grafik, yazı tipi, grafik tasarım, logo içeren bir resim

Açıklama otomatik olarak oluşturuldu

**BURSA TEKNİK ÜNİVERSİTESİ İŞLETİM SİSTEMLERİ ÖDEVİ**

**2024-2025 BAHAR DÖNEMİ**

**Çok Katlı Bir Apartmanın İnşası Üzerinden Process, Thread ve Senkronizasyon Kavramlarının Modellenmesi**

**SIDIKA FİRAT TALA KİBAR**

**2136085944 21360859116**

**İÇİNDEKİLER**

1. **GİRİŞ** ................................................................................................................ 3
   * Projenin Amacı ve Kapsamı
   * Process, Thread ve Senkronizasyon Kavramlarının Apartman İnşaatı ile İlişkilendirilmesi
2. **APARTMAN MODELİ** ...................................................................................... 3
   * Yapının Özellikleri (Kat Sayısı, Daire/Kat Oranı)
   * Kat ve Daire Modelinin Açıklanması
   * Neden Bu Modelin Seçildiği
3. **TEKNİK UYGULAMA** ...................................................................................... 5
   * Process Oluşturma (fork() Fonksiyonu)
   * Thread Oluşturma (pthread\_create() Fonksiyonu)
   * Senkronizasyon Mekanizmaları (mutex Kullanımı)
4. **SENKRONİZASYON VE YARIŞ KOŞULLARI** ............................................. 6
   * Process Senkronizasyonu (wait() Fonksiyonu)
   * Thread Senkronizasyonu (Ortak Kaynak Yönetimi)
   * Yarış Koşulu (Race Condition) ve Çözüm Yöntemleri
5. **SONUÇ VE ÖNERİLER** ................................................................................. 7
   * Projenin Başarılı Yönleri
   * Karşılaşılan Zorluklar ve Çözümleri
   * Gelecek Çalışmalar için Öneriler

**PROJE RAPORU**

**Giriş**

**1. GİRİŞ**

Bu raporda, işletim sistemlerinin temel yapı taşlarından olan process, thread ve senkronizasyon kavramları, günlük yaşamdan bir örnekle açıklanmıştır. Özellikle bir apartman inşaat süreci üzerinden bu kavramların nasıl modellenebileceğini gösteren bir simülasyon çalışması yapılmıştır. Amacımız, teorik olarak öğrendiğimiz bu konuları daha anlaşılır ve somut hale getirmektir.

Projede, 10 katlı ve her katında 4 daire bulunan bir apartman modeli ele alınmıştır. Bu modelde her katın inşası, birbirinden bağımsız çalışan process'lere benzetilmiştir. Her katta bulunan 4 daire ise, o kat process'i içinde çalışan thread'ler olarak düşünülmüştür. Yani her kat ayrı bir process iken, içindeki daireler aynı anda işlem yapabilen thread'lerdir.

Bu modelde:

* **Katlar**, fork() fonksiyonu ile oluşturulan process'leri temsil eder.
* **Daireler**, pthread\_create() fonksiyonu ile oluşturulan thread'leri ifade eder.

Katlar arasında hiyerarşik bir sıra bulunmaktadır. Bir üst katın inşasına başlanabilmesi için alt katın tamamlanmış olması gerekmektedir. Bu durum, işletim sistemlerindeki process senkronizasyonuna örnek teşkil eder. Aynı şekilde, aynı katta bulunan daireler eş zamanlı olarak çalışabilse de, su tesisatı veya elektrik sistemi gibi ortak kaynakları kullanırken mutex (karşılıklı kilitleme) yöntemleriyle senkronize edilmeleri gerekir. Bu da thread senkronizasyonu ile benzerlik gösterir.

Günümüz işletim sistemlerinde çoklu process ve thread'lerin aynı anda çalışabilmesi, kaynakların verimli kullanımı açısından büyük önem taşır. Bu projede, bir inşaat süreci üzerinden bu özellikler simüle edilerek hem teorik bilgilerin pekiştirilmesi hem de gerçek hayatla ilişkilendirilmesi hedeflenmiştir.

**Projenin Modeli**

**2. APARTMAN MODELİ**

Bu projede geliştirilen simülasyonun temelinde 10 katlı ve her katta 4 daireden oluşan bir apartman yer almaktadır. Bu yapı, işletim sistemlerinin temel bileşenleri olan **process** (süreç) ve **thread** (iş parçacığı) kavramlarını modellemek için seçilmiştir. Her bir kat, kendi içinde bağımsız olarak çalışan bir süreç gibi ele alınırken, katlar içinde yer alan daireler ise o sürecin alt görevleri olan iş parçacıkları (thread) olarak tanımlanmıştır.

**2.1 Yapının Özellikleri**

| **Özellik** | **Açıklama** |
| --- | --- |
| Kat Sayısı | 10 |
| Daire/Kat | 4 |
| Toplam Daire | 40 |

Bu modele göre apartman inşaatı şu şekilde temsil edilmiştir:

* **Her bir kat**, işletim sistemi terimiyle bir **process** olarak tanımlanmıştır. Bu süreçler birbirinden bağımsız çalışan alt programlar gibi düşünülebilir. Programda bu yapı fork() fonksiyonu kullanılarak oluşturulmuştur.
* **Her kat içindeki dört daire**, o kata ait **thread**’lerdir. Yani aynı process içerisinde çalışan ve ortak kaynaklara erişebilen alt iş parçacıklarıdır. Thread’ler pthread\_create() fonksiyonu ile oluşturulmuştur.

Bu sayede hem çoklu süreç yönetimi hem de süreç içi paralel görevlerin yönetimi aynı yapı içerisinde modellenmiştir. Aşağıda bu modelin yapısal temsili yer almaktadır:

**2.2 Kat ve Daire Modeli**

Kat 1 → Process 1

→ Thread 1.1 (Daire 1)

→ Thread 1.2 (Daire 2)

→ Thread 1.3 (Daire 3)

→ Thread 1.4 (Daire 4)

Kat 2 → Process 2

→ Thread 2.1 (Daire 5)

→ Thread 2.2 (Daire 6)

→ Thread 2.3 (Daire 7)

→ Thread 2.4 (Daire 8)

...

Kat 10 → Process 10

→ Thread 10.1 (Daire 37)

→ Thread 10.2 (Daire 38)

→ Thread 10.3 (Daire 39)

→ Thread 10.4 (Daire 40)

**2.3 Neden Bu Model?**

Bu yapı sayesinde:

* Katlar birbirinden bağımsız çalışabilir, ancak bir üst kat inşaatı başlamadan önce alt katın tamamlanması gerektiği için **process senkronizasyonu** zorunlu hale gelir.
* Aynı kat içindeki daireler eşzamanlı olarak işlem görebilir; fakat ortak kaynakları (örneğin su tesisatı veya elektrik sistemi) kullanırken dikkatli olunmalıdır. Bu durumda **thread senkronizasyonu** gereklidir.

**2.4 Kodlama Açısından Bakış**

Her fork() çağrısı, yeni bir process (kat) oluşturur. Bu process içinde pthread\_create() ile 4 adet thread (daire) oluşturulur. Bu da bize toplamda:

* **10 process (kat)** ve
* **40 thread (daire)** anlamına gelir.

Bu yapı sayesinde proje hem çoklu process hem de çoklu thread mantığını tek bir simülasyonda birleştirmiştir. Ayrıca thread'lerin aynı anda ortak kaynaklara erişmesi durumunda yaşanabilecek **yarış koşulları (race conditions)** da gerçekçi bir şekilde simüle edilmiştir.

**Teknik Uygulama**

**3. TEKNİK UYGULAMA**

Bu projede programlama dili olarak sadece **C dili** kullanılmıştır. Projenin temel amacı, işletim sistemlerinde çok süreçli ve çok iş parçacıklı çalışmayı simüle etmektir. Bu doğrultuda, her katın ayrı bir process (süreç), her dairenin ise o sürecin içinde çalışan birer thread (iş parçacığı) olması sağlanmıştır. Süreçler fork() fonksiyonu ile, thread’ler ise pthread\_create() fonksiyonu ile oluşturulmuştur.

Ayrıca, dairelerin ortak kaynaklara (örneğin su ve elektrik altyapısı) erişim sırasında çakışma yaşamaması için **senkronizasyon mekanizmaları** (özellikle mutex’ler) kullanılmıştır.

**3.1 Process Oluşturma (fork)**

Her katı bağımsız bir süreç olarak tanımlayabilmek için C dilinde fork() fonksiyonu kullanılmıştır. fork(), çalıştığı anda mevcut süreci kopyalayarak yeni bir alt süreç (child process) oluşturur. Böylece her kat kendi process’i olarak paralel çalışabilir.

pid\_t pid = fork();

if (pid == 0) {

// Burada child process yani ilgili kat işlemleri yapılır.

} else {

wait(NULL); // Parent process alt katın tamamlanmasını bekler.

}

Burada dikkat edilmesi gereken önemli nokta şudur: **Üst katın inşaatı, alt kat tamamen bitmeden başlayamaz.** Bu yüzden parent process (üst kat), wait() fonksiyonu ile child process’in (alt kat) bitmesini bekler. Bu yöntemle katlar arasında sıralı bir inşaat süreci sağlanır.

**3.2 Thread Oluşturma (pthread\_create)**

Her process içinde, o kata ait 4 dairenin inşaatı gerçekleştirilir. Bu işlem, **thread’ler** yardımıyla eşzamanlı olarak yürütülür. Thread’ler C dilinde pthread kütüphanesi ile oluşturulmuştur.

pthread\_t threadler[DAIRE\_SAYISI];

pthread\_create(&threadler[i], NULL, daire\_islemi, &daire\_no[i]);

Her thread, daire\_islemi() adlı bir fonksiyonu çalıştırarak daire inşaatını simüle eder. Bu işlem tüm dairelerde aynı anda başlayabilir; çünkü aynı process (kat) içindedirler ve birbirinden bağımsız işler yaparlar. Ancak su tesisatı ve elektrik gibi paylaşılan işler için dikkatli olunmalıdır.

**3.3 Senkronizasyon (mutex)**

Bazı işler, daireler arasında **ortak kaynaklar** üzerinden yürütüldüğü için **eş zamanlı yapılmamalıdır**. Örneğin, bir katta sadece bir daire aynı anda su tesisatına erişebilmelidir. Bu durumu kontrol altına almak için **mutex (karşılıklı dışlama kilidi)** yapısı kullanılmıştır:

pthread\_mutex\_lock(&su\_mutex);

// Su tesisatı kuruluyor

pthread\_mutex\_unlock(&su\_mutex);

Bu kod bloğunda:

* pthread\_mutex\_lock() fonksiyonu, mutex’i kilitler. Yani başka bir thread aynı anda bu bölgeye giremez.
* pthread\_mutex\_unlock() fonksiyonu ise kilidi açar, böylece sıradaki thread erişim sağlar.

Benzer şekilde elektrik altyapısı için de ayrı bir mutex kullanılmıştır. Bu sayede **yarış koşulları (race conditions)** engellenmiş ve işlemlerin güvenli şekilde yürütülmesi sağlanmıştır.

**Senkronizasyon & Yarış Durumu**

**4. SENKRONİZASYON VE YARIŞ KOŞULLARI**

Bu projede, işlemlerin doğru sırayla ve çakışmadan gerçekleşmesi için **senkronizasyon** mekanizmaları kullanılmıştır. Senkronizasyon, birden fazla işlem veya iş parçacığı (thread) aynı kaynaklara erişmeye çalıştığında oluşabilecek hataları önlemek için oldukça önemlidir.

Projede iki temel düzeyde senkronizasyon uygulanmıştır:

**4.1. Process (Süreç) Senkronizasyonu**

Her kat kendi başına bağımsız bir process olarak çalışsa da, **inşaat sırası** mantığı gereği katlar üst üste inşa edildiği için belirli bir sıraya uymak zorundadır. Örneğin:

* **Kat 2**, ancak **Kat 1 tamamen bittikten sonra** inşa edilmeye başlanabilir.
* Aynı şekilde Kat 3, Kat 2 tamamlanmadan başlayamaz.

Bu sıralı ilerlemeyi sağlamak için **wait() fonksiyonu** kullanılmıştır. fork() ile yeni bir kat (process) başlatıldığında, ana süreç (parent process) wait() komutuyla bir önceki sürecin (alt katın) bitmesini bekler. Bu sayede katlar arası yapı güvenliği sağlanır ve bir üst kat, altyapısı tamamlanmadan başlamaz.

**4.2. Thread (İş Parçacığı) Senkronizasyonu**

Her kat kendi içinde 4 daireye sahiptir ve bu dairelerin inşaatı eş zamanlı olarak (paralel) başlatılır. Ancak bu dairelerin bazı işleri **ortak kaynaklar üzerinden yapılır.**

Örneğin:

* Aynı kattaki tüm dairelerde **su tesisatı** ve **elektrik döşemesi** işlemleri yapılır.
* Ancak her dairenin aynı anda bu altyapıya müdahale etmesi, karışıklığa ve hatalara yol açabilir.

Bu durumu engellemek için **pthread\_mutex\_t** türünde tanımlanan **mutex (karşılıklı dışlama)** yapıları kullanılmıştır. Yani:

* Bir daire su tesisatıyla ilgilenirken diğer daireler bekler.
* Su işi bitince kilit açılır ve sıradaki daire işlemini yapar.

Bu sayede her daire, ortak kaynaklara sırayla erişmiş olur ve çakışma yaşanmaz.

**4.3 Yarış Koşulu (Race Condition)**

**Yarış koşulu**, birden fazla thread’in aynı anda aynı kaynağa erişmeye çalıştığında oluşabilecek bir problemdir. Bu durumda kaynak doğru ve güvenli şekilde kullanılamayabilir, sistem beklenmeyen şekilde davranabilir.

Bu projede örnek olarak:

* Su tesisatı, elektrik döşemesi gibi altyapılar,
* Veya tek bir vinç/asansör varsa bu araçlar ortak kaynak sayılır.

Eğer bu kaynaklara **kontrolsüz erişim** olsaydı, daireler aynı anda bu kaynakları kullanmaya çalışırdı ve bu da **yarış koşulu** yaratırdı.

İşte bu yüzden:

* **mutex** kullanılarak bu kaynaklar kilitlenmiş,
* Sadece bir thread’in erişmesine izin verilmiş,
* Diğer thread’ler ise sırayla beklemiştir.

Bu yöntem sayesinde simülasyon hem gerçekçiliğini korumuş hem de işletim sistemlerindeki senkronizasyon kavramları başarıyla modellenmiştir.

**Sonuç ve Öneriler**

**5. SONUÇ**

Bu projede, işletim sistemleri dersinde işlediğimiz process (süreç), thread (iş parçacığı) ve senkronizasyon konularını, apartman inşaatı örneği üzerinden uygulamalı olarak çalıştık. Hocamızın derste anlattığı bu kavramları somutlaştırmak ve daha iyi anlamak için böyle bir proje geliştirmeye karar verdik.

Projemizde şu şekilde bir yaklaşım benimsedik:

* Her katı fork() fonksiyonu ile oluşturduğumuz ayrı bir process olarak tasarladık
* Her daireyi ise pthread\_create() ile oluşturulan thread'lerle temsil ettik

**Uygulama Aşamasında Şunları Gerçekleştirdik:**

1. Katlar arasındaki sıralı ilerlemeyi sağlamak için wait() fonksiyonunu kullandık. Böylece bir üst katın inşasına ancak alt kat tamamlandıktan sonra başlanabildi.
2. Aynı kattaki dairelerin paralel çalışabilmesi için thread yapısını uyguladık.
3. Dairelerin ortak kaynaklara (su tesisatı, elektrik) erişiminde mutex kullanarak:
   * Kritik bölgeleri koruduk
   * Yarış durumlarını (race condition) engelledik
   * İşlemlerin güvenli şekilde tamamlanmasını sağladık

**Projeden Edindiğimiz Kazanımlar:**

* C dilinde çoklu süreç ve thread yönetimini uygulamalı olarak öğrendik
* Senkronizasyon mekanizmalarının önemini somut örneklerle gördük
* Teorik bilgilerimizi pratiğe dökme fırsatı bulduk
* Takım çalışmasıyla problem çözme becerilerimizi geliştirdik

**Karşılaştığımız Zorluklar:**

* İlk başta thread'lerin senkronizasyonunda sorunlar yaşadık
* Mutex kullanımında bazı deadlock durumları oluştu
* Process'ler arası iletişimi ayarlamakta zorlandık

Bu proje sayesinde işletim sistemlerinin temel çalışma prensiplerini daha iyi anladık. Özellikle süreç yönetimi ve eş zamanlı programlama konularında önemli deneyimler kazandık. Derste gördüğümüz teorik bilgilerin pratikte nasıl uygulandığını görmek, konuyu kavramamızı kolaylaştırdı.