SAKARYA
ÜNİVERSİTESİ

SAKARYA ÜNİVERSİTESİ

BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

İŞLETİM SİSTEMLERİ ÖDEV RAPORU

G211210577 / 1.B / Meys Elim İsaelhatib

B221210572 / 1.C / Loubana Almlhem

G231210557 / 2.C / Rania kazziha

G221210564 / 2.A / Dua Halluf Edib

G231210572 / 1.C / Nuseybe elabdullah

1. Giriş

Gerçek zamanlı işletim sistemleri (RTOS), gömülü sistemlerde görevlerin zamanında ve deterministik şekilde çalışmasını sağlamak için kullanılan özel sistemlerdir. FreeRTOS, açık kaynaklı, hafif ve taşınabilir bir RTOS olarak akademik dünyada ve endüstride yaygın olarak kullanılmaktadır.

Bu projede amaç, FreeRTOS'un temel prensiplerini kullanarak **4 seviyeli bir görev zamanlayıcı (scheduler)** geliştirmek ve farklı öncelik seviyelerine sahip görevlerin nasıl çalıştırıldığını simüle etmektir. Tasarlanan sistem gerçek zamanlı görevleri, kullanıcı görevlerini, geri beslemeli kuyruk yapısını ve preemption (öncelikli kesme) mantığını içermektedir.

2. Projenin Amacı ve Yöntemi Amaç

Bu projenin temel amacı, bir RTOS'un görev zamanlama mekanizmasını anlamak ve FreeRTOS'un scheduler yapısına benzer bir simülasyon geliştirmektir. Özellikle hedeflenen kazanımlar:

- Görev önceliklendirme mantığını anlamak
- Gerçek zamanlı görevlerin kritik rolünü kavramak
- Çok seviyeli geri beslemeli (MLFQ) zamanlayıcı tasarlamak
- Kuyruk, görev durumu (READY, RUNNING, SUSPENDED, FINISHED) ve kesme (preemption) kavramlarını uygulamak
- Simülasyon çıktısını analiz ederek davranışı incelemek

Yöntem

Projede üç temel bileşen kullanılmıştır:

1. **Görev (Task) yapısı :**
   Varış zamanı , Öncelik seviyesi (0 = RT)
   Çalışma süresi , Kalan süre ve kuantum
2. **Kuyruklar**:
   1 adet gerçek zamanlı kuyruk ,
   3 adet çok seviyeli geri beslemeli kuyruk (öncelikler1,2,3)
3. **Zamanlayıcı (Scheduler) :**
   RT görevler her zaman en yüksek önceliklidir
   Kullanıcı görevleri kuantum dolduğunda daha düşük seviyeye düşürülür
   RT görevi geldiğinde çalışan kullanıcı görevi kesilir (preemption)
   Tüm görevler bitene kadar sistem zaman ilerletilir

3. Uygulama ve Sistem Tasarımı

3.1. Görev Yapısı

Her görev şu bileşenlerden oluşmaktadır:

- ID , Varış zamanı
- Temel öncelik
- Kalan çalışma süresi , Bulunduğu kuyruk
- Durum (READY, RUNNING, SUSPENDED, FINISHED)

Bu yapı sayesinde görevlerin yaşam döngüsü simülasyon boyunca izlenmiştir.

3.2. Kuyruk Kullanımı

Projede iki tür kuyruk bulunmaktadır:

Gerçek zamanlı kuyruk (RTQ)

- Öncelik = 0 olan görevler buraya gelir.
- Kesilmezler, kuantuma tabi değildirler.
- Her zaman en yüksek önceliğe sahiptirler.

Kullanıcı kuyrukları (MLFQ)

- 3 seviye vardır: 1, 2 ve 3
- Görevler kuantum bitince alt seviyeye düşürülür
- Düşük seviyeler daha az CPU hakkına sahiptir
- Bu yapı adil CPU paylaşımı sağlar

3.3. Zamanlayıcı (Scheduler) Mantığı

Zamanlayıcı her saniyede şu adımları gerçekleştirir:

1. Yeni gelen görevleri ilgili kuyruğa alır
2. Çalışan kullanıcı görevi varken RT görevi gelirse kullanıcı görevi askıya alınır (preemption)
3. Çalışacak en yüksek öncelikli görevi seçer
4. Görevi 1 saniye çalıştırır
5. Quantum dolarsa görev alt seviyeye düşürülür
6. Görev biterse FINISHED durumuna geçer

3.4. Simülasyon Çıktısı (Özet)

Simülasyon sonunda elde edilen bilgiler:

- Toplam görev sayısı
- Bitiren görev sayısı
- Simülasyonun toplam sürdüğü süre (son zaman birimi)
- RT görevlerinin baskınlığı ve preemption olayları
- Hangi görevlerin kaçıncı saniyede başladığı/bittiği Simülasyon sonunda ekrana şu özet

basılmıştır:

```
----------------------------------------
SIMULASYON OZETI:
Tamamlanan gorev sayisi : 25
Toplam gecen sure       : 70 saniye
----------------------------------------
```

4. Modüllerin Açıklanması

4.1. scheduler.c

- Görevlerin yönetildiği ana modüldür.
- Kuyruklar burada oluşturulmuş ve yönetilmiştir.
- run_one_tick() fonksiyonu her zaman birimini simüle eder.
- Preemption, quantum takibi, görev bitirme burada yapılır. 4.2. tasks.c
- Loglama fonksiyonları bulunur.
- Görev başlama, bekleme, askıya alma ve bitirme olayları burada yazdırılır.

4.3. main.c

- Program başlangıç noktasıdır.
- giriş dosyasını okur ve scheduler'ı çalıştırır.

5. Bellek ve Kaynak Yönetimi

FreeRTOS benzeri yapıda her görev:

- Bir kuyrukta saklanır
- Kalan çalışma süresi bulunduğu değişkende tutulur
- Ek bir bellek tahsisi yapılmaz (statik yapı kullanılmıştır) Bu nedenle:
- Parçalanma (fragmentation) oluşmaz
- Bellek yönetimi deterministik kalır
- Sistem RTOS mantığına uygun çalışır

Bu tasarım gerçek bir gömülü sistemde düşük flash/RAM tüketimi sağlar.

6. Sonuç

Bu projede FreeRTOS'un görev zamanlama prensipleri temel alınarak çok seviyeli bir scheduler tasarlanmış ve simülasyonu gerçekleştirilmiştir.

Elde edilen sonuçlar:

- RT görevler sistem üzerinde belirgin şekilde baskın davranış gösterir
- Kullanıcı görevleri quantum doldukça alt seviyelere iner
- Preemption mekanizması CPU paylaşımını etkili şekilde düzenler
- Sistem deterministiktir; her zaman aynı girdiye aynı çıktıyı üretir

Bu çalışma, gerçek zamanlı işletim sistemlerinin temel prensiplerini pekiştirmek için etkili bir uygulama olmuştur.

7. Kaynaklar

1. FreeRTOS Official Documentation — https://freertos.org
2. ARM Cortex-M Programming Guide
3. Silberschatz — Operating System Concepts
4. Andrew S. Tanenbaum – Modern Operating Systems

8. Github linki:

https://github.com/MaisAlreemKhatib/FreeRTOS_PC_Scheduler