İŞLETİM SİSTEMLERİ DERSİ
PROJE RAPORU

Bilgisayar Mühendisliği BLM22371

Elif Uyar-2221221030

1.PROJENİN AMACI

Bu projenin amacı, Linux işletim sistemi üzerinde çalışan, birden fazla terminalden (multi-instance) kontrol edilebilen basit ama gelişmiş bir süreç (process) yönetim sistemi geliştirmektir.

Geliştirilen ProcX uygulaması ile kullanıcılar:

- Yeni programlar başlatabilmek,
- Çalışan programları listeleyebilmek,
- Programları güvenli bir şekilde sonlandırabilmek,
- Attached ve Detached modlarda süreç yönetimi yapabilmek

gibi temel işletim sistemi kavramlarını uygulamalı olarak deneyimleyebilmektedir.

Proje kapsamında;

- fork / execvp
- waitpid
- POSIX sinyalleri
- Shared Memory
- Semaphore
- Message Queue
- Thread (pthread)

gibi işletim sistemlerinin temel yapı taşları gerçek bir senaryo üzerinden kullanılmıştır.

2. PROJENİN ÇALIŞMA MANTIĞI

Bu proje, POSIX standartlarını kullanan çoklu süreç ve çoklu thread tabanlı bir süreç yönetim sistemidir. Programın temel amacı, birden fazla terminal üzerinden başlatılan süreçlerin ortak bir bellek alanı üzerinden senkronize şekilde yönetilmesini sağlamaktır.

2.1. Genel Mimari

ProcX, aşağıdaki POSIX mekanizmaları üzerine kuruludur:

- Process Management: fork(), execvp(), waitpid()
- Thread Management: pthread_create(), pthread_join()
- Shared Memory: shm_open(), mmap()
- Synchronization: POSIX semaphore (sem_open / sem_wait / sem_post)

- Inter-Process Communication (IPC): POSIX Message Queue (mq_open / mq_send / mq_receive)
- Signals: SIGINT, SIGTERM

Her ProcX çalıştırılması, ayrı bir process instance olarak kabul edilir ancak tüm instance'lar aynı IPC kaynaklarını paylaşır.

2.2. Süreç Başlatma Mekanizması

Kullanıcı yeni bir program çalıştırmak istediğinde:

1. fork() ile yeni bir child process oluşturulur.
2. Child process:
    - Attached modda ise terminale bağlı çalışır.
    - Detached modda ise setsid() ile terminalden koparılır.
3. execvp() ile kullanıcıdan alınan komut çalıştırılır.
4. Parent process, süreç bilgilerini shared memory'deki process tablosuna ekler.
5. Diğer ProcX instance'larına START olayı message queue üzerinden bildirilir.

Bu yapı sayesinde tüm terminaller aynı süreç listesini görebilir.

2.3. Process Takibi ve Zombie Önleme (Monitor Thread)

Her ProcX instance'ı bir Monitor Thread çalıştırır.

Monitor thread'in görevleri:

- Kendi oluşturduğu DETACHED child process'leri waitpid(WNOHANG) ile kontrol etmek,
- Biten süreçleri zombie kalmadan sistemden temizlemek,
- Owner'ı başka bir instance olan ancak artık çalışmayan (hayalet) süreçleri tespit edip tablodan silmek.

Önemli POSIX kuralı:
Bir process, child olmayan bir süreci waitpid() ile toplayamaz.Bu nedenle non-child süreçler kill(pid, 0) kontrolü ile temizlenir.

2.4. Süreçler Arası Haberleşme (IPC Listener Thread)

Her instance ayrıca bir IPC Listener Thread çalıştırır.

Bu thread:

- POSIX Message Queue üzerinden gelen START / TERMINATE olaylarını dinler,
- Olaylar başka bir ProcX instance'ından geldiyse ekrana bildirir,
- Çoklu terminal ortamında senkron kullanıcı geri bildirimi sağlar.

Bu sayede bir terminalde başlatılan veya sonlandırılan süreç, diğer terminallerde anında görünür.

2.5. Senkronizasyon ve Veri Tutarlılığı

Shared memory üzerindeki tüm erişimler:

- POSIX semaphore ile korunur,
- Race condition oluşması engellenir,
- Process tablosu tutarlı kalır.

Özellikle:

- Süreç ekleme,
- Süreç silme,
- Sayaç güncellemeleri

  kritik bölge (critical section) olarak ele alınmıştır.

3.6. Program Sonlandırma ve Sinyal Yönetimi

- SIGINT (CTRL+C) sinyali yakalanarak kontrollü kapanma sağlanır.
- Program kapanırken:
  - Yalnızca o instance tarafından başlatılan ATTACHED süreçler sonlandırılır,
  - Thread'ler güvenli şekilde kapatılır,
  - Eğer son ProcX instance'ı ise IPC kaynakları sistemden silinir.

3.KARŞILAŞILAN SORUNLAR VE ÇÖZÜMLER

3.1. Zombie Process Problemi

Sorun: Child process'ler bittiğinde waitpid() çağrılmazsa zombie kalıyordu.

Çözüm: Monitor thread içinde waitpid(WNOHANG) kullanılarak biten süreçler düzenli olarak temizlendi.

3.2. CTRL+C (SIGINT) ile Ani Program Kapanması

Sorun: CTRL+C (SIGINT) ile program ani şekilde kapanıyor, bu durumda ATTACHED süreçler, shared memory kayıtları ve IPC kaynakları temizlenmeden sistemde kalıyordu.

Çözüm: SIGINT sinyali yakalanarak kontrollü kapanma sağlandı. Program kapanırken sadece ilgili ProcX instance'ının başlattığı ATTACHED süreçler sonlandırıldı.

4.SONUÇ

Bu projede, İşletim Sistemleri dersinde öğrendiğimiz süreç yönetimi ve eşzamanlılık konuları uygulamalı olarak çalışılmıştır. Paylaşılan bellek ve semaphore kullanılarak birden fazla sürecin birlikte ve güvenli şekilde çalışması sağlanmıştır. Çalışma sırasında, senkronizasyonun doğru yapılmaması durumunda sistemde nasıl tutarsızlıklar oluşabildiği net bir şekilde görülmüştür. Bu proje, işletim sistemlerinde kaynak yönetiminin ne kadar dikkatli planlanması gerektiğini anlamama yardımcı olmuştur.