji ve mühendislik alanında önde gelen bir araştırma üniversitesidir.
MIT-MAGIC-COOKIE	X Window System'de basit kimlik doğrulama için kullanılan bir yöntemdir.
Mikro-Çekirdek	Temel işlevlerin çekirdekte, diğerlerinin kullanıcı alanında olduğu mimari.
Modülerlik	Sistemin bağımsız bileşenlerle genişletilebilir olması.
Monolitik Çekirdek	Tüm sistem bileşenlerinin çekirdek içinde çalıştığı mimari.
Multi-Görev (Çoklu Görev)	Aynı anda birden fazla işlemin eş zamanlı yürütülmesi.
Net (Network)	Ağ
Network Stack	Veri iletişimini katmanlar halinde yöneten ağ protokolleri ve araçları bütünü.
Open Source	Açık Kaynak
openssl (Open Secure Sockets Layer)	Açık kaynaklı şifreleme ve güvenlik protokolü kütüphanesi.
PDP-7	1960'larda üretilen dijital bir bilgisayar modeli.
Pencere Yöneticisi	Pencerelerin boyut, konum ve görünümünü yöneten yazılım.
Process Scheduling	İşlemlerin ne zaman ve nasıl çalıştırılacağını düzenleyen süreç.
Process Switching	İşlemler arası geçişi yöneten mekanizma.
procfs (Process File System)	Süreç ve sistem bilgilerini sanal dosyalarla sunan dosya sistemi.
Proses Address Spaces	Her sürecin sanal bellek bölgesi.
Proses Management	Süreç Yönetimi.
Qemu	Quick Emulator (Basit ve hızlı bir sanal makina yazılımı).
Qt (Q Toolkit)	Birden çok platformu destekleyen bir grafiksel kullanıcı arayüzü geliştirme araç takımıdır.
Real-Time Scheduling	Gerçek Zamanlı Zamanlama.
Red-Black Tree	Dengeli bir ikili ağaç yapısı (CFS'de görev sıralaması için kullanılır).
Render	Görsel ögelerin işlenmesi.
Sanal Bellek	Fiziksel bellek ile disk alanını birleştirerek programlara daha geniş bellek alanı sunan soyutlama katmanı.
Sayfa Tabloları	Sanal bellek adreslerini fiziksel adreslere eşleyen veri yapıları.
Sayfa Tahsisi	Fiziksel belleği sabit boyutlu (genellikle 4 KB) bloklara bölerek yönetme işlemi.
Sh	Shell (Kabuk).
Space Travel	Ken Thompson'ın PDP-7'de geliştirdiği erken dönem bir bilgisayar oyunu.
Swap (Takas Alanı)	Diskin bir bölümünü RAM - bellek - gibi kullanmamızı sağlayan biçim.
TCP/IP	İnternet üzerinden veri iletişimini sağlayan protokol paketi.
The Cathedral and the Bazaar	Açık kaynak geliştirme modelini anlatan kitap.
Top Half (Üst Yarı)	Kesme anında hızlıca çalıştırılan kritik kod bölümü.

Tanım	Açıklama
twm	X11 sistemleri için basit ve hafif olan temel pencere yöneticisidir.
Ubuntu	Kullanıcı dostu bir Linux dağıtımı.
UC Berkeley	University of California, Berkeley
UNIX	Çok kullanıcılı ve çok görevli bir işletim sistemi mimarisi.
Usenet (User's Network)	Kullanıcıların haber grupları aracılığıyla iletişim kurduğu erken dönem internet ağı.
User Space	Kullanıcı programlarının çalıştığı bellek bölgesi.
Virtual Memory Areas (VMAs)	Bir sürecin bellek bölümlerini tanımlayan veri yapısı.
Virtual Runtime (vruntime)	Sanal Çalışma Süresi.
Virtualization	Sanallaştırma.
Wayland	X11'in sınırlamalarını aşmak için geliştirilen modern grafik sunucu protokolü.
X İstemcileri	X sunucusu üzerinden grafik işlemleri gerçekleştiren uygulamalar.
X Protokolü	X Window System'in iletişim kurallarını tanımlayan protokol.
X Sunucusu	Grafik donanımını yöneten ve istemcilere hizmet veren yazılım.
X.Org Foundation	X Window System'in açık kaynak geliştirilmesini yöneten organizasyon.
X11 (X Window System Version 11)	UNIX sistemlerinde grafiksel arayüz sağlayan 11. sürüm pencereleme sistemi.
XComposite (X Compo. Extension)	Pencerelerin kompozit (birleşik) görüntülenmesini sağlayan X eklentisi.
XFCE (XForms Common Enviroment)	X Formları Ortak Ortamı: X Pencere Sistemi için ücretsiz ve açık kaynak bir masaüstü ortamıdır.
XFS (X File System)	Büyük veri ve yüksek ölçeklenebilirlik için tasarlanmış dosya sistemi.
XRender (X Rendering Extension)	X Window System'de grafik işleme işlemlerini optimize eden eklenti.

EK-B: Fonksiyon, Kütüphane ve İsimler

Fonksiyon/Kütüphane v.s	Açıklama
apt (Advanced Package Tool)	Debian tabanlı dağıtımlarda kullanılan paket yönetim sistemidir.
Aptly	Debian paket depolarını yönetmek için kullanılan bir araç.
buildroot	Kolay ve toplu bir şekilde kaynak kod yapılandırmaya ve derlemeye yarayan araç.
busybox	Tek bir dosyadan oluşan temel komutları içeren yazılım.
clone()	Kaynak paylaşımlı süreç veya thread oluşturan sistem çağrısı.
Context Switch (Bağlam Değişimi)	CPU'nun bir süreçten diğerine geçişi.
Contrib	Özgür yazılım olan ancak çalışması için özgür olmayan bileşenlere ihtiyaç duyan paketlerin bulunduğu Debian deposu.
elf	Çalıştırılabilir ve linklenebilir dosya türü.
exec()	Sürecin çalışan programını değiştiren sistem çağrısı ailesi.
exit()	Süreci sonlandıran sistem çağrısı.
Fair Scheduling	İşlemci kaynaklarının eşit paylaşımını hedefleyen zamanlama.
files_struct	Sürecin açık dosyalarını saklayan veri yapısı.
fork()	Yeni bir süreç oluşturan sistem çağrısı.
glibc	GNU tarafından geliştirilen temel C kütüphanesi.
Intel 80386	1985'te piyasaya sürülen 32-bit bir işlemci.
Interrupt (Kesme)	Donanım veya yazılım tarafından oluşturulan acil işlem talebi.
Interrupt Handling	Kesmeleri algılama ve yönetme süreci.
i386	Eski Intel işlemciler için 32 bit x86 mimarisi.
init	Bir linux dağıtımında ilk olarak çalıştırılan yazılım.
initramfs	Bir linux dağıtımında ilk olarak belleğe yüklenen dosya.
ldd	Bir elf dosyasının bağımlılıklarını görmek için kullanılan komut.
libc	C programlama dilinde kullanılan temel kütüphane.
live-build	Debian tabanlı canlı sistemler oluşturmak için kullanılan bir araç.
make	Kaynak kodları derlemek ve yönetmek için kullanılan otomasyon aracı.
mm_struct	Sürecin bellek haritasını yöneten çekirdek veri yapısı.
Mount	Bir dosya sistemini belirli bir dizin altında erişilebilir hale getirme işlemi.
ncurses (New Curses)	Metin tabanlı arayüzler oluşturmak için kütüphane.
non-free	Özgür olmayan yazılımları içeren Debian deposu.
page fault (Sayfa Hatası)	Sanal bir sayfaya erişimde fiziksel bellek bulunamaması durumu.
Page Tables (Sayfa Tabloları)	Sanal adresleri fiziksel adreslere eşleyen hiyerarşik tablolar.
pthread_create()	POSIX thread'i oluşturan kullanıcı alanı fonksiyonu.
root	Sistem yöneticisi kullanıcı adı.
Root (/)	Dosya sistemi kök dizini.
rootfs	Bir linux dağıtımının sistem dosyalarının bulunduğu ana dizin.
SCHED_DEADLINE	Son Tarih Tabanlı Zamanlama (Görevlerin belirlenen sürelerde tamamlanmasını garanti eder).
SCHED_FIFO	İlk Giren İlk Çıkar (Gerçek zamanlı görevler için kesintisiz çalışma politikası).

Fonksiyon/Kütüphane v.s	Açıklama
SCHED_RR	Round-Robin (Zaman dilimli gerçek zamanlı politikası).
SCSI (Small Computer System Interface)	Sabit Disk, CD sürücü, tarayıcı, yazıcı gibi aygıtları paralel arabirim standartlarından daha uyumlu ve gelişmiş bir şekilde kontrol eden standart.
SIGSEGV (Segmentation Fault Signal)	Bellek Segmentasyon Hatası Sinyali (Geçersiz bellek erişiminde gönderilen sinyal).
Slab Allocator	Küçük ve sık kullanılan bellek nesnelerini verimli şekilde yönetmek için kullanılan bellek ayırma mekanizması.
SLOB (Simple List Of Blocks)	Basit Blok Listesi (Düşük kaynaklı sistemlerde kullanılan bellek tahsisçisi).
SLUB (Unqueued Slab Allocator)	Kuyruksuz Slab Tahsisçisi (Modern ve ölçeklenebilir bellek yönetimi).
Softirqs (Yazılım Kesmeleri)	Düşük gecikmeli arka plan işlemleri için çekirdek içi mekanizma.
struct dentry	Dizin girişlerini ve önbelleğini yöneten yapı.
struct inode	Dosya meta verilerini (izinler, sahiplik vb.) saklayan yapı.
struct super_block	Dosya sisteminin mount edilmiş örneğini temsil eden yapı.
swappiness	Çekirdeğin swap kullanma eğilimini kontrol eden parametre.
SYSCALL_DEFINE0	Sistem Çağrısı Makrosu (Argümansız sistem çağrılarını tanımlamak için kullanılır).
sysfs (System File System)	Donanım ve çekirdek verilerini kullanıcıya sunan dosya sistemi.
System Call (Sistem Çağrısı)	Kullanıcı programlarının çekirdek işlevlerine erişmesini sağlayan mekanizma.
TASK_INTERRUPTIBLE	Kesilebilir bekleyiş durumu (örneğin girdi beklerken).
TASK_RUNNING	Sürecin çalışmaya hazır veya çalışıyor olduğu durum.
task_struct	Sürecin durumunu, PID'sini ve kaynaklarını saklayan çekirdek veri yapısı.
Tasklets (Görevcikler)	Softirq üzerine kurulu, tek seferde bir CPU'da çalışan geri çağırımlar.
TLB (Translation Lookaside Buffer)	Adres Çeviri Önbelleği (Sanal-fiziksel adres eşleşmelerini saklayan donanım önbelleği).
TLB Flush	Adres Çeviri Önbelleğinin temizlenmesi.
tmpfs (Temporary File System)	Bellek tabanlı geçici dosya sistemi.
vfork()	Bellek kopyalamadan hızlı süreç oluşturan sistem çağrısı.
wait()	Child süreçlerin bitmesini bekleyen sistem çağrısı.
x86_64	64-bit x86 işlemci mimarisi.
XInput (X Input Extension)	X Window System'de fare, klavye ve dokunmatik gibi giriş cihazlarını destekleyen eklenti.
XInput2 (X Input Extension 2)	Çoklu dokunmatik ve gelişmiş giriş cihazları için XInput'ın güncellenmiş sürümü.

İşçi Blokları Mahallesi Muhsin Yazıcıoğlu Caddesi No:51/C 06530 Çankaya/ANKARA

+90 (312) 289 92 22 - yte.bilgi@tubitak.gov.tr

TÜBİTAK - BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü (YTE)