T.C. MARMARA ÜNİVERSİTESİ TEKNOLOJİ FAKÜLTESİ

2025-2026 EĞİTİM ÖĞRETİM YILI

GÜZ DÖNEMİ BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ İŞLETİM SİSTEMLERİ DERSİ PROJE DOKÜMANI

171423008 – Hasan Dural

170423039 – Emine Umay Kılıç

171423005 – Doğukan Yılmaz

170420510 – Ömer Musab Çiçek

İŞLETİM SİSTEMLERİ PROJE RAPORU

1. GİRİŞ VE PROJENİN AMACI

1.1. Giriş

Günümüz modern işletim sistemlerinin en kritik görevlerinden biri, sistem kaynaklarını (CPU, RAM, Disk) kullanıcılar arasında adil ve güvenli bir şekilde paylaştırmaktır. Çok kullanıcılı sistemlerde, veri güvenliğinin sağlanması (izolasyon) ve disk alanının kontrolsüz kullanımının engellenmesi (kota yönetimi) hayati önem taşır.

Bu proje kapsamında, istemci-sunucu (client-server) mimarisine dayalı, web tabanlı bir **Sanal İşletim Sistemi Dosya Yöneticisi** geliştirilmiştir. Proje, gerçek bir işletim sistemi çekirdeğinin (kernel) dosya yönetim modülünü simüle ederek; kullanıcı oluşturma, oturum yönetimi, güvenli dosya saklama, işlem denetimi (logging) ve disk kotası takibi gibi temel fonksiyonları yerine getirir.

1.2. Projenin Temel Amaçları

Bu çalışmanın temel hedefleri şunlardır:

1. **Güvenlik ve Kimlik Doğrulama:** Kullanıcı şifrelerinin açık metin olarak değil, **SHA-256** algoritması ile hashlenerek saklanması.
2. **Kaynak Yönetimi (Kota):** Her kullanıcıya tanımlanan disk alanının (Quota) anlık olarak takip edilmesi ve limit aşımının engellenmesi.
3. **İzin Mekanizmaları (RWX):** Okuma (Read), Yazma (Write), Değiştirme (Overwrite) ve Çalıştırma (Execute) izinlerinin mantıksal düzeyde simüle edilmesi.
4. **Kalıcılık (Persistence):** Sistem kapatılıp açıldığında veri kaybını önleyen bir veritabanı (JSON) ve fiziksel dosya yapısının kurulması.
5. **Denetim İzleri (Logging):** Sistem güvenliği için yapılan tüm kritik işlemlerin (giriş, silme, oluşturma) zaman damgasıyla kayıt altına alınması.

---

2. SİSTEM MİMARİSİ VE KULLANILAN TEKNOLOJİLER

Proje, Modüler Programlama prensiplerine uygun olarak geliştirilmiştir. Sistem üç ana katmandan oluşur.

2.1. Kullanılan Teknolojiler

- **Backend (Sunucu):** Python 3 programlama dili ve Flask web çatısı kullanılmıştır. Python'un güçlü dosya işleme (File I/O) kütüphaneleri, disk üzerindeki fiziksel işlemleri yönetmek için tercih edilmiştir.

- **Frontend (İstemci):** HTML5, CSS3 ve JavaScript kullanılarak, kullanıcıya komut satırı (Terminal/Shell) deneyimi sunan bir arayüz tasarlanmıştır. Sunucu ile iletişim asenkron olarak **Fetch API** üzerinden sağlanır.
- **Veri Depolama:** Kullanıcı meta verileri (ID, Hashli Şifre, Kota Bilgisi) `users.json` dosyasında, fiziksel dosyalar ise sunucu diskinde kullanıcıya özel klasörlerde (`user_home/`) saklanır.
- **Güvenlik:** Python `hashlib` kütüphanesi ile tek yönlü şifreleme sağlanmıştır.

2.2. Modül Yapısı

1. **`app.py` (API Katmanı):** İstemciden gelen HTTP (POST/GET) isteklerini karşılar ve `FileSystem` sınıfına yönlendirir.
2. **`FileSystem.py` (İş Mantığı Katmanı):** Dosya oluşturma, silme, okuma ve yazma işlemlerinin mantıksal kontrollerini (izinler, sahiplik) yapar. Ayrıca loglama mekanizması burada çalışır.
3. **`QuotaManager.py` (Veri Katmanı):** Kullanıcı veritabanını yönetir, şifreleri hashler ve kota hesaplamalarını gerçekleştirir.

3. GERÇEKLEŞTİRİLEN İŞLETİM SİSTEMİ KAVRAMLARI

3.1. Güvenlik ve Hashing (SHA-256)

Gerçek işletim sistemlerinde (Linux, Windows) şifreler asla açık metin (plaintext) olarak saklanmaz. Bu prensipten hareketle, projemizde kullanıcı şifreleri `hashlib` kütüphanesi kullanılarak **SHA-256** algoritması ile şifrelenmiştir.

- **Kayıt Anı:** Kullanıcı kayıt olurken girilen şifre hashlenir (Örn: `1234` -> `03ac674...`) ve veritabanına yazılır.
- **Giriş Anı:** Girilen şifre tekrar hashlenir ve veritabanındaki hash ile karşılaştırılır. Bu sayede veritabanı dosyası çalınsa bile şifreler güvendedir.

3.2. Denetim ve Loglama (System Logging)

Sistemin izlenebilirliğini artırmak amacıyla bir Loglama mekanizması geliştirilmiştir. Kullanıcı girişleri, dosya oluşturma/silme işlemleri ve hatalar `system.log` dosyasına saniye hassasiyetiyle kaydedilir.

- *Örnek Log Kaydı:* `[2025-12-16 14:30:00] USER: user1 | ACTION: OVERWRITE_FILE | Path: /home/user1/not.txt`

3.3. Kota Yönetimi (Quota Enforcement)

Sistem, "Yazmadan Önce Kontrol Et" (Check-Before-Write) prensibiyle çalışır.

- Bir dosya oluşturulurken (`create`) veya dosya içeriği büyütülürken (`write`), sistem kullanıcının mevcut kullanımını ve limitini karşılaştırır.

- `Mevcut + Yeni Boyut > Limit` ise işlem işletim sistemi seviyesinde reddedilir.
- Dosya silindiğinde (`delete`) veya içi boşaltıldığında (`truncate`), serbest kalan alan anında kotadan düşülür ve kullanıcıya iade edilir.

3.4. Dosya İşlemleri ve İzinler

- **Overwrite vs Append:** Dosya sistemimiz iki farklı yazma modunu destekler. `write` komutu dosyanın sonuna ekleme yaparken (`append` mode), `overwrite` komutu dosyanın içeriğini tamamen silip yeniden yazar (`write` mode).
- **Truncate:** Dosyayı silmeden (metadata korunur) içeriğini 0 byte'a indirme özelliği eklenmiştir.
- **Admin Yetkileri:** `admin` kullanıcısı "root" yetkilerine sahiptir; tüm kullanıcıları silebilir, kotalarını değiştirebilir ve özel parametrelerle kullanıcı oluşturabilir.

---

4. KOD İNCELEMESİ VE ALGORİTMALAR

Bu bölümde, projenin en kritik fonksiyonlarına ait kod blokları ve çalışma mantıkları açıklanmıştır.

4.1. Güvenli Kullanıcı Oluşturma ve Hashing (`QuotaManager.py`)

Aşağıdaki kod bloğu, sadece Admin yetkisiyle çalışan, opsiyonel kota parametresi alan ve şifreyi hashleyerek kaydeden metottur.

Python

```python
def add_user(self, user_id, password, quota_mb=None):
        if quota_mb is None or quota_mb == "":
            final_quota_mb = DEFAULT_QUOTA_MB
        else:
            final_quota_mb = float(quota_mb)

        quota_bytes = final_quota_mb * self.MB

        self.user_quotas[user_id] = {
            'limit': quota_bytes,
            'usage': 0
        }
        # Şifreyi hashleyerek kaydet
        self.passwords[user_id] = self._hash_password(password)
        self.save_data()
        return final_quota_mb
```

4.2. Dosya İçeriğini Değiştirme (Overwrite) ve Loglama (`FileSystem.py`)

Aşağıdaki `overwrite_file` metodu, dosya içeriğini tamamen değiştirmek için kullanılır. İşlem başarılı olduğunda `log_action` tetiklenir ve sistem günlüğüne yazılır.

```
def overwrite_file(self, file_path, content):
        """Dosyanın içeriğini silip yenisini yazar (Overwrite)."""
        try: user_id = self._get_active_user()
        except PermissionError as e: return str(e)
        if user_id == "admin": return "HATA: Admin dosya içeriği
değiştiremez."
        if file_path not in self.files: return "HATA: Dosya
bulunamadı."
        if self.files[file_path]['owner'] != user_id: return "Erişim
Reddedildi."

        physical_path = self._get_physical_path(user_id, file_path)
        try:
            # 'w' kipi dosyayı açarken içeriğini siler!
            with open(physical_path, 'w', encoding='utf-8') as f:
                f.write(content)

            self.log_action(user_id, "OVERWRITE_FILE", f"Path:
{file_path}")
            return f"BAŞARILI: '{file_path}' içeriği değiştirildi."
        except Exception as e: return f"HATA: {e}"
```

4.3. Dosya Temizleme (Truncate) (`FileSystem.py`)

Bu metot, dosyayı diskten silmeden sadece içeriğini temizler (0 Byte yapar). Bu işlem dosya meta verisini korurken disk alanını boşaltır.

```
def truncate_file(self, file_path):
        """Dosyanın içeriğini tamamen temizler."""
        try: user_id = self._get_active_user()
        except PermissionError as e: return str(e)
        if user_id == "admin": return "HATA: Admin dosya değiştiremez."
        if file_path not in self.files: return "HATA: Dosya
bulunamadı."
        if self.files[file_path]['owner'] != user_id: return "Erişim
Reddedildi."

        physical_path = self._get_physical_path(user_id, file_path)
        try:
```

```
            # İçine hiçbir şey yazmadan 'w' ile açıp kapatmak içini
temizler.
            with open(physical_path, 'w', encoding='utf-8') as f:
                pass

            self.log_action(user_id, "TRUNCATE_FILE", f"Path:
{file_path} (Cleared)")
            return f"BAŞARILI: '{file_path}' içi temizlendi."
        except Exception as e: return f"HATA: {e}"
```

5. KOMUT LİSTESİ VE KULLANIM SENARYOLARI

Sistemde kullanılan komut seti ve işlevleri aşağıdadır:

Komut	Parametreler	Açıklama
register	<id> <pass> [quota]	**(Sadece Admin)** Yeni kullanıcı oluşturur. Kota opsiyoneldir. Şifre hashlenir.
login	<id> <pass>	Sisteme güvenli giriş yapar. Giriş işlemi loglanır.
create	<MB> <yol>	Belirtilen boyutta dosya oluşturur (Kota kontrolü yapılır).
write	<yol> <metin>	Dosyanın sonuna metin ekler (**Append** modu).
overwrite	<yol> <metin>	Dosya içeriğini siler ve yenisini yazar (**Replace** modu).
truncate	<yol>	Dosya içeriğini tamamen temizler (0 Byte).
cat	<yol>	Dosya içeriğini okur.

ls	-	Mevcut dizindeki dosyaları ve boyutlarını listeler.
delete	<yol>	Dosyayı siler ve kullanılan kotayı kullanıcıya geri verir.
status	-	Kullanıcının doluluk oranını ve limitini gösterir.
list_users	-	**(Sadece Admin)** Tüm kullanıcıları ve kotalarını raporlar.
set_quota	<id> <MB>	**(Sadece Admin)** Bir kullanıcının kotasını günceller.

6. SONUÇ VE KAZANIMLAR

Bu proje ile İşletim Sistemleri dersinde teorik olarak görülen;

1. **Kaynak İzolasyonu:** Kullanıcıların birbirlerinin dosyalarına erişememesi,
2. **Kaynak Kısıtlaması:** Disk kotaları ile adil kullanımın sağlanması,
3. **Veri Güvenliği:** Hashing ile şifre saklama ve Loglama ile denetim izi oluşturma,
4. **Dosya Sistemi Yönetimi:** `overwrite` ve `truncate` gibi gelişmiş dosya manipülasyonları

kavramları pratik bir uygulama üzerinde başarıyla gerçekleştirilmiştir. Sistem, kararlı yapısı ve güvenlik önlemleriyle basit bir simülasyonun ötesine geçerek gerçekçi bir prototip halini almıştır. Özellikle veri tutarlılığı sorunları (silinen dosyanın kotadan düşmemesi vb.) algoritmik düzenlemelerle çözülmüş ve hata yönetimi güçlendirilmiştir.