İŞLETİM SİSTEMLERİ DÖNEM PROJESİ RAPORU

Proje Konusu: Çok Katlı Apartman İnşaatı Üzerinden Process, Thread ve Senkronizasyon Kavramlarının C Dilinde Modellenmesi
Hazırlayanlar: Erva Aygüneş, Sude Naz Doğdu
Teslim Tarihi: 26 Mayıs 2025

---

1. GİRİŞ

Modern işletim sistemleri, kullanıcı taleplerine hızlı ve etkili yanıt verebilmek amacıyla çoklu görev yapısını destekler. Bu yapı içinde **process (süreç)** ve **thread (iş parçacığı)** kavramları önemli yer tutar. Bu projede, bu kavramlar somutlaştırılarak 10 katlı ve her katta 4 daire bulunan bir apartmanın inşa süreci üzerinden modellendi.

Her kat bağımsız bir process olarak, her daire ise bir thread olarak tanımlandı. Bu yapı sayesinde işletim sistemlerinde sıkça karşılaşılan senkronizasyon, kaynak paylaşımı, süreç yönetimi ve yarış durumu (race condition) gibi konuların gerçek bir senaryoya uygulanması hedeflendi.

---

2. SİSTEM TASARIMI

2.1 Kat ve Daire Yapısı

- Katlar fork() sistem çağrısıyla ayrı bir process olarak başlatıldı.
- Her process içinde 4 adet pthread_create() çağrısıyla thread (daire) oluşturuldu.

2.2 Kaynak Yönetimi

- **Vinç:** Tüm dairelerin ortak kullanması gereken bu kaynak pthread_mutex_t ile korundu.
- **Elektrik ve Su Tesisatı:** Paylaşılan altyapı kaynakları olduğu için semaphore ile sınırlı erişim sağlandı. Elektrik için eşzamanlı 2, su için 1 erişim izni tanımlandı.

2.3 Temel Atma Mekanizması

- Temelin atılmadan üst yapıların başlamaması gerektiğinden temel_sem adlı bir semaforla başlangıç kontrolü sağlandı.

2.4 Kat Tamamlama Kontrolü

- Her daire işini bitirdiğinde bir sayaç arttırılır. Sayaç 4 olduğunda pthread_cond_signal() ile o katın tamamlandığı bilgisi ana iş parçacığına bildirilir.

---

3. TEKNİK GERÇEKLEŞTİRİM VE KOD YAPISI

3.1 Process Oluşturma

Katlar fork() kullanılarak bağımsız birimler olarak başlatılmıştır. Her child process sadece kendi dairelerini çalıştırır ve tamamlandığında ana process'e geri dönülür.

3.2 Thread Kullanımı

Her process içinde 4 thread tanımlanır. Her thread şu işleri sırayla yapar:

- Vinç kullanımı (mutex)
- Elektrik tesisatı (semaphore)
- Su tesisatı (semaphore)
- Bağımsız işlem (sıva, boya vb.)

3.3 Senkronizasyon Yapıları

- **Vinç** – pthread_mutex_t vinç_mutex
- **Elektrik** – sem_t elektrik_sem (init değeri: 2)
- **Su** – sem_t su_sem (init değeri: 1)
- **Temel kontrolü** – sem_t temel_sem
- **Kat tamamlama** – pthread_mutex_t, pthread_cond_t, tamamlanan_daire_sayisi

3.4 Zaman Damgası

time.h kullanılarak her işlem başlangıç ve bitişi zaman damgasıyla printf() üzerinden loglandı. Bu loglar, hata ayıklama ve işlem sırası takibi için kullanıldı.

---

4. TASARIM TERCİHLERİ ve GEREKÇELERİ

4.1 Semaphore ve Mutex Seçimi

- Mutex: Tek bir thread'in gireceği kritik bölgelerde (vinç gibi).
- Semaphore: Eşzamanlı fakat sınırlı erişim gereken bölgelerde (elektrik/su).

4.2 Neden fork() Kullanıldı?

- Her kat, tamamen bağımsız bir inşa süreci gibi düşünüldüğü için ayrı process olarak çalıştırıldı. thread pool yerine fork() tercih edilerek katlar arası bağımsızlık ve sıralama kolay sağlandı.

4.3 Elektrik/Su İşlemlerinin Engellenmesi

- Aynı anda fazla thread erişirse altyapı çakışmaları olabileceği için sınırlı erişim sağlandı. Elektrik için 2, su için 1 thread aynı anda çalışabilir hale getirildi.

4.4 Kat Bitim Koşulu

- Her daire tamamlandığında tamamlanan_daire_sayisi++. Bu sayaç 4 olduğunda pthread_cond_signal() ile ana thread bilgilendirilir.

---

5. KARŞILAŞILAN ZORLUKLAR VE ÇÖZÜMLER

5.1 Platform Uyumluluğu

Kod POSIX standardına uygun olduğu için hem Linux hem de macOS ortamlarında sorunsuz derlendi ve çalıştırıldı.

5.2 Race Condition Problemleri

İlk aşamada vinç ve altyapı sistemlerinde çakışmalar yaşandı. Mutex ve semaphore yapıları ile bunlar giderildi.

5.3 Thread'lerin Kontrolsüz Bitmesi

Thread'lerin iş bitiminden sonra kat sürecinin erken bitmemesi için pthread_join() ve pthread_cond_wait() yapıları birlikte kullanıldı.

---

6. SONUÇ

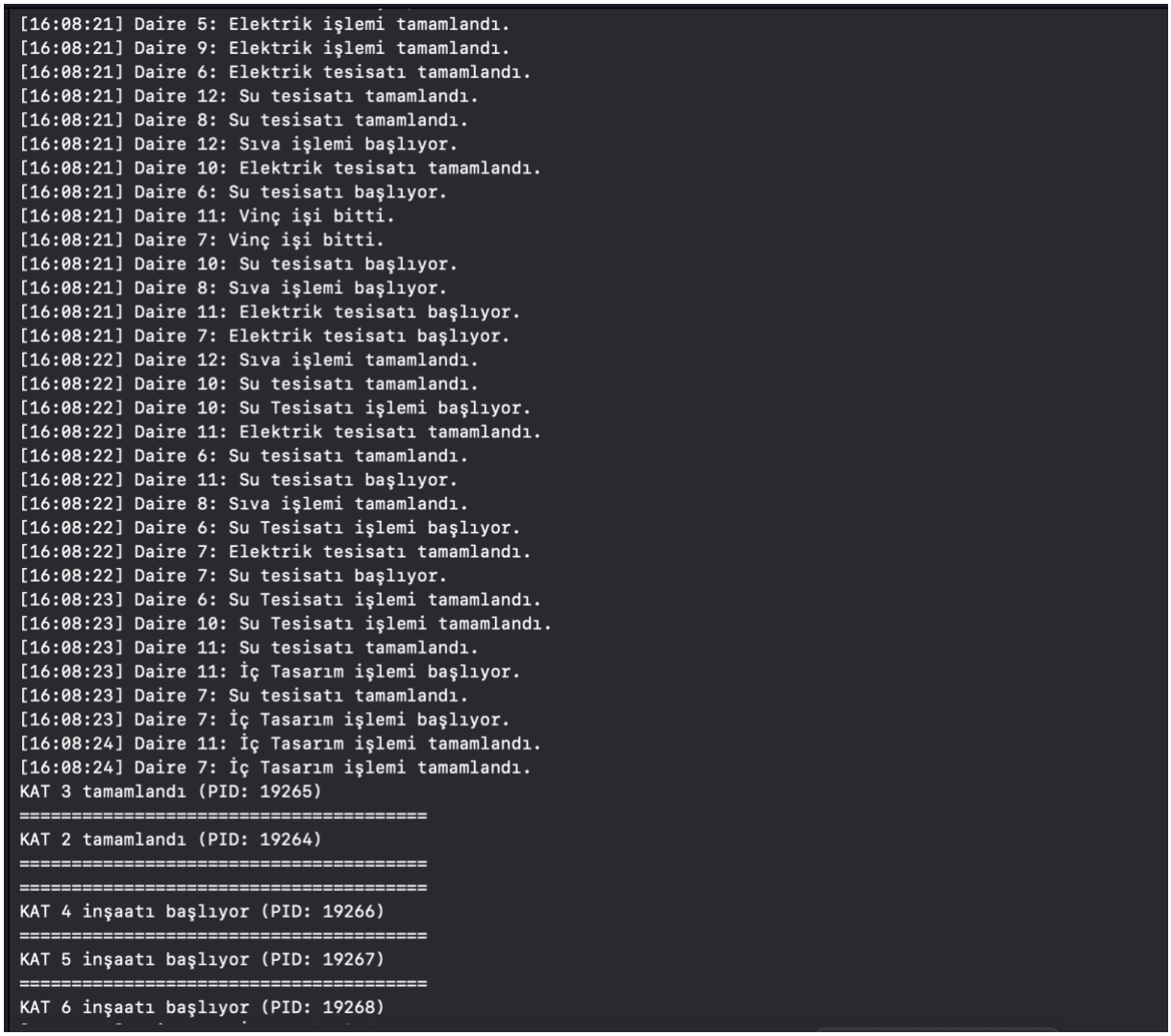
Bu proje kapsamında, işletim sistemleri dersinin temel kavramları olan process/thread kullanımı, senkronizasyon, mutex, semaphore, koşul değişkenleri ve kritik bölgeler somutlaştırıldı. Gerçek bir apartman inşa senaryosu üzerinden yapılan modelleme ile hem sistem kaynaklarının verimli paylaşımı hem de veri güvenliği sağlandı.

Loglamalar sayesinde tüm süreç takip edilebilir hale geldi. Gelişmiş yapı, süreç bazlı kontrol akışı ve kaynak paylaşımı açısından başarılı bir örnek sunmaktadır.

---

7. KOD ÇIKTISI

Aşağıdaki ekran görüntüsü, simülasyon çalıştırıldığında terminalde üretilen çıktının bir kısmını göstermektedir. Her satır, bir dairenin belirli bir işlemine ait zaman damgalı log bilgisidir. Bu çıktılar, programın senkronize bir şekilde çalıştığını ve dairelerin inşa sürecini başarılı biçimde simüle ettiğini kanıtlar.



Yapı Özeti:

- Her kat, sırasıyla ve bağımsız bir fork() çağrısıyla başlatılmıştır.
- Her katta 4 adet thread (daire) aynı anda başlar, ancak ortak kaynaklara erişirken sıra bekler.
- Her işlem başında ve sonunda zaman damgası basılır (örneğin: [16:08:21]).
- İşlemler: Vinç kullanımı, elektrik tesisatı, su tesisatı, bağımsız görev (örneğin sıva).

Gözlemlenebilecek Özellikler:

1. **Zaman Damgası (Timestamp):**
   Örneğin [16:08:21], işlemin başlama/bitiş anını net bir şekilde gösterir. Bu, hem senkronizasyonu hem de gecikmeleri gözlemlemeyi sağlar.
2. **Kaynak Senkronizasyonu:**
   - Vinç: Tek thread erişebilir. Daire 11 vinç kullandıktan sonra diğer dairelerin erişmesine izin verilir.
   - Elektrik: Aynı anda yalnızca 2 thread. Daire 10 ve 11 elektrik tesisatına aynı anda erişebilirken, diğer daireler bekler.
   - Su: Sadece 1 thread erişebilir. Daire 6 su tesisatına girerken diğerleri sırayla beklemiş.
3. **Sıralı Kat İnşası:**

   KAT 3 tamamlandı (PID: 19265)
   KAT 2 tamamlandı (PID: 19264)
   ...

   Her kat tamamlandığında log'a yazılır. Ebeveyn process, bir alt kat tamamlanmadan üst katın başlamasına izin vermez. Bu, wait() çağrısıyla sağlanır.

4. **Paralel Çalışan Daireler:**
   Aynı kat üzerindeki dairelerin işlemleri zaman olarak örtüşür (örneğin: Daire 11 elektrik çekerken daire 10 su tesisatı yapabilir).
5. **Kat Başlangıç ve Bitiş Logları:**

   ==========
   KAT 4 inşaatı başlıyor (PID: 19266)

   Bu loglar, her process'in (katın) başladığını ve tamamlandığını gösterir. PID bilgisi sayesinde hangi process'in hangi kata ait olduğu takip edilebilir.