İŞLETİM SİSTEMLERİ PROJE RAPORU

FreeRTOS Kullanarak 4 Seviyeli Öncelikli Sıralayıcı (Scheduler) Simülasyonu

Ders	İşletim Sistemleri
Dönem	2025-26 Güz
Uygulama	FreeRTOS + POSIX (PC üzerinde)
Hazırlayan	YUNUS EMRE SEVİNDİK TALHA KAYA TUNAHAN AVŞAR ANIL GEZERTAŞAR MEHMET ZAHİD GÖRGEÇ
Tarih	18.12.2025

Not: Bu rapor, teslim edilen kaynak koddaki mevcut uygulamayı (main.c, scheduler.c/.h, FreeRTOSConfig.h, Makefile ve giris.txt) temel alır ve kaynak kod satırları doğrudan rapora eklenmemiştir.

1. Giriş

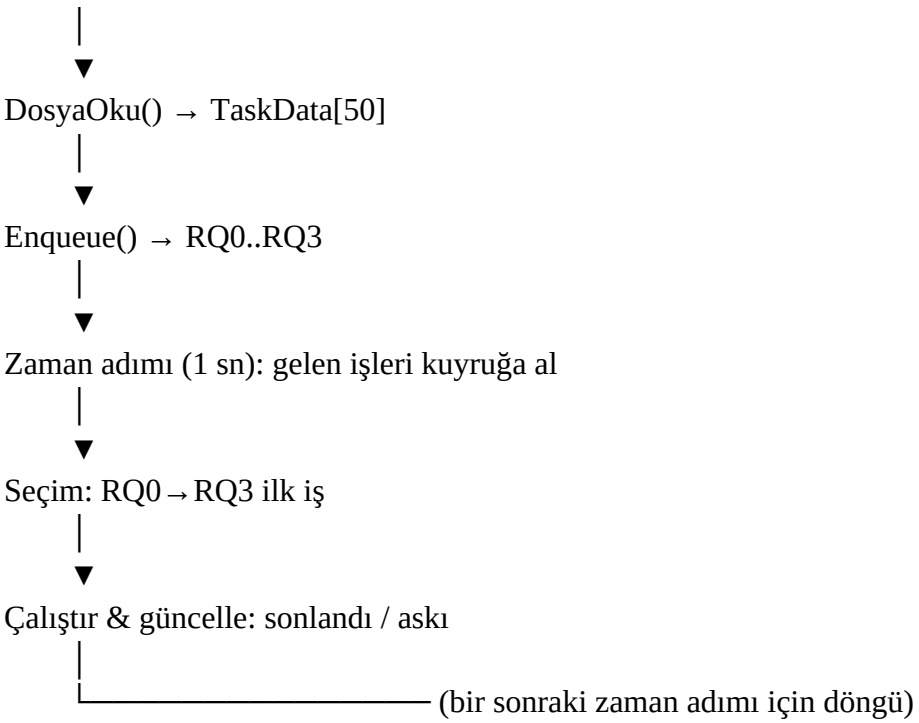
Gerçek zamanlı işletim sistemlerinde (RTOS), görevlerin (task) hangi sırayla ve ne kadar süre çalıştırılacağı zamanlayıcı (scheduler) tarafından belirlenir. Bu projede, FreeRTOS çekirdeği PC üzerinde POSIX portu ile çalıştırılarak görev zamanlama davranışını gözlemlemek amaçlanmıştır. Mevcut uygulama, FreeRTOS üzerinde yalnızca bir "simülasyon görevi" çalıştırır; sistemdeki işler ise gerçek FreeRTOS task'leri olarak yaratılmak yerine TaskData veri yapısı ile simüle edilir.

2. Amaç

Simülasyonun hedefi, giriş dosyasından okunan işlerin (varış zamanı, öncelik, CPU süresi) 4 seviyeli öncelikli kuyruk yapısı ile yürütülmesini modellemek ve özellikle: Öncelik 0 gerçek zamanlı işlerin (FCFS) kesintisiz çalışmasını, Öncelik 1-3 kullanıcı işlerinin MLFQ mantığıyla 1 saniyelik kuantum sonunda askıya alınıp alt seviyeye indirilmesini, Her zaman adımında (1 sn) çıktıda "başladı/yürütülüyor/askıda/sonlandı/zamanaşımı" durumlarının görünür olmasını sağlamaktır.

Öncelik	**Tanım**	**Adet (giris.txt)**
0	Gerçek zamanlı (FCFS, kesmesiz)	9
1	Kullanıcı - yüksek	4
2	Kullanıcı - orta	5
3	Kullanıcı - düşük (RR)	7

Şekil 1. Simülasyon akışının özeti (mevcut uygulama)



3. Yöntem ve Tasarım

Uygulama, FreeRTOS çekirdeğinin POSIX portu üzerinde çalışır. main() fonksiyonu giriş dosyasını okur ve SimulationTask adlı tek bir FreeRTOS görevi başlatır. Simülasyon her 1 saniyede bir “zaman adımı” ilerletir (vTaskDelay ile) ve globalTime değişkenini artırır.

3.1 Veri Yapıları

Her iş/görev, TaskData yapısı ile temsil edilir: arrivalTime: işe varış zamanı (sn) priority: kuyruk seviyesi (0..3) burstTime / remainingTime: toplam/kalan CPU süresi startTime, hasStarted: zamanaşımı ve çıktı mantığı için durum bilgisi. Hazır kuyruklar, 4 seviye için sabit boyutlu pointer dizileri olarak tutulur: RQ0..RQ3.

3.2 Kuyruklama ve Seçim Mantığı

Her zaman adımında arrivalTime==globalTime olan işler ilgili kuyruğa alınır (Enqueue). Seçim aşamasında, en yüksek öncelikli boş olmayan kuyruk başındaki iş “currentTask” olarak belirlenir. Öncelik 0 (gerçek zamanlı) iş varsa ve bitmemişse, kesintisiz çalıştırılmaya devam edilir. Öncelik 1-3 işleri için kuantum 1 saniyedir: iş 1 saniye çalıştırılır; bitmemişse “askıda” mesajı basılır ve mümkünse bir alt önceliğe indirilerek kuyruğun sonuna eklenir.

3.3 Bellek ve Kaynak Yönetimi

Mevcut uygulamada işlerin ve kuyrukların belleği statik (sabit boyutlu) diziler ile ayrılmıştır (TaskData[50] ve 4×50 pointer dizisi). Bu yaklaşım, dinamik bellek parçalanmasını önler ve basitlik sağlar. Öte yandan, kuyruktan çıkarma işlemi eleman kaydırdığı için O(n) maliyetlidir ve büyük iş sayılarında verim düşer. FreeRTOS tarafında ise gerçek task’ler için TCB/stack bellek yönetimi heap_4 ile yapılır; ancak bu projede yalnızca SimulationTask gerçek bir FreeRTOS görevidir.

4. Uygulama Detayları ve Çıktı

4.1 Modüller

Dosya	Sorumluluk (özet)
src/main.c	Giriş dosyasını okutma ve SimulationTask oluşturup scheduler’ı başlatma
src/scheduler.c/.h	İş listesi, 4 seviye kuyruklar, seçim/askı/sonlandırma ve çıktılar
include/FreeRTOSConfig.h	POSIX port üzerinde FreeRTOS yapılandırması (tick=1000 Hz, max prio=5)
Makefile	GCC ile derleme ve çalıştırma hedefleri

4.2 Derleme ve Çalıştırma

```
cd FreeRTOS_PC_Scheduler
make
./freertos_sim giris.txt
```

Not: Terminal çıktısında durumlar ANSI renk kodları ile ayırt edilir (başladı=mavi, yürütülüyor=yeşil, askıda=sarı, sonlandı=mor, zamanaşımı=kırmızı).

4.3 Örnek Çalışma Çıktısı (ilk zaman adımları)

```
Simulasyon Basliyor...
0 .0000 sn proses başladı     (id:0000  öncelik: 1  kalan süre: 2 sn)
1 .0000 sn proses askıda      (id:0000  öncelik: 2  kalan süre: 1 sn)
1 .0000 sn proses başladı     (id:0001  öncelik: 0  kalan süre: 1 sn)
2 .0000 sn proses sonlandı    (id:0001  öncelik: 0  kalan süre: 0 sn)
2 .0000 sn proses başladı     (id:0003  öncelik: 0  kalan süre: 3 sn)
3 .0000 sn proses yürütülüyor (id:0003  öncelik: 0  kalan süre: 2 sn)
4 .0000 sn proses yürütülüyor (id:0003  öncelik: 0  kalan süre: 1 sn)
5 .0000 sn proses sonlandı    (id:0003  öncelik: 0  kalan süre: 0 sn)
5 .0000 sn proses başladı     (id:0006  öncelik: 0  kalan süre: 4 sn)
6 .0000 sn proses yürütülüyor (id:0006  öncelik: 0  kalan süre: 3 sn)
7 .0000 sn proses yürütülüyor (id:0006  öncelik: 0  kalan süre: 2 sn)
```

Örnek çıktıda 0. saniyede öncelik-1 bir kullanıcı işi başlar; 1. saniyede kuantum dolduğu için askıya alınıp bir alt seviyeye (öncelik 2) indirilir. Aynı saniyede varan öncelik-0 gerçek zamanlı iş hemen çalışmaya başlar ve bittiğinde sıradaki gerçek zamanlı işler kesintisiz biçimde devam eder. Bu, “gerçek zamanlı işler her zaman önce” kuralının simülasyonda görünür hale gelmesini sağlar.

5. Sonuç ve Değerlendirme

Bu çalışma, FreeRTOS çekirdeğini PC üzerinde çalıştırarak 4 seviyeli öncelikli kuyruk yapısının temel davranışlarını göstermektedir. Öncelik-0 işlerin kesintisiz yürütülmesi deterministik (öngörülebilir) bir davranış sağlar; kullanıcı işleri ise MLFQ ile daha adil paylaşım amacıyla kuantum sonunda askıya alınıp alt seviyeye indirilir.

5.1 Mevcut Uygulamanın Kısıtları

İşler gerçek FreeRTOS task’leri olarak yaratılmadığı için gerçek bağlam değiştirme (context switch) ve task durumları (vTaskSuspend/vTaskResume gibi) yalnızca çıktı seviyesinde simüle edilir. Zamanaşımı hesabı, işin varış zamanına göre yapılır (arrivalTime). Daha gerçekçi bir yaklaşım, işin ilk çalıştığı anı esas alarak yaşam süresi ölçmektir. Kuyruk yapısı sabit diziler ve eleman kaydırma ile kurulduğundan ölçeklenebilirlik sınırlıdır.

5.2 İyileştirme Önerileri

Her iş için xTaskCreate ile gerçek task oluşturmak; scheduler görevini “dispatcher” olarak kullanıp task’leri suspend/resume ederek gerçek FreeRTOS davranışına yaklaşmak. FreeRTOS Queue/List API’lerini kullanarak kuyruk işlemlerini daha verimli ve güvenli hale getirmek. Çıktıda “devam edildi” durumunu ayrı mesaj ile göstermek ve ölçüm/istatistik (bekleme süresi, turnaround time) raporlamak.

Kaynaklar

[1] FreeRTOS Kernel Documentation:
https://www.freertos.org/Documentation/RTOS_book.html
[2] FreeRTOS POSIX Port (portable/ThirdParty/GCC/Posix):
https://github.com/FreeRTOS/FreeRTOS-Kernel