Yusuf Mert Özkul

21360859057

Çok İpli (Multithreaded) Bina İnşaatı Simülasyonu Açıklama Dokümanı

1. Giriş

Bu belge, C dili kullanılarak geliştirilen ve POSIX thread (pthread) kütüphanesinden faydalanan bir çok iş parçacıklı bina inşaatı simülasyonu uygulamasının detaylı açıklamasını sunar. Kod, her katta 4 dairenin bulunduğu toplam 10 katlı bir apartmanın inşaat sürecini modellemektedir.

Bu simülasyonun temel amacı, eşzamanlı çalışan iş parçacıklarının (threads) senkronizasyonunun nasıl sağlanacağını göstermektir. Özellikle:

- Vinç kullanımı

- Boyacı ekibi paylaşımı

- Katlar arası bağımlılık (önceki kat tamamlanmadan üst kat başlayamaz)

durumları başarıyla simüle edilmiştir.

2. Temel Yapılar ve Global Değişkenler

2.1. Sabit Tanımlar

```
// Sabit degerler
#define FLOOR_COUNT 10
#define APARTMENT_COUNT 4
```

Apartmanda 10 kat, her katta 4 daire olduğunu belirtir.

2.2. Yapılar (Structs)

Floor Yapısı

```
/ Her katin durumunu takip eden yapi
ypedef struct {
    int floor_number;
    int is_completed;
    pthread_mutex_t floor_mutex;
    sem_t floor_ready;
Floor;
```

Kat bilgilerini ve senkronizasyon yapılarını tutar.

Apartment Yapısı

```
/ Her dairenin bilgilerini tutan yapi
ypedef struct {
    int floor_number;
    int apartment_number;
Apartment;
```

Her bir daire için kat ve daire numarası tutulur.

2.3. Global Değişkenler

```
// Global degiskenler
Floor building_floors[FLOOR_COUNT];
pthread_mutex_t crane_mutex;     // Vinc erisimi icin mutex
pthread_mutex_t painter_mutex;   // Boyaci ekibi erisimi icin mutex
```

3. Fonksiyonlar ve İşleyiş

3.1. construct_apartment (Daire İnşaatı)

Bir daireyi simüle eder. İşlem adımları:

1. Başlama bildirimi

2. Vinç kullanımı için mutex kilidi

3. İnşaat işlemleri

4. Boyacı ekibi için mutex kilidi

5. Tamamlanma bildirimi

```c
void* construct_apartment(void* arg) {
    Apartment* apartment = (Apartment*)arg;

    printf("Kat %d, Daire %d insaata basladi\n", apartment->floor_number + 1, apartment->apartment_number + 1);

    // Vinç kullanımı için mutex kilidi
    // Aynı anda sadece bir daire vinç kullanabilir
    pthread_mutex_lock(&crane_mutex);
    printf("Kat %d, Daire %d vinç kullaniyor\n", apartment->floor_number + 1, apartment->apartment_number + 1);
    Sleep(500); // Vinç kullanım simülasyonu - 1 saniye
    pthread_mutex_unlock(&crane_mutex);

    // Daire inşaat işlemleri simülasyonu
    // Temel işlemler: duvar örme, tesisat, sıva vb.
    Sleep(500); // İnşaat işlemi simülasyonu - 2 saniye

    // Boyacı ekibi kullanımı için mutex kilidi
    // Aynı anda sadece bir daire boyacı ekibini kullanabilir
    pthread_mutex_lock(&painter_mutex);
    printf("Kat %d, Daire %d boya islemi yapiliyor\n", apartment->floor_number + 1, apartment->apartment_number + 1);
    Sleep(500); // Boya işlemi simülasyonu - 1.5 saniye
    printf("Kat %d, Daire %d boya islemi tamamlandi\n", apartment->floor_number + 1, apartment->apartment_number + 1);
    pthread_mutex_unlock(&painter_mutex);

    printf("Kat %d, Daire %d tamamlandi\n", apartment->floor_number + 1, apartment->apartment_number + 1);
    free(apartment);
    return NULL;
}
```

3.2. construct_floor (Kat İnşaatı)

Bir katı simüle eder. İşlem adımları:

1. Alt katın tamamlanmasını bekler
2. Kat inşaatı başlatılır
3. Daireler için thread'ler oluşturulur
4. Dairelerin tamamlanmasını bekler
5. Üst kata sinyal gönderilir

```c
// Kat insaati thread fonksiyonu
// Her kat kendi dairelerinin insaatini yonetir
void* construct_floor(void* arg) {
    int floor_number = *((int*)arg);
    pthread_t apartment_threads[APARTMENT_COUNT];

    // Alt katin tamamlanmasini bekle
    // Ilk kat haric tum katlar kendinden onceki katin tamamlanmasini bekler
    if (floor_number > 0) {
        sem_wait(&building_floors[floor_number-1].floor_ready);
    }

    printf("Kat %d insaati basladi\n", floor_number + 1);

    // Kattaki tum dairelerin insaatini baslat
    for (int i = 0; i < APARTMENT_COUNT; i++) {
        Apartment* new_apartment = malloc(sizeof(Apartment));
        new_apartment->floor_number = floor_number;
        new_apartment->apartment_number = i;
        pthread_create(&apartment_threads[i], NULL, construct_apartment, new_apartment);
    }

    // Kattaki tum dairelerin tamamlanmasini bekle
    for (int i = 0; i < APARTMENT_COUNT; i++) {
        pthread_join(apartment_threads[i], NULL);
    }

    printf("Kat %d tamamen tamamlandi\n", floor_number + 1);
    building_floors[floor_number].is_completed = 1;

    // Bir sonraki kata baslamasi icin sinyal gonder
    sem_post(&building_floors[floor_number].floor_ready);

    free(arg);
    return NULL;
}
```

3.3. main Fonksiyonu

Ana işleyiş:

1. Mutex ve semaforlar başlatılır

2. Katlar için iş parçacıkları başlatılır

3. Katların tamamlanması beklenir

4. Temizlik işlemleri yapılır

```c
int main() {
    pthread_t floor_threads[FLOOR_COUNT];

    // Mutex ve semaforlari baslat
    pthread_mutex_init(&crane_mutex, NULL);
    pthread_mutex_init(&painter_mutex, NULL);  // Yeni boyaci mutex'i baslatiliyor
    for (int i = 0; i < FLOOR_COUNT; i++) {
        building_floors[i].floor_number = i;
        building_floors[i].is_completed = 0;
        pthread_mutex_init(&building_floors[i].floor_mutex, NULL);
        sem_init(&building_floors[i].floor_ready, 0, 0);
    }

    printf("Bina insaati basliyor...\n");

    // Tum katlarin insaatini baslat
    for (int i = 0; i < FLOOR_COUNT; i++) {
        int* current_floor = malloc(sizeof(int));
        *current_floor = i;
        pthread_create(&floor_threads[i], NULL, construct_floor, current_floor);
    }

    // Tum katlarin tamamlanmasini bekle
    for (int i = 0; i < FLOOR_COUNT; i++) {
        pthread_join(floor_threads[i], NULL);
    }

    printf("Bina insaati tamamlandi!\n");

    // Kullanilan kaynaklari temizle
    pthread_mutex_destroy(&crane_mutex);
    pthread_mutex_destroy(&painter_mutex);  // Boyaci mutex'i temizleniyor
    for (int i = 0; i < FLOOR_COUNT; i++) {
        pthread_mutex_destroy(&building_floors[i].floor_mutex);
        sem_destroy(&building_floors[i].floor_ready);
    }

    return 0;
}
```

4. Eşzamanlılık ve Senkronizasyon

4.1. Mutex Kullanımı

- crane_mutex: Vinç paylaşımı

- painter_mutex: Boyacı paylaşımı

4.2. Semaphore Kullanımı
- Her kat bir önceki tamamlandığında başlar

- `sem_wait` ve `sem_post` kullanılır

5. Bellek Yönetimi ve Temizlik
- malloc ile alınan bellekler `free` ile serbest bırakılır

- Mutex ve semaforlar `destroy` edilir

6. Sonuç
Bu kod, çok iş parçacıklı programlamanın temel yapı taşlarını çok iyi bir şekilde sergilemektedir.

7. Ekran Çıktıları ve Görseller