25.05.2025

2024/2025 BAHAR DÖNEMİ
BLM0332 İŞLETİM SİSTEMLERİ DERSİ
PROJE RAPORU

Apartman İnşası ile Process, Thread ve Senkronizasyon Modelleme

Aleks DULDA – 21360859025

Büşra UZUNLAR – 20360859098

Dr. Öğr. Üyesi Mustafa Özgür CİNGİZ

İçindekiler Tablosu

Giriş

1.1 Projenin Amacı

Bu projenin temel amacı, işletim sistemi kavramlarını gerçek hayattan bir senaryo üzerinden modelleyerek kavramsal bilgileri uygulamalı hâle getirmektir. Özellikle process (süreç), thread (iş parçacığı) ve senkronizasyon konularının anlaşılmasını sağlamak hedeflenmiştir. Apartman inşası gibi adım adım ilerleyen ve kaynak paylaşımına ihtiyaç duyan bir süreç, bu kavramları simüle etmek için ideal bir ortam sunmaktadır.

1.2 Neden Apartman Modeli?

Apartman inşaatı, katlar ve dairelerden oluşan hiyerarşik yapısıyla işletim sistemlerinde kullanılan çoklu süreç ve çoklu iş parçacığı yapısına benzerlik göstermektedir. Katlar, bağımsız ilerleyen ama üst üste inşa edilmesi gereken yapılar olarak process kavramını temsil eder.

- Her kat içindeki daireler ise aynı anda çalışabilen ancak bazı kaynakları paylaşan thread yapılarıyla örtüşür.
- Bu yapı sayesinde hem sıralı (kat bazlı) hem de eş zamanlı (daire bazlı) işlem yönetimi modellenebilmiştir.

1.3 İşletim Sistemiyle İlişkisi (Process/Thread Kavramı)

Modern işletim sistemlerinde process'ler bağımsız çalışan, kendi kaynaklarına sahip programlardır. Thread'ler ise bu process'ler içinde çalışan alt görevlerdir.

Bu projede:

- Her kat ayrı bir fork() ile oluşturulan process'tir.
- Her daire ise pthread_create() ile başlatılan thread'dir.
- Daireler bazı işlemleri aynı anda yapabilirken (örneğin sıva), bazıları ortak kaynak kullanımı nedeniyle sırayla yapılmalıdır (örneğin vinç veya iskele kullanımı). Bu da senkronizasyon gerektirir.
- sem_wait, sem_post, wait() ve pthread_join gibi sistem çağrıları

gibi yapıların yönetimi sağlanmıştır.

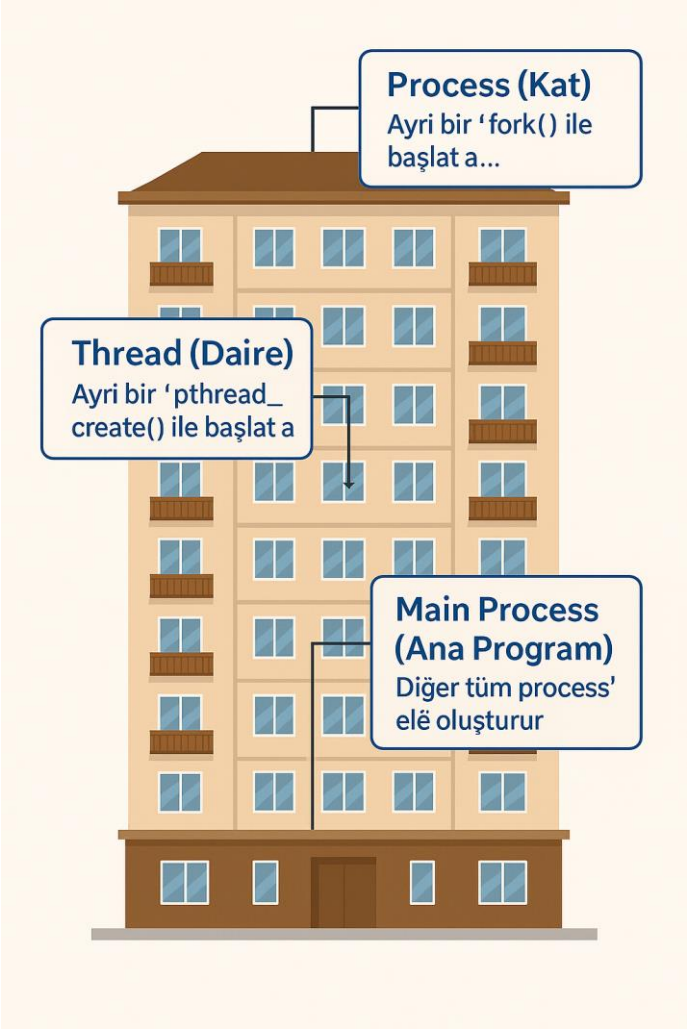
Sistem Mimarisi

Proje mimarisinde işletim sistemi kavramları gerçek dünya senaryoları ile eşleştirilmiştir. Her kat, kendi başına bağımsız çalışabilen birimler olarak process (süreç) olarak modellenmiştir. Bu process'ler fork() sistem çağrısı kullanılarak oluşturulmuştur.

Her process (yani kat), kendi içerisinde 4 adet daireye sahiptir. Bu daireler, kat içinde paralel olarak çalışabilen görevleri temsil eder ve thread (iş parçacığı) olarak modellenmiştir. Thread'ler pthread_create() fonksiyonu ile başlatılmıştır.

Bu yapı sayesinde:

- Katlar sırayla ve bağımsız olarak inşa edilirken,
- Aynı kat içerisindeki daire işlemleri paralel olarak ilerleyebilmektedir.



Şekil – 2.1

2.1 Süreçlerin Yapısı

Her kat (process) şu mantıkla çalışır:

1. Kat fork() ile oluşturulur.
2. Kat içerisinde 4 thread başlatılır.
3. Bu thread'ler sırasıyla inşaat adımlarını (tesisat, sıva, boya, vinç, iskele, temizlik) gerçekleştirir.
4. Kat ancak tüm daireleri tamamladığında exit() ile kapatılır.
5. Ana process (ana program), wait() ile bir kat bitmeden diğerini başlatmaz.

Bu yapı, işletim sistemlerinde process'ler arası sıralı çalışma ve thread'ler arası eşzamanlılık kavramlarını yansıtır.

2.2 Kullanılan Senkronizasyon Yapıları

İş parçacıkları bazı kaynakları birlikte kullanmak zorundadır. Bu nedenle senkronizasyon mekanizmaları kullanılarak yarış durumları (race condition) önlenmiştir.

Projede aşağıdaki semaphore kullanılmıştır:

- usta_sem: Aynı anda en fazla 3 dairenin tesisat ustası kullanmasını sınırlar.
- vinc_sem: Vinç sınırlıdır, sadece 2 daire aynı anda kullanabilir.
- iskele_sem: Dış cephe işlemlerinde kullanılan iskele de 2 daire ile sınırlıdır.

Her semafor için:

- Kaynağa erişim öncesi sem_wait()
- Kaynak serbest bırakıldığında sem_post() kullanılmıştır.

Ayrıca:

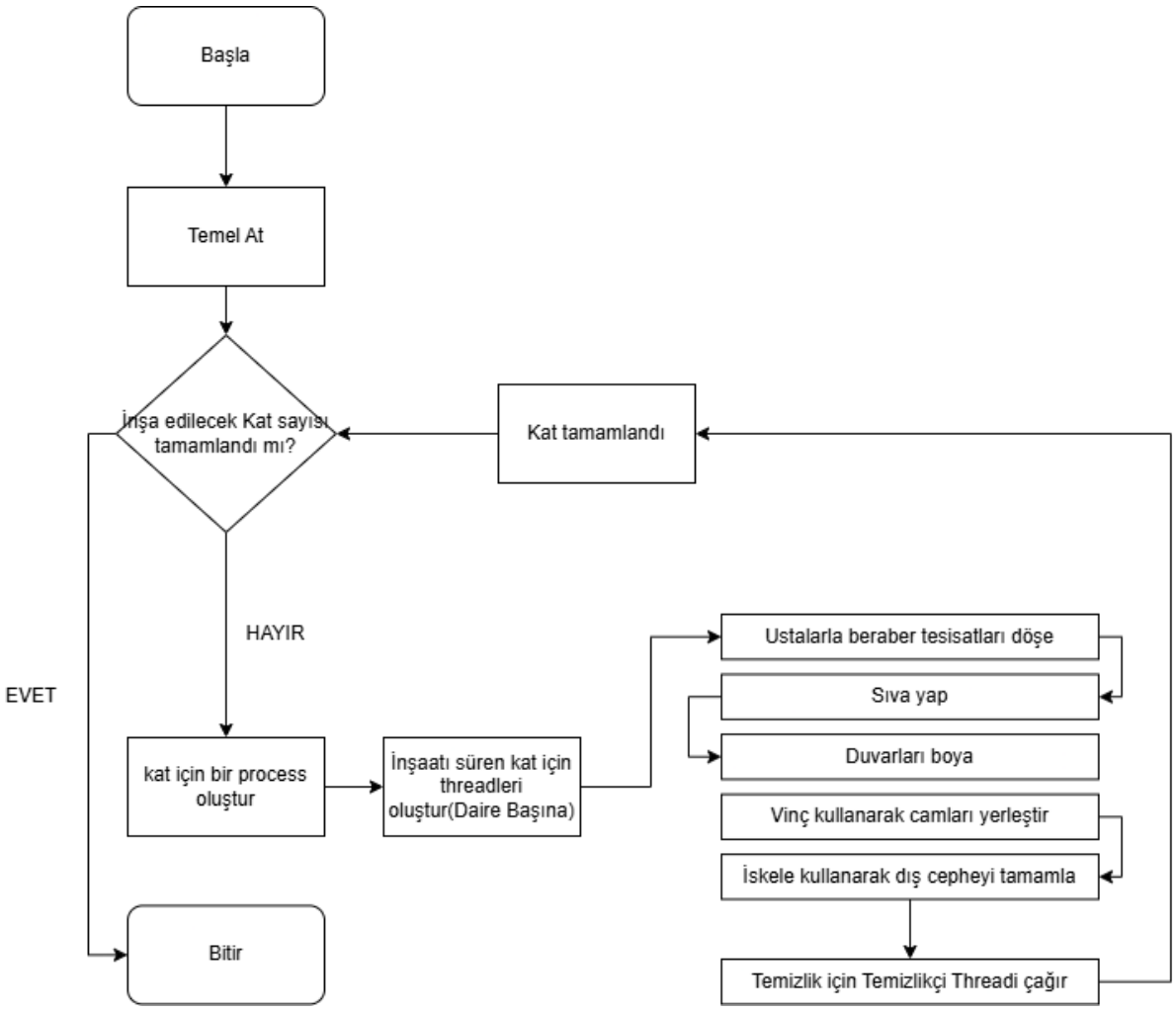
- Katlar arası sıralı inşaat için wait() fonksiyonu
- Temizlik işlemi gibi görevler için pthread_join() ile thread tamamlanması beklenmiştir.

Bu mekanizmalar, işletim sistemlerinde görülen kritik bölgelerin korunması, kaynak paylaşımı, ve eş zamanlılık yönetimi konularını yansıtmaktadır.

Akış Diyagramı (Flowchart)

Bu bölümde, apartman inşaatının süreçsel olarak nasıl modellendiği, hangi adımların hangi sırayla ve hangi koşullarla çalıştığı açıklanacaktır. Simülasyon, hem process (kat bazlı süreçler) hem de thread (daire bazlı işler) üzerinden yapılandırılmıştır. Katlar sırayla, daireler ise paralel şekilde inşa edilmektedir. Ayrıca bazı işlemler ortak kaynak kullanımına bağlı olduğu için senkronizasyon mekanizmaları da devreye girmektedir.

3.1 Apartman inşaatının genel akış diyagramı



Şekil – 3.1

3.2 Genel Akış

Programın genel yapısı aşağıdaki şekilde işler:

1. **Temel Atma:** İnşaata başlamadan önce program belirli bir bekleme süresi ile temel atma işlemini simüle eder.
2. **Kat Süreci (Process):** Her kat, fork() fonksiyonu ile ayrı bir process olarak başlatılır.
3. **Kat Sıralaması:** Bir kat tamamlanmadan bir sonraki kata geçilmez. Bu, wait(NULL) ile sağlanır.
4. **Daire Süreci (Thread):** Her kat içerisinde 4 daire vardır. Her daire bir pthread_create() çağrısı ile ayrı bir thread olarak başlatılır.
5. **Daire Görevleri:** Her daire sırasıyla aşağıdaki işleri yapar:
    - Tesisat
    - Sıva
    - Boya
    - Vinç kullanımı (Vinç kullanılırken %15 ihtimalle bozulma ihtimali vardır.)
    - İskele kullanımı
    - Temizlik (ayrı thread ile)
6. **Kaynak Paylaşımı:** Bazı görevlerde sınırlı kaynaklar (usta, vinç, iskele) ortak kullanılır. Bu durumlar semaforlarla kontrol altına alınır.
7. **Kat Tamamlanınca:** Tüm daireler tamamlandığında, ilgili process kendini sonlandırır. Bir üst katın süreci başlatılır.
8. **Bina Tamamlanınca:** Son kat da tamamlandığında program kaynakları temizleyerek sonlandırılır.

NOT: Temizlik işlemini ayrı bir thread olarak yapmamızın sebebi, bu adımın dairenin diğer işlemlerinden sonra geldiği ve bağımsız bir şekilde yürütülebilecek olmasıdır. Ne kadar böyle olmasını düşünsek de kod üzerinde tam işlevini gerçekleştiremedik.

Ana thread, inşaat işlemlerini bitirince temizlik için başka bir thread oluşturuyor. Bu sayede hem temizlik sırasında sistem kilitlenmiyor hem de temizlik bitene kadar beklenip işin gerçekten tamamlandığından emin olunuyor. Bununla birlikte, durum loglarda yansıdığında temizlik bitmeden diğer kata geçilmediğini gösteriyor. Aslında öyle de oluyor. Buradaki amaç, diğer katların inşası sürerken temizlikçilerin bir yandan temizlik işleminin yapılmasıydı fakat başarılı sonuç vermedi.

Sıralı ve Paralel Yapılar

4.1 Sıralı Yapı: Katlar Arası İşleyiş

Katlar arasında tam bir sıralılık söz konusudur. Program, her kat için fork() fonksiyonu ile ayrı bir process oluşturur. Ancak, bir kat tamamlanmadan bir sonraki katın sürecine geçilmesine izin verilmez. Bu kontrol wait(NULL) fonksiyonu ile sağlanır. Bu sayede üst üste inşaat yapılma mantığı gerçek hayattaki gibi korunur.

4.2 Paralel Yapı: Dairelerin Eşzamanlı İşleyişi

Her katın içinde yer alan 4 daire, pthread_create() fonksiyonu ile başlatılan bağımsız thread'ler olarak çalışır. Bu daireler kendi iç işlemlerini (tesisat, sıva, boya, vinç, iskele, temizlik) paralel şekilde yürütür.

4.3 Kaynak Paylaşımı ve Senkronizasyon

Bazı işlemler ortak kaynaklara ihtiyaç duyduğundan, burada senkronizasyon devreye girer. Örneğin:

- Tesisat işlemi: En fazla 3 daire aynı anda → usta_sem
- Vinç ve iskele: En fazla 2 daire → vinc_sem, iskele_sem

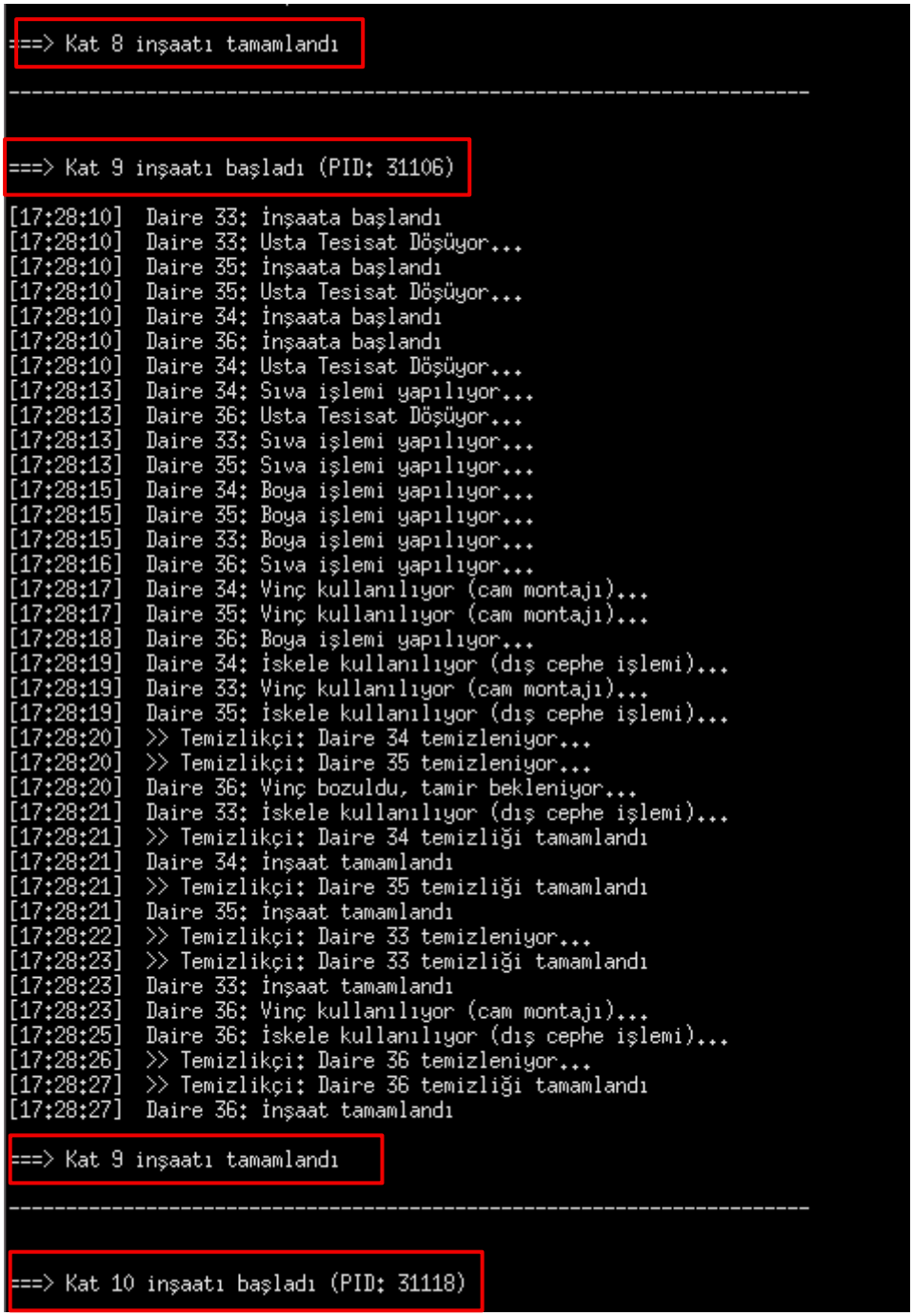
Bu kısıtlamalar sem_wait() ve sem_post() işlemleriyle kontrol edilir. Böylece yarış durumları (race condition) önlenmiş olur.

Aynı zamanda vinç kullanılacağı zaman %15 ihtimalle bozulma ihtimali eklenmiştir. Bu uygulama biraz aksama yaratıp paralleliği görmemize kolaylık sağlamaktadır.

Örnek Terminal Çıktıları

5.1 Katların Sıralı İnşası (Process Sırası)

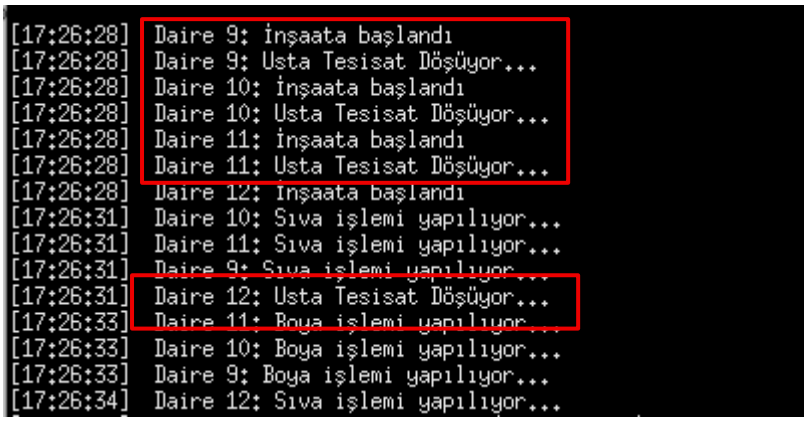
Şekil – 5.1 . Her katın inşaatı sırasıyla başlatılıp tamamlandığı görülmektedir.



Şekil – 5.1

5.2 Usta Semaforu – İlk 3 Daire Giriyor, 4. Bekliyor

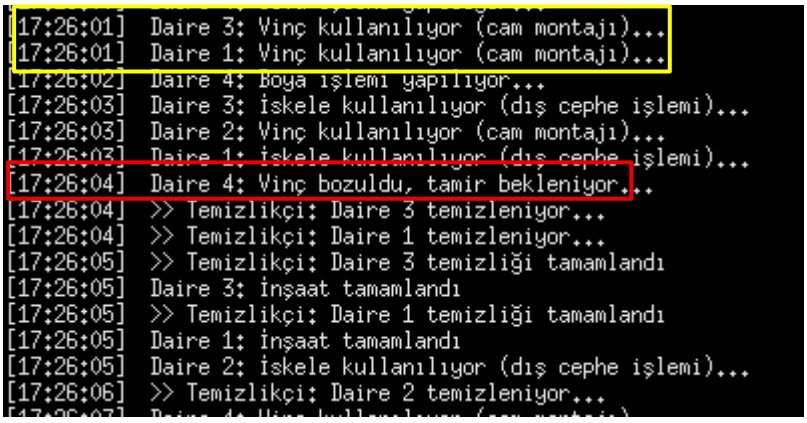
Şekil – 5.2. Yukarıdaki çıktıda ilk üç daire'nin aynı anda tesisat işlemine başlayabildiği; ancak Daire 4'ün bu işlemi gecikmeli gerçekleştirdiği görülmektedir.



Şekil – 5.2

5.3 Vinç Bozulması ve Vinç Sınırlaması

Şekil – 5.3. Vinç yalnızca iki daire tarafından aynı anda kullanılabilmekte ve bu durum semafor ile sınırlandırılmıştır; ayrıca, rastgele bir olasılık tanımlanarak vinç arızası senaryosu da başarıyla simüle edilmiştir.



Şekil – 5.3

Sonuç

Bu projede, işletim sistemlerinde sıkça kullanılan process, thread ve senkronizasyon kavramları apartman inşaatı senaryosu üzerinden başarıyla modellenmiştir. Katlar fork() ile process olarak, daireler ise pthread_create() ile thread olarak oluşturulmuş ve sıralı-paralel yürütme yapıları bir arada kullanılmıştır.

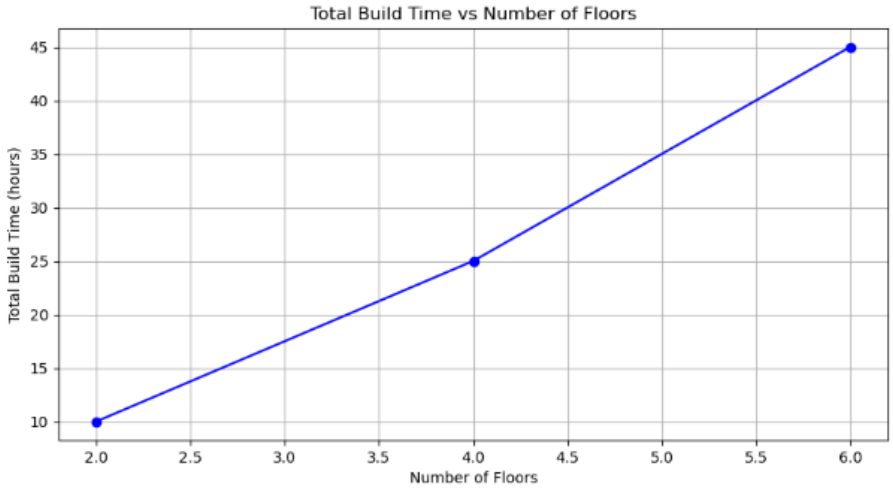
Sınırlı kaynaklara erişim semaforlarla kontrol edilmiş, vinç arızası gibi rastgele senaryolarla gerçekçilik sağlanmıştır. Temizlik işlemi ise ayrı bir thread olarak tanımlanarak çoklu iş parçacığı yapısı derinleştirilmiştir.

Genel olarak proje, işletim sistemi mantığının somut bir senaryo üzerinden uygulanarak pekiştirilmesini sağlamıştır.

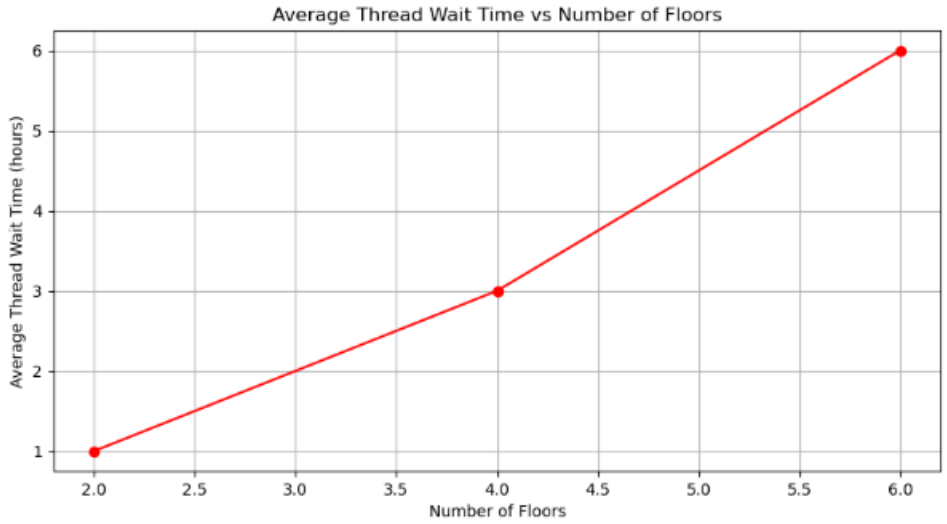
Performans ve Ölçeklenebilirlik Üzerine Değerlendirme

Proje, 2, 4 ve 6 katlı bina senaryoları ile test edilmiştir. Kat sayısı arttıkça toplam işlem süresi doğrusal olarak artarken, kaynak paylaşımı nedeniyle thread'lerin bekleme süresi de artış göstermiştir. Özellikle vinç ve iskele gibi sınırlı kaynaklar, yüksek katlı senaryolarda darboğaz oluşturmuştur. Bu durum, semaphore'ların etkili kullanımının sistem performansı üzerindeki kritik rolünü ortaya koymaktadır.

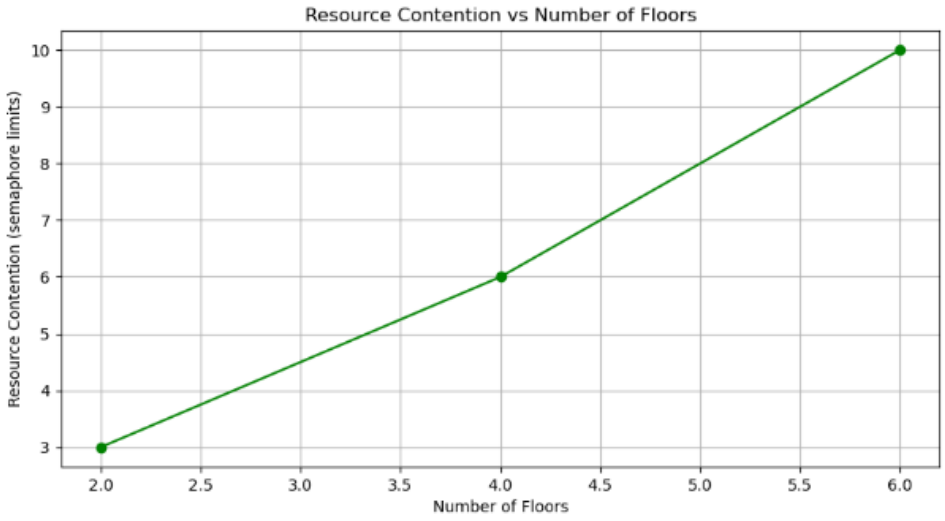
.



Şekil 7.1 – Toplam İnşaat Süresi ile Kat Sayısı



Şekil 7.2 – Ortalama Thread Bekleme Süresi ile Kat Sayısı



Şekil 7.3 – Kaynak Kullanımı (Semaphore) ile Kat Sayısı

Kaynakça

- POSIX Semaphores Documentation – Linux Manual Pages
  - https://man7.org/linux/man-pages/man7/sem_overview.7.html
- pthreads Programming (POSIX Threads) – Linux Manual Pages
  - https://man7.org/linux/man-pages/man7/pthreads.7.html
- GNU C Library – sem_init, sem_wait, sem_post – GNU Documentation
  - https://www.gnu.org/software/libc/manual/html_node/Semaphores.html
- C Programming Reference – cplusplus.com
  - https://cplusplus.com/reference
- İşletim Sistemleri Dersi Notları – Bursa Teknik Üniversitesi, 2024–2025 Bahar Dönemi
- ChatGPT (OpenAI, 2025) – Yalnızca açıklayıcı içerik ve yapısal öneriler için kullanılmıştır.
- COPILOT – Raporun içerik geliştirilmesinde kullanılmıştır.