SAKARYA
ÜNİVERSİTESİ

2025 – 2026 Güz Dönemi

İŞLETİM SİSTEMLERİ

Proje Ödevi

Hazırlayanlar

Hasan Basri Açar / B221210104

Emir Çağlar Demirci / B201210003

Mahmut Karaman / B221210048

Numan Ak / B221210105

GİRİŞ

FreeRTOS (Free Real-Time Operating System), gömülü sistemler için geliştirilmiş açık kaynaklı, hafif ve taşınabilir bir gerçek zamanlı işletim sistemi çekirdeğidir. Gerçek zamanlı işletim sistemleri (RTOS), görevlerin belirlenen zaman sınırları içinde yürütülmesini garanti eden sistemlerdir. Bu sistemlerde görevler, önceliklerine göre sıralanır ve zamanlama (scheduling) algoritmaları ile yönetilir.

FreeRTOS, preemptive (kesmeli) çok görevliliği destekler ve statik/dinamik bellek ayırma seçenekleri sunar. Küçük ayak izi, düşük gecikme süreleri ve geniş donanım desteği ile gömülü sistemlerde yaygın olarak kullanılır. Bu projede, FreeRTOS'un görev zamanlama mantığının anlaşılması için PC ortamında bir scheduler simülasyonu geliştirilmiştir.

AMAÇ

Projenin amacı, FreeRTOS'un görev sıralayıcısının (scheduler) çalışma prensiplerini anlamak, dört seviyeli öncelikli bir görev zamanlayıcıyı simüle etmek ve görev öncelikleri, bağlam değişimleri ve zaman dilimleri arasındaki etkileşimi gözlemlemektir.

Proje Yöntemi: POSIX (Linux/Windows) ortamında FreeRTOS'un POSIX portu kullanılarak, C dili ile scheduler simülasyonu geliştirilmiştir. Görevler giris.txt dosyasından okunmuş, kuyruk yapıları ile yönetilmiş ve renkli terminal çıktıları ile görselleştirilmiştir.

Scheduler Yapısının Tasarlanması:

1. Gerçek Zamanlı (RT) Görevler (Öncelik 0): FCFS (First-Come First-Served) algoritması ile kesintisiz çalıştırılır. Yüksek önceliğe sahiptir ve tamamlanana kadar askıya alınmaz.

2. Kullanıcı Görevleri (Öncelik 1-3): Üç seviyeli geri beslemeli (Multi-Level Feedback Queue - MLFQ) bir yapıda çalıştırılır:

- Seviye 1: Yüksek öncelikli kullanıcı görevleri
- Seviye 2: Orta öncelikli kullanıcı görevleri
- Seviye 3: Düşük öncelikli kullanıcı görevleri
- Her görev 1 saniyelik zaman dilimi (quantum) alır, bitmezse bir alt seviyeye düşürülür.
- En düşük seviye (3) Round-Robin algoritması ile çalışır.

YÖNTEM

FreeRTOS Görev Yönetimi:

FreeRTOS'ta her görev bir fonksiyon olarak tanımlanır ve xTaskCreate() ile oluşturulur. Görevler öncelik değerlerine göre hazır kuyruğuna alınır. Scheduler, en yüksek öncelikli hazır görevi çalıştırır. Preemptive özelliği sayesinde, daha yüksek öncelikli bir görev hazır olduğunda, mevcut görev askıya alınır.

Geliştirilen Scheduler Fonksiyonları:

1. **load_task_list():** giriş.txt dosyasını okuyarak görevleri diziye yükler.
2. **run_scheduler():** Ana zamanlama algoritmasını yürütür:

- Gerçek zamanlı görev kuyruğu (FCFS)
- Kullanıcı görev kuyrukları (3 seviyeli MLFQ)
- Zaman ilerlermesi ve durum geçişlerini yönetir.

3. **Kuyruk Yapıları:** IntQueue yapısı ile basit dairesel kuyruk implementasyonu.
4. **Görev Yaşam Döngüsü:** WAITING => READY => RUNNING => FINISHED durum geçişleri
5. **Zaman Sınırlaması:** Her görev maksimum 20 saniye CPU kullanabilir.

UYGULAMA

Kod açıklamaları

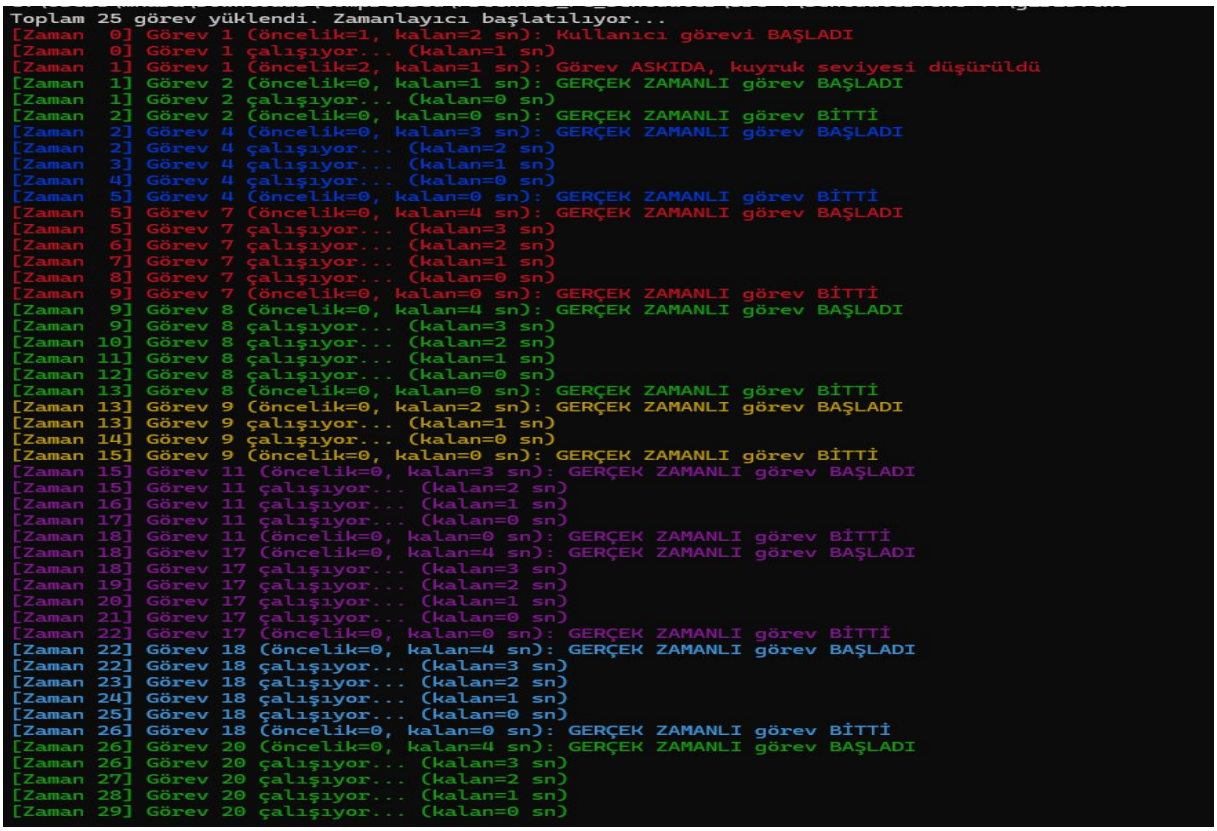
- main.c: Program giriş noktası, argüman kontrolü ve scheduler başlatma.
- scheduler.h/.c: Scheduler yapıları, kuyruk işlemleri ve zamanlama algoritması.
- tasks.c: Görev renklendirme, olay mesajları ve çıktı fonksiyonları.

Çalışma Çıktısı

Program, giriş.txt dosyasındaki 25 görevi işler. Her görev için;

- Başlangıç, çalışma, askıya alma, devam etme ve bitiş mesajları
- Renkli çıktılar
- Zaman bilgisi ve kalan süre gösterimi

Çıktı:



SONUÇ

1. **Bellek ve Kaynak Yönetim Yapıları**

   Kuyruk Yapıları:

   - IntQueue: Basit dairesel kuyruk implementasyonu. Görev indekslerini tutar, O(1) ekleme/çıkarma sağlar.
   - Bellek Tahsisi: Tüm görevler ve kuyruklar statik dizilerde saklanır. Bu yaklaşım:
     - Bellek sızıntısı riskini ortadan kaldırır.
     - Gerçek zamanlı sistemlerde öngörülebilir bellek kullanımı sağlar.
     - Ancak, maksimum görev sayısı ile sınırlıdır (MAX_TASKS = 128)

   Kaynak Ayırma:

   - Görev yapıları başlangıçta tahsis edilir, dinamik bellek ayırma kullanılmaz.
   - Bu, deterministik yürütme süresi sağlar. Gerçek zamanlı sistemler için kritiktir.

2. **Program Yapısı ve Modüller**

   Genel Yapı:

   1. Giriş Modülü: load_tasks_list() - Dosya okuma ve görev oluşturma.
   2. Zamanlayıcı Modülü: run_scheduler() – Ana zamanlama algoritması.
   3. Kuyruk Modülü: IntQueue işlemleri – Görev sıralama.
   4. Çıktı Modülü: print_task_event(), print_task_tick() – Görselleştirme.

   Arayüz Fonksiyonları:

   - get_task_color(): Görev kimliğine göre ANSI renk kodu döndürür.
   - print_task_event(): Görev olaylarını (başlama, bitme, askıya alma) formatlar.
   - print_task_tick(): Görevin her zaman dilimindeki çalışma durumunu gösterir.

   Gerekçelendirme:

   Modüler yapı, kodun bakımını ve anlaşılırlığını kolaylaştırır. Her modül belirli bir sorumluluğa sahiptir, bu da hata ayıklamayı ve geliştirmeyi basitleştirir.

3. **Çok Düzeyli Görevlendirme Şemasının Değerlendirilmesi**

   Çok düzeyli geri beslemeli kuyruk (MLFQ) şeması, kısa görevlerin hızlı tamamlanmasını sağlayarak yanıt süresini iyileştirirken, uzun görevlerin düşük öncelikli seviyelerde adil şekilde paylaşımını garanti eder. Geri besleme mekanizması, düşük öncelikli görevlerin sürekli ertelenmesini (aç kalmasını) engelleyerek sistem dengelenir.

   Gerçek İşletim Sistemleri ile Karşılaştırma:

   Unix/Linux benzer MLFQ kullanır, ancak daha gelişmiş öncelik hesaplama ve dinamik zaman dilimi ayarlama içerir. Windows ise öncelik temelli kesmeli zamanlama kullanır, ancak geri besleme daha az belirgindir. Bu projedeki implementasyon, gerçek

sistemlerde bulunan dinamik öncelik ayarlama ve I/O bağlamı gibi mekanizmaları içermeyen basitleştirilmiş bir MLFQ'dur.

Eksiklikler:

1. I/O bağlamı eksikliği (gerçek sistemlerde I/O bekleyen görevler öncelik kazanır).
2. Tüm seviyelerde sabit zaman dilimi (1 saniye) kullanılması.
3. Öncelik tersinmesi (priority inversion) korumasının olmaması.

Olası İyileştirmeler:

- Dinamik Öncelik Ayarlama: CPU-bound ve I/O-bound görevlere göre öncelik ayarlama.
- Değişken Zaman Dilimleri: Seviye arttıkça quantum süresini artırma.
- Kaynak Paylaşımı: Mutex ve semafor mekanizmaları eklenerek kaynak çakışması önlenebilir.
- Dinamik Bellek Yönetimi: Görev sayısı sınırını kaldırmak için dinamik bellek tahsisi.
- Bellek tahsisinde havuz (pool) veya SLAB ayırıcılar kullanılarak performans artırılabilir.

Genel Değerlendirme

Bu proje, FreeRTOS scheduler'ının temel prensiplerini başarıyla simüle etmektedir. Dört seviyeli öncelik mekanizması, gerçek zamanlı ve kullanıcı görevleri arasındaki etkileşimi net bir şekilde göstermektedir. Renkli çıktılar, görevlerin zaman içindeki davranışını görselleştirmede etkilidir.

Ancak, gerçek dünya uygulamalarında dikkate alınması gereken ek faktörler bulunmaktadır: kesmeler (interrupts), kaynak paylaşımı, öncelik tersinmesi (priority inversion) ve en kötü durum yürütme süresi (WCET) analizi. Bu simülasyon, gerçek zamanlı sistem tasarımının temellerini anlamak için değerli bir başlangıç noktası sunmaktadır.

KAYNAKLAR

1. **FreeRTOS Resmi Dokümantasyonu** – FreeRTOS API referansı ve kullanım kılavuzu. Erişim: https://www.freertos.org/documentation
2. **FreeRTOS GitHub Repository** – FreeRTOS çekirdek kaynak kodu ve POSIX portu. Erişim: https://github.com/FreeRTOS/FreeRTOS-Kernel
3. **POSIX Threads Programming** – Lawrence Livermore National Laboratory. Erişim: https://computing.llnl.gov/tutorials/pthreads/
4. **STM32 Mikrodenetleyici Ailesi Veri Sayfaları** – FreeRTOS'un sıklıkla uygulandığı ARM tabanlı mikrodenetleyicilerin teknik spesifikasyonları.