BURSA TEKNİK ÜNİVERSİTESİ BLM0332 İŞLETİM SİSTEMLERİ DERSİ DÖNEM ÖDEVİ

21360859054 – İbrahim Semih Temiz

21360859061 – Furkan Çapkın

2024 - 2025

BAHAR DÖNEMİ

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

1. Giriş

Modern işletim sistemlerinde çok görevli çalışma (multitasking), çoklu iş parçacığı (multi-threading) ve çoklu işlem (multi-processing) kavramları, kaynakların verimli kullanımı ve aynı anda birden fazla işlemin gerçekleşmesi amacıyla kullanılmaktadır. Bu proje kapsamında, 10 katlı ve her katta 4 daire bulunan bir apartmanın inşa süreci, bu kavramların somut bir benzetimi olarak modellenmiştir.

Bu raporda, geliştirilen simülasyon programı detaylarıyla açıklanmakta, process ve thread yapıları, senkronizasyon yöntemleri ve yarış durumlarına (race condition) karşı alınan önlemler aktarılmaktadır.

2. Proje Modeli ve Process Kavramı

Apartman Yapısı:

- Toplam Kat: 10
- Her Kat: 4 daire
- Toplam Daire: 40

Kavramsal Eşleme:

- Her bir **kat** bir "process" olarak ele alınmıştır. Ancak bu proje C dilinde ve Windows API üzerinden gerçekleştirildiği için tüm kat işlemleri aynı fiziksel process içerisinde mantıksal olarak sıralı şekilde uygulanmıştır.
- Her kat altında yer alan **daireler**, bağımsız **thread**'ler olarak modellenmiştir.

Process Farklılığı Açıklaması:
Projenin Windows ortamında sorunsuz çalışabilmesi için Windows API kullanılmıştır. Projenin tanımında her katın ayrı bir "process" olarak gerçekleştirilmesi önerilmektedir. Ancak Windows API ile C dili ortamında process oluşturma (CreateProcess) hem daha karmaşık bir yapı gerektirmekte hem de thread'ler arası paylaşılan verilerin senkronizasyonunu güçleştirmektedir. Bu nedenle, **her katı temsili olarak mantıksal bir process gibi düşünüp**, kodda **sıralı çalışan döngü** ile uygulanmıştır. Bu yaklaşımda bir kat tamamlanmadan diğerinin başlamaması sağlanmış, böylece process senkronizasyonu join() benzeri mantıkla gerçekleştirilmiştir.

Koddan Alıntı – Kat Döngüsü:

```
for (int kat = 1; kat <= KAT_SAYISI; kat++) {
    ...
    // her katta daire thread'leri başlatılır
    for (int i = 0; i < DAIRE_SAYISI; i++) {
        int *daire_no = malloc(sizeof(int));
        *daire_no = (kat - 1) * DAIRE_SAYISI + i + 1;
        daire_threadleri[i] = CreateThread(NULL, 0, daire_insa_et, daire_no, 0, NULL);
    }
    WaitForMultipleObjects(DAIRE_SAYISI, daire_threadleri, TRUE, INFINITE);
    ...
}
```

Bu yapı, her katın sıralı ve bağımsız bir süreç (mantıksal process) olarak işlediğini göstermektedir.

3. Kod Yapısı ve Thread İşlemleri

Her kat için döngü ile işlem başlatılmıştır. Her katta, 4 adet thread yaratılarak dairelerin inşası başlatılmış, bu thread'ler WaitForMultipleObjects fonksiyonu ile tamamlanmaları beklenmiştir.

Thread fonksiyonlarında sırayla aşağıdaki işlemler gerçekleştirilmektedir:

1. Hazırlık
2. Asansör kullanımı (mutex ile korunur)
3. Vinç işlemi (semaphore ile sınırlı paylaşım)
4. Tesisat kurulumu (mutex ile korunur)
5. İç düzenleme (serbest paralel işlem)

Koddan Alıntı – Thread Fonksiyonu Örneği:

```
DWORD WINAPI daire_insa_et(LPVOID arg) {
    int daire_no = *((int *)arg);
    free(arg);
    guvenli_yaz("(Daire %d) Calismaya basladi.\n", daire_no);
    ...
    guvenli_yaz("(Daire %d) TAMAMLANDI!\n", daire_no);
    return 0;
}
```

Koddan Alıntı – Konsol Yazımının Senkronizasyonu:

```
void guvenli_yaz(const char* format, ...) {
   WaitForSingleObject(print_mutex, INFINITE);
   zaman_damgasi_yaz();
   va_list args;
   va_start(args, format);
   vprintf(format, args);
   va_end(args);
   ReleaseMutex(print_mutex);
}
```

Bu özel fonksiyon sayesinde konsol çıktıları karışmadan düzgün biçimde yazdırılmaktadır.

4. Senkronizasyon ve Yarış Durumu

Kullanılan Senkronizasyon Araçları:

- HANDLE asansor_mutex;
- HANDLE tesisat_mutex;
- HANDLE vinc_semaphore;
- HANDLE print_mutex; (konsol çıktısının karışmaması için)

Bu senkronizasyon araçları sayesinde thread'lerin ortak kaynaklara aynı anda erişmesi engellenmiştir. Mutex'ler sadece bir thread'e izin verirken, semaphore sınırlı sayıda thread'e izin vermiştir.

Race Condition Önlemleri:

- Asansör, vinç ve tesisat gibi paylaşılan kaynaklara mutex/semaphore ile erişim kontrolü sağlanmıştır.
- Konsola çıktı yazımı da mutex ile kontrol altına alınmıştır, böylece çıktılar karışmadan düzenli şekilde yazılmıştır.

Koddan Alıntı – WaitForMultipleObjects Kullanımı:

```
WaitForMultipleObjects(DAIRE_SAYISI, daire_threadleri, TRUE, INFINITE);
```

Bu komut ile aynı kattaki tüm dairelerin tamamlanması beklenmiştir. Üst katın inşası, ancak bu fonksiyonun tamamlanmasından sonra başlamaktadır.

5. Sonuç ve Değerlendirme

Bu proje ile işletim sistemlerindeki thread ve process kavramları, senkronizasyon mekanizmaları ve kaynak paylaşımı konuları somut olarak uygulanmıştır. Kodda her aşama detaylı yorumlarla anlatılmış, tüm thread'ler senkronize edilmiş ve çıktı düzeni güvence altına alınmıştır.

Ekstra Katkı: Konsol çıktıları mutex ile kontrol altına alınarak güvenli yazdırma fonksiyonu (guvenli_yaz) geliştirilmiştir.

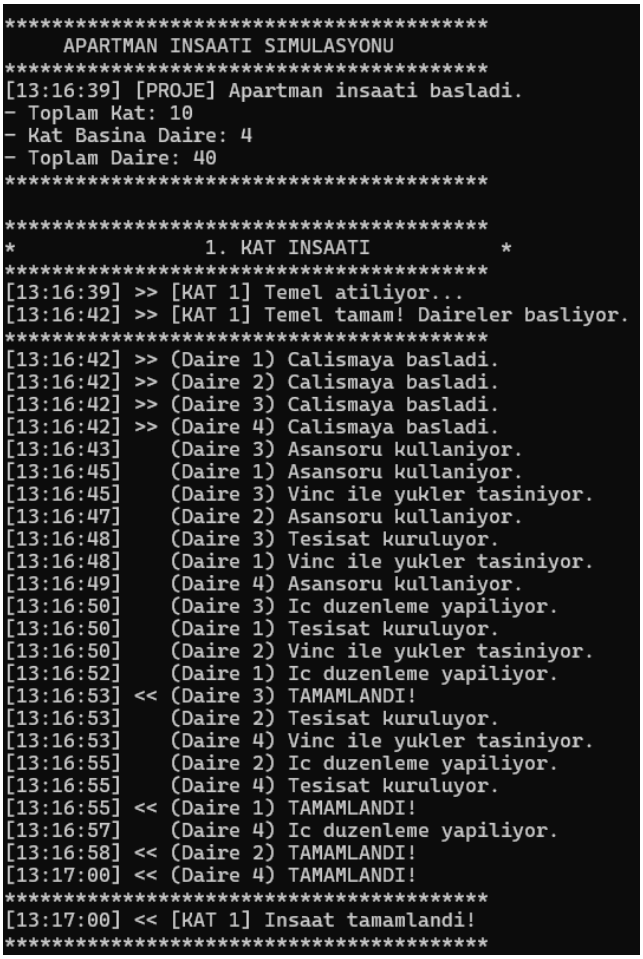
Video: Projeye ait çalışmanın video anlatımı YouTube üzerinden izleyebilirsiniz:

C ile Çok Katlı Apartman İnşaatı Simülasyonu | Thread, Process ve Senkronizasyon

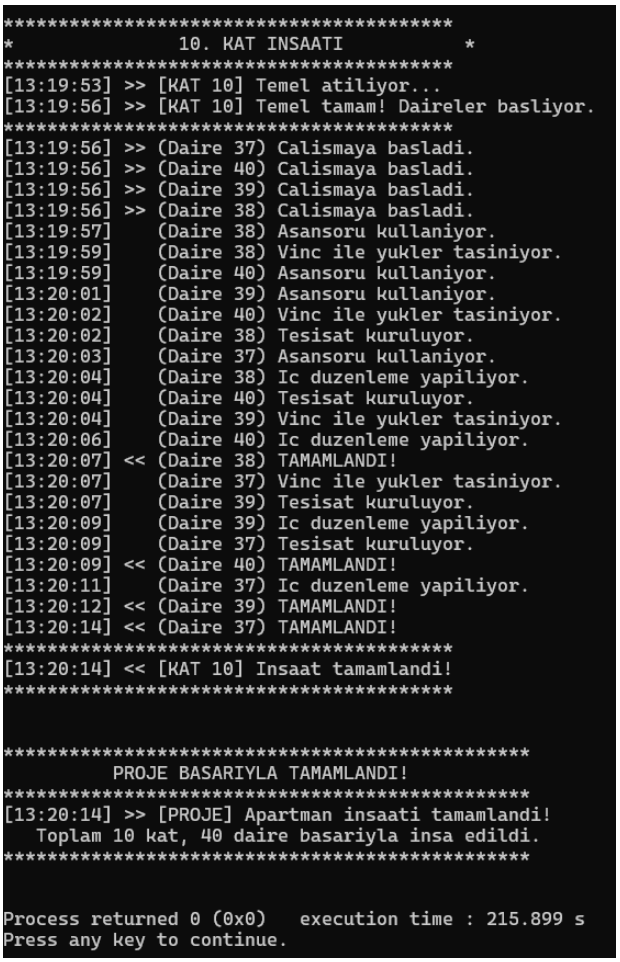
6. Konsol Ekran Görüntüleri

Aşağıda, proje kodunun çalıştırılması sonucu elde edilen bazı örnek konsol ekran çıktıları yer almaktadır. Bu çıktılar, thread'lerin eş zamanlı ve senkronize şekilde nasıl çalıştığını ve çıktının karışmadan üretildiğini göstermektedir.

Örnek Çıktı – Kat Başlangıcı ve Daire İşlemleri:



...



Bu çıktılar, proje çıktısının düzenli bir zaman çizelgesinde ve çakışmadan işlendiğini ortaya koymaktadır.