# Gerçek-Zamanlı Sistemlerin Doğrulanması

**Cengiz ERBAŞ**

ASELSAN, MGEO Grubu, 06011, Ankara

e-posta: cerbas@mgeo.aselsan.com.tr

## Özet

Bu bildiride, gerçek-zamanlı sistemlerin doğrulanması için kullanılabilecek matematiksel bir yaklaşım geliştirilmektedir. Sıralı ve eşzamanlı programların doğruluğu, yalnızca program kaynak kodundan türetilebilen işlevsel özelliklerine bağlıdır. Gerçek-zamanlı programların doğruluğu ise zamanlama özelliklerine de bağlı bulunmaktadır. Bu sistemlerin doğrulanması, dolayısıyla, program kaynak kodunun yanında işletim ortamının özelliklerini bilmeyi de gerektirmektedir. Bu bildiri, aksiyomatik doğrulama tekniğini işletim ortamının ilgili özelliklerini de içine alabilecek biçimde genişletmektedir.

## Abstract

This paper develops a mathematical foundation to be used for verification of real-time systems. Correctness of sequential and concurrent programs depends on their functional properties, which can be derived solely from the program source code. Correctness of real-time programs, however, depends also on their timing properties. Verification of such systems, therefore, requires the knowledge of the run-time environment as well. This paper extends the axiomatic verification technique to include the relevant properties of the run-time environment.

## 1. Giriş

Bilgisayar programları *sıralı* (sequential), *eşzamanlı* (concurrent) ve *gerçek-zamanlı* (real-time) olarak üç ayrı kategoriye ayrılabilirler. Modelleme ve doğrulama açısından sıralı programları en yalın, gerçek-zamanlı progamları ise en karmaşık program türü olarak görebiliriz. Aksiyomatik doğrulama tekniği ilk olarak sıralı programlar için geliştirilmiş [2], ve daha sonra eşzamanlı programları kapsayacak biçimde genişletilmiştir [4]. Sıralı ve eşzamanlı programların doğruluğu, yalnızca program kaynak kodundan türetilebilen işlevsel özelliklerine bağlıdır. Gerçek-zamanlı programların doğruluğu ise işlevsel özelliklerinin yanında zamanlama özelliklerine de bağlı bulunmaktadır. Bu sistemlerin doğrulanması, program kaynak kodunun yanında işletim ortamının özelliklerini bilmeyi de gerektirmektedir.

Aksiyomatik doğrulama tekniğini gerçek-zamanlı sistemler için genişletmeyi amaçlayan çalışmalar yapılmıştır [1,3,5]. Fakat bu çalışmalar teorik düzeyde kalmış ve pratik uygulama alanları bulamamıştır. Bu bildiri, aksiyomatik doğrulama tekniğini işletim ortamının ilgili özelliklerini de içine alabilecek biçimde gerçek-zamanlı programlar için genişletmektedir. Pratik uygulamalarda

kullanılabilmesi için, tekniğin kullandığı parametreler, mikroişleyici ve işletim sistemlerinin özellikleri ile ilişkilendirilmiştir.

## 2. Sıralı Programlar

İşletimleri sırasında her an yalnızca bir komutu işletilebilen programlara sıralı programlar diyoruz. Bir başka deyişle, sıralı programlar tek bir işletim örgüsünden (*thread*) oluşmaktadır. Bu tür programların davranışları, yalnızca *girdileri* ve *çıktıları* aracılığıyla betimlenebilir. Sıralı programların işlevsel davranışlarını

$$\{p\}\ S\ \{q\} \tag{1}$$

üçlüsü ile gösteriyoruz. Burada, $p$ ve $q$, program değişkenlerinin $S$'ye girişi ve çıkışı sırasındaki değerleri üzerine önermelerdir. Burada $p$ önermesi $S$'nin *önkoşulu* olarak adlandırılır, ve $S$'nin işletiminden önce program değişkenlerinin durumunu tanımlar. $q$ önermesi de $S$'nin *ardkoşulu* olarak adlandırılır ve $S$'nin işletimi sona erdikten sonra program değişkenlerinin durumunu gösterir. Eğer $S$'nin işletimi $p$ önermesi doğru iken başlarsa, işletim sona erdiğinde $q$ önermesi de doğru olur. Bir sıralı program birden çok program kesiminden oluşabilir. $S_1, S_2, ..., S_n$ adlı program kesimlerinin sıralı birleşimini,

$$S = S_1; S_2; ...; S_n \tag{2}$$

notasyonu ile gösteriyoruz. Burada $S_1, S_2, ..., S_n$'in herbirinin davranışı kendi önkoşul ve ardkoşullarıyla şöyle ifade edilebilir:

$$\begin{array}{c} \{p_1\}\ S_1\ \{q_1\} \\ \{p_2\}\ S_2\ \{q_2\} \\ ... \\ \{p_n\}\ S_n\ \{q_n\} \end{array} \tag{3}$$

$S_2$'nin işletimi $S_1$'in işletiminin hemen ardından olacağından $S_2$'nin önkoşulu $S_1$'in ardkoşulundan çıkarsanabilir, yani $p_2 \rightarrow q_1$, olacaktır. Benzer biçimde $p_3 \rightarrow q_2$, ve bunun gibi. Sıralı birleşimin bu özelliği aşağıdaki notasyon ile gösteriyoruz[1]:

$$\frac{\{p\}\ S_1\ \{r\}, \{r\}\ S_2\ \{q\}}{\{p\}\ S_1; S_2\ \{q\}} \tag{4}$$

Bu kural programları daha küçük program kesimlerine ayrıştırmamıza izin vermektedir. Burada $S_1$ ve $S_2$ amaca bağlı olarak modül, yordam, bir programlama dilinin yapıları ya da rasgele komut

---

[1] Genel olarak $\frac{H_1, H_2, ..., H_n}{C}$ notasyonu $C$ ile gösterilen gözlemlenebilir program davranışının $H_1$, $H_2$, …, $H_n$'nin mantıksal birleşimi olarak ifade etmek amacıyla kullanılır.

dizileri olabilir. En ilkel duzeyde bir program atomik komutlarının sıralı birleşimi olarak ifade edilebilir.

### 3. Eşzamanlı Programlar

Birden çok işletim örgüsünden oluşan programlara eşzamanlı programlar diyoruz. Bir eşzamanlı programın işletim örgüleri birbiri ile program değişkenlerini paylaşarak ya da ileti alışverişi ile işbirliğine girer. $S_1, S_2, ..., S_n$ adlı program kesimlerinden oluşan eşzamanlı bir program,

$$S = S_1 \parallel S_2 \parallel ... \parallel S_n \tag{5}$$

notasyonu ile gösterilir. Sıralı program kesimlerine benzer olarak, $S_1$, $S_2$,… $S_n$ eşzamanlı örgülerinin herbirinin davranışını, örgünün işletiminden önce ve sonraki program değiskenlerinin değerlerine bağlı olarak şöyle ifade edilebiliriz:

$$\begin{gathered} \{p_1\}\, S_1\, \{q_1\} \\ ... \\ \{p_n\}\, S_n\, \{q_n\} \end{gathered} \tag{6}$$

Bununla birlikte, herbir örgünün davranışını öteki örgülerden yalıtılmış olarak incelenemez. Bir komutun işletiminden sonra program değiskenlerinin bir sonraki komutun uygulanmasına kadar aynı kalacağını garanti edemeyiz, çünkü örgünün işletimi sırasında başka bir örgünün işletimi araya girebilir. Dolayısı ile örgülerarası girişim (*interference*) aşağıdaki gibi dikkate alınmak zorundadır:

$$\frac{\{p_1\}\, S_1\, \{q_1\}, \{p_2\}\, S_2\, \{q_2\},\ \textit{örgülerarası girisim}}{\{p_1 \wedge p_2\}\, S_1 \parallel S_2\, \{q_1 \wedge q_2\}} \tag{7}$$

Eşzamanlı programlarda örgülerarası girişim temelde iki ayrı biçimde olmaktadır. Bunlardan ilki ileti alışverişi, ikincisi de ortak değişken paylaşımıdır. İki örgü arasındaki zamanuyumlu ileti alışverişini $\{a\}P!e\{b\}$ ve $\{c\}Q?v\{d\}$ üçlüleri ile gösteriyoruz. Burada $P$ iletiyi gönderen, $Q$ ise iletiyi alan örgüye; $e$ gönderilen, $v$ ise iletiyi alan değişkene karşı gelmektedir. Bu tür bir iletişim $e$'nin değerinin $v$'ye aktarılması ile aşağıdaki ilişkiyi[2] sağlayacak biçimde sona erer:

$$(a \wedge c) \rightarrow (b \wedge d)_e^v$$

Eş zamanlı örgüler arasında girişim olmasının nedenlerinden ikincisi değişken paylaşımıdır. İki örgünün $S_1 = s_{11}; s_{12}; ...; s_{1e}$ ve $S_2 = s_{21}; s_{22}; ...; s_{2f}$ olduğunu varsayalım. Burada, program doğrulaması aşamasında, $s_{2l}$'nin işletiminin $S_1$'e ilişkin önermeleri yanlışlamadığını göstermemiz gerekmektedir. Başka bir deyişle, $1 \le j \le e$ ve $1 \le l \le f$ için,

$$\{p_{1j} \wedge p_{2l}\}\, s_{2l}\, \{p_{2j}\}$$

---

[2] Burada $p_e^v$ , $v$ değişkeninin $p$ önermesinde her geçtiği yerde $e$ ile değiştirilmesinin sonucunu göstermektedir.

$$\{q_1 \wedge p_{2l}\}\, s_{2l}\, \{q_1\}$$

olmalıdır. Eğer bu koşullar sağlanmazsa, $s_{2l}$ komutu $S_1$ örgüsü ile girişime uğrar.

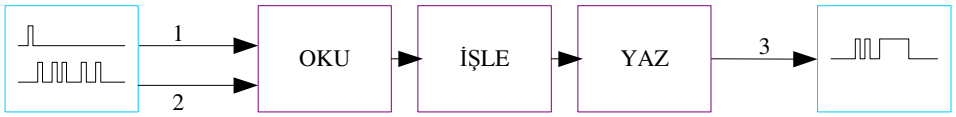
## 4. Gerçek-Zamanlı Programlar

Fiziksel ortam ile tahmin edilebilir bir zaman aralığı içinde etkileşen programlara gerçek-zamanlı programlar diyoruz. Gerçek-zamanlı programların örgülerini *denetleyici* süreçler olarak adlandırıyoruz. Denetleyici süreçler ile fiziksel ortam arasındaki etkileşim, eşzamanlı süreçler arasındaki etkileşime benzer. Fakat, bu kez, eşzamanlı süreçlerden en az birisi fiziksel ortamı temsil eden *çevresel* süreçlere karşı gelir. Çevresel süreçler ile olan etkileşimler gerçek-zamanlı programların zamanlama davranışı üzerine bazı kısıtlamalar getirir.

Bir gerçek-zamanlı program fiziksel ortamdaki süreçler ile etkileşirken:

1. Fiziksel ortamda ortaya çıkan olayları izler,
2. Bir olay algılandığında ortamdan veri toplar, ve
3. Topladığı verileri işler ve ortamdaki gerekli parametereleri değiştirir.

Gerçek-zamanlı program ile çevre arasındaki bu üç tür etkileşim aşağıda Şekil 1'de gösterilmiştir.




**Şekil 1.** Gerçek-zamanlı programların çevre ile etkileşimi

Gerçek-zamanlı bir program fiziksel ortamda ortaya çıkan olayları ya kesilmeler aracılığıyla ya da giriş kapılarını tarayarak algılar. Ortamdan giriş kapılarını okuyarak veri toplar ve çıkış kapılarına yazarak ortamdaki parametreleri değiştirir.

Fiziksel ortamın doğası ve uygulamanın özellikleri gerçek-zamanlı programlar için bazı zaman kısıtlamaları empoze etmektedir. Birbirini izleyen iki olay arasındaki minimum zaman uzaklığı fiziksel ortam tarafından belirlenir. Bir olaya tepki göstermek için gerekli kabul edilebilir zaman gecikmesinini de uygulamanın gereksinimleri tarafından tanımlanır. Gerçek-zamanlı programın ortamdaki tüm olayları algılaması, ya da tüm olaylara kabul edilebilir olan zaman gecikmesi içerisinde tepki göstermesi gerekebilir. Bundan dolayı, gerçek-zamanlı programların işlevsel gereksinmelere ek olarak bazı zamanlama gereksinimlerini de sağlaması gerekmektedir.

## 5. Gerçek-Zamanlı Programların Doğrulanması

Sıralı ya da eşzamanlı bir programın $S$ işlevsel davranışı, $\{p\}\, S\, \{q\}$, doğrudan programın kaynak kodundan çıkarılabilir. Fakat, program kodu gerçek-zamanlı programların davranışını belirlemekte tek başına yeterli değildir. Örnegin, bir programın daha hızlı bir işleyici üzerinde işletilmesi, aynı programın daha yavaş bir işleyici üzerinde işletilmesinden farklı bir zamanlama davranışı gösterir. Benzer biçimde bir programı bir işleyici üzerinde tek süreç olarak işletmek, aynı programı aynı işleyici üzerinde başka süreçlerle birlikte işletilmesinden farklı bir zamanlama davranışı sergiler.

İşletim sisteminin zaman atama politikasi o platform üzerinde işletilen programların zamanlama özelliklerini etkiler. Bundan dolayı, program koduna ek olarak, *işletim ortamının* özellikleri de gerçek-zamanlı programların davranışın doğrulanmasını etkileyen bir etmen olarak dikkate almamız gerekmektedir. Bu etmenleri zaman kısıtlamaları ve zaman atamaları başlıkları altında aşağıda ayrı ayrı ele alacağız.

### 5.1. Zaman Kısıtlamaları

Programların belirtimlerine zaman eklemek için, $S = S_1; S_2; ... S_n$ program kodunu $< t = t + T_{S_i} ;>$ biçiminde komutlarla genişletiriz. Burada $t$ zamana karşı gelen değişken, $T_{S_i}$'de $S_i$ program kesiminin verilen işleyici üzerinde çalışabilmesi için gerekli zamandır. Burada $t$ işleyicinin saat imlerine karşılık gelen program değişkenleridir. Genişletilmiş program kodunu $S^t$ ile göstereceğiz.

$$S^t = S; < t = t + T_S > \tag{8}$$

Ardından, programın önkoşul ve ardkoşulunu aşağıdaki gibi zaman kısıtlamaları ile genişleteceğiz:

$$\{p \wedge p^t\}\ S^t\ \{q \wedge q^t\} \tag{9}$$

Burada $p^t$ ve $q^t$ zaman değişkeni $t$'nin $S$ program kesimine giriş ve çıkışı sırasındaki değerine ilişkin önermelerdir. Eğer $S$'nin işletimi $p^t$ önermesinin doğru olduğu bir $t$ zamanında başlarsa, işletim $q^t$ önermesinin doğru olduğu bir zamanda sona erer. Zaman kısıtlamalarını sıralı birleşim kuralına aşağıdaki gibi ekleyebiliriz:

$$\frac{\{p \wedge p^t\}\ S_1^t\ \{r \wedge r^t\}, \{r \wedge r^t\}\ S_2^t\ \{q \wedge q^t\}}{\{p \wedge p^t\}\ S_1^t; S_2^t\ \{q \wedge q^t\}} \tag{10}$$

Bu kural program kodunu aşağıda gösterildiği gibi daha küçük program kesimlerinin birleşimi olarak ele almamıza izin vermektedir:

$$S^t = S_1; S_2; ... S_n; t = t + \sum_{i=1}^{n} T_{S_i} \tag{11}$$

Program kodunu, yukarıda gösterildiği gibi zaman komutları ve zaman kısıtları ile genişletilmesi, $n$ değişkenli sıralı bir programın gerçek-zaman kısıtlarının $n+1$ değişkenli bir sistemle doğrulanabilmesini sağlamaktadır.

### 5.2. Zaman Atamaları

İşletim ortamının zamanlama özellikleri büyük bir öneme sahip olduğundan dolayı, eş zamanlı sistemlerin zamanlama özelliklerini analiz ederken çoklu işlem (*multiprocessing*) ile çoklu programlamayı (*multiprogramming*) birbirinden ayıracağız.

1. *Çoklu İşlem*: *n* örgünün *n* işleyici üzerinde, herbiri ayrı bir işleyici üzerinde işletildiği durumlar. $S_1, S_2, ... S_n$ örgülerinin çoklu işlem platformunda eşzamanlı olarak işletilmesini $S = S_1 \parallel S_2 \parallel ... \parallel S_n$ notasyonu ile göstereceğiz.
2. *Çoklu Programlama*: *n* örgünün tek bir işleyici üzerinde aralıklı olarak işletildiği durumlar. $S_1, S_2, ... S_n$ örgülerinin çoklu programlama platformu üzerinde eşzamanlı olarak işletilmesi $S = S_1 \parallel\parallel S_2 \parallel\parallel ... \parallel\parallel S_n$ notasyonu ile gösterilir.

Çoklu İşlem en yüksek düzeyde koşutluk içerir. Bu sistemlerde her işleyici, dolayısıyla her $S_i$ örgüsü ayrı bir zaman değişkeni $t_i$ atayacağız. Dolayısıyla, çoklu işlem platformunda işletilen iki örgünün zamanlama davranışı şöyle betimleyebiliriz:

$$\frac{\{p_1 \wedge p_1^{t_1}\}\, S_1^{t_1}\, \{q_1 \wedge q_1^{t_1}\}, \{p_2 \wedge p_2^{t_2}\}\, S_2^{t_2}\, \{q_2 \wedge q_2^{t_2}\}}{\{p_1 \wedge p_1^{t_1} \wedge p_2 \wedge p_2^{t_2}\}\, S_1^{t_1} \parallel S_2^{t_2}\, \{q_1 \wedge q_1^{t_1} \wedge q_2 \wedge q_2^{t_2}\}} \tag{12}$$

Her işleyici kendi sistem saatine sahip olduğundan dolayı, çoklu işlem sistemlerinde her işleyici için ayrı bir zaman değişkeni $t_i$ tutuyoruz. Bu değişken yalnızca $S^{t_i}$ tarafından güncellenmekte, ve dolayısı ile $S_i$'nin işletim zamanını tutmaktadır.

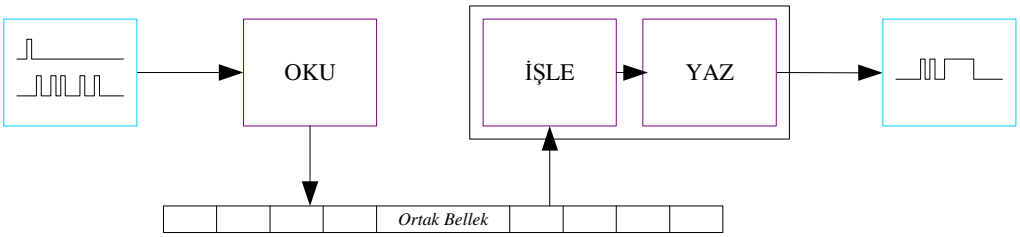
Çoklu programlama sistemleri eşzamanlı örgülerin bir zaman atama politikasına bağlı olarak aralıklı olarak işletilmesi temeline dayanır. Dolayısıyla, çoklu programlama sistemlerinin zamanlama davranışı ancak bir zaman atama politikasına göre analiz edilebilir. Çoklu programlama platformunda işletilen iki örgünün zamanlama davranışı şöyle gösterilebilir:

$$\frac{\{p_1 \wedge p_1^{t}\}\, S_1^{t}\, \{q_1 \wedge q_1^{t}\}, \{p_2 \wedge p_2^{t}\}\, S_2^{t}\, \{q_2 \wedge q_2^{t}\}, \textit{zaman atama politikası}}{\{p_1 \wedge p_1^{t} \wedge p_2 \wedge p_2^{t}\}\, S_1^{t} \parallel\parallel S_2^{t}\, \{q_1 \wedge q_1^{t} \wedge q_2 \wedge q_2^{t}\}} \tag{13}$$

Dikkat edilirse, çoklu programlama ortamında işletilen örgüler aynı zaman değişkeni *t*'yi paylaşmaktadırlar. Bir başka deyişle, *t* tüm örgüler için paylaşılmış değişken olarak kullanılır. Dolayısı ile (7) nolu eşitlikte verilen örgülerarası etkileşim faktörleri buradaki zaman değişkeni *t* için de geçerli olmaktadır. Bu zaman aralıkları, zaman atayıcı tarafından belirlendiğinden dolayı, sistemin zaman atama politikasının da örgülerarası etkileşimin bir parçası olarak ele alınması gerekmektedir. Kesilmelerin ele alınması, aralıklı işletimin en yalın biçimi olarak görülebilir.

Örneğin, gelen kesilmelere bağlı olarak çevre birimlerinden gelen verileri okuyan ve bunları işleyerek tekrar bir çıkış kapısına yazan bir program düşünelim. Böyle bir program Şekil 2'deki gibi, birisi dışardan veri okuyan, ötekisi de bu veriyi işleyerek dışarı yazan iki örgüden $S = S_O \parallel (S_I ; S_Y)$ oluşabilir. Bu sistemin doğruluğu şöyle gösterilebilir:

$$\frac{\{p_1 \wedge p_1^{t_1}\}\, S_O^{t_1}\, \{q_1 \wedge q_1^{t_1}\}, \{p_2 \wedge p_2^{t_2}\}\, S_I^{t_2} ; S_Y^{t_2}\, \{q_2 \wedge q_2^{t_2}\}}{\{p_1 \wedge p_1^{t_1} \wedge p_2 \wedge p_2^{t_2}\}\, S_O^{t_1} \parallel (S_I^{t_2} ; S_Y^{t_2})\, \{q_1 \wedge q_1^{t_1} \wedge q_2 \wedge q_2^{t_2}\}} \tag{14}$$




**Şekil 2.** Gerçek-zamanlı bir program örneği

## 6. İşletim Ortamından Beklenenler

Sıralı ve eşzamanlı programların işlevsel davranışının doğrudan kaynak koddan elde edilebildiğini, fakat gerçek-zamanlı bir programı tanımlamak için kaynak kodun yeterli olmadığını yukarıda gördük. Bundan dolayı, gerçek-zamanlı bir sistemin zamanlama davranışını doğrulayabilmek için $\{p\}\ S\ \{q\}$ üçlüsünü, işletim ortamının özelliklerini de yansıtacak biçimde genişlettik.

1. *Zaman komutları*: İlk olarak $t$ adlı bir zaman değişkeni yaratarak $S = S_1; S_2; \ldots S_n$ program kodunu $< t = t + T_{S_i}; >$ biçimindeki zaman komutları ile genişletilmiştir.
2. *Zaman önermeleri*: Daha sonra $p$ önkoşulu ile $q$ ardkoşuluna, $S$ program kesimine giriş ve çıkışta $t$'nin değeri üzerine $p^t$ ve $q^t$ önermeleri eklenmiştir.
3. *Zaman atama politikaları*: En son olarakta çoklu programlama sistemleri, çoklu işlem sistemlerinden ayrılmış, ve eşzamanlılık kuralı zaman atama politikaları ile genişletilmiştir.

Bu üç ekleme sistemin üzerinde çalıştığı işleyici ve işletim sistemi üzerine bazı varsayımlar içermektedir. İşletim ortamı ile ilgili bu beklentiler Tablo 1'de özetlenmiştir:

**Tablo 1.** Gerçek-zamanlı programların doğrulanması ve işletim ortamı

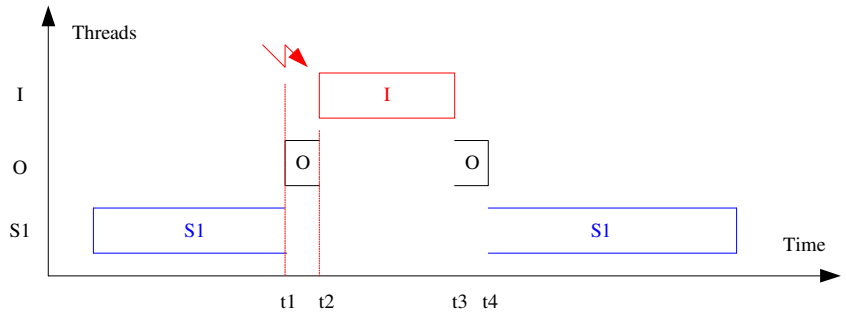
| **Eklemeler** | **İşletim Ortamına ilişkin Gereksinimler** |
|---|---|
| Zaman komutları | Komutların işletim zamanının belirlenebilir olması<br>İşletim sisteminin hizmetleri belirlenebilir zaman sınırları içinde verebilmesi |
| Zaman önermeleri | İki ardışık olay arasındaki zaman uzaklığının biliniyor olması<br>Bir olaya karşı tepki gösterebilmek için kabul edilebilir zaman aralığının biliniyor olması |
| Zaman atama politikaları | Kesilme gecikmesinin belirlenebilir olması<br>Bağlam anahtarlama zamanının belirlenebilir olmasi<br>Eşitlikçi değil, önceliğe dayalı zaman atama politikası olması |

Program kodunu zaman komutları ile genişletebilmek için, komutların işletim zamanının belirlenebilir olması gerekmektedir. İşleyiciler genellikle komutlar için belirlenebilir işletim zamanları sağlamaktadırlar, fakat bunun istisnaları da bulunmaktadır. Bu istisnalardan birisi önbellek, bellek ya da görüntü belleğe erişmek için harcanan zamanlardaki farklılıklardır. Doğrulama yöntemimizde zaman komutlarının bulunması, işletim sisteminin hizmet sağlama süresinin de belirlenebilir olmasını gerektirmektedir. Örneğin bir semaforun kilitlenmesi, bu kilitin

açılması, ya da bir iletinin gönderilmesi belirlenebilir zaman sınırları içinde olmalıdır. Bu hizmetlerin verilme süresi sistemin durumuna, örneğin semafor için bekleyen görev sayısına bağlı olmamalıdır. Gerçek-zamanlı işletim sistemleri, genel-amaçlı işletim sistemlerinden farklı olarak, programlara belirlenebilir zamanlama arabirimi sunmalıdır.

Önkoşul ve ardkoşulların zaman önermeleri ile genişletebilmek için fiziksel ortamın zamanlama davranışını bilmemiz gerekir. Bu iki ardışık olay arasındaki en küçük zaman aralığını ve bir olaya tepki gösterebilmek için kabul edilebilir en büyük zaman gecikmesini içerir. Bunlardan ilki, bir olay meydana geldikten sonra sistemin ne çabuklukta ortam parametrelerini okuması gerektiğini, ikincisi de sistemin okuduğu verileri ne kadar hızla işleyip yanıt vermesi gerektiğini tanımlar.

Zaman atama politikasını çoklu programlama sistemlerinin zamanlama davranışı analizine dahil edebilmek için, zaman atayıcının zamanlama davranışını bilmemiz gerekir. Buna kesilmeleri ele alma ve örgülerarası bağlam anahtarlama yapmak dahildir. Şekil 3'de gösterildiği gibi işletim sistemi kesilme hizmet yordamını (I) çağırmadan önce ve kesilme hizmet yordamı işletildikten sonra bazı işlemler (O) yapabilir. Bu tür sistemlerin zamanlama analizlerinin yapılabilmesi için kesilme gecikmesinin ve bağlam-anahtarlama zamanının belirlenebilir olması gerekmektedir.




**Şekil 3.** Bağlam-anahtarlamada işletim sisteminin rolü

## 6. Sonuç

Bu bildiride, sıralı ve eşzamanlı programlar için geliştirilen aksiyomatik doğrulama sistemi gerçek-zamanlı programların doğrulamasını yapabilecek biçimde genişletilmiştir. Bu teknik:

- $n$ değişkenli sıralı bir programın gerçek-zaman kısıtlarının $n+1$ değişkenli bir sistemle,
- $n$ değişkenli ve $m$ örgülü çoklu işlem sisteminin $n+m$ değişkenli bir sistemle,
- $n$ değişkenli ve $m$ örgülü çoklu programlama sisteminin, birisi örgülerarasında paylaşılan olmak üzere, toplam $n+1$ değişkenli bir sistemle doğrulanabilmesini sağlamaktadır.

Pratik uygulamalarda kullanılabilmesi için, geliştirdiğimiz tekniğin kullandığı parametreler, işleyici ve işletim sistemlerinin özellikleri ile ilişkilendirilmiştir. Bu doğrulama tekniğinin pratik uygulamaları sonraki bir bildiride incelenecektir.

## Kaynakça


1. Erbas C., Tanik M.M., "Modeling and Verification of Real-Time Systems", *Transactions of the SDPS, Journal of Integrated Design*, Eylül, 2003.
2. Hoare C.A., "An axiomatic basis for computer programming", *Comm. ACM*, 12(10), s. 576-580, 1969.
3. Hooman J., "Compositional verification of real-time systems using extended Hoare triples", *Real-Time: Theory in Practice*, REX Workshop Proceedings, LNCS 600, Springer-Verlag, s. 185-222, 1991.
4. Owicki S., Gries D., "An axiomatic proof technique for parallel programs", *ACTA Informatica*, 6, s. 319-340, 1976.
5. Schneider F.B., Bloom B., Marzello K., "Putting time into proof outlines", *Real-Time: Theory in Practice*, REX Workshop Proceedings, LNCS 600, Springer-Verlag, 1991.