Sistem Operasi (INF-201)

Pertemuan 5: Concurrency & Sinkronisasi Proses

Tim Dosen Sistem Operasi

Program Studi Informatika – Universitas Bengkulu

Semester Ganjil 2026/2027

Capaian Pembelajaran (Sub-CPMK 5)

- Mahasiswa mampu memecahkan masalah Race Condition pada sistem multi-threading.
- Mahasiswa memahami instruksi Hardware Atomic (`CompareAndSwap`) & Spinlock vs Mutex (OSTEP Ch. 28).
- Mahasiswa memahami Semaphores (Counting & Binary) pada Producer-Consumer.
- Mahasiswa mampu menguji dan mendeteksi data race menggunakan `ThreadSanitizer`.

Race Condition & Critical Section Problem

Race Condition (OSTEP Ch. 26)

Kondisi di mana beberapa thread mengakses dan memodifikasi data bersama (*shared data*) secara *concurrent*, sehingga hasil akhir acak tergantung urutan eksekusi scheduler.

3 Syarat Solusi Critical Section:

1. **Mutual Exclusion**: Hanya 1 thread yang boleh berada di Critical Section.
2. **Progress**: Pemilihan thread berikutnya tidak boleh tertunda tanpa alasan.
3. **Bounded Waiting**: Batas berapa kali thread lain boleh masuk sebelum thread peminta diizinkan.

Dukungan Hardware: Atomic Primitives & Spinlock

Instruksi CPU Atomic (`CompareAndSwap`)

Instruksi yang dieksekusi oleh hardware CPU secara mutlak (*indivisible/atomic*):

$$\text{CompareAndSwap}(ptr, expected, new\_val)$$

Spinlock (Busy Waiting): Thread berputar dalam *while loop* menunggu lock terbuka. Mengonsumsi 100% CPU time. Baik untuk lock durasi sangat singkat.

Mutex Lock (Sleep / Wakeup): Thread memblokir diri sendiri (*yield/sleep*) jika lock dipegang thread lain, menghemat CPU time. Baik untuk lock durasi panjang.

Sinkronisasi POSIX Threads (pthread) & Mutex

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

int counter = 0;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void* count_up(void* arg) {
    for (int i = 0; i < 100000; i++) {
        pthread_mutex_lock(&lock);
        counter++; // Critical Section terproteksi
        pthread_mutex_unlock(&lock);
    }
    return NULL;
}
```

Tugas Terstruktur 2: POSIX Thread Synchronizer

Praktikum 2 (Tugas Terstruktur - Bobot 6%)

Mengimplementasikan penyelesaian masalah *Producer-Consumer* atau *Dining Philosophers* menggunakan **POSIX Threads (`pthread`)**, **Mutex**, dan **Semaphores**.

- Gunakan flag `-fsanitize=thread` pada `gcc` saat kompilasi untuk memverifikasi bahwa program bebas dari *Data Race*.

Tugas Membaca Mingguan (Reading Assignment)

Persiapan Pertemuan 6: Deadlock & Banker's Algorithm

Sebelum mengikuti kuliah minggu depan, mahasiswa wajib membaca:

- **[OSTEP]** Chapter 32: *Common Concurrency Bugs (Deadlock)* (hal. 379–394)
- **[Think OS]** Chapter 9: *Semaphores in C (Deadlock Conditions)* (hal. 77–82)
- **[xv6 RISC-V]** Chapter 6: *Locking: Deadlock Avoidance* (hal. 66–68)

Pertanyaan Pemicu Diskusi Mandiri

Jelaskan 4 syarat mutlak terjadinya Deadlock! Manakah syarat yang paling mungkin dihindari melalui metode *Lock Ordering*?