Uydu Uçuş Yazılımında Gerçek Zamanlı İşletim Sistemleri Zamanlama Algoritmaları

Real Time Operating Systems Scheduling Algorithms in Satellite Flight Software

Ezgi KUTLU
Uzay Sistemleri Yazılım Mühendisliği
Türk Havacılık ve Uzay Sanayii A.Ş.
Ankara, Türkiye
ezgi.kutlu@tai.com.tr
ORCID: 0000-0002-7271-2010

Camal GENÇTÜRK
Uzay Sistemleri Yazılım Mühendisliği
Türk Havacılık ve Uzay Sanayii A.Ş.
Ankara, Türkiye
camal.gencturk@tai.com.tr
ORCID: 0000-0001-7293-8557

Öz

Gerçek zamanlı sistemlerin zaman kritik ihtiyaçları gerçek zamanlı bir işletim sistemi üzerinde çalışan yazılım uygulamaları tarafından karşılanmaktadır. Zaman kritik ihtiyaçları karşılayacak olan yazılım görevlerinin belirli bir zaman çizelgesine göre çalışmalarının planlanması gerekmektedir. Bu planlamayı tetikleyen unsurlar donanım ve/veya yazılım tabanlı olaylardır. Gerçek zamanlı işletim sistemleri yazılım görevlerinin kontrolü ile ilgili altyapılar sunmakla (öncelik tabanlı çalıştırma vb.) beraber bazı durumlarda zamanlama algoritmalarının kullanılması ihtiyacı duyulmaktadır. Bu ihtiyaç özellikle zaman kritik yazılım görevlerinin farklı çalışma periyotlarında ve belirli bir zaman çizelgesinde çalışması gerektiğinde ortaya çıkmaktadır. Gerçek zamanlı sistemin ihtiyaçları doğrultusunda farklı zamanlama algoritmaları sistem ihtiyaçlarını karşılamak için seçilebilmektedir. Uzay alanında özellikle uydu platformlarında gerçek zamanlı işletim sistemi üzerinde çalışan uçuş yazılımı; uydunun operasyonu için gerekli olan komuta etme, veri alma, depolama ve otonomi kabiliyetlerini sağlamakta ve bu sayede uydudaki bütün alt sistemlerin kontrolünü gerçekleştirmektedir. Uçuş yazılımı bu işlemleri gerçekleştirirken uydunun zaman kritik ihtiyaçlarını karşılamaktan sorumludur. Bu makale gerçek zamanlı işletim sistemleri üzerinde çalışan Round Robin (Zaman Dilimli), Rate Monotonic (Oransal Monoton), Deadline Monotonic (Zaman Sınırı Monoton), Earliest Deadline First (En Yakın Zaman Sınırı Önce), Least Laxity First (En Az İhmal Edilebilir Önce) ve Enhanced Least Laxity First (Gelişmiş En Az İhmal Edilebilir Önce) zamanlama algoritmalarının karşılaştırmasını, bu algoritmaların gerçek zamanlı bir işletim sistemi olan RTEMS (Real-Time Executive for Multiprocessor Systems) üzerinde uygulanması için kullanılan yöntemi ve bu uygulama neticesinde elde edilen, algoritmalara ait performans sonuçlarını sunmayı amaçlamaktadır. Bunlara ek olarak uzay alanının ihtiyaçları göz önünde bulundurulduğunda bu algoritmalardan hangilerinin tercih edildiğinden bahsedilmektedir.

Anahtar sözcükler: *Uzay Alanı, İşletim Sistemi, İşlem, Zamanlama Algoritmaları, Round Robin Algoritması, Rate Monotonic Algoritması, Deadline Monotonic Algoritması, Earliest Deadline First Algoritması, Least Laxity First Algoritması, Enhanced Least Laxity First Algoritması*

Abstract

Time-critical requirements of real time systems are provided by software applications running on real time operating systems. These software tasks must be scheduled based on software and hardware events. There are some services (priority based preemption etc.) in real time operating systems to control software tasks. But in some situations there is a need for scheduling algorithms in real time systems. This need arises especially when time-critical software tasks need to run at different working periods and on a

Gönderme ve kabul tarihi: 24.11.2020 - 01.05.2021

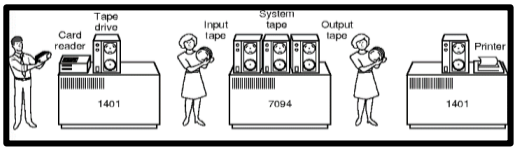
Makale türü: Araştırma

specific timeline. Different scheduling algorithms can be selected to meet the requirements of the system. In space domain, especially on space segment, system have to be real-time and time-critical. This behavior is provided by flight software which is responsible for command, data acquisition, storage and autonomy capabilities of satellite. Most of these functionalities should be implemented as time critical manner. Main goals of this paper are providing analyze of Round Robin, Rate Monotonic, Deadline Monotonic, Earliest Deadline First, Least Laxity First ve Enhanced Least Laxity First scheduling algorithms using in real time operating systems, giving the method which is used to apply scheduling algorithms on RTEMS (Real-Time Executive for Multiprocessor Systems) which is a real time operating system and providing performance results of these scheduling algorithms obtained as a result of this application. In addition, the selection approach of scheduling algorithms considering the space domain is mentioned.

Keywords: *Space Domain, Operating System, Task, Scheduling Algorithms, Round Robin Algorithm, Rate Monotonic Algorithm, Deadline Monotonic Algorithm, Earliest Deadline First Algorithm, Least Laxity First Algorithm, Enhanced Least Laxity First Algorithm*

1. Giriş

İşletim sistemlerinin çıkış noktası bilgisayarların yazılımdan daha çok donanımdan ibaret olduğu zamanlara dayanır. Bilgisayara bir işlem yaptırılmak istendiğinde programcılar tarafından kodlanan delikli kartlar önce kart okuyucu ve bant sürücü tarafından bantlara yazılır, oluşturulan giriş bandı bilgisayar operatörleri tarafından bilgisayara yüklenir, bilgisayar işlemleri yaptıktan sonra sonucu görüntülemek için çıkış bandı yine operatörler tarafından yazıcıya yüklenir ve sonuç alınırdı. Şekil-1'de IBM 1401 ve IBM 7094'ün bir arada olduğu toplu bir sistem ve bilgisayar operatörlerinin de yer aldığı süreci anlatan bir örnek gösterilmektedir. Bilgisayar operatörlerinin sürekli olarak yaptığı basit işlemlerin arka planda çalışan bir yazılım tarafından yapılması ile birlikte işletim sistemleri ortaya çıkmıştır.



Şekil-1: Bilgisayardaki Toplu Sistemlerin İlk Örneği [13]

İşletim sistemi donanım üzerinde programları çalıştıran bir yazılımdır. Sistemin ve sistemde karşılaşılabilecek problemlerin kontrolü, donanımın doğru ve verimli bir şekilde kullanılabilmesini amaçlar. Donanımın verimli bir şekilde kullanılabilmesi, kaynakların yönetilmesi ve işlem yönetimleri konularına dayanmaktadır. Aynı anda birden fazla işlem yapmak istenilebilir. Ama anlık olarak bir işlemcide bir tane işlem yapılabilir. Bu durumda yapmak istenilen işlemleri bir zamanlamaya yayarak tüm işlemler gerçekleştirilebilir. Yani Merkezi İşlem Birimi'ni (MİB) belirli zamanlarda belirli işlemler paylaşıp kullanarak tüm işlemler gerçekleştirilebilir. Kaynağı verimli kullanmak için yapılan bu zaman paylaşımı için çeşitli zamanlama algoritmaları kullanılmaktadır.

Literatürde daha çok Round Robin [6], Rate Monotonic [8], Deadline Monotonic [5], Earliest Deadline First [10], Least Laxity First ve Enhanced Least Laxity First [12] gibi zamanlama algoritmalarının ayrı ayrı ele alınarak incelendiği, bazı çalışmalarda ise birkaç tanesinin bir arada incelenerek farklı açılardan karşılaştırıldığı görülmüştür [4]. Fakat uzay alanında özellikle uydu sistemleri özelinde tercih edilen zamanlama algoritmaları ve seçim faktörleri ile ilgili kapsamlı bir çalışmaya rastlanmamıştır. Bu nedenle uydu sistemlerindeki zaman kritik ihtiyaçlar [1] göz önüne alındığında bu tarz bir çalışmaya ihtiyaç olduğu ve bu çalışmanın literatüre katkı sağlayacağı düşünülmektedir.

Bu çalışma kapsamında uydu uçuş yazılımında en çok tercih edilen gerçek zamanlı işletim sistemleri zamanlama algoritmaları incelenip karşılaştırılacaktır ve sonrasında da bu algoritmalar gerçek zamanlı bir işletim sistemi olan RTEMS [2] üzerine uygulanarak elde edilen sonuçlar değerlendirilecektir.

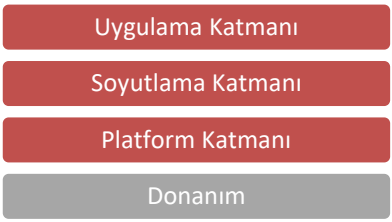
2. Uçuş Yazılımı

Uçuş yazılımı uydunun operasyonu sırasında yerine getirilmesi gereken işlemlerin gerçekleştirilmesinden sorumlu olan uydu uçuş bilgisayarı üzerinde koşan yazılımdır. Uçuş yazılımı tekrar kullanılabilirliği sağlayan, aşağıda tanımları verilen donanım

seviyesinden uygulama seviyesine kadar olan katmanlardan oluşmaktadır.

- Uygulama katmanı: Tekrar kullanılabilir yazılım bileşenleri ve uyduya özel uygulamaların yer aldığı katmandır.
- Soyutlama Katmanı: İşletim sistemine ve donanım arayüzlerine bağımlılığın uygulama seviyesinde ortadan kaldırılması için soyutlama katmanı oluşturulmuştur.
- Platform Katmanı: İşletim sisteminin ve donanım bağımlı yazılımların yer aldığı katmandır. Bu katmanda bulunan işletim sistemi, uçuş yazılımının ihtiyacı olan gerçek zamanlı çekirdeği ve uçuş bilgisayarında kullanılacak olan işlemci birimine ait kart destek paketlerini içermektedir. Bu katmanda yer alan donanım bağımlı yazılımlar ise kullanılan donanıma ait fiziksel arayüzlere erişimi sağlayan sürücüleri içermektedir.

Uçuş yazılımının katmanlı mimari yapısı Şekil-2'de verilmiştir.



Şekil-2: Uçuş Yazılımı Mimarisi

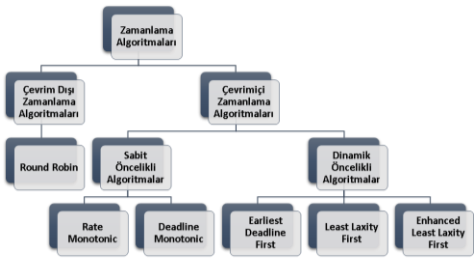
Uçuş yazılımı temelde aşağıdaki işlevleri yerine getirmektedir:

- Paketlerin alınması, kontrol edilmesi, yönlendirilmesi ve depolanması,
- Giriş-çıkış veri yollarının yönetilmesi,
- Haberleşmenin sağlanması,
- Uydu merkezi zamanının yönetilmesi,
- Komutların zamanı gelince otomatik olarak çalıştırılması, tespit edilen olay bilgilerine göre ilgili aksiyonların otomatik olarak alınması
- Alt sistemlerin yönetimi (Yörünge Yönelim Belirleme ve Kontrol, Isıl, İtki, Güç vb.)

Uçuş yazılımının yerine getirmesi gereken bu işlevlerin zamanlama ihtiyaçlarını karşılayabilmek için zamanlama algoritmaları kullanılmaktadır.

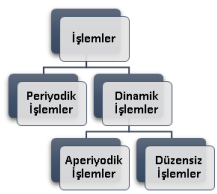
3. Gerçek Zamanlı İşletim Sistemlerinde Zamanlama Algoritmaları

Gerçek zamanlı işletim sistemleri yazılım görevlerinin zaman kritik ihtiyaçlarını karşılayacak altyapılar sunmaktadır. Gerçek zamanlı sistemlerde zamanlama kavramı, harici bir olaya en hızlı şekilde cevap verilmesi kabiliyetidir. Özellikle oluşan bu harici olaya karşılık yazılım görevlerinin belirlenen zaman aralığında çalışmasının sağlanmasıdır. Gerçek zamanlı işletim sistemlerinde kullanılan zamanlama algoritmaları Şekil-3'te verilmiştir [9].



Şekil-3: Zamanlama Algoritmalarının Sınıflandırılması

Gerçek zamanlı işletim sistemlerinde iki tip işlem vardır. Bunlardan ilki belirli zaman aralıklarıyla tekrarlanan işlemler olan periyodik işlemlerdir. Diğeri ise işlemin yerine getirilmesinin başka bir olay tarafından tetiklendiği dinamik işlemlerdir. Dinamik işlemler aperiyodik işlemler ve düzensiz işlemler olmak üzere ikiye ayrılır. Aperiyodik işlemlerin yumuşak zaman sınırları varken, düzensiz işlemlerin zaman sınırları serttir. Gerçek zamanlı işletim sistemlerindeki işlemlerin sınıflandırılması Şekil-4'te verilmiştir [9].



Şekil-4: İşlemlerin Sınıflandırılması

3.1 İşlemci Kullanım Faktörü

İşlemci kullanım faktörü, verilen n tane periyodik işlemin çalıştırılması sırasında harcanacak işlemci zamanının katsayısıdır.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \quad (1)$$

U: İşlemci kullanım faktörü

n: Görev sayısı

C_i: En kötü durumda görevin tamamlanma süresi

T_i: Zaman periyodu

Formül (1) dikkate alındığında işlemci kullanım faktörünün değerine göre sistem davranışı aşağıdaki şekildedir.

- U>1 (Çok yüklenme): Bazı işlemler belirlenen zaman sınırlarını aşacaktır (Hangi algoritma seçilirse seçilsin)
- U<=1: İşlemlerin belirlenen zaman sınırlarını aşmamaları için kullanılacak zamanlama algoritması önemlidir.
- İşlemci kullanım faktörünün 1'e eşit olması, işlemcinin kullanımının %100 olduğu anlamına gelir.

3.2 Zamanlama Faktörü

Farklı zamanlama algoritmalarının zamanlama probleminin çözümü için farklı yaklaşımları bulunmaktadır ve bu problemin çözümü için algoritma seçilirken çeşitli faktörler dikkate alınmaktadır. Zamanlama algoritmalarının karşılaştırılması sırasında kullanılan faktörler aşağıda özetlenmiştir.

CPU kullanımı: İdeal koşullarda tüm CPU döngülerini kullanabilmek için CPU kullanımı %100 olmalıdır. Normal koşullarda ise CPU kullanımı %40-%90 olmalıdır.

Çıktı: Birim zamanda tamamlanan işlem sayısıdır.

Geri Dönüş Süresi: Belirli bir işlemin tamamlanması için gerekli olan süredir. İşlemin başlangıcı ile bitişi arasındaki farktır.

Bekleme Süresi: Bir işlemin CPU üzerinde tekrar çalıştırılana kadar beklemesi gereken süredir.

Cevap Süresi: Bir komutun ilk yayınlanması ile o komuta dair cevabın üretilmesi arasındaki süredir.

3.3 Çevrimdışı Zamanlama Algoritmaları

Çevrimdışı zamanlama algoritmaları, uygulamaların hangi sırayla çalıştırılacağını seçerken daha önceden belirlenmiş ve kendini belirli zaman aralıklarıyla tekrarlayan çalışma programını kullanır. Bu programda uygulamaların hangi sırayla çalışacağı belirlidir [9].

3.3.1 Round Robin Algoritması

Round Robin (RR) zamanlama tekniği tüm işlemlere eşit zaman parçaları (kuantum) ayırarak, tüm işlemleri önceliğe dikkat etmeden dairesel sırada çalıştırır. RR kesintili bir zamanlama algoritma tekniğidir. Bu algoritmaya göre sırası gelen işlem, işlemcide işi bitmese bile belirli bir zaman parçasından (kuantum) sonra işlemciyi terk etmek zorundadır [6].

3.4 Çevrimiçi Zamanlama Algoritmaları

Çevrimiçi zamanlama algoritmalarında işlemlerin çalışma sırası işlemlere ait önceliklere göre belirlenir. İşlemlerin öncelikleri belirli kurallara göre belirlenir ve bu öncelikler çalışma sırasında değişebilir. Çevrimiçi zamanlama algoritmaları, sabit öncelikli algoritmalar ve dinamik öncelikli algoritmalar olmak üzere 2 gruba ayrılır [9].

3.4.1 Sabit Öncelikli Algoritmalar

Sabit öncelikli algoritmalarda işlemlere ait öncelikler sabittir. Çalışma sırasında işlemlere ait öncelikler değişmez. Rate Monotonic ve Deadline Monotonic algoritmaları sabit öncelikli zamanlama algoritmalardır.

3.4.1.1 Rate Monotonic Algoritması

Rate Monotonic (RM) zamanlama algoritmasında işlemlerin öncelikleri işlemlere ait zaman periyotlarına göre belirlenir. Periyodu kısa olan işlemlerin öncelikleri daha yüksektir. İşlemlere ait periyotlar değişmediği için öncelikler de sabit kalır ve çalışma sırasında değişmez [8].

3.4.1.2 Deadline Monotonic Algoritması

Deadline Monotonic (DM) zamanlama algoritmasında işlemlerin öncelikleri işlemlerin zaman sınırlarına göre belirlenir. Zaman sınırı küçük olan işlemlerin öncelikleri daha yüksektir. Çalışma sırasında işlemlerin zaman sınırları değişmediği için öncelikler de sabit kalır ve çalışma sırasında değişmez [5].

3.4.2 Dinamik Öncelikli Algoritmalar

Dinamik öncelikli algoritmalarda işlemlerin öncelikleri çalışma sırasında değişebilir. Earliest Deadline First, Least Laxity First ve Enhanced Least Laxity First algoritmaları dinamik öncelikli algoritmalardır.

3.4.2.1 Earliest Deadline First Algoritması

Earliest Deadline First (EDF) gerçek zamanlı gömülü sistemlerde kullanılan bir zamanlama algoritmasıdır. Bu algoritmada işlemlerin öncelikleri belirlenirken, işlemlere ait mutlak zaman sınırları dikkate alınır. Mutlak zaman sınırı anlık olarak değişir. Bu nedenle

çalışma sırasında işlemlerin öncelikleri de mutlak zaman sınırları ile birlikte değişir [10].

3.4.2.2 Least Laxity First Algoritması

Least Laxity First (LLF) algoritmasında işlemlerin öncelikleri işlemlerin o anki gevşeklik değerlerine göre belirlenir. Bir işlemin gevşekliği ne kadar az ise önceliği o kadar yüksektir. Yani aciliyeti yüksek olan işlemlerin öncelikleri de aynı şekilde yüksektir.

$$L_i = D_i - (t_i + {C_i}^R) \quad (2)$$

L_i: Gevşeklik

D_i: Zaman sınırı

${C_i}^R$: En kötü durumda işlemin tamamlanma süresi

t_i: Şimdiki zaman

Formül (2) kullanılarak her işlemin gevşeklik değeri dinamik olarak hesaplanmaktadır. İşlemlere ait gevşeklik değerleri çalışma sırasında değiştiğinden işlemlerin öncelikleri de değişir. Burada bir birim zamanda diğer işlemlere ait gevşeklik değerleri değişirken, o an çalışan işlemin gevşeklik değerinin değişmediğine dikkat edilmelidir. LLF ideal bir zamanlama algoritmasıdır çünkü eğer bir işlem setine ait işlemci kullanım faktörü değeri 1'den küçük veya 1'e eşit ise o görev seti LLF algoritması kullanılarak zamanlanabilir demektir. Ayrıca hangi işlemin zaman sınırını aşacağı bilgisini de vermektedir. Ancak yüksek hesaplama talebi bu zamanlama algoritmasının dezavantajlarından biridir. Ayrıca bu algoritma, işlem seti içerisinde en düşük gevşeklik değerine sahip birden fazla işlem olması durumunda düşük performans sergilemektedir [12].

LLF algoritmasının en önemli avantajlarından biri de EDF algoritmasında aşırı yüklenme sonucunda görülen domino etkisi (işlem seti içerisindeki işlemlerden birinin zaman sınırını aşması durumunda diğer işlemlerin de birbiri ardına zaman sınırlarını kaçırması durumu) sorununun önüne geçme konusunda başarılı olmasıdır.

3.4.2.3 Enhanced Least Laxity First Algoritması

Least Laxity First (LLF) algoritması EDF algoritmasında görülen aşırı yüklenme ve domino etkisi durumlarının önüne geçme konusunda başarılı olsa da işlem seti içerisinde aynı anda birden fazla en düşük gevşekliğe sahip işlem bulunması durumunda dezavantajlıdır. Aynı anda en düşük gevşeklik değerine sahip birden fazla işlem olması durumunda sürekli içerik değişimi sorunu (çalışan işlemler arasında sürekli olarak geçiş yapılmaya çalışılması) görülür. Bu da hesaplama sırasında işlem yükünü arttırır ve hesaplama zamanında kayıplara neden olur. LLF algoritmasında görülen bu dezavantajın önüne geçmek için Enhanced Least Laxity First (ELLF) algoritması geliştirilmiştir. ELLF algoritmasında işlem seti içerisinde aynı anda en düşük gevşekliğe sahip birden fazla işlem olması durumunda bu işlemler gruplanarak hangi işlemin çalıştırılacağına EDF algoritması kullanılarak karar verilir [12].

4. Zamanlama Algoritmalarının Uygulanması

Daha önceki bölümde anlatılan zamanlama algoritmaları gerçek zamanlı bir işletim sisteminde uygulanmıştır. Uygulama birkaç aşamadan oluşmaktadır.

İlk olarak periyodik ve aperiyodik görevler; isimleri, bırakma süresi, en kötü durum çalışma, periyot ve zaman sınırı özellikleri ile birlikte kullanılmak istenilen zamanlama algoritması seçilerek ilgili girdilerin tanımlandığı dosyada belirlenmektedir. Girdi dosyasında, uçuş yazılımının çalışması sırasında sorumlu olduğu paketlerin alınması, kontrol edilmesi, yönlendirilmesi ve depolanması, giriş-çıkış veri yollarının yönetilmesi, haberleşmenin sağlanması, uydu merkezi zamanının yönetilmesi ve alt sistemlerin yönetimi gibi uygulamalar 1'den 10'a kadar numaralandırılmıştır. Zamanlama algoritmalarının karşılaştırılması bu 10 uygulama üzerinden değerlendirilmiştir. Zamanlama algoritması olarak ise Round Robin, Rate Monotonic, Deadline Monotonic, Earliest Deadline First, Least Laxity First, Enhanced Least Laxity First algoritmaları seçilebilmektedir. Round Robin algoritması seçilmesi durumunda kuantum tanımlaması yapılmalıdır. Ayrıca ana çerçeve olarak belirlenen süre tanımlanan görevlerin periyotlarının en küçük ortak katıdır.

Uygulama Sparc V8 mimarisine sahip olan, 32 bit Leon3FT işlemci üzerinde, RTEMS [2] gerçek zamanlı işletim sistemi kullanılarak gerçekleştirilmiştir. Uygulama hazırlanırken geliştirme ortamı olarak Eclipse, programlama dili olarak C kullanılmıştır.

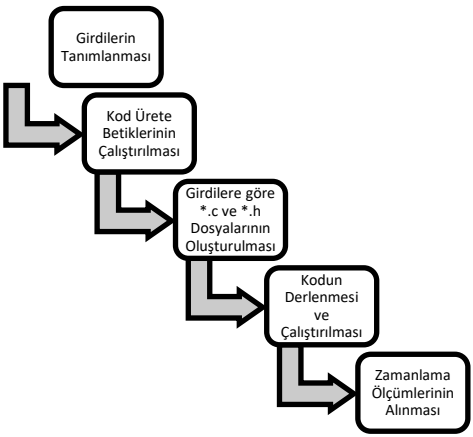
Girdi dosyasında tanımlamalar yapıldıktan sonra groovy betik dili ile yazılan kod üretme betikleri çalıştırılarak İşletim Sisteminde girdi olarak kullanılmak üzere *.c ve *.h dosyaları meydana getirilmektedir. Kod üretme betiklerinde öncelikle girdi dosyasındaki tanımlamaların doğruluğu kontrol edilmektedir. Bunlar; alanların doğru olup olmadığı, girdilerin eksik olup olmadığı ve son süre-periyot tutarlılığıdır. Bu kontroller sırasında bir hata alınırsa

kod üretilmeyecek ve ekrana hata basacaktır, hata alınmaz ise kod üretimine başarılı bir şekilde devam edilecektir. Sonrasında seçilen algoritmalar üzerinden kontroller yapılacaktır. Bunlar; zamanlama algoritmasının uygulanabilirliği son sürenin kaçırılması ve periyotlara göre ana çerçevenin uygunluğudur. Kontroller sırasında bu durumlardan biri sağlanırsa ekrana hata basacak ve kod üretilmeyecek ya da ekrana uyarı basılacak ve kod üretilmeye devam edilecektir, hiçbiri sağlanmaz ise kod üretimine başarılı bir şekilde devam edilecektir.

Seçilen algoritmaya ve tanımlamalara göre sırası ile çalışacak olan görevler ve bu görevlerin çalışma zamanları belirlenen ana çerçeve süresi boyunca oluşturulur. Kullanıcıyı bilgilendirme amacı ile ekrana basılır. İşletim sisteminde kullanılmak üzere gerekli tanımlamaların olduğu *.c ve *.h dosyaları oluşturulur.

Kod üretme betikleri çalıştırıldıktan ve başarı ile kod üretildikten sonra işletim sistemi belirlenen girdilere ve algoritmaya göre derlenip çalışmaktadır. Derleme aracı olarak Scons kullanılmıştır.

Zamanlama sonuçlarını görmek amacı ile tanımlanan görevlerin numaraları, başlangıç ve bitiş zamanları hafızaya yazılmaktadır. Bu bilgiler hafızadan indirilip bir dosyaya yazılır. Sonrasında zamanlama sonuçları görselleştirilebilir. Yukarıda anlatılan işlemler sırası ile Şekil-5'te gösterilmiştir.



Şekil-5: Uygulamanın İşlem Akışı

5. Sonuçlar

Belirlenen zamanlama algoritmalarında benzer tanımlamalar yapılarak işletim sistemi üzerindeki performans sonuçları elde edilmiştir. Sonuçlar şekiller ile açıklanmakta olup, şekillerin x-ekseni zamanı (milisaniye), y-ekseni ise uygulama numaralarını göstermektedir. Ayrıca uygulama numaralarının renkleri şekillerin en altında gösterilmiştir. Çizelgelerde belirtilen uygulamalar girdi dosyasında tanımlanmış olup, bu uygulamalar haricinde başka bir uygulama çalıştırılmamıştır.

5.1 Round Robin Algoritması Sonuçları

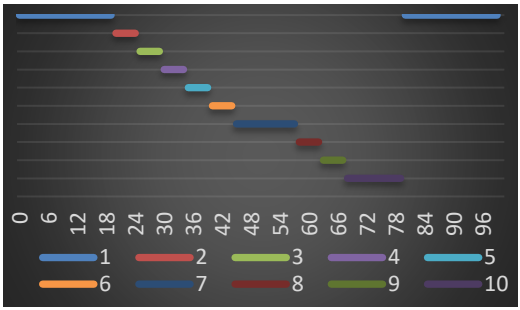
Round Robin algoritmasında girdiler Çizelge-1 ve Çizelge-2'de, ölçümler Şekil-6, Şekil-7 ve Şekil-8'de gösterilmiştir. Kuantum değeri 20 ms iken birinci durumda 1 numaralı uygulamanın en kötü durum çalışma süresi en başta ve en sonda tanımlanmış 20 ms olarak belirlenmiştir ve ikinci durumda 1 numaralı uygulamanın en kötü durum çalışma süresi 40 ms'dir, uygulama en başta tanımlanmıştır. Birinci ve ikinci durumun sonuçları kuantum değeri 20 ms olduğu için beklendiği gibi aynıdır. Üçüncü durumda Çizelge-2'de tanımlanan değerler kullanılmış, kuantum değeri 10 ms yapılmış ve içerik değiştirme miktarı artmıştır. Son durumda ise Çizelge-2'de tanımlanan değerler kullanılmış, kuantum değeri 5 ms yapılmış ve içerik değiştirme miktarı çok daha fazla artmıştır. İçerik değiştirme miktarı arttıkça işlemci zamanını içerik değiştirme için harcamakta ve verimlilik düşmektedir.

Çizelge-1: Round Robin Algoritmasında 1. Durum İçin Kullanılan Değerler

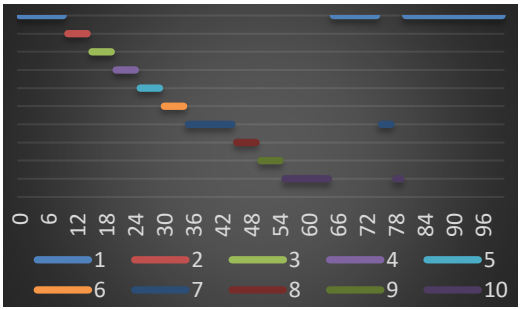
Test Durumu	**Uygulama Özellikleri**				
	Uygulama Numarası	***Bırakma Süresi (ms)***	***En Kötü Çalışma Süresi (ms)***	***Periyot (ms)***	***Zaman Sınırı (ms)***
Durum 1	1	0	20	100	100
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	100
	6	0	5	100	100
	7	0	13	100	100
	8	0	5	100	100
	9	0	5	100	100
	10	0	12	100	100
	1	0	20	100	100

Çizelge-2: Round Robin Algoritmasında 2. Durum İçin Kullanılan Değerler

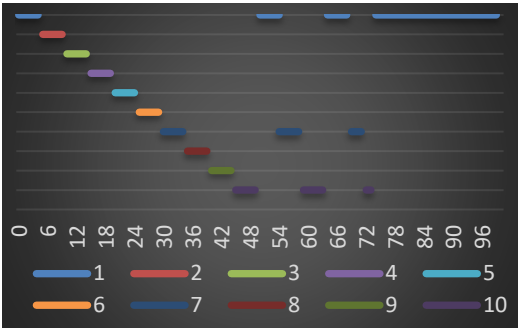
Test Durumu	Uygulama Özellikleri				
	Uygulama Numarası	*Bırakma Süresi (ms)*	*En Kötü Çalışma Süresi (ms)*	*Periyot (ms)*	*Zaman Sınırı (ms)*
Durum 2	1	0	40	100	100
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	100
	6	0	5	100	100
	7	0	13	100	100
	8	0	5	100	100
	9	0	5	100	100
	10	0	12	100	100



Şekil-6: Round Robin Algoritması 1. ve 2. Durum Sonuçları



Şekil-7: Round Robin Algoritması 3. Durum Sonuçları



Şekil-8: Round Robin Algoritması 4. Durum Sonuçları

5.2 Rate Monotonic Algoritması Sonuçları

Rate Monotonic algoritmasında girdiler Çizelge-3, Çizelge-4 ve Çizelge-5'te, ölçümler Şekil-9 ve Şekil-10'da gösterilmiştir. Birinci ve ikinci durum arasındaki fark 10 numaralı uygulamanın zaman sınırının yarıya indirilmesidir. Rate Monotonic periyoda göre önceliklendirme yaptığı için bu değişimden etkilenmemiş ve sonuçlar aynı çıkmıştır. Fakat 10 numaralı uygulama belirlenen zaman sınırı içinde işlemini tamamlayamamıştır. Üçüncü durumda ise 10 numaralı uygulamanın periyodu yarıya indirilmiştir. Bu durumda 1 numaralı ve 10 numaralı uygulamaların periyotları yani öncelikleri aynı olmaktadır. İki uygulama arasında seçim en çok bekleyene göre, sıralamaya göre veya rastgele yapılabilmektedir. Bu uygulamada sıralamaya göre yapılmaktadır. Bu durumda tüm uygulamalar belirlenen zaman sınırı içinde işlemlerini tamamlamıştır.

Çizelge-3: Rate Monotonic Algoritmasında 1. Durum İçin Kullanılan Değerler

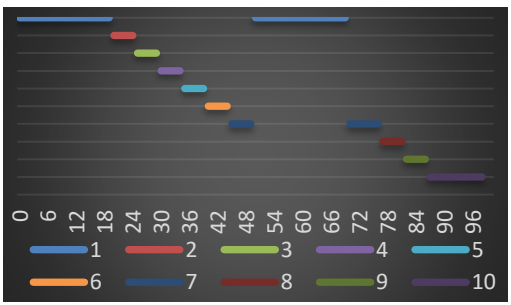
Test Durumu	Uygulama Özellikleri				
	Uygulama Numarası	*Bırakma Süresi (ms)*	*En Kötü Çalışma Süresi (ms)*	*Periyot (ms)*	*Zaman Sınırı (ms)*
Durum 1	1	0	20	50	50
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	100
	6	0	5	100	100
	7	0	13	100	100
	8	0	5	100	100
	9	0	5	100	100
	10	0	12	100	100

Çizelge-4: Rate Monotonic Algoritmasında 2. Durum İçin Kullanılan Değerler

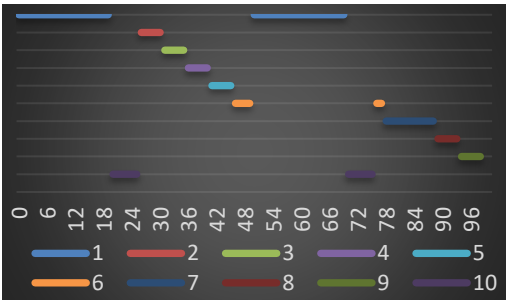
Test Durumu	Uygulama Özellikleri				
	Uygulama Numarası	*Bırakma Süresi (ms)*	*En Kötü Çalışma Süresi (ms)*	*Periyot (ms)*	*Zaman Sınırı (ms)*
Durum 2	1	0	20	50	50
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	100
	6	0	5	100	100
	7	0	13	100	100
	8	0	5	100	100
	9	0	5	100	100
	10	0	12	100	50

Çizelge-5: Rate Monotonic Algoritmasında 3. Durum İçin Kullanılan Değerler

Test Durumu	Uygulama Özellikleri				
	Uygulama Numarası	*Bırakma Süresi (ms)*	*En Kötü Çalışma Süresi (ms)*	*Periyot (ms)*	*Zaman Sınırı (ms)*
Durum 3	1	0	20	50	50
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	100
	6	0	5	100	100
	7	0	13	100	100
	8	0	5	100	100
	9	0	5	100	100
	10	0	6	50	50



Şekil-9: Rate Monotonic Algoritması 1. ve 2. Durum Sonuçları



Şekil-10: Rate Monotonic Algoritması 3. Durum Sonuçları

5.3 Deadline Monotonic Algoritması Sonuçları

Deadline Monotonic algoritmasında girdiler Çizelge-6, Çizelge-7 ve Çizelge-8'de ölçümler Şekil-11, Şekil-12 ve Şekil-13'te gösterilmiştir. Rate Monotonic algoritması için kullanılan birinci durum ve ikinci durum girdileri ile Deadline Monotonic için kullanılan girdiler aynıdır. İkinci durumda 10 numaralı uygulamanın zaman sınırı yarıya indirilmiştir. Bu uygulamada önceliklendirme zaman sınırına göre yapıldığından uygulamalar zaman sınırı içinde işlemlerini tamamlamıştır. Rate Monotonic'e göre zaman sınırının kaçırılması konusunda daha başarılıdır. Fakat üçüncü durumda aynı zaman sınırına sahip birçok uygulama girilmiştir. Deadline Monotonic sadece zaman sınırına göre önceliklendirme yaptığı için bu durumun üstesinden gelememiştir ve 10 numaralı uygulama belirlenen zaman sınırı içinde işlemini tamamlayamamıştır.

Çizelge-6: Deadline Monotonic Algoritmasında 1. Durum İçin Kullanılan Değerler

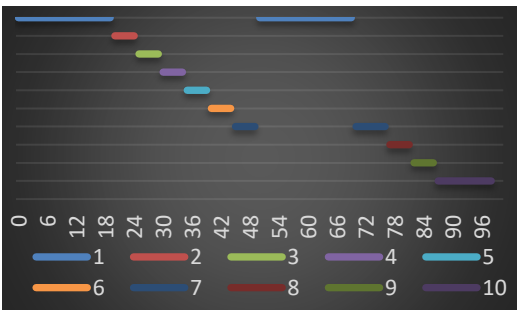
Test Durumu	Uygulama Özellikleri				
	Uygulama Numarası	*Bırakma Süresi (ms)*	*En Kötü Çalışma Süresi (ms)*	*Periyot (ms)*	*Zaman Sınırı (ms)*
Durum 1	1	0	20	50	50
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	100
	6	0	5	100	100
	7	0	13	100	100
	8	0	5	100	100
	9	0	5	100	100
	10	0	12	100	100

Çizelge-7: Deadline Monotonic Algoritmasında 2. Durum İçin Kullanılan Değerler

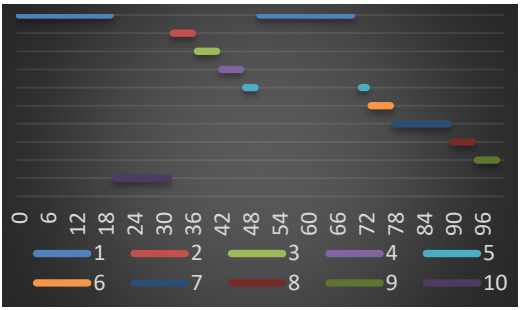
Test Durumu	Uygulama Özellikleri				
	Uygulama Numarası	*Bırakma Süresi (ms)*	*En Kötü Çalışma Süresi (ms)*	*Periyot (ms)*	*Zaman Sınırı (ms)*
Durum 2	1	0	20	50	50
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	100
	6	0	5	100	100
	7	0	13	100	100
	8	0	5	100	100
	9	0	5	100	100
	10	0	12	100	50

Çizelge-8: Deadline Monotonic Algoritmasında 3. Durum İçin Kullanılan Değerler

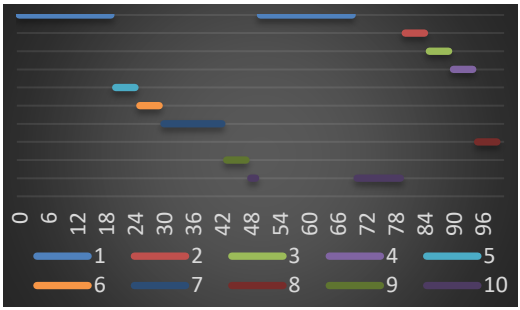
Test Durumu	Uygulama Özellikleri				
	Uygulama Numarası	*Bırakma Süresi (ms)*	*En Kötü Çalışma Süresi (ms)*	*Periyot (ms)*	*Zaman Sınırı (ms)*
Durum 3	1	0	20	50	50
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	60
	6	0	5	100	60
	7	0	13	100	60
	8	0	5	100	100
	9	0	5	100	60
	10	0	12	100	60



Şekil-11: Deadline Monotonic Algoritması 1. Durum Sonuçları



Şekil-12: Deadline Monotonic Algoritması 2. Durum Sonuçları



Şekil-13: Deadline Monotonic Algoritması 3. Durum Sonuçları

5.4 Earliest Deadline First Algoritması Sonuçları

Earliest Deadline First algoritmasında girdiler Çizelge-9, Çizelge-10 ve Çizelge-11'de ölçümler Şekil-14, Şekil-15 ve Şekil-16'da gösterilmiştir. Üç durum için de girdiler Deadline Monotonic ile aynıdır. Earliest Deadline First dinamik olarak mutlak zaman sınırını hesaplamakta ve bu değere göre önceliklendirme yapmaktadır. Bu nedenle üçüncü durumda Deadline Monotonic'te görülen zaman sınırının kaçırılması burada görülmemiştir. Zaman sınırının kaçırılması konusunda Rate Monotonic'e göre ve Deadline Monotonic'e göre daha başarılıdır. Fakat geçici aşırı yükleme durumlarında uygulamalar zaman sınırını aşmakta ve bundan sonra gelen tüm uygulamalar sıralı olarak zaman sınırını aşmaya devam etmektedir.

Çizelge-9: Earliest Deadline First Algoritmasında 1. Durum İçin Kullanılan Değerler

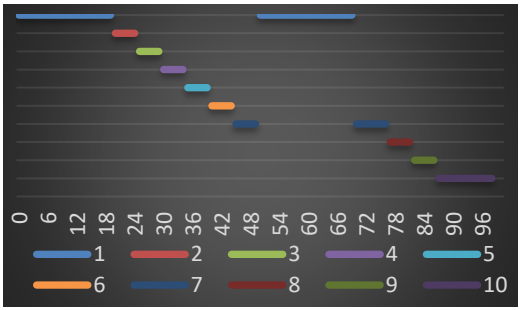
Test Durumu	Uygulama Özellikleri				
	Uygulama Numarası	*Bırakma Süresi (ms)*	*En Kötü Çalışma Süresi (ms)*	*Periyot (ms)*	*Zaman Sınırı (ms)*
Durum 1	1	0	20	50	50
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	100
	6	0	5	100	100
	7	0	13	100	100
	8	0	5	100	100
	9	0	5	100	100
	10	0	12	100	100

Çizelge-10: Earliest Deadline First Algoritmasında 2. Durum İçin Kullanılan Değerler

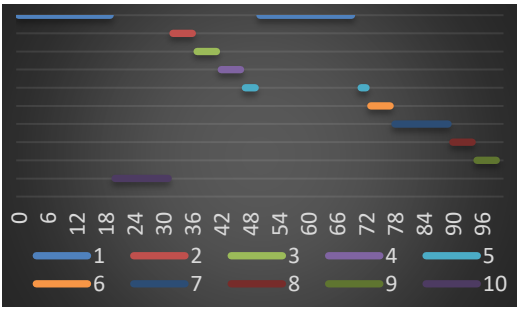
Test Durumu	Uygulama Özellikleri				
	Uygulama Numarası	*Bırakma Süresi (ms)*	*En Kötü Çalışma Süresi(ms)*	*Periyot (ms)*	*Zaman Sınırı (ms)*
Durum 2	1	0	20	50	50
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	100
	6	0	5	100	100
	7	0	13	100	100
	8	0	5	100	100
	9	0	5	100	100
	10	0	12	100	50

Çizelge-11: Earliest Deadline First Algoritmasında 3. Durum İçin Kullanılan Değerler

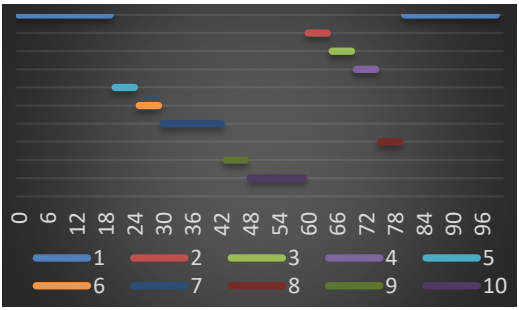
Test Durumu	Uygulama Özellikleri				
	Uygulama Numarası	*Bırakma Süresi (ms)*	*En Kötü Çalışma Süresi (ms)*	*Periyot (ms)*	*Zaman Sınırı (ms)*
Durum 3	1	0	20	50	50
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	60
	6	0	5	100	60
	7	0	13	100	60
	8	0	5	100	100
	9	0	5	100	60
	10	0	12	100	60



Şekil-14: Earliest Deadline First Algoritması 1. Durum Sonuçları



Şekil-15: Earliest Deadline First Algoritması 2. Durum Sonuçları



Şekil-16: Earliest Deadline First Algoritması 3. Durum Sonuçları

5.5 Least Laxity First Algoritması Sonuçları

Least Laxity First algoritmasında girdiler Çizelge-12, Çizelge-13 ve Çizelge-14'te gösterilmiştir. Least Laxity First algoritması dinamik olarak gevşeklik zamanını hesaplamakta ve bu değere göre

önceliklendirme yapmaktadır. Dinamik olarak çalıştığı için zaman sınırının kaçırılması konusunda Rate Monotonic ve Deadline Monotonic algoritmalarına göre daha başarılıdır. Fakat birden fazla uygulamanın gevşeklik süreleri birbirine eşit olduğu durumda sürekli olarak bu uygulamalar arasında kısa süreli olarak geçiş yapılmaktadır. Bu durumda sürekli içerik değiştirmeye çalışılmakta ve verim oldukça düşmektedir. Üç durumda da gevşeklik değerleri birbirine eşit birden fazla uygulama vardır. 100 ms'lik periyot içerisinde yaklaşık olarak 55 defa içerik değiştirmeye çalışmıştır, uygulamalar kendilerine ayrılan sürelerde işlemlerini tamamlayamamıştır. Bu nedenle zaman ölçümü alınamamıştır. Birden fazla uygulamanın aynı gevşeklik değerine eşit olduğu durumlarda bu algoritmanın kullanımı uygun olmamaktadır.

Çizelge-12: Least Laxity First Algoritmasında 1. Durum İçin Kullanılan Değerler

Test Durumu	Uygulama Özellikleri				
	Uygulama Numarası	*Bırakma Süresi (ms)*	*En Kötü Çalışma Süresi (ms)*	*Periyot (ms)*	*Zaman Sınırı (ms)*
Durum 1	1	0	20	50	50
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	100
	6	0	5	100	100
	7	0	13	100	100
	8	0	5	100	100
	9	0	5	100	100
	10	0	12	100	100

Çizelge-13: Least Laxity First Algoritmasında 2. Durum İçin Kullanılan Değerler

Test Durumu	Uygulama Özellikleri				
	Uygulama Numarası	*Bırakma Süresi (ms)*	*En Kötü Çalışma Süresi (ms)*	*Periyot (ms)*	*Zaman Sınırı (ms)*
Durum 2	1	0	20	50	50
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	100
	6	0	5	100	100
	7	0	13	100	100
	8	0	5	100	100
	9	0	5	100	100
	10	0	12	100	50

Çizelge-14: Least Laxity First Algoritmasında 3. Durum İçin Kullanılan Değerler

Test Durumu	Uygulama Özellikleri				
	Uygulama Numarası	*Bırakma Süresi (ms)*	*En Kötü Çalışma Süresi (ms)*	*Periyot (ms)*	*Zaman Sınırı (ms)*
Durum 3	1	0	20	50	50
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	60
	6	0	5	100	60
	7	0	13	100	60
	8	0	5	100	100
	9	0	5	100	60
	10	0	12	100	60

5.6 Enhanced Least Laxity First Algoritması Sonuçları

Enhanced Least Laxity First algoritmasında girdiler Çizelge-15, Çizelge-16 ve Çizelge-17'de ölçümler Şekil-17, Şekil-18 ve Şekil-19'da gösterilmiştir. Enhanced Least Laxity First algoritması Least Laxity First ve Earliest Deadline First algoritmalarının birleşimidir. Üç durum için de girdiler Deadline Monotonic ile aynıdır. Öncelikle dinamik olarak gevşeklik değeri hesaplanır ve bu değere göre önceliklendirme yapılır. Bu şekilde çalışması ile Least Laxity First algoritması gibidir. Fakat gevşeklik değerlerinin birbirine eşit olduğu durumda Earliest Deadline First algoritması gibi çalışmaya başlayıp mutlak zaman sınırı üzerinden önceliklendirme yapmaya devam etmektedir. Bu durumda hem zaman sınırının kaçırılması problemi yaşanmamakta hem de Least Laxity First algoritmasındaki içerik değiştirme probleminin üstesinden gelinmektedir. Üç durumda da başarılı bir şekilde çalışmıştır.

Çizelge-15: Enhanced Least Laxity First Algoritmasında 1. Durum İçin Kullanılan Değerler

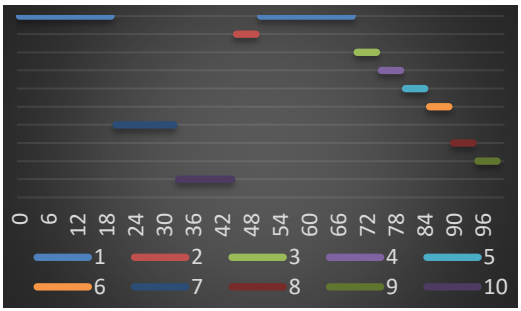
Test Durumu	Uygulama Özellikleri				
	Uygulama Numarası	*Bırakma Süresi (ms)*	*En Kötü Çalışma Süresi (ms)*	*Periyot (ms)*	*Zaman Sınırı (ms)*
Durum 1	1	0	20	50	50
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	100
	6	0	5	100	100
	7	0	13	100	100
	8	0	5	100	100
	9	0	5	100	100
	10	0	12	100	100

Çizelge-16: Enhanced Least Laxity First Algoritmasında 2. Durum İçin Kullanılan Değerler

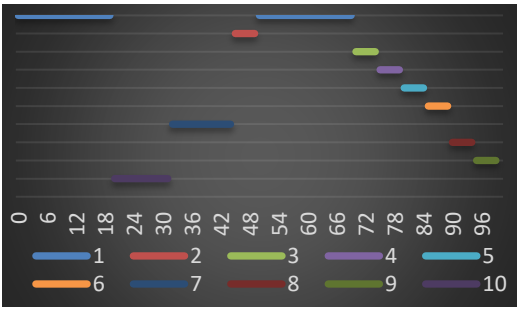
Test Durumu	Uygulama Özellikleri				
	Uygulama Numarası	*Bırakma Süresi (ms)*	*En Kötü Çalışma Süresi (ms)*	*Periyot (ms)*	*Zaman Sınırı (ms)*
Durum 2	1	0	20	50	50
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	100
	6	0	5	100	100
	7	0	13	100	100
	8	0	5	100	100
	9	0	5	100	100
	10	0	12	100	50

Çizelge-17: Enhanced Least Laxity First Algoritmasında 3. Durum İçin Kullanılan Değerler

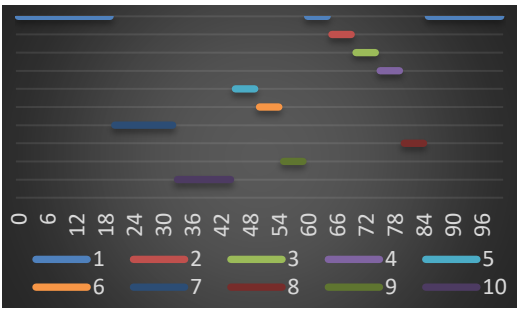
Test Durumu	Uygulama Özellikleri				
	Uygulama Numarası	*Bırakma Süresi (ms)*	*En Kötü Çalışma Süresi (ms)*	*Periyot (ms)*	*Zaman Sınırı (ms)*
Durum 3	1	0	20	50	50
	2	0	5	100	100
	3	0	5	100	100
	4	0	5	100	100
	5	0	5	100	60
	6	0	5	100	60
	7	0	13	100	60
	8	0	5	100	100
	9	0	5	100	60
	10	0	12	100	60



Şekil-17: Enhanced Least Laxity First Algoritması 1. Durum Sonuçları



Şekil-18: Enhanced Least Laxity First Algoritması 2. Durum Sonuçları



Şekil-19: Enhanced Least Laxity First Algoritması 3. Durum Sonuçları

6. Algoritma Sonuçlarının Karşılaştırılması

İşletim sistemi üzerinde yapılan çalışmalar neticesinde zamanlama algoritmalarının performans sonuçları incelenerek, zamanlama algoritmalarının davranış karakteristikleri gözlemlenmiştir.

İncelenen algoritmalardan ilki Round Robin (RR) algoritmasıdır. RR algoritmasında kuantum değeri ile her bir işlemin çalıştırılacağı zaman parçaları belirlenebilmektedir. Ancak kuantumun çok küçük seçilmesi, işlemler arasında çok fazla geçiş gerektireceğinden sistemin verimsiz çalışmasına neden olabilmektedir. Kuantum değeri belirlenirken bu hususa dikkat edilmelidir.

İncelenen ikinci algoritma işlemlerin önceliklerini periyot değerlerine göre atayan Rate Monotonic (RM) algoritmasıdır. Bu algoritma kullanılırken işlem seti içerisinde aynı periyot değerine sahip birden fazla işlem olup olmasına dikkat edilmelidir. Aynı periyota sahip birden fazla işlem olması halinde işlemler arasında seçim en çok bekleyene göre, sıralamaya göre

veya rastgele yapılabilmektedir. Bunlardan hangisinin seçileceği işlemlerin çalışma sırasını etkilemektedir.

Üçüncü algoritma işlemlerin önceliklerini işlemlerin zaman sınırlarına göre atayan Deadline Monotonic (DM) zamanlama algoritmasıdır. DM algoritması kullanılırken işlem seti içerisinde aynı zaman sınırına sahip birden fazla işlem olması sistem çalışmasını olumsuz etkilemektedir. İşlem seti belirlenirken bu hususa dikkat edilmelidir.

Dördüncü algoritma olan Earliest Deadline First (EDF) algoritmasında işlem öncelikleri belirlenirken dinamik olarak işlemlerin mutlak zaman sınırları hesaplandığından DM ve RM algoritmalarına göre işlemlerin zaman sınırlarını aşmaması konusunda daha başarılıdır. Ancak geçici aşırı yüklenme durumlarında görülen domino etkisi nedeniyle geçici aşırı yükleme durumlarında dezavantajlıdır.

Beşinci zamanlama algoritması işlemlerin önceliklerini belirlerken dinamik olarak işlemlerin gevşeklik değerlerini hesaplayan Least Laxity First (LLF) algoritmasıdır. Bu algoritma işlemlerin zaman sınırlarını aşmaması konusunda RM ve DM algoritmalarından, domino etkisinin önüne geçmesi açısından da geçici aşırı yükleme durumunda EDF algoritmasından daha başarılıdır. Fakat işlem seti içerisinde aynı gevşeklik değerine sahip birden fazla işlem bulunması durumunda işlemler arasında sürekli olarak geçiş yapılmaya çalışılması sistem verimini oldukça düşürmektedir. İşlem seti belirlenirken aynı gevşeklik değerine sahip algoritmaların bulunmamasına dikkat edilmelidir.

Son zamanlama algoritması EDF ve LLF algoritmalarının birleşimi olan Enhanced Least Laxity First (ELLF) algoritmasıdır. Aynı gevşeklik değerine sahip birden fazla işlem olması durumunda çalıştırılacak işlemi seçerken EDF algoritmasını kullanarak LLF algoritmasında görülen sürekli içerik değiştirme sorununun önüne geçmektedir. Bu nedenle işlem setinde aynı gevşeklik değerine sahip birden fazla işlem olması senaryosunda LLF algoritmasına göre daha verimli çalışmaktadır

İncelenen zamanlama algoritmalarına ait uygulama zorluğu, öncelik atama şekli, zamanlama faktörü, MİB kullanımı ve verim karşılaştırması Çizelge-18'de verilmiştir [4].

Çizelge-18: Zamanlama Algoritmalarının Karşılaştırılması

Algoritma	*Uygulama*	*Öncelik Atama*	*MİB Kullanımı*	*Verim*
RR	Basit	Çevrimdışı	Kuantuma bağlı	Verimli
RM	Basit	Çevrimiçi/ Sabit	Az	Verimli
DM	Basit	Çevrimiçi/ Sabit	RM'den daha fazla	Verimli
EDF	Zor	Çevrimiçi/ Dinamik	Tam kullanım	Az yüklenme durumunda verimli
LLF	Zor	Çevrimiçi/ Dinamik	Tam kullanım	Aşırı yüklenme durumunda verimli
ELLF	Zor	Çevrimiçi/ Dinamik	Tam kullanım	Az yüklenme ve aşırı yüklenme durumunda verimli

7. Değerlendirme

Yapılan çalışma neticesinde girdi dosyasına girilen veriler aracılığıyla zamanlama algoritmaları arasında geçiş yapılması sağlanmaktadır. Ayrıca bu uygulama aracılığıyla işletim sisteminin seçilen algoritmalara göre davranışı çalışma sırasından önce incelenebilmektedir. Bu sayede yazılım ihtiyaçlarına göre zamanlama algoritması seçimi kolayca yapılabilmektedir. Bu uygulama kullanılarak gerçek zamanlı işletim sistemlerinde kullanılan RR, RM, DM, EDF, LLF ve ELLF zamanlama algoritmaları incelenerek zamanlama sonuçları elde edilmiştir. Bu sonuçlar çerçevesinde zamanlama algoritmalarının birbirlerine göre avantajlı ve dezavantajlı olduğu durumlar görülmüştür. Bu durumda algoritmaların iyi veya kötü algoritma olarak sınıflandırılması yazılımın ihtiyacına göre yapılmalıdır. Yazılımın ihtiyaçları; maksimum MİB kullanımı, aşırı yükleme durumlarında verimli çalışma, algoritmanın uygulanma kolaylığı vb. olabilir.

Uçuş yazılımı özelinde değerlendirirsek; uzay alanında kullanılan zamanlama algoritmalarında verimlilik ve MİB kullanımından ziyade işlem seti içerisindeki görevlerin belirlenen sırada saptanabilir bir yapıda çalışması algoritmaların seçim faktörü olarak daha önceliklidir. Ayrıca bu durum [1]'de detaylı olarak ele alınmıştır. Bu nedenle sabit öncelikli zamanlama

algoritmaları olan Rate Monotonic ve Deadline Monotonic algoritmaları tercih edilmektedir. Uzay alanında periyot tabanlı önceliklendirme yaklaşımı sunan Rate Monotonic algoritması daha çok tercih edilmektedir. Yazılım geliştiricisinin yazılım görevlerinin zamanlama davranışını sadece periyot bilgileri ile hızlı bir şekilde kurgulamasını sağlamaktadır. Ek olarak uzay alanındaki projelerde sıklıkla kullanılan açık kaynak kodlu işletim sistemi RTEMS Rate Monotonic algoritması ile ilgili servisler sunmaktadır. Bu avantajları göz önüne alındığında Rate Monotonic algoritması seçilmiştir.

Kaynakça

[1] ECSS-E-HB-40A - Software Engineering Handbook, 11 December 2013.

[2] RTEMS Classic API Guide, Release 5.4007591, 5th Edition, September 2019.

[3] A. Silberschatz, P. B. Galvin, G. Gagne, Operating System Concepts, 9th Edition, Wiley, 2012.

[4] Vijayshree Shinde, Seema C. Biday, Comparison of Real Time Task Scheduling Algorithms, International Journal of Computer Applications (0975 – 8887) Volume 158 – No 6, January 2017.

[5] Neil C. Audsley, Deadline Monotonic Scheduling, September 1990.

[6] Lipika Datta, Efficient Round Robin Scheduling Algorithm with Dynamic Time Slice, International Journal of Education and Management Engineering, June 2015.

[7] Andysah Putera, Utama Siahaan, Comparison Analysis of CPU Scheduling : FCFS, SJF and Round Robin, International Journal of Engineering Development and Research, 2016.

[8] Yi Wang, Uppsala University, Scheduling Periodic Tasks http://user.it.uu.se/~yi/courses/rts/dvp-rts-08/notes/Scheduling-periodic.pdf, Last accessed: May 2020.

[9] Frank Singhoff, University of Brest, France, Real-Time Scheduling Analysis, http://beru.univ-brest.fr/~singhoff/ENS/USTH/sched.pdf, Last accessed: May 2020.

[10] Thomas Plagemann, University of Oslo, CPU Scheduling, https://www.uio.no/studier/emner/matnat/ifi/nedlagte-emner/INF3150/h05/undervisningsmateriale/cpu-scheduling1.pdf, Last accessed: May 2020.

[11] Chenyang Lu, Washington University, Real-Time Scheduling, https://www.cse.wustl.edu/~lu/cse467s/slides/scheduling.pdf, Last accessed: May 2020.

[12] Jens Hildebrandt, Frank Golatowski, Dirk Timmermann, Scheduling Coprocessor for Enhanced Least-Laxity-First Scheduling in Hard Real Time Systems, Proceedings – Euromicro Conference on Real Time Systems, Jaunary 1999

[13] Andrew S. Tanenbaum, Herbert Bos, Modern Operatıng Systems, 4th Edition, Pearson, 2014.