MODUL PRAKTIKUM 05 (TUGAS TERSTRUKTUR 2)

TOPIK: SINKRONISASI THREAD POSIX (`pthread`), MUTEX, & `ThreadSanitizer`

Mata Kuliah: Sistem Operasi (INF-201) – Alokasi Waktu: 170 Menit – Bobot Evaluasi: 6%

Program Studi Informatika – Fakultas Teknik – Universitas Bengkulu

---

1. CAPAIAN PEMBELAJARAN MODUL (SUB-CPMK 5)

Setelah menyelesaikan Modul Praktikum 05 ini, mahasiswa diharapkan mampu:

1. Memprogram aplikasi multi-threading menggunakan POSIX Threads library (`pthread_create`, `pthread_join`).
2. Mengidentifikasi dan memecahkan masalah *Race Condition* menggunakan `pthread_mutex_t`.
3. Menguji dan memverifikasi kebebasan *Data Race* menggunakan alat otomatis `ThreadSanitizer`.

2. PRASYARAT TUGAS MEMBACA

Sebelum memulai percobaan laboratorium ini, mahasiswa **wajib** telah membaca bab referensi berikut:

- **[OSTEP]** Chapter 26: *Concurrency: An Introduction* (hal. 291–304).
- **[OSTEP]** Chapter 28: *Locks* (hal. 319–336) & Chapter 31: *Semaphores* (hal. 363–378).
- **[Think OS]** Chapter 7–9: *Threads, Mutex, Semaphores* (hal. 55–82).

3. LANDASAN TEORI RINGKAS

3.1 Race Condition & Critical Section

Race Condition terjadi ketika dua atau lebih thread mengakses dan memodifikasi lokasi memori data bersama (*shared memory*) secara simultan tanpa kontrol sinkronisasi. Bagian kode yang mengkases *shared memory* disebut **Critical Section**.

3.2 Dukungan Hardware & POSIX Mutex

Untuk menjamin *Mutual Exclusion*, hardware menyediakan instruksi atomic seperti `CompareAndSwap`. Di tingkat aplikasi C, pustaka POSIX menyediakan `pthread_mutex_t` untuk memblokir (*sleep/wakeup*) thread lain saat satu thread berada di Critical Section.

4. KODE ACUAN UTUH (EXECUTABLE C LISTING: `race_mutex.c`)

```c
#include <stdio.h>
#include <pthread.h>

#define NUM_THREADS 2
#define N 100000

int counter = 0; // Shared Variable
pthread_mutex_t lock; // Mutex Lock

void* worker(void* arg) {
```

```c
    for (int i = 0; i < N; i++) {
        pthread_mutex_lock(&lock);
        counter++; // Critical Section terproteksi
        pthread_mutex_unlock(&lock);
    }
    return NULL;
}

int main() {
    pthread_t threads[NUM_THREADS];
    pthread_mutex_init(&lock, NULL);

    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_create(&threads[i], NULL, worker, NULL);
    }
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    printf("Hasil Akhir Counter: %d (Ekspektasi: %d)\n", counter,
       NUM_THREADS * N);
    pthread_mutex_destroy(&lock);
    return 0;
}
```

Listing 1: Program Multi-threading Mutex C (`race_mutex.c`)

5. PROSEDUR LANGKAH KERJA PERCOBAAN

1. Kompilasi `race_mutex.c` dengan mengaktifkan **ThreadSanitizer**:

```
$ gcc -fsanitize=thread -g -pthread race_mutex.c -o race_demo
$ ./race_demo
```

2. Hapus sementara baris `pthread_mutex_lock` dan `unlock`, lalu jalankan kembali `./race_demo` dan amati traceback peringatan *Data Race* dari ThreadSanitizer.

6. PANDUAN TROUBLESHOOTING ERROR UMUM

Pesan Error / Kendala	Penyebab Utama	Solusi Solutif
`WARNING: ThreadSanitizer: data race`	Ada variabel global yang diakses bersamaan tanpa lock.	Bungkus operasi *read/write* variabel global dengan Mutex Lock.

7. TUGAS TERSTRUKTUR 2 (BOBOT: 6% NILAI AKHIR)

Implementasikan penyelesaian masalah **Producer-Consumer Bounded Buffer** menggunakan POSIX Threads, Mutex, dan Semaphores (`sem_t`). Pastikan kompilasi dengan `-fsanitize=thread` bersih tanpa peringatan!