İstanbul Sabahattin Zaim Üniversitesi Fen Bilimleri Enstitüsü Dergisi
Istanbul Sabahattin Zaim University Journal of the Institute of Science and Technology
https://dergipark.org.tr/tr/pub/izufbed

Açık Kaynak Kodlu Gerçek Zamanlı İşletim Sistemlerinin İncelenmesi

Seçkin CANBAZ[a1,*], Gökhan ERDEMİR[a2]

[a1] *İstanbul Sabahattin Zaim Üniversitesi Lisansüstü Eğitim Enstitüsü, Bilgisayar Bilimleri ve Mühendisliği Tezli Yüksek Lisans Programı, İstanbul, Türkiye*
[a2] *İstanbul Sabahattin Zaim Üniversitesi, Mühendislik ve Doğa Bilimleri Fakültesi, Elektrik-Elektronik Mühendisliği Bölümü, İstanbul, Türkiye*

İstanbul Sabahattin Zaim Üniversitesi Fen Bilimleri Enstitüsü Dergisi (2021) 3 (1): 30-37
https://doi.org/10.47769/izufbed.877030

[1] 0000-0001-7289-016X; [2] 0000-0003-4095-6333

YAYIN BİLGİSİ

Yayın geçmişi:
Gönderilen tarih: 8 Şubat 2021
Kabul tarihi: 18 Şubat 2021

Anahtar kelimeler:
Açık kaynak kod
Gerçek zaman
İşletim sistemi
RTOS
GPOS

ÖZET

Yazılım geliştirme süreçleri için zaman, bir yazılım tasarlanırken, birden fazla değişkene bağlı olduğunda hesaplanması zor olan, süreçten sürece değişen ve durdurulamaz bir kavramdır. Özellikle çok hızlı veri toplanması ve buna bağlı olarak hesap yapılması gereken yazılım mimarilerinde zaman kavramı mimari için belirleyici bir parametre olarak ön plana çıkmaktadır. Hayati önem taşıyan birçok yazılım tasarımında, örneğin trafik kazalarında hava yastığının açılmasını sağlayan sistemlerin tasarımında, 100 milisaniyelik bir zamanlama hatası, çok ciddi sonuçlar doğurabilir. Günlük hayatta kullandığımız bilgisayarlarda genel amaçlı işletim sistemi (GPOS) adını verdiğimiz işletim sistemleri bulunur. Bu tip işletim sistemlerinin kendisinde ya da üzerinde çalışan uygulamalarda oluşan hatalar, bir sonraki görevin gerçekleşmesi konusunda aksaklık meydana getirebilirler. Yukarıda verdiğimiz örnekte ise bu tip bir sorun kesinlikle kabul edilebilir değildir. Bu gibi sorunların önüne geçmek için uygulama tasarlanırken, gerçek zamanlı olmasını isteriz. Gerçek zamanlı uygulamalarda, uygulama üzerindeki her bir sürecin başlama ve bitiş anları kesin olarak tanımlanır. Hata veren bir süreç sistem kaynaklarını erişilmez kılamaz ve görev süresi bittiğinde kesinlikle sonlandırılır. Bu sayede hata verse dahi kendisinden sonra gelecek bir görev işleme alınır ve sistem her zaman ayakta kalır. Bu şekilde bir uygulama tasarlayabilmek ve çalıştırabilmek için kaynak kodunu istediğimiz şekilde düzenleyebildiğimiz işletim sistemlerine ihtiyaç duyarız. Bunlar genellikle açık kaynak kodlu işletim sistemleridir ve yukarıda bahsedilen şekilde gerçek zamanlı çalışan tiplerine gerçek zamanlı işletim sistemi (RTOS) adı verilir. Bu çalışmada, Linux'un farklı dağıtımlarının gerçek zamanlı mimari alt yapıları incelenmiştir ve performansları karşılaştırılmıştır.

Examination of Open Source Real Time Operating Systems

ARTICLE INFO

Article history:
Received: 8 February 2021
Accepted: 18 February 2021

Key words:
Open source code
Real time
Operating system
RTOS
GPOS

ABSTRACT

For software development processes, time is an unstoppable concept that is difficult to calculate when a software is designed when it depends on more than one variable, it changes from process to process. Especially in software architectures where very fast data collection and calculations need to be made, the concept of time stands out as a determining parameter for the architecture. A timing error of 100 milliseconds can have serious consequences in many vital software designs, for example in the design of systems that enable the airbag to be deployed in traffic accidents. The computers we use in daily life have operating systems that we call general purpose operating system (GPOS). Errors that occur in such operating systems themselves or in applications running on them may cause problems in the performance of the next task. In the example we gave above, this type of problem is definitely not acceptable. While designing the application to avoid such problems, we want it to be in real time. In real time applications, the starting and ending moments of each process on the application are precisely defined. A faulty process cannot make system resources inaccessible and will definitely be terminated when the task time expires. In this way, even if it fails, a task that will come after it is processed and the system always remains standing. In order to design and run an application in this way, we need operating systems that we can edit the source code as we want. These are usually open source operating systems, and as mentioned above, the real time operating systems are called real-time operating systems (RTOS). In this study, real time architectural infrastructures of different distributions of Linux are examined and their performances are compared.

*Sorumlu yazar.
E-mail adresi: seckin@canbaz.info (Seçkin CANBAZ)

1. Giriş

Hayatımız, gelişmekte olan birçok teknoloji sayesinde her gün daha fazla kolaylaşmakta ve insanoğlu kendisini daha güvenli, daha rahat ve her şeyin daha kolay ulaşılabilir olduğu bir dünya düzenine adapte etmeye devam etmektedir. Bundan 15-20 yıl önce hayatımızda olmayan giyilebilir teknoloji, havada, denizde ve karada kullanılabilen otonom araçlar ve daha birçok yenilik, bilgisayar bilimlerine olan ilginin 7'den 70'e artması ve bu alanda profesyonel olarak çalışan kişi sayısının yükselmesiyle olmuştur. Günümüze kadar hızla gelişmekte olan mikro denetleyiciler, IoT (Internet Of Things-Nesnelerin İnterneti) cihazlar gibi birçok teknoloji, bu yeniliklerin sağlanmasında etkili bir rol oynamıştır.

Açık kaynak koda olan ilginin artması ve açık kaynak kod ile üretilen teknolojinin çok hızlı gelişmesi sayesinde, açık kaynak kodlu yazılımlar ve açık kaynak kodlu işletim sistemleri günümüzde çok önemli hale gelmiştir. Savunma sanayinden evimizde kullandığımız akıllı televizyonlara kadar birçok alanda kullanılan açık kaynak kodlu yazılımlar ve işletim sistemleri; tasarımı, geliştirilmesi ve son kullanıcıya hazır hale getirilene kadar birçok süreçten geçmektedir. Bu süreçlerin sonunda çok büyük emeklerle tasarlanan açık kaynak kodlu işletim sistemlerinde oluşan zaman problemi nedeniyle açık kaynak kodlu gerçek zamanlı işletim sistemlerinin tasarlanmasına da ihtiyaç duyulmuştur.

Bu yazımızda temel kavramlardan başlayarak, açık kaynak kodlu gerçek zamanlı işletim sistemlerinin ne olduğunu ve nasıl bir yapıda olduğunu inceleyeceğiz.

2. Temel Kavramlar

Uygulama yazılımları ile donanım birimlerinin yönetimini sağlayan ve kullanıcıya bunları bir arayüz ile gösteren sistem yazılımlarının bütününe işletim sistemi (Operating System–OS) denir.

İşletim sistemleri denince akla ilk gelen bilgisayar olmasına rağmen günümüzde akıllı telefonlar, akıllı saatler, akıllı tahtalar, tabletler gibi birçok alanda kullanılabilmektedirler. İşletim sistemlerinin temel görevleri arasında; işlemci-merkezi işlem birimini yönetmek, bellekleri yönetmek, donanım birimlerini yönetmek, dosya ve klasör yönetimini sağlamak, sistem güvenliğini sağlamak ve grafik arayüz kontrolünü sağlamak gibi görevler sayılabilir.

İşletim sistemleri açık kaynak kodlu ve kapalı kaynak kodlu olabilirler.

2.1 Açık-Kapalı Kaynak Kodlu Yazılım Kavramları

Açık kaynak kodlu yazılım, kaynak kodu isteyen herkese açık olarak sunulan yazılımlara denir. Açık kaynak kodlu yazılımlar, kullanıcılara yazılımı istedikleri şekilde değiştirme özgürlüğü sunar. Bu tür yazılımlar güvenilir, hızlı, sağlam, uyarlanabilirdir ve ücretsiz olarak dağıtılırlar. Açık kaynak kodlu olmalarından dolayı, yazılım ile ilgili oluşan sorunlarda ya da yazılımın geliştirme aşamasında, binlerce yazılım uzmanı aynı anda çalışabilirler. En çok bilinen yazılımlar Linux ve Open Office'dir.

Yazılım oluşturulurken kullanılan kaynak kodlarının gizli tutularak, kullanıcılar ile paylaşılmadan dağıtılan yazılımlara kapalı kaynak kodlu yazılım denir. Bu tür yazılımlar tasarımcısı hariç kimse tarafından değiştirilemez. Genellikle ücretli olarak dağıtılırlar. En bilinen yazılım örnekleri Microsoft Windows ve MS Office olarak sayılabilir.

2.2 Unix İşletim Sistemi

Unix birçok işletim sisteminin tasarımına öncülük etmiş bir işletim sistemidir. Tasarlanması 1970'li yıllara dayanır. Unix o dönemde sadece programcılar tarafından kullanılan küçük, esnek bir işletim sistemiydi. Unix üst düzey bir dil kullanarak yazılan ilk işletim sistemidir. Farklı donanımlara sahip cihazlarda çalışacak şekilde ayarlanabilir (Spinellis, D. 2017). Unix, üniversite üyesi olmayan kullanıcılar için 20.000USD ve üniversite üyesi olan kullanıcılar için ise 150USD gibi yüksek bir lisans ücretine sahiptir. Bu sebeple Unix işletim sisteminden esinlenerek Linux gibi ücretsiz dağıtılan birçok işletim sistemi tasarlanmıştır. Ayrıca günümüzde kullanılan MacOS X işletim sistemlerinin de temelini oluşturmaktadır (Spinellis, 2017).

2.3 GNU is not Unix (GNU) ve General Public Licence (GPL) Kavramları

2.3.1 *GNU*

GNU'nun açılımı "GNU is not Unix" yani "GNU Unix değildir" şeklinde tasarlanmıştır. Verilen bu isim aslında tasarımın ve mimarinin altındaki tüm gizemi açığa çıkartmaktadır. Tasarımı Unix ile çok benzemesine rağmen tasarım aşamasında kullanılan çekirdek, sistem araçları, kütüphaneleri ve son kullanıcı yazılımlarının hepsi GNU Tasarısı kapsamında geliştirilmiştir. Kendisi tamamen özgür bir yazılımdır ve Unix'e ait hiçbir kod bulunmamaktadır. Günümüzde halen tamamlanmış GNU sistemi bulunmamaktadır (Stallman, 1985).

Çekirdeği GNU Hurd'dur fakat henüz tamamlanmamıştır. Linux'un devreye girmesiyle birçok GNU Hurd kullanıcısı Linux çekirdeğine geçiş yapmıştır. Linux çekirdeğini kullandıkları için birçok kullanıcı sistemlerini Linux olarak tanımlamaktadır (Stallman, 1985).

2.3.2 *GPL*

Genel Kamu Lisansı kısaca GPL bir özgür yazılım lisansıdır. Geliştiricisi Richard Stallman'dır. Lisansın en önem verdiği konu, bu lisansa sahip yazılımların kaynak kodları ile dağıtılması, açık kaynak kodlu olmasıdır. Kullanıcılar kodlar üzerinde değişiklik yapabilir ve bunları ücretli şekilde dağıtabilirler. Dağıtım yapabilmelerinin tek şartı GPL ile lisanslanmasıdır (Engelfriet, 2009).

3. Açık Kaynak Kodlu İşletim Sistemleri

Kaynak kodu herkese açık olan, kullanıcılarına kaynak kodlarını değiştirme özgürlüğü veren işletim sistemlerine açık kaynak kodlu işletim sistemleri denir. Aşağıda açık kaynak kodlu işletim sistemlerine birkaç örnek verilmiştir.

3.1 Linux

Unix'in ücretli olmasından sonra Prof. Andrew Tanenbaum "Minix" adını verdiği ve Unix türevi olan bir işletim sistemi ortaya çıkardı. Daha sonra Linus Torwalds, çalıştığı bir projenin Unix çekirdeği gibi çalışabileceğini fark etti ve 1991 yılında, Linux adını verdiği bir işletim sistemi üzerinde çalıştığını duyurdu. Daha sonra Linux, internet üzerindeki birçok programcı yardımıyla geliştirildi (Hasan, 2002).

Linux aslında Kernel diye adlandırılan çekirdeğin adıdır. Çekirdek, donanım ve yazılım arasındaki köprüdür. İşletim sistemi başlatılırken belleğe yüklenir, işletim sistemi kapatılıncaya kadar bellekte varlığını sürdürmeye devam eder.

İşlemcinin görevlerini organize eder. Kısacası tüm yapılan işlemlerden sorumludur.

İşletim sistemi yapısında kullanıcı ile ilişkilendirilen bir yapı daha bulunmaktadır. Bu yapıya da kabuk (shell) denir. Kabuğun görevi kullanıcı ile çekirdeğin bağlantısını sağlamaktır. Bilgisayara girilen komutları yorumlayarak çekirdeğe iletir ve çekirdeğin çıktılarını da kullanıcıya sonuç olarak gösterir. Birçok Linux kabuğu bulunmaktadır. En bilinen ve en sık kullanılanı "Bash Shell"dir. Ash, tcsh, zsh gibi farklı kabuklarda mevcuttur.

Linux tabanlı işletim sistemleri şu anda kişisel kullanımı karşılayabilecek yeterliliğe sahiptir. Günümüzde kişisel kullanımdan çok yazılım geliştirme ve internet sunucusu gibi farklı alanlarda kullanılmaktadır. Ücretsiz ve açık kaynak kodlu olmasının bu ihtiyaçları karşılamasının yanı sıra, yüksek performanslı ve güvenilir olması tercih edilmesinin diğer sebeplerindendir. Günlük kullanımda Windows'a göre daha az tercih edilmektedir. Bunun temel sebeplerinden bazıları, bilgisayar kullanmaya yeni başlayan bir kullanıcı için ya da bilgisayar kullanmayı çok iyi bilmeyen bir kullanıcı için arayüzü biraz karmaşık gelebilir. Oluşan sorunları çözmek biraz bilgi gerektirdiğinden ve alışması zaman aldığından rakibi karşısında bu konuda biraz yetersiz kalmaktadır. Linux çekirdeği kullanan bazı sürüm işletim sistemleri arayüzünü Windows'a benzeterek bu sorunun biraz olsun üstesinden gelebilmişlerdir.

Distrowatch web sitesi referans alınarak en çok kullanılan Linux tabanlı işletim sistemleri hakkında aşağıda örnekler paylaşılmıştır.

3.1.1 *Ubuntu*

Eski bir Afrika kelimesidir. "Başkalarına insanlık" anlamına gelir. Kelime anlamından da anlaşılacağı üzere, Ubuntu'nun ana fikri yazılım ücretsiz olarak sağlanmalı, yazılım araçları her bir insan tarafından özgürce kullanılabilmeli ve kullanırken hiçbir engelle karşılaşılmamalı ve en önemlisi, istedikleri gibi özelleştirebilmelidirler. Ubuntu'nun ilk resmi sürümü 2004 yılı, Ekim ayında ortaya çıkmıştır. 2 yılda bir LTS-Long Term Support (Uzun Süreli Destek) adını taşıyan, adından da anlaşılacağı üzere uzun bir süre güncelleme ve yazılım desteği sunulacağı taahhüt edilen bir sürüm paylaşılmaktadır. 6 ayda bir de yeni bir dağıtım paylaşmaktadır. Dünya üzerinde en yaygın kullanılan Linux çekirdeğini kullanan dağıtımdır. Birçok büyük bilgisayar markası cihazlarını Ubuntu yüklü şekilde piyasaya sürmektedir (Tabassum & Mathew, 2014).

3.1.2 *Debian*

Linux çekirdeğini kullanan yaygın işletim sistemlerinden bir diğeri de Debian'dır. Özgür bir şekilde işletim sistemi oluşturmaya karar veren ve tamamı gönüllülerden oluşan bir topluluğun ortaya çıkardığı bir işletim sistemidir. Debian üzerinde 59000'den fazla kullanıma hazır, ücretsiz yazılım ile gelmektedir. Temelinde Linux çekirdeği vardır. Üzerinde temel araçları ve daha sonra kullanabileceğiniz programlar bulunur. En üstte ise her şeyi uyum içerisinde çalıştıran Debian vardır. Mimarisi bir kule şeklinde inşa edilmiştir (Claes, Mens, Di Cosmo & Vouillon, 2015).

3.1.3 *Mint*

Linux Mint, Ubuntu ve Debian'a göre daha modern, tasarımı daha şık, kullanımı daha kolay ve rahat bir işletim sistemidir. Debian ve Ubuntu altyapısını kullanmasına rağmen paketinden tam multimedya desteği ile çıkar ve kullanımı son derece kolaydır. Kullanıcıların fikirleri Linux Mint'in gelişmesi açısından son derece önemlidir. Çok az bakım gerektirmesine rağmen güvenilir bir işletim sistemidir. Tasarımı ve menüleri Windows işletim sistemi ara yüzü ile çok benzerlik göstermektedir (Ovadia, 2013).

3.1.4 *Fedora*

Ücretsiz ve açık kaynaklı bir lisans altında dağıtılan yazılımları içerir ve bu tür teknolojilerin öncüsü olmayı hedefler. Varsayılan masaüstü GNOME masaüstü ortamıdır ve varsayılan arayüz Gnome kabuğudur. Xfce, Mate, LXDE, KDE ve Cinnamon gibi diğer masaüstü ortamları da mevcuttur. Ayrıca Fedora spinleri adı verilen varyasyonları da vardır ve bunlar alternatif masaüstü ortamları sunarak veya oyun, tasarım, güvenlik, bilimsel bilgi işlem ve robotik gibi belirli alanları hedefleyen belirli yazılım paketi setleriyle birlikte sunulur (Tyler, 2006).

3.1.5 *Pardus*

TÜBİTAK ve ULAKBİM tarafından ortaklaşa olarak geliştirilen, özgür ve açık kaynak kodlu bir GNU/Linux dağıtımıdır. Kurulmaya gerek kalmadan da kullanılabilir. Türkçe dışında çok az dil desteği bulunmaktadır. Debian altyapısını kullanmaktadır. Hızlı, güvenli ve kullanımı kolaydır. Kamu kurum ve kuruluşları gibi birçok alanda kullanılabilmesi için ihtiyaçları karşılayacak birden fazla açık kaynak kodlu alt projeleri bulunmaktadır (Karakoç & Varol, 2016).

3.2 Cosmos (C# Open Source Managed Operating System)

Cosmos bir işletim sistemi değil, bir işletim sistemi geliştirme kitidir. Çoğunluğu C# ve küçük bir kısmı X# kullanılarak oluşturulmuştur. İki sürümü vardır. Userkit (Kullanıcı Kiti) ve Devkit (Geliştirici Kiti). Geliştirici kiti, Cosmos'u geliştirmek için kullanılan kittir. Kullanıcı kiti ise kendi işletim sistemini oluşturmak isteyen kullanıcıların kullandığı kittir. Cosmos kendisi tam bir işletim sistemi olmamasına rağmen, kendi işletim sistemini oluşturmak isteyen kullanıcılara yardımcı olan bir kittir (Cosmos W.S., 2021).

3.3 FreeDOS

DOS (Disk Operating System) grafik arayüzü bulunmayan, komut tabanlı işletim sistemidir. FreeDOS ise, MS-DOS'un Microsoft tarafından geliştirilmesinin durdurulmasından sonra, GNU kapsamında açık kaynak kodlu olarak geliştirilmiş bir DOS tabanlı işletim sistemidir. DOS işletim sistemi komutlarla çalışan bir işletim sistemi olduğu için FreeDOS kurulu bir bilgisayarda çalışırken DOS komutlarına hakim olmanız gerekmektedir (Villani, 1996).

3.4 Genode

Hem ücretsiz hemde açık kaynak olarak dağıtılan Genode, Unix gibi bir işletim sisteminden türetilmeyen birkaç açık kaynak kodlu işletim sisteminden biridir. Genode tasarımcılarının felsefesine göre kod ne kadar küçük ve basit olursa kodun doğruluğunu ve güvenilirliğini sağlamak o derece kolay olur. Küçük bileşenlerden, daha büyük ve daha karmaşık uygulamalar oluşturarak yazılımı ortaya çıkartır. Her katman bir ebeveyn-çocuk ilişkisi içerisinde ilerler. Her

katman, alt katmanların kaynaklarını düzenleyebilir ve yönetebilir, bu sayede ilkelerin her düzeyde uygulanabileceği hiyerarşiler oluşturulabilir. Böyle katı bir yönetim sisteminde güvenlik açısından günümüz işletim sistemlerine göre daha güvenlidir (Feske, 2015).

3.5 Ghost İşletim Sistemi

Ghost kişisel bilgisayarlar için geliştirilen açık kaynak kodlu, ücretsiz bir işletim sistemidir. 2014 yılında hobi olarak geliştirilmeye başlanmıştır. Geliştirilmesi halen devam etmektedir ve Intel x86 platformuyla uyumludur. Ghost çekirdeğini kullanır. Geliştirmenin çoğu C++ altyapısı ile yapılmıştır (Ghost Kernel W.S., 2021).

3.6 Android

Hepimizin günlük hayatta cep telefonlarında ya da tabletlerde kullandığı, en bilinen açık kaynak kod işletim sistemidir. 2003 yılında Android Inc. olarak kurulan şirket Linux çekirdeğini kullanarak oluşturduğu işletim sistemini cep telefonlarına uyarladı. 2 yıl sonra bu şirket Google tarafından satın alındı ve tasarımcıları da Google'da çalışmaya başlayarak projelerini daha hızlı geliştirme imkanına sahip oldular. Kaynak kodu Apache Lisansı altında lisanslanan Android Açık Kaynak Projesi (AOSP) olarak bilinir. Android için ana donanım platformu ARM (ARMv7 ve ARMv8-A) mimarileridir. 2012 yılından itibaren cep telefonlarında ve tabletlerde Intel işlemcili Android cihazlar görülmeye başlanmıştır. Günümüzde x86 ve x64 uyumluluğu sağlanmıştır. Cihaz üreticileri kendi donanımlarında çalışacak şekilde kaynak kodlarını özelleştirirler. 2020 yılı itibariyle Linux çekirdeğinin 4.4, 4.9 veya 4.14 sürümlerini kullanırlar (Krajci & Cummings, 2013), (Rani & Krishna, 2014).

3.7 Robot İşletim Sistemi [Robot Operating System (ROS)

Robot işletim sistemi denilince ilk akla gelen bir işletim sistemi türü olsa da aslında gerçek bir işletim sistemi değildir. Linux tabanlı bir işletim sistemi üzerine kurulan BSD lisanslı açık kaynak kodlu bir yazılımdır. Yazılımın görevi, bilgisayar üzerinden robot bileşenlerini kontrol etmemizi sağlar. Kısacası robot ve programcı arasında iletişimi sağlayan bir yapıdır. ROS kendi içerisinde barındırdığı kütüphaneler ve geliştiricilerin sağladığı kütüphaneler ile desteklediği robot sayısını günden güne arttırmaktadır (Quigley, Conley, Gerkey, Faust, Foote, Leibs & Ng, 2009), (Wei, Shao, Huang, Chen, Guan, Tan & Shao, 2016).

ROS genel olarak iki dil kullanılarak geliştirilmiştir. Bunlar C++ ve Python'dur. Robotik uygulamalar geliştirilirken en çok tercih edilen diller C++ ve Python'dur. C++ kodu yazarken roscpp kütüphanesini, python kodu yazarken rospy kütüphanesini kullanmak gerekir. İletişim katmanının dil katmanı altında olmasından dolayı farklı dillerde yazılmış alt programlar, birbirleri arasında iletişim kurabilmektedirler (Quigley, Conley, Gerkey, Faust, Foote, Leibs & Ng, 2009), (Wei, Shao, Huang, Chen, Guan, Tan & Shao, 2016).

Tasarlayacağınız robot basit Arduino kartı ile istediklerinizi yapamıyorsa, ROS kullanmanız sizin açınızdan çok daha faydalı olacaktır. Robotunuzda birçok farklı sensör varsa ya da birden fazla robot kullanarak bir robot sürüsü ile çalışmak istiyorsanız ROS daha hızlı, daha anlaşılır ve daha net veriler ile çalışmanıza imkan sağlar

ROS üzerinde bir temel bilgi ile birden fazla robot üzerinde çalışabilirsiniz ve yeni bir robot ile çalışırken herşeye yeniden başlamanız gerekmez. Çalışılan robot için gerekli paketlerin çoğu kütüphanelerde bulunduğu için, örneğin yörünge hesaplarken daha önceden hazırlanmış paketleri hızlıca bulup kullanabilirsiniz. ROS, Rviz ve Gazebo gibi birçok similasyon aracına sahiptir. Bir drone yardımı ile 3D olarak haritalandırdığınız bir ortamı Gazeboya ekleyebilir, bu ortam içerisinde kullanmak istediğiniz robotun özelliklerini oluşturarak Gazeboda robotunuzu gerçek ortamdaymış gibi test edebilirsiniz. Robot işletim sistemini sadece birkaç dakika içerisinde kurabilir ve hemen kullanmaya başlayabilirsiniz.

4. Açık Kaynak Kodlu Gerçek Zamanlı İşletim Sistemleri

Günümüzde kişisel cihazlarımızda kullandığımız işletim sistemleri, yani gerçek zamanlı olmayan işletim sistemleri, GPOS (Normal General Purpose Operating System) Normal Genel Amaçlı İşletim Sistemi olarak adlandırılır. Genel amaçlı işletim sistemlerine verilebilecek en bilinen örnekler, Windows, MacOs gibi işletim sistemleridir. GPOS'ta verimlilik esastır. GPOS verimliliği, birim zamanda tamamlanan toplam iş sayısı olarak adlandırılır. Dolayısıyla yüksek öncelikli bir görevi gerçekleştirmek yerine, düşük öncelikli birden fazla görev gerçekleştirilerek sistem daha verimli hale getirilir. Genel amaçlı işletim sistemleri gecikme ve senkronizasyon sorunları yaşabilirler. Bu yüzden zamana duyarlı uygulamalar için tercih edilebilir değildirler. İşlemler sırasında oluşacak bir gecikme, büyük ve hayati sonuçlar doğurabilir. Uygulamalarda oluşacak herhangi bir gecikmede, işlem sonlandırılamayabilir ve bu da diğer işlemlerin sıraya alınmasında gecikmeye sebep olabilir. Bu yüzden genel amaçlı işletim sistemleri, gerçek zamanlı işletim sistemlerine göre daha güvensiz olarak kabul edilir.

İşletim sistemlerinin temel amacı, sistem üzerinde bulunan uygulama yazılımlarını ve donanım kaynaklarını yönetmektir. Tüm bu işlemleri yaparken işleme alınan bir görevi, belirli bir maksimum süre sonunda kesinlikle sonlandıran işletim sistemlerine, gerçek zamanlı işletim sistemleri (RTOS) denir. Gerçek zamanlı işletim sistemlerinde her bir görevin ne zaman biteceği kesin olarak bilindiği için, görevlerle ilgili bir aksama olmaz ve belirli bir anda yerine getirilmesi gereken bir görev, hiçbir gecikme olmaksızın o anda işleme alınır. Günümüzde savunma sanayinde, nükleer reaktörlerde, tıbbi cihazlarda, uzay araştırmalarında kullanılan araçlar gibi birçok alanda gerçek zamanlı işletim sistemleri kullanılmaktadır.

4.1 Zamana Bağımlılıklarına Göre RTOS Çeşitleri

4.1.1 *Zor Gerçek Zamanlı-Hard Real Time*

Bu tarz işletim sistemlerinde işleme alınan görevlerin bitirilme zamanında meydana gelen gecikmelere tolerans oldukça azdır. İşleme alınan görev zamanında bitirilmezse, sistem için yıkıcı bir etki oluşturabilirler. Örneğin, araç kontrol sistemleri (hava yastığı), havayolu kontrol sistemi vb. (Kotecha & Shah, 2008).

4.1.2 *Yumuşak Gerçek Zamanlı- Soft Real Time*

Görevin bitmesi için belirtilen süre çok önemli değildir. Gecikmeler tolere edilebilir. Biraz RTOS ve GPOS arasında bir işletim sistemidir. Kameralar, akıllı telefonlar vb. (Kotecha & Shah, 2008).

4.1.3 *Kesin Gerçek Zamanlı- Firm Real Time*

Bu tür işletim sistemlerinde görevlerin kesinlikle belirtilen zamanda bitirilmesi gerekmektedir. Zamanında bitirilmeyen görevler iptal edilir. Kaçırılan birkaç görev kaliteyi düşürebilecek iken, kaçırılan çok fazla görev sistemi çökertebilir (Kotecha & Shah, 2008).

4.2 Çekirdek Mimarilerine Göre RTOS Çeşitleri

4.2.1 *Tek Parçalı Çekirdek-Monolithic Kernel*

Bütün işletim sistemi görevlerinin çekirdekte çalıştırıldığı bir çekirdek yapısı tasarımıdır. Aygıt sürücüleri, dosya yönetimi, ağ iletişimi ve grafik yığını gibi işlemlerin tamamı çekirdekte çalışır fakat uygulamalar kullanıcı alanında çalışır. Uygulamaların çekirdekte çalışmamasının en büyük avantajı, hatalı kullanıcı kodlarının çekirdek yapısına zarar verememesidir. Fakat bu yapıda çalışan bir işletim sistemi çekirdeğinde meydana gelecek herhangi bir hatadan tüm sistem etkilenebilir. Diğer çekirdek türlerine göre hızlı olmasına rağmen, kod fazlalığı olması nedeniyle çok karmaşıktır. Çekirdek üzerinde yapılan en ufak bir değişiklikte tüm çekirdeğin yeniden derlenmesi gerekir. Yüksek hafıza ihtiyacı vardır. Örneğin; Unix çekirdekleri (BSD), Unix benzeri çekirdekler (Linux), DOS. (Roch & Wien, 2004).

4.2.2 *Mikro Çekirdek-Micro Kernel*

İşletim sistemi görevlerinin çekirdekte değil, kullanıcı alanında ayrı ayrı çalıştırıldığı bir tasarımdır. Bellek yönetimi, sürücüler, ağ iletişimi, dosya sistemi gibi süreçler birbirleriyle iletişim kurarak haberleşir ve mesaj trafiği çok olduğu için daha yavaştır fakat herhangi bir modülde oluşan bir hata diğer modülleri etkilemez ve herhangi bir modül çöktüğünde sistem tamamiyle çökmez. Çöken modül hızlıca yeniden başlatılabilir. Tek parçalı çekirdeğe göre daha az hafızaya ihtiyaç duyar, yeni bir sürücü ya da bileşen eklendiğinde çekirdeğin tamamını yeniden derlemeye gerek kalmaz. Daha az kod bulunduğu için daha az karmaşıktır. Çekirdeğin görevi, birimler arası iletişim ve süreçleri sıralamaktır ve sadece çekirdeğin tüm sisteme erişim izni bulunmasından dolayı daha güvenlidir. Mikro çekirdeğe örnek olarak QNX, Minix, Symbian gibi işletim sistemleri örnek verilebilir (Roch & Wien, 2004).

4.3 Açık Kaynak Kodlu Gerçek Zamanlı Bir İşletim Sisteminin Çalışma Prensipleri

Bir RTOS görevleri zamanlayabilmeli, görevlerin son teslim tarihlerini tutturabilmeli, oluşan hataları kurtarabilmeli, harici donanımları görevlere dahil edebilmeli ve düşük öncelikli görevleri değiştirebilmelidir. Bu seçenekler bir gerçek zamanlı işletim sisteminden beklenen özelliklerdir. İşletim sisteminin çekirdeği görev zamanlaması, görev dağıtımı ve görevler arası iletişimi sağlar. Gömülü sistemlerde çekirdek RTOS görevi görebilir fakat ticari olarak kullanılan bazı RTOS'lar genel amaçlı işletim sistemlerinin tüm işlevlerini yapmasını gerektirebilir.

4.3.1 *RTOS'un Temel Gereksinimleri*

4.3.1.1 *Öncelikli ve çok görevli:* Zamanlayıcı kaynağı en çok ihtiyaç duyulan göreve yönlendirebilmeli ve birden çok görevi destekleyebilmelidir (Baskiyar & Meghanathan 2005).

4.3.1.2 *Son tarih dinamik olarak belirlenebilmeli:* Bir başka görevin ön siparişine ulaşabilmek için, sıraya alınan görevin en erken son teslim tarihini dinamik olarak belirlemesi gerekmektedir. Bunu yapabilmek için ise kaynak atama için kullanılan öncelik seviyelerinin kullanılması gerekmektedir (Baskiyar & Meghanathan 2005).

4.3.1.3 *Öngörülebilir senkronizasyon:* İş parçacıklarının kendi aralarında iletişim kurmaları gerekmektedir. Bunun için ise görevler arası iletişim ve senkronizasyon mekanizmaları gereklidir (Baskiyar & Meghanathan 2005).

4.3.1.4 *Yeterli öncelik seviyeleri:* RTOS'un sağlıklı ve etkili çalışabilmesi için görevlerin öncelikleri olması ve bu önceliklerin görev oluşturulurken tanımlanması gereklidir. Görevlerin düzgün tanımlanabilmesi için yeterli öncelik seviyelerine sahip olması gerekmektedir. Öncelik seviye sayısı az olan bir sistemde görevler birbirlerine üstünlük sağlayamayacağından, görevlerde gecikme ya da hatalı sonuç alma sorunları olabilir. Sistemde çalışan bir görevden daha öncelikli bir görev oluşabilmeli ve oluştuğunda, sistemin çalışan görevi beklemeye alarak, kaynaklarını öncelikli olan göreve aktarması gerekebilir (Baskiyar & Meghanathan 2005).

4.3.1.5 *Önceden tanımlı gecikmeler:* Sonuç değerinde gecikme olmaması için, sistem çağrılarının gecikmeleri önceden tanımlanmalıdır. Görev değiştirme gecikmesi, yürütülen görevin sona erip yeni göreve geçme süresi, kesilen bir görevin son işleminin yerine getirilmesi ile yeni görevin ilk talimatı arasında geçen sürelerin önceden tanımlanarak görevlerin başlangıcıyla bitişi arasındaki sürede oluşan hata toleransı en aza indirgenir (Baskiyar & Meghanathan 2005).

4.3.2 *RTOS'ta Bellek Yönetimi*

RTOS kullanılacak uygulama tiplerine göre bellekleri iki sekilde kullanır.

4.3.2.1 *Statik bellek yönetimi:* Sistemin boş olan belleği, sabit boyutlu bellek bloklarından oluşan bir havuza bölünerek, görevlerin bu kısımları talep etmesini bekler. Belirli bir bellek bloğunu kullanan bir görev tamamlandığında, bloğun havuza geri dönmesi beklenir. Yüksek öncelikli görevler ve düşük öncelikli görevler için farklı bellek havuzları bulunmaktadır. Yüksek öncelikli havuz, sistemin en yoğun durumunu karşılayabilecek şekilde boyutlandırılmalıdır. Geri kalan bellek boyutu ise düşük öncelikli görevler için ayrılır. Düşük öncelikli görevler ise kendi bellek havuzlarını doldurduklarında, yeni bir görevin çalışabilmesi için, çalışan görevlerden en az birinin bitmesi beklenir. Biten görevlerin kullandığı bellek bloğu havuza geri döner ve yeniden kullanılabilir hale gelir (Shah & Shah, 2016).

4.3.2.2 *Dinamik bellek yönetimi:* Bu yönetim şeklinde bellek görev gerçekleşmeye başlarken tahsis edilir. Çekirdek ve uygulamalar birlikte çalışır. Dinamik bellek yönetimi iki çesittir. Birincisi manuel bellek yönetimi; yazılımcı bellek üzerinde doğrudan kontrole sahiptir ve belleğin ayrılacağı ve boşaltılacağı zamanı belirleyebilir. İkincisi ise otomatik bellek yönetimidir. Burada otomatik bellek yöneticileri vardır. Görev sona erdiğinde kullanılmayan kısımları geri dönüştürerek belleğin kullanımı sona ermiş kısımlarını tekrar kullanıma açarlar. Otomatik bellek yönetimi daha az karmaşık görünsede işlemci üzerine büyük miktarda yük bindirirler ve bazen belirlenemeyen şekilde davranarak bellekte sızıntılara neden olabilirler (Shalan, 2003), (Shah & Shah, 2016).

4.3.3 *RTOS'ta Görev Planlama*

Görev planlamaları için çok çeşitli değişkenler vardır. Öncelik atanabilir ya da atanmayabilir, statik ya da dinamik bellek kullanılabilir. Zamanlama hesaplanırken, kaynak gereksinimleri ve görevlerin yürütülme önceliği olmak üzere yaygın olarak kullanılan iki kısıtlama bulunmaktadır. Bir görev planlanırken göz önüne alınması gereken bazı parametreler vardır. Bunlar; en kötü durum uygulama süresi, ortalama uygulama süresi, görevin sıradan çıkıp işleme alınmasına kadar geçen zaman, periyodik bir işlem yapılacaksa işlemin periyodu göz önünde bulundurulmalıdır.
Programlama yaparken program uzunluğu ve ortalama gecikme sürelerini en aza indirerek olası hataların önüne geçmeye çalışılır. Bazen de buna alternatif olarak ortalama erken başlama süresi ve değerlerin son teslim süresine denk gelen varış süresini en üst düzeye çıkartarak bu sorun aşılabilir. Bu tercihlere göre bir takım görev planlamaları geliştirilmiştir.

4.3.3.1 *Statik tabloya dayalı planlama:* İşleme alınacak tüm görev ve özellikler, önceden belirlenmiş bir tablo dahilinde çalışırlar. Görevler bu tablo dahilindeki sıra ile gönderilirler. Her döngüde görevlerinin maksimum gereksinimleri bilinmelidir ve sisteme yeni bir görev eklendiğinde hesaplamalar yeniden yapılmalı, tablo yeniden oluşturulmalıdır (Rhodes, 2002), (Baskiyar & Meghanathan 2005).

4.3.3.2 *Statik önceliğe dayalı, önleyici planlama:* Görevler, önceliklere göre dinamik olarak gönderilir. Yeni bir görev gönderilebilmesi için sistemin önceki görevin çıktısına yanıt vermesi gereklidir. Sistemin yanıt süresi, ard arda gelen görevler arasındaki süreden daha kısa olmalıdır (Rhodes, 2002), (Baskiyar & Meghanathan 2005).

4.3.3.3 *Dinamik planlama:* Görev analizi dinamik olarak yapılır ve görev uygulanabilir olduğunda kabul edilir. Eğer görevin yerine getirilme süresi, en kötü durumda yürütülme süresi üzerinde ise sistemi riske atabileceği için kabul edilmez (Rhodes, 2002), (Baskiyar & Meghanathan 2005).

4.3.3.4 *En iyi dinamik performansa dayalı planlama:* Görevden önce herhangi bir analiz yapılmaz. Görev son teslim tarihinden en kısa olanını kaçıran görev iptal edilir. Sistemlere kolay uygulanabilir.

4.3.3.5 *Hata toleransı ile planlama:* Görevde bir sorun meydana gelmezse görev çalıştırılmaya devam eder. Eğer çalışan görevde bir sorun olursa, çalışan görev için tanımlanmış son ana kadar bir sonraki görevi çalıştırır (Rhodes, 2002), (Baskiyar & Meghanathan 2005).

4.3.3.6 *Kaynak geri alma ile planlama:* Yürütülen görev, planlanan süreden daha önce sonuçlanabilir. Bu durumda görev dağıtıcısı bu boşlukları geri almayı deneyebilir (Rhodes, 2002), (Baskiyar & Meghanathan 2005).

5. Açık Kaynak Kodlu Gerçek Zamanlı İşletim Sistemleri (Rtos) ile Genel Amaçlı İşletim Sistemlerinin (Gpos) Bir Örnek Üzerinden Karşılaştırılması

Genel amaçlı işletim sistemleri, günlük hayattaki işlerimizin yanı sıra yazılım geliştirme, tasarım programları gibi teknolojiyi geliştirmeye yönelik birçok alanda bizlere fayda sağlamaktadır. Açık kaynak kodlu gerçek zamanlı işletim sistemleri ise daha çok kesin sonuçlara ihtiyacımız olan ve hata toleransının sıfıra yakın olduğu alanlarda tercih edilmektedir. Kullanılacak nesne tanıma algoritması, yukarıda bahsedilen alanların ortak kesişimi olan, en az hata payı ile nesneleri tanıyan, yazılım geliştirme ve yeni yazılımların tasarlanmasında bize kolaylık sağlayan bir algoritmadır. Bahsedilen nesne tanıma algoritmasını kullanarak açık kaynak kodlu gerçek zamanlı işletim sistemleri (RTOS) ve açık kaynak kodlu genel amaçlı işletim sistemlerinin (GPOS) sabit bir donanım üzerindeki donanım kaynaklarını nasıl kullandığını inceleyeceğiz.

5.1 Çalışmada Kullanılan Araçlar

5.1.1 *Donanım Kaynakları*

İşlemcisi Intel(R) Core(TM) i7-4500U, belleği 8 GB, ekran kartı Nvidia GeForce 840M ve Samsung 850 Evo model bir SSD'si bulunan bilgisayar bu çalışmada kullanılmıştır.

5.1.2 *Yazılım Kaynakları*

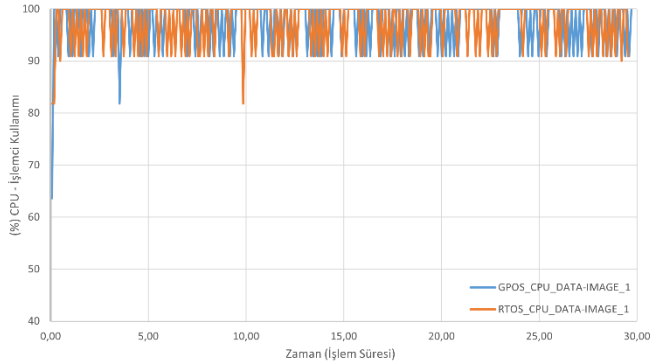
İşletim sistemi olarak Pardus işletim sistemi kullanıldı. Pardus çekirdeği, pardus kütüphanesinde bulunan gerçek zamanlı çekirdek ile değiştirilerek gerçek zamanlı şekline getirildi. Headers güncellemeleri alınarak son sürüme güncellendi.
GPOS çekirdeği sürümü: Linux pardus 4.19.0.12-amd64 #1 SMP Debian 4.19.152-1 (2020-10-18) x86_64 GNU/Linux
RTOS çekirdeği sürümü: Linux pardus 4.19.0.12-rt-amd64 #1 SMP Debian 4.19.152-1 (2020-10-18) x86_64 GNU/Linux
YOLOv3 ise nesne tanıma algoritması olarak kullanıldı (Redmon & Farhadi, 2018).
Linux üzerinde görev yöneticisi olarak kullanılan ve anlık olarak sistem üzerinde yürütülen işleri görmemizi sağlayan "Top" paketi kullanılarak, işlemci ve belleğin, YOLOv3 kullanımı sırasındaki hareketleri bir metin belgesine kaydedildi.
Microsoft Excel paketi kullanılarak verilerin grafikleri oluşturuldu.

5.2 Kalabalık Ortamlardaki İnsan Fotoğrafları Üzerinde YOLOv3 Modeli Kullanılarak Sistem Performanslarını Ölçme

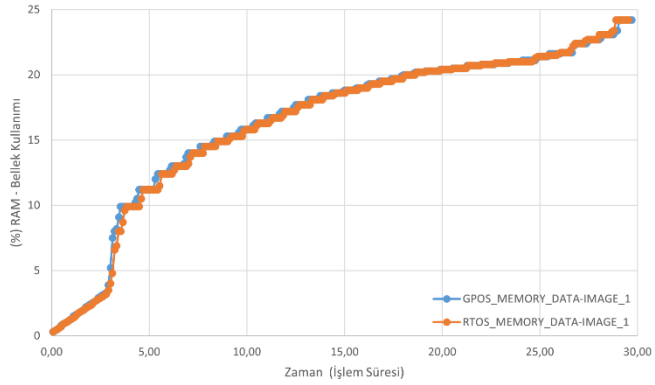
Bu çalışmadaki amaç, YOLOv3 modeli ile kalabalık ortamlarda bulunan insan fotoğraflarını analiz ederek, analiz sırasında YOLOv3 modelinin, RTOS ve GPOS üzerindeki donanım kaynaklarını ne ölçüde ve ne hızda kullandığını incelemekti. İki adet farklı fotoğraf üzerinde, YOLOv3 modelini, sırasıyla önce genel amaçlı açık kaynak kodlu Pardus işletim sistemi ve sonrasında açık kaynak kodlu gerçek zamanlı Pardus işletim sistemi üzerinde çalıştırarak, YOLOv3 modelinin CPU ve Bellek kullanımlarının yüzde olarak kayıtlarını tuttuk. Daha sonrasında ise, iki farklı işletim sisteminden elde ettiğimiz değerleri, tek bir grafikte çizdirdik.
YOLOv3 modeli seçilen fotoğrafı analiz etmek ve objeleri belirlerken CPU kullandığı için işlemci frekansı işlem bitene kadar %90 ve %100 arasında çalışıyor. Şekil-2, Şekil-3, Şekil-5 ve Şekil-6'da da görüleceği üzere kaynak kullanımları çok yakın fakat en belirgin fark kaynak kullanma sürelerindedir. Gerçek zamanlı işletim sistemi, daha karmaşık ve algılanması zor olan kişi topluluklarında genel amaçlı işletim sistemlerine yakın performans verirken, daha net ve daha algılanabilir olan kalabalık kişi topluluklarında 30 saniye süren işlemleri yaklaşık olarak 1 saniye geç bitirmektedir. Aynı tanıma algoritması kullanıldığı için tanımlanan kişiler ve tanımlama yüzdeleri iki işletim sisteminde aynı hesaplanarak işlem sonuçlandı.

5.2.1 *Birinci Fotoğraf Üzerinde Çalışma*

Şekil 1. Kalabalık ortamlarda bulunan insanların tespiti: Örnek 1 (İZÜ Mezunlarına Görkemli Tören, 2018)



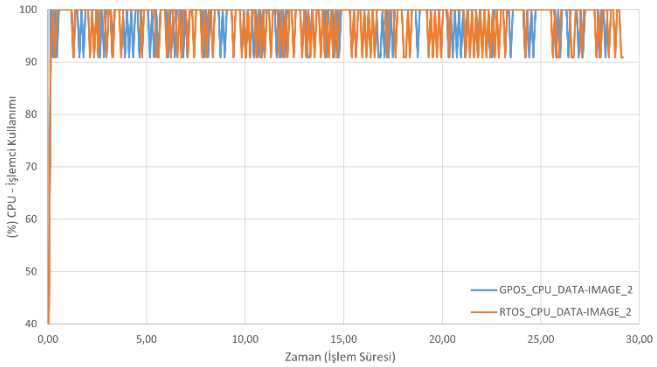
Şekil 2. Örnek 1 için işlemci kullanım grafiği



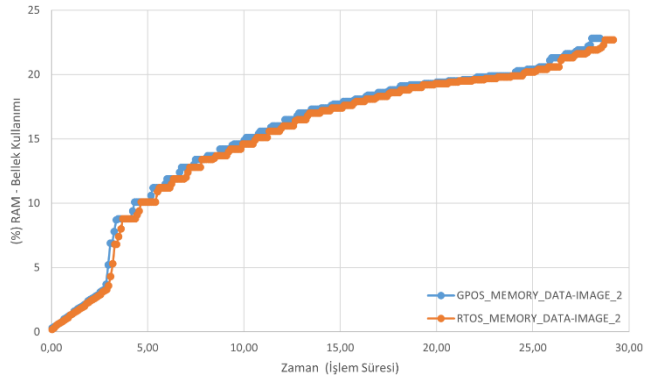
Şekil 3. Örnek 1 için bellek kullanım grafiği

5.2.2 *İkinci Fotoğraf Üzerinde Çalışma*

Şekil 4. Kalabalık ortamlarda bulunan insanların tespiti: Örnek 2 (İZÜ Mezunlarına Görkemli Tören, 2018)



Şekil 5. Örnek 2 için işlemci kullanım grafiği



Şekil 6. Örnek 2 için bellek kullanım grafiği

6. Sonuç

Açık kaynak kodlu gerçek zamanlı işletim sistemleri, yazılımcılara bir program tasarlarken, bir işin ne zaman başlaması gerektiğini ve bir işin tam olarak ne zaman bitmesi gerektiğini hesaplama özgürlüğü sunar. Bu hesaplamalar sayesinde birden çok modülü birbirine bağlayarak çok gelişmiş yazılımlar tasarlayabilirler. İşletim sisteminin çalışması için gerekli olan donanım kaynaklarını önceden bilerek gereksiz donanım maliyetinden kurtulabilirler. Açık kaynak kodlu gerçek zamanlı işletim sistemlerinin tercih edilmesindeki en büyük sebeplerden birkaçı da güvenilir olmaları, ihtiyaca göre programlanabilir olmaları ve tasarımcısının oluşabilecek hatalara karşı önlem alabilmesinden dolayı hata oranlarının genel amaçlı işletim sistemlerine karşı çok düşük olmasıdır. İnsan hayatını kurtaran hava yastıklarında, hava alanlarında hava trafiğini düzenlemek için kullanılan uygulamalarda ya da hava savunma sistemleri gibi hayati birçok alanda kullanılmaktadırlar.

Açık kaynak kodlu işletim sistemleri, çekirdekler üzerine yapılan yamalar sayesinde gerçek zamanlı işletim sistemlerine dönüştürülebilirler. Linux çekirdekleri için birçok çalışma bulunmaktadır. Fakat bu yamalar, en başından itibaren gerçek zamanlı olarak tasarlanan açık kaynak kodlu gerçek zamanlı işletim sistemlerine karşı daha az performans sağlamaktadırlar. Günümüzde açık kaynak kodlu gerçek zamanlı işletim sistemleri daha çok gömülü sistemlerde, IOT (Internet of Things) nesnelerin interneti şeklinde tanımlanan cihazlarda, mikroişlemciler gibi çok küçük belleğe ve işlem gücüne sahip cihazlarda, sadece belirli görevleri doğru bir şekilde yerine getirmek için tasarlanmaktadırlar. En başından itibaren firm real time, kesin gerçek zamanlı olarak tasarlanan bir Linux çekirdeğine sahip işletim sistemi, günlük kullanım için çok uygunsuz bir işletim sistemi olur ve tercih edilmezdi.

Her işlem sırayla ve bir önceki işlemin bitişini bekleyeceğinden dolayı, fareyi ekranın bir köşesinden başka bir köşesine götürmek bile normalden çok daha uzun sürecektir. Yaptığımız çalışmadan edindiğimiz bilgiler de bunları doğrulamaktadır. Yaklaşık 10 adet fotoğraf üzerinde gerçekleştirdiğimiz testlerden sadece 1 tanesinde gerçek zamanlı işletim sistemi, genel amaçlı işletim sisteminden daha erken sürede görevini bitirmiştir. Farklı olan tek örnek, çalışmamızda paylaşılan birinci örnektir. Çalışma ile ilgili tüm verileri Github hesabımda bulabilirsiniz (RTOSvsGPOS, 2021)

Açık kaynak kodlu gerçek zamanlı işletim sistemleri, planlandığı gibi, görevlerini yapmaya, hayatımızı kolaylaştırmaya ve hayat kurtarmaya devam ediyorlar.

Her an güncel tutulan OSRTOS isimli web sitesinden açık kaynak kodlu gerçek zamanlı işletim sistemleri örneklerine ulaşabilir, açıklamalarını okuyarak, tasarımınız için gerekli olan işletim sistemini seçebilirsiniz (OSRTOS, 2021).

Kaynaklar

Spinellis, D. (2017). A repository of Unix history and evolution. Empirical Software Engineering, 22(3), 1372-1404.

Stallman, R. (1985). The GNU manifesto. https://www.gnu.org/gnu/manifesto.html adresinden 10 Ocak 2021 tarihinde alıntılandı.

Engelfriet, A. (2009). Choosing an open source license. IEEE software, 27(1), 48-49.

Hasan, R. (2002). History of linux. https://bazaarmodel.net/ftp/Project-C/Bazaarmodel/Materiaal /Rechten-Law/History%20of%20Linux.doc

Tabassum, M., & Mathew, K. (2014, August). Software evolution analysis of linux (Ubuntu) OS. In 2014 International Conference on Computational Science and Technology (ICCST) (pp. 1-7). IEEE.

Claes, M., Mens, T., Di Cosmo, R., & Vouillon, J. (2015, May). A historical analysis of debian package incompatibilities. In 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (pp. 212-223). IEEE.

Ovadia, S. (2013). Linux for Academics, Part I. Behavioral & Social Sciences Librarian, 32(4), 252-256.

Tyler, C. (2006). Fedora Linux: A Complete Guide to Red Hat's Community Distribution. " O'Reilly Media, Inc.".

Karakoç, M. M., & Varol, A. (2016). National Distribution Project and Pardus Operating System. Turkish Journal of Science and Technology, 11(2), 25-34.

Cosmos Web Sitesi. (2021). https://www.gocosmos.org web sitesinden 10 Ocak 2021 tarihinde alıntılandı.

Villani, P. (1996). FreeDOS Kernel: An MS-DOS Emulator for Platform Independence & Embedded System Development. CRC Press.

Feske, N. (2015). Genode operating system framework.

Ghost Kernel Web Sitesi. (2021). https://ghostkernel.org/about web sitesinden 10 Ocak 2021 tarihinde alıntılandı.

Krajci, I., & Cummings, D. (2013). History and Evolution of the Android OS. In Android on x86 (pp. 1-8). Apress, Berkeley, CA.

Rani, N. U., & Krishna, Y. R. (2014). Overview of Android for User Applications. International Journal on Recent and Innovation Trends in Computing and Communication, 2(11), 3722-3725.

Wei, H., Shao, Z., Huang, Z., Chen, R., Guan, Y., Tan, J., & Shao, Z. (2016). RT-ROS: A real-time ROS architecture on multi-core processors. Future Generation Computer Systems, 56, 171-178.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009, May). ROS: an open-source Robot Operating System. In ICRA workshop on open source software (Vol. 3, No. 3.2, p. 5).

Kotecha, K., & Shah, A. (2008, June). Adaptive scheduling algorithm for real-time operating system. In 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence) (pp. 2109-2112). IEEE.

Roch, B. & Wien T. (2004). Monolithic kernel vs. Microkernel.

Baskiyar, S., & Meghanathan, N. (2005). A Survey of Contemporary Real-time Operating Systems. Informatica, 29(2).

Shah, V. H., & Shah, A. (2016). An analysis and review on memory management algorithms for real time operating system. International Journal of Computer Science and Information Security, 14(5), 236.

Shalan, M. A. (2003). Dynamic memory management for embedded real-time multiprocessor system-on-a-chip (Doctoral dissertation, Georgia Institute of Technology).

Rhodes L. (2002). Real-time Scheduling. http://faculty.juniata.edu/rhodes/os/ch10b.htm adresinden 10 Ocak 2021 tarihinde alıntılandı.

Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

İZÜ Mezunlarına Görkemli Tören. (2018) https://www.izu.edu.tr/haberler/2018/06/29/i-z%C3%BC-d%C3%B6rd%C3%BCnc%C3%BC-d%C3%B6nem-mezunlar%C4%B1n%C4%B1-co%C5%9Fkuyla-u%C4%9Furlad%C4%B1 adresinden 10 Ocak 2021 tarihinde alıntılandı.

RTOS vs GPOS. (2021). https://github.com/seckincanbaz/RTOSvsGPOS/.

OSRTOS. (2021). https://www.osrtos.com/ adresinden 10 Ocak 2021 tarihinde alıntılandı.