Gerçek-Zamanlı Uyarlamacı Görev Planlaması Tekniğinin RT-Linux'ta Uygulanması [1]

Tolga Ayav, Sinan Yılmaz
İzmir Yüksek Teknoloji Enstitüsü, Bilgisayar Mühendisliği Bölümü, 35430, Urla İzmir
Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü, 35100, Bornova İzmir
TolgaAyav@iyte.edu.tr, Yilmaz@staff.ege.edu.tr

Özetçe

Bu çalışma tamamlanmamış hesaplamalar yöntemine dayalı olan bir uyarlamacı gerçek-zamanlı geribesleme kontrollü görev planlaması tekniğinin ("Real-Time Feedback Control Scheduling"), Gerçek-Zamanlı Linux (RT-Linux) üzerindeki uygulamasını sunmaktadır. Zorunlu ve seçmeli alt-görevlerden oluşan ve geribesleme kontrollü "rate-monotonic" tekniğine göre planlanan iki-versiyonlu görev modeli seçilmiştir. Oransal-Integral-Türev (PID) kontrol, seçmeli alt-görevlerin çalıştırılması veya reddedilmesinde gerekli geribesleme stratejisini sağlamak için kullanılmıştır. Oluşturulan bu model RT-Linux işletim sistemi üzerinde uygulanmış ve sistem performansı yapay bir iş yükü altında test edilerek önerilen görev planlama modelinin sistemi verilen bir CPU kullanım oranı seviyesinde kararlı halde çalıştırabildiği gösterilmiştir.

Abstract

Implementation of Real-Time Adaptive Scheduling Technique on RT-Linux

This paper presents an RT-Linux implementation of real-time feedback control scheduling technique based on imprecise computations. We consider two-version tasks: one defined as mandatory and the other defined as optional for scheduling according to a feedback control rate-monotonic algorithm. A Proportional-Integral-Derivative (PID) control action provides the feedback strategy for deciding on the execution or rejection of the optional sub-tasks. The proposed model is implemented and evaluated on RT-Linux. The experimental results show that the proposed model can successfully keep the system stable at a desired level of CPU utilization.

1. Giriş

Geleneksel gerçek-zamanlı görev planlaması tekniklerinin çoğu iş yükünün ve çalışma-zamanı ortamının durağan ve kararlı olduğu varsayımına, bunun sonucu olarak da bu iş yükünün ve çalışma-zamanı ortamının tasarım sürecinde tamamıyla karekterize edilmesi esasına dayanmaktadır. Bu yaklaşımla birlikte iş yükünün modellemesinde olası en kötü durumların gözönünde bulundurulması gerçek-zamanlı sistemlerde boş bir kapasite doğurmaktadır (örn."rate-monotonic" görev planlaması tekniğinde işlemci kaynağının %69'unun kullanılması gibi).

[12], [3], [4] ve [5]'e göre gelecek kuşak gerçek-zamanlı sistemler, daha açık ve önceden tahmin edilemeyen ortamlarla karşı karşıya kalacak, bunun sonucunda da özel hedefe yönelik ve güvenlik açısından kritik fonksiyonların zaman kısıtlamalarını karşılamanın yanısıra daha karmaşık ve uyarlamacı yapılara sahip olacaktır. Örneğin, akıllı robot sistemlerinin iş yükünde, robot kon-

[1] Bu çalışma İzmir Yüksek Teknoloji Enstitüsü ve Institute Aéronautique et Spatial tarafından ortak yürütülen "Uydu Haberleşme ve Bağıntılı Laboratuvarlar Projesi" kapsamında desteklenmektedir.

trol algoritmalarının (algılayıcı yorumlama, hareket planlama, ters kinematik ve dinamik algoritmaları vb.) oldukça değişken çalışma süreleri sebebiyle yüksek değişimler ve aşırı yüklenme durumları meydana gelebilmektedir [7].

Geleneksel algoritmaların gerçek-zamanlı sistemlerdeki bu yeni gereksinimlere yeterli cevap veremeyeceği düşüncesiyle mevcut araştırmalar, yukarıda anılan bu boş kapasitenin değerlendirilmesi ve önceden tahminlenemeyen ortamlarda güvenli çalışmanın sağlanması amacıyla uyarlamacı yaklaşımlar getirilmesi üzerine odaklanmaktadır. Bir çoğu, tamamlanmamış hesaplamalar yöntemine dayalı olan uyarlamacı görev planlaması teknikleri bu nedenle ortaya atılmış olup şu ana kadar çeşitli çalışmalarda incelenmiştir. Bu çalışmanın da üzerinde duracağı geribesleme kontrollü görev planlaması tekniği [12], [4], [11] ve [6]'da incelenmiş olmasına rağmen yine bu çalışmalarda ortaya atılan ve cevap bekleyen bir çok soruyu da beraberinde getirmektedir. Bu sorulardan bazıları kontrol sistemindeki kontrol edilen değişken, manipüle edilen değişken ve ayar noktasının ne olması gerektiği, PID kontrolün yeterli olup olmadığı veya doğru kontrol algoritmasının ne olması gerektiği, kontrol sisteminin kararlılık kriterlerinin ne olduğu ve geribesleme kontrol ve kontrolörün sisteme getirdiği ek yükün ne derece etkili olduğudur.

Bu çalışmada, yeni bir geri besleme kontrollü "Rate-Monotonic" (RM) görev planlaması tekniği sunulacak ve bu tekniğin RT-Linux işletim sistemi üzerindeki uygulaması anlatılacaktır. Bölüm 2.1'de tamamlanmamış hesaplamalar yönteminden kısaca bahsedilecektir. Bölüm 2.2 ve 2.3'te çalışmada kullanılan görev modeli ve iş yükü modeli sunulacak, ardından geribesleme kontrollü "rate-monotonic" görev planlama detaylı olarak anlatılacaktır. Burada görüleceği gibi kontrol stratejisi olarak PID seçilmiştir. Bundaki amaç bu stratejinin getireceği ek yükün az olması ve daha önceki çalışmalarda sunulan PID kontrolün sistemi kararlı halde çalıştırmadaki yeterliliğidir. Bölüm 3'te sistemin RT-Linux üzerindeki uygulamasının detayları verilmiş, son olarak da bölüm 4'te denemeler ve elde edilen sonuçlar aktarılmıştır.

2. Geribesleme Kontrollü "Rate-Monotonic" Görev Planlaması Mimarisi

Bu bölümde, RM görev planlaması ve geribesleme kontrol tekniklerini bütünleştiren geribesleme kontrollü "rate-monotonic" (GK-RM) görev planlaması konusu incelenecektir. İlk olarak, bu bütünleşik tekniğin altında yatan tamamlanmamış hesaplamalar yöntemine kısaca değinilecek, daha sonra denemelerde kullanılan görev modelleri ve iş yükü tanıtılacak, son olarak da GK-RM görev planlaması detaylı olarak sunulacaktır.

2.1. Tamamlanmamış Hesaplamalar Yöntemi

Gerçek-zamanlı sistemlerin aşırı yüklendigi durumlarda meydana gelebilecek planlama hatalarını tolere etmenin bir yolu tüm *önemli* görevlerin zamanında tamamlanmasıdır. Diğer bir deyişle, işletim sistemi tüm görevleri eşit olarak ele alacağına, programcı bazı önemli görevleri "zorunlu", bazı önemli olmayan görevleri de "seçmeli" olarak sisteme tanıtabilir. Zorunlu görevler daima kendileri için tanımlanan son zamandan önce çalışmalarını bitirmek zorundadırlar. Seçmeli görevlerse yeterli CPU kaynağı olduğunda çalıştırılacak olup, yüklenme durumlarında olası zamanlama hatalarının önlenmesi için atlanabilecektir. Herhangi bir görevin tüm kısıtlamalara uygun bir biçimde tamamıyla çalıştığında ürettiği sonuç tam veya eksiksiz ("precise") sonuç olarak adlandırılabilir. Eğer görev atlanıyor veya sadece bir kısmı çalıştırılıyorsa görev tam olarak sonlandırılmamış olur ve bu şekilde görevin erken sonlandırılmasıyla elde edilen ara sonuç tamamlanmamış ("imprecise") olarak adlandırılır. Örneğin, bir radardan gelen sinyalin işlenmesinde kullanılan filtreleme görevi aşırı yüklenme sırasında bir veya birkaç örnekleme dönemi boyunca çalıştırılmayabilir, ve bir önceki değer veya sensörden gelen işlenmemiş değer kullanılabilir. Tamamlanmamış hesaplamalar yönteminin "Milestone", "Sieve" ve "Multiple-version" gibi uygulama metodları [1] ve [2]'de detaylı olarak incelenmiş olup aşağıda kısaca bahsedilecektir.

2.1.1. Uygulama Metodları

Bir görevin ürettiği ara sonucun doğruluğu görev çalıştırılmaya devam ettikçe artıyorsa, bu görev tekdüze ("monotone")'dir. Eğer görev tamamlanmadan önce sonlandırılırsa bu görevden elde edilen ara sonuç, sonlandırmadan önce üretilen tüm ara sonuçların içerisinde en iyisi olacaktır. Tekdüze görevler nümerik hesaplamalar, istatistiksel tahminleme, buluşsal arama, sıralama ve veritabanı sorgulamaları için uygundur. Bir tekdüze görevden ara sonuç elde edebilmek için çalıştığı sürece bu görevin ürettiği ara sonuçların uygun anlarda kaydedilmesi gerekir. En son kaydedilen ara sonuç, görev önceden sonlandırıldığında kullanıma hazır durumda olmalıdır. Bu metod "milestone" olarak adlandırılır ve açıktır ki en büyük dezavantajı ara sonuçların kaydedilmesinin sisteme getireceği ek yüktür.

Bazı uygulamalarda tüm görevlerin tekdüze olması mümkün değildir. Bu durumda "sieve" metodu uygulanabilir. Bu metodta her bir görev yukarıda radar örneğinde olduğu gibi tam olarak çalıştırılır veya atlanır. "Sieve" görevlerin 0/1 kısıtlamasını karşıladığı söylenir ve daha az esnek bir görev planlamasına sebep olurlar.

"Milestone" ve "sieve" metodunun uygulanamadığı durumlarda ise "Multiple-version" metodu uygulanabilir. Bu metoda göre görev en az iki versiyondan oluşur: birincil versiyon ve öteki versiyon(lar). Görevlerin birincil versiyonları istenilen tam sonucu üretirler fakat daha uzun çalışma süresine sahiptirler. Öteki versiyonlar daha kısa çalışma zamanın sahiptirler fakat tamamlanmamış sonuç üretirler. Geçici bir aşırı yüklenme durumunda, eğer tüm birincil versiyonların zamanında tamamlanması imkansız ise, sistem bazı görevlerin birincil versiyonları yerine öteki versiyonlarını çalıştırabilir. Daha esnek bir görev planlaması için fazla sayıda versiyon kullanılması durumunda ise tüm versiyonların saklanmasının getireceği ek yük dezavantajı kendini gösterir. Önceki tanımlamalar doğrultusunda, öteki versiyon sadece zorunlu alt-görevi içermekteyken, birincil versiyon hem zorunlu hem de seçmeli alt-görevi kapsamaktadır. Bu çalışmada ikili-versiyon metodu kullanılmıştır.

Görev planlaması açısından bakılacak olursa, tamamlanmamış hesaplamalar yönteminin tam ve etkin çalışabilmesi için tüm zorunlu görevlerin sınırlı bir kaynak ihtiyacına sahip olması gerekir. Bunun da ötesinde, zorunlu görevlere son zamanlarından önce çalışmalarını tamamlayabilmeleri için yeterli işlemci zamanının verilmesi gerekir. Daha sonra sistem geri kalan zamanını mümkün olduğunca çok sayıda seçmeli görevi çalıştırmak için kullanabilir. Belirli bir performansı garantilemek için, zorunlu görevler daha geleneksel bir algoritmaya göre planlanmalıdır. Diğer yandan, seçmeli görevleri planlamak için daha dinamik bir politika uygulanabilir. Bu çalışma, zorunlu görevler için "rate-monotonic", seçmeli görevler içinse geribesleme kontrollü görev planlaması tekniklerini önermektedir.

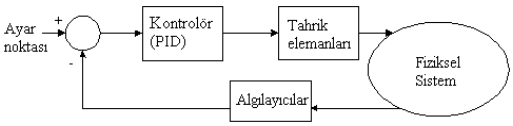
2.2. Görev Modelleri ve Çalışma Yükü

Geribesleme kontrollü görev planlamasını uygulayabilmek için gerçek zamanlı sistemlerin klasik görev tanımlaması bir kaç küçük eklemeyle kullanılabilir. q görevin oluşturduğu küme $\{\tau_1, \tau_2, ..., \tau_q\}$ olsun ve her bir görev, son zaman d_i (peryodik görevlerde peryod T_i olarak ele alınabilir), öncelik p_i, çalışma süresi C_i ve seçmeli kısım için izin biti e_i gibi bazı parametrelerle karakterize edilsin. Daha önce söylendiği gibi τ_i zorunlu alt-görev M_i ve seçmeli alt-görev O_i'den oluşmaktadır. M_i ve O_i'nin çalışma süreleri ise $C_{i,m}$ ve $C_{i,o}$ olsun. Bu durumda $C_{i,m} + C_{i,o} = C_i$. Klasik katı gerçek-zamanlı sistemler $C_{i,o} = 0, \quad \forall i$ koşuluna sahip olup tamamlanmamış hesaplama modelinin özel bir durumudur. Benzer şekilde, yumuşak gerçek-zamanlı sistemler ise $C_{i,m} = 0, \quad \forall i$ özel durumudur. İkili-versiyon metoduna göre görevler aşağıdaki yapıya sahip olurlar:

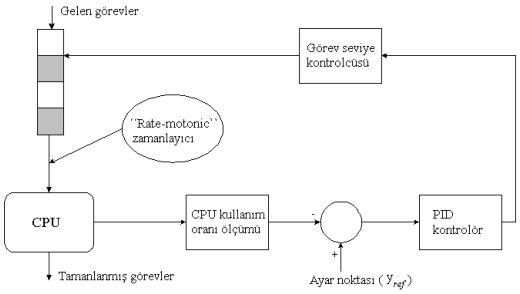
```
τ_i {
   Zorunlu_Görev_i();

   eğer(τ_i->e_i==DOĞRU)
      Seçmeli_Görev_i();
}
```

Eğer bir görev, zorunlu ve seçmeli olarak iki kısma ayrılamıyorsa bu durumda aşağıdaki gibi iki versiyonlu olarak da yazılabilir. Birinci versiyon, zorunlu ve seçmeli alt-görevleri içerecek şekilde ikincisinden daha fazla çalışma süresine sahiptir.



Şekil 1: Geribesleme Kontrol Sistemi



Şekil 2: GK-RM Görev Planlama

```
τi {
   eğer(τi->ei==DOĞRU)
      Görev_Versiyon_Ai();
   değilse
      Görev_Versiyon_Bi();
}
```

Daha sonra açıklanacağı gibi, görevler e_i bitlerine göre hangi versiyonu çalıştıracağına veya seçmeli alt-görevi çalıştırıp çalıştırmayacağına karar verir. Bu bitler ise görev seviye kontrolörünün kontrolü altındadır. Bu çalışmadaki tüm deneylerde, tablo 1'de yer alan parametreler kullanılmıştır. Ayrıca, tüm görevler peryodik yapılmış ve çalışma süreleri $C_{i,m}$ ve $C_{i,o}$ birörnek olarak alınmıştır.

Tablo 1: **Deneylerde kullanılan iş yükü ve sistem parametreleri**

Değişken	Değerler
q (görev sayısı)	20
$C_{i,m}$, $C_{i,o}$	uniform[15μs, 25μs]
$T_{i\in\{1,\ldots,q\}}$	{ 1s,2s,2s,2s,100ms,120ms, 130ms,170ms,12ms,12.2ms, 13.4ms,18.6ms,1.2ms,8.8ms, 6.6ms,9.5ms,100μs,110μs, 120μs,130μs}
T	100 ms
K_p, K_i, K_d	0.1, 2, 0.1
IW, y_{ref}	100, 0.7

2.3. GK-RM Görev Planlaması

"Rate-monotonic" görev planlaması tekniği, "static priority driven preemptive" kategorisinde yer alan bir teknik olup şu şekilde işler: Her göreve çalışma sıklığıyla orantılı olacak şekilde bir öncelik değeri atanır. Görev planlayıcısı tüm hazır görevler içerisinde en yüksek önceliğe sahip olanı seçer ve CPU kontrolünü ona verir. "Rate-monotonic" literatürde önemli yeri olan, iyi tanımlanmış bir tekniktir ve CPU kullanım oranı aşağıdaki eşitsizliği sağladığı sürece tüm zorunlu alt-görevlerin son zamanlarından önce çalışmalarını tamamlamasını garanti eder (Peryodik görevler için $f_i = 1/T_i$ alınır) [9].

$$\sum_{i=1}^{q} C_{i,m} f_i \leq q(2^{1/q} - 1) \qquad (1)$$

RT-Linux çekirdeğinin varsayılan görev planlaması olan "Rate-monotonic", aşağıdaki sözde-kodla ifade edilmektedir.

```
Görevler= {τ1, τ2, ··· , τq},
τi = {di, Ci, pi, ei, ··· },
E={[tT,(t+1)T] zaman aralığında
kuyrukta hazır konumdaki görevler}.


''Rate-Monotonic'' Görev Planlayıcı{
  E kümesini hesapla;
  eğer(E==∅)
    Yeni_Görev=Std._Linux_Çekirdek;
  değilse
    Yeni_Görev=τ arg( max i∈{1,2,...,q} pi);
  Yeni_Görev'e geç;
}
```

Öte yandan, seçmeli görevlerden hangilerinin veya kaçının çalıştırılacağına karar verilmesi çok daha

karmaşık bir problemdir. Geribesleme kontrollü görev planlaması bunun çözümü için ortaya atılan, klasik kontrol teorisine dayanan bir metodtur ve bu alana yeni bir yaklaşım getirir: Kapalı-döngü görev planlama. "Rate-monotonic" ve "earliest-deadline-first" gibi algoritmalar açık-döngü çalışırlar. Burada açık-döngü şunu ifade eder: Tasarım sırasında öncelikler ve görev planlamaları bir kere belirlendikten sonra bunların üzerinde çalışma-zamanında geribeslemeye dayalı herhangi bir değişiklik yapılmaz. Bu algoritmalar iş yükünün önceden modellenebildiği statik ve dinamik sistemlerde iyi performans göstermekle birlikte önceden tahminlemeyen dinamik sistemlerde başarısız olmaktadır. Geribesleme kontrollü görev planlaması fikri şekil 1'de görülen bir kontrolör, kontrol edilecek fiziksel sistem, tahrik ve algılayıcı elemanlardan oluşan tipik bir kontrol sisteminden gelmektedir.

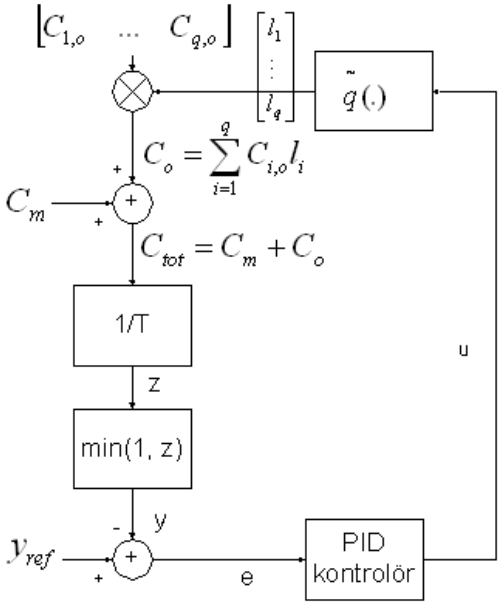
Kontrol teorisini uygulayabilmek için ilk olarak kontrol sisteminin elemanlarının bu çalışmada sunulan sistemdeki karşılıklarını bulmak gerekir. Bu uyarlama işinde çeşitli seçenekler bulunmakla birlikte [4], bu çalışmada kontrol edilen değişken olarak CPU kullanım oranı seçilmiştir. Katı gerçek-zamanlı sistemlerde görevlerin zaman sınırlarını aşmalarına kesinlikle izin verilmediğinden, CPU kullanım oranının da %100'e ulaşmaması istenir. Bunun istenmesine bir başka neden de şudur: Geribesleme olarak ölçülen CPU kullanım oranı, %100'e ulaştığında sistemin ne kadar yüklü olduğuna dair gerçek bilgiyi veremez ve bu da kontrol sisteminde doğrusal olmayan bir etki yaratır. Örneğin aşırı yüklenmiş bir sistemde talep edilen CPU kaynağı ihtiyacı %150 olabilir ancak ölçülebilen değer en fazla %100 olacağından, ayar noktasının %99 olması durumunda kontrol sistemindeki hata değeri de %-1 olacaktır. Aslında hata değerinin %99-%150=%-51 olması gerekirken %-1 olması kontrolörün gerekli tepkiyi verememesine sebep olur. Bu sebeple, ayar noktası (y_{ref}) %100'den yeteri kadar uzak seçilmelidir [2][10]. Bu çalışmada y_{ref} %70 olarak alınmıştır.

CPU kullanım oranının verilen ayar noktasında tutulabilmesi için kontrol ünitesi ve kontrol ünitesinin çıkışını girdi olarak alan görev seviye kontrolörü bir sonraki örnekleme aralığında çalıştırılacak seçmeli görevlerin sayısını ayarlar. Belirtilmelidir ki bu kontrol mekanizmasının getireceği ek yükün kabul edilebilir bir seviyede tutulabilmesi için kontrol ünitesinin çıkışı $1/T$ sıklığında güncellenir. T örnekleme dönemi veya kontrol dönemi olarak anılır ve bu çalışmada sabit $100ms$ olarak alınmıştır. T'nin değişken olmasına ve o anki iş yüküne göre uyarlamacı bir yaklaşımla ayarlanmasına ilişkin bir çalışma [13]'te sunulmaktadır. Şekil 2 geri besleme kontrollü "rate-monotonic" görev planlayıcı sisteminin yapısını göstermektedir. Burada görülen kontrol ünitesinde Oransal-Integral-Türev ("Proportional-Integral-Derivative" [PID]) fonksiyonu iki sebeple kullanılmıştır. Birincisi, uygulanması kolay olan bu fonksiyonun sisteme getireceği ek yük azdır. İkincisi, [3], [4] ve [12]'de belirtildiği gibi PID kontrol, sistemi kararlı halde tutabilmek için yeterlidir. PID kontrol fonksiyonu aşağıdaki sözde-kod ile açıklanmaktadır.

```
PID kontrolör{
   e_t=Ayar noktası-CPU kullanım oranı;
   çıkış=K_p e_t + K_i Σ_{i=t-IW}^{t} e_i + K_d(e_t - e_{t-1});
   eğer(çıkış < -1/2q) çıkış=-1/2q;
   eğer(çıkış > 1 + 1/2q) çıkış=1 + 1/2q;
   e_{t-1}=e_t;
}
```

Bir görev CPU kontrolünü aldığında ilk olarak zorunlu alt-görevini çalıştırır, sonra görev yapısında yer alan e_i bitine göre seçmeli alt-görevini çalıştırıp çalıştırmayacağına karar verir. Bu bitler görev seviye kontrolörünün kontrolü altındadır. Görev seviye kontrolörü $[0, 1]$ aralığında değişen PID çıkışını alır ve bu değere göre seçmeli alt-görevlerini çalıştırması gereken görevlerin e_i bitlerini "1" konumuna, çalıştırmayacak olanlarınkini ise "0" konumuna getirir. Bu fonksiyon aşağıdaki sözde-kod ile açıklanmaktadır.

```
Görev Seviye Kontrolörü{
   döngü i :  1'den q'ya
      eğer( PID Çıkışı ≥ (2p_i - 1)/2q )
         τ_i → e_i=1;
      değilse
         τ_i → e_i=0;
   döngü sonu
}
```



Şekil 3: GK-RM görev planlayıcısının sistem modeli

Sistemin çalışması şekil 3'te verilen matematiksel model yardımıyla daha net olarak açıklanabilir. Bu model bölüm 2.2'de açıklanan ve tablo 1'de verilen işyükü modeline göre oluşturulmuştur. Modelde görülen tüm sinyallerin bir kontrol döneminde alacakları değerler aşağıda hesaplanacaktır. Bu hesaplamalarda $t \in \mathbb{N}$ kesikli zaman endeksidir, gerçek zaman $\tilde{t}$ ise Tt'ye eşittir. Çalışma dönemi $T_i(t)$ ile belirtilen görev τ_i'nin bir kontrol dönemindeki çalışma sayısı $N_i(t) = T/T_i(t)$'dir. Bir kontrol dönemindeki toplam görev sayısı

$$N(t) = \sum_{i=1}^{q} N_i(t) = \sum_{i=1}^{q} \frac{T}{T_i(t)} \qquad (2)$$

τ_i'nin zorunlu ve seçmeli olmak üzere iki alt-görevden oluştuğu ve bu alt-görevlerin çalışma sürelerinin $C_{j,m}$ ve $C_{j,o}$ olduğu gözönüne alındığında, bu görevin t zamanındaki zorunlu ve seçmeli tüm alt-görevlerinin toplam çalışma süreleri şu şekilde hesaplanır:

$$C_{i,m,T}(t) = \sum_{j=1}^{N_i(t)} C_{j,m}(t),\ C_{i,o,T}(t) = \sum_{j=1}^{N_i(t)} C_{j,o}(t) \qquad (3)$$

Bir kontrol dönemindeki tüm görevlere ait zorunlu alt-görevlerin toplam çalışma süresi ise

$$C_m(t) = \sum_{i=1}^{q} C_{i,m,T}(t) \qquad (4)$$

Tüm seçmeli alt-görevlerin toplam çalışma süresi görev planlama politikasına bağlıdır. Bu çalışmada seçmeli alt-görevler "sieve" metoduna göre planlandığına göre bu alt-görevler kontrolörün çıkışına bağlı olarak tümüyle çalıştırılacak veya gözardı edilecektir. Bu yapı, $l^k(t) \in \{0,1\}^q$, yani $l^k(t) = [1, 1, ..., 1, 0, 0, ..., 0]^T$ vektörü ile ifade edilebilir. Öyle ki PID ve görev seviye kontrolörleri tarafından üretilen bu vektördeki k adet "1", bir sonraki dönemde çalıştırılacak olan ve en yüksek önceliklere sahip seçmeli alt-görevlere karşılık gelmektedir. Buna göre bir sonraki dönemde $\tau_1, \tau_2, ..., \tau_k$ görevlerinin seçmeli alt-görevleri çalışır. Formül (5), $t+1$ zamanında çalışacak olan tüm seçmeli alt-görevlerin toplam çalışma süresini vermektedir ($\tau_{1,o}, \tau_{2,o}, ...\tau_{q,o}$ görevlerinin $p_1 > p_2 > ... > p_q$ olacak şekilde sıralandığı varsayımıyla).

$$\begin{aligned} C_o(t) &= \left[\ C_{1,o,T}(t)...C_{q,o,T}(t)\ \right] . \begin{bmatrix} l_1(t) \\ l_2(t) \\ \vdots \\ l_q(t) \end{bmatrix}^{(k)} \\ &= \sum_{i=1}^{q} C_{i,o,T}(t).l_i(t) = \sum_{i=1}^{k} C_{i,o,T}(t) \end{aligned} \qquad (5)$$

Şekil 3'teki $z(t)$ ve CPU kullanım oranı $y(t)$ ise

$$z(t) = \frac{C_m(t) + C_o(t)}{T},\ y(t) = min\{1,\ z(t)\} \qquad (6)$$

Hata sinyali $e(t)$, ayar noktası y_{ref} ile ölçülen CPU kullanım oranının arasındaki farktır:

$$e(t) = y_{ref} - y(t) \qquad (7)$$

Kontrolör bu hata sinyalini işleyerek $[0, 1]$ aralığında bir değer üretir. Kontrolör PID ise bu

durumda $u(t)$ şu formüle göre hesaplanır [8]:

$$u(t) = K_p e(t) + K_i \sum_{i=t-IW}^{t} e(i) + K_d(e(t) - e(t-1)) \quad (8)$$

Ardından, PID kontrolörün çıkışı şekil 3'te görülen $\widetilde{q}(.)$ bloğu tarafından nicelendirilir. Bu işlem sözde-kodu daha önce verilen görev seviye kontrolörü tarafından gerçekleştirilir. Nicelendiricinin fonksiyonu formül (9)'de verilmiştir. Nicelendirici $l^{(k)}(t)$ vektörünü elde etmede kullanılır ki bu vektör de önceden söylendiği gibi seçmeli alt-görevlerden hangilerinin çalıştırılacağını belirtir. Örneğin $k = 3$ ise $l^{(3)} = [1, 1, 1, 0, ..., 0]^T$ olacak ve en yüksek p_1, p_2 and p_3 önceliklerine sahip 3 adet seçmeli alt-görev bir sonraki kontrol döneminde çalıştırılacaktır. Görev seviye kontrolörünün fonksiyonu aşağıda verilmiştir.

$$l^{(k)} = \begin{bmatrix} \left\{ \begin{array}{ll} 1 & \text{eğer } -\frac{1}{2q} \le u < \frac{1}{2q} \\ 0 & \text{değilse} \end{array} \right\} \\ \left\{ \begin{array}{ll} 1 & \text{eğer } -\frac{1}{2q} \le u < \frac{3}{2q} \\ 0 & \text{değilse} \end{array} \right\} \\ \vdots \\ \left\{ \begin{array}{ll} 1 & \text{eğer } -\frac{1}{2q} \le u < \frac{1+2q}{2q} \\ 0 & \text{değilse} \end{array} \right\} \end{bmatrix} \quad (9)$$

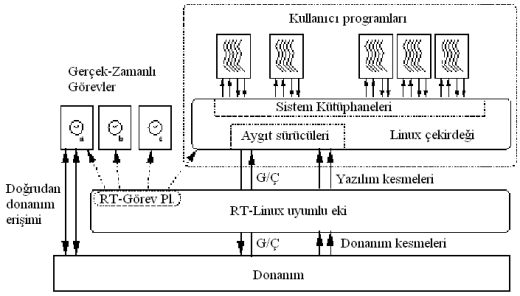
Bir başka deyişle çalıştırılacak seçmeli alt-görevlerin sayısı ise

$$k = \begin{cases} 0 & \text{eğer } -\frac{1}{2q} \le u < \frac{1}{2q} \\ 1 & \text{eğer } \frac{1}{2q} \le u < \frac{3}{2q} \\ \vdots & \\ q & \text{eğer } \frac{2q-1}{2q} \le u < \frac{2q+1}{2q} \end{cases} \quad (10)$$

3. GK-RM Tekniğinin RT-Linux Sisteminde Uygulanması

Bu çalışmada önerilen görev planlaması tekniğini uygulayabilmek için RT-Linux işletim sistemi test ortamı olarak seçilmiştir. Bu bölümde RT-Linux ile ilgili çok kısa bilgi verilerek, uygulamanın detayları anlatılacaktır.

Gerçek-zamanlı sistemler açısından standart Linux'ta yaşanan problem, seçilemez kılınan kesmelerin yaratacağı tahminlenemez gecikmelerdir. Bu roblem standart Linux çekirdeği ile donanım kesme kontrolörünün arasına yerleştirilen bir öykünüm programı ile aşılmıştır. Linux kaynak kodunun içerisinde yer alan tüm *cli*, *sti* ve *iret* komutları *S_CLI*, *S_STI* ve *S_IRET* makroları ile değiştirilmiştir. Bu durumda oluşan tüm kesmeler ilk olarak öykünüm programı tarafında yakalanır. Standart Linux kaynak kodunda bir kesme seçilemez kılındıysa, bunun yerine öykünme programındaki bir değişken sıfırlanır. Bir kesme oluştuğunda program önce bu değişkeni kontrol eder. Eğer bu değişken bir değere sahipse, Linux bu kesmeyi seçilir kılmıştır demektir ve bunun sonucunda Linux kesme işleyicisi çalıştırılır. Diğer durumda kesme işleyicisi çalıştırılmaz onun yerine tüm askıda bekleyen kesmelerin bilgisinin tutulduğu bir değişkendeki ilgili bit belirlenir. Linux'un kesme kontrolörü üzerinde doğrudan bir kontrolü olmadığından, gerçek-zamanlı kesmelerin işlenmesi standart Linux çekirdeğinin çalışmasından etkilenmez. Şekil 4 RT-Linux yapısını göstermektedir. RT-Linux'ta gerçek-zamanlı görevler çekirdek modülü olarak ve yüksek performans amacıyla çekirdeğin adres boşluğunda tanımlanır. Bunun en büyük dezavantajıysa görevlerdeki en küçük bir yanlışın sistemin kilitlenmesine sebep olabilmesidir. RT-Linux peryodik ve kesmelere bağlı çalıştırılabilen görevleri destekler ve çekirdek modülü olarak yüklenebilen küçük bir görev planlayıcısına sahiptir. Bu sebeple de görev planlayıcısının yeni bir versiyonu ile değiştirilmesi oldukça kolaydır. RT-Linux işletim sistemi bu çalışmanın kapsamı dışında olduğundan daha fazla bilgi verilmeyecektir.



Şekil 4: RT-Linux işletim sisteminin yapısı

GK-RM planlayıcısının ugulanmasında ilk ihtiyaç duyulan CPU kullanım oranının ölçümü için aşağıdaki kod parçası `rtl_schedule()` fonksiyonunun içerisine eklenmiştir:

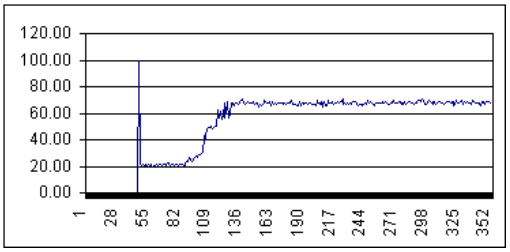
```
now = sched -> clock -> gethrtime(
sched->clock ); /*fonksiyonun
başında system saati (nano saniye
cinsinden) alınır.*/
/*Görev planlama kararı burada
verilir*/
sched -> rtl_current -> tinsec
+= ( now-oldnow ); /*tinsec,
görev yapısının içerisinde
tanımlanmıştır*/
now2 = sched -> clock ->
gethrtime( sched->clock );
scheduler_overhead += ( now2-now
); /*Bu satır planlayıcının ek
yükünün hesaplanması içindir*/
oldnow = sched -> clock ->
gethrtime(sched -> clock);
/*Görev değişimi
(‘‘task-switching’’) burada
yapılır*/
```

Her kontrol döneminde (bu çalışmada $100ms$ alınmıştır) bir kere çalışmak üzere yazılan “cpu_util” isimli peryodik iş parçacığı aşağıdaki kodu çalıştırarak CPU kullanım oranını hesaplar:

```
total_rt=linuxtime=0;
for(t=sched->rtl_tasks,i=0;
  t;
  t=t->next,i++){
 if(t==&sched->rtl_linux_task)
  linuxtime+=(float)t->tinsec;
 else total_rt+=(float)t->tinsec;

 t->tinsec = 0;
}
CPU_utilization = total_rt /
(total_rt + linuxtime);
```

PID ve görev seviye kontrolörleri için de $100ms$ peryoduna sahip birer iş parçacığı yazılmış olup, bunlara ait daha önce verilen sözde-kodlar yeterli açıklamaya sahip olduğundan burada daha fazla bir açıklamaya gerek olmadığı düşünülmektedir.



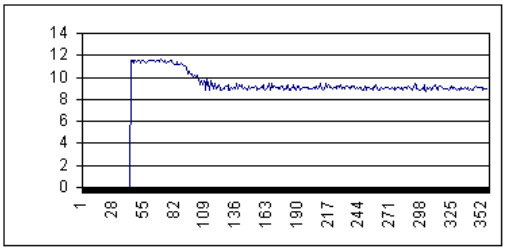
Şekil 5: CPU kullanım oranı (ayar noktası=%70)

4. Deneyler ve Sonuçlar

Sistem, peryodları tablo 1’de belirtilen 20 adet bağımsız peryodik görev kullanılarak test edilmiştir. Görevler, çalışma-zamanında birörnek dağılımdan üretilen çalışma sürelerine sahip olup, bu süre boyunca boş döngülerle işlemciyi oyalamaktadır. Şekil 5 bu denemeden elde edilen CPU kullanım oranını göstermektedir. Görüldüğü gibi, $t = 42$ anında 20 peryodik görev bir anda sisteme yüklenmiş, yaklaşık $t = 105$ anında ise sistem kararlı hale ulaşarak %70 CPU kullanım oranını yakalamıştır. Tüm görevlerin bir anda yüklenmesindeki amaç, kontrol teorisinde sıkça uygulanan ve karşılaşılabilecek en kötü durumu ifade eden rampa girişe karşı sistemin tepkisinin ölçülmesidir. Şekil 6 ise görev planlayıcısının sisteme getirdiği ek yükü göstermektedir. “cpu_util” ve PID kontrolör iş parçacıklarının ek yükleri de sırasıyla %0.008 ve %0.019 olarak ölçülmüş olup, bunların görev planlayıcısının ek yükü yanında ihmal edilebilir olduğu görülmüştür.

5. Gelecek Çalışmalar

Bu çalışmada, tamamlanmamış hesaplamalar yöntemine dayalı GK-RM görev planlama tekniğinin RT-Linux işletim sisteminde uygulanması sunulmuş ve deneme sonuçları bu yaklaşımın CPU kullanım oranını istenilen noktada kararlı halde başarıyla tutabildiğini göstermiştir. Bunun yanında sistem kararlığının teorik çalışmalarla ve daha gerçekçi iş yükleri altında incelenmesi gereklidir. Önerilen bir başka çalışma konusu da



Şekil 6: Görev planlayıcının, PID hesaplaması ve CPU kullanım oranı ölçümü de dahil olmak üzere sisteme getirdiği ek yük

yine daha gerçekçi iş yükleri altında GK-RM görev planlayıcısının performansının geleneksel görev planlama algoritmalarıyla karşılaştırmalarının yapılmasıdır.

6. Türkçe-İngilizce Sözlük

Ayar noktası:	Setpoint
Birörnek	Uniform
CPU kullanım oranı:	CPU Utilization
Görev planlayıcısı:	Scheduler
İş yükü:	Workload
Nicelendirici:	Quantizer
Son zaman:	Deadline
Tamamlanmamış hesaplamalar:	Imprecise Computations
Öykünüm:	Emulation

Kaynakça

[1] **J.W.S. Liu, K.J. Lin, W.K. Shih, A.C.S. Yu, J.Y. Chung, W. Zhao**, "Algorithms for scheduling imprecise computations", IEEE Computer, May (1991) 58 ± 68. (5) (1987).

[2] **Jane W.S. Liu, Wei-Kuan Shih, Kwei-Jay Lin, Riccardo Bettati and Jen-Yao Chung**, "Imprecise Computations", Proceedings of the IEEE, Vol. 82, No.1, January 1994.

[3] **Chenyang Lu, John A. Stankovic, Gang Tao, Sang H. Son**, "Design and Evaluation of a Feedback Control EDF Scheduling Algorithm". 20^{th} IEEE Real-Time Systems Symposium, 1999, pg. 56-67.

[4] **Chenyang Lu, John A. Stankovic, Gang Tao, Sang H. Son**, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms", Real-Time Systems Journal Special Issue on Control Theoretical Approach to Real-Time Computing 23(1/2):85-126 September, 2002.

[5] **Chenyang Lu, John A. Stankovic, Tarek F. Abdelzaher**, "Performance Specifications and Metrics for Adaptive Real-Time Systems", IEEE Real-Time System Symposium (RTS2000), December 2000.

[6] **D.A.Lawrence, J.Guan, S.Mehta, L.R. Welch**, "Adaptive Scheduling via Feedback Control for Dynamic Real-Time Systems", 20^{th} International Performance, Computing and Communications Conference, April 2001.

[7] **G. Beccari, et. al.**, "Rate Modulation of Soft Real-Time Tasks in Autonomous Robot Control Systems", Euro Micro Conference on Real-Time Systems, June 1999.

[8] **Gene F. Franklin, J. David Powell**, Abbas Emami-Naeini, 2002, Feedback Control of Dynamic Systems, Prentice Hall.

[9] **Liu, C.L., Layland, J.W.**, 1973. "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment". JACM. 20(1): 40-61.

[10] **Tolga Ayav, Sinan Yilmaz**, "Neuro-Fuzzy Controller in Real-Time Feedback Schedulers", In Proceedings of the 10^{th} Workshop on Nonlinear Dynamics of Electronic Systems, pp. 3.37-3.40, Izmir, Turkey, 2002.

[11] **J. Eker, P. Hagander, K. Arzen**, "A Feedback Scheduler for Real-Time Controller Tasks", In IFAC Control Engineering Practice, 2000.

[12] **John A. Stankovic, Chenyang Lu, Sang H. Son and Gang Tao**, "The Case for Feedback Control Real-Time Scheduling", In Proceedings of the 11^{th} Euromicro Conference on Real-Time Systems, pp. 11-20, York, UK, 1999.

[13] **Giorgio Buttazo and Luca Abeni**, "Adaptive rate control through elastic scheduling", Proceedings of the 39^{th} IEEE Conference on Decision and Control, Sydney, Australia, Dec. 2000.